

Document Number: MCUXSDKAPIRM  
Rev 2.13.0  
Feb 2023

# MCUXpresso SDK API Reference Manual

NXP Semiconductors



# Contents

## Chapter 1 Introduction

## Chapter 2 Trademarks

## Chapter 3 Architectural Overview

## Chapter 4 Clock Driver

<b>4.1 Overview</b> .....	<b>7</b>
<b>4.2 Data Structure Documentation</b> .....	<b>21</b>
4.2.1 struct firc_trim_config_t .....	21
4.2.2 struct sirc_trim_config_t .....	22
4.2.3 struct vbat_osc_config_t .....	22
4.2.4 struct pll_config_t .....	23
4.2.5 struct pll_setup_t .....	23
<b>4.3 Macro Definition Documentation</b> .....	<b>24</b>
4.3.1 FSL_CLOCK_DRIVER_VERSION .....	24
4.3.2 FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL .....	24
4.3.3 CLOCK_USR_CFG_PLL_CONFIG_CACHE_COUNT .....	24
4.3.4 ROM_CLOCKS .....	24
4.3.5 SRAM_CLOCKS .....	24
4.3.6 FMC_CLOCKS .....	25
4.3.7 INPUTMUX_CLOCKS .....	25
4.3.8 ETH_CLOCKS .....	25
4.3.9 GPIO_CLOCKS .....	25
4.3.10 PINT_CLOCKS .....	26
4.3.11 DMA_CLOCKS .....	26
4.3.12 EDMA_CLOCKS .....	26
4.3.13 CRC_CLOCKS .....	26
4.3.14 WWDT_CLOCKS .....	26
4.3.15 MAILBOX_CLOCKS .....	27
4.3.16 LPADC_CLOCKS .....	27
4.3.17 MRT_CLOCKS .....	27
4.3.18 OSTIMER_CLOCKS .....	27
4.3.19 SCT_CLOCKS .....	27
4.3.20 UTICK_CLOCKS .....	28

Section No.	Title	Page No.
4.3.21	LP_FLEXCOMM_CLOCKS .....	28
4.3.22	LPUART_CLOCKS .....	28
4.3.23	LPI2C_CLOCKS .....	28
4.3.24	LPSPI_CLOCKS .....	29
4.3.25	CTIMER_CLOCKS .....	29
4.3.26	FREQME_CLOCKS .....	29
4.3.27	POWERQUAD_CLOCKS .....	29
4.3.28	PLU_CLOCKS .....	29
4.3.29	PUF_CLOCKS .....	30
4.3.30	VREF_CLOCKS .....	30
4.3.31	LPDAC_CLOCKS .....	30
4.3.32	HPDAC_CLOCKS .....	30
4.3.33	PWM_CLOCKS .....	30
4.3.34	ENC_CLOCKS .....	31
4.3.35	FLEXIO_CLOCKS .....	31
4.3.36	FLEXCAN_CLOCKS .....	31
4.3.37	EMVSIM_CLOCKS .....	31
4.3.38	USDHC_CLOCKS .....	31
4.3.39	SAI_CLOCKS .....	32
4.3.40	RTC_CLOCKS .....	32
4.3.41	PDM_CLOCKS .....	32
4.3.42	ERM_CLOCKS .....	32
4.3.43	EIM_CLOCKS .....	32
4.3.44	OPAMP_CLOCKS .....	33
4.3.45	TSI_CLOCKS .....	33
4.3.46	TRNG_CLOCKS .....	33
4.3.47	LPCMP_CLOCKS .....	33
4.3.48	CLK_GATE_REG_OFFSET_SHIFT .....	33
4.3.49	BUS_CLK .....	33
4.3.50	CLK_ATTACH_ID .....	33
4.3.51	PLL_CONFIGFLAG_FORCE_NOREACT .....	34
<b>4.4</b>	<b>Enumeration Type Documentation .....</b>	<b>34</b>
4.4.1	clock_ip_name_t .....	34
4.4.2	clock_name_t .....	37
4.4.3	clock_attach_id_t .....	38
4.4.4	clock_div_name_t .....	46
4.4.5	osc32k_clk_gate_id_t .....	48
4.4.6	clk16k_clk_gate_id_t .....	48
4.4.7	clock_ctrl_enable_t .....	48
4.4.8	clock_usb_phy_src_t .....	48
4.4.9	_scg_status .....	49
4.4.10	firc_trim_mode_t .....	49
4.4.11	firc_trim_src_t .....	49
4.4.12	sirc_trim_mode_t .....	49

Section No.	Title	Page No.
4.4.13	<code>sirc_trim_src_t</code>	49
4.4.14	<code>scg_sosc_monitor_mode_t</code>	50
4.4.15	<code>scg_rosc_monitor_mode_t</code>	50
4.4.16	<code>scg_upll_monitor_mode_t</code>	50
4.4.17	<code>scg_pll0_monitor_mode_t</code>	50
4.4.18	<code>scg_pll1_monitor_mode_t</code>	50
4.4.19	<code>vbat_osc_xtal_cap_t</code>	51
4.4.20	<code>vbat_osc_extal_cap_t</code>	51
4.4.21	<code>vbat_osc_coarse_adjustment_value_t</code>	52
4.4.22	<code>pll_clk_src_t</code>	52
4.4.23	<code>ss_progmodfm_t</code>	52
4.4.24	<code>ss_progmoddp_t</code>	52
4.4.25	<code>ss_modwvctrl_t</code>	53
4.4.26	<code>pll_error_t</code>	53
<b>4.5</b>	<b>Function Documentation</b>	<b>53</b>
4.5.1	<code>CLOCK_EnableClock</code>	53
4.5.2	<code>CLOCK_DisableClock</code>	53
4.5.3	<code>CLOCK_SetupFROHFClocking</code>	54
4.5.4	<code>CLOCK_SetupExtClocking</code>	54
4.5.5	<code>CLOCK_SetupOsc32KClocking</code>	54
4.5.6	<code>CLOCK_SetupClk16KClocking</code>	54
4.5.7	<code>CLOCK_FROHFTrimConfig</code>	55
4.5.8	<code>CLOCK_FRO12MTrimConfig</code>	55
4.5.9	<code>CLOCK_SetSysOscMonitorMode</code>	55
4.5.10	<code>CLOCK_SetRoscMonitorMode</code>	56
4.5.11	<code>CLOCK_SetUppllMonitorMode</code>	56
4.5.12	<code>CLOCK_SetPll0MonitorMode</code>	56
4.5.13	<code>CLOCK_SetPll1MonitorMode</code>	56
4.5.14	<code>VBAT_SetOscConfig</code>	57
4.5.15	<code>CLOCK_AttachClk</code>	57
4.5.16	<code>CLOCK_GetClockAttachId</code>	57
4.5.17	<code>CLOCK_SetClkDiv</code>	57
4.5.18	<code>CLOCK_GetClkDiv</code>	58
4.5.19	<code>CLOCK_HaltClkDiv</code>	58
4.5.20	<code>CLOCK_SetupClockCtrl</code>	58
4.5.21	<code>CLOCK_GetFreq</code>	59
4.5.22	<code>CLOCK_GetCoreSysClkFreq</code>	59
4.5.23	<code>CLOCK_GetCTimerClkFreq</code>	59
4.5.24	<code>CLOCK_GetAdcClkFreq</code>	59
4.5.25	<code>CLOCK_GetUsb0ClkFreq</code>	59
4.5.26	<code>CLOCK_GetLPFlexCommClkFreq</code>	60
4.5.27	<code>CLOCK_GetSctClkFreq</code>	60
4.5.28	<code>CLOCK_GetTsiClkFreq</code>	60
4.5.29	<code>CLOCK_GetSincFilterClkFreq</code>	60

Section No.	Title	Page No.
4.5.30	CLOCK_GetDacClkFreq .....	60
4.5.31	CLOCK_GetFlexspiClkFreq .....	60
4.5.32	CLOCK_GetPll0OutFreq .....	61
4.5.33	CLOCK_GetPll1OutFreq .....	61
4.5.34	CLOCK_GetPllClkDivFreq .....	61
4.5.35	CLOCK_GetI3cClkFreq .....	61
4.5.36	CLOCK_GetI3cSTCclkFreq .....	61
4.5.37	CLOCK_GetI3cSclkFreq .....	61
4.5.38	CLOCK_GetMicfilClkFreq .....	62
4.5.39	CLOCK_GetUsdhcClkFreq .....	62
4.5.40	CLOCK_GetFlexioClkFreq .....	62
4.5.41	CLOCK_GetFlexcanClkFreq .....	62
4.5.42	CLOCK_GetEnetRmiiClkFreq .....	62
4.5.43	CLOCK_GetEnetPtpRefClkFreq .....	62
4.5.44	CLOCK_SetupEnetTxClk .....	62
4.5.45	CLOCK_GetEnetTxClkFreq .....	63
4.5.46	CLOCK_GetEwm0ClkFreq .....	63
4.5.47	CLOCK_GetWdtClkFreq .....	63
4.5.48	CLOCK_GetOstimerClkFreq .....	63
4.5.49	CLOCK_GetCmpFclkFreq .....	63
4.5.50	CLOCK_GetCmpRrclockFreq .....	64
4.5.51	CLOCK_GetSaiClkFreq .....	64
4.5.52	CLOCK_SetupSaiMclk .....	64
4.5.53	CLOCK_SetupSaiTxBclk .....	64
4.5.54	CLOCK_SetupSaiRxBclk .....	64
4.5.55	CLOCK_GetSaiMclkFreq .....	65
4.5.56	CLOCK_GetSaiTxBclkFreq .....	65
4.5.57	CLOCK_GetSaiRxBclkFreq .....	65
4.5.58	CLOCK_GetEmvsimClkFreq .....	65
4.5.59	CLOCK_GetPLL0InClockRate .....	65
4.5.60	CLOCK_GetPLL1InClockRate .....	66
4.5.61	CLOCK_GetExtUpllFreq .....	66
4.5.62	CLOCK_SetExtUpllFreq .....	66
4.5.63	CLOCK_IsPLL0Locked .....	66
4.5.64	CLOCK_IsPLL1Locked .....	66
4.5.65	CLOCK_GetPLLOutFromSetup .....	66
4.5.66	CLOCK_SetupPLLData .....	67
4.5.67	CLOCK_SetPLL0Freq .....	67
4.5.68	CLOCK_SetPLL1Freq .....	68
4.5.69	CLOCK_EnableOstimer32kClock .....	69
4.5.70	CLOCK_EnableUsbfsClock .....	69
4.5.71	CLOCK_EnableUsbhsPhyPllClock .....	69
4.5.72	CLOCK_DisableUsbhsPhyPllClock .....	69
4.5.73	CLOCK_EnableUsbhsClock .....	69

Section No.	Title	Page No.
<b>Chapter 5 Power Driver</b>		
<b>5.1</b>	<b>Overview</b>	<b>70</b>
<b>5.2</b>	<b>Macro Definition Documentation</b>	<b>70</b>
5.2.1	FSL_POWER_DRIVER_VERSION	70
<b>5.3</b>	<b>Function Documentation</b>	<b>70</b>
5.3.1	POWER_EnterSleep	70
5.3.2	POWER_EnterDeepSleep	71
5.3.3	POWER_EnterPowerDown	71
5.3.4	POWER_EnterDeepPowerDown	72
5.3.5	POWER_SetVoltageForFreq	73
<b>Chapter 6 Reset Driver</b>		
<b>6.1</b>	<b>Overview</b>	<b>74</b>
<b>6.2</b>	<b>Macro Definition Documentation</b>	<b>76</b>
6.2.1	FSL_RESET_DRIVER_VERSION	76
6.2.2	ADC_RSTS	76
<b>6.3</b>	<b>Enumeration Type Documentation</b>	<b>76</b>
6.3.1	SYSCON_RSTn_t	76
<b>6.4</b>	<b>Function Documentation</b>	<b>78</b>
6.4.1	RESET_SetPeripheralReset	78
6.4.2	RESET_ClearPeripheralReset	79
6.4.3	RESET_PeripheralReset	79
<b>Chapter 7 ANACTRL: Analog Control Driver</b>		
<b>7.1</b>	<b>ANACTRL function groups</b>	<b>80</b>
<b>7.2</b>	<b>Overview</b>	<b>80</b>
<b>7.3</b>	<b>Function groups</b>	<b>80</b>
7.3.1	Initialization and deinitialization	80
7.3.2	Set oscillators	80
7.3.3	Measure Frequency	80
7.3.4	Interrupt	80
7.3.5	Status	80
<b>7.4</b>	<b>Data Structure Documentation</b>	<b>83</b>
7.4.1	struct anactrl_fro192M_config_t	83
7.4.2	struct anactrl_xo32M_config_t	83

Section No.	Title	Page No.
<b>7.5</b>	<b>Macro Definition Documentation</b>	<b>83</b>
7.5.1	FSL_ANACTRL_DRIVER_VERSION	83
<b>7.6</b>	<b>Enumeration Type Documentation</b>	<b>84</b>
7.6.1	_anactrl_interrupt_flags	84
7.6.2	_anactrl_interrupt	84
7.6.3	_anactrl_flags	84
7.6.4	_anactrl_osc_flags	84
<b>7.7</b>	<b>Function Documentation</b>	<b>84</b>
7.7.1	ANACTRL_Init	84
7.7.2	ANACTRL_Deinit	85
7.7.3	ANACTRL_SetFro192M	85
7.7.4	ANACTRL_GetDefaultFro192MConfig	85
7.7.5	ANACTRL_SetXo32M	85
7.7.6	ANACTRL_GetDefaultXo32MConfig	86
7.7.7	ANACTRL_MeasureFrequency	86
7.7.8	ANACTRL_EnableInterrupts	87
7.7.9	ANACTRL_DisableInterrupts	87
7.7.10	ANACTRL_ClearInterrupts	87
7.7.11	ANACTRL_GetStatusFlags	87
7.7.12	ANACTRL_GetOscStatusFlags	88
7.7.13	ANACTRL_GetInterruptStatusFlags	88

## Chapter 8 CASPER: The Cryptographic Accelerator and Signal Processing Engine with R- A- M sharing

<b>8.1</b>	<b>Overview</b>	<b>90</b>
<b>8.2</b>	<b>CASPER Driver Initialization and deinitialization</b>	<b>90</b>
<b>8.3</b>	<b>Comments about API usage in RTOS</b>	<b>90</b>
<b>8.4</b>	<b>Comments about API usage in interrupt handler</b>	<b>90</b>
<b>8.5</b>	<b>CASPER Driver Examples</b>	<b>90</b>
8.5.1	Simple examples	90
<b>8.6</b>	<b>casper_driver</b>	<b>92</b>
8.6.1	Overview	92
8.6.2	Macro Definition Documentation	92
8.6.3	Enumeration Type Documentation	94
8.6.4	Function Documentation	94

Section No.	Title	Page No.
<b>8.7 casper_driver_pkha .....</b>		<b>97</b>
8.7.1 Overview .....		97
8.7.2 Function Documentation .....		97

## Chapter 9 CMP: Analog Comparator Driver

<b>9.1 Overview .....</b>		<b>103</b>
<b>9.2 Function groups .....</b>		<b>103</b>
9.2.1 Initialization and deinitialization .....		103
9.2.2 Compare .....		103
9.2.3 Interrupt .....		103
9.2.4 Status .....		103
<b>9.3 Typical use case .....</b>		<b>104</b>
9.3.1 Polling Configuration .....		104
9.3.2 Interrupt Configuration .....		104
<b>9.4 Data Structure Documentation .....</b>		<b>106</b>
9.4.1 struct cmp_config_t .....		106
<b>9.5 Macro Definition Documentation .....</b>		<b>106</b>
9.5.1 FSL_CMP_DRIVER_VERSION .....		106
<b>9.6 Enumeration Type Documentation .....</b>		<b>106</b>
9.6.1 _cmp_input_mux .....		106
9.6.2 _cmp_interrupt_type .....		106
9.6.3 cmp_vref_source_t .....		107
9.6.4 cmp_filtercfg_samplemode_t .....		107
9.6.5 cmp_filtercfg_clkdiv_t .....		107
<b>9.7 Function Documentation .....</b>		<b>107</b>
9.7.1 CMP_Init .....		107
9.7.2 CMP_Deinit .....		108
9.7.3 CMP_GetDefaultConfig .....		108
9.7.4 CMP_SetVREF .....		108
9.7.5 CMP_GetOutput .....		108
9.7.6 CMP_EnableInterrupt .....		108
9.7.7 CMP_EnableFilteredInterruptSource .....		109
9.7.8 CMP_GetPreviousInterruptStatus .....		109
9.7.9 CMP_GetInterruptStatus .....		109
9.7.10 CMP_FilterSampleConfig .....		109

## Chapter 10 Common Driver

<b>10.1 Overview .....</b>		<b>111</b>
----------------------------	--	------------

Section No.	Title	Page No.
<b>10.2 Macro Definition Documentation</b>		<b>113</b>
10.2.1 FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ		113
10.2.2 MAKE_STATUS		113
10.2.3 MAKE_VERSION		114
10.2.4 FSL_COMMON_DRIVER_VERSION		114
10.2.5 DEBUG_CONSOLE_DEVICE_TYPE_NONE		114
10.2.6 DEBUG_CONSOLE_DEVICE_TYPE_UART		114
10.2.7 DEBUG_CONSOLE_DEVICE_TYPE_LPUART		114
10.2.8 DEBUG_CONSOLE_DEVICE_TYPE_LPSCI		114
10.2.9 DEBUG_CONSOLE_DEVICE_TYPE_USBCDC		114
10.2.10 DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM		114
10.2.11 DEBUG_CONSOLE_DEVICE_TYPE_IUART		114
10.2.12 DEBUG_CONSOLE_DEVICE_TYPE_VUSART		114
10.2.13 DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART		114
10.2.14 DEBUG_CONSOLE_DEVICE_TYPE_SWO		114
10.2.15 DEBUG_CONSOLE_DEVICE_TYPE_QSCI		114
10.2.16 ARRAY_SIZE		114
<b>10.3 Typedef Documentation</b>		<b>114</b>
10.3.1 status_t		114
<b>10.4 Enumeration Type Documentation</b>		<b>115</b>
10.4.1 _status_groups		115
10.4.2 anonymous enum		117
<b>10.5 Function Documentation</b>		<b>118</b>
10.5.1 SDK_Malloc		118
10.5.2 SDK_Free		118
10.5.3 SDK_DelayAtLeastUs		118

## Chapter 11 CTIMER: Standard counter/timers

<b>11.1 Overview</b>		<b>119</b>
<b>11.2 Function groups</b>		<b>119</b>
11.2.1 Initialization and deinitialization		119
11.2.2 PWM Operations		119
11.2.3 Match Operation		119
11.2.4 Input capture operations		119
<b>11.3 Typical use case</b>		<b>120</b>
11.3.1 Match example		120
11.3.2 PWM output example		120
<b>11.4 Data Structure Documentation</b>		<b>123</b>
11.4.1 struct ctimer_match_config_t		123

Section No.	Title	Page No.
11.4.2	struct ctimer_config_t .....	124
<b>11.5</b>	<b>Enumeration Type Documentation .....</b>	<b>124</b>
11.5.1	ctimer_capture_channel_t .....	124
11.5.2	ctimer_capture_edge_t .....	124
11.5.3	ctimer_match_t .....	124
11.5.4	ctimer_external_match_t .....	125
11.5.5	ctimer_match_output_control_t .....	125
11.5.6	ctimer_interrupt_enable_t .....	125
11.5.7	ctimer_status_flags_t .....	125
11.5.8	ctimer_callback_type_t .....	126
<b>11.6</b>	<b>Function Documentation .....</b>	<b>126</b>
11.6.1	CTIMER_Init .....	126
11.6.2	CTIMER_Deinit .....	126
11.6.3	CTIMER_GetDefaultConfig .....	126
11.6.4	CTIMER_SetupPwmPeriod .....	127
11.6.5	CTIMER_SetupPwm .....	127
11.6.6	CTIMER_UpdatePwmPulsePeriod .....	128
11.6.7	CTIMER_UpdatePwmDutycycle .....	128
11.6.8	CTIMER_SetupMatch .....	129
11.6.9	CTIMER_GetOutputMatchStatus .....	129
11.6.10	CTIMER_SetupCapture .....	130
11.6.11	CTIMER_GetTimerCountValue .....	131
11.6.12	CTIMER_RegisterCallBack .....	131
11.6.13	CTIMER_EnableInterrupts .....	131
11.6.14	CTIMER_DisableInterrupts .....	132
11.6.15	CTIMER_GetEnabledInterrupts .....	133
11.6.16	CTIMER_GetStatusFlags .....	133
11.6.17	CTIMER_ClearStatusFlags .....	133
11.6.18	CTIMER_StartTimer .....	134
11.6.19	CTIMER_StopTimer .....	134
11.6.20	CTIMER_Reset .....	134
11.6.21	CTIMER_SetPrescale .....	134
11.6.22	CTIMER_GetCaptureValue .....	135
11.6.23	CTIMER_EnableResetMatchChannel .....	135
11.6.24	CTIMER_EnableStopMatchChannel .....	135
11.6.25	CTIMER_EnableRisingEdgeCapture .....	136
11.6.26	CTIMER_EnableFallingEdgeCapture .....	136

## Chapter 12 FLEXCOMM: FLEXCOMM Driver

<b>12.1</b>	<b>Overview .....</b>	<b>137</b>
<b>12.2</b>	<b>FLEXCOMM Driver .....</b>	<b>138</b>

Section No.	Title	Page No.
12.2.1	Overview .....	138
12.2.2	Macro Definition Documentation .....	138
12.2.3	Typedef Documentation .....	139
12.2.4	Enumeration Type Documentation .....	139
12.2.5	Function Documentation .....	139
12.2.6	Variable Documentation .....	139

## Chapter 13 I2C: Inter-Integrated Circuit Driver

<b>13.1</b>	<b>Overview .....</b>	<b>140</b>
<b>13.2</b>	<b>Typical use case .....</b>	<b>140</b>
13.2.1	Master Operation in functional method .....	140
13.2.2	Master Operation in interrupt transactional method .....	141
13.2.3	Master Operation in DMA transactional method .....	142
13.2.4	Slave Operation in functional method .....	142
13.2.5	Slave Operation in interrupt transactional method .....	143
<b>13.3</b>	<b>I2C Driver .....</b>	<b>145</b>
13.3.1	Overview .....	145
13.3.2	Macro Definition Documentation .....	147
13.3.3	Enumeration Type Documentation .....	147
<b>13.4</b>	<b>I2C Master Driver .....</b>	<b>150</b>
13.4.1	Overview .....	150
13.4.2	Data Structure Documentation .....	152
13.4.3	Typedef Documentation .....	154
13.4.4	Enumeration Type Documentation .....	156
13.4.5	Function Documentation .....	156
<b>13.5</b>	<b>I2C Slave Driver .....</b>	<b>168</b>
13.5.1	Overview .....	168
13.5.2	Data Structure Documentation .....	170
13.5.3	Typedef Documentation .....	173
13.5.4	Enumeration Type Documentation .....	173
13.5.5	Function Documentation .....	174
<b>13.6</b>	<b>I2C DMA Driver .....</b>	<b>183</b>
13.6.1	Overview .....	183
13.6.2	Data Structure Documentation .....	184
13.6.3	Macro Definition Documentation .....	184
13.6.4	Typedef Documentation .....	185
13.6.5	Function Documentation .....	185
<b>13.7</b>	<b>I2C FreeRTOS Driver .....</b>	<b>187</b>
13.7.1	Overview .....	187

Section No.	Title	Page No.
13.7.2	Data Structure Documentation .....	187
13.7.3	Macro Definition Documentation .....	187
13.7.4	Function Documentation .....	188

<b>13.8</b>	<b>I2C CMSIS Driver .....</b>	<b>189</b>
13.8.1	I2C CMSIS Driver .....	189

## Chapter 14 I2S: I2S Driver

<b>14.1</b>	<b>Overview .....</b>	<b>191</b>
<b>14.2</b>	<b>I2S Driver Initialization and Configuration .....</b>	<b>191</b>
<b>14.3</b>	<b>I2S Transmit Data .....</b>	<b>191</b>
<b>14.4</b>	<b>I2S Interrupt related functions .....</b>	<b>192</b>
<b>14.5</b>	<b>I2S Other functions .....</b>	<b>192</b>
<b>14.6</b>	<b>I2S Data formats .....</b>	<b>192</b>
14.6.1	DMA mode .....	192
14.6.2	Interrupt mode .....	195
<b>14.7</b>	<b>I2S Driver Examples .....</b>	<b>196</b>
14.7.1	Interrupt mode examples .....	196
14.7.2	DMA mode examples .....	197
<b>14.8</b>	<b>I2S Driver .....</b>	<b>200</b>
14.8.1	Overview .....	200
14.8.2	Data Structure Documentation .....	202
14.8.3	Macro Definition Documentation .....	204
14.8.4	Typedef Documentation .....	204
14.8.5	Enumeration Type Documentation .....	205
14.8.6	Function Documentation .....	206

## Chapter 15 SPI: Serial Peripheral Interface Driver

<b>15.1</b>	<b>Overview .....</b>	<b>215</b>
<b>15.2</b>	<b>Typical use case .....</b>	<b>215</b>
15.2.1	SPI master transfer using an interrupt method .....	215
15.2.2	SPI Send/receive using a DMA method .....	216
<b>15.3</b>	<b>SPI Driver .....</b>	<b>218</b>
15.3.1	Overview .....	218
15.3.2	Data Structure Documentation .....	221

Section No.	Title	Page No.
15.3.3	Macro Definition Documentation .....	224
15.3.4	Typedef Documentation .....	224
15.3.5	Enumeration Type Documentation .....	225
15.3.6	Variable Documentation .....	227
<b>15.4</b>	<b>SPI DMA Driver .....</b>	<b>228</b>
15.4.1	Overview .....	228
15.4.2	Data Structure Documentation .....	229
15.4.3	Macro Definition Documentation .....	229
15.4.4	Typedef Documentation .....	229
15.4.5	Function Documentation .....	230
<b>15.5</b>	<b>SPI FreeRTOS driver .....</b>	<b>235</b>
15.5.1	Overview .....	235
15.5.2	Data Structure Documentation .....	235
15.5.3	Macro Definition Documentation .....	236
15.5.4	Function Documentation .....	236
<b>15.6</b>	<b>SPI CMSIS driver .....</b>	<b>238</b>
15.6.1	Function groups .....	238
15.6.2	Typical use case .....	239

## Chapter 16 USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver

<b>16.1</b>	<b>Overview .....</b>	<b>240</b>
<b>16.2</b>	<b>Typical use case .....</b>	<b>241</b>
16.2.1	USART Send/receive using a polling method .....	241
16.2.2	USART Send/receive using an interrupt method .....	241
16.2.3	USART Receive using the ringbuffer feature .....	242
16.2.4	USART Send/Receive using the DMA method .....	243
<b>16.3</b>	<b>USART Driver .....</b>	<b>245</b>
16.3.1	Overview .....	245
16.3.2	Data Structure Documentation .....	250
16.3.3	Macro Definition Documentation .....	252
16.3.4	Typedef Documentation .....	252
16.3.5	Enumeration Type Documentation .....	253
16.3.6	Function Documentation .....	256
<b>16.4</b>	<b>USART DMA Driver .....</b>	<b>273</b>
16.4.1	Overview .....	273
16.4.2	Data Structure Documentation .....	274
16.4.3	Macro Definition Documentation .....	274
16.4.4	Typedef Documentation .....	275
16.4.5	Function Documentation .....	275

Section No.	Title	Page No.
<b>16.5 USART FreeRTOS Driver</b>		<b>279</b>
16.5.1 Overview		279
16.5.2 Data Structure Documentation		279
16.5.3 Macro Definition Documentation		280
16.5.4 Function Documentation		280
<b>16.6 USART CMSIS Driver</b>		<b>283</b>
16.6.1 USART Send Methods		283

## Chapter 17 GINT: Group GPIO Input Interrupt Driver

<b>17.1 Overview</b>		<b>285</b>
<b>17.2 Group GPIO Input Interrupt Driver operation</b>		<b>285</b>
<b>17.3 Typical use case</b>		<b>285</b>
<b>17.4 Macro Definition Documentation</b>		<b>286</b>
17.4.1 FSL_GINT_DRIVER_VERSION		286
<b>17.5 Typedef Documentation</b>		<b>286</b>
17.5.1 gint_cb_t		286
<b>17.6 Enumeration Type Documentation</b>		<b>286</b>
17.6.1 gint_comb_t		286
17.6.2 gint_trig_t		286
<b>17.7 Function Documentation</b>		<b>287</b>
17.7.1 GINT_Init		287
17.7.2 GINT_SetCtrl		287
17.7.3 GINT_GetCtrl		287
17.7.4 GINT_ConfigPins		288
17.7.5 GINT_GetConfigPins		288
17.7.6 GINT_EnableCallback		289
17.7.7 GINT_DisableCallback		289
17.7.8 GINT_ClrStatus		289
17.7.9 GINT_GetStatus		290
17.7.10 GINT_Deinit		290

## Chapter 18 Hashcrypt: The Cryptographic Accelerator

<b>18.1 Overview</b>		<b>291</b>
<b>18.2 Hashcrypt Driver Initialization and deinitialization</b>		<b>291</b>
<b>18.3 Comments about API usage in RTOS</b>		<b>291</b>

Section No.	Title	Page No.
<b>18.4</b>	<b>Comments about API usage in interrupt handler</b>	<b>291</b>
<b>18.5</b>	<b>Hashcrypt Driver Examples</b>	<b>291</b>
18.5.1	Simple examples	291
<b>18.6</b>	<b>Hashcrypt AES</b>	<b>293</b>
18.6.1	Overview	293
18.6.2	Data Structure Documentation	294
18.6.3	Enumeration Type Documentation	294
18.6.4	Function Documentation	295
<b>18.7</b>	<b>Hashcrypt HASH</b>	<b>301</b>
18.7.1	Overview	301
18.7.2	Data Structure Documentation	301
18.7.3	Macro Definition Documentation	302
18.7.4	Typedef Documentation	302
18.7.5	Function Documentation	302
<b>18.8</b>	<b>Hashcrypt Background HASH</b>	<b>305</b>
18.8.1	Overview	305
18.8.2	Function Documentation	305
<b>Chapter 19 IAP: In Application Programming Driver</b>		
<b>19.1</b>	<b>Overview</b>	<b>307</b>
<b>19.2</b>	<b>In Application Programming operation</b>	<b>307</b>
<b>19.3</b>	<b>Typical use case</b>	<b>308</b>
19.3.1	IAP Basic Operations	308
<b>19.4</b>	<b>Data Structure Documentation</b>	<b>311</b>
19.4.1	struct flash_ecc_log_t	311
19.4.2	struct flash_mode_config_t	311
19.4.3	struct flash_ffr_config_t	311
19.4.4	struct flash_config_t	311
<b>19.5</b>	<b>Macro Definition Documentation</b>	<b>312</b>
19.5.1	MAKE_VERSION	312
19.5.2	FSL_FLASH_DRIVER_VERSION	312
19.5.3	FSL_FEATURE_FLASH_IP_IS_C040HD_ATFC	312
19.5.4	kStatusGroupGeneric	312
19.5.5	MAKE_STATUS	312
19.5.6	FOUR_CHAR_CODE	312
<b>19.6</b>	<b>Enumeration Type Documentation</b>	<b>312</b>

Section No.	Title	Page No.
19.6.1	<code>_flash_driver_version_constants</code>	313
19.6.2	<code>_flash_status</code>	313
19.6.3	<code>_flash_driver_api_keys</code>	314
19.6.4	<code>flash_property_tag_t</code>	314
19.6.5	<code>_flash_max_erase_page_value</code>	314
19.6.6	<code>_flash_alignment_property</code>	315
19.6.7	<code>_flash_read_ecc_option</code>	315
19.6.8	<code>_flash_read_margin_option</code>	315
19.6.9	<code>_flash_read_dmacc_option</code>	315
19.6.10	<code>_flash_ramp_control_option</code>	315
<b>19.7</b>	<b>Function Documentation</b>	<b>316</b>
19.7.1	<code>FLASH_Init</code>	316
19.7.2	<code>FLASH_Erase</code>	317
19.7.3	<code>FLASH_Program</code>	318
19.7.4	<code>FLASH_Read</code>	319
19.7.5	<code>FLASH_VerifyErase</code>	320
19.7.6	<code>FLASH_VerifyProgram</code>	321
19.7.7	<code>FLASH_GetProperty</code>	322
<b>19.8</b>	<b>IAP_FFR Driver</b>	<b>324</b>
19.8.1	<code>Overview</code>	324
19.8.2	<code>Macro Definition Documentation</code>	325
19.8.3	<code>Enumeration Type Documentation</code>	325
19.8.4	<code>Function Documentation</code>	326
<b>19.9</b>	<b>IAP_KBP Driver</b>	<b>331</b>
19.9.1	<code>Overview</code>	331
19.9.2	<code>Data Structure Documentation</code>	332
19.9.3	<code>Enumeration Type Documentation</code>	333
19.9.4	<code>Function Documentation</code>	333

## Chapter 20 INPUTMUX: Input Multiplexing Driver

<b>20.1</b>	<b>Overview</b>	<b>336</b>
<b>20.2</b>	<b>Input Multiplexing Driver operation</b>	<b>336</b>
<b>20.3</b>	<b>Typical use case</b>	<b>336</b>
<b>20.4</b>	<b>Enumeration Type Documentation</b>	<b>342</b>
20.4.1	<code>inputmux_connection_t</code>	342
20.4.2	<code>inputmux_signal_t</code>	344
<b>20.5</b>	<b>Function Documentation</b>	<b>344</b>
20.5.1	<code>INPUTMUX_Init</code>	344

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
20.5.2	INPUTMUX_AttachSignal .....	345
20.5.3	INPUTMUX_Deinit .....	345
<b>Chapter 21 LPADC: 12-bit SAR Analog-to-Digital Converter Driver</b>		
<b>21.1</b>	<b>Overview .....</b>	<b>346</b>
<b>21.2</b>	<b>Typical use case .....</b>	<b>346</b>
21.2.1	Polling Configuration .....	346
21.2.2	Interrupt Configuration .....	346
<b>21.3</b>	<b>Data Structure Documentation .....</b>	<b>351</b>
21.3.1	struct lpadc_config_t .....	351
21.3.2	struct lpadc_conv_command_config_t .....	353
21.3.3	struct lpadc_conv_trigger_config_t .....	354
21.3.4	struct lpadc_conv_result_t .....	355
<b>21.4</b>	<b>Macro Definition Documentation .....</b>	<b>356</b>
21.4.1	FSL_LPADC_DRIVER_VERSION .....	356
21.4.2	LPADC_GET_ACTIVE_COMMAND_STATUS .....	356
21.4.3	LPADC_GET_ACTIVE_TRIGGER_STATUE .....	356
<b>21.5</b>	<b>Enumeration Type Documentation .....</b>	<b>356</b>
21.5.1	_lpadc_status_flags .....	356
21.5.2	_lpadc_interrupt_enable .....	356
21.5.3	_lpadc_trigger_status_flags .....	357
21.5.4	lpadc_sample_scale_mode_t .....	359
21.5.5	lpadc_sample_channel_mode_t .....	359
21.5.6	lpadc.hardware_average_mode_t .....	359
21.5.7	lpadc.sample_time_mode_t .....	360
21.5.8	lpadc.hardware_compare_mode_t .....	360
21.5.9	lpadc.conversion_resolution_mode_t .....	360
21.5.10	lpadc.conversion_average_mode_t .....	361
21.5.11	lpadc.reference_voltage_source_t .....	361
21.5.12	lpadc.power_level_mode_t .....	361
21.5.13	lpadc.trigger_priority_policy_t .....	362
<b>21.6</b>	<b>Function Documentation .....</b>	<b>362</b>
21.6.1	LPADC_Init .....	362
21.6.2	LPADC_GetDefaultConfig .....	362
21.6.3	LPADC_Deinit .....	363
21.6.4	LPADC_Enable .....	363
21.6.5	LPADC_DoResetFIFO0 .....	363
21.6.6	LPADC_DoResetFIFO1 .....	363
21.6.7	LPADC_DoResetConfig .....	364

Section No.	Title	Page No.
21.6.8	LPADC_GetStatusFlags .....	365
21.6.9	LPADC_ClearStatusFlags .....	365
21.6.10	LPADC_GetTriggerStatusFlags .....	365
21.6.11	LPADC_ClearTriggerStatusFlags .....	365
21.6.12	LPADC_EnableInterrupts .....	366
21.6.13	LPADC_DisableInterrupts .....	366
21.6.14	LPADC_EnableFIFO0WatermarkDMA .....	366
21.6.15	LPADC_EnableFIFO1WatermarkDMA .....	366
21.6.16	LPADC_GetConvResultCount .....	367
21.6.17	LPADC_GetConvResult .....	367
21.6.18	LPADC_SetConvTriggerConfig .....	367
21.6.19	LPADC_GetDefaultConvTriggerConfig .....	368
21.6.20	LPADC_DoSoftwareTrigger .....	368
21.6.21	LPADC_SetConvCommandConfig .....	368
21.6.22	LPADC_GetDefaultConvCommandConfig .....	368
21.6.23	LPADC_SetOffsetValue .....	369
21.6.24	LPADC_EnableOffsetCalibration .....	369
21.6.25	LPADC_DoOffsetCalibration .....	370
21.6.26	LPADC_DoAutoCalibration .....	370

## Chapter 22 CRC: Cyclic Redundancy Check Driver

<b>22.1</b>	<b>Overview .....</b>	<b>371</b>
<b>22.2</b>	<b>CRC Driver Initialization and Configuration .....</b>	<b>371</b>
<b>22.3</b>	<b>CRC Write Data .....</b>	<b>371</b>
<b>22.4</b>	<b>CRC Get Checksum .....</b>	<b>371</b>
<b>22.5</b>	<b>Comments about API usage in RTOS .....</b>	<b>372</b>
<b>22.6</b>	<b>Data Structure Documentation .....</b>	<b>373</b>
22.6.1	struct crc_config_t .....	373
<b>22.7</b>	<b>Macro Definition Documentation .....</b>	<b>374</b>
22.7.1	FSL_CRC_DRIVER_VERSION .....	374
22.7.2	CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT .....	374
<b>22.8</b>	<b>Enumeration Type Documentation .....</b>	<b>374</b>
22.8.1	crc_polynomial_t .....	374
<b>22.9</b>	<b>Function Documentation .....</b>	<b>374</b>
22.9.1	CRC_Init .....	375
22.9.2	CRC_Deinit .....	376
22.9.3	CRC_Reset .....	376

Section No.	Title	Page No.
22.9.4	CRC_WriteSeed .....	376
22.9.5	CRC_GetDefaultConfig .....	376
22.9.6	CRC_GetConfig .....	377
22.9.7	CRC_WriteData .....	377
22.9.8	CRC_Get32bitResult .....	377
22.9.9	CRC_Get16bitResult .....	378

## Chapter 23 DMA: Direct Memory Access Controller Driver

<b>23.1</b>	<b>Overview .....</b>	<b>380</b>
<b>23.2</b>	<b>Typical use case .....</b>	<b>380</b>
23.2.1	DMA Operation .....	380
<b>23.3</b>	<b>Data Structure Documentation .....</b>	<b>385</b>
23.3.1	struct dma_descriptor_t .....	385
23.3.2	struct dma_xfercfg_t .....	385
23.3.3	struct dma_channel_trigger_t .....	386
23.3.4	struct dma_channel_config_t .....	386
23.3.5	struct dma_transfer_config_t .....	387
23.3.6	struct dma_handle_t .....	387
<b>23.4</b>	<b>Macro Definition Documentation .....</b>	<b>387</b>
23.4.1	FSL_DMA_DRIVER_VERSION .....	388
23.4.2	FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE .....	388
23.4.3	DMA_ALLOCATE_HEAD_DESCRIPTOR .....	388
23.4.4	DMA_ALLOCATE_HEAD_DESCRIPTOR_AT_NONCACHEABLE .....	388
23.4.5	DMA_ALLOCATE_LINK_DESCRIPTOR .....	388
23.4.6	DMA_ALLOCATE_LINK_DESCRIPTOR_AT_NONCACHEABLE .....	389
23.4.7	DMA_DESCRIPTOR_END_ADDRESS .....	389
<b>23.5</b>	<b>Typedef Documentation .....</b>	<b>389</b>
23.5.1	dma_callback .....	389
<b>23.6</b>	<b>Enumeration Type Documentation .....</b>	<b>389</b>
23.6.1	anonymous enum .....	389
23.6.2	anonymous enum .....	390
23.6.3	anonymous enum .....	390
23.6.4	dma_priority_t .....	390
23.6.5	dma_irq_t .....	390
23.6.6	dma_trigger_type_t .....	390
23.6.7	anonymous enum .....	391
23.6.8	dma_trigger_burst_t .....	391
23.6.9	dma_burst_wrap_t .....	392
23.6.10	dma_transfer_type_t .....	392

Section No.	Title	Page No.
<b>23.7 Function Documentation</b>		<b>392</b>
23.7.1 DMA_Init		392
23.7.2 DMA_Deinit		392
23.7.3 DMA_InstallDescriptorMemory		392
23.7.4 DMA_ChannelIsActive		393
23.7.5 DMA_ChannelIsBusy		393
23.7.6 DMA_EnableChannelInterrupts		393
23.7.7 DMA_DisableChannelInterrupts		394
23.7.8 DMA_EnableChannel		394
23.7.9 DMA_DisableChannel		394
23.7.10 DMA_EnableChannelPeriphRq		394
23.7.11 DMA_DisableChannelPeriphRq		395
23.7.12 DMA_ConfigureChannelTrigger		395
23.7.13 DMA_SetChannelConfig		395
23.7.14 DMA_SetChannelXferConfig		396
23.7.15 DMA_GetRemainingBytes		396
23.7.16 DMA_SetChannelPriority		397
23.7.17 DMA_GetChannelPriority		397
23.7.18 DMA_SetChannelConfigValid		397
23.7.19 DMA_DoChannelSoftwareTrigger		398
23.7.20 DMA_LoadChannelTransferConfig		398
23.7.21 DMA_CreateDescriptor		398
23.7.22 DMA_SetupDescriptor		399
23.7.23 DMA_SetupChannelDescriptor		399
23.7.24 DMA_LoadChannelDescriptor		400
23.7.25 DMA_AbortTransfer		400
23.7.26 DMA_CreateHandle		400
23.7.27 DMA_SetCallback		401
23.7.28 DMA_PrepTransfer		402
23.7.29 DMA_PrepChannelTransfer		402
23.7.30 DMA_SubmitTransfer		403
23.7.31 DMA_SubmitChannelTransferParameter		403
23.7.32 DMA_SubmitChannelDescriptor		404
23.7.33 DMA_SubmitChannelTransfer		405
23.7.34 DMA_StartTransfer		406
23.7.35 DMA_IRQHandle		406

## Chapter 24 GPIO: General Purpose I/O

<b>24.1 Overview</b>	<b>408</b>
<b>24.2 Function groups</b>	<b>408</b>
24.2.1 Initialization and deinitialization	408
24.2.2 Pin manipulation	408
24.2.3 Port manipulation	408

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
24.2.4	Port masking .....	408
<b>24.3</b>	<b>Typical use case .....</b>	<b>408</b>
<b>24.4</b>	<b>Data Structure Documentation .....</b>	<b>410</b>
24.4.1	struct gpio_pin_config_t .....	410
<b>24.5</b>	<b>Macro Definition Documentation .....</b>	<b>410</b>
24.5.1	FSL_GPIO_DRIVER_VERSION .....	410
<b>24.6</b>	<b>Enumeration Type Documentation .....</b>	<b>410</b>
24.6.1	gpio_pin_direction_t .....	410
<b>24.7</b>	<b>Function Documentation .....</b>	<b>410</b>
24.7.1	GPIO_PortInit .....	410
24.7.2	GPIO_PinInit .....	410
24.7.3	GPIO_PinWrite .....	411
24.7.4	GPIO_PinRead .....	411
24.7.5	GPIO_PortSet .....	412
24.7.6	GPIO_PortClear .....	412
24.7.7	GPIO_PortToggle .....	412

## Chapter 25 IOCON: I/O pin configuration

<b>25.1</b>	<b>Overview .....</b>	<b>414</b>
<b>25.2</b>	<b>Function groups .....</b>	<b>414</b>
25.2.1	Pin mux set .....	414
25.2.2	Pin mux set .....	414
<b>25.3</b>	<b>Typical use case .....</b>	<b>414</b>
<b>25.4</b>	<b>Data Structure Documentation .....</b>	<b>415</b>
25.4.1	struct iocon_group_t .....	415
<b>25.5</b>	<b>Macro Definition Documentation .....</b>	<b>415</b>
25.5.1	FSL_IOCON_DRIVER_VERSION .....	415
25.5.2	IOCON_FUNC0 .....	415
<b>25.6</b>	<b>Function Documentation .....</b>	<b>415</b>
25.6.1	IOCON_PinMuxSet .....	415
25.6.2	IOCON_SetPinMuxing .....	416

## Chapter 26 RTC: Real Time Clock

<b>26.1</b>	<b>Overview .....</b>	<b>417</b>
-------------	-----------------------	------------

Section No.	Title	Page No.
<b>26.2 Function groups</b>		<b>417</b>
26.2.1 Initialization and deinitialization		417
26.2.2 Set & Get Datetime		417
26.2.3 Set & Get Alarm		417
26.2.4 Start & Stop timer		417
26.2.5 Status		418
26.2.6 Interrupt		418
26.2.7 High resolution timer		418
<b>26.3 Typical use case</b>		<b>418</b>
26.3.1 RTC tick example		418
<b>26.4 Data Structure Documentation</b>		<b>420</b>
26.4.1 struct rtc_datetime_t		420
<b>26.5 Enumeration Type Documentation</b>		<b>421</b>
26.5.1 rtc_interrupt_enable_t		421
26.5.2 rtc_status_flags_t		421
<b>26.6 Function Documentation</b>		<b>421</b>
26.6.1 RTC_Init		421
26.6.2 RTC_Deinit		421
26.6.3 RTC_SetDatetime		421
26.6.4 RTC_GetDatetime		422
26.6.5 RTC_SetAlarm		422
26.6.6 RTC_GetAlarm		422
26.6.7 RTC_EnableWakeupTimer		423
26.6.8 RTC_GetEnabledWakeupTimer		423
26.6.9 RTC_SetSecondsTimerMatch		423
26.6.10 RTC_GetSecondsTimerMatch		424
26.6.11 RTC_SetSecondsTimerCount		424
26.6.12 RTC_GetSecondsTimerCount		424
26.6.13 RTC_SetWakeupCount		424
26.6.14 RTC_GetWakeupCount		425
26.6.15 RTC_EnableWakeUpTimerInterruptFromDPD		425
26.6.16 RTC_EnableAlarmTimerInterruptFromDPD		425
26.6.17 RTC_EnableInterrupts		426
26.6.18 RTC_DisableInterrupts		426
26.6.19 RTC_GetEnabledInterrupts		426
26.6.20 RTC_GetStatusFlags		427
26.6.21 RTC_ClearStatusFlags		427
26.6.22 RTC_EnableTimer		427
26.6.23 RTC_StartTimer		428
26.6.24 RTC_StopTimer		428
26.6.25 RTC_Reset		428

Section No.	Title	Page No.
<b>Chapter 27 Mailbox</b>		
<b>27.1</b>	<b>Overview</b>	<b>430</b>
<b>27.2</b>	<b>Typical use case</b>	<b>430</b>
<b>27.3</b>	<b>Macro Definition Documentation</b>	<b>431</b>
27.3.1	FSL_MAILBOX_DRIVER_VERSION	431
<b>27.4</b>	<b>Function Documentation</b>	<b>431</b>
27.4.1	MAILBOX_Init	431
27.4.2	MAILBOX_Deinit	431
27.4.3	MAILBOX_GetMutex	431
27.4.4	MAILBOX_SetMutex	432
<b>Chapter 28 MRT: Multi-Rate Timer</b>		
<b>28.1</b>	<b>Overview</b>	<b>433</b>
<b>28.2</b>	<b>Function groups</b>	<b>433</b>
28.2.1	Initialization and deinitialization	433
28.2.2	Timer period Operations	433
28.2.3	Start and Stop timer operations	433
28.2.4	Get and release channel	434
28.2.5	Status	434
28.2.6	Interrupt	434
<b>28.3</b>	<b>Typical use case</b>	<b>434</b>
28.3.1	MRT tick example	434
<b>28.4</b>	<b>Data Structure Documentation</b>	<b>436</b>
28.4.1	struct mrt_config_t	436
<b>28.5</b>	<b>Enumeration Type Documentation</b>	<b>436</b>
28.5.1	mrt_chnl_t	436
28.5.2	mrt_timer_mode_t	436
28.5.3	mrt_interrupt_enable_t	437
28.5.4	mrt_status_flags_t	437
<b>28.6</b>	<b>Function Documentation</b>	<b>437</b>
28.6.1	MRT_Init	437
28.6.2	MRT_Deinit	437
28.6.3	MRT_GetDefaultConfig	437
28.6.4	MRT_SetupChannelMode	438
28.6.5	MRT_EnableInterrupts	438
28.6.6	MRT_DisableInterrupts	438

Section No.	Title	Page No.
28.6.7	MRT_GetEnabledInterrupts .....	438
28.6.8	MRT_GetStatusFlags .....	439
28.6.9	MRT_ClearStatusFlags .....	439
28.6.10	MRT_UpdateTimerPeriod .....	439
28.6.11	MRT_GetCurrentTimerCount .....	440
28.6.12	MRT_StartTimer .....	440
28.6.13	MRT_StopTimer .....	441
28.6.14	MRT_GetIdleChannel .....	441
28.6.15	MRT_ReleaseChannel .....	441

## Chapter 29 OSTIMER: OS Event Timer Driver

<b>29.1</b>	<b>Overview .....</b>	<b>442</b>
<b>29.2</b>	<b>Function groups .....</b>	<b>442</b>
29.2.1	Initialization and deinitialization .....	442
29.2.2	OSTIMER status .....	442
29.2.3	OSTIMER set match value .....	442
29.2.4	OSTIMER get timer count .....	442
<b>29.3</b>	<b>Typical use case .....</b>	<b>443</b>
<b>29.4</b>	<b>Macro Definition Documentation .....</b>	<b>444</b>
29.4.1	FSL_OSTIMER_DRIVER_VERSION .....	444
<b>29.5</b>	<b>Typedef Documentation .....</b>	<b>444</b>
29.5.1	ostimer_callback_t .....	444
<b>29.6</b>	<b>Enumeration Type Documentation .....</b>	<b>444</b>
29.6.1	_ostimer_flags .....	444
<b>29.7</b>	<b>Function Documentation .....</b>	<b>444</b>
29.7.1	OSTIMER_Init .....	444
29.7.2	OSTIMER_Deinit .....	444
29.7.3	OSTIMER_GrayToDecimal .....	444
29.7.4	OSTIMER_DecimalToGray .....	445
29.7.5	OSTIMER_GetStatusFlags .....	445
29.7.6	OSTIMER_ClearStatusFlags .....	445
29.7.7	OSTIMER_SetMatchRawValue .....	446
29.7.8	OSTIMER_SetMatchValue .....	446
29.7.9	OSTIMER_SetMatchRegister .....	447
29.7.10	OSTIMER_EnableMatchInterrupt .....	447
29.7.11	OSTIMER_DisableMatchInterrupt .....	447
29.7.12	OSTIMER_GetCurrentTimerRawValue .....	448
29.7.13	OSTIMER_GetCurrentTimerValue .....	448
29.7.14	OSTIMER_GetCaptureRawValue .....	448

Section No.	Title	Page No.
29.7.15	OSTIMER_GetCaptureValue .....	449
29.7.16	OSTIMER_HandleIRQ .....	449

## Chapter 30 PINT: Pin Interrupt and Pattern Match Driver

<b>30.1</b>	<b>Overview .....</b>	<b>450</b>
<b>30.2</b>	<b>Pin Interrupt and Pattern match Driver operation .....</b>	<b>450</b>
30.2.1	Pin Interrupt use case .....	450
30.2.2	Pattern match use case .....	450
<b>30.3</b>	<b>Typedef Documentation .....</b>	<b>453</b>
30.3.1	pint_cb_t .....	453
<b>30.4</b>	<b>Enumeration Type Documentation .....</b>	<b>453</b>
30.4.1	pint_pin_enable_t .....	453
30.4.2	pint_pin_int_t .....	453
30.4.3	pint_pmatch_input_src_t .....	454
30.4.4	pint_pmatch_bslice_t .....	454
30.4.5	pint_pmatch_bslice_cfg_t .....	454
<b>30.5</b>	<b>Function Documentation .....</b>	<b>454</b>
30.5.1	PINT_Init .....	455
30.5.2	PINT_PinInterruptConfig .....	456
30.5.3	PINT_PinInterruptGetConfig .....	456
30.5.4	PINT_PinInterruptClrStatus .....	457
30.5.5	PINT_PinInterruptGetStatus .....	457
30.5.6	PINT_PinInterruptClrStatusAll .....	457
30.5.7	PINT_PinInterruptGetStatusAll .....	458
30.5.8	PINT_PinInterruptClrFallFlag .....	458
30.5.9	PINT_PinInterruptGetFallFlag .....	458
30.5.10	PINT_PinInterruptClrFallFlagAll .....	459
30.5.11	PINT_PinInterruptGetFallFlagAll .....	459
30.5.12	PINT_PinInterruptClrRiseFlag .....	460
30.5.13	PINT_PinInterruptGetRiseFlag .....	461
30.5.14	PINT_PinInterruptClrRiseFlagAll .....	461
30.5.15	PINT_PinInterruptGetRiseFlagAll .....	461
30.5.16	PINT_PatternMatchConfig .....	462
30.5.17	PINT_PatternMatchGetConfig .....	462
30.5.18	PINT_PatternMatchGetStatus .....	463
30.5.19	PINT_PatternMatchGetStatusAll .....	463
30.5.20	PINT_PatternMatchResetDetectLogic .....	463
30.5.21	PINT_PatternMatchEnable .....	464
30.5.22	PINT_PatternMatchDisable .....	464
30.5.23	PINT_PatternMatchEnableRXEV .....	464

Section No.	Title	Page No.
30.5.24	PINT_PatternMatchDisableRXEV .....	465
30.5.25	PINT_EnableCallback .....	465
30.5.26	PINT_DisableCallback .....	465
30.5.27	PINT_Deinit .....	466
30.5.28	PINT_EnableCallbackByIndex .....	466
30.5.29	PINT_DisableCallbackByIndex .....	466

## Chapter 31 PLU: Programmable Logic Unit

<b>31.1</b>	<b>Overview .....</b>	<b>468</b>
<b>31.2</b>	<b>Function groups .....</b>	<b>468</b>
31.2.1	Initialization and de-initialization .....	468
31.2.2	Set input/output source and Truth Table .....	468
31.2.3	Read current Output State .....	468
31.2.4	Wake-up/Interrupt Control .....	468
<b>31.3</b>	<b>Typical use case .....</b>	<b>469</b>
31.3.1	PLU combination example .....	469
<b>31.4</b>	<b>Data Structure Documentation .....</b>	<b>473</b>
31.4.1	struct plu_wakeint_config_t .....	473
<b>31.5</b>	<b>Enumeration Type Documentation .....</b>	<b>473</b>
31.5.1	plu_lut_index_t .....	473
31.5.2	plu_lut_in_index_t .....	474
31.5.3	plu_lut_input_source_t .....	474
31.5.4	plu_output_index_t .....	475
31.5.5	plu_output_source_t .....	475
31.5.6	_plu_interrupt_mask .....	476
31.5.7	plu_wakeint_filter_mode_t .....	477
31.5.8	plu_wakeint_filter_clock_source_t .....	477
<b>31.6</b>	<b>Function Documentation .....</b>	<b>477</b>
31.6.1	PLU_Init .....	477
31.6.2	PLU_Deinit .....	477
31.6.3	PLU_SetLutInputSource .....	478
31.6.4	PLU_SetOutputSource .....	478
31.6.5	PLU_SetLutTruthTable .....	478
31.6.6	PLU_ReadOutputState .....	479
31.6.7	PLU_GetDefaultWakeIntConfig .....	479
31.6.8	PLU_EnableWakeIntRequest .....	479
31.6.9	PLU_LatchInterrupt .....	481
31.6.10	PLU_ClearLatchedInterrupt .....	481

Section No.	Title	Page No.
<b>Chapter 32 POWERQUAD: PowerQuad hardware accelerator</b>		
<b>32.1</b>	<b>Overview</b>	<b>482</b>
<b>32.2</b>	<b>Function groups</b>	<b>483</b>
32.2.1	POWERQUAD functional Operation	483
<b>32.3</b>	<b>Data Structure Documentation</b>	<b>489</b>
32.3.1	struct pq_prescale_t	489
32.3.2	struct pq_config_t	490
32.3.3	struct pq_biquad_param_t	491
32.3.4	struct pq_biquad_state_t	491
32.3.5	struct pq_biquad_cascade_df2_instance	492
32.3.6	union pq_float_t	492
<b>32.4</b>	<b>Macro Definition Documentation</b>	<b>492</b>
32.4.1	FSL_POWERQUAD_DRIVER_VERSION	492
32.4.2	PQ_Initiate_Vector_Func	492
32.4.3	PQ_End_Vector_Func	493
32.4.4	PQ_StartVector	493
32.4.5	PQ_StartVectorFixed16	493
32.4.6	PQ_StartVectorQ15	495
32.4.7	PQ_EndVector	495
32.4.8	PQ_Vector8F32	495
32.4.9	PQ_Vector8Fixed32	496
32.4.10	PQ_Vector8Fixed16	496
32.4.11	PQ_Vector8Q15	496
32.4.12	PQ_DF2_Vector8_FP	497
32.4.13	PQ_DF2_Vector8_FX	497
32.4.14	PQ_Vector8BiquadDf2F32	498
32.4.15	PQ_Vector8BiquadDf2Fixed32	498
32.4.16	PQ_Vector8BiquadDf2Fixed16	499
32.4.17	PQ_DF2_Cascade_Vector8_FP	499
32.4.18	PQ_DF2_Cascade_Vector8_FX	500
32.4.19	PQ_Vector8BiquadDf2CascadeF32	501
32.4.20	PQ_Vector8BiquadDf2CascadeFixed32	501
32.4.21	PQ_Vector8BiquadDf2CascadeFixed16	502
32.4.22	POWERQUAD_MAKE_MATRIX_LEN	503
32.4.23	PQ_Q31_2_FLOAT	503
32.4.24	PQ_Q15_2_FLOAT	503
<b>32.5</b>	<b>Enumeration Type Documentation</b>	<b>503</b>
32.5.1	pq_computationengine_t	503
32.5.2	pq_format_t	503
32.5.3	pq_cordic_iter_t	503

Section No.	Title	Page No.
<b>32.6 Function Documentation</b>		<b>503</b>
32.6.1 <a href="#">PQ_GetDefaultConfig</a>		503
32.6.2 <a href="#">PQ_SetConfig</a>		504
32.6.3 <a href="#">PQ_SetCoprocessorScaler</a>		504
32.6.4 <a href="#">PQ_Init</a>		504
32.6.5 <a href="#">PQ_Deinit</a>		505
32.6.6 <a href="#">PQ_SetFormat</a>		505
32.6.7 <a href="#">PQ_WaitDone</a>		505
32.6.8 <a href="#">PQ_LnF32</a>		505
32.6.9 <a href="#">PQ_InvF32</a>		505
32.6.10 <a href="#">PQ_SqrtF32</a>		506
32.6.11 <a href="#">PQ_InvSqrtF32</a>		506
32.6.12 <a href="#">PQ_EtoxF32</a>		506
32.6.13 <a href="#">PQ_EtonxF32</a>		506
32.6.14 <a href="#">PQ_SinF32</a>		507
32.6.15 <a href="#">PQ_CosF32</a>		507
32.6.16 <a href="#">PQ_BiquadF32</a>		507
32.6.17 <a href="#">PQ_DivF32</a>		507
32.6.18 <a href="#">PQ_Biquad1F32</a>		508
32.6.19 <a href="#">PQ_LnFixed</a>		508
32.6.20 <a href="#">PQ_InvFixed</a>		508
32.6.21 <a href="#">PQ_SqrtFixed</a>		508
32.6.22 <a href="#">PQ_InvSqrtFixed</a>		509
32.6.23 <a href="#">PQ_EtoxFixed</a>		509
32.6.24 <a href="#">PQ_EtonxFixed</a>		509
32.6.25 <a href="#">PQ_SinQ31</a>		509
32.6.26 <a href="#">PQ_SinQ15</a>		510
32.6.27 <a href="#">PQ_CosQ31</a>		510
32.6.28 <a href="#">PQ_CosQ15</a>		510
32.6.29 <a href="#">PQ_BiquadFixed</a>		510
32.6.30 <a href="#">PQ_VectorLnF32</a>		511
32.6.31 <a href="#">PQ_VectorInvF32</a>		511
32.6.32 <a href="#">PQ_VectorSqrtF32</a>		511
32.6.33 <a href="#">PQ_VectorInvSqrtF32</a>		511
32.6.34 <a href="#">PQ_VectorEtoxF32</a>		512
32.6.35 <a href="#">PQ_VectorEtonxF32</a>		512
32.6.36 <a href="#">PQ_VectorSinF32</a>		512
32.6.37 <a href="#">PQ_VectorCosF32</a>		512
32.6.38 <a href="#">PQ_VectorLnFixed32</a>		513
32.6.39 <a href="#">PQ_VectorInvFixed32</a>		513
32.6.40 <a href="#">PQ_VectorSqrtFixed32</a>		513
32.6.41 <a href="#">PQ_VectorInvSqrtFixed32</a>		513
32.6.42 <a href="#">PQ_VectorEtoxFixed32</a>		514
32.6.43 <a href="#">PQ_VectorEtonxFixed32</a>		514
32.6.44 <a href="#">PQ_VectorSinQ15</a>		514

Section No.	Title	Page No.
32.6.45	PQ_VectorCosQ15	514
32.6.46	PQ_VectorSinQ31	515
32.6.47	PQ_VectorCosQ31	515
32.6.48	PQ_VectorLnFixed16	515
32.6.49	PQ_VectorInvFixed16	515
32.6.50	PQ_VectorSqrtFixed16	516
32.6.51	PQ_VectorInvSqrtFixed16	516
32.6.52	PQ_VectorEtoxFixed16	516
32.6.53	PQ_VectorEtonxFixed16	517
32.6.54	PQ_VectorBiquadDf2F32	517
32.6.55	PQ_VectorBiquadDf2Fixed32	517
32.6.56	PQ_VectorBiquadDf2Fixed16	517
32.6.57	PQ_VectorBiquadCascadeDf2F32	518
32.6.58	PQ_VectorBiquadCascadeDf2Fixed32	518
32.6.59	PQ_VectorBiquadCascadeDf2Fixed16	518
32.6.60	PQ_ArctanFixed	519
32.6.61	PQ_ArctanhFixed	519
32.6.62	PQ_Biquad1Fixed	520
32.6.63	PQ_TransformCFFT	520
32.6.64	PQ_TransformRFFT	520
32.6.65	PQ_TransformIFFT	521
32.6.66	PQ_TransformCDCT	521
32.6.67	PQ_TransformRDCT	522
32.6.68	PQ_TransformIDCT	522
32.6.69	PQ_BiquadBackUpInternalState	522
32.6.70	PQ_BiquadRestoreInternalState	522
32.6.71	PQ_BiquadCascadeDf2Init	523
32.6.72	PQ_BiquadCascadeDf2F32	523
32.6.73	PQ_BiquadCascadeDf2Fixed32	523
32.6.74	PQ_BiquadCascadeDf2Fixed16	524
32.6.75	PQ_FIR	525
32.6.76	PQ_FIRIncrement	525
32.6.77	PQ_MatrixAddition	526
32.6.78	PQ_MatrixSubtraction	527
32.6.79	PQ_MatrixMultiplication	527
32.6.80	PQ_MatrixProduct	528
32.6.81	PQ_VectorDotProduct	529
32.6.82	PQ_MatrixInversion	529
32.6.83	PQ_MatrixTranspose	530
32.6.84	PQ_MatrixScale	531
33.1	Overview	532

## Chapter 33 PRINCE: PRINCE bus crypto engine

33.1	Overview	532
------	----------	-----

Section No.	Title	Page No.
<b>33.2 Macro Definition Documentation</b>		<b>534</b>
33.2.1 FSL_PRINCE_DRIVER_VERSION		534
<b>33.3 Enumeration Type Documentation</b>		<b>534</b>
33.3.1 skboot_status_t		534
33.3.2 secure_bool_t		535
33.3.3 prince_region_t		535
33.3.4 prince_lock_t		535
33.3.5 prince_flags_t		535
<b>33.4 Function Documentation</b>		<b>536</b>
33.4.1 PRINCE_EncryptEnable		536
33.4.2 PRINCE_EncryptDisable		536
33.4.3 PRINCE_IsEncryptEnable		536
33.4.4 PRINCE_SetMask		536
33.4.5 PRINCE_SetLock		537
33.4.6 PRINCE_GenNewIV		537
33.4.7 PRINCE_LoadIV		537
33.4.8 PRINCE_SetEncryptForAddressRange		538
33.4.9 PRINCE_GetRegionSREnable		538
33.4.10 PRINCE_GetRegionBaseAddress		539
33.4.11 PRINCE_SetRegionIV		539
33.4.12 PRINCE_SetRegionBaseAddress		540
33.4.13 PRINCE_SetRegionSREnable		540
33.4.14 PRINCE_FlashEraseWithChecker		540
33.4.15 PRINCE_FlashProgramWithChecker		541

## Chapter 34 PUF: Physical Unclonable Function

<b>34.1 Overview</b>		<b>543</b>
<b>34.2 PUF Driver Initialization and deinitialization</b>		<b>543</b>
<b>34.3 Comments about API usage in RTOS</b>		<b>543</b>
<b>34.4 Comments about API usage in interrupt handler</b>		<b>543</b>
<b>34.5 PUF Driver Examples</b>		<b>543</b>
34.5.1 Simple examples		543
<b>34.6 Macro Definition Documentation</b>		<b>544</b>
34.6.1 FSL_PUF_DRIVER_VERSION		544
34.6.2 PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE		545
<b>34.7 Enumeration Type Documentation</b>		<b>545</b>
34.7.1 puf_key_slot_t		545

Section No.	Title	Page No.
34.7.2	anonymous enum .....	545

## Chapter 35 RNG: Random Number Generator

<b>35.1</b>	<b>Overview</b> .....	<b>546</b>
<b>35.2</b>	<b>Get random data from RNG</b> .....	<b>546</b>
<b>35.3</b>	<b>Macro Definition Documentation</b> .....	<b>546</b>
35.3.1	FSL_RNG_DRIVER_VERSION .....	546
<b>35.4</b>	<b>Function Documentation</b> .....	<b>547</b>
35.4.1	RNG_Init .....	547
35.4.2	RNG_Deinit .....	547
35.4.3	RNG_GetRandomData .....	547
35.4.4	RNG_GetRandomWord .....	548

## Chapter 36 SCTimer: SCTimer/PWM (SCT)

<b>36.1</b>	<b>Overview</b> .....	<b>549</b>
<b>36.2</b>	<b>Function groups</b> .....	<b>549</b>
36.2.1	Initialization and deinitialization .....	549
36.2.2	PWM Operations .....	549
36.2.3	Status .....	549
36.2.4	Interrupt .....	549
<b>36.3</b>	<b>SCTimer State machine and operations</b> .....	<b>550</b>
36.3.1	SCTimer event operations .....	550
36.3.2	SCTimer state operations .....	550
36.3.3	SCTimer action operations .....	550
<b>36.4</b>	<b>16-bit counter mode</b> .....	<b>550</b>
<b>36.5</b>	<b>Typical use case</b> .....	<b>551</b>
36.5.1	PWM output .....	551
<b>36.6</b>	<b>Data Structure Documentation</b> .....	<b>556</b>
36.6.1	struct sctimer_pwm_signal_param_t .....	556
36.6.2	struct sctimer_config_t .....	556
<b>36.7</b>	<b>Typedef Documentation</b> .....	<b>557</b>
36.7.1	sctimer_event_callback_t .....	557
<b>36.8</b>	<b>Enumeration Type Documentation</b> .....	<b>557</b>
36.8.1	sctimer_pwm_mode_t .....	557

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
36.8.2	sctimer_counter_t .....	558
36.8.3	sctimer_input_t .....	558
36.8.4	sctimer_out_t .....	558
36.8.5	sctimer_pwm_level_select_t .....	558
36.8.6	sctimer_clock_mode_t .....	559
36.8.7	sctimer_clock_select_t .....	559
36.8.8	sctimer_conflict_resolution_t .....	559
36.8.9	sctimer_event_active_direction_t .....	560
36.8.10	sctimer_interrupt_enable_t .....	560
36.8.11	sctimer_status_flags_t .....	560
<b>36.9</b>	<b>Function Documentation</b>	<b>561</b>
36.9.1	SCTIMER_Init .....	561
36.9.2	SCTIMER_Deinit .....	561
36.9.3	SCTIMER_GetDefaultConfig .....	561
36.9.4	SCTIMER_SetupPwm .....	562
36.9.5	SCTIMER_UpdatePwmDutycycle .....	562
36.9.6	SCTIMER_EnableInterrupts .....	563
36.9.7	SCTIMER_DisableInterrupts .....	563
36.9.8	SCTIMER_GetEnabledInterrupts .....	563
36.9.9	SCTIMER_GetStatusFlags .....	564
36.9.10	SCTIMER_ClearStatusFlags .....	564
36.9.11	SCTIMER_StartTimer .....	564
36.9.12	SCTIMER_StopTimer .....	565
36.9.13	SCTIMER_CreateAndScheduleEvent .....	565
36.9.14	SCTIMER_ScheduleEvent .....	566
36.9.15	SCTIMER_IncreaseState .....	566
36.9.16	SCTIMER_GetCurrentState .....	566
36.9.17	SCTIMER_SetCounterState .....	567
36.9.18	SCTIMER_GetCounterState .....	567
36.9.19	SCTIMER_SetupCaptureAction .....	567
36.9.20	SCTIMER_SetCallback .....	568
36.9.21	SCTIMER_SetupStateLdMethodAction .....	568
36.9.22	SCTIMER_SetupNextStateActionwithLdMethod .....	569
36.9.23	SCTIMER_SetupNextStateAction .....	569
36.9.24	SCTIMER_SetupEventActiveDirection .....	570
36.9.25	SCTIMER_SetupOutputSetAction .....	570
36.9.26	SCTIMER_SetupOutputClearAction .....	570
36.9.27	SCTIMER_SetupOutputToggleAction .....	571
36.9.28	SCTIMER_SetupCounterLimitAction .....	572
36.9.29	SCTIMER_SetupCounterStopAction .....	572
36.9.30	SCTIMER_SetupCounterStartAction .....	572
36.9.31	SCTIMER_SetupCounterHaltAction .....	573
36.9.32	SCTIMER_SetupDmaTriggerAction .....	573
36.9.33	SCTIMER_SetCOUNTValue .....	573

Section No.	Title	Page No.
36.9.34	SCTIMER_GetCOUNTValue .....	574
36.9.35	SCTIMER_SetEventInState .....	574
36.9.36	SCTIMER_ClearEventInState .....	574
36.9.37	SCTIMER_GetEventInState .....	575
36.9.38	SCTIMER_EventHandleIRQ .....	575
<b>Chapter 37 SDIF: SD/MMC/SDIO card interface</b>		
<b>37.1</b>	<b>Overview .....</b>	<b>576</b>
<b>37.2</b>	<b>Typical use case .....</b>	<b>576</b>
37.2.1	sdif Operation .....	576
<b>37.3</b>	<b>Data Structure Documentation .....</b>	<b>581</b>
37.3.1	struct sdif_dma_descriptor_t .....	581
37.3.2	struct sdif_dma_config_t .....	582
37.3.3	struct sdif_data_t .....	582
37.3.4	struct sdif_command_t .....	583
37.3.5	struct sdif_transfer_t .....	583
37.3.6	struct sdif_config_t .....	583
37.3.7	struct sdif_capability_t .....	584
37.3.8	struct sdif_transfer_callback_t .....	584
37.3.9	struct sdif_handle_t .....	584
37.3.10	struct sdif_host_t .....	585
<b>37.4</b>	<b>Macro Definition Documentation .....</b>	<b>585</b>
37.4.1	FSL_SDIF_DRIVER_VERSION .....	585
37.4.2	SDIF_CLOCK_RANGE_NEED_DELAY .....	585
37.4.3	SDIF_HIGHSPEED_SAMPLE_DELAY .....	585
37.4.4	SDIF_HIGHSPEED_DRV_DELAY .....	585
37.4.5	SDIF_DEFAULT_MODE_SAMPLE_DELAY .....	586
<b>37.5</b>	<b>Typedef Documentation .....</b>	<b>586</b>
37.5.1	sdif_transfer_function_t .....	586
<b>37.6</b>	<b>Enumeration Type Documentation .....</b>	<b>586</b>
37.6.1	anonymous enum .....	586
37.6.2	anonymous enum .....	586
37.6.3	anonymous enum .....	587
37.6.4	sdif_bus_width_t .....	587
37.6.5	anonymous enum .....	587
37.6.6	anonymous enum .....	588
37.6.7	anonymous enum .....	588
37.6.8	anonymous enum .....	588
37.6.9	anonymous enum .....	589

Section No.	Title	Page No.
37.6.10	anonymous enum .....	589
<b>37.7</b>	<b>Function Documentation .....</b>	<b>589</b>
37.7.1	SDIF_Init .....	589
37.7.2	SDIF_Deinit .....	590
37.7.3	SDIF_SendCardActive .....	590
37.7.4	SDIF_EnableCardClock .....	590
37.7.5	SDIF_EnableLowPowerMode .....	590
37.7.6	SDIF_EnableCardPower .....	591
37.7.7	SDIF_SetCardBusWidth .....	591
37.7.8	SDIF_DetectCardInsert .....	591
37.7.9	SDIF_SetCardClock .....	591
37.7.10	SDIF_Reset .....	592
37.7.11	SDIF_GetCardWriteProtect .....	592
37.7.12	SDIF_AssertHardwareReset .....	592
37.7.13	SDIF_SendCommand .....	593
37.7.14	SDIF_EnableGlobalInterrupt .....	593
37.7.15	SDIF_EnableInterrupt .....	593
37.7.16	SDIF_DisableInterrupt .....	594
37.7.17	SDIF_GetInterruptStatus .....	594
37.7.18	SDIF_GetEnabledInterruptStatus .....	594
37.7.19	SDIF_ClearInterruptStatus .....	594
37.7.20	SDIF_TransferCreateHandle .....	595
37.7.21	SDIF_EnableDmaInterrupt .....	595
37.7.22	SDIF_DisableDmaInterrupt .....	595
37.7.23	SDIF_GetInternalDMAStatus .....	595
37.7.24	SDIF_GetEnabledDMAInterruptStatus .....	596
37.7.25	SDIF_ClearInternalDMAStatus .....	596
37.7.26	SDIF_InternalDMAConfig .....	596
37.7.27	SDIF_EnableInternalDMA .....	597
37.7.28	SDIF_SendReadWait .....	597
37.7.29	SDIF_AbortReadData .....	597
37.7.30	SDIF_EnableCEATAInterrupt .....	597
37.7.31	SDIF_TransferNonBlocking .....	598
37.7.32	SDIF_TransferBlocking .....	598
37.7.33	SDIF_ReleaseDMADescriptor .....	598
37.7.34	SDIF_GetCapability .....	599
37.7.35	SDIF_GetControllerStatus .....	599
37.7.36	SDIF_SendCCSD .....	599
37.7.37	SDIF_ConfigClockDelay .....	599

## Chapter 38 SYSCTL: I2S bridging and signal sharing Configuration

<b>38.1</b>	<b>Overview .....</b>	<b>601</b>
-------------	-----------------------	------------

Section No.	Title	Page No.
<b>38.2 Macro Definition Documentation</b>		<b>602</b>
38.2.1 FSL_SYSCTL_DRIVER_VERSION		602
<b>38.3 Enumeration Type Documentation</b>		<b>602</b>
38.3.1 _sysctl_share_set_index		602
38.3.2 sysctl_fcctrlsel_signal_t		603
38.3.3 _sysctl_share_src		603
38.3.4 _sysctl_dataout_mask		603
38.3.5 sysctl_sharedctrlset_signal_t		603
<b>38.4 Function Documentation</b>		<b>604</b>
38.4.1 SYSCTL_Init		604
38.4.2 SYSCTL_Deinit		604
38.4.3 SYSCTL_SetFlexcommShareSet		604
38.4.4 SYSCTL_SetShareSet		604
38.4.5 SYSCTL_SetShareSetSrc		605
38.4.6 SYSCTL_SetShareSignalSrc		605
<b>Chapter 39 UTICK: MictoTick Timer Driver</b>		
<b>39.1 Overview</b>		<b>606</b>
<b>39.2 Typical use case</b>		<b>606</b>
<b>39.3 Macro Definition Documentation</b>		<b>607</b>
39.3.1 FSL_UTICK_DRIVER_VERSION		607
<b>39.4 Typedef Documentation</b>		<b>607</b>
39.4.1 utick_callback_t		607
<b>39.5 Enumeration Type Documentation</b>		<b>607</b>
39.5.1 utick_mode_t		607
<b>39.6 Function Documentation</b>		<b>607</b>
39.6.1 UTICK_Init		607
39.6.2 UTICK_Deinit		607
39.6.3 UTICK_GetStatusFlags		607
39.6.4 UTICK_ClearStatusFlags		608
39.6.5 UTICK_SetTick		608
39.6.6 UTICK_HandleIRQ		608

## Chapter 40 WWDT: Windowed Watchdog Timer Driver

<b>40.1 Overview</b>	<b>610</b>
<b>40.2 Function groups</b>	<b>610</b>

Section No.	Title	Page No.
40.2.1	Initialization and deinitialization .....	610
40.2.2	Status .....	610
40.2.3	Interrupt .....	610
40.2.4	Watch dog Refresh .....	610
<b>40.3</b>	<b>Typical use case .....</b>	<b>610</b>
<b>40.4</b>	<b>Data Structure Documentation .....</b>	<b>611</b>
40.4.1	struct wwdt_config_t .....	612
<b>40.5</b>	<b>Macro Definition Documentation .....</b>	<b>612</b>
40.5.1	FSL_WWDT_DRIVER_VERSION .....	612
<b>40.6</b>	<b>Enumeration Type Documentation .....</b>	<b>612</b>
40.6.1	_wwdt_status_flags_t .....	612
<b>40.7</b>	<b>Function Documentation .....</b>	<b>612</b>
40.7.1	WWDT_GetDefaultConfig .....	613
40.7.2	WWDT_Init .....	613
40.7.3	WWDT_Deinit .....	613
40.7.4	WWDT_Enable .....	614
40.7.5	WWDT_Disable .....	614
40.7.6	WWDT_GetStatusFlags .....	614
40.7.7	WWDT_ClearStatusFlags .....	615
40.7.8	WWDT_SetWarningValue .....	615
40.7.9	WWDT_SetTimeoutValue .....	615
40.7.10	WWDT_SetWindowValue .....	617
40.7.11	WWDT_Refresh .....	617

## Chapter 41 Debug Console

<b>41.1</b>	<b>Overview .....</b>	<b>618</b>
<b>41.2</b>	<b>Function groups .....</b>	<b>618</b>
41.2.1	Initialization .....	618
41.2.2	Advanced Feature .....	619
41.2.3	SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART .....	623
<b>41.3</b>	<b>Typical use case .....</b>	<b>624</b>
<b>41.4</b>	<b>Macro Definition Documentation .....</b>	<b>626</b>
41.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN .....	626
41.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK .....	626
41.4.3	DEBUGCONSOLE_DISABLE .....	626
41.4.4	SDK_DEBUGCONSOLE .....	626
41.4.5	PRINTF .....	626

Section No.	Title	Page No.
<b>41.5 Function Documentation</b>		<b>626</b>
41.5.1 DbgConsole_Init		626
41.5.2 DbgConsole_Deinit		627
41.5.3 DbgConsole_EnterLowpower		627
41.5.4 DbgConsole_ExitLowpower		627
41.5.5 DbgConsole_Printf		628
41.5.6 DbgConsole_Vprintf		629
41.5.7 DbgConsole_Putchar		629
41.5.8 DbgConsole_Scanf		629
41.5.9 DbgConsole_Getchar		630
41.5.10 DbgConsole_BlockingPrintf		630
41.5.11 DbgConsole_BlockingVprintf		631
41.5.12 DbgConsole_Flush		631
<b>41.6 Semihosting</b>		<b>632</b>
41.6.1 Guide Semihosting for IAR		632
41.6.2 Guide Semihosting for Keil µVision		632
41.6.3 Guide Semihosting for MCUXpresso IDE		633
41.6.4 Guide Semihosting for ARMGCC		633
<b>41.7 SWO</b>		<b>636</b>
41.7.1 Guide SWO for SDK		636
41.7.2 Guide SWO for Keil µVision		637
41.7.3 Guide SWO for MCUXpresso IDE		638
41.7.4 Guide SWO for ARMGCC		638
<b>Chapter 42 Notification Framework</b>		
<b>42.1 Overview</b>		<b>639</b>
<b>42.2 Notifier Overview</b>		<b>639</b>
<b>42.3 Data Structure Documentation</b>		<b>641</b>
42.3.1 struct notifier_notification_block_t		641
42.3.2 struct notifier_callback_config_t		642
42.3.3 struct notifier_handle_t		642
<b>42.4 Typedef Documentation</b>		<b>643</b>
42.4.1 notifier_user_config_t		643
42.4.2 notifier_user_function_t		643
42.4.3 notifier_callback_t		644
<b>42.5 Enumeration Type Documentation</b>		<b>644</b>
42.5.1 _notifier_status		644
42.5.2 notifier_policy_t		645
42.5.3 notifier_notification_type_t		645

Section No.	Title	Page No.
42.5.4	notifier_callback_type_t .....	645
<b>42.6</b>	<b>Function Documentation .....</b>	<b>645</b>
42.6.1	NOTIFIER_CreateHandle .....	646
42.6.2	NOTIFIER_SwitchConfig .....	647
42.6.3	NOTIFIER_GetErrorCallbackIndex .....	648
<b>Chapter 43 Shell</b>		
<b>43.1</b>	<b>Overview .....</b>	<b>649</b>
<b>43.2</b>	<b>Function groups .....</b>	<b>649</b>
43.2.1	Initialization .....	649
43.2.2	Advanced Feature .....	649
43.2.3	Shell Operation .....	649
<b>43.3</b>	<b>Data Structure Documentation .....</b>	<b>651</b>
43.3.1	struct shell_command_t .....	651
<b>43.4</b>	<b>Macro Definition Documentation .....</b>	<b>652</b>
43.4.1	SHELL_NON_BLOCKING_MODE .....	652
43.4.2	SHELL_AUTO_COMPLETE .....	652
43.4.3	SHELL_BUFFER_SIZE .....	652
43.4.4	SHELL_MAX_ARGS .....	652
43.4.5	SHELL_HISTORY_COUNT .....	652
43.4.6	SHELL_HANDLE_SIZE .....	652
43.4.7	SHELL_USE_COMMON_TASK .....	652
43.4.8	SHELL_TASK_PRIORITY .....	652
43.4.9	SHELL_TASK_STACK_SIZE .....	652
43.4.10	SHELL_HANDLE_DEFINE .....	653
43.4.11	SHELL_COMMAND_DEFINE .....	653
43.4.12	SHELL_COMMAND .....	654
<b>43.5</b>	<b>Typedef Documentation .....</b>	<b>654</b>
43.5.1	cmd_function_t .....	654
<b>43.6</b>	<b>Enumeration Type Documentation .....</b>	<b>654</b>
43.6.1	shell_status_t .....	654
<b>43.7</b>	<b>Function Documentation .....</b>	<b>654</b>
43.7.1	SHELL_Init .....	654
43.7.2	SHELL_RegisterCommand .....	656
43.7.3	SHELL_UnregisterCommand .....	657
43.7.4	SHELL_Write .....	657
43.7.5	SHELL_Printf .....	657
43.7.6	SHELL_WriteSynchronization .....	658

Section No.	Title	Page No.
43.7.7	SHELL_PrintfSynchronization .....	658
43.7.8	SHELL_ChangePrompt .....	659
43.7.9	SHELL_PrintPrompt .....	659
43.7.10	SHELL_Task .....	659
43.7.11	SHELL_checkRunningInIsr .....	660

## Chapter 44 Cards: Secure Digital Card/Embedded MultiMedia Card/SDIO Card

<b>44.1</b>	<b>Overview .....</b>	<b>661</b>
<b>44.2</b>	<b>SDIO Card Driver .....</b>	<b>662</b>
44.2.1	Overview .....	662
44.2.2	SDIO CARD Operation .....	662
44.2.3	Data Structure Documentation .....	664
44.2.4	Macro Definition Documentation .....	666
44.2.5	Enumeration Type Documentation .....	666
44.2.6	Function Documentation .....	666
<b>44.3</b>	<b>SD Card Driver .....</b>	<b>683</b>
44.3.1	Overview .....	683
44.3.2	SD CARD Operation .....	683
44.3.3	Data Structure Documentation .....	686
44.3.4	Macro Definition Documentation .....	687
44.3.5	Enumeration Type Documentation .....	687
44.3.6	Function Documentation .....	687
<b>44.4</b>	<b>MMC Card Driver .....</b>	<b>697</b>
44.4.1	Overview .....	697
44.4.2	MMC CARD Operation .....	697
44.4.3	Data Structure Documentation .....	699
44.4.4	Macro Definition Documentation .....	701
44.4.5	Enumeration Type Documentation .....	701
44.4.6	Function Documentation .....	701
<b>44.5</b>	<b>SDMMC HOST Driver .....</b>	<b>715</b>
44.5.1	Overview .....	715
<b>44.6</b>	<b>SDMMC OSA .....</b>	<b>716</b>
44.6.1	Overview .....	716
44.6.2	Data Structure Documentation .....	717
44.6.3	Function Documentation .....	717
44.6.4	SDIF HOST adapter Driver .....	721
<b>44.7</b>	<b>SDMMC Common .....</b>	<b>732</b>
44.7.1	Overview .....	732
44.7.2	Data Structure Documentation .....	751

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
44.7.3	Macro Definition Documentation .....	764
44.7.4	Enumeration Type Documentation .....	764
44.7.5	Function Documentation .....	781
<b>Chapter 45 CODEC Driver</b>		
<b>45.1</b>	<b>Overview .....</b>	<b>786</b>
<b>45.2</b>	<b>CODEC Common Driver .....</b>	<b>787</b>
45.2.1	Overview .....	787
45.2.2	Data Structure Documentation .....	792
45.2.3	Macro Definition Documentation .....	793
45.2.4	Enumeration Type Documentation .....	793
45.2.5	Function Documentation .....	798
<b>45.3</b>	<b>CODEC I2C Driver .....</b>	<b>802</b>
45.3.1	Overview .....	802
45.3.2	Data Structure Documentation .....	803
45.3.3	Enumeration Type Documentation .....	803
45.3.4	Function Documentation .....	803
<b>45.4</b>	<b>CS42888 Driver .....</b>	<b>806</b>
45.4.1	Overview .....	806
45.4.2	Data Structure Documentation .....	808
45.4.3	Macro Definition Documentation .....	809
45.4.4	Enumeration Type Documentation .....	809
45.4.5	Function Documentation .....	810
45.4.6	CS42888 Adapter .....	816
<b>45.5</b>	<b>DA7212 Driver .....</b>	<b>824</b>
45.5.1	Overview .....	824
45.5.2	Data Structure Documentation .....	827
45.5.3	Macro Definition Documentation .....	828
45.5.4	Enumeration Type Documentation .....	828
45.5.5	Function Documentation .....	830
45.5.6	DA7212 Adapter .....	835
<b>45.6</b>	<b>SGTL5000 Driver .....</b>	<b>843</b>
45.6.1	Overview .....	843
45.6.2	Data Structure Documentation .....	845
45.6.3	Macro Definition Documentation .....	846
45.6.4	Enumeration Type Documentation .....	846
45.6.5	Function Documentation .....	848
45.6.6	SGTL5000 Adapter .....	854
<b>45.7</b>	<b>WM8960 Driver .....</b>	<b>862</b>

Section No.	Title	Page No.
45.7.1	Overview .....	862
45.7.2	Data Structure Documentation .....	865
45.7.3	Macro Definition Documentation .....	867
45.7.4	Enumeration Type Documentation .....	867
45.7.5	Function Documentation .....	869
45.7.6	WM8960 Adapter .....	876
<b>45.8</b>	<b>WM8904 Driver .....</b>	<b>884</b>
45.8.1	Overview .....	884
45.8.2	Data Structure Documentation .....	888
45.8.3	Macro Definition Documentation .....	889
45.8.4	Enumeration Type Documentation .....	889
45.8.5	Function Documentation .....	892
45.8.6	WM8904 Adapter .....	901
<b>Chapter 46 Serial Manager</b>		
<b>46.1</b>	<b>Overview .....</b>	<b>909</b>
<b>46.2</b>	<b>Data Structure Documentation .....</b>	<b>912</b>
46.2.1	struct serial_manager_config_t .....	912
46.2.2	struct serial_manager_callback_message_t .....	912
<b>46.3</b>	<b>Macro Definition Documentation .....</b>	<b>913</b>
46.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE .....	913
46.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE .....	913
46.3.3	SERIAL_MANAGER_USE_COMMON_TASK .....	913
46.3.4	SERIAL_MANAGER_HANDLE_SIZE .....	913
46.3.5	SERIAL_MANAGER_HANDLE_DEFINE .....	913
46.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE .....	913
46.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE .....	914
46.3.8	SERIAL_MANAGER_TASK_PRIORITY .....	914
46.3.9	SERIAL_MANAGER_TASK_STACK_SIZE .....	914
<b>46.4</b>	<b>Enumeration Type Documentation .....</b>	<b>914</b>
46.4.1	serial_port_type_t .....	914
46.4.2	serial_manager_type_t .....	915
46.4.3	serial_manager_status_t .....	915
<b>46.5</b>	<b>Function Documentation .....</b>	<b>915</b>
46.5.1	SerialManager_Init .....	915
46.5.2	SerialManager_Deinit .....	916
46.5.3	SerialManager_OpenWriteHandle .....	917
46.5.4	SerialManager_CloseWriteHandle .....	918
46.5.5	SerialManager_OpenReadHandle .....	918

<b>Section No.</b>	<b>Title</b>	<b>Page No.</b>
46.5.6	SerialManager_CloseReadHandle .....	919
46.5.7	SerialManager_WriteBlocking .....	920
46.5.8	SerialManager_ReadBlocking .....	920
46.5.9	SerialManager_EnterLowpower .....	921
46.5.10	SerialManager_ExitLowpower .....	921
46.5.11	SerialManager_SetLowpowerCriticalCb .....	922
<b>46.6</b>	<b>Serial Port Uart .....</b>	<b>923</b>
46.6.1	Overview .....	923
46.6.2	Enumeration Type Documentation .....	923
<b>46.7</b>	<b>Serial Port USB .....</b>	<b>924</b>
46.7.1	Overview .....	924
46.7.2	Data Structure Documentation .....	924
46.7.3	Enumeration Type Documentation .....	925
46.7.4	USB Device Configuration .....	926
<b>46.8</b>	<b>Serial Port SWO .....</b>	<b>927</b>
46.8.1	Overview .....	927
46.8.2	Data Structure Documentation .....	927
46.8.3	Enumeration Type Documentation .....	927
<b>Chapter 47 I2s_dma_driver</b>		
<b>47.1</b>	<b>Overview .....</b>	<b>928</b>
<b>47.2</b>	<b>Data Structure Documentation .....</b>	<b>929</b>
47.2.1	struct _i2s_dma_handle .....	929
<b>47.3</b>	<b>Macro Definition Documentation .....</b>	<b>929</b>
47.3.1	FSL_I2S_DMA_DRIVER_VERSION .....	929
<b>47.4</b>	<b>Typedef Documentation .....</b>	<b>929</b>
47.4.1	i2s_dma_transfer_callback_t .....	929
<b>47.5</b>	<b>Function Documentation .....</b>	<b>930</b>
47.5.1	I2S_TxTransferCreateHandleDMA .....	930
47.5.2	I2S_TxTransferSendDMA .....	930
47.5.3	I2S_TransferAbortDMA .....	931
47.5.4	I2S_RxTransferCreateHandleDMA .....	931
47.5.5	I2S_RxTransferReceiveDMA .....	931
47.5.6	I2S_DMACallback .....	932
47.5.7	I2S_TransferInstallLoopDMADescriptorMemory .....	932
47.5.8	I2S_TransferSendLoopDMA .....	933
47.5.9	I2S_TransferReceiveLoopDMA .....	933

Section No.	Title	Page No.
<b>Chapter 48 Hashcrypt_driver</b>		
<b>48.1 Overview</b>	.....	<b>935</b>
<b>48.2 Macro Definition Documentation</b>	.....	<b>935</b>
48.2.1 FSL_HASHCRYPT_DRIVER_VERSION	.....	935
<b>48.3 Enumeration Type Documentation</b>	.....	<b>936</b>
48.3.1 hashcrypt_algo_t	.....	936
<b>48.4 Function Documentation</b>	.....	<b>937</b>
48.4.1 HASHCRYPT_Init	.....	937
48.4.2 HASHCRYPT_Deinit	.....	937
<b>Chapter 49 Skboot_authenticate</b>		
<b>49.1 Overview</b>	.....	<b>938</b>
<b>49.2 Enumeration Type Documentation</b>	.....	<b>938</b>
49.2.1 skboot_status_t	.....	938
49.2.2 secure_bool_t	.....	939
<b>49.3 Function Documentation</b>	.....	<b>939</b>
49.3.1 skboot_authenticate	.....	939
49.3.2 CODEC Adapter	.....	940

# Chapter 1

## Introduction

The MCUXpresso Software Development Kit (MCUXpresso SDK) is a collection of software enablement for NXP Microcontrollers that includes peripheral drivers, multicore support and integrated RTOS support for FreeRTOS™. In addition to the base enablement, the MCUXpresso SDK is augmented with demo applications, driver example projects, and API documentation to help users quickly leverage the support provided by MCUXpresso SDK. The [MCUXpresso SDK Web Builder](#) is available to provide access to all MCUXpresso SDK packages. See the *MCUXpresso Software Development Kit (SDK) Release Notes* (document MCUXSDKRNN) in the Supported Devices section at [MCUXpresso-SDK: Software Development Kit for MCUXpresso](#) for details.

The MCUXpresso SDK is built with the following runtime software components:

- Arm® and DSP standard libraries, and CMSIS-compliant device header files which provide direct access to the peripheral registers.
- Peripheral drivers that provide stateless, high-performance, ease-of-use APIs. Communication drivers provide higher-level transactional APIs for a higher-performance option.
- RTOS wrapper driver built on top of MCUXpresso SDK peripheral drivers and leverage native RTOS services to better comply to the RTOS cases.
- Real time operation systems (RTOS) for FreeRTOS OS.
- Stacks and middleware in source or object formats including:
- CMSIS-DSP, a suite of common signal processing functions.
- The MCUXpresso SDK comes complete with software examples demonstrating the usage of the peripheral drivers, RTOS wrapper drivers, middleware, and RTOSes.

The peripheral drivers and RTOS driver wrappers can be used across multiple devices within the product family without modification. The configuration items for each driver are encapsulated into C language data structures. Device-specific configuration information is provided as part of the MCUXpresso SDK and need not be modified by the user. If necessary, the user is able to modify the peripheral driver and RTOS wrapper driver configuration during runtime. The driver examples demonstrate how to configure the drivers by passing the proper configuration data to the APIs. The folder structure is organized to reduce the total number of includes required to compile a project.

The rest of this document describes the API references in detail for the peripheral drivers and RTOS wrapper drivers. For the latest version of this and other MCUXpresso SDK documents, see the [mcuxpresso.nxp.com/apidoc/](#).



<b>Deliverable</b>	<b>Location</b>
Demo Applications	<install_dir>/boards/<board_name>/demo_apps
Driver Examples	<install_dir>/boards/<board_name>/driver_examples
Documentation	<install_dir>/docs
Middleware	<install_dir>/middleware
Drivers	<install_dir>/<device_name>/drivers/
CMSIS Standard Arm Cortex-M Headers, math and DSP Libraries	<install_dir>/CMSIS
Device Startup and Linker	<install_dir>/<device_name>/<toolchain>/
MCUXpresso SDK Utilities	<install_dir>/devices/<device_name>/utilities
RTOS Kernel Code	<install_dir>/rtos

### **MCUXpresso SDK Folder Structure**

# Chapter 2

## Trademarks

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.

How to Reach Us:

Home Page: [nxp.com](http://nxp.com)

Web Support: [nxp.com/support](http://nxp.com/support)

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](http://nxp.com/SalesTermsandConditions).

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EM-BRACE, GREENCHIP, HITAG, I2C BUS,ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4M-OBILE, 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, mobileGT, 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. AMBA, Arm, Arm7, Arm7TD-MI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. 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.

© 2021 NXP B.V.

# Chapter 3

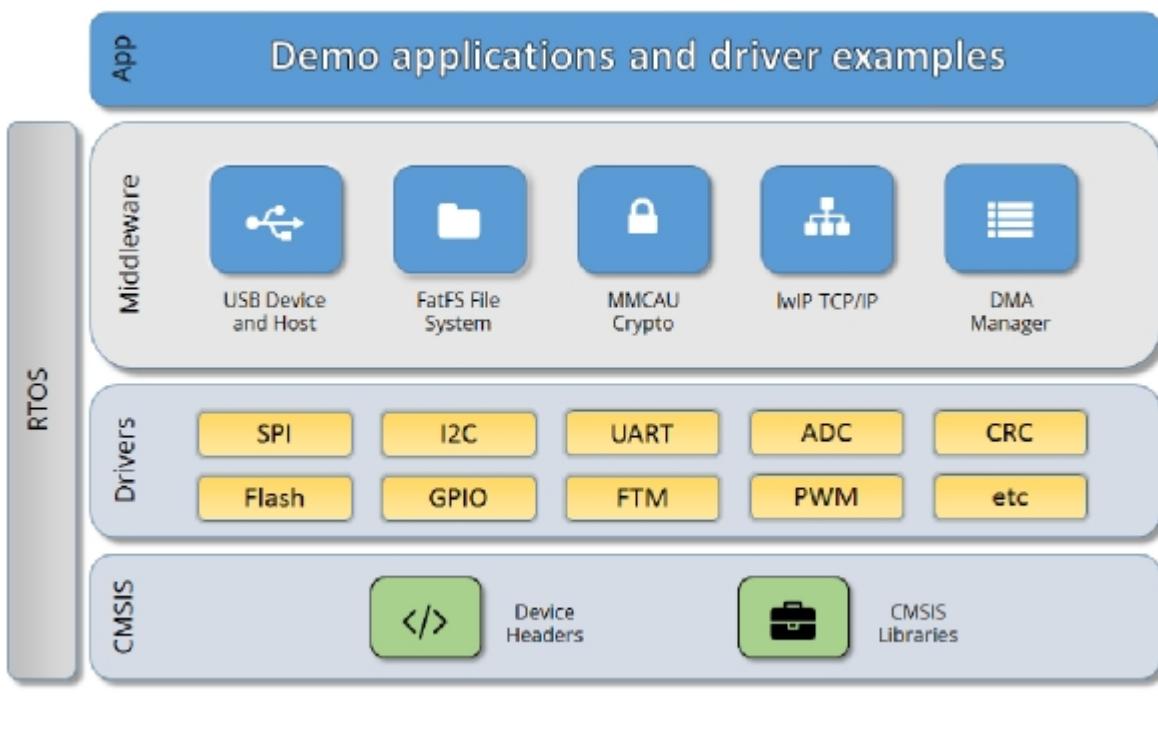
## Architectural Overview

This chapter provides the architectural overview for the MCUXpresso Software Development Kit (MCUXpresso SDK). It describes each layer within the architecture and its associated components.

### Overview

The MCUXpresso SDK architecture consists of five key components listed below.

1. The Arm Cortex Microcontroller Software Interface Standard (CMSIS) CORE compliance device-specific header files, SOC Header, and CMSIS math/DSP libraries.
2. Peripheral Drivers
3. Real-time Operating Systems (RTOS)
4. Stacks and Middleware that integrate with the MCUXpresso SDK
5. Demo Applications based on the MCUXpresso SDK



### MCU header files

Each supported MCU device in the MCUXpresso SDK has an overall System-on Chip (SoC) memory-

mapped header file. This header file contains the memory map and register base address for each peripheral and the IRQ vector table with associated vector numbers. The overall SoC header file provides access to the peripheral registers through pointers and predefined bit masks. In addition to the overall SoC memory-mapped header file, the MCUXpresso SDK includes a feature header file for each device. The feature header file allows NXP to deliver a single software driver for a given peripheral. The feature file ensures that the driver is properly compiled for the target SOC.

## CMSIS Support

Along with the SoC header files and peripheral extension header files, the MCUXpresso SDK also includes common CMSIS header files for the Arm Cortex-M core and the math and DSP libraries from the latest CMSIS release. The CMSIS DSP library source code is also included for reference.

## MCUXpresso SDK Peripheral Drivers

The MCUXpresso SDK peripheral drivers mainly consist of low-level functional APIs for the MCU product family on-chip peripherals and also of high-level transactional APIs for some bus drivers/DM-A driver/eDMA driver to quickly enable the peripherals and perform transfers.

All MCUXpresso SDK peripheral drivers only depend on the CMSIS headers, device feature files, fsl\_common.h, and fsl\_clock.h files so that users can easily pull selected drivers and their dependencies into projects. With the exception of the clock/power-relevant peripherals, each peripheral has its own driver. Peripheral drivers handle the peripheral clock gating/ungating inside the drivers during initialization and deinitialization respectively.

Low-level functional APIs provide common peripheral functionality, abstracting the hardware peripheral register accesses into a set of stateless basic functional operations. These APIs primarily focus on the control, configuration, and function of basic peripheral operations. The APIs hide the register access details and various MCU peripheral instantiation differences so that the application can be abstracted from the low-level hardware details. The API prototypes are intentionally similar to help ensure easy portability across supported MCUXpresso SDK devices.

Transactional APIs provide a quick method for customers to utilize higher-level functionality of the peripherals. The transactional APIs utilize interrupts and perform asynchronous operations without user intervention. Transactional APIs operate on high-level logic that requires data storage for internal operation context handling. However, the Peripheral Drivers do not allocate this memory space. Rather, the user passes in the memory to the driver for internal driver operation. Transactional APIs ensure the NVIC is enabled properly inside the drivers. The transactional APIs do not meet all customer needs, but provide a baseline for development of custom user APIs.

Note that the transactional drivers never disable an NVIC after use. This is due to the shared nature of interrupt vectors on devices. It is up to the user to ensure that NVIC interrupts are properly disabled after usage is complete.

## Interrupt handling for transactional APIs

A double weak mechanism is introduced for drivers with transactional API. The double weak indicates two levels of weak vector entries. See the examples below:

```
PUBWEAK SPI0_IRQHandler  
PUBWEAK SPI0_DriverIRQHandler  
SPI0_IRQHandler
```

```
LDR      R0, =SPI0_DriverIRQHandler  
BX      R0
```

The first level of the weak implementation are the functions defined in the vector table. In the devices/<DEVICE\_NAME>/<TOOLCHAIN>/startup\_<DEVICE\_NAME>.s/.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0\_DriverIRQHandler) jumps to itself (B). The MCUXpresso SDK drivers with transactional APIs provide the reimplementation of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0\_DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

The reason for implementing the double weak functions is to provide a better user experience when using the transactional APIs. For drivers with a transactional function, call the transactional APIs and the drivers complete the interrupt-driven flow. Users are not required to redefine the vector entries out of the box. At the same time, if users are not satisfied by the second layer weak function implemented in the MCUXpresso SDK drivers, users can redefine the first layer weak function and implement their own interrupt handler functions to suit their implementation.

The limitation of the double weak mechanism is that it cannot be used for peripherals that share the same vector entry. For this use case, redefine the first layer weak function to enable the desired peripheral interrupt functionality. For example, if the MCU's UART0 and UART1 share the same vector entry, redefine the UART0\_UART1\_IRQHandler according to the use case requirements.

## Feature Header Files

The peripheral drivers are designed to be reusable regardless of the peripheral functional differences from one MCU device to another. An overall Peripheral Feature Header File is provided for the MCUXpresso SDK-supported MCU device to define the features or configuration differences for each sub-family device.

## Application

See the *Getting Started with MCUXpresso SDK* document (MCUXSDKGSUG).

# Chapter 4

## Clock Driver

### 4.1 Overview

The MCUXpresso SDK provides APIs for MCUXpresso SDK devices' clock operation.

The clock driver supports:

- Clock generator (PLL, FLL, and so on) configuration
- Clock mux and divider configuration
- Getting clock frequency

### Files

- file [fsl\\_clock.h](#)

### Data Structures

- struct [firc\\_trim\\_config\\_t](#)  
*firc trim configuration. [More...](#)*
- struct [sirc\\_trim\\_config\\_t](#)  
*sirc trim configuration. [More...](#)*
- struct [vbat\\_osc\\_config\\_t](#)  
*The structure of oscillator configuration. [More...](#)*
- struct [pll\\_config\\_t](#)  
*PLL configuration structure. [More...](#)*
- struct [pll\\_setup\\_t](#)  
*PLL0 setup structure This structure can be used to pre-build a PLL setup configuration at run-time and quickly set the PLL to the configuration. [More...](#)*

### Macros

- #define [FSL\\_SDK\\_DISABLE\\_DRIVER\\_CLOCK\\_CONTROL](#) 0  
*Configure whether driver controls clock.*
- #define [CLOCK\\_USR\\_CFG\\_PLL\\_CONFIG\\_CACHE\\_COUNT](#) 2U  
*User-defined the size of cache for CLOCK\_PllGetConfig() function.*
- #define [ROM\\_CLOCKS](#)  
*Clock ip name array for ROM.*
- #define [SRAM\\_CLOCKS](#)  
*Clock ip name array for SRAM.*
- #define [FMC\\_CLOCKS](#)  
*Clock ip name array for FMC.*
- #define [INPUTMUX\\_CLOCKS](#)  
*Clock ip name array for INPUTMUX.*
- #define [ETH\\_CLOCKS](#)  
*Clock ip name array for ENET.*

- #define **GPIO\_CLOCKS**  
*Clock ip name array for GPIO.*
- #define **PINT\_CLOCKS**  
*Clock ip name array for PINT.*
- #define **DMA\_CLOCKS**  
*Clock ip name array for DMA.*
- #define **EDMA\_CLOCKS**  
*Clock gate name array for EDMA.*
- #define **CRC\_CLOCKS**  
*Clock ip name array for CRC.*
- #define **WWDT\_CLOCKS**  
*Clock ip name array for WWDT.*
- #define **MAILBOX\_CLOCKS**  
*Clock ip name array for Mailbox.*
- #define **LPADC\_CLOCKS**  
*Clock ip name array for LPADC.*
- #define **MRT\_CLOCKS**  
*Clock ip name array for MRT.*
- #define **OSTIMER\_CLOCKS**  
*Clock ip name array for OSTIMER.*
- #define **SCT\_CLOCKS**  
*Clock ip name array for SCT0.*
- #define **UTICK\_CLOCKS**  
*Clock ip name array for UTICK.*
- #define **LP\_FLEXCOMM\_CLOCKS**  
*Clock ip name array for LP\_FLEXCOMM.*
- #define **LPUART\_CLOCKS**  
*Clock ip name array for LPUART.*
- #define **LPI2C\_CLOCKS**  
*Clock ip name array for LPI2C.*
- #define **LPSPI\_CLOCKS**  
*Clock ip name array for LSPI.*
- #define **CTIMER\_CLOCKS**  
*Clock ip name array for CTIMER.*
- #define **FREQME\_CLOCKS**  
*Clock ip name array for FREQME.*
- #define **POWERQUAD\_CLOCKS**  
*Clock ip name array for PowerQuad.*
- #define **PLU\_CLOCKS**  
*Clock ip name array for PLU.*
- #define **PUF\_CLOCKS**  
*Clock ip name array for PUF.*
- #define **VREF\_CLOCKS**  
*Clock ip name array for VREF.*
- #define **LPDAC\_CLOCKS**  
*Clock ip name array for LPDAC.*
- #define **HPDAC\_CLOCKS**  
*Clock ip name array for HPDAC.*
- #define **PWM\_CLOCKS**  
*Clock ip name array for PWM.*
- #define **ENC\_CLOCKS**

- `#define FLEXIO_CLOCKS`  
*Clock ip name array for ENC.*
- `#define FLEXCAN_CLOCKS`  
*Clock ip name array for FLEXCAN.*
- `#define EMVSIM_CLOCKS`  
*Clock ip name array for EMVSIM.*
- `#define I3C_CLOCKS`  
*Clock ip name array for I3C.*
- `#define USDHC_CLOCKS`  
*Clock ip name array for USDHC.*
- `#define FLEXSPI_CLOCKS`  
*Clock ip name array for FLEXSPI.*
- `#define SAI_CLOCKS`  
*Clock ip name array for SAI.*
- `#define RTC_CLOCKS`  
*Clock ip name array for RTC.*
- `#define PDM_CLOCKS`  
*Clock ip name array for PDM.*
- `#define ERM_CLOCKS`  
*Clock ip name array for ERM.*
- `#define EIM_CLOCKS`  
*Clock ip name array for EIM.*
- `#define OPAMP_CLOCKS`  
*Clock ip name array for OPAMP.*
- `#define TSI_CLOCKS`  
*Clock ip name array for TSI.*
- `#define TRNG_CLOCKS`  
*Clock ip name array for TRNG.*
- `#define LPCMP_CLOCKS`  
*Clock ip name array for LPCMP.*
- `#define CLK_GATE_REG_OFFSET_SHIFT 8U`  
*Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.*
- `#define BUS_CLK kCLOCK_BusClk`  
*Peripherals clock source definition.*
- `#define CLK_ATTACH_ID(mux, sel, pos) (((uint32_t)(mux) << 0U) | (((uint32_t)(sel) + 1U) & 0xFU) << 12U) << ((uint32_t)(pos)*16U))`  
*Clock Mux Switches The encoding is as follows each connection identified is 32bits wide while 24bits are valuable starting from LSB upwards.*
- `#define PLL_CONFIGFLAG_FORCENOFRAC (1U << 2U)`  
*PLL configuration structure flags for 'flags' field These flags control how the PLL configuration function sets up the PLL setup structure.*

## Enumerations

- enum `clock_ip_name_t` {
   
`kCLOCK_IpInvalid` = 0U,
   
`kCLOCK_None` = 0U,
   
`kCLOCK_Rom` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 1),
   
`kCLOCK_Sram1` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 2),
   
`kCLOCK_Sram2` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 3),
   
`kCLOCK_Sram3` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 4),
   
`kCLOCK_Sram4` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 5),
   
`kCLOCK_Sram5` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 6),
   
`kCLOCK_Sram6` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 7),
   
`kCLOCK_Sram7` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 8),
   
`kCLOCK_Fmu` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 9),
   
`kCLOCK_Fmc` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 10),
   
`kCLOCK_Flexspi` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 11),
   
`kCLOCK_InputMux0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 12),
   
`kCLOCK_InputMux` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 12),
   
`kCLOCK_Port0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 13),
   
`kCLOCK_Port1` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 14),
   
`kCLOCK_Port2` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 15),
   
`kCLOCK_Port3` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 16),
   
`kCLOCK_Port4` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 17),
   
`kCLOCK_Gpio0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 19),
   
`kCLOCK_Gpio1` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 20),
   
`kCLOCK_Gpio2` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 21),
   
`kCLOCK_Gpio3` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 22),
   
`kCLOCK_Gpio4` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 23),
   
`kCLOCK_Pint` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 25),
   
`kCLOCK_Dma0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 26),
   
`kCLOCK_Crc0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 27),
   
`kCLOCK_Wwdt0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 28),
   
`kCLOCK_Wwdt1` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 29),
   
`kCLOCK_Mailbox` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, 31),
   
`kCLOCK_Mrt` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 0),
   
`kCLOCK_OsTimer` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 1),
   
`kCLOCK_Sct` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 2),
   
`kCLOCK_Adc0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 3),
   
`kCLOCK_Adc1` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 4),
   
`kCLOCK_Dac0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 5),
   
`kCLOCK_Rtc0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 6),
   
`kCLOCK_Evsim0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 8),
   
`kCLOCK_Evsim1` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 9),
   
`kCLOCK_Utick` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 10),
   
`kCLOCK_LPFlexComm0` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 11),
   
`kCLOCK_LPFlexComm1` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 12),
   
`kCLOCK_LPFlexComm2` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 13),
   
`kCLOCK_LPFlexComm3` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 14),
   
`kCLOCK_LPFlexComm4` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 15),
   
`kCLOCK_LPFlexComm5` = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL1, 16),

```
kCLOCK_Sema42 = CLK_GATE_DEFINE(AHB_CLK_CTRL3, 27) }
```

*Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.*

- enum `clock_name_t` {  
  kCLOCK\_CoreSysClk,  
  kCLOCK\_BusClk,  
  kCLOCK\_SystickClk0,  
  kCLOCK\_SystickClk1,  
  kCLOCK\_ClockOut,  
  kCLOCK\_Fro12M,  
  kCLOCK\_Clk1M,  
  kCLOCK\_FroHf,  
  kCLOCK\_Clk48M,  
  kCLOCK\_Clk144M,  
  kCLOCK\_Clk16K0,  
  kCLOCK\_Clk16K1,  
  kCLOCK\_Clk16K2,  
  kCLOCK\_Clk16K3,  
  kCLOCK\_ExtClk,  
  kCLOCK\_Osc32K0,  
  kCLOCK\_Osc32K1,  
  kCLOCK\_Osc32K2,  
  kCLOCK\_Osc32K3,  
  kCLOCK\_PlloOut,  
  kCLOCK\_Pllo1Out,  
  kCLOCK\_UsbPlloOut,  
  kCLOCK\_LpOsc }

*Clock name used to get clock frequency.*

- enum `clock_attach_id_t` {

kCLK\_IN\_to\_MAIN\_CLK = MUX\_A(CM\_SCGRCCRSCSCLKSEL, 1),  
 kFRO12M\_to\_MAIN\_CLK = MUX\_A(CM\_SCGRCCRSCSCLKSEL, 2),  
 kFRO\_HF\_to\_MAIN\_CLK = MUX\_A(CM\_SCGRCCRSCSCLKSEL, 3),  
 kXTAL32K2\_to\_MAIN\_CLK = MUX\_A(CM\_SCGRCCRSCSCLKSEL, 4),  
 kPLL0\_to\_MAIN\_CLK = MUX\_A(CM\_SCGRCCRSCSCLKSEL, 5),  
 kPLL1\_to\_MAIN\_CLK = MUX\_A(CM\_SCGRCCRSCSCLKSEL, 6),  
 kUSB\_PLL\_to\_MAIN\_CLK = MUX\_A(CM\_SCGRCCRSCSCLKSEL, 7),  
 kNONE\_to\_MAIN\_CLK = MUX\_A(CM\_SCGRCCRSCSCLKSEL, 15),  
 kSYSTICK\_DIV0\_to\_SYSTICK0 = MUX\_A(CM\_SYSTICKCLKSEL0, 0),  
 kCLK\_1M\_to\_SYSTICK0 = MUX\_A(CM\_SYSTICKCLKSEL0, 1),  
 kLPOSC\_to\_SYSTICK0 = MUX\_A(CM\_SYSTICKCLKSEL0, 2),  
 kNONE\_to\_SYSTICK0 = MUX\_A(CM\_SYSTICKCLKSEL0, 7),  
 kSYSTICK\_DIV1\_to\_SYSTICK1 = MUX\_A(CM\_SYSTICKCLKSEL1, 0),  
 kCLK\_1M\_to\_SYSTICK1 = MUX\_A(CM\_SYSTICKCLKSEL1, 1),  
 kLPOSC\_to\_SYSTICK1 = MUX\_A(CM\_SYSTICKCLKSEL1, 2),  
 kNONE\_to\_SYSTICK1 = MUX\_A(CM\_SYSTICKCLKSEL1, 7),  
 kTRACE\_DIV\_to\_TRACE = MUX\_A(CM\_TRACECLKSEL, 0),  
 kCLK\_1M\_to\_TRACE = MUX\_A(CM\_TRACECLKSEL, 1),  
 kLPOSC\_to\_TRACE = MUX\_A(CM\_TRACECLKSEL, 2),  
 kNONE\_to\_TRACE = MUX\_A(CM\_TRACECLKSEL, 7),  
 kCLK\_1M\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 0),  
 kPLL0\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 1),  
 kPLL1\_CLK0\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 2),  
 kFRO\_HF\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 3),  
 kFRO12M\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 4),  
 kSAI0\_MCLK\_IN\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 5),  
 kLPOSC\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 6),  
 kSAI1\_MCLK\_IN\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 8),  
 kSAI0\_TX\_BCLK\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 9),  
 kSAI0\_RX\_BCLK\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 10),  
 kSAI1\_TX\_BCLK\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 11),  
 kSAI1\_RX\_BCLK\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 12),  
 kNONE\_to\_TIMERO = MUX\_A(CM\_CTIMERCLKSEL0, 15),  
 kCLK\_1M\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 0),  
 kPLL0\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 1),  
 kPLL1\_CLK0\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 2),  
 kFRO\_HF\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 3),  
 kFRO12M\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 4),  
 kSAI0\_MCLK\_IN\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 5),  
 kLPOSC\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 6),  
 kSAI1\_MCLK\_IN\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 8),  
 kSAI0\_TX\_BCLK\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 9),  
 kSAI0\_RX\_BCLK\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 10),  
 kSAI1\_TX\_BCLK\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 11),  
 kSAI1\_RX\_BCLK\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 12),  
 kNONE\_to\_TIMER1 = MUX\_A(CM\_CTIMERCLKSEL1, 15),  
 kCLK\_1M\_to\_TIMER2 = MUX\_A(CM\_CTIMERCLKSEL2, 0)

```
kNONE_to_NONE = (int)0x80000000U }
```

*The enumerator of clock attach Id.*

- enum [clock\\_div\\_name\\_t](#) {

```

kCLOCK_DivSystickClk0 = 0,
kCLOCK_DivSystickClk1 = ((0x304 - 0x300) / 4),
kCLOCK_DivTraceClk = ((0x308 - 0x300) / 4),
kCLOCK_DivSlowClk = ((0x378 - 0x300) / 4),
kCLOCK_DivTsiClk = ((0x37C - 0x300) / 4),
kCLOCK_DivAhbClk = ((0x380 - 0x300) / 4),
kCLOCK_DivClkOut = ((0x384 - 0x300) / 4),
kCLOCK_DivFrohfClk = ((0x388 - 0x300) / 4),
kCLOCK_DivWdt0Clk = ((0x38C - 0x300) / 4),
kCLOCK_DivAdc0Clk = ((0x394 - 0x300) / 4),
kCLOCK_DivUsb0Clk = ((0x398 - 0x300) / 4),
kCLOCK_DivSctClk = ((0x3B4 - 0x300) / 4),
kCLOCK_DivPllClk = ((0x3C4 - 0x300) / 4),
kCLOCK_DivCtimer0Clk = ((0x3D0 - 0x300) / 4),
kCLOCK_DivCtimer1Clk = ((0x3D4 - 0x300) / 4),
kCLOCK_DivCtimer2Clk = ((0x3D8 - 0x300) / 4),
kCLOCK_DivCtimer3Clk = ((0x3DC - 0x300) / 4),
kCLOCK_DivCtimer4Clk = ((0x3E0 - 0x300) / 4),
kCLOCK_DivPLL1Clk0 = ((0x3E4 - 0x300) / 4),
kCLOCK_DivPLL1Clk1 = ((0x3E8 - 0x300) / 4),
kCLOCK_DivAdc1Clk = ((0x468 - 0x300) / 4),
kCLOCK_DivDac0Clk = ((0x494 - 0x300) / 4),
kCLOCK_DivDac1Clk = ((0x49C - 0x300) / 4),
kCLOCK_DivDac2Clk = ((0x4A4 - 0x300) / 4),
kCLOCK_DivFlexspiClk = ((0x4AC - 0x300) / 4),
kCLOCK_DivI3c0FClkStc = ((0x538 - 0x300) / 4),
kCLOCK_DivI3c0FClkS = ((0x53C - 0x300) / 4),
kCLOCK_DivI3c0FClk = ((0x540 - 0x300) / 4),
kCLOCK_DivMicfilFClk = ((0x54C - 0x300) / 4),
kCLOCK_DivEspiClk = ((0x554 - 0x300) / 4),
kCLOCK_DivUSdhcClk = ((0x55C - 0x300) / 4),
kCLOCK_DivFlexioClk = ((0x564 - 0x300) / 4),
kCLOCK_DivFlexcan0Clk = ((0x5A4 - 0x300) / 4),
kCLOCK_DivFlexcan1Clk = ((0x5AC - 0x300) / 4),
kCLOCK_DivEnetrmiiClk = ((0x5B4 - 0x300) / 4),
kCLOCK_DivEnetptprefClk = ((0x5BC - 0x300) / 4),
kCLOCK_DivWdt1Clk = ((0x5DC - 0x300) / 4),
kCLOCK_DivCmp0FClk = ((0x5F4 - 0x300) / 4),
kCLOCK_DivCmp0rrClk = ((0x5FC - 0x300) / 4),
kCLOCK_DivCmp1FClk = ((0x604 - 0x300) / 4),
kCLOCK_DivCmp1rrClk = ((0x60C - 0x300) / 4),
kCLOCK_DivCmp2FClk = ((0x614 - 0x300) / 4),
kCLOCK_DivCmp2rrClk = ((0x61C - 0x300) / 4),
kCLOCK_DivFlexcom0Clk = ((0x850 - 0x300) / 4),
kCLOCK_DivFlexcom1Clk = ((0x854 - 0x300) / 4),
kCLOCK_DivFlexcom2Clk = ((0x858 - 0x300) / 4),
kCLOCK_DivFlexcom3Clk = ((0x85C - 0x300) / 4),
kCLOCK_DivFlexcom4Clk = ((0x860 - 0x300) / 4),
kCLOCK_DivFlexcom5Clk = ((0x864 - 0x300) / 4),

```

- `kCLOCK_DivI3c1FClk = ((0xB40 - 0x300) / 4) }`
- Clock dividers.*
- enum `osc32k_clk_gate_id_t` {
   
`kCLOCK_Osc32kToVbat = 0x1,`
  
`kCLOCK_Osc32kToVsys = 0x2,`
  
`kCLOCK_Osc32kToWake = 0x4,`
  
`kCLOCK_Osc32kToMain = 0x8,`
  
`kCLOCK_Osc32kToAll = 0xF }`
- OSC32K clock gate.*
- enum `clk16k_clk_gate_id_t` {
   
`kCLOCK_Clk16KToVbat = 0x1,`
  
`kCLOCK_Clk16KToVsys = 0x2,`
  
`kCLOCK_Clk16KToWake = 0x4,`
  
`kCLOCK_Clk16KToMain = 0x8,`
  
`kCLOCK_Clk16KToAll = 0xF }`
- CLK16K clock gate.*
- enum `clock_ctrl_enable_t` {
   
`kCLOCK_PLU_DEGLITCH_CLK_ENA,`
  
`kCLOCK_FRO1MHZ_CLK_ENA,`
  
`kCLOCK_CLKIN_ENA,`
  
`kCLOCK_FRO_HF_ENA,`
  
`kCLOCK_FRO12MHZ_ENA = SYSCON_CLOCK_CTRL_FRO12MHZ_ENA_MASK,`
  
`kCLOCK_FRO1MHZ_ENA,`
  
`kCLOCK_CLKIN_ENA_FM_USBH_LPT }`
- system clocks enable controls*
- enum `clock_usb_phy_src_t` { `kCLOCK_Usbphy480M = 0` }
- Source of the USB HS PHY.*
- enum `_scg_status` {
   
`kStatus_SCG_Busy = MAKE_STATUS(kStatusGroup_SCG, 1),`
  
`kStatus_SCG_InvalidSrc = MAKE_STATUS(kStatusGroup_SCG, 2) }`
- SCG status return codes.*
- enum `firc_trim_mode_t` {
   
`kSCG_FircTrimNonUpdate = SCG_FIRCCSR_FIRCTREN_MASK,`
  
`kSCG_FircTrimUpdate = SCG_FIRCCSR_FIRCTREN_MASK | SCG_FIRCCSR_FIRCTRUP_-MASK }`
- firc trim mode.*
- enum `firc_trim_src_t` {
   
`kSCG_FircTrimSrcUsb0 = 0U,`
  
`kSCG_FircTrimSrcSysOsc = 2U,`
  
`kSCG_FircTrimSrcRtcOsc = 3U }`
- firc trim source.*
- enum `sirc_trim_mode_t` {
   
`kSCG_SircTrimNonUpdate = SCG_SIRCCSR_SIRCTREN_MASK,`
  
`kSCG_SircTrimUpdate = SCG_SIRCCSR_SIRCTREN_MASK | SCG_SIRCCSR_SIRCTRUP_-MASK }`
- sirc trim mode.*
- enum `sirc_trim_src_t` {

- kSCG\_SircTrimSrcSysOsc = 2U,  
 kSCG\_SircTrimSrcRtcOsc = 3U }  
*sirc trim source.*
- enum `scg_sosc_monitor_mode_t` {  
 kSCG\_SysOscMonitorDisable = 0U,  
 kSCG\_SysOscMonitorInt = SCG\_SOSCCSR\_SOSCCM\_MASK,  
 kSCG\_SysOscMonitorReset }  
*SCG system OSC monitor mode.*
  - enum `scg_rosr_monitor_mode_t` {  
 kSCG\_RoscMonitorDisable = 0U,  
 kSCG\_RoscMonitorInt = SCG\_ROSCCSR\_ROSCCM\_MASK,  
 kSCG\_RoscMonitorReset }  
*SCG ROSC monitor mode.*
  - enum `scg_upll_monitor_mode_t` {  
 kSCG\_UpllMonitorDisable = 0U,  
 kSCG\_UpllMonitorInt = SCG\_UPLLCSR\_UPLLCM\_MASK,  
 kSCG\_UpllMonitorReset }  
*SCG UPLL monitor mode.*
  - enum `scg_pll0_monitor_mode_t` {  
 kSCG\_PlloMonitorDisable = 0U,  
 kSCG\_PlloMonitorInt = SCG\_APPLLCSR\_APLLCM\_MASK,  
 kSCG\_PlloMonitorReset }  
*SCG PLL0 monitor mode.*
  - enum `scg_pll1_monitor_mode_t` {  
 kSCG\_PlloMonitorDisable = 0U,  
 kSCG\_PlloMonitorInt = SCG\_SPLLCSR\_SPLLCM\_MASK,  
 kSCG\_PlloMonitorReset }  
*SCG PLL1 monitor mode.*
  - enum `vbat_osc_xtal_cap_t` {  
 kVBAT\_OscXtal0pFCap = 0x0U,  
 kVBAT\_OscXtal2pFCap = 0x1U,  
 kVBAT\_OscXtal4pFCap = 0x2U,  
 kVBAT\_OscXtal6pFCap = 0x3U,  
 kVBAT\_OscXtal8pFCap = 0x4U,  
 kVBAT\_OscXtal10pFCap = 0x5U,  
 kVBAT\_OscXtal12pFCap = 0x6U,  
 kVBAT\_OscXtal14pFCap = 0x7U,  
 kVBAT\_OscXtal16pFCap = 0x8U,  
 kVBAT\_OscXtal18pFCap = 0x9U,  
 kVBAT\_OscXtal20pFCap = 0xAU,  
 kVBAT\_OscXtal22pFCap = 0xBU,  
 kVBAT\_OscXtal24pFCap = 0xCU,  
 kVBAT\_OscXtal26pFCap = 0xDU,  
 kVBAT\_OscXtal28pFCap = 0xEU,  
 kVBAT\_OscXtal30pFCap = 0xFU }

*The enumerator of internal capacitance of OSC's XTAL pin.*

- enum `vbat_osc_extal_cap_t` {
   
kVBAT\_OscExtal0pFCap = 0x0U,
   
kVBAT\_OscExtal2pFCap = 0x1U,
   
kVBAT\_OscExtal4pFCap = 0x2U,
   
kVBAT\_OscExtal6pFCap = 0x3U,
   
kVBAT\_OscExtal8pFCap = 0x4U,
   
kVBAT\_OscExtal10pFCap = 0x5U,
   
kVBAT\_OscExtal12pFCap = 0x6U,
   
kVBAT\_OscExtal14pFCap = 0x7U,
   
kVBAT\_OscExtal16pFCap = 0x8U,
   
kVBAT\_OscExtal18pFCap = 0x9U,
   
kVBAT\_OscExtal20pFCap = 0xAU,
   
kVBAT\_OscExtal22pFCap = 0xBU,
   
kVBAT\_OscExtal24pFCap = 0xCU,
   
kVBAT\_OscExtal26pFCap = 0xDU,
   
kVBAT\_OscExtal28pFCap = 0xEU,
   
kVBAT\_OscExtal30pFCap = 0xFU }

*The enumerator of internal capacitance of OSC's EXTAL pin.*

- enum `vbat_osc_coarse_adjustment_value_t`

*The enumerator of osc amplifier gain fine adjustment.*

- enum `pll_clk_src_t` {
   
kPlI\_ClkSrcSysOsc = (0 << 25),
   
kPlI\_ClkSrcFirc = (1 << 25),
   
kPlI\_ClkSrcRosc = (2 << 25) }

*PLL clock source.*

- enum `ss_progmodfm_t` {
   
kSS\_MF\_512 = (0 << 2),
   
kSS\_MF\_384 = (1 << 2),
   
kSS\_MF\_256 = (2 << 2),
   
kSS\_MF\_128 = (3 << 2),
   
kSS\_MF\_64 = (4 << 2),
   
kSS\_MF\_32 = (5 << 2),
   
kSS\_MF\_24 = (6 << 2),
   
kSS\_MF\_16 = (7 << 2) }

*PLL Spread Spectrum (SS) Programmable modulation frequency See (MF) field in the PLL0SSCG1 register in the UM.*

- enum `ss_progmoddp_t` {
   
kSS\_MR\_K0 = (0 << 5),
   
kSS\_MR\_K1 = (1 << 5),
   
kSS\_MR\_K1\_5 = (2 << 5),
   
kSS\_MR\_K2 = (3 << 5),
   
kSS\_MR\_K3 = (4 << 5),
   
kSS\_MR\_K4 = (5 << 5),
   
kSS\_MR\_K6 = (6 << 5),
   
kSS\_MR\_K8 = (7 << 5) }

*PLL Spread Spectrum (SS) Programmable frequency modulation depth See (MR) field in the PLL0SSCG1*

- register in the UM.
  - enum `ss_modwvctrl_t` {
   
kSS\_MC\_NOC = (0 << 8),
   
kSS\_MC\_RECC = (2 << 8),
   
kSS\_MC\_MAXC = (3 << 8) }
- PLL Spread Spectrum (SS) Modulation waveform control See (MC) field in the PLL0SSCG1 register in the UM.*
- enum `pll_error_t` {
   
kStatus\_PLL\_Success = MAKE\_STATUS(kStatusGroup\_Generic, 0),
   
kStatus\_PLL\_OutputTooLow = MAKE\_STATUS(kStatusGroup\_Generic, 1),
   
kStatus\_PLL\_OutputTooHigh = MAKE\_STATUS(kStatusGroup\_Generic, 2),
   
kStatus\_PLL\_OutputError = MAKE\_STATUS(kStatusGroup\_Generic, 3),
   
kStatus\_PLL\_InputTooLow = MAKE\_STATUS(kStatusGroup\_Generic, 4),
   
kStatus\_PLL\_InputTooHigh = MAKE\_STATUS(kStatusGroup\_Generic, 5),
   
kStatus\_PLL\_OutsideIntLimit = MAKE\_STATUS(kStatusGroup\_Generic, 6),
   
kStatus\_PLL\_CCOTooLow = MAKE\_STATUS(kStatusGroup\_Generic, 7),
   
kStatus\_PLL\_CCOTooHigh = MAKE\_STATUS(kStatusGroup\_Generic, 8) }
- PLL status definitions.*

## Functions

- static void `CLOCK_EnableClock (clock_ip_name_t clk)`
  - Enable the clock for specific IP.*
- static void `CLOCK_DisableClock (clock_ip_name_t clk)`
  - Disable the clock for specific IP.*
- status\_t `CLOCK_SetupFROHFClocking (uint32_t iFreq)`
  - Initialize the Core clock to given frequency (48 or 144 MHz). This function turns on FIRC and select the given frequency as the source of fro\_hf.*
- status\_t `CLOCK_SetupExtClocking (uint32_t iFreq)`
  - Initialize the external osc clock to given frequency.*
- status\_t `CLOCK_SetupOsc32KClocking (uint32_t id)`
  - Initialize the XTAL32/EXTAL32 input clock to given frequency.*
- status\_t `CLOCK_SetupClk16KClocking (uint32_t id)`
  - Initialize the FRO16K input clock to given frequency.*
- status\_t `CLOCK_FROHFTrimConfig (firc_trim_config_t config)`
  - Setup FROHF trim.*
- status\_t `CLOCK_FRO12MTrimConfig (sirc_trim_config_t config)`
  - Setup FRO 12M trim.*
- void `CLOCK_SetSysOscMonitorMode (scg_sosc_monitor_mode_t mode)`
  - Sets the system OSC monitor mode.*
- void `CLOCK_SetRoscMonitorMode (scg_rosc_monitor_mode_t mode)`
  - Sets the ROSC monitor mode.*
- void `CLOCK_SetUpllMonitorMode (scg_upll_monitor_mode_t mode)`
  - Sets the UPLL monitor mode.*
- void `CLOCK_SetPll0MonitorMode (scg_pll0_monitor_mode_t mode)`
  - Sets the PLL0 monitor mode.*
- void `CLOCK_SetPll1MonitorMode (scg_pll1_monitor_mode_t mode)`
  - Sets the PLL1 monitor mode.*
- void `VBAT_SetOscConfig (VBAT_Type *base, const vbat_osc_config_t *config)`

- **Config 32k Crystal Oscillator.**
- void **CLOCK\_AttachClk** (*clock\_attach\_id\_t connection*)
 

*Configure the clock selection muxes.*
- **clock\_attach\_id\_t CLOCK\_GetClockAttachId** (*clock\_attach\_id\_t attachId*)
 

*Get the actual clock attach id. This function uses the offset in input attach id, then it reads the actual source value in the register and combine the offset to obtain an actual attach id.*
- void **CLOCK\_SetClkDiv** (*clock\_div\_name\_t div\_name, uint32\_t divided\_by\_value*)
 

*Setup peripheral clock dividers.*
- uint32\_t **CLOCK\_GetClkDiv** (*clock\_div\_name\_t div\_name*)
 

*Get peripheral clock dividers.*
- void **CLOCK\_HaltClkDiv** (*clock\_div\_name\_t div\_name*)
 

*Halt peripheral clock dividers.*
- void **CLOCK\_SetupClockCtrl** (*uint32\_t mask*)
 

*system clocks enable controls.*
- uint32\_t **CLOCK\_GetFreq** (*clock\_name\_t clockName*)
 

*Return Frequency of selected clock.*
- uint32\_t **CLOCK\_GetCoreSysClkFreq** (void)
 

*Return Frequency of core.*
- uint32\_t **CLOCK\_GetCTimerClkFreq** (*uint32\_t id*)
 

*Return Frequency of CTimer functional Clock.*
- uint32\_t **CLOCK\_GetAdcClkFreq** (*uint32\_t id*)
 

*Return Frequency of Adc Clock.*
- uint32\_t **CLOCK\_GetUsb0ClkFreq** (void)
 

*Return Frequency of Usb Clock.*
- uint32\_t **CLOCK\_GetLPFlexCommClkFreq** (*uint32\_t id*)
 

*Return Frequency of LPFlexComm Clock.*
- uint32\_t **CLOCK\_GetSctClkFreq** (void)
 

*Return Frequency of SCTimer Clock.*
- uint32\_t **CLOCK\_GetTsiClkFreq** (void)
 

*Return Frequency of TSI Clock.*
- uint32\_t **CLOCK\_GetSincFilterClkFreq** (void)
 

*Return Frequency of SINC FILTER Clock.*
- uint32\_t **CLOCK\_GetDacClkFreq** (*uint32\_t id*)
 

*Return Frequency of DAC Clock.*
- uint32\_t **CLOCK\_GetFlexspiClkFreq** (void)
 

*Return Frequency of FlexSPI.*
- uint32\_t **CLOCK\_GetPll0OutFreq** (void)
 

*Return Frequency of PLL.*
- uint32\_t **CLOCK\_GetPll1OutFreq** (void)
 

*Return Frequency of USB PLL.*
- uint32\_t **CLOCK\_GetPllClkDivFreq** (void)
 

*Return Frequency of PLLCLKDIV.*
- uint32\_t **CLOCK\_GetI3cClkFreq** (*uint32\_t id*)
 

*Return Frequency of I3C function Clock.*
- uint32\_t **CLOCK\_GetI3cSTCClkFreq** (*uint32\_t id*)
 

*Return Frequency of I3C function slow TC Clock.*
- uint32\_t **CLOCK\_GetI3cSClkFreq** (*uint32\_t id*)
 

*Return Frequency of I3C function slow Clock.*
- uint32\_t **CLOCK\_GetMicfilClkFreq** (void)
 

*Return Frequency of MICFIL Clock.*
- uint32\_t **CLOCK\_GetUsdhcClkFreq** (void)

- `uint32_t CLOCK_GetFlexioClkFreq (void)`  
*Return Frequency of uSDHC.*
- `uint32_t CLOCK_GetFlexcanClkFreq (uint32_t id)`  
*Return Frequency of FLEXCAN.*
- `uint32_t CLOCK_GetEnetRmiiClkFreq (void)`  
*Return Frequency of Ethernet RMII Clock.*
- `uint32_t CLOCK_GetEnetPtpRefClkFreq (void)`  
*Return Frequency of Ethernet PTP REF Clock.*
- `void CLOCK_SetupEnetTxClk (uint32_t iFreq)`  
*Initialize the ENET TX CLK to given frequency.*
- `uint32_t CLOCK_GetEnetTxClkFreq (void)`  
*Return Frequency of ENET TX CLK.*
- `uint32_t CLOCK_GetEwm0ClkFreq (void)`  
*Return Frequency of EWM0 Clock.*
- `uint32_t CLOCK_GetWdtClkFreq (uint32_t id)`  
*Return Frequency of Watchdog.*
- `uint32_t CLOCK_GetOstimerClkFreq (void)`  
*Return Frequency of OSTIMER.*
- `uint32_t CLOCK_GetCmpFClkFreq (uint32_t id)`  
*Return Frequency of CMP Function Clock.*
- `uint32_t CLOCK_GetCmpRRClkFreq (uint32_t id)`  
*Return Frequency of CMP Round Robin Clock.*
- `uint32_t CLOCK_GetSaiClkFreq (uint32_t id)`  
*Return Frequency of SAI Clock.*
- `void CLOCK_SetupSaiMclk (uint32_t id, uint32_t iFreq)`  
*Initialize the SAI MCLK to given frequency.*
- `void CLOCK_SetupSaiTxBclk (uint32_t id, uint32_t iFreq)`  
*Initialize the SAI TX BCLK to given frequency.*
- `void CLOCK_SetupSaiRxBclk (uint32_t id, uint32_t iFreq)`  
*Initialize the SAI RX BCLK to given frequency.*
- `uint32_t CLOCK_GetSaiMclkFreq (uint32_t id)`  
*Return Frequency of SAI MCLK.*
- `uint32_t CLOCK_GetSaiTxBclkFreq (uint32_t id)`  
*Return Frequency of SAI TX BCLK.*
- `uint32_t CLOCK_GetSaiRxBclkFreq (uint32_t id)`  
*Return Frequency of SAI RX BCLK.*
- `uint32_t CLOCK_GetEmvsimClkFreq (uint32_t id)`  
*Return Frequency of EMVSIM Clock.*
- `uint32_t CLOCK_GetPLL0InClockRate (void)`  
*Return PLL0 input clock rate.*
- `uint32_t CLOCK_GetPLL1InClockRate (void)`  
*Return PLL1 input clock rate.*
- `uint32_t CLOCK_GetExtUpllFreq (void)`  
*Gets the external UPLL frequency.*
- `void CLOCK_SetExtUpllFreq (uint32_t freq)`  
*Sets the external UPLL frequency.*
- `__STATIC_INLINE bool CLOCK_IsPLL0Locked (void)`  
*Check if PLL is locked or not.*
- `__STATIC_INLINE bool CLOCK_IsPLL1Locked (void)`  
*Check if PLL1 is locked or not.*

- `uint32_t CLOCK_GetPLLOutFromSetup (pll_setup_t *pSetup)`  
*Return PLL0 output clock rate from setup structure.*
- `pll_error_t CLOCK_SetupPLLData (pll_config_t *pControl, pll_setup_t *pSetup)`  
*Set PLL output based on the passed PLL setup data.*
- `pll_error_t CLOCK_SetPLL0Freq (const pll_setup_t *pSetup)`  
*Set PLL output from PLL setup structure (precise frequency)*
- `pll_error_t CLOCK_SetPLL1Freq (const pll_setup_t *pSetup)`  
*Set PLL output from PLL setup structure (precise frequency)*
- `void CLOCK_EnableOstimer32kClock (void)`  
*Enable the OSTIMER 32k clock.*
- `bool CLOCK_EnableUsbfsClock (void)`  
*brief Enable USB FS clock.*
- `bool CLOCK_EnableUsbhsPhyPllClock (clock_usb_phy_src_t src, uint32_t freq)`  
*brief Enable USB HS PHY PLL clock.*
- `void CLOCK_DisableUsbhsPhyPllClock (void)`  
*brief Disable USB HS PHY PLL clock.*
- `bool CLOCK_EnableUsbhsClock (void)`  
*brief Enable USB HS clock.*

## Driver version

- `#define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))`  
*CLOCK driver version 1.0.0.*

## 4.2 Data Structure Documentation

### 4.2.1 struct firc\_trim\_config\_t

#### Data Fields

- `firc_trim_mode_t trimMode`  
*Trim mode.*
- `firc_trim_src_t trimSrc`  
*Trim source.*
- `uint16_t trimDiv`  
*Divider of SOSC.*
- `uint8_t trimCoar`  
*Trim coarse value; Irrelevant if trimMode is kSCG\_TrimUpdate.*
- `uint8_t trimFine`  
*Trim fine value; Irrelevant if trimMode is kSCG\_TrimUpdate.*

#### Field Documentation

- (1) `firc_trim_mode_t firc_trim_config_t::trimMode`
- (2) `firc_trim_src_t firc_trim_config_t::trimSrc`
- (3) `uint16_t firc_trim_config_t::trimDiv`
- (4) `uint8_t firc_trim_config_t::trimCoar`

(5) `uint8_t firc_trim_config_t::trimFine`

#### 4.2.2 struct sirc\_trim\_config\_t

##### Data Fields

- `sirc_trim_mode_t trimMode`  
*Trim mode.*
- `sirc_trim_src_t trimSrc`  
*Trim source.*
- `uint16_t trimDiv`  
*Divider of SOSC.*
- `uint8_t cltrim`  
*Trim coarse value; Irrelevant if trimMode is kSCG\_TrimUpdate.*
- `uint8_t ccotrim`  
*Trim fine value; Irrelevant if trimMode is kSCG\_TrimUpdate.*

##### Field Documentation

(1) `sirc_trim_mode_t sirc_trim_config_t::trimMode`

(2) `sirc_trim_src_t sirc_trim_config_t::trimSrc`

(3) `uint16_t sirc_trim_config_t::trimDiv`

(4) `uint8_t sirc_trim_config_t::cltrim`

(5) `uint8_t sirc_trim_config_t::ccotrim`

#### 4.2.3 struct vbat\_osc\_config\_t

##### Data Fields

- `bool enableInternalCapBank`  
*enable/disable the internal capacitance bank.*
- `bool enableCrystalOscillatorBypass`  
*enable/disable the crystal oscillator bypass.*
- `vbat_osc_xtal_cap_t xtalCap`  
*The internal capacitance for the OSC XTAL pin from the capacitor bank, only useful when the internal capacitance bank is enabled.*
- `vbat_osc_extal_cap_t extalCap`  
*The internal capacitance for the OSC EXTAL pin from the capacitor bank, only useful when the internal capacitance bank is enabled.*
- `vbat_osc_coarse_adjustment_value_t coarseAdjustment`  
*32kHz crystal oscillator amplifier coarse adjustment value.*

##### Field Documentation

- (1) **bool vbat\_osc\_config\_t::enableInternalCapBank**
- (2) **bool vbat\_osc\_config\_t::enableCrystalOscillatorBypass**
- (3) **vbat\_osc\_xtal\_cap\_t vbat\_osc\_config\_t::xtalCap**
- (4) **vbat\_osc\_extal\_cap\_t vbat\_osc\_config\_t::extalCap**
- (5) **vbat\_osc\_coarse\_adjustment\_value\_t vbat\_osc\_config\_t::coarseAdjustment**

#### 4.2.4 struct pll\_config\_t

This structure can be used to configure the settings for a PLL setup structure. Fill in the desired configuration for the PLL and call the PLL setup function to fill in a PLL setup structure.

#### Data Fields

- **uint32\_t desiredRate**  
*Desired PLL rate in Hz.*
- **uint32\_t inputSource**  
*PLL input source.*
- **uint32\_t flags**  
*PLL configuration flags, Or'ed value of PLL\_CONFIGFLAG\_\* definitions.*
- **ss\_progmodfm\_t ss\_mf**  
*SS Programmable modulation frequency, only applicable when not using PLL\_CONFIGFLAG\_FORCE\_NOFRACT flag.*
- **ss\_progmddp\_t ss\_mr**  
*SS Programmable frequency modulation depth, only applicable when not using PLL\_CONFIGFLAG\_FORCEORCENOFRACT flag.*
- **ss\_modwvctrl\_t ss\_mc**  
*SS Modulation waveform control, only applicable when not using PLL\_CONFIGFLAG\_FORCENOFRACCT flag.*
- **bool mfDither**  
*false for fixed modulation frequency or true for dithering, only applicable when not using PLL\_CONFIGFLAG\_FORCENOFRACT flag*

#### 4.2.5 struct pll\_setup\_t

It can be populated with the PLL setup function. If powering up or waiting for PLL lock, the PLL input clock source should be configured prior to PLL setup.

#### Data Fields

- **uint32\_t pllctrl**  
*PLL Control register APLLCTRL.*

- `uint32_t pllndiv`  
*PLL N Divider register APOLLNDIV.*
- `uint32_t pllpdiv`  
*PLL P Divider register APOLLPDIV.*
- `uint32_t pllmdiv`  
*PLL M Divider register APOLLMDIV.*
- `uint32_t pllsscgr [2]`  
*PLL Spread Spectrum Control registers APOLLSSCG.*
- `uint32_t pllRate`  
*Acutal PLL rate.*

## 4.3 Macro Definition Documentation

### 4.3.1 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(1, 0, 0))

### 4.3.2 #define FSL\_SDK\_DISABLE\_DRIVER\_CLOCK\_CONTROL 0

When set to 0, peripheral drivers will enable clock in initialize function and disable clock in de-initialize function. When set to 1, peripheral driver will not control the clock, application could control the clock out of the driver.

Note

All drivers share this feature switcher. If it is set to 1, application should handle clock enable and disable for all drivers.

### 4.3.3 #define CLOCK\_USR\_CFG\_PLL\_CONFIG\_CACHE\_COUNT 2U

Once define this MACRO to be non-zero value, CLOCK\_PllGetConfig() function would cache the recent calculation and accelerate the execution to get the right settings.

### 4.3.4 #define ROM\_CLOCKS

**Value:**

```
{
    kCLOCK_Rom \
}
```

### 4.3.5 #define SRAM\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Sram1, kCLOCK_Sram2, kCLOCK_Sram3,
    kCLOCK_Sram4, kCLOCK_Sram5, kCLOCK_Sram6,
    kCLOCK_Sram7 \
}
```

#### 4.3.6 #define FMC\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Fmc \
}
```

#### 4.3.7 #define INPUTMUX\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_InputMux0 \
}
```

#### 4.3.8 #define ETH\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Enet \
}
```

#### 4.3.9 #define GPIO\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Gpio0, kCLOCK_Gpio1, kCLOCK_Gpio2,
    kCLOCK_Gpio3, kCLOCK_Gpio4 \
}
```

#### 4.3.10 #define PINT\_CLOCKS

**Value:**

```
{  
    kCLOCK_Pint \  
}
```

#### 4.3.11 #define DMA\_CLOCKS

**Value:**

```
{  
    kCLOCK_Dma0, kCLOCK_Dma1 \  
}
```

#### 4.3.12 #define EDMA\_CLOCKS

**Value:**

```
{  
    kCLOCK_Dma0, kCLOCK_Dma1 \  
}
```

#### 4.3.13 #define CRC\_CLOCKS

**Value:**

```
{  
    kCLOCK_Crc0 \  
}
```

#### 4.3.14 #define WWDT\_CLOCKS

**Value:**

```
{  
    kCLOCK_Wwdt0, kCLOCK_Wwdt1 \  
}
```

#### 4.3.15 #define MAILBOX\_CLOCKS

**Value:**

```
{  
    kCLOCK_Mailbox \  
}
```

#### 4.3.16 #define LPADC\_CLOCKS

**Value:**

```
{  
    kCLOCK_Adc0, kCLOCK_Adc1 \  
}
```

#### 4.3.17 #define MRT\_CLOCKS

**Value:**

```
{  
    kCLOCK_Mrt \  
}
```

#### 4.3.18 #define OSTIMER\_CLOCKS

**Value:**

```
{  
    kCLOCK_OsTimer \  
}
```

#### 4.3.19 #define SCT\_CLOCKS

**Value:**

```
{  
    kCLOCK_Sct \  
}
```

### 4.3.20 #define UTICK\_CLOCKS

**Value:**

```
{
    \KCLOCK_Utick \
}
```

### 4.3.21 #define LP\_FLEXCOMM\_CLOCKS

**Value:**

```
{
    \
    \KCLOCK_LPFlexComm0, \KCLOCK_LPFlexComm1,
    \KCLOCK_LPFlexComm2, \KCLOCK_LPFlexComm3,
    \KCLOCK_LPFlexComm4, \
    \KCLOCK_LPFlexComm5, \KCLOCK_LPFlexComm6,
    \KCLOCK_LPFlexComm7, \KCLOCK_LPFlexComm8,
    \KCLOCK_LPFlexComm9 \
}
```

### 4.3.22 #define LPUART\_CLOCKS

**Value:**

```
{
    \
    \KCLOCK_LPUart0, \KCLOCK_LPUart1,
    \KCLOCK_LPUart2, \KCLOCK_LPUart3, \KCLOCK_LPUart4,
    \KCLOCK_LPUart5, \
    \KCLOCK_LPUart6, \KCLOCK_LPUart7,
    \KCLOCK_LPUart8, \KCLOCK_LPUart9 \
}
```

### 4.3.23 #define LPI2C\_CLOCKS

**Value:**

```
{
    \
    \KCLOCK_LPI2c0, \KCLOCK_LPI2c1,
    \KCLOCK_LPI2c2, \KCLOCK_LPI2c3, \KCLOCK_LPI2c4,
    \KCLOCK_LPI2c5, \KCLOCK_LPI2c6, \
    \KCLOCK_LPI2c7, \KCLOCK_LPI2c8,
    \KCLOCK_LPI2c9 \
}
```

#### 4.3.24 #define LPSPI\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_LPSpi0, kCLOCK_LPSpi1,
    kCLOCK_LPSpi2, kCLOCK_LPSpi3, kCLOCK_LPSpi4,
    kCLOCK_LPSpi5, kCLOCK_LPSpi6, \
        kCLOCK_LPSpi7, kCLOCK_LPSpi8,
    kCLOCK_LPSpi9
}
```

#### 4.3.25 #define CTIMER\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Timer0, kCLOCK_Timer1,
    kCLOCK_Timer2, kCLOCK_Timer3, kCLOCK_Timer4 \
}
```

#### 4.3.26 #define FREQME\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Freqme \
}
```

#### 4.3.27 #define POWERQUAD\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_PowerQuad \
}
```

#### 4.3.28 #define PLU\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Plut \
}
```

#### 4.3.29 #define PUF\_CLOCKS

**Value:**

```
{
    kCLOCK_Puf \
}
```

#### 4.3.30 #define VREF\_CLOCKS

**Value:**

```
{
    kCLOCK_Vref \
}
```

#### 4.3.31 #define LPDAC\_CLOCKS

**Value:**

```
{
    kCLOCK_Dac0, kCLOCK_Dac1 \
}
```

#### 4.3.32 #define HPDAC\_CLOCKS

**Value:**

```
{
    kCLOCK_Dac2 \
}
```

#### 4.3.33 #define PWM\_CLOCKS

**Value:**

```
{
    {kCLOCK_Pwm0, kCLOCK_Pwm0, kCLOCK_Pwm0, kCLOCK_Pwm0}, \
    {
        kCLOCK_Pwm1, kCLOCK_Pwm1,
        kCLOCK_Pwm1, kCLOCK_Pwm1 \
    }
}
```

#### 4.3.34 #define ENC\_CLOCKS

**Value:**

```
{  
    kCLOCK_Enc0, kCLOCK_Enc1 \  
}
```

#### 4.3.35 #define FLEXIO\_CLOCKS

**Value:**

```
{  
    kCLOCK_Flexio \  
}
```

#### 4.3.36 #define FLEXCAN\_CLOCKS

**Value:**

```
{  
    kCLOCK_Flexcan0, kCLOCK_Flexcan1 \  
}
```

#### 4.3.37 #define EMVSIM\_CLOCKS

**Value:**

```
{  
    kCLOCK_Evsim0, kCLOCK_Evsim1 \  
}
```

#### 4.3.38 #define USDHC\_CLOCKS

**Value:**

```
{  
    kCLOCK_uSdhc \  
}
```

#### 4.3.39 #define SAI\_CLOCKS

**Value:**

```
{  
    kCLOCK_Sai0, kCLOCK_Sai1 \  
}
```

#### 4.3.40 #define RTC\_CLOCKS

**Value:**

```
{  
    kCLOCK_Rtc0 \  
}
```

#### 4.3.41 #define PDM\_CLOCKS

**Value:**

```
{  
    kCLOCK_Micfil \  
}
```

#### 4.3.42 #define ERM\_CLOCKS

**Value:**

```
{  
    kCLOCK_Erm \  
}
```

#### 4.3.43 #define EIM\_CLOCKS

**Value:**

```
{  
    kCLOCK_Eim \  
}
```

**4.3.44 #define OPAMP\_CLOCKS****Value:**

```
{
    kCLOCK_Opamp0, kCLOCK_Opamp1,
    kCLOCK_Opamp2 \
}
```

**4.3.45 #define TSI\_CLOCKS****Value:**

```
{
    kCLOCK_Tsi \
}
```

**4.3.46 #define TRNG\_CLOCKS****Value:**

```
{
    kCLOCK_Trng \
}
```

**4.3.47 #define LPCMP\_CLOCKS****Value:**

```
{
    kCLOCK_None, kCLOCK_Cmp2, kCLOCK_None \
}
```

**4.3.48 #define CLK\_GATE\_REG\_OFFSET\_SHIFT 8U****4.3.49 #define BUS\_CLK kCLOCK\_BusClk****4.3.50 #define CLK\_ATTACH\_ID( mux, sel, pos ) (((uint32\_t)(mux) << 0U) |  
((uint32\_t)(sel) + 1U) & 0xFU) << 12U) << ((uint32\_t)(pos)\*16U))**

[4 bits for choice, 0 means invalid choice] [8 bits mux ID]\*

#### 4.3.51 #define PLL\_CONFIGFLAG\_FORCE\_NOFRACT (1U << 2U)

When the PLL\_CONFIGFLAG\_FORCE\_NOFRACT flag is selected, the PLL hardware for the automatic bandwidth selection, Spread Spectrum (SS) support, and fractional M-divider are not used.

Force non-fractional output mode, PLL output will not use the fractional, automatic bandwidth, or SS hardware

## 4.4 Enumeration Type Documentation

### 4.4.1 enum clock\_ip\_name\_t

Enumerator

- kCLOCK\_IpInvalid* Invalid Ip Name.
- kCLOCK\_None* None clock gate.
- kCLOCK\_Rom* Clock gate name: Rom.
- kCLOCK\_Sram1* Clock gate name: Sram1.
- kCLOCK\_Sram2* Clock gate name: Sram2.
- kCLOCK\_Sram3* Clock gate name: Sram3.
- kCLOCK\_Sram4* Clock gate name: Sram4.
- kCLOCK\_Sram5* Clock gate name: Sram5.
- kCLOCK\_Sram6* Clock gate name: Sram6.
- kCLOCK\_Sram7* Clock gate name: Sram7.
- kCLOCK\_Fmu* Clock gate name: Fmu.
- kCLOCK\_Fmc* Clock gate name: Fmc.
- kCLOCK\_Flexspi* Clock gate name: Flexspi.
- kCLOCK\_InputMux0* Clock gate name: InputMux0.
- kCLOCK\_InputMux* Clock gate name: InputMux0.
- kCLOCK\_Port0* Clock gate name: Port0.
- kCLOCK\_Port1* Clock gate name: Port1.
- kCLOCK\_Port2* Clock gate name: Port2.
- kCLOCK\_Port3* Clock gate name: Port3.
- kCLOCK\_Port4* Clock gate name: Port4.
- kCLOCK\_Gpio0* Clock gate name: Gpio0.
- kCLOCK\_Gpio1* Clock gate name: Gpio1.
- kCLOCK\_Gpio2* Clock gate name: Gpio2.
- kCLOCK\_Gpio3* Clock gate name: Gpio3.
- kCLOCK\_Gpio4* Clock gate name: Gpio4.
- kCLOCK\_Pint* Clock gate name: Pint.
- kCLOCK\_Dma0* Clock gate name: Dma0.
- kCLOCK\_Crc0* Clock gate name: Crc.
- kCLOCK\_Wwdt0* Clock gate name: Wwdt0.
- kCLOCK\_Wwdt1* Clock gate name: Wwdt1.
- kCLOCK\_Mailbox* Clock gate name: Mailbox.
- kCLOCK\_Mrt* Clock gate name: Mrt.

***kCLOCK\_OsTimer*** Clock gate name: OsTimer.  
***kCLOCK\_Sct*** Clock gate name: Sct.  
***kCLOCK\_Adc0*** Clock gate name: Adc0.  
***kCLOCK\_Adc1*** Clock gate name: Adc1.  
***kCLOCK\_Dac0*** Clock gate name: Dac0.  
***kCLOCK\_Rtc0*** Clock gate name: Rtc.  
***kCLOCK\_Evsim0*** Clock gate name: Evsim0.  
***kCLOCK\_Evsim1*** Clock gate name: Evsim1.  
***kCLOCK\_Utick*** Clock gate name: Utick.  
***kCLOCK\_LPFlexComm0*** Clock gate name: LPFlexComm0.  
***kCLOCK\_LPFlexComm1*** Clock gate name: LPFlexComm1.  
***kCLOCK\_LPFlexComm2*** Clock gate name: LPFlexComm2.  
***kCLOCK\_LPFlexComm3*** Clock gate name: LPFlexComm3.  
***kCLOCK\_LPFlexComm4*** Clock gate name: LPFlexComm4.  
***kCLOCK\_LPFlexComm5*** Clock gate name: LPFlexComm5.  
***kCLOCK\_LPFlexComm6*** Clock gate name: LPFlexComm6.  
***kCLOCK\_LPFlexComm7*** Clock gate name: LPFlexComm7.  
***kCLOCK\_LPFlexComm8*** Clock gate name: LPFlexComm8.  
***kCLOCK\_LPFlexComm9*** Clock gate name: LPFlexComm9.  
***kCLOCK\_LPUart0*** Clock gate name: LPUart0.  
***kCLOCK\_LPUart1*** Clock gate name: LPUart1.  
***kCLOCK\_LPUart2*** Clock gate name: LPUart2.  
***kCLOCK\_LPUart3*** Clock gate name: LPUart3.  
***kCLOCK\_LPUart4*** Clock gate name: LPUart4.  
***kCLOCK\_LPUart5*** Clock gate name: LPUart5.  
***kCLOCK\_LPUart6*** Clock gate name: LPUart6.  
***kCLOCK\_LPUart7*** Clock gate name: LPUart7.  
***kCLOCK\_LPUart8*** Clock gate name: LPUart8.  
***kCLOCK\_LPUart9*** Clock gate name: LPUart9.  
***kCLOCK\_LP Spi0*** Clock gate name: LP Spi0.  
***kCLOCK\_LP Spi1*** Clock gate name: LP Spi1.  
***kCLOCK\_LP Spi2*** Clock gate name: LP Spi2.  
***kCLOCK\_LP Spi3*** Clock gate name: LP Spi3.  
***kCLOCK\_LP Spi4*** Clock gate name: LP Spi4.  
***kCLOCK\_LP Spi5*** Clock gate name: LP Spi5.  
***kCLOCK\_LP Spi6*** Clock gate name: LP Spi6.  
***kCLOCK\_LP Spi7*** Clock gate name: LP Spi7.  
***kCLOCK\_LP Spi8*** Clock gate name: LP Spi8.  
***kCLOCK\_LP Spi9*** Clock gate name: LP Spi9.  
***kCLOCK\_LPI2c0*** Clock gate name: LPI2c0.  
***kCLOCK\_LPI2c1*** Clock gate name: LPI2c1.  
***kCLOCK\_LPI2c2*** Clock gate name: LPI2c2.  
***kCLOCK\_LPI2c3*** Clock gate name: LPI2c3.  
***kCLOCK\_LPI2c4*** Clock gate name: LPI2c4.  
***kCLOCK\_LPI2c5*** Clock gate name: LPI2c5.

***kCLOCK\_LPI2c6*** Clock gate name: LPI2c6.  
***kCLOCK\_LPI2c7*** Clock gate name: LPI2c7.  
***kCLOCK\_LPI2c8*** Clock gate name: LPI2c8.  
***kCLOCK\_LPI2c9*** Clock gate name: LPI2c9.  
***kCLOCK\_Micfil*** Clock gate name: Micfil.  
***kCLOCK\_Timer2*** Clock gate name: Timer2.  
***kCLOCK\_Usb0Ram*** Clock gate name: Usb0Ram.  
***kCLOCK\_Usb0FsDcd*** Clock gate name: Usb0FsDcd.  
***kCLOCK\_Usb0Fs*** Clock gate name: Usb0Fs.  
***kCLOCK\_Timer0*** Clock gate name: Timer0.  
***kCLOCK\_Timer1*** Clock gate name: Timer1.  
***kCLOCK\_PkcRam*** Clock gate name: PkcRam.  
***kCLOCK\_SmartDma*** Clock gate name: SmartDma.  
***kCLOCK\_Espi*** Clock gate name: Espi.  
***kCLOCK\_Dma1*** Clock gate name: Dma1.  
***kCLOCK\_Enet*** Clock gate name: Enet.  
***kCLOCK\_uSdhc*** Clock gate name: uSdhc.  
***kCLOCK\_Flexio*** Clock gate name: Flexio.  
***kCLOCK\_Sai0*** Clock gate name: Sai0.  
***kCLOCK\_Sai1*** Clock gate name: Sai1.  
***kCLOCK\_Tro*** Clock gate name: Tro.  
***kCLOCK\_Freqme*** Clock gate name: Freqme.  
***kCLOCK\_Trng*** Clock gate name: Trng.  
***kCLOCK\_Flexcan0*** Clock gate name: Flexcan0.  
***kCLOCK\_Flexcan1*** Clock gate name: Flexcan1.  
***kCLOCK\_UsbHs*** Clock gate name: UsbHs.  
***kCLOCK\_UsbHsPhy*** Clock gate name: UsbHsPhy.  
***kCLOCK\_Css*** Clock gate name: Css.  
***kCLOCK\_PowerQuad*** Clock gate name: PowerQuad.  
***kCLOCK\_Plut*** Clock gate name: Plut.  
***kCLOCK\_Timer3*** Clock gate name: Timer3.  
***kCLOCK\_Timer4*** Clock gate name: Timer4.  
***kCLOCK\_Puf*** Clock gate name: Puf.  
***kCLOCK\_Pkc*** Clock gate name: Pkc.  
***kCLOCK\_Scg*** Clock gate name: Scg.  
***kCLOCK\_Gdet*** Clock gate name: Gdet.  
***kCLOCK\_Sm3*** Clock gate name: Sm3.  
***kCLOCK\_I3c0*** Clock gate name: I3c0.  
***kCLOCK\_I3c1*** Clock gate name: I3c1.  
***kCLOCK\_Sinc*** Clock gate name: Sinc.  
***kCLOCK\_CoolFlux*** Clock gate name: CoolFlux.  
***kCLOCK\_Enc0*** Clock gate name: Enc0.  
***kCLOCK\_Enc1*** Clock gate name: Enc1.  
***kCLOCK\_Pwm0*** Clock gate name: Pwm0.  
***kCLOCK\_Pwm1*** Clock gate name: Pwm1.

*kCLOCK\_Evtg* Clock gate name: Evtg.  
*kCLOCK\_Dac1* Clock gate name: Dac1.  
*kCLOCK\_Dac2* Clock gate name: Dac2.  
*kCLOCK\_Opamp0* Clock gate name: Opamp0.  
*kCLOCK\_Opamp1* Clock gate name: Opamp1.  
*kCLOCK\_Opamp2* Clock gate name: Opamp2.  
*kCLOCK\_Cmp2* Clock gate name: Cmp2.  
*kCLOCK\_Vref* Clock gate name: Vref.  
*kCLOCK\_CoolFluxApb* Clock gate name: CoolFluxApb.  
*kCLOCK\_Neutron* Clock gate name: Neutron.  
*kCLOCK\_Tsi* Clock gate name: Tsi.  
*kCLOCK\_Ewm* Clock gate name: Ewm.  
*kCLOCK\_Ewm0* Clock gate name: Ewm.  
*kCLOCK\_Eim* Clock gate name: Eim.  
*kCLOCK\_Erm* Clock gate name: Erm.  
*kCLOCK\_Intm* Clock gate name: Intm.  
*kCLOCK\_Sema42* Clock gate name: Sema42.

#### 4.4.2 enum clock\_name\_t

Enumerator

*kCLOCK\_CoreSysClk* Core/system clock (aka MAIN\_CLK)  
*kCLOCK\_BusClk* Bus clock (AHB clock)  
*kCLOCK\_SystickClk0* Systick clock0.  
*kCLOCK\_SystickClk1* Systick clock1.  
*kCLOCK\_ClockOut* CLOCKOUT.  
*kCLOCK\_Fro12M* FRO12M.  
*kCLOCK\_Clk1M* CLK1M.  
*kCLOCK\_FroHf* FRO48/144.  
*kCLOCK\_Clk48M* CLK48M.  
*kCLOCK\_Clk144M* CLK144M.  
*kCLOCK\_Clk16K0* CLK16K[0].  
*kCLOCK\_Clk16K1* CLK16K[1].  
*kCLOCK\_Clk16K2* CLK16K[2].  
*kCLOCK\_Clk16K3* CLK16K[3].  
*kCLOCK\_ExtClk* External Clock.  
*kCLOCK\_Osc32K0* OSC32K[0].  
*kCLOCK\_Osc32K1* OSC32K[1].  
*kCLOCK\_Osc32K2* OSC32K[2].  
*kCLOCK\_Osc32K3* OSC32K[3].  
*kCLOCK\_Pll0Out* PLL0 Output.  
*kCLOCK\_Pll1Out* PLL1 Output.  
*kCLOCK\_UsbPllOut* USB PLL Output.

**kCLOCK\_LpOsc** lp\_osc

#### 4.4.3 enum clock\_attach\_id\_t

Enumerator

*kCLK\_IN\_to\_MAIN\_CLK* Attach clk\_in to MAIN\_CLK.  
*kFRO12M\_to\_MAIN\_CLK* Attach FRO\_12M to MAIN\_CLK.  
*kFRO\_HF\_to\_MAIN\_CLK* Attach FRO\_HF to MAIN\_CLK.  
*kXTAL32K2\_to\_MAIN\_CLK* Attach xtal32k[2] to MAIN\_CLK.  
*kPLL0\_to\_MAIN\_CLK* Attach PLL0 to MAIN\_CLK.  
*kPLL1\_to\_MAIN\_CLK* Attach PLL1 to MAIN\_CLK.  
*kUSB\_PLL\_to\_MAIN\_CLK* Attach USB PLL to MAIN\_CLK.  
*kNONE\_to\_MAIN\_CLK* Attach NONE to MAIN\_CLK.  
*kSYSTICK\_DIV0\_to\_SYSTICK0* Attach SYSTICK\_DIV0 to SYSTICK0.  
*kCLK\_1M\_to\_SYSTICK0* Attach Clk 1 MHz to SYSTICK0.  
*kLPOSC\_to\_SYSTICK0* Attach LP Oscillator to SYSTICK0.  
*kNONE\_to\_SYSTICK0* Attach NONE to SYSTICK0.  
*kSYSTICK\_DIV1\_to\_SYSTICK1* Attach SYSTICK\_DIV1 to SYSTICK1.  
*kCLK\_1M\_to\_SYSTICK1* Attach Clk 1 MHz to SYSTICK1.  
*kLPOSC\_to\_SYSTICK1* Attach LP Oscillator to SYSTICK1.  
*kNONE\_to\_SYSTICK1* Attach NONE to SYSTICK1.  
*kTRACE\_DIV\_to\_TRACE* Attach TRACE\_DIV to TRACE.  
*kCLK\_1M\_to\_TRACE* Attach Clk 1 MHz to TRACE.  
*kLPOSC\_to\_TRACE* Attach LP Oscillator to TRACE.  
*kNONE\_to\_TRACE* Attach NONE to TRACE.  
*kCLK\_1M\_to\_CTIMER0* Attach CLK\_1M to CTIMER0.  
*kPLL0\_to\_CTIMER0* Attach PLL0 to CTIMER0.  
*kPLL1\_CLK0\_to\_CTIMER0* Attach PLL1\_clk0 to CTIMER0.  
*kFRO\_HF\_to\_CTIMER0* Attach FRO\_HF to CTIMER0.  
*kFRO12M\_to\_CTIMER0* Attach FRO 12MHz to CTIMER0.  
*kSAI0\_MCLK\_IN\_to\_CTIMER0* Attach SAI0 MCLK IN to CTIMER0.  
*kLPOSC\_to\_CTIMER0* Attach LP Oscillator to CTIMER0.  
*kSAI1\_MCLK\_IN\_to\_CTIMER0* Attach SAI1 MCLK IN to CTIMER0.  
*kSAI0\_TX\_BCLK\_to\_CTIMER0* Attach SAI0 TX\_BCLK to CTIMER0.  
*kSAI0\_RX\_BCLK\_to\_CTIMER0* Attach SAI0 RX\_BCLK to CTIMER0.  
*kSAI1\_TX\_BCLK\_to\_CTIMER0* Attach SAI1 TX\_BCLK to CTIMER0.  
*kSAI1\_RX\_BCLK\_to\_CTIMER0* Attach SAI1 RX\_BCLK to CTIMER0.  
*kNONE\_to\_CTIMER0* Attach NONE to CTIMER0.  
*kCLK\_1M\_to\_CTIMER1* Attach CLK\_1M to CTIMER1.  
*kPLL0\_to\_CTIMER1* Attach PLL0 to CTIMER1.  
*kPLL1\_CLK0\_to\_CTIMER1* Attach PLL1\_clk0 to CTIMER1.  
*kFRO\_HF\_to\_CTIMER1* Attach FRO\_HF to CTIMER1.  
*kFRO12M\_to\_CTIMER1* Attach FRO 12MHz to CTIMER1.

***kSAI0\_MCLK\_IN\_to\_CTIMER1*** Attach SAI0 MCLK IN to CTIMER1.

***kLPOSC\_to\_CTIMER1*** Attach LP Oscillator to CTIMER1.

***kSAI1\_MCLK\_IN\_to\_CTIMER1*** Attach SAI1 MCLK IN to CTIMER1.

***kSAI0\_TX\_BCLK\_to\_CTIMER1*** Attach SAI0 TX\_BCLK to CTIMER1.

***kSAI0\_RX\_BCLK\_to\_CTIMER1*** Attach SAI0 RX\_BCLK to CTIMER1.

***kSAI1\_TX\_BCLK\_to\_CTIMER1*** Attach SAI1 TX\_BCLK to CTIMER1.

***kSAI1\_RX\_BCLK\_to\_CTIMER1*** Attach SAI1 RX\_BCLK to CTIMER1.

***kNONE\_to\_CTIMER1*** Attach NONE to CTIMER1.

***kCLK\_1M\_to\_CTIMER2*** Attach CLK\_1M to CTIMER2.

***kPLL0\_to\_CTIMER2*** Attach PLL0 to CTIMER2.

***kPLL1\_CLK0\_to\_CTIMER2*** Attach PLL1\_clk0 to CTIMER2.

***kFRO\_HF\_to\_CTIMER2*** Attach FRO\_HF to CTIMER2.

***kFRO12M\_to\_CTIMER2*** Attach FRO 12MHz to CTIMER2.

***kSAI0\_MCLK\_IN\_to\_CTIMER2*** Attach SAI0 MCLK IN to CTIMER2.

***kLPOSC\_to\_CTIMER2*** Attach LP Oscillator to CTIMER2.

***kSAI1\_MCLK\_IN\_to\_CTIMER2*** Attach SAI1 MCLK IN to CTIMER2.

***kSAI0\_TX\_BCLK\_to\_CTIMER2*** Attach SAI0 TX\_BCLK to CTIMER2.

***kSAI0\_RX\_BCLK\_to\_CTIMER2*** Attach SAI0 RX\_BCLK to CTIMER2.

***kSAI1\_TX\_BCLK\_to\_CTIMER2*** Attach SAI1 TX\_BCLK to CTIMER2.

***kSAI1\_RX\_BCLK\_to\_CTIMER2*** Attach SAI1 RX\_BCLK to CTIMER2.

***kNONE\_to\_CTIMER2*** Attach NONE to CTIMER2.

***kCLK\_1M\_to\_CTIMER3*** Attach CLK\_1M to CTIMER3.

***kPLL0\_to\_CTIMER3*** Attach PLL0 to CTIMER3.

***kPLL1\_CLK0\_to\_CTIMER3*** Attach PLL1\_clk0 to CTIMER3.

***kFRO\_HF\_to\_CTIMER3*** Attach FRO\_HF to CTIMER3.

***kFRO12M\_to\_CTIMER3*** Attach FRO 12MHz to CTIMER3.

***kSAI0\_MCLK\_IN\_to\_CTIMER3*** Attach SAI0 MCLK IN to CTIMER3.

***kLPOSC\_to\_CTIMER3*** Attach LP Oscillator to CTIMER3.

***kSAI1\_MCLK\_IN\_to\_CTIMER3*** Attach SAI1 MCLK IN to CTIMER3.

***kSAI0\_TX\_BCLK\_to\_CTIMER3*** Attach SAI0 TX\_BCLK to CTIMER3.

***kSAI0\_RX\_BCLK\_to\_CTIMER3*** Attach SAI0 RX\_BCLK to CTIMER3.

***kSAI1\_TX\_BCLK\_to\_CTIMER3*** Attach SAI1 TX\_BCLK to CTIMER3.

***kSAI1\_RX\_BCLK\_to\_CTIMER3*** Attach SAI1 RX\_BCLK to CTIMER3.

***kNONE\_to\_CTIMER3*** Attach NONE to CTIMER3.

***kCLK\_1M\_to\_CTIMER4*** Attach CLK\_1M to CTIMER4.

***kPLL0\_to\_CTIMER4*** Attach PLL0 to CTIMER4.

***kPLL1\_CLK0\_to\_CTIMER4*** Attach PLL1\_clk0 to CTIMER4.

***kFRO\_HF\_to\_CTIMER4*** Attach FRO\_HF to CTIMER4.

***kFRO12M\_to\_CTIMER4*** Attach FRO 12MHz to CTIMER4.

***kSAI0\_MCLK\_IN\_to\_CTIMER4*** Attach SAI0 MCLK IN to CTIMER4.

***kLPOSC\_to\_CTIMER4*** Attach LP Oscillator to CTIMER4.

***kSAI1\_MCLK\_IN\_to\_CTIMER4*** Attach SAI1 MCLK IN to CTIMER4.

***kSAI0\_TX\_BCLK\_to\_CTIMER4*** Attach SAI0 TX\_BCLK to CTIMER4.

***kSAI0\_RX\_BCLK\_to\_CTIMER4*** Attach SAI0 RX\_BCLK to CTIMER4.

***kSAI1\_TX\_BCLK\_to\_CTIMER4*** Attach SAI1 TX\_BCLK to CTIMER4.

***kSAI1\_RX\_BCLK\_to\_TIMER4*** Attach SAI1 RX\_BCLK to TIMER4.

***kNONE\_to\_TIMER4*** Attach NONE to TIMER4.

***kMAIN\_CLK\_to\_CLKOUT*** Attach MAIN\_CLK to CLKOUT.

***kPLL0\_to\_CLKOUT*** Attach PLL0 to CLKOUT.

***kCLK\_IN\_to\_CLKOUT*** Attach Clk\_in to CLKOUT.

***kFRO\_HF\_to\_CLKOUT*** Attach FRO\_HF to CLKOUT.

***kFRO12M\_to\_CLKOUT*** Attach FRO 12 MHz to CLKOUT.

***kPLL1\_CLK0\_to\_CLKOUT*** Attach PLL1\_clk0 to CLKOUT.

***kLPOSC\_to\_CLKOUT*** Attach LP Oscillator to CLKOUT.

***kUSB\_PLL\_to\_CLKOUT*** Attach USB\_PLL to CLKOUT.

***kNONE\_to\_CLKOUT*** Attach NONE to CLKOUT.

***kPLL0\_to\_ADC0*** Attach PLL0 to ADC0.

***kFRO\_HF\_to\_ADC0*** Attach FRO\_HF to ADC0.

***kFRO12M\_to\_ADC0*** Attach FRO 12 MHz to ADC0.

***kCLK\_IN\_to\_ADC0*** Attach Clk\_in to ADC0.

***kPLL1\_CLK0\_to\_ADC0*** Attach PLL1\_clk0 to ADC0.

***kUSB\_PLL\_to\_ADC0*** Attach USB\_PLL to ADC0.

***kNONE\_to\_ADC0*** Attach NONE to ADC0.

***kPLL0\_to\_USB0*** Attach PLL0 to USB0.

***kCLK\_48M\_to\_USB0*** Attach Clk 48 MHz to USB0.

***kCLK\_IN\_to\_USB0*** Attach Clk\_in to USB0.

***kPLL1\_CLK0\_to\_USB0*** Attach PLL1\_clk0 to USB0.

***kUSB\_PLL\_to\_USB0*** Attach USB\_PLL to USB0.

***kNONE\_to\_USB0*** Attach NONE to USB0.

***kPLL\_DIV\_to\_FLEXCOMM0*** Attach PLL\_DIV to FLEXCOMM0.

***kFRO12M\_to\_FLEXCOMM0*** Attach FRO12M to FLEXCOMM0.

***kFRO\_HF\_DIV\_to\_FLEXCOMM0*** Attach FRO\_HF\_DIV to FLEXCOMM0.

***kCLK\_1M\_to\_FLEXCOMM0*** Attach CLK\_1MHz to FLEXCOMM0.

***kUSB\_PLL\_to\_FLEXCOMM0*** Attach USB\_PLL to FLEXCOMM0.

***kLPOSC\_to\_FLEXCOMM0*** Attach LP Oscillator to FLEXCOMM0.

***kNONE\_to\_FLEXCOMM0*** Attach NONE to FLEXCOMM0.

***kPLL\_DIV\_to\_FLEXCOMM1*** Attach PLL\_DIV to FLEXCOMM1.

***kFRO12M\_to\_FLEXCOMM1*** Attach FRO12M to FLEXCOMM1.

***kFRO\_HF\_DIV\_to\_FLEXCOMM1*** Attach FRO\_HF\_DIV to FLEXCOMM1.

***kCLK\_1M\_to\_FLEXCOMM1*** Attach CLK\_1MHz to FLEXCOMM1.

***kUSB\_PLL\_to\_FLEXCOMM1*** Attach USB\_PLL to FLEXCOMM1.

***kLPOSC\_to\_FLEXCOMM1*** Attach LP Oscillator to FLEXCOMM1.

***kNONE\_to\_FLEXCOMM1*** Attach NONE to FLEXCOMM1.

***kPLL\_DIV\_to\_FLEXCOMM2*** Attach PLL\_DIV to FLEXCOMM2.

***kFRO12M\_to\_FLEXCOMM2*** Attach FRO12M to FLEXCOMM2.

***kFRO\_HF\_DIV\_to\_FLEXCOMM2*** Attach FRO\_HF\_DIV to FLEXCOMM2.

***kCLK\_1M\_to\_FLEXCOMM2*** Attach CLK\_1MHz to FLEXCOMM2.

***kUSB\_PLL\_to\_FLEXCOMM2*** Attach USB\_PLL to FLEXCOMM2.

***kLPOSC\_to\_FLEXCOMM2*** Attach LP Oscillator to FLEXCOMM2.

***kNONE\_to\_FLEXCOMM2*** Attach NONE to FLEXCOMM2.

***kPLL\_DIV\_to\_FLEXCOMM3*** Attach PLL\_DIV to FLEXCOMM3.

***kFRO12M\_to\_FLEXCOMM3*** Attach FRO12M to FLEXCOMM3.

***kFRO\_HF\_DIV\_to\_FLEXCOMM3*** Attach FRO\_HF\_DIV to FLEXCOMM3.

***kCLK\_1M\_to\_FLEXCOMM3*** Attach CLK\_1MHz to FLEXCOMM3.

***kUSB\_PLL\_to\_FLEXCOMM3*** Attach USB\_PLL to FLEXCOMM3.

***kLPOSC\_to\_FLEXCOMM3*** Attach LP Oscillator to FLEXCOMM3.

***kNONE\_to\_FLEXCOMM3*** Attach NONE to FLEXCOMM3.

***kPLL\_DIV\_to\_FLEXCOMM4*** Attach PLL\_DIV to FLEXCOMM4.

***kFRO12M\_to\_FLEXCOMM4*** Attach FRO12M to FLEXCOMM4.

***kFRO\_HF\_DIV\_to\_FLEXCOMM4*** Attach FRO\_HF\_DIV to FLEXCOMM4.

***kCLK\_1M\_to\_FLEXCOMM4*** Attach CLK\_1MHz to FLEXCOMM4.

***kUSB\_PLL\_to\_FLEXCOMM4*** Attach USB\_PLL to FLEXCOMM4.

***kLPOSC\_to\_FLEXCOMM4*** Attach LP Oscillator to FLEXCOMM4.

***kNONE\_to\_FLEXCOMM4*** Attach NONE to FLEXCOMM4.

***kPLL\_DIV\_to\_FLEXCOMM5*** Attach PLL\_DIV to FLEXCOMM5.

***kFRO12M\_to\_FLEXCOMM5*** Attach FRO12M to FLEXCOMM5.

***kFRO\_HF\_DIV\_to\_FLEXCOMM5*** Attach FRO\_HF\_DIV to FLEXCOMM5.

***kCLK\_1M\_to\_FLEXCOMM5*** Attach CLK\_1MHz to FLEXCOMM5.

***kUSB\_PLL\_to\_FLEXCOMM5*** Attach USB\_PLL to FLEXCOMM5.

***kLPOSC\_to\_FLEXCOMM5*** Attach LP Oscillator to FLEXCOMM5.

***kNONE\_to\_FLEXCOMM5*** Attach NONE to FLEXCOMM5.

***kPLL\_DIV\_to\_FLEXCOMM6*** Attach PLL\_DIV to FLEXCOMM6.

***kFRO12M\_to\_FLEXCOMM6*** Attach FRO12M to FLEXCOMM6.

***kFRO\_HF\_DIV\_to\_FLEXCOMM6*** Attach FRO\_HF\_DIV to FLEXCOMM6.

***kCLK\_1M\_to\_FLEXCOMM6*** Attach CLK\_1MHz to FLEXCOMM6.

***kUSB\_PLL\_to\_FLEXCOMM6*** Attach USB\_PLL to FLEXCOMM6.

***kLPOSC\_to\_FLEXCOMM6*** Attach LP Oscillator to FLEXCOMM6.

***kNONE\_to\_FLEXCOMM6*** Attach NONE to FLEXCOMM6.

***kPLL\_DIV\_to\_FLEXCOMM7*** Attach PLL\_DIV to FLEXCOMM7.

***kFRO12M\_to\_FLEXCOMM7*** Attach FRO12M to FLEXCOMM7.

***kFRO\_HF\_DIV\_to\_FLEXCOMM7*** Attach FRO\_HF\_DIV to FLEXCOMM7.

***kCLK\_1M\_to\_FLEXCOMM7*** Attach CLK\_1MHz to FLEXCOMM7.

***kUSB\_PLL\_to\_FLEXCOMM7*** Attach USB\_PLL to FLEXCOMM7.

***kLPOSC\_to\_FLEXCOMM7*** Attach LP Oscillator to FLEXCOMM7.

***kNONE\_to\_FLEXCOMM7*** Attach NONE to FLEXCOMM7.

***kPLL\_DIV\_to\_FLEXCOMM8*** Attach PLL\_DIV to FLEXCOMM8.

***kFRO12M\_to\_FLEXCOMM8*** Attach FRO12M to FLEXCOMM8.

***kFRO\_HF\_DIV\_to\_FLEXCOMM8*** Attach FRO\_HF\_DIV to FLEXCOMM8.

***kCLK\_1M\_to\_FLEXCOMM8*** Attach CLK\_1MHz to FLEXCOMM8.

***kUSB\_PLL\_to\_FLEXCOMM8*** Attach USB\_PLL to FLEXCOMM8.

***kLPOSC\_to\_FLEXCOMM8*** Attach LP Oscillator to FLEXCOMM8.

***kNONE\_to\_FLEXCOMM8*** Attach NONE to FLEXCOMM8.

***kPLL\_DIV\_to\_FLEXCOMM9*** Attach PLL\_DIV to FLEXCOMM9.

***kFRO12M\_to\_FLEXCOMM9*** Attach FRO12M to FLEXCOMM9.

***kFRO\_HF\_DIV\_to\_FLEXCOMM9*** Attach FRO\_HF\_DIV to FLEXCOMM9.

***kCLK\_1M\_to\_FLEXCOMM9*** Attach CLK\_1MHz to FLEXCOMM9.  
***kUSB\_PLL\_to\_FLEXCOMM9*** Attach USB\_PLL to FLEXCOMM9.  
***kLPOSC\_to\_FLEXCOMM9*** Attach LP Oscillator to FLEXCOMM9.  
***kNONE\_to\_FLEXCOMM9*** Attach NONE to FLEXCOMM9.  
***kPLL0\_to\_SCT*** Attach NONE to SCT.  
***kCLK\_IN\_to\_SCT*** Attach CLK\_in to SCT.  
***kFRO\_HF\_to\_SCT*** Attach FRO\_HF to SCT.  
***kPLL1\_CLK0\_to\_SCT*** Attach PLL1\_clk0 to SCT.  
***kSAI0\_MCLK\_IN\_to\_SCT*** Attach SAI0 MCLK\_IN to SCT.  
***kUSB\_PLL\_to\_SCT*** Attach USB PLL to SCT.  
***kSAI1\_MCLK\_IN\_to\_SCT*** Attach SAI1 MCLK\_IN to SCT.  
***kNONE\_to\_SCT*** Attach NONE to SCT.  
***kCLK\_IN\_to\_TSI*** Attach clk\_in to TSI.  
***kFRO12M\_to\_TSI*** Attach FRO\_12Mhz to TSI.  
***kNONE\_to\_TSI*** Attach NONE to TSI.  
***kPLL0\_to\_SINCFILT*** Attach PLL0 to SINCFILT.  
***kCLK\_IN\_to\_SINCFILT*** Attach clk\_in to SINCFILT.  
***kFRO\_HF\_to\_SINCFILT*** Attach FRO\_HF to SINCFILT.  
***kFRO12M\_to\_SINCFILT*** Attach FRO\_12Mhz to SINCFILT.  
***kPLL1\_CLK0\_to\_SINCFILT*** Attach PLL1\_clk0 to SINCFILT.  
***kUSB\_PLL\_to\_SINCFILT*** Attach USB PLL to SINCFILT.  
***kNONE\_to\_SINCFILT*** Attach NONE to SINCFILT.  
***kPLL0\_to\_ADC1*** Attach PLL0 to ADC1.  
***kFRO\_HF\_to\_ADC1*** Attach FRO\_HF to ADC1.  
***kFRO12M\_to\_ADC1*** Attach FRO12M to ADC1.  
***kCLK\_IN\_to\_ADC1*** Attach clk\_in to ADC1.  
***kPLL1\_CLK0\_to\_ADC1*** Attach PLL1\_clk0 to ADC1.  
***kUSB\_PLL\_to\_ADC1*** Attach USB PLL to ADC1.  
***kNONE\_to\_ADC1*** Attach NONE to ADC1.  
***kPLL0\_to\_DAC0*** Attach PLL0 to DAC0.  
***kCLK\_IN\_to\_DAC0*** Attach Clk\_in to DAC0.  
***kFRO\_HF\_to\_DAC0*** Attach FRO\_HF to DAC0.  
***kFRO12M\_to\_DAC0*** Attach FRO\_12M to DAC0.  
***kPLL1\_CLK0\_to\_DAC0*** Attach PLL1\_clk0 to DAC0.  
***kNONE\_to\_DAC0*** Attach NONE to DAC0.  
***kPLL0\_to\_DAC1*** Attach PLL0 to DAC1.  
***kCLK\_IN\_to\_DAC1*** Attach Clk\_in to DAC1.  
***kFRO\_HF\_to\_DAC1*** Attach FRO\_HF to DAC1.  
***kFRO12M\_to\_DAC1*** Attach FRO\_12M to DAC1.  
***kPLL1\_CLK0\_to\_DAC1*** Attach PLL1\_clk0 to DAC1.  
***kNONE\_to\_DAC1*** Attach NONE to DAC1.  
***kPLL0\_to\_DAC2*** Attach PLL0 to DAC2.  
***kCLK\_IN\_to\_DAC2*** Attach Clk\_in to DAC2.  
***kFRO\_HF\_to\_DAC2*** Attach FRO\_HF to DAC2.  
***kFRO12M\_to\_DAC2*** Attach FRO\_12M to DAC2.

***kPLL1\_CLK0\_to\_DAC2*** Attach PLL1\_clk0 to DAC2.

***kNONE\_to\_DAC2*** Attach NONE to DAC2.

***kPLL0\_to\_FLEXSPI*** Attach PLL0 to FLEXSPI.

***kFRO\_HF\_to\_FLEXSPI*** Attach FRO\_HF to FLEXSPI.

***kPLL1\_to\_FLEXSPI*** Attach PLL1 to FLEXSPI.

***kUSB\_PLL\_to\_FLEXSPI*** Attach USB PLL to FLEXSPI.

***kNONE\_to\_FLEXSPI*** Attach NONE to FLEXSPI.

***kPLL0\_to\_PLLCLKDIV*** Attach PLL0 to PLLCLKDIV.

***kPLL1\_CLK0\_to\_PLLCLKDIV*** Attach pll1\_clk0 to PLLCLKDIV.

***kNONE\_to\_PLLCLKDIV*** Attach NONE to PLLCLKDIV.

***kPLL0\_to\_I3C0FCLK*** Attach PLL0 to I3C0FCLK.

***kCLK\_IN\_to\_I3C0FCLK*** Attach Clk\_in to I3C0FCLK.

***kFRO\_HF\_to\_I3C0FCLK*** Attach FRO\_HF to I3C0FCLK.

***kPLL1\_CLK0\_to\_I3C0FCLK*** Attach PLL1\_clk0 to I3C0FCLK.

***kUSB\_PLL\_to\_I3C0FCLK*** Attach USB PLL to I3C0FCLK.

***kNONE\_to\_I3C0FCLK*** Attach NONE to I3C0FCLK.

***kI3C0FCLK\_to\_I3C0FCLKSTC*** Attach I3C0FCLK to I3C0FCLKSTC.

***kCLK\_1M\_to\_I3C0FCLKSTC*** Attach CLK\_1M to I3C0FCLKSTC.

***kNONE\_to\_I3C0FCLKSTC*** Attach NONE to I3C0FCLKSTC.

***kCLK\_1M\_to\_I3C0FCLKS*** Attach CLK\_1M to I3C0FCLKS.

***kNONE\_to\_I3C0FCLKS*** Attach NONE to I3C0FCLKS.

***kFRO12M\_to\_MICFILF*** Attach FRO\_12M to MICFILF.

***kPLL0\_to\_MICFILF*** Attach PLL0 to MICFILF.

***kCLK\_IN\_to\_MICFILF*** Attach Clk\_in to MICFILF.

***kFRO\_HF\_to\_MICFILF*** Attach FRO\_HF to MICFILF.

***kPLL1\_CLK0\_to\_MICFILF*** Attach PLL1\_clk0 to MICFILF.

***kSAI0\_MCLK\_to\_MICFILF*** Attach SAI0\_MCLK to MICFILF.

***kUSB\_PLL\_to\_MICFILF*** Attach USB PLL to MICFILF.

***kSAI1\_MCLK\_to\_MICFILF*** Attach SAI1\_MCLK to MICFILF.

***kNONE\_to\_MICFILF*** Attach NONE to MICFILF.

***kPLL0\_to\_ESPI*** Attach PLL0 to ESPI.

***kCLK\_48M\_to\_ESPI*** Attach CLK\_48M to ESPI.

***kPLL1\_CLK0\_to\_ESPI*** Attach PLL1\_clk0 to ESPI.

***kUSB\_PLL\_to\_ESPI*** Attach USB PLL to ESPI.

***kNONE\_to\_ESPI*** Attach NONE to ESPI.

***kPLL0\_to\_USDHC*** Attach PLL0 to uSDHC.

***kCLK\_IN\_to\_USDHC*** Attach Clk\_in to uSDHC.

***kFRO\_HF\_to\_USDHC*** Attach FRO\_HF to uSDHC.

***kFRO12M\_to\_USDHC*** Attach FRO\_12M to uSDHC.

***kPLL1\_CLK1\_to\_USDHC*** Attach pll1\_clk1 to uSDHC.

***kUSB\_PLL\_to\_USDHC*** Attach USB PLL to uSDHC.

***kNONE\_to\_USDHC*** Attach NONE to uSDHC.

***kPLL0\_to\_FLEXIO*** Attach PLL0 to FLEXIO.

***kCLK\_IN\_to\_FLEXIO*** Attach Clk\_in to FLEXIO.

***kFRO\_HF\_to\_FLEXIO*** Attach FRO\_HF to FLEXIO.

***kFRO12M\_to\_FLEXIO*** Attach FRO\_12M to FLEXIO.

***kPLL1\_CLK0\_to\_FLEXIO*** Attach pll1\_clk0 to FLEXIO.

***kUSB\_PLL\_to\_FLEXIO*** Attach USB PLL to FLEXIO.

***kNONE\_to\_FLEXIO*** Attach NONE to FLEXIO.

***kPLL0\_to\_FLEXCAN0*** Attach PLL0 to FLEXCAN0.

***kCLK\_IN\_to\_FLEXCAN0*** Attach Clk\_in to FLEXCAN0.

***kFRO\_HF\_to\_FLEXCAN0*** Attach FRO\_HF to FLEXCAN0.

***kPLL1\_CLK0\_to\_FLEXCAN0*** Attach pll1\_clk0 to FLEXCAN0.

***kUSB\_PLL\_to\_FLEXCAN0*** Attach USB PLL to FLEXCAN0.

***kNONE\_to\_FLEXCAN0*** Attach NONE to FLEXCAN0.

***kPLL0\_to\_FLEXCAN1*** Attach PLL0 to FLEXCAN1.

***kCLK\_IN\_to\_FLEXCAN1*** Attach Clk\_in to FLEXCAN1.

***kFRO\_HF\_to\_FLEXCAN1*** Attach FRO\_HF to FLEXCAN1.

***kPLL1\_CLK0\_to\_FLEXCAN1*** Attach pll1\_clk0 to FLEXCAN1.

***kUSB\_PLL\_to\_FLEXCAN1*** Attach USB PLL to FLEXCAN1.

***kNONE\_to\_FLEXCAN1*** Attach NONE to FLEXCAN1.

***kNONE\_to\_ENETRMII*** Attach NONE to ENETRMII.

***kPLL0\_to\_ENETRMII*** Attach PLL0 to ENETRMII.

***kCLK\_IN\_to\_ENETRMII*** Attach Clk\_in to ENETRMII.

***kPLL1\_CLK0\_to\_ENETRMII*** Attach pll1\_clk0 to ENETRMII.

***kPLL0\_to\_ENETPTPREF*** Attach PLL0 to ENETPTPREF.

***kCLK\_IN\_to\_ENETPTPREF*** Attach Clk\_in to ENETPTPREF.

***kENET0\_TX\_CLK\_to\_ENETPTPREF*** Attach enet0\_tx\_clk to ENETPTPREF.

***kPLL1\_CLK1\_to\_ENETPTPREF*** Attach pll1\_clk1 to ENETPTPREF.

***kNONE\_to\_ENETPTPREF*** Attach NONE to ENETPTPREF.

***kCLK\_16K2\_to\_EWM0*** Attach clk\_16k[2] to EWM0.

***kXTAL32K2\_to\_EWM0*** Attach xtal32k[2] to EWM0.

***kCLK\_16K2\_to\_WDT1*** Attach FRO16K clock 2 to WDT1.

***kFRO\_HF\_DIV\_to\_WDT1*** Attach FRO\_HF\_DIV to WDT1.

***kCLK\_1M\_to\_WDT1*** Attach clk\_1m to WDT1.

***kNONE\_to\_WDT1*** Attach NONE to WDT1.

***kCLK\_16K2\_to\_OSTIMER*** Attach clk\_16k[2] to OSTIMER.

***kXTAL32K2\_to\_OSTIMER*** Attach xtal32k[2] to OSTIMER.

***kCLK\_1M\_to\_OSTIMER*** Attach clk\_1m to OSTIMER.

***kNONE\_to\_OSTIMER*** Attach NONE to OSTIMER.

***kPLL0\_to\_CMP0F*** Attach PLL0 to CMP0F.

***kFRO\_HF\_to\_CMP0F*** Attach FRO\_HF to CMP0F.

***kFRO12M\_to\_CMP0F*** Attach FRO\_12M to CMP0F.

***kCLK\_IN\_to\_CMP0F*** Attach Clk\_in to CMP0F.

***kPLL1\_CLK0\_to\_CMP0F*** Attach PLL1\_clk0 to CMP0F.

***kUSB\_PLL\_to\_CMP0F*** Attach USB PLL to CMP0F.

***kNONE\_to\_CMP0F*** Attach NONE to CMP0F.

***kPLL0\_to\_CMP0RR*** Attach PLL0 to CMP0RR.

***kFRO\_HF\_to\_CMP0RR*** Attach FRO\_HF to CMP0RR.

***kFRO12M\_to\_CMP0RR*** Attach FRO\_12M to CMP0RR.

***kCLK\_IN\_to\_CMP0RR*** Attach Clk\_in to CMP0RR.

***kPLL1\_CLK0\_to\_CMP0RR*** Attach PLL1\_clk0 to CMP0RR.

***kUSB\_PLL\_to\_CMP0RR*** Attach USB PLL to CMP0RR.

***kNONE\_to\_CMP0RR*** Attach NONE to CMP0RR.

***kPLL0\_to\_CMP1F*** Attach PLL0 to CMP1F.

***kFRO\_HF\_to\_CMP1F*** Attach FRO\_HF to CMP1F.

***kFRO12M\_to\_CMP1F*** Attach FRO\_12M to CMP1F.

***kCLK\_IN\_to\_CMP1F*** Attach Clk\_in to CMP1F.

***kPLL1\_CLK0\_to\_CMP1F*** Attach PLL1\_clk0 to CMP1F.

***kUSB\_PLL\_to\_CMP1F*** Attach USB PLL to CMP1F.

***kNONE\_to\_CMP1F*** Attach NONE to CMP1F.

***kPLL0\_to\_CMP1RR*** Attach PLL0 to CMP1RR.

***kFRO\_HF\_to\_CMP1RR*** Attach FRO\_HF to CMP1RR.

***kFRO12M\_to\_CMP1RR*** Attach FRO\_12M to CMP1RR.

***kCLK\_IN\_to\_CMP1RR*** Attach Clk\_in to CMP1RR.

***kPLL1\_CLK0\_to\_CMP1RR*** Attach PLL1\_clk0 to CMP1RR.

***kUSB\_PLL\_to\_CMP1RR*** Attach USB PLL to CMP1RR.

***kNONE\_to\_CMP1RR*** Attach NONE to CMP1RR.

***kPLL0\_to\_CMP2F*** Attach PLL0 to CMP2F.

***kFRO\_HF\_to\_CMP2F*** Attach FRO\_HF to CMP2F.

***kFRO12M\_to\_CMP2F*** Attach FRO\_12M to CMP2F.

***kCLK\_IN\_to\_CMP2F*** Attach Clk\_in to CMP2F.

***kPLL1\_CLK0\_to\_CMP2F*** Attach PLL1\_clk0 to CMP2F.

***kUSB\_PLL\_to\_CMP2F*** Attach USB PLL to CMP2F.

***kNONE\_to\_CMP2F*** Attach NONE to CMP2F.

***kPLL0\_to\_CMP2RR*** Attach PLL0 to CMP2RR.

***kFRO\_HF\_to\_CMP2RR*** Attach FRO\_HF to CMP2RR.

***kFRO12M\_to\_CMP2RR*** Attach FRO\_12M to CMP2RR.

***kCLK\_IN\_to\_CMP2RR*** Attach Clk\_in to CMP2RR.

***kPLL1\_CLK0\_to\_CMP2RR*** Attach PLL1\_clk0 to CMP2RR.

***kUSB\_PLL\_to\_CMP2RR*** Attach USB PLL to CMP2RR.

***kNONE\_to\_CMP2RR*** Attach NONE to CMP2RR.

***kPLL0\_to\_SAI0*** Attach PLL0 to SAI0.

***kCLK\_IN\_to\_SAI0*** Attach Clk\_in to SAI0.

***kFRO\_HF\_to\_SAI0*** Attach FRO\_HF to SAI0.

***kPLL1\_CLK0\_to\_SAI0*** Attach PLL1\_clk0 to SAI0.

***kUSB\_PLL\_to\_SAI0*** Attach USB PLL to SAI0.

***kNONE\_to\_SAI0*** Attach NONE to SAI0.

***kPLL0\_to\_SAI1*** Attach PLL0 to SAI1.

***kCLK\_IN\_to\_SAI1*** Attach Clk\_in to SAI1.

***kFRO\_HF\_to\_SAI1*** Attach FRO\_HF to SAI1.

***kPLL1\_CLK0\_to\_SAI1*** Attach PLL1\_clk0 to SAI1.

***kUSB\_PLL\_to\_SAI1*** Attach USB PLL to SAI1.

***kNONE\_to\_SAI1*** Attach NONE to SAI1.

***kPLL0\_to\_EMVSIM0*** Attach PLL0 to EMVSIM0.

*kCLK\_IN\_to\_EMVSIM0* Attach Clk\_in to EMVSIM0.  
*kFRO\_HF\_to\_EMVSIM0* Attach FRO\_HF to EMVSIM0.  
*kFRO12M\_to\_EMVSIM0* Attach FRO\_12M to EMVSIM0.  
*kPLL1\_CLK0\_to\_EMVSIM0* Attach PLL1\_clk0 to EMVSIM0.  
*kNONE\_to\_EMVSIM0* Attach NONE to EMVSIM0.  
*kPLL0\_to\_EMVSIM1* Attach PLL0 to EMVSIM1.  
*kCLK\_IN\_to\_EMVSIM1* Attach Clk\_in to EMVSIM1.  
*kFRO\_HF\_to\_EMVSIM1* Attach FRO\_HF to EMVSIM1.  
*kFRO12M\_to\_EMVSIM1* Attach FRO\_12M to EMVSIM1.  
*kPLL1\_CLK0\_to\_EMVSIM1* Attach PLL1\_clk0 to EMVSIM1.  
*kNONE\_to\_EMVSIM1* Attach NONE to EMVSIM1.  
*kPLL0\_to\_I3C1FCLK* Attach PLL0 to I3C1FCLK.  
*kCLK\_IN\_to\_I3C1FCLK* Attach Clk\_in to I3C1FCLK.  
*kFRO\_HF\_to\_I3C1FCLK* Attach FRO\_HF to I3C1FCLK.  
*kPLL1\_CLK0\_to\_I3C1FCLK* Attach PLL1\_clk0 to I3C1FCLK.  
*kUSB\_PLL\_to\_I3C1FCLK* Attach USB PLL to I3C1FCLK.  
*kNONE\_to\_I3C1FCLK* Attach NONE to I3C1FCLK.  
*kI3C1FCLK\_to\_I3C1FCLKSTC* Attach I3C1FCLK to I3C1FCLKSTC.  
*kCLK\_1M\_to\_I3C1FCLKSTC* Attach CLK\_1M to I3C1FCLKSTC.  
*kNONE\_to\_I3C1FCLKSTC* Attach NONE to I3C1FCLKSTC.  
*kCLK\_1M\_to\_I3C1FCLKS* Attach CLK\_1M to I3C1FCLKS.  
*kNONE\_to\_I3C1FCLKS* Attach NONE to I3C1FCLKS.  
*kNONE\_to\_NONE* Attach NONE to NONE.

#### 4.4.4 enum clock\_div\_name\_t

Enumerator

*kCLOCK\_DivSystickClk0* Systick Clk0 Divider.  
*kCLOCK\_DivSystickClk1* Systick Clk1 Divider.  
*kCLOCK\_DivTraceClk* Trace Clk Divider.  
*kCLOCK\_DivSlowClk* SLOW CLK Divider.  
*kCLOCK\_DivTsiClk* Tsi Clk Divider.  
*kCLOCK\_DivAhbClk* Ahb Clk Divider.  
*kCLOCK\_DivClkOut* ClkOut Clk Divider.  
*kCLOCK\_DivFrohfClk* Frohf Clk Divider.  
*kCLOCK\_DivWdt0Clk* Wdt0 Clk Divider.  
*kCLOCK\_DivAdc0Clk* Adc0 Clk Divider.  
*kCLOCK\_DivUsb0Clk* Usb0 Clk Divider.  
*kCLOCK\_DivSctClk* Sct Clk Divider.  
*kCLOCK\_DivPllClk* Pll Clk Divider.  
*kCLOCK\_DivCtimer0Clk* Ctimer0 Clk Divider.  
*kCLOCK\_DivCtimer1Clk* Ctimer1 Clk Divider.  
*kCLOCK\_DivCtimer2Clk* Ctimer2 Clk Divider.

*kCLOCK\_DivTimer3Clk* Ctimer3 Clk Divider.  
*kCLOCK\_DivTimer4Clk* Ctimer4 Clk Divider.  
*kCLOCK\_DivPLL1Clk0* PLL1 Clk0 Divider.  
*kCLOCK\_DivPLL1Clk1* Pll1 Clk1 Divider.  
*kCLOCK\_DivAdc1Clk* Adc1 Clk Divider.  
*kCLOCK\_DivDac0Clk* Dac0 Clk Divider.  
*kCLOCK\_DivDac1Clk* Dac1 Clk Divider.  
*kCLOCK\_DivDac2Clk* Dac2 Clk Divider.  
*kCLOCK\_DivFlexspiClk* Flexspi Clk Divider.  
*kCLOCK\_DivI3c0FClkStc* I3C0 FCLK STC Divider.  
*kCLOCK\_DivI3c0FClkS* I3C0 FCLK S Divider.  
*kCLOCK\_DivI3c0FClk* I3C0 FClk Divider.  
*kCLOCK\_DivMicfilFClk* MICFILFCLK Divider.  
*kCLOCK\_DivEspiClk* Espi Clk Divider.  
*kCLOCK\_DivUSdhcClk* USdhc Clk Divider.  
*kCLOCK\_DivFlexioClk* Flexio Clk Divider.  
*kCLOCK\_DivFlexcan0Clk* Flexcan0 Clk Divider.  
*kCLOCK\_DivFlexcan1Clk* Flexcan1 Clk Divider.  
*kCLOCK\_DivEnetrmiiClk* Enetrmii Clk Divider.  
*kCLOCK\_DivEnetptprefClk* Enetptpref Clk Divider.  
*kCLOCK\_DivWdt1Clk* Wdt1 Clk Divider.  
*kCLOCK\_DivCmp0FClk* Cmp0 FClk Divider.  
*kCLOCK\_DivCmp0rrClk* Cmp0rr Clk Divider.  
*kCLOCK\_DivCmp1FClk* Cmp1 FClk Divider.  
*kCLOCK\_DivCmp1rrClk* Cmp1rr Clk Divider.  
*kCLOCK\_DivCmp2FClk* Cmp2 FClk Divider.  
*kCLOCK\_DivCmp2rrClk* Cmp2rr Clk Divider.  
*kCLOCK\_DivFlexcom0Clk* Flexcom0 Clk Divider.  
*kCLOCK\_DivFlexcom1Clk* Flexcom1 Clk Divider.  
*kCLOCK\_DivFlexcom2Clk* Flexcom2 Clk Divider.  
*kCLOCK\_DivFlexcom3Clk* Flexcom3 Clk Divider.  
*kCLOCK\_DivFlexcom4Clk* Flexcom4 Clk Divider.  
*kCLOCK\_DivFlexcom5Clk* Flexcom5 Clk Divider.  
*kCLOCK\_DivFlexcom6Clk* Flexcom6 Clk Divider.  
*kCLOCK\_DivFlexcom7Clk* Flexcom7 Clk Divider.  
*kCLOCK\_DivFlexcom8Clk* Flexcom8 Clk Divider.  
*kCLOCK\_DivFlexcom9Clk* Flexcom9 Clk Divider.  
*kCLOCK\_DivSai0Clk* Sai0 Clk Divider.  
*kCLOCK\_DivSai1Clk* Sai1 Clk Divider.  
*kCLOCK\_DivEmvsim0Clk* Emvsim0 Clk Divider.  
*kCLOCK\_DivEmvsim1Clk* Emvsim1 Clk Divider.  
*kCLOCK\_DivI3c1FClkStc* I3C1 FCLK STC Divider.  
*kCLOCK\_DivI3c1FClkS* I3C1 FCLK S Divider.  
*kCLOCK\_DivI3c1FClk* I3C1 FClk Divider.

#### 4.4.5 enum osc32k\_clk\_gate\_id\_t

Enumerator

*kCLOCK\_Osc32kToVbat* OSC32K[0] to VBAT domain.  
*kCLOCK\_Osc32kToVsys* OSC32K[1] to VSYS domain.  
*kCLOCK\_Osc32kToWake* OSC32K[2] to WAKE domain.  
*kCLOCK\_Osc32kToMain* OSC32K[3] to MAIN domain.  
*kCLOCK\_Osc32kToAll* OSC32K to VBAT,VSYS,WAKE,MAIN domain.

#### 4.4.6 enum clk16k\_clk\_gate\_id\_t

Enumerator

*kCLOCK\_Clk16KToVbat* Clk16k[0] to VBAT domain.  
*kCLOCK\_Clk16KToVsys* Clk16k[1] to VSYS domain.  
*kCLOCK\_Clk16KToWake* Clk16k[2] to WAKE domain.  
*kCLOCK\_Clk16KToMain* Clk16k[3] to MAIN domain.  
*kCLOCK\_Clk16KToAll* Clk16k to VBAT,VSYS,WAKE,MAIN domain.

#### 4.4.7 enum clock\_ctrl\_enable\_t

Enumerator

*kCLOCK\_PLU\_DEGLITCH\_CLK\_ENA* Enables clocks FRO\_1MHz and FRO\_12MHz for PLU deglitching.  
*kCLOCK\_FRO1MHZ\_CLK\_ENA* Enables FRO\_1MHz clock for clock muxing in clock gen.  
*kCLOCK\_CLKIN\_ENA* Enables clk\_in clock for MICD, EMVSIM0/1, CAN0/1, I3C0/1, SAI0/1, SINC Filter (SINC), TSI, USBFS, SCT, uSDHC, clkout.  
*kCLOCK\_FRO\_HF\_ENA* Enables FRO HF clock for the Frequency Measure module.  
*kCLOCK\_FRO12MHZ\_ENA* Enables the FRO\_12MHz clock for the Flash, LPTIMER0/1, and Frequency Measurement modules.  
*kCLOCK\_FRO1MHZ\_ENA* Enables the FRO\_1MHz clock for RTC module and for UTICK.  
*kCLOCK\_CLKIN\_ENA\_FM\_USBH\_LPT* Enables the clk\_in clock for the Frequency Measurement, USB HS and LPTIMER0/1 modules.

#### 4.4.8 enum clock\_usb\_phy\_src\_t

Enumerator

*kCLOCK\_Usbphy480M* Use 480M.

#### 4.4.9 enum \_scg\_status

Enumerator

- kStatus\_SCG\_Busy* Clock is busy.
- kStatus\_SCG\_InvalidSrc* Invalid source.

#### 4.4.10 enum firc\_trim\_mode\_t

Enumerator

- kSCG\_FircTrimNonUpdate* Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure trim\_config\_t.
- kSCG\_FircTrimUpdate* Trim enable and trim value update enable. In this mode, the trim value is auto update.

#### 4.4.11 enum firc\_trim\_src\_t

Enumerator

- kSCG\_FircTrimSrcUsb0* USB0 start of frame (1kHz).
- kSCG\_FircTrimSrcSysOsc* System OSC.
- kSCG\_FircTrimSrcRtcOsc* RTC OSC (32.768 kHz).

#### 4.4.12 enum sirc\_trim\_mode\_t

Enumerator

- kSCG\_SircTrimNonUpdate* Trim enable but not enable trim value update. In this mode, the trim value is fixed to the initialized value which is defined by trimCoar and trimFine in configure structure trim\_config\_t.
- kSCG\_SircTrimUpdate* Trim enable and trim value update enable. In this mode, the trim value is auto update.

#### 4.4.13 enum sirc\_trim\_src\_t

Enumerator

- kSCG\_SircTrimSrcSysOsc* System OSC.
- kSCG\_SircTrimSrcRtcOsc* RTC OSC (32.768 kHz).

**4.4.14 enum scg\_sosc\_monitor\_mode\_t**

Enumerator

*kSCG\_SysOscMonitorDisable* Monitor disabled.*kSCG\_SysOscMonitorInt* Interrupt when the SOSC error is detected.*kSCG\_SysOscMonitorReset* Reset when the SOSC error is detected.**4.4.15 enum scg\_rosc\_monitor\_mode\_t**

Enumerator

*kSCG\_RoscMonitorDisable* Monitor disabled.*kSCG\_RoscMonitorInt* Interrupt when the RTC OSC error is detected.*kSCG\_RoscMonitorReset* Reset when the RTC OSC error is detected.**4.4.16 enum scg\_upll\_monitor\_mode\_t**

Enumerator

*kSCG\_UpllMonitorDisable* Monitor disabled.*kSCG\_UpllMonitorInt* Interrupt when the UPLL error is detected.*kSCG\_UpllMonitorReset* Reset when the UPLL error is detected.**4.4.17 enum scg\_pll0\_monitor\_mode\_t**

Enumerator

*kSCG\_Pll0MonitorDisable* Monitor disabled.*kSCG\_Pll0MonitorInt* Interrupt when the PLL0 Clock error is detected.*kSCG\_Pll0MonitorReset* Reset when the PLL0 Clock error is detected.**4.4.18 enum scg\_pll1\_monitor\_mode\_t**

Enumerator

*kSCG\_Pll1MonitorDisable* Monitor disabled.*kSCG\_Pll1MonitorInt* Interrupt when the PLL1 Clock error is detected.*kSCG\_Pll1MonitorReset* Reset when the PLL1 Clock error is detected.

#### 4.4.19 enum vbat\_osc\_xtal\_cap\_t

Enumerator

<i>kVBAT_OscXtal0pFCap</i>	The internal capacitance for XTAL pin is 0pF.
<i>kVBAT_OscXtal2pFCap</i>	The internal capacitance for XTAL pin is 2pF.
<i>kVBAT_OscXtal4pFCap</i>	The internal capacitance for XTAL pin is 4pF.
<i>kVBAT_OscXtal6pFCap</i>	The internal capacitance for XTAL pin is 6pF.
<i>kVBAT_OscXtal8pFCap</i>	The internal capacitance for XTAL pin is 8pF.
<i>kVBAT_OscXtal10pFCap</i>	The internal capacitance for XTAL pin is 10pF.
<i>kVBAT_OscXtal12pFCap</i>	The internal capacitance for XTAL pin is 12pF.
<i>kVBAT_OscXtal14pFCap</i>	The internal capacitance for XTAL pin is 14pF.
<i>kVBAT_OscXtal16pFCap</i>	The internal capacitance for XTAL pin is 16pF.
<i>kVBAT_OscXtal18pFCap</i>	The internal capacitance for XTAL pin is 18pF.
<i>kVBAT_OscXtal20pFCap</i>	The internal capacitance for XTAL pin is 20pF.
<i>kVBAT_OscXtal22pFCap</i>	The internal capacitance for XTAL pin is 22pF.
<i>kVBAT_OscXtal24pFCap</i>	The internal capacitance for XTAL pin is 24pF.
<i>kVBAT_OscXtal26pFCap</i>	The internal capacitance for XTAL pin is 26pF.
<i>kVBAT_OscXtal28pFCap</i>	The internal capacitance for XTAL pin is 28pF.
<i>kVBAT_OscXtal30pFCap</i>	The internal capacitance for XTAL pin is 30pF.

#### 4.4.20 enum vbat\_osc\_extal\_cap\_t

Enumerator

<i>kVBAT_OscExtal0pFCap</i>	The internal capacitance for EXTAL pin is 0pF.
<i>kVBAT_OscExtal2pFCap</i>	The internal capacitance for EXTAL pin is 2pF.
<i>kVBAT_OscExtal4pFCap</i>	The internal capacitance for EXTAL pin is 4pF.
<i>kVBAT_OscExtal6pFCap</i>	The internal capacitance for EXTAL pin is 6pF.
<i>kVBAT_OscExtal8pFCap</i>	The internal capacitance for EXTAL pin is 8pF.
<i>kVBAT_OscExtal10pFCap</i>	The internal capacitance for EXTAL pin is 10pF.
<i>kVBAT_OscExtal12pFCap</i>	The internal capacitance for EXTAL pin is 12pF.
<i>kVBAT_OscExtal14pFCap</i>	The internal capacitance for EXTAL pin is 14pF.
<i>kVBAT_OscExtal16pFCap</i>	The internal capacitance for EXTAL pin is 16pF.
<i>kVBAT_OscExtal18pFCap</i>	The internal capacitance for EXTAL pin is 18pF.
<i>kVBAT_OscExtal20pFCap</i>	The internal capacitance for EXTAL pin is 20pF.
<i>kVBAT_OscExtal22pFCap</i>	The internal capacitance for EXTAL pin is 22pF.
<i>kVBAT_OscExtal24pFCap</i>	The internal capacitance for EXTAL pin is 24pF.
<i>kVBAT_OscExtal26pFCap</i>	The internal capacitance for EXTAL pin is 26pF.
<i>kVBAT_OscExtal28pFCap</i>	The internal capacitance for EXTAL pin is 28pF.
<i>kVBAT_OscExtal30pFCap</i>	The internal capacitance for EXTAL pin is 30pF.

**4.4.21 enum vbat\_osc\_coarse\_adjustment\_value\_t**

Changes the oscillator amplitude by modifying the automatic gain control (AGC).

**4.4.22 enum pll\_clk\_src\_t**

Enumerator

*kPlI\_ClkSrcSysOsc* System OSC.

*kPlI\_ClkSrcFirc* Fast IRC.

*kPlI\_ClkSrcRosc* RTC OSC.

**4.4.23 enum ss\_progmodfm\_t**

Enumerator

*kSS\_MF\_512* NSS = 512 (fm  $\sim=$  3.9 - 7.8 kHz)

*kSS\_MF\_384* NSS  $\sim=$  384 (fm  $\sim=$  5.2 - 10.4 kHz)

*kSS\_MF\_256* NSS = 256 (fm  $\sim=$  7.8 - 15.6 kHz)

*kSS\_MF\_128* NSS = 128 (fm  $\sim=$  15.6 - 31.3 kHz)

*kSS\_MF\_64* NSS = 64 (fm  $\sim=$  32.3 - 64.5 kHz)

*kSS\_MF\_32* NSS = 32 (fm  $\sim=$  62.5 - 125 kHz)

*kSS\_MF\_24* NSS  $\sim=$  24 (fm  $\sim=$  83.3 - 166.6 kHz)

*kSS\_MF\_16* NSS = 16 (fm  $\sim=$  125 - 250 kHz)

**4.4.24 enum ss\_progmoddp\_t**

Enumerator

*kSS\_MR\_K0* k = 0 (no spread spectrum)

*kSS\_MR\_K1* k  $\sim=$  1

*kSS\_MR\_K1\_5* k  $\sim=$  1.5

*kSS\_MR\_K2* k  $\sim=$  2

*kSS\_MR\_K3* k  $\sim=$  3

*kSS\_MR\_K4* k  $\sim=$  4

*kSS\_MR\_K6* k  $\sim=$  6

*kSS\_MR\_K8* k  $\sim=$  8

#### 4.4.25 enum ss\_modwvctrl\_t

Compensation for low pass filtering of the PLL to get a triangular modulation at the output of the PLL, giving a flat frequency spectrum.

Enumerator

- kSS\_MC\_NOC* no compensation
- kSS\_MC\_RECC* recommended setting
- kSS\_MC\_MAXC* max. compensation

#### 4.4.26 enum pll\_error\_t

Enumerator

- kStatus\_PLL\_Success* PLL operation was successful.
- kStatus\_PLL\_OutputTooLow* PLL output rate request was too low.
- kStatus\_PLL\_OutputTooHigh* PLL output rate request was too high.
- kStatus\_PLL\_OutputError* PLL output rate error.
- kStatus\_PLL\_InputTooLow* PLL input rate is too low.
- kStatus\_PLL\_InputTooHigh* PLL input rate is too high.
- kStatus\_PLL\_OutsideIntLimit* Requested output rate isn't possible.
- kStatus\_PLL\_CCOTooLow* Requested CCO rate isn't possible.
- kStatus\_PLL\_CCOTooHigh* Requested CCO rate isn't possible.

### 4.5 Function Documentation

#### 4.5.1 static void CLOCK\_EnableClock ( `clock_ip_name_t clk` ) [inline], [static]

Parameters

<i>clk</i>	: Clock to be enabled.
------------	------------------------

Returns

Nothing

#### 4.5.2 static void CLOCK\_DisableClock ( `clock_ip_name_t clk` ) [inline], [static]

Parameters

<code>clk</code>	: Clock to be Disabled.
------------------	-------------------------

Returns

Nothing

#### 4.5.3 `status_t CLOCK_SetupFROHFClocking ( uint32_t iFreq )`

Parameters

<code>iFreq</code>	: Desired frequency (must be one of CLK_FRO_44MHZ or CLK_FRO_144MHZ)
--------------------	--

Returns

returns success or fail status.

#### 4.5.4 `status_t CLOCK_SetupExtClocking ( uint32_t iFreq )`

Parameters

<code>iFreq</code>	: Desired frequency (must be equal to exact rate in Hz)
--------------------	---

Returns

returns success or fail status.

#### 4.5.5 `status_t CLOCK_SetupOsc32KClocking ( uint32_t id )`

Parameters

<code>id</code>	: OSC 32 kHz output clock to specified modules, it should use osc32k_clk_gate_id_t value
-----------------	--

Returns

returns success or fail status.

#### 4.5.6 `status_t CLOCK_SetupClk16KClocking ( uint32_t id )`

Parameters

<i>id</i>	: FRO 16 kHz output clock to specified modules, it should use clk16k_clk_gate_id_t value
-----------	--

Returns

returns success or fail status.

#### 4.5.7 status\_t CLOCK\_FROHFTrimConfig ( firc\_trim\_config\_t config )

Parameters

<i>config</i>	: FROHF trim value
---------------	--------------------

Returns

returns success or fail status.

#### 4.5.8 status\_t CLOCK\_FRO12MTrimConfig ( sirc\_trim\_config\_t config )

Parameters

<i>config</i>	: FRO 12M trim value
---------------	----------------------

Returns

returns success or fail status.

#### 4.5.9 void CLOCK\_SetSysOscMonitorMode ( scg\_sosc\_monitor\_mode\_t mode )

This function sets the system OSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

<i>mode</i>	Monitor mode to set.
-------------	----------------------

#### 4.5.10 void CLOCK\_SetRoscMonitorMode ( scg\_rosc\_monitor\_mode\_t *mode* )

This function sets the ROSC monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

<i>mode</i>	Monitor mode to set.
-------------	----------------------

#### 4.5.11 void CLOCK\_SetUpllMonitorMode ( scg\_upll\_monitor\_mode\_t *mode* )

This function sets the UPLL monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

<i>mode</i>	Monitor mode to set.
-------------	----------------------

#### 4.5.12 void CLOCK\_SetPll0MonitorMode ( scg\_pll0\_monitor\_mode\_t *mode* )

This function sets the PLL0 monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

<i>mode</i>	Monitor mode to set.
-------------	----------------------

#### 4.5.13 void CLOCK\_SetPll1MonitorMode ( scg\_pll1\_monitor\_mode\_t *mode* )

This function sets the PLL1 monitor mode. The mode can be disabled, it can generate an interrupt when the error is disabled, or reset when the error is detected.

Parameters

<i>mode</i>	Monitor mode to set.
-------------	----------------------

#### 4.5.14 void VBAT\_SetOscConfig ( **VBAT\_Type** \* *base*, **const vbat\_osc\_config\_t** \* *config* )

Parameters

<i>base</i>	VBAT peripheral base address.
<i>config</i>	The pointer to the structure <a href="#">vbat_osc_config_t</a> .

#### 4.5.15 void CLOCK\_AttachClk ( **clock\_attach\_id\_t** *connection* )

Parameters

<i>connection</i>	: Clock to be configured.
-------------------	---------------------------

Returns

Nothing

#### 4.5.16 **clock\_attach\_id\_t** CLOCK\_GetClockAttachId ( **clock\_attach\_id\_t** *attachId* )

Parameters

<i>attachId</i>	: Clock attach id to get.
-----------------	---------------------------

Returns

Clock source value.

#### 4.5.17 void CLOCK\_SetClkDiv ( **clock\_div\_name\_t** *div\_name*, **uint32\_t** *divided\_by\_value* )

Parameters

<i>div_name</i>	: Clock divider name
<i>divided_by_value,:</i>	Value to be divided

Returns

Nothing

#### 4.5.18 `uint32_t CLOCK_GetClkDiv ( clock_div_name_t div_name )`

Parameters

<i>div_name</i>	: Clock divider name
-----------------	----------------------

Returns

peripheral clock dividers

#### 4.5.19 `void CLOCK_HaltClkDiv ( clock_div_name_t div_name )`

Parameters

<i>div_name</i>	: Clock divider name
-----------------	----------------------

Returns

Nothing

#### 4.5.20 `void CLOCK_SetupClockCtrl ( uint32_t mask )`

Parameters

---

<i>mask</i>	: system clocks enable value, it should use <code>clock_ctrl_enable_t</code> value
-------------	--

Returns

Nothing

#### **4.5.21 `uint32_t CLOCK_GetFreq( clock_name_t clockName )`**

Returns

Frequency of selected clock

#### **4.5.22 `uint32_t CLOCK_GetCoreSysClkFreq( void )`**

Returns

Frequency of the core

#### **4.5.23 `uint32_t CLOCK_GetCTimerClkFreq( uint32_t id )`**

Returns

Frequency of CTimer functional Clock

#### **4.5.24 `uint32_t CLOCK_GetAdcClkFreq( uint32_t id )`**

Returns

Frequency of Adc.

#### **4.5.25 `uint32_t CLOCK_GetUsb0ClkFreq( void )`**

Returns

Frequency of Adc.

**4.5.26 `uint32_t CLOCK_GetLPFlexCommClkFreq ( uint32_t id )`**

Returns

Frequency of LPFlexComm Clock

**4.5.27 `uint32_t CLOCK_GetSctClkFreq ( void )`**

Returns

Frequency of SCTimer Clock.

**4.5.28 `uint32_t CLOCK_GetTsiClkFreq ( void )`**

Returns

Frequency of TSI Clock.

**4.5.29 `uint32_t CLOCK_GetSincFilterClkFreq ( void )`**

Returns

Frequency of SINC FILTER Clock.

**4.5.30 `uint32_t CLOCK_GetDacClkFreq ( uint32_t id )`**

Returns

Frequency of DAC Clock

**4.5.31 `uint32_t CLOCK_GetFlexspiClkFreq ( void )`**

Returns

Frequency of FlexSPI Clock

**4.5.32 uint32\_t CLOCK\_GetPII0OutFreq ( void )**

Returns

Frequency of PLL

**4.5.33 uint32\_t CLOCK\_GetPII1OutFreq ( void )**

Returns

Frequency of PLL

**4.5.34 uint32\_t CLOCK\_GetPIIClkDivFreq ( void )**

Returns

Frequency of PLLCLKDIV Clock

**4.5.35 uint32\_t CLOCK\_GetI3cClkFreq ( uint32\_t *id* )**

Returns

Frequency of I3C function Clock

**4.5.36 uint32\_t CLOCK\_GetI3cSTCClkFreq ( uint32\_t *id* )**

Returns

Frequency of I3C function slow TC Clock

**4.5.37 uint32\_t CLOCK\_GetI3cSClkFreq ( uint32\_t *id* )**

Returns

Frequency of I3C function slow Clock

**4.5.38 uint32\_t CLOCK\_GetMicfilClkFreq ( void )**

Returns

Frequency of MICFIL.

**4.5.39 uint32\_t CLOCK\_GetUsdhcClkFreq ( void )**

Returns

Frequency of uSDHC Clock

**4.5.40 uint32\_t CLOCK\_GetFlexioClkFreq ( void )**

Returns

Frequency of FLEXIO Clock

**4.5.41 uint32\_t CLOCK\_GetFlexcanClkFreq ( uint32\_t id )**

Returns

Frequency of FLEXCAN Clock

**4.5.42 uint32\_t CLOCK\_GetEnetRmiiClkFreq ( void )**

Returns

Frequency of Ethernet RMII.

**4.5.43 uint32\_t CLOCK\_GetEnetPtpRefClkFreq ( void )**

Returns

Frequency of Ethernet PTP REF.

**4.5.44 void CLOCK\_SetupEnetTxClk ( uint32\_t iFreq )**

Parameters

<i>iFreq</i>	: Desired frequency
--------------	---------------------

Returns

Nothing

#### **4.5.45 uint32\_t CLOCK\_GetEnetTxClkFreq ( void )**

Returns

Frequency of ENET TX CLK

#### **4.5.46 uint32\_t CLOCK\_GetEwm0ClkFreq ( void )**

Returns

Frequency of EWM0.

#### **4.5.47 uint32\_t CLOCK\_GetWdtClkFreq ( uint32\_t id )**

Returns

Frequency of Watchdog

#### **4.5.48 uint32\_t CLOCK\_GetOstimerClkFreq ( void )**

Returns

Frequency of OSTIMER Clock

#### **4.5.49 uint32\_t CLOCK\_GetCmpFClkFreq ( uint32\_t id )**

Returns

Frequency of CMP Function.

**4.5.50 `uint32_t CLOCK_GetCmpRRClkFreq ( uint32_t id )`**

Returns

Frequency of CMP Round Robin.

**4.5.51 `uint32_t CLOCK_GetSaiClkFreq ( uint32_t id )`**

Returns

Frequency of SAI Clock.

**4.5.52 `void CLOCK_SetupSaiMclk ( uint32_t id, uint32_t iFreq )`**

Parameters

<i>iFreq</i>	: Desired frequency
--------------	---------------------

Returns

Nothing

**4.5.53 `void CLOCK_SetupSaiTxBclk ( uint32_t id, uint32_t iFreq )`**

Parameters

<i>iFreq</i>	: Desired frequency
--------------	---------------------

Returns

Nothing

**4.5.54 `void CLOCK_SetupSaiRxBclk ( uint32_t id, uint32_t iFreq )`**

Parameters

<i>iFreq</i>	: Desired frequency
--------------	---------------------

Returns

Nothing

#### 4.5.55 **uint32\_t CLOCK\_GetSaiMclkFreq ( uint32\_t id )**

Returns

Frequency of SAI MCLK

#### 4.5.56 **uint32\_t CLOCK\_GetSaiTxBclkFreq ( uint32\_t id )**

Returns

Frequency of SAI TX BCLK

#### 4.5.57 **uint32\_t CLOCK\_GetSaiRxBclkFreq ( uint32\_t id )**

Returns

Frequency of SAI RX BCLK

#### 4.5.58 **uint32\_t CLOCK\_GetEmvsimClkFreq ( uint32\_t id )**

Returns

Frequency of EMVSIM Clock.

#### 4.5.59 **uint32\_t CLOCK\_GetPLL0InClockRate ( void )**

Returns

PLL0 input clock rate

**4.5.60 uint32\_t CLOCK\_GetPLL1InClockRate ( void )**

Returns

PLL1 input clock rate

**4.5.61 uint32\_t CLOCK\_GetExtUpllFreq ( void )**

Returns

The frequency of the external UPLL.

**4.5.62 void CLOCK\_SetExtUpllFreq ( uint32\_t freq )**

Parameters

<i>The</i>	frequency of external UPLL.
------------	-----------------------------

**4.5.63 \_\_STATIC\_INLINE bool CLOCK\_IsPLL0Locked ( void )**

Returns

true if the PLL is locked, false if not locked

**4.5.64 \_\_STATIC\_INLINE bool CLOCK\_IsPLL1Locked ( void )**

Returns

true if the PLL1 is locked, false if not locked

**4.5.65 uint32\_t CLOCK\_GetPLLOutFromSetup ( pll\_setup\_t \* pSetup )**

Parameters

<i>pSetup</i>	: Pointer to a PLL setup structure
---------------	------------------------------------

Returns

System PLL output clock rate the setup structure will generate

#### 4.5.66 **pll\_error\_t CLOCK\_SetupPLLData ( *pll\_config\_t \* pControl*, *pll\_setup\_t \* pSetup* )**

Parameters

<i>pControl</i>	: Pointer to populated PLL control structure to generate setup with
<i>pSetup</i>	: Pointer to PLL setup structure to be filled

Returns

PLL\_ERROR\_SUCCESS on success, or PLL setup error code

Note

Actual frequency for setup may vary from the desired frequency based on the accuracy of input clocks, rounding, non-fractional PLL mode, etc.

#### 4.5.67 **pll\_error\_t CLOCK\_SetPLL0Freq ( *const pll\_setup\_t \* pSetup* )**

Parameters

<i>pSetup</i>	: Pointer to populated PLL setup structure
---------------	--

Returns

kStatus\_PLL\_Success on success, or PLL setup error code

Note

This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

4.5.68 **pll\_error\_t CLOCK\_SetPLL1Freq ( const pll\_setup\_t \* *pSetup* )**

Parameters

<i>pSetup</i>	: Pointer to populated PLL setup structure
---------------	--

Returns

kStatus\_PLL\_Success on success, or PLL setup error code

Note

This function will power off the PLL, setup the PLL with the new setup data, and then optionally powerup the PLL, wait for PLL lock, and adjust system voltages to the new PLL rate. The function will not alter any source clocks (ie, main system clock) that may use the PLL, so these should be setup prior to and after exiting the function.

#### 4.5.69 void CLOCK\_EnableOstimer32kClock ( void )

Returns

Nothing

#### 4.5.70 bool CLOCK\_EnableUsbfsClock ( void )

Enable USB Full Speed clock.

#### 4.5.71 bool CLOCK\_EnableUsbhsPhyPllClock ( clock\_usb\_phy\_src\_t *src*, uint32\_t *freq* )

This function enables the internal 480MHz USB PHY PLL clock.

param src USB HS PHY PLL clock source. param freq The frequency specified by src. retval true The clock is set successfully. retval false The clock source is invalid to get proper USB HS clock.

#### 4.5.72 void CLOCK\_DisableUsbhsPhyPllClock ( void )

This function disables USB HS PHY PLL clock.

#### 4.5.73 bool CLOCK\_EnableUsbhsClock ( void )

retval true The clock is set successfully. retval false The clock source is invalid to get proper USB HS clock.

# Chapter 5

## Power Driver

### 5.1 Overview

Power driver provides APIs to control peripherals power and control the system power mode.

### Functions

- void [POWER\\_EnterSleep](#) (void)  
*Configures and enters in SLEEP low power mode.*
- void [POWER\\_EnterDeepSleep](#) (uint32\_t exclude\_from\_pd, uint32\_t sram\_retention\_ctrl, uint64\_t wakeup\_interrupts, uint32\_t hardware\_wake\_ctrl)  
*Configures and enters in DEEP-SLEEP low power mode.*
- void [POWER\\_EnterPowerDown](#) (uint32\_t exclude\_from\_pd, uint32\_t sram\_retention\_ctrl, uint64\_t wakeup\_interrupts, uint32\_t cpu\_retention\_ctrl)  
*Configures and enters in POWERDOWN low power mode.*
- void [POWER\\_EnterDeepPowerDown](#) (uint32\_t exclude\_from\_pd, uint32\_t sram\_retention\_ctrl, uint64\_t wakeup\_interrupts, uint32\_t wakeup\_io\_ctrl)  
*Configures and enters in DEEPPowerDOWN low power mode.*
- void [POWER\\_SetVoltageForFreq](#) (uint32\_t system\_freq\_hz)  
*Power Library API to choose normal regulation and set the voltage for the desired operating frequency.*

### Driver version

- #define [FSL\\_POWER\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(1, 0, 0))  
*power driver version 1.0.0.*

### 5.2 Macro Definition Documentation

#### 5.2.1 #define [FSL\\_POWER\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(1, 0, 0))

### 5.3 Function Documentation

#### 5.3.1 void [POWER\\_EnterSleep](#) ( void )

Parameters

---

:

Returns

Nothing

**5.3.2 void POWER\_EnterDeepSleep ( *uint32\_t exclude\_from\_pd,*  
*uint32\_t sram\_retention\_ctrl, uint64\_t wakeup\_interrupts, uint32\_t*  
*hardware\_wake\_ctrl* )**

Parameters

<i>exclude_from_pd,: </i>	
<i>sram_retention_ctrl,: </i>	
<i>wakeup_interrupts,: </i>	
<i>hardware_wake_ctrl,: </i>	

Returns

Nothing

!!! IMPORTANT NOTES :

0 - CPU0 & System CLock frequency is switched to FRO12MHz and is NOT restored back by the API.  
 1 - CPU0 Interrupt Enable registers (NVIC->ISER) are modified by this function. They are restored back in case of CPU retention or if POWERDOWN is not taken (for instance because an interrupt is pending).  
 2 - The Non Maskable Interrupt (NMI) is disabled and its configuration before calling this function will be restored back if POWERDOWN is not taken (for instance because an RTC or OSTIMER interrupt is pending).  
 3 - The HARD FAULT handler should execute from SRAM. (The Hard fault handler should initiate a full chip reset) reset)

**5.3.3 void POWER\_EnterPowerDown ( *uint32\_t exclude\_from\_pd, uint32\_t*  
*sram\_retention\_ctrl, uint64\_t wakeup\_interrupts, uint32\_t cpu\_retention\_ctrl*  
 $)$**

Parameters

<i>exclude_from_pd,:</i>	
<i>sram_retention_ctrl,:</i>	
<i>wakeup_interrupts,:</i>	
<i>cpu_retention_ctrl,:</i>	0 = CPU retention is disable / 1 = CPU retention is enabled, all other values are RESERVED.

Returns

Nothing

!!! IMPORTANT NOTES :

0 - CPU0 & System CLock frequency is switched to FRO12MHz and is NOT restored back by the API.  
 1 - CPU0 Interrupt Enable registers (NVIC->ISER) are modified by this function. They are restored back in case of CPU retention or if POWERDOWN is not taken (for instance because an interrupt is pending).  
 2 - The Non Maskable Interrupt (NMI) is disabled and its configuration before calling this function will be restored back if POWERDOWN is not taken (for instance because an RTC or OSTIMER interrupt is pending).  
 3 - In case of CPU retention, it is the responsibility of the user to make sure that SRAM instance containing the stack used to call this function WILL BE preserved during low power (via parameter "sram\_retention\_ctrl")  
 4 - The HARD FAULT handler should execute from SRAM. (The Hard fault handler should initiate a full chip reset)

### 5.3.4 void POWER\_EnterDeepPowerDown ( *uint32\_t exclude\_from\_pd, uint32\_t sram\_retention\_ctrl, uint64\_t wakeup\_interrupts, uint32\_t wakeup\_io\_ctrl* )

Parameters

<i>exclude_from_pd,:</i>	
<i>sram_retention_ctrl,:</i>	

<i>wakeup_- interrupts,:</i>	
<i>wakeup_io_- ctrl,:</i>	

Returns

Nothing

!!! IMPORTANT NOTES :

0 - CPU0 & System CLock frequency is switched to FRO12MHz and is NOT restored back by the API. 1 - CPU0 Interrupt Enable registers (NVIC->ISER) are modified by this function. They are restored back if DEEPPOWERDOWN is not taken (for instance because an RTC or OSTIMER interrupt is pending). 2 - The Non Maskable Interrupt (NMI) is disabled and its configuration before calling this function will be restored back if DEEPPOWERDOWN is not taken (for instance because an RTC or OSTIMER interrupt is pending). 3 - The HARD FAULT handler should execute from SRAM. (The Hard fault handler should initiate a full chip reset)

### 5.3.5 void POWER\_SetVoltageForFreq ( uint32\_t system\_freq\_hz )

Parameters

<i>system_freq_hz</i>	- The desired frequency (in Hertz) at which the part would like to operate, note that the voltage and flash wait states should be set before changing frequency
-----------------------	---

Returns

none

# Chapter 6

## Reset Driver

### 6.1 Overview

Reset driver supports peripheral reset and system reset.

#### Macros

- #define [ADC\\_RSTS](#)

## Enumerations

- enum `SYSCON_RSTn_t` {
   
`kFMU_RST_SHIFT_RSTn` = 0 | 9U,
   
`kFLEXSPI_RST_SHIFT_RSTn` = 0 | 11U,
   
`kMUX_RST_SHIFT_RSTn` = 0 | 12U,
   
`kPORT0_RST_SHIFT_RSTn` = 0 | 13U,
   
`kPORT1_RST_SHIFT_RSTn` = 0 | 14U,
   
`kPORT2_RST_SHIFT_RSTn` = 0 | 15U,
   
`kPORT3_RST_SHIFT_RSTn` = 0 | 16U,
   
`kPORT4_RST_SHIFT_RSTn` = 0 | 17U,
   
`kGPIO0_RST_SHIFT_RSTn` = 0 | 19U,
   
`kGPIO1_RST_SHIFT_RSTn` = 0 | 20U,
   
`kGPIO2_RST_SHIFT_RSTn` = 0 | 21U,
   
`kGPIO3_RST_SHIFT_RSTn` = 0 | 22U,
   
`kGPIO4_RST_SHIFT_RSTn` = 0 | 23U,
   
`kPINT_RST_SHIFT_RSTn` = 0 | 25U,
   
`kDMA0_RST_SHIFT_RSTn` = 0 | 26U,
   
`kCRC_RST_SHIFT_RSTn` = 0 | 27U,
   
`kMAILBOX_RST_SHIFT_RSTn` = 0 | 31U,
   
`kMRT_RST_SHIFT_RSTn` = 65536 | 0U,
   
`kOSTIMER_RST_SHIFT_RSTn` = 65536 | 1U,
   
`kSCT_RST_SHIFT_RSTn` = 65536 | 2U,
   
`kADC0_RST_SHIFT_RSTn` = 65536 | 3U,
   
`kADC1_RST_SHIFT_RSTn` = 65536 | 4U,
   
`kDAC0_RST_SHIFT_RSTn` = 65536 | 5U,
   
`kEVSIM0_RST_SHIFT_RSTn` = 65536 | 8U,
   
`kEVSIM1_RST_SHIFT_RSTn` = 65536 | 9U,
   
`kUTICK_RST_SHIFT_RSTn` = 65536 | 10U,
   
`kFC0_RST_SHIFT_RSTn` = 65536 | 11U,
   
`kFC1_RST_SHIFT_RSTn` = 65536 | 12U,
   
`kFC2_RST_SHIFT_RSTn` = 65536 | 13U,
   
`kFC3_RST_SHIFT_RSTn` = 65536 | 14U,
   
`kFC4_RST_SHIFT_RSTn` = 65536 | 15U,
   
`kFC5_RST_SHIFT_RSTn` = 65536 | 16U,
   
`kFC6_RST_SHIFT_RSTn` = 65536 | 17U,
   
`kFC7_RST_SHIFT_RSTn` = 65536 | 18U,
   
`kFC8_RST_SHIFT_RSTn` = 65536 | 19U,
   
`kFC9_RST_SHIFT_RSTn` = 65536 | 20U,
   
`kMICFIL_RST_SHIFT_RSTn` = 65536 | 21U,
   
`kCTIMER2_RST_SHIFT_RSTn` = 65536 | 22U,
   
`kUSB0_RAM_RST_SHIFT_RSTn` = 65536 | 23U,
   
`kUSB0_FS_DCD_RST_SHIFT_RSTn` = 65536 | 24U,
   
`kUSB0_FS_RST_SHIFT_RSTn` = 65536 | 25U,
   
`kCTIMER0_RST_SHIFT_RSTn` = 65536 | 26U,
   
`kCTIMER1_RST_SHIFT_RSTn` = 65536 | 27U,
   
`kSMARTDMA_RST_SHIFT_RSTn` = 65536 | 31U,
   
`kDMA1_RST_SHIFT_RSTn` = 131072 | 2U,
   
`kUSDHC_RST_SHIFT_RSTn` = 131072 | 3U,

`kSEMA42_RST_SHIFT_RSTn = 196608 | 27U }`

*Enumeration for peripheral reset control bits.*

## Functions

- void `RESET_SetPeripheralReset (reset_ip_name_t peripheral)`  
*Assert reset to peripheral.*
- void `RESET_ClearPeripheralReset (reset_ip_name_t peripheral)`  
*Clear reset to peripheral.*
- void `RESET_PeripheralReset (reset_ip_name_t peripheral)`  
*Reset peripheral module.*

## Driver version

- `#define FSL_RESET_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))`  
*reset driver version 1.0.0.*

## 6.2 Macro Definition Documentation

### 6.2.1 `#define FSL_RESET_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))`

### 6.2.2 `#define ADC_RSTS`

#### Value:

```
{
    kADC0_RST_SHIFT_RSTn, kADC1_RST_SHIFT_RSTn \
} /* Reset bits for ADC peripheral */
```

Array initializers with peripheral reset bits

## 6.3 Enumeration Type Documentation

### 6.3.1 `enum SYSCON_RSTn_t`

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

Enumerator

`kFMU_RST_SHIFT_RSTn` Flash management unit reset control  
`kFLEXSPI_RST_SHIFT_RSTn` FLEXSPI reset control  
`kMUX_RST_SHIFT_RSTn` Input mux reset control  
`kPORT0_RST_SHIFT_RSTn` PORT0 reset control  
`kPORT1_RST_SHIFT_RSTn` PORT1 reset control  
`kPORT2_RST_SHIFT_RSTn` PORT2 reset control  
`kPORT3_RST_SHIFT_RSTn` PORT3 reset control

*kPORT4\_RST\_SHIFT\_RSTn* PORT4 reset control  
*kGPIO0\_RST\_SHIFT\_RSTn* GPIO0 reset control  
*kGPIO1\_RST\_SHIFT\_RSTn* GPIO1 reset control  
*kGPIO2\_RST\_SHIFT\_RSTn* GPIO2 reset control  
*kGPIO3\_RST\_SHIFT\_RSTn* GPIO3 reset control  
*kGPIO4\_RST\_SHIFT\_RSTn* GPIO4 reset control  
*kPINT\_RST\_SHIFT\_RSTn* Pin interrupt (PINT) reset control  
*kDMA0\_RST\_SHIFT\_RSTn* DMA0 reset control  
*kCRC\_RST\_SHIFT\_RSTn* CRC reset control  
*kMAILBOX\_RST\_SHIFT\_RSTn* Mailbox reset control  
*kmRt\_RST\_SHIFT\_RSTn* Multi-rate timer (MRT) reset control  
*kOSTIMER\_RST\_SHIFT\_RSTn* OSTimer reset control  
*ksCT\_RST\_SHIFT\_RSTn* SCTimer/PWM(SCT) reset control  
*kADC0\_RST\_SHIFT\_RSTn* ADC0 reset control  
*kADC1\_RST\_SHIFT\_RSTn* ADC1 reset control  
*kDAC0\_RST\_SHIFT\_RSTn* DAC0 reset control  
*keVSIM0\_RST\_SHIFT\_RSTn* EVSIM0 reset control  
*keVSIM1\_RST\_SHIFT\_RSTn* EVSIM1 reset control  
*kUTICK\_RST\_SHIFT\_RSTn* Micro-tick timer reset control  
*kFC0\_RST\_SHIFT\_RSTn* Flexcomm Interface 0 reset control  
*kFC1\_RST\_SHIFT\_RSTn* Flexcomm Interface 1 reset control  
*kFC2\_RST\_SHIFT\_RSTn* Flexcomm Interface 2 reset control  
*kFC3\_RST\_SHIFT\_RSTn* Flexcomm Interface 3 reset control  
*kFC4\_RST\_SHIFT\_RSTn* Flexcomm Interface 4 reset control  
*kFC5\_RST\_SHIFT\_RSTn* Flexcomm Interface 5 reset control  
*kFC6\_RST\_SHIFT\_RSTn* Flexcomm Interface 6 reset control  
*kFC7\_RST\_SHIFT\_RSTn* Flexcomm Interface 7 reset control  
*kFC8\_RST\_SHIFT\_RSTn* Flexcomm Interface 8 reset control  
*kFC9\_RST\_SHIFT\_RSTn* MICFIL reset control  
*kmICFIL\_RST\_SHIFT\_RSTn* Flexcomm Interface 7 reset control  
*kCTIMER2\_RST\_SHIFT\_RSTn* CTimer 2 reset control  
*kUSB0\_RAM\_RST\_SHIFT\_RSTn* USB0 RAM reset control  
*kUSB0\_FS\_DCD\_RST\_SHIFT\_RSTn* USB0-FS DCD reset control  
*kUSB0\_FS\_RST\_SHIFT\_RSTn* USB0-FS reset control  
*kCTIMER0\_RST\_SHIFT\_RSTn* CTimer 0 reset control  
*kCTIMER1\_RST\_SHIFT\_RSTn* CTimer 1 reset control  
*kSMARTDMA\_RST\_SHIFT\_RSTn* SmartDMA reset control  
*kDMA1\_RST\_SHIFT\_RSTn* DMA1 reset control  
*kENET\_RST\_SHIFT\_RSTn* Ethernet reset control  
*kUSDHC\_RST\_SHIFT\_RSTn* uSDHC reset control  
*kFLEXIO\_RST\_SHIFT\_RSTn* FLEXIO reset control  
*kSAI0\_RST\_SHIFT\_RSTn* SAI0 reset control  
*kSAI1\_RST\_SHIFT\_RSTn* SAI1 reset control  
*kTRO\_RST\_SHIFT\_RSTn* TRO reset control  
*kFREQME\_RST\_SHIFT\_RSTn* FREQME reset control

*kTRNG\_RST\_SHIFT\_RSTn* TRNG reset control  
*kFLEXCAN0\_RST\_SHIFT\_RSTn* Flexcan0 reset control  
*kFLEXCAN1\_RST\_SHIFT\_RSTn* Flexcan1 reset control  
*kUSB\_HS\_RST\_SHIFT\_RSTn* USB HS reset control  
*kUSB\_HS\_PHY\_RST\_SHIFT\_RSTn* USB HS PHY reset control  
*kPOWERQUAD\_RST\_SHIFT\_RSTn* PowerQuad reset control  
*kPLU\_RST\_SHIFT\_RSTn* PLU reset control  
*kCTIMER3\_RST\_SHIFT\_RSTn* CTimer 3 reset control  
*kCTIMER4\_RST\_SHIFT\_RSTn* CTimer 4 reset control  
*kPUF\_RST\_SHIFT\_RSTn* PUF reset control  
*kPKC\_RST\_SHIFT\_RSTn* PKC reset control  
*kSM3\_RST\_SHIFT\_RSTn* SM3 reset control  
*kI3C0\_RST\_SHIFT\_RSTn* I3C0 reset control  
*kI3C1\_RST\_SHIFT\_RSTn* I3C1 reset control  
*ksINC\_RST\_SHIFT\_RSTn* SINC reset control  
*kCOOLFLUX\_RST\_SHIFT\_RSTn* CoolFlux reset control  
*kENC0\_RST\_SHIFT\_RSTn* ENC0 reset control  
*kENC1\_RST\_SHIFT\_RSTn* ENC1 reset control  
*kPWM0\_RST\_SHIFT\_RSTn* PWM0 reset control  
*kPWM1\_RST\_SHIFT\_RSTn* PWM1 reset control  
*kAOI0\_RST\_SHIFT\_RSTn* AOI0 reset control  
*kDAC1\_RST\_SHIFT\_RSTn* DAC1 reset control  
*kDAC2\_RST\_SHIFT\_RSTn* DAC2 reset control  
*kOPAMP0\_RST\_SHIFT\_RSTn* OPAMP0 reset control  
*kOPAMP1\_RST\_SHIFT\_RSTn* OPAMP1 reset control  
*kOPAMP2\_RST\_SHIFT\_RSTn* OPAMP2 reset control  
*kCMP2\_RST\_SHIFT\_RSTn* CMP2 reset control  
*kvREF\_RST\_SHIFT\_RSTn* VREF reset control  
*kCOOLFLUX\_APB\_RST\_SHIFT\_RSTn* CoolFlux APB reset control  
*kNEUTRON\_RST\_SHIFT\_RSTn* Neutron mini reset control  
*kTSI\_RST\_SHIFT\_RSTn* TSI reset control  
*KEWM\_RST\_SHIFT\_RSTn* EWM reset control  
*KEIM\_RST\_SHIFT\_RSTn* EIM reset control  
*kSEMA42\_RST\_SHIFT\_RSTn* Semaphore reset control

## 6.4 Function Documentation

### 6.4.1 void RESET\_SetPeripheralReset ( *reset\_ip\_name\_t peripheral* )

Asserts reset signal to specified peripheral module.

Parameters

<i>peripheral</i>	Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.
-------------------	--

#### 6.4.2 void RESET\_ClearPeripheralReset ( *reset\_ip\_name\_t peripheral* )

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

<i>peripheral</i>	Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.
-------------------	---

#### 6.4.3 void RESET\_PeripheralReset ( *reset\_ip\_name\_t peripheral* )

Reset peripheral module.

Parameters

<i>peripheral</i>	Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.
-------------------	--

# Chapter 7

## ANACTRL: Analog Control Driver

### 7.1 ANACTRL function groups

#### 7.2 Overview

#### 7.3 Function groups

The ANACTRL driver supports initialization/configuration/operation for optimization/customization purpose.

##### 7.3.1 Initialization and deinitialization

This function group is to enable/disable the clock for the ANACTRL module.

##### 7.3.2 Set oscillators

The function `ANACTRL_SetFro192M` sets the on-chip high-speed Free Running Oscillator. The function `ANACTRL_GetDefaultFro192MConfig()` gets the default configuration.

The function `ANACTRL_SetXo32M` sets the 32 MHz Crystal oscillator. The function `ANACTRL_GetDefaultXo32MConfig()` gets the default configuration.

##### 7.3.3 Measure Frequency

This function measures the target frequency according to the reference frequency.

##### 7.3.4 Interrupt

Provides functions to enable/disable/clear ANACTRL interrupts.

##### 7.3.5 Status

Provides functions to get the ANACTRL status.

## Data Structures

- struct `anactrl_fro192M_config_t`

- Configuration for FRO192M. [More...](#)
- struct `anactrl_xo32M_config_t`  
Configuration for XO32M. [More...](#)

## Macros

- #define `FSL_ANACTRL_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 0)`) /\*!< Version 2.3.0.  
\*/  
*ANACTRL driver version.*

## Enumerations

- enum `_anactrl_interrupt_flags` {
   
`kANACTRL_BodVbatFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_BODVBAT_STATUS_MASK`,  
`kANACTRL_BodVbatInterruptFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_BODVBAT_INT_STATUS_MASK`,  
`kANACTRL_BodVbatPowerFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_BODVBAT_VAL_MASK`,  
`kANACTRL_BodCoreFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_BODCORE_STATUS_MASK`,  
`kANACTRL_BodCoreInterruptFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_BODCORE_INT_STATUS_MASK`,  
`kANACTRL_BodCorePowerFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_BODCORE_VAL_MASK`,  
`kANACTRL_DcdcFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_DCDC_STATUS_MASK`,  
`kANACTRL_DcdcInterruptFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_DCDC_INT_STATUS_MASK`,  
`kANACTRL_DcdcPowerFlag` = `ANACTRL_BOD_DCDC_INT_STATUS_DCDC_VAL_MASK`
  
 }
   
*ANACTRL interrupt flags.*
- enum `_anactrl_interrupt` {
   
`kANACTRL_BodVbatInterruptEnable` = `ANACTRL_BOD_DCDC_INT_CTRL_BODVBAT_INT_ENABLE_MASK`,  
`kANACTRL_BodCoreInterruptEnable` = `ANACTRL_BOD_DCDC_INT_CTRL_BODCORE_INT_ENABLE_MASK`,  
`kANACTRL_DcdcInterruptEnable` = `ANACTRL_BOD_DCDC_INT_CTRL_DCDC_INT_ENABLE_MASK`
  
 }
   
*ANACTRL interrupt control.*
- enum `_anactrl_flags` {
   
`kANACTRL_FlashPowerDownFlag` = `ANACTRL_ANALOG_CTRL_STATUS_FLASH_PWRDWN_MASK`,  
`kANACTRL_FlashInitErrorFlag` = `ANACTRL_ANALOG_CTRL_STATUS_FLASH_INIT_ERROR_MASK`
  
 }
   
*ANACTRL status flags.*
- enum `_anactrl_osc_flags` {

`kANACTRL_OutputClkValidFlag` = ANACTRL\_FRO192M\_STATUS\_CLK\_VALID\_MASK,  
`kANACTRL_CCOThresholdVoltageFlag` = ANACTRL\_FRO192M\_STATUS\_ATB\_VCTRL\_M-ASK,  
`kANACTRL_XO32MOutputReadyFlag` = ANACTRL\_XO32M\_STATUS\_XO\_READY\_MASK  
`<< 16U }`

*ANACTRL FRO192M and XO32M status flags.*

## Initialization and deinitialization

- void `ANACTRL_Init` (ANACTRL\_Type \*base)  
*Initializes the ANACTRL mode, the module's clock will be enabled by invoking this function.*
- void `ANACTRL_Deinit` (ANACTRL\_Type \*base)  
*De-initializes ANACTRL module, the module's clock will be disabled by invoking this function.*

## Set oscillators

- void `ANACTRL_SetFro192M` (ANACTRL\_Type \*base, const `anactrl_fro192M_config_t` \*config)  
*Configs the on-chip high-speed Free Running Oscillator(FRO192M), such as enabling/disabling 12 MHZ clock output and enable/disable 96MHZ clock output.*
- void `ANACTRL_GetDefaultFro192MConfig` (`anactrl_fro192M_config_t` \*config)  
*Gets the default configuration of FRO192M.*
- void `ANACTRL_SetXo32M` (ANACTRL\_Type \*base, const `anactrl_xo32M_config_t` \*config)  
*Configs the 32 MHz Crystal oscillator(High-speed crystal oscillator), such as enable/disable output to CPU system, and so on.*
- void `ANACTRL_GetDefaultXo32MConfig` (`anactrl_xo32M_config_t` \*config)  
*Gets the default configuration of XO32M.*

## Measure Frequency

- uint32\_t `ANACTRL_MeasureFrequency` (ANACTRL\_Type \*base, uint8\_t scale, uint32\_t refClk-Freq)  
*Measures the frequency of the target clock source.*

## Interrupt Interface

- static void `ANACTRL_EnableInterrupts` (ANACTRL\_Type \*base, uint32\_t mask)  
*Enables the ANACTRL interrupts.*
- static void `ANACTRL_DisableInterrupts` (ANACTRL\_Type \*base, uint32\_t mask)  
*Disables the ANACTRL interrupts.*
- static void `ANACTRL_ClearInterrupts` (ANACTRL\_Type \*base, uint32\_t mask)  
*Clears the ANACTRL interrupts.*

## Status Interface

- static uint32\_t `ANACTRL_GetStatusFlags` (ANACTRL\_Type \*base)  
*Gets ANACTRL status flags.*
- static uint32\_t `ANACTRL_GetOscStatusFlags` (ANACTRL\_Type \*base)  
*Gets ANACTRL oscillators status flags.*
- static uint32\_t `ANACTRL_GetInterruptStatusFlags` (ANACTRL\_Type \*base)  
*Gets ANACTRL interrupt status flags.*

## 7.4 Data Structure Documentation

### 7.4.1 struct anactrl\_fro192M\_config\_t

This structure holds the configuration settings for the on-chip high-speed Free Running Oscillator. To initialize this structure to reasonable defaults, call the [ANACTRL\\_GetDefaultFro192MConfig\(\)](#) function and pass a pointer to your config structure instance.

#### Data Fields

- bool [enable12MHzClk](#)  
*Enable 12MHz clock.*
- bool [enable96MHzClk](#)  
*Enable 96MHz clock.*

#### Field Documentation

- (1) [bool anactrl\\_fro192M\\_config\\_t::enable12MHzClk](#)
- (2) [bool anactrl\\_fro192M\\_config\\_t::enable96MHzClk](#)

### 7.4.2 struct anactrl\_xo32M\_config\_t

This structure holds the configuration settings for the 32 MHz crystal oscillator. To initialize this structure to reasonable defaults, call the [ANACTRL\\_GetDefaultXo32MConfig\(\)](#) function and pass a pointer to your config structure instance.

#### Data Fields

- bool [enableACBufferBypass](#)  
*Enable XO AC buffer bypass in pll and top level.*
- bool [enablePllUsbOutput](#)  
*Enable XO 32 MHz output to USB HS PLL.*
- bool [enableSysClkOutput](#)  
*Enable XO 32 MHz output to CPU system, SCT, and CLKOUT.*

#### Field Documentation

- (1) [bool anactrl\\_xo32M\\_config\\_t::enableACBufferBypass](#)
- (2) [bool anactrl\\_xo32M\\_config\\_t::enablePllUsbOutput](#)

## 7.5 Macro Definition Documentation

### 7.5.1 #define FSL\_ANACTRL\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 0)) /\*!< Version 2.3.0. \*/^

## 7.6 Enumeration Type Documentation

### 7.6.1 enum \_anactrl\_interrupt\_flags

Enumerator

*kANACTRL\_BodVbatFlag* BOD VBAT Interrupt status before Interrupt Enable.  
*kANACTRL\_BodVbatInterruptFlag* BOD VBAT Interrupt status after Interrupt Enable.  
*kANACTRL\_BodVbatPowerFlag* Current value of BOD VBAT power status output.  
*kANACTRL\_BodCoreFlag* BOD CORE Interrupt status before Interrupt Enable.  
*kANACTRL\_BodCoreInterruptFlag* BOD CORE Interrupt status after Interrupt Enable.  
*kANACTRL\_BodCorePowerFlag* Current value of BOD CORE power status output.  
*kANACTRL\_DcdcFlag* DCDC Interrupt status before Interrupt Enable.  
*kANACTRL\_DcdcInterruptFlag* DCDC Interrupt status after Interrupt Enable.  
*kANACTRL\_DcdcPowerFlag* Current value of DCDC power status output.

### 7.6.2 enum \_anactrl\_interrupt

Enumerator

*kANACTRL\_BodVbatInterruptEnable* BOD VBAT interrupt control.  
*kANACTRL\_BodCoreInterruptEnable* BOD CORE interrupt control.  
*kANACTRL\_DcdcInterruptEnable* DCDC interrupt control.

### 7.6.3 enum \_anactrl\_flags

Enumerator

*kANACTRL\_FlashPowerDownFlag* Flash power-down status.  
*kANACTRL\_FlashInitErrorFlag* Flash initialization error status.

### 7.6.4 enum \_anactrl\_osc\_flags

Enumerator

*kANACTRL\_OutputClkValidFlag* Output clock valid signal.  
*kANACTRL\_CCOThresholdVoltageFlag* CCO threshold voltage detector output (signal vcco\_ok).  
*kANACTRL\_XO32MOutputReadyFlag* Indicates XO out frequency stability.

## 7.7 Function Documentation

### 7.7.1 void ANACTRL\_Init( ANACTRL\_Type \* base )

Parameters

<i>base</i>	ANACTRL peripheral base address.
-------------	----------------------------------

### 7.7.2 void ANACTRL\_Deinit ( ANACTRL\_Type \* *base* )

Parameters

<i>base</i>	ANACTRL peripheral base address.
-------------	----------------------------------

### 7.7.3 void ANACTRL\_SetFro192M ( ANACTRL\_Type \* *base*, const anactrl\_fro192M\_config\_t \* *config* )

Parameters

<i>base</i>	ANACTRL peripheral base address.
<i>config</i>	Pointer to FRO192M configuration structure. Refer to <a href="#">anactrl_fro192M_config_t</a> structure.

### 7.7.4 void ANACTRL\_GetDefaultFro192MConfig ( anactrl\_fro192M\_config\_t \* *config* )

The default values are:

```
config->enable12MHzClk = true;
config->enable96MHzClk = false;
```

Parameters

<i>config</i>	Pointer to FRO192M configuration structure. Refer to <a href="#">anactrl_fro192M_config_t</a> structure.
---------------	--

### 7.7.5 void ANACTRL\_SetXo32M ( ANACTRL\_Type \* *base*, const anactrl\_xo32M\_config\_t \* *config* )

## Parameters

<i>base</i>	ANACTRL peripheral base address.
<i>config</i>	Pointer to XO32M configuration structure. Refer to <a href="#">anactrl_xo32M_config_t</a> structure.

**7.7.6 void ANACTRL\_GetDefaultXo32MConfig ( [anactrl\\_xo32M\\_config\\_t](#) \* *config* )**

The default values are:

```
config->enableSysClkOutput = false;
config->enableACBufferBypass = false;
```

## Parameters

<i>config</i>	Pointer to XO32M configuration structure. Refer to <a href="#">anactrl_xo32M_config_t</a> structure.
---------------	--

**7.7.7 uint32\_t ANACTRL\_MeasureFrequency ( [ANACTRL\\_Type](#) \* *base*, [uint8\\_t](#) *scale*, [uint32\\_t](#) *refClkFreq* )**

This function measures target frequency according to a accurate reference frequency. The formula is:  
 $F_{target} = (\text{CAPVAL} * \text{Preference}) / ((1 << \text{SCALE}) - 1)$

## Note

Both target and reference clocks are selectable by programming the target clock select FREQMEAS\_TARGET register in INPUTMUX and reference clock select FREQMEAS\_REF register in INPUTMUX.

## Parameters

<i>base</i>	ANACTRL peripheral base address.
<i>scale</i>	Define the power of 2 count that ref counter counts to during measurement, ranges from 2 to 31.

<i>refClkFreq</i>	frequency of the reference clock.
-------------------	-----------------------------------

Returns

frequency of the target clock.

### 7.7.8 static void ANACTRL\_EnableInterrupts ( ANACTRL\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	ANACTRL peripheral base address.
<i>mask</i>	The interrupt mask. Refer to "_anactrl_interrupt" enumeration.

### 7.7.9 static void ANACTRL\_DisableInterrupts ( ANACTRL\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	ANACTRL peripheral base address.
<i>mask</i>	The interrupt mask. Refer to "_anactrl_interrupt" enumeration.

### 7.7.10 static void ANACTRL\_ClearInterrupts ( ANACTRL\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	ANACTRL peripheral base address.
<i>mask</i>	The interrupt mask. Refer to "_anactrl_interrupt" enumeration.

### 7.7.11 static uint32\_t ANACTRL\_GetStatusFlags ( ANACTRL\_Type \* *base* ) [inline], [static]

This function gets Analog control status flags. The flags are returned as the logical OR value of the enumerators [\\_anactrl\\_flags](#). To check for a specific status, compare the return value with enumerators in the [\\_anactrl\\_flags](#). For example, to check whether the flash is in power down mode:

```

*     if (kANACTRL_FlashPowerDownFlag & ANACTRL_ANACTRL_GetStatusFlags(ANACTRL))
*     {
*         ...
*     }
*

```

## Parameters

<i>base</i>	ANACTRL peripheral base address.
-------------	----------------------------------

## Returns

ANACTRL status flags which are given in the enumerators in the [\\_anactrl\\_flags](#).

### 7.7.12 static uint32\_t ANACTRL\_GetOscStatusFlags ( ANACTRL\_Type \* *base* ) [inline], [static]

This function gets Anactrl oscillators status flags. The flags are returned as the logical OR value of the enumerators [\\_anactrl\\_osc\\_flags](#). To check for a specific status, compare the return value with enumerators in the [\\_anactrl\\_osc\\_flags](#). For example, to check whether the FRO192M clock output is valid:

```

*     if (kANACTRL_OutputClkValidFlag & ANACTRL_ANACTRL_GetOscStatusFlags(
ANACTRL))
*     {
*         ...
*     }
*

```

## Parameters

<i>base</i>	ANACTRL peripheral base address.
-------------	----------------------------------

## Returns

ANACTRL oscillators status flags which are given in the enumerators in the [\\_anactrl\\_osc\\_flags](#).

### 7.7.13 static uint32\_t ANACTRL\_GetInterruptStatusFlags ( ANACTRL\_Type \* *base* ) [inline], [static]

This function gets Anactrl interrupt status flags. The flags are returned as the logical OR value of the enumerators [\\_anactrl\\_interrupt\\_flags](#). To check for a specific status, compare the return value with enumerators in the [\\_anactrl\\_interrupt\\_flags](#). For example, to check whether the VBAT voltage level is above the threshold:

```
*     if (kANACTRL_BodVbatPowerFlag & ANACTRL_ANACTRL_GetInterruptStatusFlags(
ANACTRL))
*     {
*         ...
*     }
*
```

### Parameters

<i>base</i>	ANACTRL peripheral base address.
-------------	----------------------------------

### Returns

ANACTRL oscillators status flags which are given in the enumerators in the [\\_anactrl\\_osc\\_flags](#).

# Chapter 8

## CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing

### 8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Cryptographic Accelerator and Signal Processing Engine with RAM sharing (CASPER) module of MCUXpresso SDK devices. The CASPER peripheral provides acceleration of asymmetric cryptographic algorithms as well as optionally of certain signal processing algorithms. The cryptographic acceleration is normally used in conjunction with pure-hardware blocks for hashing and symmetric cryptography, thereby providing performance and energy efficiency for a range of cryptographic uses.

Blocking synchronous APIs are provided for selected cryptographic algorithms using CASPER hardware. The driver interface intends to be easily integrated with generic software crypto libraries such as mbedTLS or wolfSSL. The CASPER operations are complete (and results are made available for further usage) when a function returns. When called, these functions do not return until an CASPER operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and also for plaintext or ciphertext data movements. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

### 8.2 CASPER Driver Initialization and deinitialization

CASPER Driver is initialized by calling the `CASPER_Init()` function, it resets the CASPER module and enables its clock. CASPER Driver is deinitialized by calling the `CASPER_Deinit()` function, it disables CASPER module clock.

### 8.3 Comments about API usage in RTOS

CASPER operations provided by this driver are not re-entrant. Thus, application software shall ensure the CASPER module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

### 8.4 Comments about API usage in interrupt handler

All APIs shall not be used from interrupt handler as global variables are used.

### 8.5 CASPER Driver Examples

#### 8.5.1 Simple examples

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/casper/

## Modules

- [casper\\_driver](#)
- [casper\\_driver\\_pkha](#)

## 8.6 casper\_driver

### 8.6.1 Overview

#### Enumerations

- enum `casper_operation_t` {
   
    `kCASPER_OpMul64Sum`,
   
    `kCASPER_OpMul64FullSum`,
   
    `kCASPER_OpMul64Reduce`,
   
    `kCASPER_OpAdd64` = 0x08,
   
    `kCASPER_OpSub64` = 0x09,
   
    `kCASPER_OpDouble64` = 0x0A,
   
    `kCASPER_OpXor64` = 0x0B,
   
    `kCASPER_OpRSub64` = 0x0C,
   
    `kCASPER_OpShiftLeft32`,
   
    `kCASPER_OpShiftRight32` = 0x11,
   
    `kCASPER_OpCopy` = 0x14,
   
    `kCASPER_OpRemask` = 0x15,
   
    `kCASPER_OpFill` = 0x16,
   
    `kCASPER_OpZero` = 0x17,
   
    `kCASPER_OpCompare` = 0x18,
   
    `kCASPER_OpCompareFast` = 0x19
 }

*CASPER operation.*

- enum `casper_algo_t` {
   
    `kCASPER_ECC_P256` = 0x01,
   
    `kCASPER_ECC_P384` = 0x02,
   
    `kCASPER_ECC_P521` = 0x03
 }

*Algorithm used for CASPER operation.*

#### Functions

- void `CASPER_Init` (`CASPER_Type` \*base)
   
*Enables clock and disables reset for CASPER peripheral.*
- void `CASPER_Deinit` (`CASPER_Type` \*base)
   
*Disables clock for CASPER peripheral.*

#### Driver version

- #define `FSL_CASPER_DRIVER_VERSION` (`MAKE_VERSION`(2, 2, 3))
   
*CASPER driver version.*

### 8.6.2 Macro Definition Documentation

### 8.6.2.1 #define FSL\_CASPER\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 3))

Version 2.2.3.

Current version: 2.2.3

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Bug fix KPSDK-24531 double\_scalar\_multiplication() result may be all zeroes for some specific input
- Version 2.0.2
  - Bug fix KPSDK-25015 CASPER\_MEMCPY hard-fault on LPC55xx when both source and destination buffers are outside of CASPER\_RAM
- Version 2.0.3
  - Bug fix KPSDK-28107 RSUB, FILL and ZERO operations not implemented in enum\_casper\_operation.
- Version 2.0.4
  - For GCC compiler, enforce O1 optimize level, specifically to remove strict-aliasing option.  
This driver is very specific and requires -fno-strict-aliasing.
- Version 2.0.5
  - Fix sign-compare warning.
- Version 2.0.6
  - Fix IAR Pa082 warning.
- Version 2.0.7
  - Fix MISRA-C 2012 issue.
- Version 2.0.8
  - Add feature macro for CASPER\_RAM\_OFFSET.
- Version 2.0.9
  - Remove unused function Jac\_oncurve().
  - Fix ECC384 build.
- Version 2.0.10
  - Fix MISRA-C 2012 issue.
- Version 2.1.0
  - Add ECC NIST P-521 elliptic curve.
- Version 2.2.0
  - Rework driver to support multiple curves at once.
- Version 2.2.1
  - Fix MISRA-C 2012 issue.
- Version 2.2.2
  - Enable hardware interleaving to RAMX0 and RAMX1 for CASPER by feature macro FSL\_FEATURE\_CASPER\_RAM\_HW\_INTERLEAVE
- Version 2.2.3
  - Added macro into CASPER\_Init and CASPER\_Deinit to support devices without clock and

reset control.

### 8.6.3 Enumeration Type Documentation

#### 8.6.3.1 enum casper\_operation\_t

Enumerator

*kCASPER\_OpMul64x64Sum* Walking 1 or more of J loop, doing  $r=a*b$  using  $64x64=128$ .

*kCASPER\_OpMul64x64FullSum* Walking 1 or more of J loop, doing  $c,r=r+a*b$  using  $64x64=128$ , but assume inner j loop.

*kCASPER\_OpMul64x64Reduce* Walking 1 or more of J loop, doing  $c,r=r+a*b$  using  $64x64=128$ , but sum all of w.

*kCASPER\_OpAdd64* Walking 1 or more of J loop, doing  $c,r[-1]=r+a*b$  using  $64x64=128$ , but skip 1st write.

*kCASPER\_OpSub64* Walking add with off\_AB, and in/out off\_RES doing  $c,r=r+a+c$  using  $64+64=65$ .

*kCASPER\_OpDouble64* Walking subtract with off\_AB, and in/out off\_RES doing  $r=r-a$  using  $64-64=64$ , with last borrow implicit if any.

*kCASPER\_OpXor64* Walking add to self with off\_RES doing  $c,r=r+r+c$  using  $64+64=65$ .

*kCASPER\_OpRSub64* Walking XOR with off\_AB, and in/out off\_RES doing  $r=r^a$  using  $64^64=64$ .

*kCASPER\_OpShiftLeft32* Walking subtract with off\_AB, and in/out off\_RES using  $r=a-r$ .

*kCASPER\_OpShiftRight32* Walking shift left doing  $r1,r=(b*D)|r1$ , where D is  $2^{\text{amt}}$  and is loaded by app (off\_CD not used)

*kCASPER\_OpCopy* Walking shift right doing  $r,r1=(b*D)|r1$ , where D is  $2^{(32-\text{amt})}$  and is loaded by app (off\_CD not used) and off\_RES starts at MSW.

*kCASPER\_OpRemask* Copy from ABoff to resoff, 64b at a time.

*kCASPER\_OpFill* Copy and mask from ABoff to resoff, 64b at a time.

*kCASPER\_OpZero* Fill RESOFF using 64 bits at a time with value in A and B.

*kCASPER\_OpCompare* Fill RESOFF using 64 bits at a time of 0s.

*kCASPER\_OpCompareFast* Compare two arrays, running all the way to the end.

#### 8.6.3.2 enum casper\_algo\_t

Enumerator

*kCASPER\_ECC\_P256* ECC\_P256.

*kCASPER\_ECC\_P384* ECC\_P384.

*kCASPER\_ECC\_P521* ECC\_P521.

### 8.6.4 Function Documentation

#### 8.6.4.1 void CASPER\_Init ( CASPER\_Type \* *base* )

Enable clock and disable reset for CASPER.

Parameters

<i>base</i>	CASPER base address
-------------	---------------------

#### 8.6.4.2 void CASPER\_Deinit( CASPER\_Type \* *base* )

Disable clock and enable reset.

Parameters

<i>base</i>	CASPER base address
-------------	---------------------

## 8.7 casper\_driver\_pkha

### 8.7.1 Overview

#### Functions

- void [CASPER\\_ModExp](#) (CASPER\_Type \*base, const uint8\_t \*signature, const uint8\_t \*pubN, size\_t wordLen, uint32\_t pubE, uint8\_t \*plaintext)
 

*Performs modular exponentiation -  $(A^E) \bmod N$ .*
- void [CASPER\\_ecc\\_init](#) (casper\_algo\_t curve)
 

*Initialize prime modulus mod in Casper memory.*
- void [CASPER\\_ECC\\_SECP256R1\\_Mul](#) (CASPER\_Type \*base, uint32\_t resX[8], uint32\_t resY[8], uint32\_t X[8], uint32\_t Y[8], uint32\_t scalar[8])
 

*Performs ECC secp256r1 point single scalar multiplication.*
- void [CASPER\\_ECC\\_SECP256R1\\_MulAdd](#) (CASPER\_Type \*base, uint32\_t resX[8], uint32\_t resY[8], uint32\_t X1[8], uint32\_t Y1[8], uint32\_t scalar1[8], uint32\_t X2[8], uint32\_t Y2[8], uint32\_t scalar2[8])
 

*Performs ECC secp256r1 point double scalar multiplication.*
- void [CASPER\\_ECC\\_SECP384R1\\_Mul](#) (CASPER\_Type \*base, uint32\_t resX[12], uint32\_t resY[12], uint32\_t X[12], uint32\_t Y[12], uint32\_t scalar[12])
 

*Performs ECC secp384r1 point single scalar multiplication.*
- void [CASPER\\_ECC\\_SECP384R1\\_MulAdd](#) (CASPER\_Type \*base, uint32\_t resX[12], uint32\_t resY[12], uint32\_t X1[12], uint32\_t Y1[12], uint32\_t scalar1[12], uint32\_t X2[12], uint32\_t Y2[12], uint32\_t scalar2[12])
 

*Performs ECC secp384r1 point double scalar multiplication.*
- void [CASPER\\_ECC\\_SECP521R1\\_Mul](#) (CASPER\_Type \*base, uint32\_t resX[18], uint32\_t resY[18], uint32\_t X[18], uint32\_t Y[18], uint32\_t scalar[18])
 

*Performs ECC secp521r1 point single scalar multiplication.*
- void [CASPER\\_ECC\\_SECP521R1\\_MulAdd](#) (CASPER\_Type \*base, uint32\_t resX[18], uint32\_t resY[18], uint32\_t X1[18], uint32\_t Y1[18], uint32\_t scalar1[18], uint32\_t X2[18], uint32\_t Y2[18], uint32\_t scalar2[18])
 

*Performs ECC secp521r1 point double scalar multiplication.*

### 8.7.2 Function Documentation

#### 8.7.2.1 void [CASPER\\_ModExp](#) ( CASPER\_Type \* base, const uint8\_t \* signature, const uint8\_t \* pubN, size\_t wordLen, uint32\_t pubE, uint8\_t \* plaintext )

This function performs modular exponentiation.

Parameters

	<i>base</i>	CASPER base address
	<i>signature</i>	first addend (in little endian format)
	<i>pubN</i>	modulus (in little endian format)
	<i>wordLen</i>	Size of pubN in bytes
	<i>pubE</i>	exponent
out	<i>plaintext</i>	Output array to store result of operation (in little endian format)

### 8.7.2.2 void CASPER\_ecc\_init ( casper\_algo\_t *curve* )

Set the prime modulus mod in Casper memory and set N\_wordlen according to selected algorithm.

Parameters

<i>curve</i>	elliptic curve algoritm
--------------	-------------------------

### 8.7.2.3 void CASPER\_ECC\_SECP256R1\_Mul ( CASPER\_Type \* *base*, uint32\_t *resX*[8], uint32\_t *resY*[8], uint32\_t *X*[8], uint32\_t *Y*[8], uint32\_t *scalar*[8] )

This function performs ECC secp256r1 point single scalar multiplication [resX; resY] = scalar \* [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	<i>base</i>	CASPER base address
out	<i>resX</i>	Output X affine coordinate in normal form, little endian.
out	<i>resY</i>	Output Y affine coordinate in normal form, little endian.
	<i>X</i>	Input X affine coordinate in normal form, little endian.
	<i>Y</i>	Input Y affine coordinate in normal form, little endian.
	<i>scalar</i>	Input scalar integer, in normal form, little endian.

### 8.7.2.4 void CASPER\_ECC\_SECP256R1\_MulAdd ( CASPER\_Type \* *base*, uint32\_t *resX*[8], uint32\_t *resY*[8], uint32\_t *X1*[8], uint32\_t *Y1*[8], uint32\_t *scalar1*[8], uint32\_t *X2*[8], uint32\_t *Y2*[8], uint32\_t *scalar2*[8] )

This function performs ECC secp256r1 point double scalar multiplication [resX; resY] = scalar1 \* [X1; Y1] + scalar2 \* [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All

arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	<i>base</i>	CASPER base address
out	<i>resX</i>	Output X affine coordinate.
out	<i>resY</i>	Output Y affine coordinate.
	<i>X1</i>	Input X1 affine coordinate.
	<i>Y1</i>	Input Y1 affine coordinate.
	<i>scalar1</i>	Input scalar1 integer.
	<i>X2</i>	Input X2 affine coordinate.
	<i>Y2</i>	Input Y2 affine coordinate.
	<i>scalar2</i>	Input scalar2 integer.

#### 8.7.2.5 void CASPER\_ECC\_SECP384R1\_Mul ( **CASPER\_Type \* base, uint32\_t resX[12], uint32\_t resY[12], uint32\_t X[12], uint32\_t Y[12], uint32\_t scalar[12]** )

This function performs ECC secp384r1 point single scalar multiplication [resX; resY] = scalar \* [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	<i>base</i>	CASPER base address
out	<i>resX</i>	Output X affine coordinate in normal form, little endian.
out	<i>resY</i>	Output Y affine coordinate in normal form, little endian.
	<i>X</i>	Input X affine coordinate in normal form, little endian.
	<i>Y</i>	Input Y affine coordinate in normal form, little endian.
	<i>scalar</i>	Input scalar integer, in normal form, little endian.

#### 8.7.2.6 void CASPER\_ECC\_SECP384R1\_MulAdd ( **CASPER\_Type \* base, uint32\_t resX[12], uint32\_t resY[12], uint32\_t X1[12], uint32\_t Y1[12], uint32\_t scalar1[12], uint32\_t X2[12], uint32\_t Y2[12], uint32\_t scalar2[12]** )

This function performs ECC secp384r1 point double scalar multiplication [resX; resY] = scalar1 \* [X1; Y1] + scalar2 \* [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	<i>base</i>	CASPER base address
out	<i>resX</i>	Output X affine coordinate.
out	<i>resY</i>	Output Y affine coordinate.
	<i>X1</i>	Input X1 affine coordinate.
	<i>Y1</i>	Input Y1 affine coordinate.
	<i>scalar1</i>	Input scalar1 integer.
	<i>X2</i>	Input X2 affine coordinate.
	<i>Y2</i>	Input Y2 affine coordinate.
	<i>scalar2</i>	Input scalar2 integer.

#### 8.7.2.7 void CASPER\_ECC\_SECP521R1\_Mul ( **CASPER\_Type \* base, uint32\_t resX[18], uint32\_t resY[18], uint32\_t X[18], uint32\_t Y[18], uint32\_t scalar[18]** )

This function performs ECC secp521r1 point single scalar multiplication [resX; resY] = scalar \* [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	<i>base</i>	CASPER base address
out	<i>resX</i>	Output X affine coordinate in normal form, little endian.
out	<i>resY</i>	Output Y affine coordinate in normal form, little endian.
	<i>X</i>	Input X affine coordinate in normal form, little endian.
	<i>Y</i>	Input Y affine coordinate in normal form, little endian.
	<i>scalar</i>	Input scalar integer, in normal form, little endian.

#### 8.7.2.8 void CASPER\_ECC\_SECP521R1\_MulAdd ( **CASPER\_Type \* base, uint32\_t resX[18], uint32\_t resY[18], uint32\_t X1[18], uint32\_t Y1[18], uint32\_t scalar1[18], uint32\_t X2[18], uint32\_t Y2[18], uint32\_t scalar2[18]** )

This function performs ECC secp521r1 point double scalar multiplication [resX; resY] = scalar1 \* [X1; Y1] + scalar2 \* [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32\_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

## Parameters

	<i>base</i>	CASPER base address
out	<i>resX</i>	Output X affine coordinate.
out	<i>resY</i>	Output Y affine coordinate.
	<i>X1</i>	Input X1 affine coordinate.
	<i>Y1</i>	Input Y1 affine coordinate.
	<i>scalar1</i>	Input scalar1 integer.
	<i>X2</i>	Input X2 affine coordinate.
	<i>Y2</i>	Input Y2 affine coordinate.
	<i>scalar2</i>	Input scalar2 integer.

# Chapter 9

## CMP: Analog Comparator Driver

### 9.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Analog Comparator (CMP) module of MCUXpresso SDK devices.

### 9.2 Function groups

The driver provides a set of functions to set two input sources of the on-chip comparator and compare the voltage of them.

#### 9.2.1 Initialization and deinitialization

The function `CMP_Init()` initializes the CMP with specified configurations. The function `CMP_GetDefaultConfig()` gets the default configurations.

The function `CMP_Deinit()` disables the module clock.

#### 9.2.2 Compare

The function `CMP_SetInputChannels()` configures the P-side and N-side input sources.

The function `CMP_SetVREF()` sets the reference voltage which can be dedicated to input 0 of both P and N sides.

The function `CMP_GetOutput()` gets the compare result of the two sides.

#### 9.2.3 Interrupt

Provides functions to enable/disable/clear CMP interrupts.

The function `CMP_EnableFilteredInterruptSource()` allows users to select which analog comparator output (filtered or un-filtered) is used for interrupt detection.

#### 9.2.4 Status

Provides functions to get the CMP status.

## 9.3 Typical use case

### 9.3.1 Polling Configuration

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/cmp\_1

### 9.3.2 Interrupt Configuration

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/cmp\_1

## Data Structures

- struct `cmp_config_t`  
*CMP configuration structure.* [More...](#)

## Enumerations

- enum `_cmp_input_mux` {
   
`kCMP_InputVREF` = 0U,  
`kCMP_Input1` = 1U,  
`kCMP_Input2` = 2U,  
`kCMP_Input3` = 3U,  
`kCMP_Input4` = 4U,  
`kCMP_Input5` = 5U }  
*CMP input mux for positive and negative sides.*
- enum `_cmp_interrupt_type` {
   
`kCMP_EdgeDisable` = 0U,  
`kCMP_EdgeRising` = 2U,  
`kCMP_EdgeFalling` = 4U,  
`kCMP_EdgeRisingFalling` = 6U,  
`kCMP_LevelDisable` = 1U,  
`kCMP_LevelHigh` = 3U,  
`kCMP_LevelLow` = 5U }  
*CMP interrupt type.*
- enum `cmp_vref_source_t` {
   
`KCMP_VREFSourceVDDA` = 1U,  
`KCMP_VREFSourceInternalVREF` = 0U }  
*CMP Voltage Reference source.*
- enum `cmp_filtercfg_samplemode_t` {
   
`kCMP_FilterSampleMode0` = 0U,  
`kCMP_FilterSampleMode1` = 1U,  
`kCMP_FilterSampleMode2` = 2U,  
`kCMP_FilterSampleMode3` = 3U }  
*CMP Filter sample mode.*

- enum `cmp_filtercfg_clkdiv_t` {
 `kCMP_FilterClockDivide1` = 0U,
 `kCMP_FilterClockDivide2` = 1U,
 `kCMP_FilterClockDivide4` = 2U,
 `kCMP_FilterClockDivide8` = 3U,
 `kCMP_FilterClockDivide16` = 4U,
 `kCMP_FilterClockDivide32` = 5U,
 `kCMP_FilterClockDivide64` = 6U }

*CMP Filter clock divider.*

## Driver version

- #define `FSL_CMP_DRIVER_VERSION` (MAKE\_VERSION(2U, 2U, 1U))  
*Driver version 2.2.1.*

## Initialization and deinitialization

- void `CMP_Init` (const `cmp_config_t` \*config)  
*CMP initialization.*
- void `CMP_Deinit` (void)  
*CMP deinitialization.*
- void `CMP_GetDefaultConfig` (`cmp_config_t` \*config)  
*Initializes the CMP user configuration structure.*

## Compare Interface

- static void `CMP_SetInputChannels` (uint8\_t positiveChannel, uint8\_t negativeChannel)
- void `CMP_SetVREF` (const `cmp_vref_config_t` \*config)  
*Configures the VREFINPUT.*
- static bool `CMP_GetOutput` (void)  
*Get CMP compare output.*

## Interrupt Interface

- static void `CMP_EnableInterrupt` (uint32\_t type)  
*CMP enable interrupt.*
- static void `CMP_DisableInterrupt` (void)  
*CMP disable interrupt.*
- static void `CMP_ClearInterrupt` (void)  
*CMP clear interrupt.*
- static void `CMP_EnableFilteredInterruptSource` (bool enable)  
*Select which Analog comparator output (filtered or un-filtered) is used for interrupt detection.*

## Status Interface

- static bool `CMP_GetPreviousInterruptStatus` (void)  
*Get CMP interrupt status before interrupt enable.*
- static bool `CMP_GetInterruptStatus` (void)  
*Get CMP interrupt status after interrupt enable.*

## Filter Interface

- static void `CMP_FilterSampleConfig` (`cmp_filtercfg_samplemode_t` filterSampleMode, `cmp_filtercfg_clkdiv_t` filterClockDivider)  
*CMP Filter Sample Config.*

## 9.4 Data Structure Documentation

### 9.4.1 struct cmp\_config\_t

#### Data Fields

- bool `enableHysteresis`  
*Enable hysteresis.*
- bool `enableLowPower`  
*Enable low power mode.*

#### Field Documentation

- (1) `bool cmp_config_t::enableHysteresis`
- (2) `bool cmp_config_t::enableLowPower`

## 9.5 Macro Definition Documentation

### 9.5.1 #define FSL\_CMP\_DRIVER\_VERSION (MAKE\_VERSION(2U, 2U, 1U))

## 9.6 Enumeration Type Documentation

### 9.6.1 enum \_cmp\_input\_mux

Enumerator

- kCMP\_InputVREF* Cmp input from VREF.
- kCMP\_Input1* Cmp input source 1.
- kCMP\_Input2* Cmp input source 2.
- kCMP\_Input3* Cmp input source 3.
- kCMP\_Input4* Cmp input source 4.
- kCMP\_Input5* Cmp input source 5.

### 9.6.2 enum \_cmp\_interrupt\_type

Enumerator

- kCMP\_EdgeDisable* Disable edge interrupt.
- kCMP\_EdgeRising* Interrupt on falling edge.
- kCMP\_EdgeFalling* Interrupt on rising edge.

*kCMP\_EdgeRisingFalling* Interrupt on both rising and falling edges.

*kCMP\_LevelDisable* Disable level interrupt.

*kCMP\_LevelHigh* Interrupt on high level.

*kCMP\_LevelLow* Interrupt on low level.

### 9.6.3 enum cmp\_vref\_source\_t

Enumerator

*KCMP\_VREFSourceVDDA* Select VDDA as VREF.

*KCMP\_VREFSourceInternalVREF* Select internal VREF as VREF.

### 9.6.4 enum cmp\_filtercfg\_samplemode\_t

Enumerator

*kCMP\_FilterSampleMode0* Bypass mode. Filtering is disabled.

*kCMP\_FilterSampleMode1* Filter 1 clock period.

*kCMP\_FilterSampleMode2* Filter 2 clock period.

*kCMP\_FilterSampleMode3* Filter 3 clock period.

### 9.6.5 enum cmp\_filtercfg\_clkdiv\_t

Enumerator

*kCMP\_FilterClockDivide1* Filter clock period duration equals 1 analog comparator clock period.

*kCMP\_FilterClockDivide2* Filter clock period duration equals 2 analog comparator clock period.

*kCMP\_FilterClockDivide4* Filter clock period duration equals 4 analog comparator clock period.

*kCMP\_FilterClockDivide8* Filter clock period duration equals 8 analog comparator clock period.

*kCMP\_FilterClockDivide16* Filter clock period duration equals 16 analog comparator clock period.

*kCMP\_FilterClockDivide32* Filter clock period duration equals 32 analog comparator clock period.

*kCMP\_FilterClockDivide64* Filter clock period duration equals 64 analog comparator clock period.

## 9.7 Function Documentation

### 9.7.1 void CMP\_Init( const cmp\_config\_t \* config )

This function enables the CMP module and do necessary settings.

## Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

**9.7.2 void CMP\_Deinit ( void )**

This function gates the clock for CMP module.

**9.7.3 void CMP\_GetDefaultConfig ( cmp\_config\_t \* *config* )**

This function initializes the user configuration structure to these default values.

```
* config->enableHysteresis      = true;
* config->enableLowPower       = true;
* config->filterClockDivider  = kCMP_FilterClockDivide1;
* config->filterSampleMode    = kCMP_FilterSampleMode0;
*
```

## Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

**9.7.4 void CMP\_SetVREF ( const cmp\_vref\_config\_t \* *config* )**

## Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

**9.7.5 static bool CMP\_GetOutput ( void ) [inline], [static]**

## Returns

The output result. true: voltage on positive side is greater than negative side. false: voltage on positive side is lower than negative side.

**9.7.6 static void CMP\_EnableInterrupt ( uint32\_t *type* ) [inline], [static]**

Parameters

<i>type</i>	CMP interrupt type. See "_cmp_interrupt_type".
-------------	--

### 9.7.7 static void CMP\_EnableFilteredInterruptSource( bool *enable* ) [inline], [static]

Parameters

<i>enable</i>	false: Select Analog Comparator raw output (unfiltered) as input for interrupt detection. true: Select Analog Comparator filtered output as input for interrupt detection.
---------------	--

Note

: When CMP is configured as the wakeup source in power down mode, this function must use the raw output as the interrupt source, that is, call this function and set parameter enable to false.

### 9.7.8 static bool CMP\_GetPreviousInterruptStatus( void ) [inline], [static]

Returns

Interrupt status. true: interrupt pending, false: no interrupt pending.

### 9.7.9 static bool CMP\_GetInterruptStatus( void ) [inline], [static]

Returns

Interrupt status. true: interrupt pending, false: no interrupt pending.

### 9.7.10 static void CMP\_FilterSampleConfig( cmp\_filtercfg\_samplemode\_t *filterSampleMode*, cmp\_filtercfg\_clkdiv\_t *filterClockDivider* ) [inline], [static]

This function allows the users to configure the sampling mode and clock divider of the CMP Filter.

## Parameters

<i>filterSample-Mode</i>	CMP Select filter sample mode
<i>filterClock-Divider</i>	CMP Set fileter clock divider

# Chapter 10

## Common Driver

### 10.1 Overview

The MCUXpresso SDK provides a driver for the common module of MCUXpresso SDK devices.

#### Macros

- `#define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1`  
*Macro to use the default weak IRQ handler in drivers.*
- `#define MAKE_STATUS(group, code) (((group)*100L) + (code))`  
*Construct a status code value from a group and code number.*
- `#define MAKE_VERSION(major, minor, bugfix) (((major)*65536L) + ((minor)*256L) + (bugfix))`  
*Construct the version number for drivers.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U`  
*No debug console.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U`  
*Debug console based on UART.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U`  
*Debug console based on LPUART.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U`  
*Debug console based on LPSCI.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U`  
*Debug console based on USBCDC.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U`  
*Debug console based on FLEXCOMM.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U`  
*Debug console based on i.MX UART.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U`  
*Debug console based on LPC\_VUSART.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U`  
*Debug console based on LPC\_USART.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U`  
*Debug console based on SWO.*
- `#define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U`  
*Debug console based on QSCI.*
- `#define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))`  
*Computes the number of elements in an array.*

#### Typedefs

- `typedef int32_t status_t`  
*Type used for all status and error return values.*

## Enumerations

- enum `_status_groups` {  
    `kStatusGroup_Generic` = 0,  
    `kStatusGroup_FLASH` = 1,  
    `kStatusGroup_LP SPI` = 4,  
    `kStatusGroup_FLEXIO_SPI` = 5,  
    `kStatusGroup_DSPI` = 6,  
    `kStatusGroup_FLEXIO_UART` = 7,  
    `kStatusGroup_FLEXIO_I2C` = 8,  
    `kStatusGroup_LPI2C` = 9,  
    `kStatusGroup_UART` = 10,  
    `kStatusGroup_I2C` = 11,  
    `kStatusGroup_LPSCI` = 12,  
    `kStatusGroup_LPUART` = 13,  
    `kStatusGroup_SPI` = 14,  
    `kStatusGroup_XRDC` = 15,  
    `kStatusGroup_SEMA42` = 16,  
    `kStatusGroup_SDHC` = 17,  
    `kStatusGroup_SDMMC` = 18,  
    `kStatusGroup_SAI` = 19,  
    `kStatusGroup_MCG` = 20,  
    `kStatusGroup_SCG` = 21,  
    `kStatusGroup_SD SPI` = 22,  
    `kStatusGroup_FLEXIO_I2S` = 23,  
    `kStatusGroup_FLEXIO_MCULCD` = 24,  
    `kStatusGroup_FLASHIAP` = 25,  
    `kStatusGroup_FLEXCOMM_I2C` = 26,  
    `kStatusGroup_I2S` = 27,  
    `kStatusGroup_IUART` = 28,  
    `kStatusGroup_CSI` = 29,  
    `kStatusGroup_MIPI_DSI` = 30,  
    `kStatusGroup_SDRAMC` = 35,  
    `kStatusGroup_POWER` = 39,  
    `kStatusGroup_ENET` = 40,  
    `kStatusGroup_PHY` = 41,  
    `kStatusGroup_TRGMUX` = 42,  
    `kStatusGroup_SMARTCARD` = 43,  
    `kStatusGroup_LMEM` = 44,  
    `kStatusGroup_QSPI` = 45,  
    `kStatusGroup_DMA` = 50,  
    `kStatusGroup_EDMA` = 51,  
    `kStatusGroup_DMAMGR` = 52,  
    `kStatusGroup_FLEXCAN` = 53,  
    `kStatusGroup_LTC` = 54,  
    `kStatusGroup_FLEXIO_CAMERA` = 55,  
    `kStatusGroup_LPC_SPI` = 56,  
    `kStatusGroup_LPC_USACARD` = 58,  
    `kStatusGroup_SDIF` = 59,

```

kStatusGroup_NETC = 165 }

Status group numbers.
• enum {
    kStatus_Success = MAKE_STATUS(kStatusGroup_Generic, 0),
    kStatus_Fail = MAKE_STATUS(kStatusGroup_Generic, 1),
    kStatus_ReadOnly = MAKE_STATUS(kStatusGroup_Generic, 2),
    kStatus_OutOfRange = MAKE_STATUS(kStatusGroup_Generic, 3),
    kStatus_InvalidArgument = MAKE_STATUS(kStatusGroup_Generic, 4),
    kStatus_Timeout = MAKE_STATUS(kStatusGroup_Generic, 5),
    kStatus_NoTransferInProgress,
    kStatus_Busy = MAKE_STATUS(kStatusGroup_Generic, 7),
    kStatus_NoData }

Generic status return codes.

```

## Functions

- void \* **SDK\_Malloc** (size\_t size, size\_t alignbytes)  
*Allocate memory with given alignment and aligned size.*
- void **SDK\_Free** (void \*ptr)  
*Free memory.*
- void **SDK\_DelayAtLeastUs** (uint32\_t delayTime\_us, uint32\_t coreClock\_Hz)  
*Delay at least for some time.*

## Driver version

- #define **FSL\_COMMON\_DRIVER\_VERSION** (MAKE\_VERSION(2, 4, 0))  
*common driver version.*

## Min/max macros

- #define **MIN**(a, b) (((a) < (b)) ? (a) : (b))
- #define **MAX**(a, b) (((a) > (b)) ? (a) : (b))

## UINT16\_MAX/UINT32\_MAX value

- #define **UINT16\_MAX** ((uint16\_t)-1)
- #define **UINT32\_MAX** ((uint32\_t)-1)

## Suppress fallthrough warning macro

- #define **SUPPRESS\_FALL\_THROUGH\_WARNING()**

## 10.2 Macro Definition Documentation

### 10.2.1 #define FSL\_DRIVER\_TRANSFER\_DOUBLE\_WEAK\_IRQ 1

### 10.2.2 #define MAKE\_STATUS( *group*, *code* ) (((*group*)\*100L) + (*code*))

### 10.2.3 #define MAKE\_VERSION( *major*, *minor*, *bugfix* ) (((*major*)\*65536L) + ((*minor*)\*256L) + (*bugfix*))

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

Unused	Major Version	Minor Version	Bug Fix	
31	25 24	17 16	9 8	0

### 10.2.4 #define FSL\_COMMON\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 0))

### 10.2.5 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE 0U

### 10.2.6 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_UART 1U

### 10.2.7 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPUART 2U

### 10.2.8 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPSCI 3U

### 10.2.9 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_USBCDC 4U

### 10.2.10 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_FLEXCOMM 5U

### 10.2.11 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_IUART 6U

### 10.2.12 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_VUSART 7U

### 10.2.13 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_MINI\_USART 8U

### 10.2.14 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_SWO 9U

### 10.2.15 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_QSCI 10U

### 10.2.16 #define ARRAY\_SIZE( *x* ) (sizeof(*x*) / sizeof((*x*)[0]))

## 10.3 Typedef Documentation

### 10.3.1 typedef int32\_t status\_t

## 10.4 Enumeration Type Documentation

### 10.4.1 enum \_status\_groups

Enumerator

- kStatusGroup\_Generic*** Group number for generic status codes.
- kStatusGroup\_FLASH*** Group number for FLASH status codes.
- kStatusGroup\_LP SPI*** Group number for LP SPI status codes.
- kStatusGroup\_FLEXIO\_SPI*** Group number for FLEXIO SPI status codes.
- kStatusGroup\_DSPI*** Group number for DSPI status codes.
- kStatusGroup\_FLEXIO\_UART*** Group number for FLEXIO UART status codes.
- kStatusGroup\_FLEXIO\_I2C*** Group number for FLEXIO I2C status codes.
- kStatusGroup\_LPI2C*** Group number for LPI2C status codes.
- kStatusGroup\_UART*** Group number for UART status codes.
- kStatusGroup\_I2C*** Group number for I2C status codes.
- kStatusGroup\_LPSCI*** Group number for LPSCI status codes.
- kStatusGroup\_LPUART*** Group number for LPUART status codes.
- kStatusGroup\_SPI*** Group number for SPI status code.
- kStatusGroup\_XRDC*** Group number for XRDC status code.
- kStatusGroup\_SEMA42*** Group number for SEMA42 status code.
- kStatusGroup\_SDHC*** Group number for SDHC status code.
- kStatusGroup\_SDMMC*** Group number for SDMMC status code.
- kStatusGroup\_SAI*** Group number for SAI status code.
- kStatusGroup\_MCG*** Group number for MCG status codes.
- kStatusGroup\_SCG*** Group number for SCG status codes.
- kStatusGroup\_SD SPI*** Group number for SD SPI status codes.
- kStatusGroup\_FLEXIO\_I2S*** Group number for FLEXIO I2S status codes.
- kStatusGroup\_FLEXIO\_MCU LCD*** Group number for FLEXIO LCD status codes.
- kStatusGroup\_FLASHIAP*** Group number for FLASHIAP status codes.
- kStatusGroup\_FLEXCOMM\_I2C*** Group number for FLEXCOMM I2C status codes.
- kStatusGroup\_I2S*** Group number for I2S status codes.
- kStatusGroup\_IUART*** Group number for IUART status codes.
- kStatusGroup\_CSI*** Group number for CSI status codes.
- kStatusGroup\_MIPI\_DSI*** Group number for MIPI DSI status codes.
- kStatusGroup\_SDRAMC*** Group number for SDRAMC status codes.
- kStatusGroup\_POWER*** Group number for POWER status codes.
- kStatusGroup\_ENET*** Group number for ENET status codes.
- kStatusGroup\_PHY*** Group number for PHY status codes.
- kStatusGroup\_TRGMUX*** Group number for TRGMUX status codes.
- kStatusGroup\_SMARTCARD*** Group number for SMARTCARD status codes.
- kStatusGroup\_LMEM*** Group number for LMEM status codes.
- kStatusGroup\_QSPI*** Group number for QSPI status codes.
- kStatusGroup\_DMA*** Group number for DMA status codes.
- kStatusGroup\_EDMA*** Group number for EDMA status codes.
- kStatusGroup\_DMAMGR*** Group number for DMAMGR status codes.

*kStatusGroup\_FLEXCAN* Group number for FlexCAN status codes.  
*kStatusGroup\_LTC* Group number for LTC status codes.  
*kStatusGroup\_FLEXIO\_CAMERA* Group number for FLEXIO CAMERA status codes.  
*kStatusGroup\_LPC\_SPI* Group number for LPC\_SPI status codes.  
*kStatusGroup\_LPC\_USART* Group number for LPC\_USART status codes.  
*kStatusGroup\_DMIC* Group number for DMIC status codes.  
*kStatusGroup\_SDIF* Group number for SDIF status codes.  
*kStatusGroup\_SPIFI* Group number for SPIFI status codes.  
*kStatusGroup OTP* Group number for OTP status codes.  
*kStatusGroup\_MCAN* Group number for MCAN status codes.  
*kStatusGroup\_CAAM* Group number for CAAM status codes.  
*kStatusGroup\_ECSPI* Group number for ECSPI status codes.  
*kStatusGroup\_USDHC* Group number for USDHC status codes.  
*kStatusGroup\_LPC\_I2C* Group number for LPC\_I2C status codes.  
*kStatusGroup\_DCP* Group number for DCP status codes.  
*kStatusGroup\_MSCAN* Group number for MSCAN status codes.  
*kStatusGroup\_ESAI* Group number for ESAI status codes.  
*kStatusGroup\_FLEXSPI* Group number for FLEXSPI status codes.  
*kStatusGroup\_MMDC* Group number for MMDC status codes.  
*kStatusGroup\_PDM* Group number for MIC status codes.  
*kStatusGroup\_SDMA* Group number for SDMA status codes.  
*kStatusGroup\_ICS* Group number for ICS status codes.  
*kStatusGroup\_SPDIF* Group number for SPDIF status codes.  
*kStatusGroup\_LPC\_MINISPI* Group number for LPC\_MINISPI status codes.  
*kStatusGroup\_HASHCRYPT* Group number for Hashcrypt status codes.  
*kStatusGroup\_LPC\_SPI\_SSP* Group number for LPC\_SPI\_SSP status codes.  
*kStatusGroup\_I3C* Group number for I3C status codes.  
*kStatusGroup\_LPC\_I2C\_1* Group number for LPC\_I2C\_1 status codes.  
*kStatusGroup\_NOTIFIER* Group number for NOTIFIER status codes.  
*kStatusGroup\_DebugConsole* Group number for debug console status codes.  
*kStatusGroup\_SEMC* Group number for SEMC status codes.  
*kStatusGroup\_ApplicationRangeStart* Starting number for application groups.  
*kStatusGroup\_IAP* Group number for IAP status codes.  
*kStatusGroup\_SFA* Group number for SFA status codes.  
*kStatusGroup\_SPC* Group number for SPC status codes.  
*kStatusGroup\_PUF* Group number for PUF status codes.  
*kStatusGroup\_TOUCH\_PANEL* Group number for touch panel status codes.  
*kStatusGroup\_VBAT* Group number for VBAT status codes.  
*kStatusGroup\_HAL\_GPIO* Group number for HAL GPIO status codes.  
*kStatusGroup\_HAL\_UART* Group number for HAL UART status codes.  
*kStatusGroup\_HAL\_TIMER* Group number for HAL TIMER status codes.  
*kStatusGroup\_HAL\_SPI* Group number for HAL SPI status codes.  
*kStatusGroup\_HAL\_I2C* Group number for HAL I2C status codes.  
*kStatusGroup\_HAL\_FLASH* Group number for HAL FLASH status codes.  
*kStatusGroup\_HAL\_PWM* Group number for HAL PWM status codes.

*kStatusGroup\_HAL\_RNG* Group number for HAL RNG status codes.  
*kStatusGroup\_HAL\_I2S* Group number for HAL I2S status codes.  
*kStatusGroup\_TIMERMANAGER* Group number for TiMER MANAGER status codes.  
*kStatusGroup\_SERIALMANAGER* Group number for SERIAL MANAGER status codes.  
*kStatusGroup\_LED* Group number for LED status codes.  
*kStatusGroup\_BUTTON* Group number for BUTTON status codes.  
*kStatusGroup\_EXTERN\_EEPROM* Group number for EXTERN EEPROM status codes.  
*kStatusGroup\_SHELL* Group number for SHELL status codes.  
*kStatusGroup\_MEM\_MANAGER* Group number for MEM MANAGER status codes.  
*kStatusGroup\_LIST* Group number for List status codes.  
*kStatusGroup\_OSA* Group number for OSA status codes.  
*kStatusGroup\_COMMON\_TASK* Group number for Common task status codes.  
*kStatusGroup\_MSG* Group number for messaging status codes.  
*kStatusGroup\_SDK\_OCOTP* Group number for OCOTP status codes.  
*kStatusGroup\_SDK\_FLEXSPINOR* Group number for FLEXSPINOR status codes.  
*kStatusGroup\_CODEC* Group number for codec status codes.  
*kStatusGroup\_ASRC* Group number for codec status ASRC.  
*kStatusGroup\_OTFAD* Group number for codec status codes.  
*kStatusGroup\_SDIOSLV* Group number for SDIOSLV status codes.  
*kStatusGroup\_MECC* Group number for MECC status codes.  
*kStatusGroup\_ENET\_QOS* Group number for ENET\_QOS status codes.  
*kStatusGroup\_LOG* Group number for LOG status codes.  
*kStatusGroup\_I3CBUS* Group number for I3CBUS status codes.  
*kStatusGroup\_QSCI* Group number for QSCI status codes.  
*kStatusGroup\_SNT* Group number for SNT status codes.  
*kStatusGroup\_QUEUEDSPI* Group number for QSPI status codes.  
*kStatusGroup\_POWER\_MANAGER* Group number for POWER\_MANAGER status codes.  
*kStatusGroup\_IPED* Group number for IPED status codes.  
*kStatusGroup\_ELS\_PKC* Group number for ELS PKC status codes.  
*kStatusGroup\_HOSTIF* Group number for HOSTIF status codes.  
*kStatusGroup\_CLIF* Group number for CLIF status codes.  
*kStatusGroup\_BMA* Group number for BMA status codes.  
*kStatusGroup\_NETC* Group number for NETC status codes.

#### 10.4.2 anonymous enum

Enumerator

*kStatus\_Success* Generic status for Success.  
*kStatus\_Fail* Generic status for Fail.  
*kStatus\_ReadOnly* Generic status for read only failure.  
*kStatus\_OutOfRange* Generic status for out of range access.  
*kStatus\_InvalidArgument* Generic status for invalid argument check.  
*kStatus\_Timeout* Generic status for timeout.

***kStatus\_NoTransferInProgress*** Generic status for no transfer in progress.

***kStatus\_Busy*** Generic status for module is busy.

***kStatus\_NoData*** Generic status for no data is found for the operation.

## 10.5 Function Documentation

### 10.5.1 void\* SDK\_Malloc ( size\_t *size*, size\_t *alignbytes* )

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

<i>size</i>	The length required to malloc.
<i>alignbytes</i>	The alignment size.

Return values

<i>The</i>	allocated memory.
------------	-------------------

### 10.5.2 void SDK\_Free ( void \* *ptr* )

Parameters

<i>ptr</i>	The memory to be release.
------------	---------------------------

### 10.5.3 void SDK\_DelayAtLeastUs ( uint32\_t *delayTime\_us*, uint32\_t *coreClock\_Hz* )

Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

<i>delayTime_us</i>	Delay time in unit of microsecond.
<i>coreClock_Hz</i>	Core clock frequency with Hz.

# Chapter 11

## CTIMER: Standard counter/timers

### 11.1 Overview

The MCUXpresso SDK provides a driver for the cTimer module of MCUXpresso SDK devices.

### 11.2 Function groups

The cTimer driver supports the generation of PWM signals, input capture, and setting up the timer match conditions.

#### 11.2.1 Initialization and deinitialization

The function [CTIMER\\_Init\(\)](#) initializes the cTimer with specified configurations. The function [CTIMER\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the counter/timer mode and input selection when running in counter mode.

The function [CTIMER\\_Deinit\(\)](#) stops the timer and turns off the module clock.

#### 11.2.2 PWM Operations

The function [CTIMER\\_SetupPwm\(\)](#) sets up channels for PWM output. Each channel has its own duty cycle, however the same PWM period is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100  
0=inactive signal(0% duty cycle) and 100=always active signal (100% duty cycle).

The function [CTIMER\\_UpdatePwmDutyCycle\(\)](#) updates the PWM signal duty cycle of a particular channel.

#### 11.2.3 Match Operation

The function [CTIMER\\_SetupMatch\(\)](#) sets up channels for match operation. Each channel is configured with a match value: if the counter should stop on match, if counter should reset on match, and output pin action. The output signal can be cleared, set, or toggled on match.

#### 11.2.4 Input capture operations

The function [CTIMER\\_SetupCapture\(\)](#) sets up an channel for input capture. The user can specify the capture edge and if a interrupt should be generated when processing the input signal.

## 11.3 Typical use case

### 11.3.1 Match example

Set up a match channel to toggle output when a match occurs. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/ctimer

### 11.3.2 PWM output example

Set up a channel for PWM output. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/ctimer

## Files

- file [fsl\\_ctimer.h](#)

## Data Structures

- struct [ctimer\\_match\\_config\\_t](#)  
*Match configuration. [More...](#)*
- struct [ctimer\\_config\\_t](#)  
*Timer configuration structure. [More...](#)*

## Enumerations

- enum [ctimer\\_capture\\_channel\\_t](#) {
   
kCTIMER\_Capture\_0 = 0U,
   
kCTIMER\_Capture\_1,
   
kCTIMER\_Capture\_2,
   
kCTIMER\_Capture\_3 }
   
*List of Timer capture channels.*
- enum [ctimer\\_capture\\_edge\\_t](#) {
   
kCTIMER\_Capture\_RiseEdge = 1U,
   
kCTIMER\_Capture\_FallEdge = 2U,
   
kCTIMER\_Capture\_BothEdge = 3U }
   
*List of capture edge options.*
- enum [ctimer\\_match\\_t](#) {
   
kCTIMER\_Match\_0 = 0U,
   
kCTIMER\_Match\_1,
   
kCTIMER\_Match\_2,
   
kCTIMER\_Match\_3 }
   
*List of Timer match registers.*
- enum [ctimer\\_external\\_match\\_t](#) {
   
kCTIMER\_External\_Match\_0 = (1UL << 0),
   
kCTIMER\_External\_Match\_1 = (1UL << 1),
   
kCTIMER\_External\_Match\_2 = (1UL << 2),
   
kCTIMER\_External\_Match\_3 = (1UL << 3) }

- *List of external match.*
- enum `ctimer_match_output_control_t` {
   
    `kCTIMER_Output_NoAction` = 0U,
   
    `kCTIMER_Output_Clear`,
   
    `kCTIMER_Output_Set`,
   
    `kCTIMER_Output_Toggle` }
- List of output control options.*
- enum `ctimer_timer_mode_t`
  
    *List of Timer modes.*
- enum `ctimer_interrupt_enable_t` {
   
    `kCTIMER_Match0InterruptEnable` = CTIMER\_MCR\_MR0I\_MASK,
   
    `kCTIMER_Match1InterruptEnable` = CTIMER\_MCR\_MR1I\_MASK,
   
    `kCTIMER_Match2InterruptEnable` = CTIMER\_MCR\_MR2I\_MASK,
   
    `kCTIMER_Match3InterruptEnable` = CTIMER\_MCR\_MR3I\_MASK,
   
    `kCTIMER_Capture0InterruptEnable` = CTIMER\_CCR\_CAP0I\_MASK,
   
    `kCTIMER_Capture1InterruptEnable` = CTIMER\_CCR\_CAP1I\_MASK,
   
    `kCTIMER_Capture2InterruptEnable` = CTIMER\_CCR\_CAP2I\_MASK,
   
    `kCTIMER_Capture3InterruptEnable` = CTIMER\_CCR\_CAP3I\_MASK }
- List of Timer interrupts.*
- enum `ctimer_status_flags_t` {
   
    `kCTIMER_Match0Flag` = CTIMER\_IR\_MR0INT\_MASK,
   
    `kCTIMER_Match1Flag` = CTIMER\_IR\_MR1INT\_MASK,
   
    `kCTIMER_Match2Flag` = CTIMER\_IR\_MR2INT\_MASK,
   
    `kCTIMER_Match3Flag` = CTIMER\_IR\_MR3INT\_MASK,
   
    `kCTIMER_Capture0Flag` = CTIMER\_IR\_CR0INT\_MASK,
   
    `kCTIMER_Capture1Flag` = CTIMER\_IR\_CR1INT\_MASK,
   
    `kCTIMER_Capture2Flag` = CTIMER\_IR\_CR2INT\_MASK,
   
    `kCTIMER_Capture3Flag` = CTIMER\_IR\_CR3INT\_MASK }
- List of Timer flags.*
- enum `ctimer_callback_type_t` {
   
    `kCTIMER_SingleCallback`,
   
    `kCTIMER_MultipleCallback` }
- Callback type when registering for a callback.*

## Functions

- void `CTIMER_SetupMatch` (CTIMER\_Type \*base, `ctimer_match_t` matchChannel, const `ctimer_match_config_t` \*config)
   
    *Setup the match register.*
- uint32\_t `CTIMER_GetOutputMatchStatus` (CTIMER\_Type \*base, uint32\_t matchChannel)
   
    *Get the status of output match.*
- void `CTIMER_SetupCapture` (CTIMER\_Type \*base, `ctimer_capture_channel_t` capture, `ctimer_capture_edge_t` edge, bool enableInt)
   
    *Setup the capture.*
- static uint32\_t `CTIMER_GetTimerCountValue` (CTIMER\_Type \*base)
   
    *Get the timer count value from TC register.*
- void `CTIMER_RegisterCallBack` (CTIMER\_Type \*base, `ctimer_callback_t` \*cb\_func, `ctimer_callback_type_t` cb\_type)

- static void **CTIMER\_Reset** (CTIMER\_Type \*base)  
*Reset the counter.*
- static void **CTIMER\_SetPrescale** (CTIMER\_Type \*base, uint32\_t prescale)  
*Setup the timer prescale value.*
- static uint32\_t **CTIMER\_GetCaptureValue** (CTIMER\_Type \*base, ctimer\_capture\_channel\_t capture)  
*Get capture channel value.*
- static void **CTIMER\_EnableResetMatchChannel** (CTIMER\_Type \*base, ctimer\_match\_t match, bool enable)  
*Enable reset match channel.*
- static void **CTIMER\_EnableStopMatchChannel** (CTIMER\_Type \*base, ctimer\_match\_t match, bool enable)  
*Enable stop match channel.*
- static void **CTIMER\_EnableRisingEdgeCapture** (CTIMER\_Type \*base, ctimer\_capture\_channel\_t capture, bool enable)  
*Enable capture channel rising edge.*
- static void **CTIMER\_EnableFallingEdgeCapture** (CTIMER\_Type \*base, ctimer\_capture\_channel\_t capture, bool enable)  
*Enable capture channel falling edge.*

## Driver version

- #define **FSL\_CTIMER\_DRIVER\_VERSION** (MAKE\_VERSION(2, 3, 1))  
*Version 2.3.1.*

## Initialization and deinitialization

- void **CTIMER\_Init** (CTIMER\_Type \*base, const ctimer\_config\_t \*config)  
*Ungates the clock and configures the peripheral for basic operation.*
- void **CTIMER\_Deinit** (CTIMER\_Type \*base)  
*Gates the timer clock.*
- void **CTIMER\_GetDefaultConfig** (ctimer\_config\_t \*config)  
*Fills in the timers configuration structure with the default settings.*

## PWM setup operations

- status\_t **CTIMER\_SetupPwmPeriod** (CTIMER\_Type \*base, const ctimer\_match\_t pwmPeriodChannel, ctimer\_match\_t matchChannel, uint32\_t pwmPeriod, uint32\_t pulsePeriod, bool enableInt)  
*Configures the PWM signal parameters.*
- status\_t **CTIMER\_SetupPwm** (CTIMER\_Type \*base, const ctimer\_match\_t pwmPeriodChannel, ctimer\_match\_t matchChannel, uint8\_t dutyCyclePercent, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz, bool enableInt)  
*Configures the PWM signal parameters.*
- static void **CTIMER\_UpdatePwmPulsePeriod** (CTIMER\_Type \*base, ctimer\_match\_t matchChannel, uint32\_t pulsePeriod)  
*Updates the pulse period of an active PWM signal.*
- void **CTIMER\_UpdatePwmDutyCycle** (CTIMER\_Type \*base, const ctimer\_match\_t pwmPeriodChannel, ctimer\_match\_t matchChannel, uint8\_t dutyCyclePercent)

*Updates the duty cycle of an active PWM signal.*

## Interrupt Interface

- static void [CTIMER\\_EnableInterrupts](#) (CTIMER\_Type \*base, uint32\_t mask)  
*Enables the selected Timer interrupts.*
- static void [CTIMER\\_DisableInterrupts](#) (CTIMER\_Type \*base, uint32\_t mask)  
*Disables the selected Timer interrupts.*
- static uint32\_t [CTIMER\\_GetEnabledInterrupts](#) (CTIMER\_Type \*base)  
*Gets the enabled Timer interrupts.*

## Status Interface

- static uint32\_t [CTIMER\\_GetStatusFlags](#) (CTIMER\_Type \*base)  
*Gets the Timer status flags.*
- static void [CTIMER\\_ClearStatusFlags](#) (CTIMER\_Type \*base, uint32\_t mask)  
*Clears the Timer status flags.*

## Counter Start and Stop

- static void [CTIMER\\_StartTimer](#) (CTIMER\_Type \*base)  
*Starts the Timer counter.*
- static void [CTIMER\\_StopTimer](#) (CTIMER\_Type \*base)  
*Stops the Timer counter.*

## 11.4 Data Structure Documentation

### 11.4.1 struct ctimer\_match\_config\_t

This structure holds the configuration settings for each match register.

## Data Fields

- uint32\_t [matchValue](#)  
*This is stored in the match register.*
- bool [enableCounterReset](#)  
*true: Match will reset the counter false: Match will not reset the counter*
- bool [enableCounterStop](#)  
*true: Match will stop the counter false: Match will not stop the counter*
- [ctimer\\_match\\_output\\_control\\_t](#) [outControl](#)  
*Action to be taken on a match on the EM bit/output.*
- bool [outPinInitState](#)  
*Initial value of the EM bit/output.*
- bool [enableInterrupt](#)  
*true: Generate interrupt upon match false: Do not generate interrupt on match*

### 11.4.2 struct ctimer\_config\_t

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the [CTIMER\\_GetDefaultConfig\(\)](#) function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

#### Data Fields

- [ctimer\\_timer\\_mode\\_t mode](#)  
*Timer mode.*
- [ctimer\\_capture\\_channel\\_t input](#)  
*Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC.*
- [uint32\\_t prescale](#)  
*Prescale value.*

## 11.5 Enumeration Type Documentation

### 11.5.1 enum ctimer\_capture\_channel\_t

Enumerator

- kCTIMER\_Capture\_0*** Timer capture channel 0.
- kCTIMER\_Capture\_1*** Timer capture channel 1.
- kCTIMER\_Capture\_2*** Timer capture channel 2.
- kCTIMER\_Capture\_3*** Timer capture channel 3.

### 11.5.2 enum ctimer\_capture\_edge\_t

Enumerator

- kCTIMER\_Capture\_RiseEdge*** Capture on rising edge.
- kCTIMER\_Capture\_FallEdge*** Capture on falling edge.
- kCTIMER\_Capture\_BothEdge*** Capture on rising and falling edge.

### 11.5.3 enum ctimer\_match\_t

Enumerator

- kCTIMER\_Match\_0*** Timer match register 0.
- kCTIMER\_Match\_1*** Timer match register 1.
- kCTIMER\_Match\_2*** Timer match register 2.

*kCTIMER\_Match\_3* Timer match register 3.

#### 11.5.4 enum `ctimer_external_match_t`

Enumerator

- kCTIMER\_External\_Match\_0* External match 0.
- kCTIMER\_External\_Match\_1* External match 1.
- kCTIMER\_External\_Match\_2* External match 2.
- kCTIMER\_External\_Match\_3* External match 3.

#### 11.5.5 enum `ctimer_match_output_control_t`

Enumerator

- kCTIMER\_Output\_NoAction* No action is taken.
- kCTIMER\_Output\_Clear* Clear the EM bit/output to 0.
- kCTIMER\_Output\_Set* Set the EM bit/output to 1.
- kCTIMER\_Output\_Toggle* Toggle the EM bit/output.

#### 11.5.6 enum `ctimer_interrupt_enable_t`

Enumerator

- kCTIMER\_Match0InterruptEnable* Match 0 interrupt.
- kCTIMER\_Match1InterruptEnable* Match 1 interrupt.
- kCTIMER\_Match2InterruptEnable* Match 2 interrupt.
- kCTIMER\_Match3InterruptEnable* Match 3 interrupt.
- kCTIMER\_Capture0InterruptEnable* Capture 0 interrupt.
- kCTIMER\_Capture1InterruptEnable* Capture 1 interrupt.
- kCTIMER\_Capture2InterruptEnable* Capture 2 interrupt.
- kCTIMER\_Capture3InterruptEnable* Capture 3 interrupt.

#### 11.5.7 enum `ctimer_status_flags_t`

Enumerator

- kCTIMER\_Match0Flag* Match 0 interrupt flag.
- kCTIMER\_Match1Flag* Match 1 interrupt flag.
- kCTIMER\_Match2Flag* Match 2 interrupt flag.

- kCTIMER\_Match3Flag* Match 3 interrupt flag.
- kCTIMER\_Capture0Flag* Capture 0 interrupt flag.
- kCTIMER\_Capture1Flag* Capture 1 interrupt flag.
- kCTIMER\_Capture2Flag* Capture 2 interrupt flag.
- kCTIMER\_Capture3Flag* Capture 3 interrupt flag.

### 11.5.8 enum **ctimer\_callback\_type\_t**

When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Enumerator

- kCTIMER\_SingleCallback* Single Callback type where there is only one callback for the timer.  
based on the status flags different channels needs to be handled differently
- kCTIMER\_MultipleCallback* Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

## 11.6 Function Documentation

### 11.6.1 void **CTIMER\_Init** ( **CTIMER\_Type** \* *base*, **const ctimer\_config\_t** \* *config* )

Note

This API should be called at the beginning of the application before using the driver.

Parameters

<i>base</i>	Ctimer peripheral base address
<i>config</i>	Pointer to the user configuration structure.

### 11.6.2 void **CTIMER\_Deinit** ( **CTIMER\_Type** \* *base* )

Parameters

<i>base</i>	Ctimer peripheral base address
-------------	--------------------------------

### 11.6.3 void **CTIMER\_GetDefaultConfig** ( **ctimer\_config\_t** \* *config* )

The default values are:

```
* config->mode = kCTIMER_TimerMode;
* config->input = kCTIMER_Capture_0;
* config->prescale = 0;
*
```

## Parameters

<i>config</i>	Pointer to the user configuration structure.
---------------	--

#### 11.6.4 status\_t CTIMER\_SetupPwmPeriod ( **CTIMER\_Type** \* *base*, const **ctimer\_match\_t** *pwmPeriodChannel*, **ctimer\_match\_t** *matchChannel*, **uint32\_t** *pwmPeriod*, **uint32\_t** *pulsePeriod*, **bool** *enableInt* )

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

## Note

When setting PWM output from multiple output pins, all should use the same PWM period

## Parameters

<i>base</i>	Ctimer peripheral base address
<i>pwmPeriod-Channel</i>	Specify the channel to control the PWM period
<i>matchChannel</i>	Match pin to be used to output the PWM signal
<i>pwmPeriod</i>	PWM period match value
<i>pulsePeriod</i>	Pulse width match value
<i>enableInt</i>	Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

#### 11.6.5 status\_t CTIMER\_SetupPwm ( **CTIMER\_Type** \* *base*, const **ctimer\_match\_t** *pwmPeriodChannel*, **ctimer\_match\_t** *matchChannel*, **uint8\_t** *dutyCyclePercent*, **uint32\_t** *pwmFreq\_Hz*, **uint32\_t** *srcClock\_Hz*, **bool** *enableInt* )

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

## Note

When setting PWM output from multiple output pins, all should use the same PWM frequency. Please use CTIMER\_SetupPwmPeriod to set up the PWM with high resolution.

## Parameters

<i>base</i>	Ctimer peripheral base address
<i>pwmPeriod-Channel</i>	Specify the channel to control the PWM period
<i>matchChannel</i>	Match pin to be used to output the PWM signal
<i>dutyCycle-Percent</i>	PWM pulse width; the value should be between 0 to 100
<i>pwmFreq_Hz</i>	PWM signal frequency in Hz
<i>srcClock_Hz</i>	Timer counter clock in Hz
<i>enableInt</i>	Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

**11.6.6 static void CTIMER\_UpdatePwmPulsePeriod ( CTIMER\_Type \* *base*, ctimer\_match\_t *matchChannel*, uint32\_t *pulsePeriod* ) [inline], [static]**

## Parameters

<i>base</i>	Ctimer peripheral base address
<i>matchChannel</i>	Match pin to be used to output the PWM signal
<i>pulsePeriod</i>	New PWM pulse width match value

**11.6.7 void CTIMER\_UpdatePwmDutycycle ( CTIMER\_Type \* *base*, const ctimer\_match\_t *pwmPeriodChannel*, ctimer\_match\_t *matchChannel*, uint8\_t *dutyCyclePercent* )**

## Note

Please use CTIMER\_SetupPwmPeriod to update the PWM with high resolution. This function can manually assign the specified channel to set the PWM cycle.

Parameters

<i>base</i>	Ctimer peripheral base address
<i>pwmPeriod-Channel</i>	Specify the channel to control the PWM period
<i>matchChannel</i>	Match pin to be used to output the PWM signal
<i>dutyCycle-Percent</i>	New PWM pulse width; the value should be between 0 to 100

#### 11.6.8 void CTIMER\_SetupMatch ( **CTIMER\_Type** \* *base*, **ctimer\_match\_t** *matchChannel*, **const ctimer\_match\_config\_t** \* *config* )

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

<i>base</i>	Ctimer peripheral base address
<i>matchChannel</i>	Match register to configure
<i>config</i>	Pointer to the match configuration structure

#### 11.6.9 uint32\_t CTIMER\_GetOutputMatchStatus ( **CTIMER\_Type** \* *base*, **uint32\_t** *matchChannel* )

This function gets the status of output MAT, whether or not this output is connected to a pin. This status is driven to the MAT pins if the match function is selected via IOCON. 0 = LOW. 1 = HIGH.

Parameters

<i>base</i>	Ctimer peripheral base address
<i>matchChannel</i>	External match channel, user can obtain the status of multiple match channels at the same time by using the logic of " " enumeration <b>ctimer_external_match_t</b>

Returns

The mask of external match channel status flags. Users need to use the \_ctimer\_external\_match type to decode the return variables.

**11.6.10 void CTIMER\_SetupCapture ( CTIMER\_Type \* *base*, ctimer\_capture\_channel\_t *capture*, ctimer\_capture\_edge\_t *edge*, bool *enableInt* )**

Parameters

<i>base</i>	Ctimer peripheral base address
<i>capture</i>	Capture channel to configure
<i>edge</i>	Edge on the channel that will trigger a capture
<i>enableInt</i>	Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

**11.6.11 static uint32\_t CTIMER\_GetTimerCountValue ( CTIMER\_Type \* *base* )  
[inline], [static]**

Parameters

<i>base</i>	Ctimer peripheral base address.
-------------	---------------------------------

Returns

return the timer count value.

**11.6.12 void CTIMER\_RegisterCallBack ( CTIMER\_Type \* *base*, ctimer\_callback\_t  
\* *cb\_func*, ctimer\_callback\_type\_t *cb\_type* )**

Parameters

<i>base</i>	Ctimer peripheral base address
<i>cb_func</i>	callback function
<i>cb_type</i>	callback function type, singular or multiple

**11.6.13 static void CTIMER\_EnableInterrupts ( CTIMER\_Type \* *base*, uint32\_t  
*mask* ) [inline], [static]**

Parameters

<i>base</i>	Ctimer peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">ctimer_interrupt_enable_t</a>

#### 11.6.14 static void CTIMER\_DisableInterrupts ( **CTIMER\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	Ctimer peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">ctimer_interrupt_enable_t</a>

#### 11.6.15 static **uint32\_t** CTIMER\_GetEnabledInterrupts ( **CTIMER\_Type** \* *base* ) [inline], [static]

Parameters

<i>base</i>	Ctimer peripheral base address
-------------	--------------------------------

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [ctimer\\_interrupt\\_enable\\_t](#)

#### 11.6.16 static **uint32\_t** CTIMER\_GetStatusFlags ( **CTIMER\_Type** \* *base* ) [inline], [static]

Parameters

<i>base</i>	Ctimer peripheral base address
-------------	--------------------------------

Returns

The status flags. This is the logical OR of members of the enumeration [ctimer\\_status\\_flags\\_t](#)

11.6.17 **static void CTIMER\_ClearStatusFlags( CTIMER\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	Ctimer peripheral base address
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration <a href="#">ctimer_status_flags_t</a>

#### 11.6.18 static void CTIMER\_StartTimer ( **CTIMER\_Type** \* *base* ) [inline], [static]

Parameters

<i>base</i>	Ctimer peripheral base address
-------------	--------------------------------

#### 11.6.19 static void CTIMER\_StopTimer ( **CTIMER\_Type** \* *base* ) [inline], [static]

Parameters

<i>base</i>	Ctimer peripheral base address
-------------	--------------------------------

#### 11.6.20 static void CTIMER\_Reset ( **CTIMER\_Type** \* *base* ) [inline], [static]

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

Parameters

<i>base</i>	Ctimer peripheral base address
-------------	--------------------------------

#### 11.6.21 static void CTIMER\_SetPrescale ( **CTIMER\_Type** \* *base*, **uint32\_t** *prescale* ) [inline], [static]

Specifies the maximum value for the Prescale Counter.

Parameters

<i>base</i>	Ctimer peripheral base address
<i>prescale</i>	Prescale value

#### 11.6.22 static uint32\_t CTIMER\_GetCaptureValue ( CTIMER\_Type \* *base*, ctimer\_capture\_channel\_t *capture* ) [inline], [static]

Get the counter/timer value on the corresponding capture channel.

Parameters

<i>base</i>	Ctimer peripheral base address
<i>capture</i>	Select capture channel

Returns

The timer count capture value.

#### 11.6.23 static void CTIMER\_EnableResetMatchChannel ( CTIMER\_Type \* *base*, ctimer\_match\_t *match*, bool *enable* ) [inline], [static]

Set the specified match channel reset operation.

Parameters

<i>base</i>	Ctimer peripheral base address
<i>match</i>	match channel used
<i>enable</i>	Enable match channel reset operation.

#### 11.6.24 static void CTIMER\_EnableStopMatchChannel ( CTIMER\_Type \* *base*, ctimer\_match\_t *match*, bool *enable* ) [inline], [static]

Set the specified match channel stop operation.

Parameters

<i>base</i>	Ctimer peripheral base address.
<i>match</i>	match channel used.
<i>enable</i>	Enable match channel stop operation.

### 11.6.25 static void CTIMER\_EnableRisingEdgeCapture ( **CTIMER\_Type** \* *base*, **ctimer\_capture\_channel\_t** *capture*, **bool** *enable* ) [inline], [static]

Sets the specified capture channel for rising edge capture.

Parameters

<i>base</i>	Ctimer peripheral base address.
<i>capture</i>	capture channel used.
<i>enable</i>	Enable rising edge capture.

### 11.6.26 static void CTIMER\_EnableFallingEdgeCapture ( **CTIMER\_Type** \* *base*, **ctimer\_capture\_channel\_t** *capture*, **bool** *enable* ) [inline], [static]

Sets the specified capture channel for falling edge capture.

Parameters

<i>base</i>	Ctimer peripheral base address.
<i>capture</i>	capture channel used.
<i>enable</i>	Enable falling edge capture.

# Chapter 12

## FLEXCOMM: FLEXCOMM Driver

### 12.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FLEXCOMM drivers for the FLEXCOMM module of MCUXpresso SDK devices.

### Modules

- [FLEXCOMM Driver](#)

## 12.2 FLEXCOMM Driver

### 12.2.1 Overview

#### Typedefs

- `typedef void(* flexcomm_irq_handler_t )(void *base, void *handle)`  
*Typedef for interrupt handler.*

#### Enumerations

- `enum FLEXCOMM_PERIPH_T {  
 FLEXCOMM_PERIPH_NONE,  
 FLEXCOMM_PERIPH_USART,  
 FLEXCOMM_PERIPH_SPI,  
 FLEXCOMM_PERIPH_I2C,  
 FLEXCOMM_PERIPH_I2S_TX,  
 FLEXCOMM_PERIPH_I2S_RX }`  
*FLEXCOMM peripheral modes.*

#### Functions

- `uint32_t FLEXCOMM_GetInstance (void *base)`  
*Returns instance number for FLEXCOMM module with given base address.*
- `status_t FLEXCOMM_Init (void *base, FLEXCOMM_PERIPH_T periph)`  
*Initializes FLEXCOMM and selects peripheral mode according to the second parameter.*
- `void FLEXCOMM_SetIRQHandler (void *base, flexcomm_irq_handler_t handler, void *flexcommHandle)`  
*Sets IRQ handler for given FLEXCOMM module.*

#### Variables

- `IRQn_Type const kFlexcommIrqs []`  
*Array with IRQ number for each FLEXCOMM module.*

#### Driver version

- `#define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`  
*FlexCOMM driver version 2.0.2.*

### 12.2.2 Macro Definition Documentation

**12.2.2.1 #define FSL\_FLEXCOMM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))**

## 12.2.3 Typedef Documentation

**12.2.3.1 typedef void(\* flexcomm\_irq\_handler\_t)(void \*base, void \*handle)**

## 12.2.4 Enumeration Type Documentation

**12.2.4.1 enum FLEXCOMM\_PERIPH\_T**

Enumerator

*FLEXCOMM\_PERIPH\_NONE* No peripheral.

*FLEXCOMM\_PERIPH\_USART* USART peripheral.

*FLEXCOMM\_PERIPH\_SPI* SPI Peripheral.

*FLEXCOMM\_PERIPH\_I2C* I2C Peripheral.

*FLEXCOMM\_PERIPH\_I2S\_TX* I2S TX Peripheral.

*FLEXCOMM\_PERIPH\_I2S\_RX* I2S RX Peripheral.

## 12.2.5 Function Documentation

**12.2.5.1 uint32\_t FLEXCOMM\_GetInstance ( void \* *base* )**

**12.2.5.2 status\_t FLEXCOMM\_Init ( void \* *base*, FLEXCOMM\_PERIPH\_T *periph* )**

**12.2.5.3 void FLEXCOMM\_SetIRQHandler ( void \* *base*, flexcomm\_irq\_handler\_t *handler*, void \* *flexcommHandle* )**

It is used by drivers register IRQ handler according to FLEXCOMM mode

## 12.2.6 Variable Documentation

**12.2.6.1 IRQn\_Type const kFlexcommIrqs[]**

# Chapter 13

## I2C: Inter-Integrated Circuit Driver

### 13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of MCUXpresso SDK devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions [I2C\\_MasterTransferNonBlocking\(\)](#) set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

### 13.2 Typical use case

#### 13.2.1 Master Operation in functional method

```
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

/* Send start and slave address. */
I2C_MasterStart(EXAMPLE_I2C_MASTER_BASEADDR, 7-bit slave address,
    kI2C_Write/kI2C_Read);

/* Wait address sent out. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR)) & kI2C_IntPendingFlag))
{
}

if(status & kI2C_ReceiveNakFlag)
{
    return kStatus_I2C_Nak;
```

```

}

result = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER_BASEADDR, txBuff, BUFFER_SIZE,
                                kI2C_TransferDefaultFlag);

if(result)
{
    /* If error occurs, send STOP. */
    I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
    return result;
}

while(!(I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR) & kI2C_IntPendingFlag))
{
}

/* Wait all data sent out, send STOP. */
I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);

```

### 13.2.2 Master Operation in interrupt transactional method

```

i2c_master_handle_t g_m_handle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_handle_t *handle,
                               status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

I2C_MasterTransferCreateHandle(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle,
                             i2c_master_callback, NULL);
I2C_MasterTransferNonBlocking(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle, &
                           masterXfer);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;

```

### 13.2.3 Master Operation in DMA transactional method

```
i2c_master_dma_handle_t g_m_dma_handle;
dma_handle_t dmaHandle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_dma_handle_t *handle,
    status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

DMA_EnableChannel(EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);
DMA_CreateHandle(&dmaHandle, EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);

I2C_MasterTransferCreateHandleDMA(EXAMPLE_I2C_MASTER_BASEADDR, &
    g_m_dma_handle, i2c_master_callback, NULL, &dmaHandle);
I2C_MasterTransferDMA(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_dma_handle, &masterXfer);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

### 13.2.4 Slave Operation in functional method

```
i2c_slave_config_t slaveConfig;
uint8_t status;
status_t result = kStatus_Success;

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
    mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;
I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

/* Wait address match. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_SLAVE_BASEADDR)) & kI2C_AddressMatchFlag))
{}
```

```

/* Slave transmit, master reading from slave. */
if (status & kI2C_TransferDirectionFlag)
{
    result = I2C_SlaveWriteBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}
else
{
    I2C_SlaveReadBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}

return result;

```

### 13.2.5 Slave Operation in interrupt transactional method

```

i2c_slave_config_t slaveConfig;
i2c_slave_handle_t g_s_handle;
volatile bool g_SlaveCompletionFlag = false;

static void i2c_slave_callback(I2C_Type *base, i2c_slave_transfer_t *xfer, void *
                               userData)
{
    switch (xfer->event)
    {
        /* Transmit request */
        case kI2C_SlaveTransmitEvent:
            /* Update information for transmit process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /* Receive request */
        case kI2C_SlaveReceiveEvent:
            /* Update information for received process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /* Transfer done */
        case kI2C_SlaveCompletionEvent:
            g_SlaveCompletionFlag = true;
            break;

        default:
            g_SlaveCompletionFlag = true;
            break;
    }
}

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
                                           mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;

I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

I2C_SlaveTransferCreateHandle(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
                             i2c_slave_callback, NULL);

I2C_SlaveTransferNonBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
                            kI2C_SlaveCompletionEvent);

/* Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}

```

```
g_SlaveCompletionFlag = false;
```

## Modules

- I2C CMSIS Driver
- I2C DMA Driver
- I2C Driver
- I2C FreeRTOS Driver
- I2C Master Driver
- I2C Slave Driver

## 13.3 I2C Driver

### 13.3.1 Overview

#### Files

- file [fsl\\_i2c.h](#)

#### Macros

- `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`  
*Retry times for waiting flag.*
- `#define I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK 1U /* Define to one means master ignores the last byte's nack and considers the transfer successful. */`  
*Whether to ignore the nack signal of the last byte during master transmit.*
- `#define I2C_STAT_MSTCODE_IDLE (0U)`  
*Master Idle State Code.*
- `#define I2C_STAT_MSTCODE_RXREADY (1U)`  
*Master Receive Ready State Code.*
- `#define I2C_STAT_MSTCODE_TXREADY (2U)`  
*Master Transmit Ready State Code.*
- `#define I2C_STAT_MSTCODE_NACKADR (3U)`  
*Master NACK by slave on address State Code.*
- `#define I2C_STAT_MSTCODE_NACKDAT (4U)`  
*Master NACK by slave on data State Code.*

#### Enumerations

- enum {
 `kStatus_I2C_Busy` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 0),
 `kStatus_I2C_Idle` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 1),
 `kStatus_I2C_Nak`,
 `kStatus_I2C_InvalidParameter`,
 `kStatus_I2C_BitError` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 4),
 `kStatus_I2C_ArbitrationLost` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 5),
 `kStatus_I2C_NoTransferInProgress`,
 `kStatus_I2C_DmaRequestFail` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 7),
 `kStatus_I2C_StartStopError` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 8),
 `kStatus_I2C_UnexpectedState` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 9),
 `kStatus_I2C_Timeout`,
 `kStatus_I2C_Addr_Nak` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 11),
 `kStatus_I2C_EventTimeout` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 12),
 `kStatus_I2C_SclLowTimeout` = MAKE\_STATUS(kStatusGroup\_FLEXCOMM\_I2C, 13) }
   
*I2C status return codes.*

- enum `_i2c_status_flags` {
   
 kI2C\_MasterPendingFlag = I2C\_STAT\_MSTPENDING\_MASK,
   
 kI2C\_MasterArbitrationLostFlag,
   
 kI2C\_MasterStartStopErrorFlag,
   
 kI2C\_MasterIdleFlag = 1UL << 5U,
   
 kI2C\_MasterRxReadyFlag = 1UL << I2C\_STAT\_MSTSTATE\_SHIFT,
   
 kI2C\_MasterTxReadyFlag = 1UL << (I2C\_STAT\_MSTSTATE\_SHIFT + 1U),
   
 kI2C\_MasterAddrNackFlag = 1UL << 7U,
   
 kI2C\_MasterDataNackFlag = 1UL << (I2C\_STAT\_MSTSTATE\_SHIFT + 2U),
   
 kI2C\_SlavePendingFlag = I2C\_STAT\_SLVPENDING\_MASK,
   
 kI2C\_SlaveNotStretching = I2C\_STAT\_SLVNOTSTR\_MASK,
   
 kI2C\_SlaveSelected,
   
 kI2C\_SaveDeselected = I2C\_STAT\_SLVDESEL\_MASK,
   
 kI2C\_SlaveAddressedFlag = 1UL << 22U,
   
 kI2C\_SlaveReceiveFlag = 1UL << I2C\_STAT\_SLVSTATE\_SHIFT,
   
 kI2C\_SlaveTransmitFlag = 1UL << (I2C\_STAT\_SLVSTATE\_SHIFT + 1U),
   
 kI2C\_SlaveAddress0MatchFlag = 1UL << 20U,
   
 kI2C\_SlaveAddress1MatchFlag = 1UL << I2C\_STAT\_SLVIDX\_SHIFT,
   
 kI2C\_SlaveAddress2MatchFlag = 1UL << (I2C\_STAT\_SLVIDX\_SHIFT + 1U),
   
 kI2C\_SlaveAddress3MatchFlag = 1UL << 21U,
   
 kI2C\_MonitorReadyFlag = I2C\_STAT\_MONRDY\_MASK,
   
 kI2C\_MonitorOverflowFlag = I2C\_STAT\_MONOV\_MASK,
   
 kI2C\_MonitorActiveFlag = I2C\_STAT\_MONACTIVE\_MASK,
   
 kI2C\_MonitorIdleFlag = I2C\_STAT\_MONIDLE\_MASK,
   
 kI2C\_EventTimeoutFlag = I2C\_STAT\_EVENTTIMEOUT\_MASK,
   
 kI2C\_SclTimeoutFlag = I2C\_STAT\_SCLTIMEOUT\_MASK }

*I2C status flags.*

- enum `_i2c_interrupt_enable` {
   
 kI2C\_MasterPendingInterruptEnable,
   
 kI2C\_MasterArbitrationLostInterruptEnable,
   
 kI2C\_MasterStartStopErrorInterruptEnable,
   
 kI2C\_SlavePendingInterruptEnable = I2C\_STAT\_SLVPENDING\_MASK,
   
 kI2C\_SlaveNotStretchingInterruptEnable,
   
 kI2C\_SlaveDeselectedInterruptEnable = I2C\_STAT\_SLVDESEL\_MASK,
   
 kI2C\_MonitorReadyInterruptEnable = I2C\_STAT\_MONRDY\_MASK,
   
 kI2C\_MonitorOverflowInterruptEnable = I2C\_STAT\_MONOV\_MASK,
   
 kI2C\_MonitorIdleInterruptEnable = I2C\_STAT\_MONIDLE\_MASK,
   
 kI2C\_EventTimeoutInterruptEnable = I2C\_STAT\_EVENTTIMEOUT\_MASK,
   
 kI2C\_SclTimeoutInterruptEnable = I2C\_STAT\_SCLTIMEOUT\_MASK }

*I2C interrupt enable.*

## Driver version

- #define `FSL_I2C_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 1)`)

*I2C driver version.*

### 13.3.2 Macro Definition Documentation

**13.3.2.1 #define FSL\_I2C\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))**

**13.3.2.2 #define I2C\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

**13.3.2.3 #define I2C\_MASTER\_TRANSMIT\_IGNORE\_LAST\_NACK 1U /\* Define to one means master ignores the last byte's nack and considers the transfer successful. \*/**

### 13.3.3 Enumeration Type Documentation

#### 13.3.3.1 anonymous enum

Enumerator

***kStatus\_I2C\_Busy*** The master is already performing a transfer.

***kStatus\_I2C\_Idle*** The slave driver is idle.

***kStatus\_I2C\_Nak*** The slave device sent a NAK in response to a byte.

***kStatus\_I2C\_InvalidParameter*** Unable to proceed due to invalid parameter.

***kStatus\_I2C\_BitError*** Transferred bit was not seen on the bus.

***kStatus\_I2C\_ArbitrationLost*** Arbitration lost error.

***kStatus\_I2C\_NoTransferInProgress*** Attempt to abort a transfer when one is not in progress.

***kStatus\_I2C\_DmaRequestFail*** DMA request failed.

***kStatus\_I2C\_StartStopError*** Start and stop error.

***kStatus\_I2C\_UnexpectedState*** Unexpected state.

***kStatus\_I2C\_Timeout*** Timeout when waiting for I2C master/slave pending status to set to continue transfer.

***kStatus\_I2C\_Addr\_Nak*** NAK received for Address.

***kStatus\_I2C\_EventTimeout*** Timeout waiting for bus event.

***kStatus\_I2C\_SclLowTimeout*** Timeout SCL signal remains low.

#### 13.3.3.2 enum \_i2c\_status\_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

***kI2C\_MasterPendingFlag*** The I2C module is waiting for software interaction. bit 0

***kI2C\_MasterArbitrationLostFlag*** The arbitration of the bus was lost. There was collision on the bus. bit 4

***kI2C\_MasterStartStopErrorFlag*** There was an error during start or stop phase of the transaction. bit 6

***kI2C\_MasterIdleFlag*** The I2C master idle status. bit 5

***kI2C\_MasterRxReadyFlag*** The I2C master rx ready status. bit 1

***kI2C\_MasterTxReadyFlag*** The I2C master tx ready status. bit 2

***kI2C\_MasterAddrNackFlag*** The I2C master address nack status. bit 7

***kI2C\_MasterDataNackFlag*** The I2C master data nack status. bit 3

***kI2C\_SlavePendingFlag*** The I2C module is waiting for software interaction. bit 8

***kI2C\_SlaveNotStretching*** Indicates whether the slave is currently stretching clock (0 = yes, 1 = no). bit 11

***kI2C\_SlaveSelected*** Indicates whether the slave is selected by an address match. bit 14

***kI2C\_SaveDeselected*** Indicates that slave was previously deselected (deselect event took place, w1c). bit 15

***kI2C\_SlaveAddressedFlag*** One of the I2C slave's 4 addresses is matched. bit 22

***kI2C\_SlaveReceiveFlag*** Slave receive data available. bit 9

***kI2C\_SlaveTransmitFlag*** Slave data can be transmitted. bit 10

***kI2C\_SlaveAddress0MatchFlag*** Slave address0 match. bit 20

***kI2C\_SlaveAddress1MatchFlag*** Slave address1 match. bit 12

***kI2C\_SlaveAddress2MatchFlag*** Slave address2 match. bit 13

***kI2C\_SlaveAddress3MatchFlag*** Slave address3 match. bit 21

***kI2C\_MonitorReadyFlag*** The I2C monitor ready interrupt. bit 16

***kI2C\_MonitorOverflowFlag*** The monitor data overrun interrupt. bit 17

***kI2C\_MonitorActiveFlag*** The monitor is active. bit 18

***kI2C\_MonitorIdleFlag*** The monitor idle interrupt. bit 19

***kI2C\_EventTimeoutFlag*** The bus event timeout interrupt. bit 24

***kI2C\_SclTimeoutFlag*** The SCL timeout interrupt. bit 25

### 13.3.3.3 enum \_i2c\_interrupt\_enable

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

***kI2C\_MasterPendingInterruptEnable*** The I2C master communication pending interrupt.

***kI2C\_MasterArbitrationLostInterruptEnable*** The I2C master arbitration lost interrupt.

***kI2C\_MasterStartStopErrorInterruptEnable*** The I2C master start/stop timing error interrupt.

***kI2C\_SlavePendingInterruptEnable*** The I2C slave communication pending interrupt.

***kI2C\_SlaveNotStretchingInterruptEnable*** The I2C slave not streching interrupt, deep-sleep mode can be entered only when this interrupt occurs.

***kI2C\_SlaveDeselectedInterruptEnable*** The I2C slave deselection interrupt.

***kI2C\_MonitorReadyInterruptEnable*** The I2C monitor ready interrupt.

***kI2C\_MonitorOverflowInterruptEnable*** The monitor data overrun interrupt.

***kI2C\_MonitorIdleInterruptEnable*** The monitor idle interrupt.

***kI2C\_EventTimeoutInterruptEnable*** The bus event timeout interrupt.

***kI2C\_SclTimeoutInterruptEnable*** The SCL timeout interrupt.

## 13.4 I2C Master Driver

### 13.4.1 Overview

#### Data Structures

- struct `i2c_master_config_t`  
*Structure with settings to initialize the I2C master module. [More...](#)*
- struct `i2c_master_transfer_t`  
*Non-blocking transfer descriptor structure. [More...](#)*
- struct `i2c_master_handle_t`  
*Driver handle for master non-blocking APIs. [More...](#)*

#### Typedefs

- typedef void(\* `i2c_master_transfer_callback_t`)  
(I2C\_Type \*base, i2c\_master\_handle\_t \*handle,  
`status_t` completionStatus, void \*userData)  
*Master completion callback function pointer type.*

#### Enumerations

- enum `i2c_direction_t` {  
  kI2C\_Write = 0U,  
  kI2C\_Read = 1U }  
*Direction of master and slave transfers.*
- enum `_i2c_master_transfer_flags` {  
  kI2C\_TransferDefaultFlag = 0x00U,  
  kI2C\_TransferNoStartFlag = 0x01U,  
  kI2C\_TransferRepeatedStartFlag = 0x02U,  
  kI2C\_TransferNoStopFlag = 0x04U }  
*Transfer option flags.*
- enum `_i2c_transfer_states`  
*States for the state machine used by transactional APIs.*

#### Initialization and deinitialization

- void `I2C_MasterGetDefaultConfig` (`i2c_master_config_t` \*masterConfig)  
*Provides a default configuration for the I2C master peripheral.*
- void `I2C_MasterInit` (I2C\_Type \*base, const `i2c_master_config_t` \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the I2C master peripheral.*
- void `I2C_MasterDeinit` (I2C\_Type \*base)  
*Deinitializes the I2C master peripheral.*
- uint32\_t `I2C_GetInstance` (I2C\_Type \*base)  
*Returns an instance number given a base address.*

- static void [I2C\\_MasterReset](#) (I2C\_Type \*base)  
*Performs a software reset.*
- static void [I2C\\_MasterEnable](#) (I2C\_Type \*base, bool enable)  
*Enables or disables the I2C module as master.*

## Status

- uint32\_t [I2C\\_GetStatusFlags](#) (I2C\_Type \*base)  
*Gets the I2C status flags.*
- static void [I2C\\_ClearStatusFlags](#) (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C status flag state.*
- static void [I2C\\_MasterClearStatusFlags](#) (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C master status flag state.*

## Interrupts

- static void [I2C\\_EnableInterrupts](#) (I2C\_Type \*base, uint32\_t interruptMask)  
*Enables the I2C interrupt requests.*
- static void [I2C\\_DisableInterrupts](#) (I2C\_Type \*base, uint32\_t interruptMask)  
*Disables the I2C interrupt requests.*
- static uint32\_t [I2C\\_GetEnabledInterrupts](#) (I2C\_Type \*base)  
*Returns the set of currently enabled I2C interrupt requests.*

## Bus operations

- void [I2C\\_MasterSetBaudRate](#) (I2C\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the I2C bus frequency for master transactions.*
- void [I2C\\_MasterSetTimeoutValue](#) (I2C\_Type \*base, uint8\_t timeout\_Ms, uint32\_t srcClock\_Hz)  
*Sets the I2C bus timeout value.*
- static bool [I2C\\_MasterGetBusIdleState](#) (I2C\_Type \*base)  
*Returns whether the bus is idle.*
- [status\\_t I2C\\_MasterStart](#) (I2C\_Type \*base, uint8\_t address, [i2c\\_direction\\_t](#) direction)  
*Sends a START on the I2C bus.*
- [status\\_t I2C\\_MasterStop](#) (I2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- static [status\\_t I2C\\_MasterRepeatedStart](#) (I2C\_Type \*base, uint8\_t address, [i2c\\_direction\\_t](#) direction)  
*Sends a REPEATED START on the I2C bus.*
- [status\\_t I2C\\_MasterWriteBlocking](#) (I2C\_Type \*base, const void \*txBuff, size\_t txSize, uint32\_t flags)  
*Performs a polling send transfer on the I2C bus.*
- [status\\_t I2C\\_MasterReadBlocking](#) (I2C\_Type \*base, void \*rxBuff, size\_t rxSize, uint32\_t flags)  
*Performs a polling receive transfer on the I2C bus.*
- [status\\_t I2C\\_MasterTransferBlocking](#) (I2C\_Type \*base, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a master polling transfer on the I2C bus.*

## Non-blocking

- void [I2C\\_MasterTransferCreateHandle](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, [i2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the I2C master non-blocking APIs.*
- status\_t [I2C\\_MasterTransferNonBlocking](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking transaction on the I2C bus.*
- status\_t [I2C\\_MasterTransferGetCount](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, size\_t \*count)  
*Returns number of bytes transferred so far.*
- status\_t [I2C\\_MasterTransferAbort](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle)  
*Terminates a non-blocking I2C master transmission early.*

## IRQ handler

- void [I2C\\_MasterTransferHandleIRQ](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle)  
*Reusable routine to handle master interrupts.*

### 13.4.2 Data Structure Documentation

#### 13.4.2.1 struct i2c\_master\_config\_t

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the [I2C\\_MasterGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

#### Data Fields

- bool [enableMaster](#)  
*Whether to enable master mode.*
- uint32\_t [baudRate\\_Bps](#)  
*Desired baud rate in bits per second.*
- bool [enableTimeout](#)  
*Enable internal timeout function.*
- uint8\_t [timeout\\_Ms](#)  
*Event timeout and SCL low timeout value.*

#### Field Documentation

- (1) **bool i2c\_master\_config\_t::enableMaster**
- (2) **uint32\_t i2c\_master\_config\_t::baudRate\_Bps**

- (3) `bool i2c_master_config_t::enableTimeout`
- (4) `uint8_t i2c_master_config_t::timeout_Ms`

### 13.4.2.2 struct \_i2c\_master\_transfer

I2C master transfer typedef.

This structure is used to pass transaction parameters to the [I2C\\_MasterTransferNonBlocking\(\)](#) API.

#### Data Fields

- `uint32_t flags`  
*Bit mask of options for the transfer.*
- `uint8_t slaveAddress`  
*The 7-bit slave address.*
- `i2c_direction_t direction`  
*Either kI2C\_Read or kI2C\_Write.*
- `uint32_t subaddress`  
*Sub address.*
- `size_t subaddressSize`  
*Length of sub address to send in bytes.*
- `void *data`  
*Pointer to data to transfer.*
- `size_t dataSize`  
*Number of bytes to transfer.*

#### Field Documentation

- (1) `uint32_t i2c_master_transfer_t::flags`

See enumeration [\\_i2c\\_master\\_transfer\\_flags](#) for available options. Set to 0 or [kI2C\\_TransferDefaultFlag](#) for normal transfers.

- (2) `uint8_t i2c_master_transfer_t::slaveAddress`
- (3) `i2c_direction_t i2c_master_transfer_t::direction`
- (4) `uint32_t i2c_master_transfer_t::subaddress`

Transferred MSB first.

- (5) `size_t i2c_master_transfer_t::subaddressSize`  
Maximum size is 4 bytes.
- (6) `void* i2c_master_transfer_t::data`
- (7) `size_t i2c_master_transfer_t::dataSize`

### 13.4.2.3 struct \_i2c\_master\_handle

I2C master handle typedef.

Note

The contents of this structure are private and subject to change.

#### Data Fields

- `uint8_t state`  
*Transfer state machine current state.*
- `uint32_t transferCount`  
*Indicates progress of the transfer.*
- `uint32_t remainingBytes`  
*Remaining byte count in current state.*
- `uint8_t *buf`  
*Buffer pointer for current state.*
- `bool checkAddrNack`  
*Whether to check the nack signal is detected during addressing.*
- `i2c_master_transfer_t transfer`  
*Copy of the current transfer info.*
- `i2c_master_transfer_callback_t completionCallback`  
*Callback function pointer.*
- `void *userData`  
*Application data passed to callback.*

#### Field Documentation

- (1) `uint8_t i2c_master_handle_t::state`
- (2) `uint32_t i2c_master_handle_t::remainingBytes`
- (3) `uint8_t* i2c_master_handle_t::buf`
- (4) `bool i2c_master_handle_t::checkAddrNack`
- (5) `i2c_master_transfer_t i2c_master_handle_t::transfer`
- (6) `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`
- (7) `void* i2c_master_handle_t::userData`

### 13.4.3 Typedef Documentation

```
13.4.3.1 typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base,  
          i2c_master_handle_t *handle, status_t completionStatus, void *userData)
```

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to [I2C\\_MasterTransferCreateHandle\(\)](#).

## Parameters

<i>base</i>	The I2C peripheral base address.
<i>completionStatus</i>	Either kStatus_Success or an error code describing how the transfer completed.
<i>userData</i>	Arbitrary pointer-sized value passed from the application.

## 13.4.4 Enumeration Type Documentation

### 13.4.4.1 enum i2c\_direction\_t

Enumerator

*kI2C\_Write* Master transmit.

*kI2C\_Read* Master receive.

### 13.4.4.2 enum \_i2c\_master\_transfer\_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the [\\_i2c\\_master\\_transfer::flags](#) field.

Enumerator

*kI2C\_TransferDefaultFlag* Transfer starts with a start signal, stops with a stop signal.

*kI2C\_TransferNoStartFlag* Don't send a start condition, address, and sub address.

*kI2C\_TransferRepeatedStartFlag* Send a repeated start condition.

*kI2C\_TransferNoStopFlag* Don't send a stop condition.

### 13.4.4.3 enum \_i2c\_transfer\_states

## 13.4.5 Function Documentation

### 13.4.5.1 void I2C\_MasterGetDefaultConfig ( *i2c\_master\_config\_t* \* *masterConfig* )

This function provides the following default configuration for the I2C master peripheral:

```
* masterConfig->enableMaster          = true;
* masterConfig->baudRate_Bps         = 100000U;
* masterConfig->enableTimeout        = false;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with [I2C\\_MasterInit\(\)](#).

Parameters

out	<i>masterConfig</i>	User provided configuration structure for default values. Refer to <a href="#">i2c_master_config_t</a> .
-----	---------------------	--

#### 13.4.5.2 void I2C\_MasterInit ( **I2C\_Type** \* *base*, const **i2c\_master\_config\_t** \* *masterConfig*, **uint32\_t** *srcClock\_Hz* )

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>masterConfig</i>	User provided peripheral configuration. Use <a href="#">I2C_MasterGetDefaultConfig()</a> to get a set of defaults that you can override.
<i>srcClock_Hz</i>	Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

#### 13.4.5.3 void I2C\_MasterDeinit ( **I2C\_Type** \* *base* )

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

<i>base</i>	The I2C peripheral base address.
-------------	----------------------------------

#### 13.4.5.4 **uint32\_t** I2C\_GetInstance ( **I2C\_Type** \* *base* )

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

<i>base</i>	The I2C peripheral base address.
-------------	----------------------------------

Returns

I2C instance number starting from 0.

**13.4.5.5 static void I2C\_MasterReset( I2C\_Type \* *base* ) [inline], [static]**

Restores the I2C master peripheral to reset conditions.

Parameters

<i>base</i>	The I2C peripheral base address.
-------------	----------------------------------

### 13.4.5.6 static void I2C\_MasterEnable ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	The I2C peripheral base address.
<i>enable</i>	Pass true to enable or false to disable the specified I2C as master.

### 13.4.5.7 uint32\_t I2C\_GetStatusFlags ( I2C\_Type \* *base* )

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

<i>base</i>	The I2C peripheral base address.
-------------	----------------------------------

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_i2c\\_status\\_flags](#).

### 13.4.5.8 static void I2C\_ClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

Refer to kI2C\_CommonAllClearStatusFlags, kI2C\_MasterAllClearStatusFlags and kI2C\_SlaveAllClearStatusFlags to see the clearable flags. Attempts to clear other flags has no effect.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>statusMask</i>	A bitmask of status flags that are to be cleared. The mask is composed of the members in kI2C_CommonAllClearStatusFlags, kI2C_MasterAllClearStatusFlags and kI2C_SlaveAllClearStatusFlags. You may pass the result of a previous call to <a href="#">I2C_GetStatusFlags()</a> .

See Also

[\\_i2c\\_status\\_flags](#), [\\_i2c\\_master\\_status\\_flags](#) and [\\_i2c\\_slave\\_status\\_flags](#).

#### 13.4.5.9 static void I2C\_MasterClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

**Deprecated** Do not use this function. It has been superceded by [I2C\\_ClearStatusFlags](#) The following status register flags can be cleared:

- [kI2C\\_MasterArbitrationLostFlag](#)
- [kI2C\\_MasterStartStopErrorFlag](#)

Attempts to clear other flags has no effect.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>statusMask</i>	A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_i2c_status_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I2C_GetStatusFlags()</a> .

See Also

[\\_i2c\\_status\\_flags](#).

#### 13.4.5.10 static void I2C\_EnableInterrupts ( I2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Parameters

<i>base</i>	The I2C peripheral base address.
<i>interruptMask</i>	Bit mask of interrupts to enable. See <a href="#">_i2c_interrupt_enable</a> for the set of constants that should be OR'd together to form the bit mask.

#### **13.4.5.11 static void I2C\_DisableInterrupts ( I2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

Parameters

<i>base</i>	The I2C peripheral base address.
<i>interruptMask</i>	Bit mask of interrupts to disable. See <a href="#">_i2c_interrupt_enable</a> for the set of constants that should be OR'd together to form the bit mask.

#### **13.4.5.12 static uint32\_t I2C\_GetEnabledInterrupts ( I2C\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	The I2C peripheral base address.
-------------	----------------------------------

Returns

A bitmask composed of [\\_i2c\\_interrupt\\_enable](#) enumerators OR'd together to indicate the set of enabled interrupts.

#### **13.4.5.13 void I2C\_MasterSetBaudRate ( I2C\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )**

The I2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>srcClock_Hz</i>	I2C functional clock frequency in Hertz.
<i>baudRate_Bps</i>	Requested bus frequency in bits per second.

#### 13.4.5.14 void I2C\_MasterSetTimeoutValue ( I2C\_Type \* *base*, uint8\_t *timeout\_Ms*, uint32\_t *srcClock\_Hz* )

If the SCL signal remains low or bus does not have event longer than the timeout value, kI2C\_SclTimeout-Flag or kI2C\_EventTimeoutFlag is set. This can indicate the bus is held by slave or any fault occurs to the I2C module.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>timeout_Ms</i>	Timeout value in millisecond.
<i>srcClock_Hz</i>	I2C functional clock frequency in Hertz.

#### 13.4.5.15 static bool I2C\_MasterGetBusIdleState ( I2C\_Type \* *base* ) [inline], [static]

Requires the master mode to be enabled.

Parameters

<i>base</i>	The I2C peripheral base address.
-------------	----------------------------------

Return values

<i>true</i>	Bus is busy.
<i>false</i>	Bus is idle.

#### 13.4.5.16 status\_t I2C\_MasterStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* )

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

<i>base</i>	I2C peripheral base pointer
<i>address</i>	7-bit slave device address.
<i>direction</i>	Master transfer directions(transmit/receive).

Return values

<i>kStatus_Success</i>	Successfully send the start signal.
<i>kStatus_I2C_Busy</i>	Current bus is busy.

### 13.4.5.17 status\_t I2C\_MasterStop ( I2C\_Type \* *base* )

Return values

<i>kStatus_Success</i>	Successfully send the stop signal.
<i>kStatus_I2C_Timeout</i>	Send stop signal failed, timeout.

### 13.4.5.18 static status\_t I2C\_MasterRepeatedStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* ) [inline], [static]

Parameters

<i>base</i>	I2C peripheral base pointer
<i>address</i>	7-bit slave device address.
<i>direction</i>	Master transfer directions(transmit/receive).

Return values

<i>kStatus_Success</i>	Successfully send the start signal.
<i>kStatus_I2C_Busy</i>	Current bus is busy but not occupied by current I2C master.

### 13.4.5.19 status\_t I2C\_MasterWriteBlocking ( I2C\_Type \* *base*, const void \* *txBuff*, size\_t *txSize*, uint32\_t *flags* )

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns [kStatus\\_I2C\\_Nak](#).

## Parameters

<i>base</i>	The I2C peripheral base address.
<i>txBuff</i>	The pointer to the data to be transferred.
<i>txSize</i>	The length in bytes of the data to be transferred.
<i>flags</i>	Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

## Return values

<i>kStatus_Success</i>	Data was sent successfully.
<i>kStatus_I2C_Busy</i>	Another master is currently utilizing the bus.
<i>kStatus_I2C_Nak</i>	The slave device sent a NAK in response to a byte.
<i>kStatus_I2C_Arbitration-Lost</i>	Arbitration lost error.

**13.4.5.20 status\_t I2C\_MasterReadBlocking ( I2C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize*, uint32\_t *flags* )**

## Parameters

<i>base</i>	The I2C peripheral base address.
<i>rxBuff</i>	The pointer to the data to be transferred.
<i>rxSize</i>	The length in bytes of the data to be transferred.
<i>flags</i>	Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

## Return values

<i>kStatus_Success</i>	Data was received successfully.
<i>kStatus_I2C_Busy</i>	Another master is currently utilizing the bus.
<i>kStatus_I2C_Nak</i>	The slave device sent a NAK in response to a byte.
<i>kStatus_I2C_Arbitration-Lost</i>	Arbitration lost error.

**13.4.5.21 status\_t I2C\_MasterTransferBlocking ( I2C\_Type \* *base*, i2c\_master\_transfer\_t \* *xfer* )**

## Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

## Parameters

<i>base</i>	I2C peripheral base address.
<i>xfer</i>	Pointer to the transfer structure.

## Return values

<i>kStatus_Success</i>	Successfully complete the data transmission.
<i>kStatus_I2C_Busy</i>	Previous transmission still not finished.
<i>kStatus_I2C_Timeout</i>	Transfer error, wait signal timeout.
<i>kStatus_I2C_Arbitration-Lost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive NAK during transfer.
<i>kStatus_I2C_Addr_Nak</i>	Transfer error, receive NAK during addressing.

### 13.4.5.22 void I2C\_MasterTransferCreateHandle ( *I2C\_Type \* base, i2c\_master\_handle\_t \* handle, i2c\_master\_transfer\_callback\_t callback, void \* userData* )

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [I2C\\_MasterTransferAbort\(\)](#) API shall be called.

## Parameters

	<i>base</i>	The I2C peripheral base address.
<i>out</i>	<i>handle</i>	Pointer to the I2C master driver handle.
	<i>callback</i>	User provided pointer to the asynchronous callback function.
	<i>userData</i>	User provided pointer to the application callback data.

### 13.4.5.23 status\_t I2C\_MasterTransferNonBlocking ( *I2C\_Type \* base, i2c\_master\_handle\_t \* handle, i2c\_master\_transfer\_t \* xfer* )

Parameters

<i>base</i>	The I2C peripheral base address.
<i>handle</i>	Pointer to the I2C master driver handle.
<i>xfer</i>	The pointer to the transfer descriptor.

Return values

<i>kStatus_Success</i>	The transaction was started successfully.
<i>kStatus_I2C_Busy</i>	Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

#### 13.4.5.24 status\_t I2C\_MasterTransferGetCount ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

	<i>base</i>	The I2C peripheral base address.
	<i>handle</i>	Pointer to the I2C master driver handle.
<i>out</i>	<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Return values

<i>kStatus_Success</i>	
<i>kStatus_I2C_Busy</i>	

#### 13.4.5.25 status\_t I2C\_MasterTransferAbort ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle* )

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>handle</i>	Pointer to the I2C master driver handle.

Return values

<i>kStatus_Success</i>	A transaction was successfully aborted.
<i>kStatus_I2C_Timeout</i>	Timeout during polling for flags.

### 13.4.5.26 void I2C\_MasterTransferHandleIRQ ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle* )

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>handle</i>	Pointer to the I2C master driver handle.

## 13.5 I2C Slave Driver

### 13.5.1 Overview

#### Data Structures

- struct [i2c\\_slave\\_address\\_t](#)  
*Data structure with 7-bit Slave address and Slave address disable. [More...](#)*
- struct [i2c\\_slave\\_config\\_t](#)  
*Structure with settings to initialize the I2C slave module. [More...](#)*
- struct [i2c\\_slave\\_transfer\\_t](#)  
*I2C slave transfer structure. [More...](#)*
- struct [i2c\\_slave\\_handle\\_t](#)  
*I2C slave handle structure. [More...](#)*

#### Typedefs

- typedef void(\* [i2c\\_slave\\_transfer\\_callback\\_t](#))[\(I2C\\_Type \\*base, volatile i2c\\_slave\\_transfer\\_t \\*transfer, void \\*userData\)](#)  
*Slave event callback function pointer type.*
- typedef void(\* [flexcomm\\_i2c\\_master\\_irq\\_handler\\_t](#))[\(I2C\\_Type \\*base, i2c\\_master\\_handle\\_t \\*handle\)](#)  
*Typedef for master interrupt handler.*
- typedef void(\* [flexcomm\\_i2c\\_slave\\_irq\\_handler\\_t](#))[\(I2C\\_Type \\*base, i2c\\_slave\\_handle\\_t \\*handle\)](#)  
*Typedef for slave interrupt handler.*

#### Enumerations

- enum [i2c\\_slave\\_address\\_register\\_t](#) {
   
   kI2C\_SlaveAddressRegister0 = 0U,
   
   kI2C\_SlaveAddressRegister1 = 1U,
   
   kI2C\_SlaveAddressRegister2 = 2U,
   
   kI2C\_SlaveAddressRegister3 = 3U
 }  
*I2C slave address register.*
- enum [i2c\\_slave\\_address\\_qual\\_mode\\_t](#) {
   
   kI2C\_QualModeMask = 0U,
   
   kI2C\_QualModeExtend
 }  
*I2C slave address match options.*
- enum [i2c\\_slave\\_bus\\_speed\\_t](#)  
*I2C slave bus speed options.*
- enum [i2c\\_slave\\_transfer\\_event\\_t](#) {
   
   kI2C\_SlaveAddressMatchEvent = 0x01U,
   
   kI2C\_SlaveTransmitEvent = 0x02U,
   
   kI2C\_SlaveReceiveEvent = 0x04U,
   
   kI2C\_SlaveCompletionEvent = 0x20U,
   
   kI2C\_SlaveDeselectedEvent,
 }

- **kI2C\_SlaveAllEvents }**  
*Set of events sent to the callback for non blocking slave transfers.*
- **enum i2c\_slave\_fsm\_t**  
*I2C slave software finite state machine states.*

## Slave initialization and deinitialization

- **void I2C\_SlaveGetDefaultConfig (i2c\_slave\_config\_t \*slaveConfig)**  
*Provides a default configuration for the I2C slave peripheral.*
- **status\_t I2C\_SlaveInit (I2C\_Type \*base, const i2c\_slave\_config\_t \*slaveConfig, uint32\_t srcClock\_Hz)**  
*Initializes the I2C slave peripheral.*
- **void I2C\_SlaveSetAddress (I2C\_Type \*base, i2c\_slave\_address\_register\_t addressRegister, uint8\_t address, bool addressDisable)**  
*Configures Slave Address n register.*
- **void I2C\_SlaveDeinit (I2C\_Type \*base)**  
*Deinitializes the I2C slave peripheral.*
- **static void I2C\_SlaveEnable (I2C\_Type \*base, bool enable)**  
*Enables or disables the I2C module as slave.*

## Slave status

- **static void I2C\_SlaveClearStatusFlags (I2C\_Type \*base, uint32\_t statusMask)**  
*Clears the I2C status flag state.*

## Slave bus operations

- **status\_t I2C\_SlaveWriteBlocking (I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize)**  
*Performs a polling send transfer on the I2C bus.*
- **status\_t I2C\_SlaveReadBlocking (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize)**  
*Performs a polling receive transfer on the I2C bus.*

## Slave non-blocking

- **void I2C\_SlaveTransferCreateHandle (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, i2c\_slave\_transfer\_callback\_t callback, void \*userData)**  
*Creates a new handle for the I2C slave non-blocking APIs.*
- **status\_t I2C\_SlaveTransferNonBlocking (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, uint32\_t eventMask)**  
*Starts accepting slave transfers.*
- **status\_t I2C\_SlaveSetSendBuffer (I2C\_Type \*base, volatile i2c\_slave\_transfer\_t \*transfer, const void \*txData, size\_t txSize, uint32\_t eventMask)**  
*Starts accepting master read from slave requests.*
- **status\_t I2C\_SlaveSetReceiveBuffer (I2C\_Type \*base, volatile i2c\_slave\_transfer\_t \*transfer, void \*rxData, size\_t rxSize, uint32\_t eventMask)**

- Starts accepting master write to slave requests.  
static uint32\_t [I2C\\_SlaveGetReceivedAddress](#) (I2C\_Type \*base, volatile i2c\_slave\_transfer\_t \*transfer)
- Returns the slave address sent by the I2C master.  
void [I2C\\_SlaveTransferAbort](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)
- Aborts the slave non-blocking transfers.  
status\_t [I2C\\_SlaveTransferGetCount](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, size\_t \*count)
- Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.  
*Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.*

## Slave IRQ handler

- void [I2C\\_SlaveTransferHandleIRQ](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)  
*Reusable routine to handle slave interrupts.*

### 13.5.2 Data Structure Documentation

#### 13.5.2.1 struct i2c\_slave\_address\_t

##### Data Fields

- uint8\_t [address](#)  
7-bit Slave address SLVADR.
- bool [addressDisable](#)  
Slave address disable SADISABLE.

##### Field Documentation

- (1) [uint8\\_t i2c\\_slave\\_address\\_t::address](#)
- (2) [bool i2c\\_slave\\_address\\_t::addressDisable](#)

#### 13.5.2.2 struct i2c\_slave\_config\_t

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the [I2C\\_SlaveGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

##### Data Fields

- [i2c\\_slave\\_address\\_t address0](#)  
Slave's 7-bit address and disable.
- [i2c\\_slave\\_address\\_t address1](#)  
Alternate slave 7-bit address and disable.
- [i2c\\_slave\\_address\\_t address2](#)  
Alternate slave 7-bit address and disable.

- `i2c_slave_address_t address3`  
*Alternate slave 7-bit address and disable.*
- `i2c_slave_address_qual_mode_t qualMode`  
*Qualify mode for slave address 0.*
- `uint8_t qualAddress`  
*Slave address qualifier for address 0.*
- `i2c_slave_bus_speed_t busSpeed`  
*Slave bus speed mode.*
- `bool enableSlave`  
*Enable slave mode.*

**Field Documentation**

- (1) `i2c_slave_address_t i2c_slave_config_t::address0`
- (2) `i2c_slave_address_t i2c_slave_config_t::address1`
- (3) `i2c_slave_address_t i2c_slave_config_t::address2`
- (4) `i2c_slave_address_t i2c_slave_config_t::address3`
- (5) `i2c_slave_address_qual_mode_t i2c_slave_config_t::qualMode`
- (6) `uint8_t i2c_slave_config_t::qualAddress`
- (7) `i2c_slave_bus_speed_t i2c_slave_config_t::busSpeed`

If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The `busSpeed` value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the `busSpeed` mode is unknown at compile time, use the longest data setup time `kI2C_SlaveStandardMode` (250 ns)

- (8) `bool i2c_slave_config_t::enableSlave`

**13.5.2.3 struct i2c\_slave\_transfer\_t****Data Fields**

- `i2c_slave_handle_t * handle`  
*Pointer to handle that contains this transfer.*
- `i2c_slave_transfer_event_t event`  
*Reason the callback is being invoked.*
- `uint8_t receivedAddress`  
*Matching address send by master.*
- `uint32_t eventMask`  
*Mask of enabled events.*
- `uint8_t * rxData`  
*Transfer buffer for receive data.*

- const uint8\_t \* **txData**  
*Transfer buffer for transmit data.*
- size\_t **txSize**  
*Transfer size.*
- size\_t **rxSize**  
*Transfer size.*
- size\_t **transferredCount**  
*Number of bytes transferred during this transfer.*
- status\_t **completionStatus**  
*Success or error code describing how the transfer completed.*

### Field Documentation

- (1) i2c\_slave\_handle\_t\* i2c\_slave\_transfer\_t::handle
- (2) i2c\_slave\_transfer\_event\_t i2c\_slave\_transfer\_t::event
- (3) uint8\_t i2c\_slave\_transfer\_t::receivedAddress  
7-bits plus R/nW bit0
- (4) uint32\_t i2c\_slave\_transfer\_t::eventMask
- (5) size\_t i2c\_slave\_transfer\_t::transferredCount
- (6) status\_t i2c\_slave\_transfer\_t::completionStatus

Only applies for [kI2C\\_SlaveCompletionEvent](#).

#### 13.5.2.4 struct \_i2c\_slave\_handle

I2C slave handle typedef.

Note

The contents of this structure are private and subject to change.

### Data Fields

- volatile i2c\_slave\_transfer\_t transfer  
*I2C slave transfer.*
- volatile bool **isBusy**  
*Whether transfer is busy.*
- volatile i2c\_slave\_fsm\_t **slaveFsm**  
*slave transfer state machine.*
- i2c\_slave\_transfer\_callback\_t **callback**  
*Callback function called at transfer event.*
- void \* **userData**  
*Callback parameter passed to callback.*

## Field Documentation

- (1) `volatile i2c_slave_transfer_t i2c_slave_handle_t::transfer`
- (2) `volatile bool i2c_slave_handle_t::isBusy`
- (3) `volatile i2c_slave_fsm_t i2c_slave_handle_t::slaveFsm`
- (4) `i2c_slave_transfer_callback_t i2c_slave_handle_t::callback`
- (5) `void* i2c_slave_handle_t::userData`

### 13.5.3 Typedef Documentation

**13.5.3.1 `typedef void(* i2c_slave_transfer_callback_t)(I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *userData)`**

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I2C\_SlaveSetCallback() function after you have created a handle.

Parameters

<i>base</i>	Base address for the I2C instance on which the event occurred.
<i>transfer</i>	Pointer to transfer descriptor containing values passed to and/or from the callback.
<i>userData</i>	Arbitrary pointer-sized value passed from the application.

**13.5.3.2 `typedef void(* flexcomm_i2c_master_irq_handler_t)(I2C_Type *base, i2c_master_handle_t *handle)`**

**13.5.3.3 `typedef void(* flexcomm_i2c_slave_irq_handler_t)(I2C_Type *base, i2c_slave_handle_t *handle)`**

### 13.5.4 Enumeration Type Documentation

**13.5.4.1 `enum i2c_slave_address_register_t`**

Enumerator

- kI2C\_SlaveAddressRegister0* Slave Address 0 register.
- kI2C\_SlaveAddressRegister1* Slave Address 1 register.
- kI2C\_SlaveAddressRegister2* Slave Address 2 register.
- kI2C\_SlaveAddressRegister3* Slave Address 3 register.

### 13.5.4.2 enum i2c\_slave\_address\_qual\_mode\_t

Enumerator

**kI2C\_QualModeMask** The SLVQUAL0 field (qualAddress) is used as a logical mask for matching address0.

**kI2C\_QualModeExtend** The SLVQUAL0 (qualAddress) field is used to extend address 0 matching in a range of addresses.

### 13.5.4.3 enum i2c\_slave\_bus\_speed\_t

### 13.5.4.4 enum i2c\_slave\_transfer\_event\_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to [I2C\\_SlaveTransferNonBlocking\(\)](#) in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

**kI2C\_SlaveAddressMatchEvent** Received the slave address after a start or repeated start.

**kI2C\_SlaveTransmitEvent** Callback is requested to provide data to transmit (slave-transmitter role).

**kI2C\_SlaveReceiveEvent** Callback is requested to provide a buffer in which to place received data (slave-receiver role).

**kI2C\_SlaveCompletionEvent** All data in the active transfer have been consumed.

**kI2C\_SlaveDeselectedEvent** The slave function has become deselected (SLVSEL flag changing from 1 to 0).

**kI2C\_SlaveAllEvents** Bit mask of all available events.

## 13.5.5 Function Documentation

### 13.5.5.1 void I2C\_SlaveGetDefaultConfig ( i2c\_slave\_config\_t \* *slaveConfig* )

This function provides the following default configuration for the I2C slave peripheral:

```
* slaveConfig->enableSlave = true;
* slaveConfig->address0.disable = false;
* slaveConfig->address0.address = 0u;
* slaveConfig->address1.disable = true;
* slaveConfig->address2.disable = true;
* slaveConfig->address3.disable = true;
* slaveConfig->busSpeed = kI2C_SlaveStandardMode;
*
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with [I2C\\_SlaveInit\(\)](#). Be sure to override at least the *address0.address* member of the configuration structure with the desired slave address.

Parameters

<i>out</i>	<i>slaveConfig</i>	User provided configuration structure that is set to default values. Refer to <a href="#">i2c_slave_config_t</a> .
------------	--------------------	--

### **13.5.5.2 status\_t I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *srcClock\_Hz* )**

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>slaveConfig</i>	User provided peripheral configuration. Use <a href="#">I2C_SlaveGetDefaultConfig()</a> to get a set of defaults that you can override.
<i>srcClock_Hz</i>	Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock.

### **13.5.5.3 void I2C\_SlaveSetAddress ( I2C\_Type \* *base*, i2c\_slave\_address\_register\_t *addressRegister*, uint8\_t *address*, bool *addressDisable* )**

This function writes new value to Slave Address register.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>address-Register</i>	The module supports multiple address registers. The parameter determines which one shall be changed.
<i>address</i>	The slave address to be stored to the address register for matching.
<i>addressDisable</i>	Disable matching of the specified address register.

### **13.5.5.4 void I2C\_SlaveDeinit ( I2C\_Type \* *base* )**

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

<i>base</i>	The I2C peripheral base address.
-------------	----------------------------------

### 13.5.5.5 static void I2C\_SlaveEnable ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	The I2C peripheral base address.
<i>enable</i>	True to enable or false to disable.

### 13.5.5.6 static void I2C\_SlaveClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>statusMask</i>	A bitmask of status flags that are to be cleared. The mask is composed of _i2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to I2C_SlaveGetStatusFlags().

See Also

[\\_i2c\\_slave\\_flags](#).

### 13.5.5.7 status\_t I2C\_SlaveWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize* )

The function executes blocking address phase and blocking data phase.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>txBuff</i>	The pointer to the data to be transferred.
<i>txSize</i>	The length in bytes of the data to be transferred.

Returns

kStatus\_Success Data has been sent.

kStatus\_Fail Unexpected slave state (master data write while master read from slave is expected).

### 13.5.5.8 status\_t I2C\_SlaveReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize* )

The function executes blocking address phase and blocking data phase.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>rxBuff</i>	The pointer to the data to be transferred.
<i>rxSize</i>	The length in bytes of the data to be transferred.

Returns

kStatus\_Success Data has been received.

kStatus\_Fail Unexpected slave state (master data read while master write to slave is expected).

### 13.5.5.9 void I2C\_SlaveTransferCreateHandle ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, i2c\_slave\_transfer\_callback\_t *callback*, void \* *userData* )

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [I2C\\_SlaveTransferAbort\(\)](#) API shall be called.

Parameters

---

	<i>base</i>	The I2C peripheral base address.
out	<i>handle</i>	Pointer to the I2C slave driver handle.
	<i>callback</i>	User provided pointer to the asynchronous callback function.
	<i>userData</i>	User provided pointer to the application callback data.

### 13.5.5.10 status\_t I2C\_SlaveTransferNonBlocking ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, uint32\_t *eventMask* )

Call this API after calling [I2C\\_SlaveInit\(\)](#) and [I2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to [I2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

If no slave Tx transfer is busy, a master read from slave request invokes [kI2C\\_SlaveTransmitEvent](#) callback. If no slave Rx transfer is busy, a master write to slave request invokes [kI2C\\_SlaveReceiveEvent](#) callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [k-I2C\\_SlaveTransmitEvent](#) and [kI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

#### Parameters

<i>base</i>	The I2C peripheral base address.
<i>handle</i>	Pointer to <a href="#">i2c_slave_handle_t</a> structure which stores the transfer state.
<i>eventMask</i>	Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events.

#### Return values

<a href="#">kStatus_Success</a>	Slave transfers were successfully started.
<a href="#">kStatus_I2C_Busy</a>	Slave transfers have already been started on this handle.

### 13.5.5.11 status\_t I2C\_SlaveSetSendBuffer ( **I2C\_Type** \* *base*, volatile **i2c\_slave\_transfer\_t** \* *transfer*, const void \* *txData*, size\_t *txSize*, uint32\_t *eventMask* )

The function can be called in response to [kI2C\\_SlaveTransmitEvent](#) callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [k-I2C\\_SlaveTransmitEvent](#) and [kI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>transfer</i>	Pointer to <a href="#">i2c_slave_transfer_t</a> structure.
<i>txData</i>	Pointer to data to send to master.
<i>txSize</i>	Size of txData in bytes.
<i>eventMask</i>	Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events.

Return values

<a href="#">kStatus_Success</a>	Slave transfers were successfully started.
<a href="#">kStatus_I2C_Busy</a>	Slave transfers have already been started on this handle.

### 13.5.5.12 status\_t I2C\_SlaveSetReceiveBuffer ( **I2C\_Type** \* *base*, volatile **i2c\_slave\_transfer\_t** \* *transfer*, void \* *rxData*, size\_t *rxSize*, uint32\_t *eventMask* )

The function can be called in response to [kI2C\\_SlaveReceiveEvent](#) callback to start a new slave Rx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [k-I2C\\_SlaveTransmitEvent](#) and [kI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>transfer</i>	Pointer to <a href="#">i2c_slave_transfer_t</a> structure.
<i>rxData</i>	Pointer to data to store data from master.
<i>rxSize</i>	Size of rxData in bytes.
<i>eventMask</i>	Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events.

Return values

<a href="#">kStatus_Success</a>	Slave transfers were successfully started.
<a href="#">kStatus_I2C_Busy</a>	Slave transfers have already been started on this handle.

### 13.5.5.13 static uint32\_t I2C\_SlaveGetReceivedAddress ( I2C\_Type \* *base*, volatile i2c\_slave\_transfer\_t \* *transfer* ) [inline], [static]

This function should only be called from the address match event callback [kI2C\\_SlaveAddressMatch-Event](#).

Parameters

<i>base</i>	The I2C peripheral base address.
<i>transfer</i>	The I2C slave transfer.

Returns

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

### 13.5.5.14 void I2C\_SlaveTransferAbort ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle* )

Note

This API could be called at any time to stop slave for handling the bus events.

Parameters

<i>base</i>	The I2C peripheral base address.
<i>handle</i>	Pointer to i2c_slave_handle_t structure which stores the transfer state.

Return values

<i>kStatus_Success</i>	
<i>kStatus_I2C_Idle</i>	

### 13.5.5.15 status\_t I2C\_SlaveTransferGetCount ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

<i>base</i>	I2C base pointer.
<i>handle</i>	pointer to i2c_slave_handle_t structure.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Return values

<i>kStatus_InvalidArgument</i>	<i>count</i> is Invalid.
<i>kStatus_Success</i>	Successfully return the count.

### 13.5.5.16 void I2C\_SlaveTransferHandleIRQ ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle* )

Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

<i>base</i>	The I2C peripheral base address.
-------------	----------------------------------

<i>handle</i>	Pointer to i2c_slave_handle_t structure which stores the transfer state.
---------------	--

## 13.6 I2C DMA Driver

### 13.6.1 Overview

#### Data Structures

- struct `i2c_master_dma_handle_t`  
*I2C master dma transfer structure. More...*

#### Macros

- #define `I2C_MAX_DMA_TRANSFER_COUNT` 1024  
*Maximum lenght of single DMA transfer (determined by capability of the DMA engine)*

#### Typedefs

- typedef void(\* `i2c_master_dma_transfer_callback_t` )(I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, `status_t` status, void \*userData)  
*I2C master dma transfer callback typedef.*
- typedef void(\* `flexcomm_i2c_dma_master_irq_handler_t` )(I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle)  
*Typedef for master dma handler.*

#### Driver version

- #define `FSL_I2C_DMA_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 1)`)  
*I2C DMA driver version.*

#### I2C Block DMA Transfer Operation

- void `I2C_MasterTransferCreateHandleDMA` (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, `i2c_master_dma_transfer_callback_t` callback, void \*userData, `dma_handle_t` \*dmaHandle)  
*Init the I2C handle which is used in transactional functions.*
- `status_t I2C_MasterTransferDMA` (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, `i2c_master_transfer_t` \*xfer)  
*Performs a master dma non-blocking transfer on the I2C bus.*
- `status_t I2C_MasterTransferGetCountDMA` (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, `size_t` \*count)  
*Get master transfer status during a dma non-blocking transfer.*
- void `I2C_MasterTransferAbortDMA` (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle)  
*Abort a master dma non-blocking transfer in a early time.*

## 13.6.2 Data Structure Documentation

### 13.6.2.1 struct \_i2c\_master\_dma\_handle

I2C master dma handle typedef.

#### Data Fields

- `uint8_t state`  
*Transfer state machine current state.*
- `uint32_t transferCount`  
*Indicates progress of the transfer.*
- `uint32_t remainingBytesDMA`  
*Remaining byte count to be transferred using DMA.*
- `uint8_t * buf`  
*Buffer pointer for current state.*
- `bool checkAddrNack`  
*Whether to check the nack signal is detected during addressing.*
- `dma_handle_t * dmaHandle`  
*The DMA handler used.*
- `i2c_master_transfer_t transfer`  
*Copy of the current transfer info.*
- `i2c_master_dma_transfer_callback_t completionCallback`  
*Callback function called after dma transfer finished.*
- `void * userData`  
*Callback parameter passed to callback function.*

#### Field Documentation

- (1) `uint8_t i2c_master_dma_handle_t::state`
- (2) `uint32_t i2c_master_dma_handle_t::remainingBytesDMA`
- (3) `uint8_t* i2c_master_dma_handle_t::buf`
- (4) `bool i2c_master_dma_handle_t::checkAddrNack`
- (5) `dma_handle_t* i2c_master_dma_handle_t::dmaHandle`
- (6) `i2c_master_transfer_t i2c_master_dma_handle_t::transfer`
- (7) `i2c_master_dma_transfer_callback_t i2c_master_dma_handle_t::completionCallback`
- (8) `void* i2c_master_dma_handle_t::userData`

## 13.6.3 Macro Definition Documentation

### 13.6.3.1 #define FSL\_I2C\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

### 13.6.4 Typedef Documentation

- 13.6.4.1 `typedef void(* i2c_master_dma_transfer_callback_t)(I2C_Type *base, i2c_master_dma_handle_t *handle, status_t status, void *userData)`
- 13.6.4.2 `typedef void(* flexcomm_i2c_dma_master_irq_handler_t)(I2C_Type *base, i2c_master_dma_handle_t *handle)`

### 13.6.5 Function Documentation

- 13.6.5.1 `void I2C_MasterTransferCreateHandleDMA ( I2C_Type * base, i2c_master_dma_handle_t * handle, i2c_master_dma_transfer_callback_t callback, void * userData, dma_handle_t * dmaHandle )`

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	pointer to i2c_master_dma_handle_t structure
<i>callback</i>	pointer to user callback function
<i>userData</i>	user param passed to the callback function
<i>dmaHandle</i>	DMA handle pointer

- 13.6.5.2 `status_t I2C_MasterTransferDMA ( I2C_Type * base, i2c_master_dma_handle_t * handle, i2c_master_transfer_t * xfer )`

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	pointer to i2c_master_dma_handle_t structure
<i>xfer</i>	pointer to transfer structure of i2c_master_transfer_t

Return values

<i>kStatus_Success</i>	Sucessully complete the data transmission.
<i>kStatus_I2C_Busy</i>	Previous transmission still not finished.

<i>kStatus_I2C_Timeout</i>	Transfer error, wait signal timeout.
<i>kStatus_I2C_Arbitration-Lost</i>	Transfer error, arbitration lost.
<i>kStatus_I2C_Nak</i>	Transfer error, receive Nak during transfer.

### 13.6.5.3 **status\_t I2C\_MasterTransferGetCountDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	pointer to i2c_master_dma_handle_t structure
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

### 13.6.5.4 **void I2C\_MasterTransferAbortDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle* )**

Parameters

<i>base</i>	I2C peripheral base address
<i>handle</i>	pointer to i2c_master_dma_handle_t structure

## 13.7 I2C FreeRTOS Driver

### 13.7.1 Overview

#### Data Structures

- struct `i2c_rtos_handle_t`  
*I2C FreeRTOS handle.* [More...](#)

#### Driver version

- #define `FSL_I2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 8)`)  
*I2C FreeRTOS driver version 2.0.8.*

#### I2C RTOS Operation

- `status_t I2C_RTOS_Init` (`i2c_rtos_handle_t *handle, I2C_Type *base, const i2c_master_config_t *masterConfig, uint32_t srcClock_Hz)  
Initializes I2C.`
- `status_t I2C_RTOS_Deinit` (`i2c_rtos_handle_t *handle`)  
*Deinitializes the I2C.*
- `status_t I2C_RTOS_Transfer` (`i2c_rtos_handle_t *handle, i2c_master_transfer_t *transfer`)  
*Performs I2C transfer.*

### 13.7.2 Data Structure Documentation

#### 13.7.2.1 struct `i2c_rtos_handle_t`

##### Data Fields

- `I2C_Type * base`  
*I2C base address.*
- `i2c_master_handle_t drv_handle`  
*A handle of the underlying driver, treated as opaque by the RTOS layer.*
- `status_t async_status`  
*Transactional state of the underlying driver.*
- `SemaphoreHandle_t mutex`  
*A mutex to lock the handle during a transfer.*
- `SemaphoreHandle_t semaphore`  
*A semaphore to notify and unblock task when the transfer ends.*

### 13.7.3 Macro Definition Documentation

### 13.7.3.1 #define FSL\_I2C\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 8))

## 13.7.4 Function Documentation

### 13.7.4.1 status\_t I2C\_RTOS\_Init ( i2c\_rtos\_handle\_t \* handle, I2C\_Type \* base, const i2c\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz )

This function initializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the I2C instance to initialize.
<i>masterConfig</i>	Configuration structure to set-up I2C in master mode.
<i>srcClock_Hz</i>	Frequency of input clock of the I2C module.

Returns

status of the operation.

### 13.7.4.2 status\_t I2C\_RTOS\_Deinit ( i2c\_rtos\_handle\_t \* handle )

This function deinitializes the I2C module and the related RTOS context.

Parameters

<i>handle</i>	The RTOS I2C handle.
---------------	----------------------

### 13.7.4.3 status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* handle, i2c\_master\_transfer\_t \* transfer )

This function performs an I2C transfer according to data given in the transfer structure.

Parameters

<i>handle</i>	The RTOS I2C handle.
<i>transfer</i>	Structure specifying the transfer parameters.

Returns

status of the operation.

## 13.8 I2C CMSIS Driver

This section describes the programming interface of the I2C Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The I2C CMSIS driver includes transactional APIs.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code accessing the hardware registers.

### 13.8.1 I2C CMSIS Driver

#### 13.8.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}
/*Init I2C MASTER*/
EXAMPLE_I2C_MASTER.Initialize(I2C_MasterSignalEvent_t);

EXAMPLE_I2C_MASTER.PowerControl(ARM_POWER_FULL);

/*config transmit speed/
EXAMPLE_I2C_MASTER.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
EXAMPLE_I2C_MASTER.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

#### 13.8.1.2 Master Operation in DMA transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    /* Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Init DMA*/
DMA_Init(EXAMPLE_DMA);
```

```

/*Init I2C MASTER*/
EXAMPLE_I2C_MASTER.Initialize(I2C_MasterSignalEvent_t);

EXAMPLE_I2C_MASTER.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
EXAMPLE_I2C_MASTER.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transfer*/
EXAMPLE_I2C_MASTER.MasterReceive(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;

```

### 13.8.1.3 Slave Operation in interrupt transactional method

```

void I2C_SlaveSignalEvent_t(uint32_t event)
{
    /* Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_SlaveCompletionFlag = true;
    }
}

/*Init I2C SLAVE*/
EXAMPLE_I2C_SLAVE.Initialize(I2C_SlaveSignalEvent_t);

EXAMPLE_I2C_SLAVE.PowerControl(ARM_POWER_FULL);

/*config slave addr*/
EXAMPLE_I2C_SLAVE.Control(ARM_I2C_OWN_ADDRESS, I2C_MASTER_SLAVE_ADDR);

/*start transfer*/
EXAMPLE_I2C_SLAVE.SlaveReceive(g_slave_buff, I2C_DATA_LENGTH);

/* Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
g_SlaveCompletionFlag = false;

```

# Chapter 14

## I2S: I2S Driver

### 14.1 Overview

The MCUXpresso SDK provides the peripheral driver for the I2S function of FLEXCOMM module of MCUXpresso SDK devices.

The I2S module is used to transmit or receive digital audio data. Only transmit or receive is enabled at one time in one module.

Driver currently supports one (primary) channel pair per one I2S enabled FLEXCOMM module only.

### 14.2 I2S Driver Initialization and Configuration

[I2S\\_TxInit\(\)](#) and [I2S\\_RxInit\(\)](#) functions ungate the clock for the FLEXCOMM module, assign I2S function to FLEXCOMM module and configure audio data format and other I2S operational settings. [I2S\\_TxInit\(\)](#) is used when I2S should transmit data, [I2S\\_RxInit\(\)](#) when it should receive data.

[I2S\\_TxGetDefaultConfig\(\)](#) and [I2S\\_RxGetDefaultConfig\(\)](#) functions can be used to set the module configuration structure with default values for transmit and receive function, respectively.

[I2S\\_Deinit\(\)](#) function resets the FLEXCOMM module.

[I2S\\_TxTransferCreateHandle\(\)](#) function creates transactional handle for transmit in interrupt mode.

[I2S\\_RxTransferCreateHandle\(\)](#) function creates transactional handle for receive in interrupt mode.

[I2S\\_TxTransferCreateHandleDMA\(\)](#) function creates transactional handle for transmit in DMA mode.

[I2S\\_RxTransferCreateHandleDMA\(\)](#) function creates transactional handle for receive in DMA mode.

### 14.3 I2S Transmit Data

[I2S\\_TxTransferNonBlocking\(\)](#) function is used to add data buffer to transmit in interrupt mode. It also begins transmission if not transmitting yet.

[I2S\\_RxTransferNonBlocking\(\)](#) function is used to add data buffer to receive data into in interrupt mode. It also begins reception if not receiving yet.

[I2S\\_TxTransferSendDMA\(\)](#) function is used to add data buffer to transmit in DMA mode. It also begins transmission if not transmitting yet.

[I2S\\_RxTransferReceiveDMA\(\)](#) function is used to add data buffer to receive data into in DMA mode. It also begins reception if not receiving yet.

The transfer of data will be stopped automatically when all data buffers queued using the above functions will be processed and no new data buffer is enqueued meanwhile. If the above functions are not called frequently enough, I2S stop followed by restart may keep occurring resulting in drops audio stream.

## 14.4 I2S Interrupt related functions

[I2S\\_EnableInterrupts\(\)](#) function is used to enable interrupts in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

[I2S\\_DisableInterrupts\(\)](#) function is used to disable interrupts in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

[I2S\\_GetEnabledInterrupts\(\)](#) function returns interrupts enabled in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

[I2S\\_TxHandleIRQ\(\)](#) and [I2S\\_RxHandleIRQ\(\)](#) functions are called from ISR which is invoked when actual FIFO level decreases to configured watermark value.

[I2S\\_DMACallback\(\)](#) function is called from ISR which is invoked when DMA transfer (actual descriptor) finishes.

## 14.5 I2S Other functions

[I2S\\_Enable\(\)](#) function enables I2S function in FLEXCOMM module. Regular use cases do not require this function to be called from application code.

[I2S\\_Disable\(\)](#) function disables I2S function in FLEXCOMM module. Regular use cases do not require this function to be called from application code.

[I2S\\_TransferGetErrorCount\(\)](#) function returns the number of FIFO underruns or overruns in interrupt mode.

[I2S\\_TransferGetCount\(\)](#) function returns the number of bytes transferred in interrupt mode.

[I2S\\_TxTransferAbort\(\)](#) function aborts transmit operation in interrupt mode.

[I2S\\_RxTransferAbort\(\)](#) function aborts receive operation in interrupt mode.

[I2S\\_TransferAbortDMA\(\)](#) function aborts transmit or receive operation in DMA mode.

## I2S Functional limitations

I2S can not accurately determine when the data is sent to the end. Since program commands cannot quickly disable I2S, there will always be more clock exposure. If someone wants to get accurate data, can use [I2S\\_TransferAbortDMA\(\)](#) api in TxCallback() to terminate sending data prematurely and ensure the accuracy of the data with delay function.

## 14.6 I2S Data formats

### 14.6.1 DMA mode

Length of buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes. Data are put into or taken from FIFO unaltered in DMA mode so buffer has to be prepared according to following information. If oneChannel is enabled, then the buffer should contain valid data only, that is to say, audio data should be put continuously in the buffer Take 8 bit data as example:

LSB

L07

L06

L05

L04

L03

L02

L01

L00

If `i2s_config_t.dataLength` (channel bit width) is between 4 and 16, every word in buffer should contain data for left and right channels.

<b>MSB</b>																						<b>LSB</b>
R15	R1	R1	R1	R1	R1	R0	L00															

Rnn - right channel bit nn

Lnn - left channel bit nn

Note that for example if `i2s_config_t.dataLength` = 7, bits on positions R07-R15 and L07-L15 are ignored (buffer "wastes space").

If `i2s_config_t.dataLength` (channel bit width) is between 17 and 24 and `i2s_config_t.pack48` = false:

Even words (counting from zero):

<b>MSB</b>																						<b>LSB</b>
																						L00

Odd words (counting from zero):

<b>MSB</b>																						<b>LSB</b>
																						R00

If `i2s_config_t.dataLength` (channel bit width) is between 17 and 24 and `i2s_config_t.pack48` = true:

Even words (counting from zero):

<b>MSB</b>																						<b>LSB</b>
R07	R0	L00																				

Odd words (counting from zero):

<b>MSB</b>																						<b>LSB</b>
																						R08

If `i2s_config_t.dataLength` (channel bit width) is between 25 and 32:

Even words (counting from zero):

<b>MSB</b>																						<b>LSB</b>																														
L31	L3	0	2	2	8	2	L2	6	2	5	2	4	2	3	2	2	L2	0	1	2	1	8	L1	6	1	5	1	4	1	3	1	2	1	L1	0	0	2	0	8	0	L0	6	0	5	0	4	0	3	0	2	0	L00

Odd words (counting from zero):

<b>MSB</b>																											<b>LSB</b>	
R31	R3	R2	R1	R0	R0	R0	R0	R0	R0	R0																		

#### 14.6.2 Interrupt mode

If `i2s_config_t.dataLength` (channel bit width) is 4:

Buffer does not need to be aligned (buffer is read / written by single bytes, each byte contain left and right channel):

<b>MSB</b>																											<b>LSB</b>
R03	R02	R01	R00	L03	L02	L01	L00																				

If `i2s_config_t.dataLength` (channel bit width) is between 5 and 8:

Length of buffer for transmit or receive has to be multiply of 2 bytes. Buffer address has to be aligned to 2-bytes.

<b>MSB</b>																											<b>LSB</b>
R07	R06	R05	R04	R03	R02	R01	R00	L07	L06	L05	L04	L03	L02	L01	L00												

If `i2s_config_t.dataLength` (channel bit width) is between 9 and 16:

Length of buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes.

<b>MSB</b>																											<b>LSB</b>
R15	R1	R1	R1	R1	R1	R0																					

If `i2s_config_t.dataLength` (channel bit width) is between 17 and 24 and `i2s_config_t.pack48 = false`(mono channel audio data is supported):

Length of buffer for transmit or receive has to be multiply of 6 bytes.

<b>MSB</b>																											<b>LSB</b>
R23	R2	R2	R2	R1	R0																						

If `i2s_config_t.dataLength` (channel bit width) is between 17 and 24 and `i2s_config_t.pack48 = true`:

Length of buffer for transmit or receive has to be multiply of 6 bytes. Buffer address has to be aligned to 4-bytes.

**MSB** R23 R2R2R2R1S1S1R1K1E1E1R1B1B1B1R1R0S0S0R0E0F0O0B0B0B0D02B2D2T1201918171615141B1D1T110019080T

If `i2s_config_t.dataLength` (channel bit width) is between 25 and 32 and `i2s_config_t.oneChannel = false`:  
Buffer for transmit or receive has to be multiple of 8 bytes. Buffer address has to be aligned to 4 bytes.

Even words (counting from zero):

<b>MSB</b>																										<b>LSB</b>																																				
L31	L	3	0	2	9	2	8	2	L	2	6	2	5	2	4	2	3	2	2	2	L	2	0	1	9	1	8	1	L	1	6	1	5	1	4	1	3	1	2	1	L	1	0	2	0	8	0	L	0	6	0	5	0	4	0	3	0	2	0	L	0	0

Odd words (counting from zero):

If `i2s_config_t.dataLength` (channel bit width) is between 25 and 32 and `i2s_config_t.oneChannel = true`:  
Buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes.

<b>MSB</b>																														<b>LSB</b>																														
L31	L	3	0	2	2	8	2	L	2	6	2	5	2	4	2	3	2	2	1	2	0	1	9	1	8	1	L	1	6	1	5	1	4	1	3	1	2	1	L	1	0	2	0	8	0	L	0	6	0	5	0	4	0	3	0	2	0	L	0	0

## 14.7 I2S Driver Examples

### 14.7.1 Interrupt mode examples

## Transmit example

```

void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_handle_t handle;

    I2S_TxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_TxInit(I2S0, &config);

    I2S_TxTransferCreateHandle(I2S0, &handle, TxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_TxTransferNonBlocking(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
     * finishes */
    I2S_TxTransferNonBlocking(I2S0, &handle, someTransfer);
}

```

```

}

void TxCallback(I2S_Type *base, i2s_handle_t *handle, status_t completionStatus, void *userData)
{
    i2s_transfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_TxTransferNonBlocking(base, handle, transfer);
    }
}

```

## Receive example

```

void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_handle_t handle;

    I2S_RxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_RxInit(I2S0, &config);

    I2S_RxTransferCreateHandle(I2S0, &handle, RxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_RxTransferNonBlocking(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
     * finishes */
    I2S_RxTransferNonBlocking(I2S0, &handle, someTransfer);
}

void RxCallback(I2S_Type *base, i2s_handle_t *handle, status_t completionStatus, void *userData)
{
    i2s_transfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_RxTransferNonBlocking(base, handle, transfer);
    }
}

```

## 14.7.2 DMA mode examples

### Transmit example

```

void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_dma_handle_t handle;

```

```

I2S_TxGetDefaultConfig(&config);
config.masterSlave = kI2S_MasterSlaveNormalMaster;
config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
I2S_TxInit(I2S0, &config);

I2S_TxTransferCreateHandleDMA(I2S0, &handle, TxCallback, NULL);

transfer.data = buffer;
transfer.dataSize = sizeof(buffer);
I2S_TxTransferNonBlockingDMA(I2S0, &handle, transfer);

/* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
finishes */
I2S_TxTransferNonBlockingDMA(I2S0, &handle, someTransfer);
}

void TxCallback(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData
)
{
    i2s_transfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_TxTransferNonBlockingDMA(base, handle, transfer);
    }
}

```

## Receive example

```

void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_dma_handle_t handle;

    I2S_RxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_RxInit(I2S0, &config);

    I2S_RxTransferCreateHandleDMA(I2S0, &handle, RxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_RxTransferNonBlockingDMA(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
finishes */
    I2S_RxTransferNonBlockingDMA(I2S0, &handle, someTransfer);
}

void RxCallback(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData
)
{
    i2s_transfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_RxTransferNonBlockingDMA(base, handle, transfer);
    }
}

```

```
    }  
}
```

## Modules

- I2S Driver

## 14.8 I2S Driver

### 14.8.1 Overview

#### Files

- file [fsl\\_i2s.h](#)

#### Data Structures

- struct [i2s\\_config\\_t](#)  
*I2S configuration structure. [More...](#)*
- struct [i2s\\_transfer\\_t](#)  
*Buffer to transfer from or receive audio data into. [More...](#)*
- struct [i2s\\_handle\\_t](#)  
*Members not to be accessed / modified outside of the driver. [More...](#)*

#### Macros

- #define [I2S\\_NUM\\_BUFFERS](#) (4U)  
*Number of buffers.*

#### Typedefs

- typedef void(\* [i2s\\_transfer\\_callback\\_t](#))([I2S\\_Type](#) \*base, [i2s\\_handle\\_t](#) \*handle, [status\\_t](#) completionStatus, void \*userData)  
*Callback function invoked from transactional API on completion of a single buffer transfer.*

#### Enumerations

- enum {
 [kStatus\\_I2S\\_BufferComplete](#),
 [kStatus\\_I2S\\_Done](#) = MAKE\_STATUS([kStatusGroup\\_I2S](#), 1),
 [kStatus\\_I2S\\_Busy](#) }
   
*\_i2s\_status I2S status codes.*
- enum [i2s\\_flags\\_t](#) {
 [kI2S\\_TxErrorFlag](#) = [I2S\\_FIFOINTENSET\\_TXERR\\_MASK](#),
 [kI2S\\_TxLevelFlag](#) = [I2S\\_FIFOINTENSET\\_TXLVL\\_MASK](#),
 [kI2S\\_RxErrorFlag](#) = [I2S\\_FIFOINTENSET\\_RXERR\\_MASK](#),
 [kI2S\\_RxLevelFlag](#) = [I2S\\_FIFOINTENSET\\_RXLVL\\_MASK](#) }
   
*I2S flags.*

- enum `i2s_master_slave_t` {
   
    `kI2S_MasterSlaveNormalSlave` = 0x0,
   
    `kI2S_MasterSlaveWsSyncMaster` = 0x1,
   
    `kI2S_MasterSlaveExtSckMaster` = 0x2,
   
    `kI2S_MasterSlaveNormalMaster` = 0x3 }
   
        *Master / slave mode.*
- enum `i2s_mode_t` {
   
    `kI2S_ModeI2sClassic` = 0x0,
   
    `kI2S_ModeDspWs50` = 0x1,
   
    `kI2S_ModeDspWsShort` = 0x2,
   
    `kI2S_ModeDspWsLong` = 0x3 }
   
        *I2S mode.*
- enum {
   
    `kI2S_SecondaryChannel1` = 0U,
   
    `kI2S_SecondaryChannel2` = 1U,
   
    `kI2S_SecondaryChannel3` = 2U }
   
        *\_i2s\_secondary\_channel I2S secondary channel.*

## Driver version

- #define `FSL_I2S_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 2)`)
   
        *I2S driver version 2.3.2.*

## Initialization and deinitialization

- void `I2S_TxInit` (I2S\_Type \*base, const `i2s_config_t` \*config)
   
        *Initializes the FLEXCOMM peripheral for I2S transmit functionality.*
- void `I2S_RxInit` (I2S\_Type \*base, const `i2s_config_t` \*config)
   
        *Initializes the FLEXCOMM peripheral for I2S receive functionality.*
- void `I2S_TxGetDefaultConfig` (`i2s_config_t` \*config)
   
        *Sets the I2S Tx configuration structure to default values.*
- void `I2S_RxGetDefaultConfig` (`i2s_config_t` \*config)
   
        *Sets the I2S Rx configuration structure to default values.*
- void `I2S_Deinit` (I2S\_Type \*base)
   
        *De-initializes the I2S peripheral.*
- void `I2S_SetBitClockRate` (I2S\_Type \*base, uint32\_t sourceClockHz, uint32\_t sampleRate, uint32\_t bitWidth, uint32\_t channelNumbers)
   
        *Transmitter/Receiver bit clock rate configurations.*

## Non-blocking API

- void `I2S_TxTransferCreateHandle` (I2S\_Type \*base, i2s\_handle\_t \*handle, `i2s_transfer_callback_t` callback, void \*userData)
   
        *Initializes handle for transfer of audio data.*

- `status_t I2S_TxTransferNonBlocking` (`I2S_Type *base, i2s_handle_t *handle, i2s_transfer_t transfer`)
 

*Begins or queue sending of the given data.*
- `void I2S_TxTransferAbort` (`I2S_Type *base, i2s_handle_t *handle`)
 

*Aborts sending of data.*
- `void I2S_RxTransferCreateHandle` (`I2S_Type *base, i2s_handle_t *handle, i2s_transfer_callback_t callback, void *userData`)
 

*Initializes handle for reception of audio data.*
- `status_t I2S_RxTransferNonBlocking` (`I2S_Type *base, i2s_handle_t *handle, i2s_transfer_t transfer`)
 

*Begins or queue reception of data into given buffer.*
- `void I2S_RxTransferAbort` (`I2S_Type *base, i2s_handle_t *handle`)
 

*Aborts receiving of data.*
- `status_t I2S_TransferGetCount` (`I2S_Type *base, i2s_handle_t *handle, size_t *count`)
 

*Returns number of bytes transferred so far.*
- `status_t I2S_TransferGetErrorCount` (`I2S_Type *base, i2s_handle_t *handle, size_t *count`)
 

*Returns number of buffer underruns or overruns.*

## Enable / disable

- `static void I2S_Enable` (`I2S_Type *base`)
 

*Enables I2S operation.*
- `static void I2S_Disable` (`I2S_Type *base`)
 

*Disables I2S operation.*

## Interrupts

- `static void I2S_EnableInterrupts` (`I2S_Type *base, uint32_t interruptMask`)
 

*Enables I2S FIFO interrupts.*
- `static void I2S_DisableInterrupts` (`I2S_Type *base, uint32_t interruptMask`)
 

*Disables I2S FIFO interrupts.*
- `static uint32_t I2S_GetEnabledInterrupts` (`I2S_Type *base`)
 

*Returns the set of currently enabled I2S FIFO interrupts.*
- `status_t I2S_EmptyTxFifo` (`I2S_Type *base`)
 

*Flush the valid data in TX fifo.*
- `void I2S_TxHandleIRQ` (`I2S_Type *base, i2s_handle_t *handle`)
 

*Invoked from interrupt handler when transmit FIFO level decreases.*
- `void I2S_RxHandleIRQ` (`I2S_Type *base, i2s_handle_t *handle`)
 

*Invoked from interrupt handler when receive FIFO level decreases.*

### 14.8.2 Data Structure Documentation

### 14.8.2.1 struct i2s\_config\_t

#### Data Fields

- **i2s\_master\_slave\_t masterSlave**  
*Master / slave configuration.*
- **i2s\_mode\_t mode**  
*I2S mode.*
- **bool rightLow**  
*Right channel data in low portion of FIFO.*
- **bool leftJust**  
*Left justify data in FIFO.*
- **bool sckPol**  
*SCK polarity.*
- **bool wsPol**  
*WS polarity.*
- **uint16\_t divider**  
*Flexcomm function clock divider (1 - 4096)*
- **bool oneChannel**  
*true mono, false stereo*
- **uint8\_t dataLength**  
*Data length (4 - 32)*
- **uint16\_t frameLength**  
*Frame width (4 - 512)*
- **uint16\_t position**  
*Data position in the frame.*
- **uint8\_t watermark**  
*FIFO trigger level.*
- **bool txEmptyZero**  
*Transmit zero when buffer becomes empty or last item.*
- **bool pack48**  
*Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)*

### 14.8.2.2 struct i2s\_transfer\_t

#### Data Fields

- **uint8\_t \* data**  
*Pointer to data buffer.*
- **size\_t dataSize**  
*Buffer size in bytes.*

#### Field Documentation

- (1) **uint8\_t\* i2s\_transfer\_t::data**
- (2) **size\_t i2s\_transfer\_t::dataSize**

### 14.8.2.3 struct \_i2s\_handle

Transactional state of the initialized transfer or receive I2S operation.

#### Data Fields

- volatile uint32\_t `state`  
*State of transfer.*
- i2s\_transfer\_callback\_t `completionCallback`  
*Callback function pointer.*
- void \* `userData`  
*Application data passed to callback.*
- bool `oneChannel`  
*true mono, false stereo*
- uint8\_t `dataLength`  
*Data length (4 - 32)*
- bool `pack48`  
*Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)*
- uint8\_t `watermark`  
*FIFO trigger level.*
- bool `useFifo48H`  
*When dataLength 17-24: true use FIFOWR48H, false use FIFOWR.*
- volatile i2s\_transfer\_t `i2sQueue` [I2S\_NUM\_BUFFERS]  
*Transfer queue storing transfer buffers.*
- volatile uint8\_t `queueUser`  
*Queue index where user's next transfer will be stored.*
- volatile uint8\_t `queueDriver`  
*Queue index of buffer actually used by the driver.*
- volatile uint32\_t `errorCount`  
*Number of buffer underruns/overruns.*
- volatile uint32\_t `transferCount`  
*Number of bytes transferred.*

### 14.8.3 Macro Definition Documentation

#### 14.8.3.1 #define FSL\_I2S\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 2))

#### 14.8.3.2 #define I2S\_NUM\_BUFFERS (4U)

### 14.8.4 Typedef Documentation

#### 14.8.4.1 typedef void(\* i2s\_transfer\_callback\_t)(I2S\_Type \*base, i2s\_handle\_t \*handle, status\_t completionStatus, void \*userData)

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to I2S transaction.
<i>completion-Status</i>	status of the transaction.
<i>userData</i>	optional pointer to user arguments data.

## 14.8.5 Enumeration Type Documentation

### 14.8.5.1 anonymous enum

Enumerator

***kStatus\_I2S\_BufferComplete*** Transfer from/into a single buffer has completed.

***kStatus\_I2S\_Done*** All buffers transfers have completed.

***kStatus\_I2S\_Busy*** Already performing a transfer and cannot queue another buffer.

### 14.8.5.2 enum i2s\_flags\_t

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

***kI2S\_TxErrorFlag*** TX error interrupt.

***kI2S\_TxLevelFlag*** TX level interrupt.

***kI2S\_RxErrorFlag*** RX error interrupt.

***kI2S\_RxLevelFlag*** RX level interrupt.

### 14.8.5.3 enum i2s\_master\_slave\_t

Enumerator

***kI2S\_MasterSlaveNormalSlave*** Normal slave.

***kI2S\_MasterSlaveWsSyncMaster*** WS synchronized master.

***kI2S\_MasterSlaveExtSckMaster*** Master using existing SCK.

***kI2S\_MasterSlaveNormalMaster*** Normal master.

#### 14.8.5.4 enum i2s\_mode\_t

Enumerator

- kI2S\_ModeI2sClassic* I2S classic mode.
- kI2S\_ModeDspWs50* DSP mode, WS having 50% duty cycle.
- kI2S\_ModeDspWsShort* DSP mode, WS having one clock long pulse.
- kI2S\_ModeDspWsLong* DSP mode, WS having one data slot long pulse.

#### 14.8.5.5 anonymous enum

Enumerator

- kI2S\_SecondaryChannel1* secondary channel 1
- kI2S\_SecondaryChannel2* secondary channel 2
- kI2S\_SecondaryChannel3* secondary channel 3

### 14.8.6 Function Documentation

#### 14.8.6.1 void I2S\_TxInit ( I2S\_Type \* *base*, const i2s\_config\_t \* *config* )

Ungates the FLEXCOMM clock and configures the module for I2S transmission using a configuration structure. The configuration structure can be custom filled or set with default values by [I2S\\_TxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the I2S driver.

Parameters

<i>base</i>	I2S base pointer.
<i>config</i>	pointer to I2S configuration structure.

#### 14.8.6.2 void I2S\_RxInit ( I2S\_Type \* *base*, const i2s\_config\_t \* *config* )

Ungates the FLEXCOMM clock and configures the module for I2S receive using a configuration structure. The configuration structure can be custom filled or set with default values by [I2S\\_RxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the I2S driver.

## Parameters

<i>base</i>	I2S base pointer.
<i>config</i>	pointer to I2S configuration structure.

**14.8.6.3 void I2S\_TxGetDefaultConfig ( i2s\_config\_t \* *config* )**

This API initializes the configuration structure for use in [I2S\\_TxInit\(\)](#). The initialized structure can remain unchanged in [I2S\\_TxInit\(\)](#), or it can be modified before calling [I2S\\_TxInit\(\)](#). Example:

```
i2s_config_t config;
I2S_TxGetDefaultConfig(&config);
```

## Default values:

```
* config->masterSlave = kI2S_MasterSlaveNormalMaster;
* config->mode = kI2S_ModeI2sClassic;
* config->rightLow = false;
* config->leftJust = false;
* config->pdmData = false;
* config->sckPol = false;
* config->wsPol = false;
* config->divider = 1;
* config->oneChannel = false;
* config->dataLength = 16;
* config->frameLength = 32;
* config->position = 0;
* config->watermark = 4;
* config->txEmptyZero = true;
* config->pack48 = false;
*
```

## Parameters

<i>config</i>	pointer to I2S configuration structure.
---------------	---

**14.8.6.4 void I2S\_RxGetDefaultConfig ( i2s\_config\_t \* *config* )**

This API initializes the configuration structure for use in [I2S\\_RxInit\(\)](#). The initialized structure can remain unchanged in [I2S\\_RxInit\(\)](#), or it can be modified before calling [I2S\\_RxInit\(\)](#). Example:

```
i2s_config_t config;
I2S_RxGetDefaultConfig(&config);
```

## Default values:

```

* config->masterSlave = kI2S_MasterSlaveNormalSlave;
* config->mode = kI2S_ModeI2sClassic;
* config->rightLow = false;
* config->leftJust = false;
* config->pdmData = false;
* config->sckPol = false;
* config->wsPol = false;
* config->divider = 1;
* config->oneChannel = false;
* config->dataLength = 16;
* config->frameLength = 32;
* config->position = 0;
* config->watermark = 4;
* config->txEmptyZero = false;
* config->pack48 = false;
*

```

## Parameters

<i>config</i>	pointer to I2S configuration structure.
---------------	---

**14.8.6.5 void I2S\_Deinit ( I2S\_Type \* *base* )**

This API gates the FLEXCOMM clock. The I2S module can't operate unless I2S\_TxInit or I2S\_RxInit is called to enable the clock.

## Parameters

<i>base</i>	I2S base pointer.
-------------	-------------------

**14.8.6.6 void I2S\_SetBitClockRate ( I2S\_Type \* *base*, uint32\_t *sourceClockHz*, uint32\_t *sampleRate*, uint32\_t *bitWidth*, uint32\_t *channelNumbers* )**

## Parameters

<i>base</i>	SAI base pointer.
<i>sourceClockHz</i>	bit clock source frequency.
<i>sampleRate</i>	audio data sample rate.
<i>bitWidth</i>	audio data bitWidth.
<i>channel-Numbers</i>	audio channel numbers.

**14.8.6.7 void I2S\_TxTransferCreateHandle ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle*, i2s\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to handle structure.
<i>callback</i>	function to be called back when transfer is done or fails.
<i>userData</i>	pointer to data passed to callback.

#### 14.8.6.8 **status\_t I2S\_TxTransferNonBlocking ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle*, i2s\_transfer\_t *transfer* )**

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to handle structure.
<i>transfer</i>	data buffer.

Return values

<i>kStatus_Success</i>	
<i>kStatus_I2S_Busy</i>	if all queue slots are occupied with unsent buffers.

#### 14.8.6.9 **void I2S\_TxTransferAbort ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle* )**

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to handle structure.

#### 14.8.6.10 **void I2S\_RxTransferCreateHandle ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle*, i2s\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to handle structure.
<i>callback</i>	function to be called back when transfer is done or fails.
<i>userData</i>	pointer to data passed to callback.

#### 14.8.6.11 status\_t I2S\_RxTransferNonBlocking ( *I2S\_Type* \* *base*, *i2s\_handle\_t* \* *handle*, *i2s\_transfer\_t* *transfer* )

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to handle structure.
<i>transfer</i>	data buffer.

Return values

<i>kStatus_Success</i>	
<i>kStatus_I2S_Busy</i>	if all queue slots are occupied with buffers which are not full.

#### 14.8.6.12 void I2S\_RxTransferAbort ( *I2S\_Type* \* *base*, *i2s\_handle\_t* \* *handle* )

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to handle structure.

#### 14.8.6.13 status\_t I2S\_TransferGetCount ( *I2S\_Type* \* *base*, *i2s\_handle\_t* \* *handle*, *size\_t* \* *count* )

Parameters

	<i>base</i>	I2S base pointer.
	<i>handle</i>	pointer to handle structure.
<i>out</i>	<i>count</i>	number of bytes transferred so far by the non-blocking transaction.

Return values

<i>kStatus_Success</i>	
<i>kStatus_NoTransferInProgress</i>	there is no non-blocking transaction currently in progress.

#### 14.8.6.14 **status\_t I2S\_TransferGetErrorCount ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

	<i>base</i>	I2S base pointer.
	<i>handle</i>	pointer to handle structure.
out	<i>count</i>	number of transmit errors encountered so far by the non-blocking transaction.

Return values

<i>kStatus_Success</i>	
<i>kStatus_NoTransferInProgress</i>	there is no non-blocking transaction currently in progress.

#### 14.8.6.15 **static void I2S\_Enable ( I2S\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	I2S base pointer.
-------------	-------------------

#### 14.8.6.16 **static void I2S\_Disable ( I2S\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	I2S base pointer.
-------------	-------------------

#### 14.8.6.17 **static void I2S\_EnableInterrupts ( I2S\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]**

Parameters

<i>base</i>	I2S base pointer.
<i>interruptMask</i>	bit mask of interrupts to enable. See <a href="#">i2s_flags_t</a> for the set of constants that should be OR'd together to form the bit mask.

#### 14.8.6.18 static void I2S\_DisableInterrupts ( I2S\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Parameters

<i>base</i>	I2S base pointer.
<i>interruptMask</i>	bit mask of interrupts to enable. See <a href="#">i2s_flags_t</a> for the set of constants that should be OR'd together to form the bit mask.

#### 14.8.6.19 static uint32\_t I2S\_GetEnabledInterrupts ( I2S\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	I2S base pointer.
-------------	-------------------

Returns

A bitmask composed of [i2s\\_flags\\_t](#) enumerators OR'd together to indicate the set of enabled interrupts.

#### 14.8.6.20 status\_t I2S\_EmptyTxFifo ( I2S\_Type \* *base* )

Parameters

<i>base</i>	I2S base pointer.
-------------	-------------------

Returns

kStatus\_Fail empty TX fifo failed, kStatus\_Success empty tx fifo success.

#### 14.8.6.21 void I2S\_TxHandleIRQ ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle* )

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to handle structure.

#### 14.8.6.22 void I2S\_RxHandleIRQ ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle* )

Parameters

<i>base</i>	I2S base pointer.
<i>handle</i>	pointer to handle structure.

# Chapter 15

## SPI: Serial Peripheral Interface Driver

### 15.1 Overview

SPI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the spi\_handle\_t as the first parameter. Initialize the handle by calling the SPI\_MasterTransferCreateHandle() or SPI\_SlaveTransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SPI\_MasterTransferNonBlocking() and SPI\_SlaveTransferNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus\_SPI\_Idle status.

### 15.2 Typical use case

#### 15.2.1 SPI master transfer using an interrupt method

```
#define BUFFER_LEN (64)
spi_master_handle_t spiHandle;
spi_master_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished = false;

const uint8_t sendData[BUFFER_LEN] = [.....];
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_master_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    SPI_MasterGetDefaultConfig(&masterConfig);

    SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);
    SPI_MasterTransferCreateHandle(SPI0, &spiHandle, SPI_UserCallback, NULL);

    // Prepare to send.
```

```

xfer.txData = sendData;
xfer.rxData = receiveBuff;
xfer.dataSize = sizeof(sendData);

// Send out.
SPI_MasterTransferNonBlocking(SPI0, &spiHandle, &xfer);

// Wait send finished.
while (!isFinished)
{
}

// ...
}

```

### 15.2.2 SPI Send/receive using a DMA method

```

#define BUFFER_LEN (64)
spi_dma_handle_t spiHandle;
dma_handle_t g_spiTxDmaHandle;
dma_handle_t g_spiRxDmaHandle;
spi_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished;

uint8_t sendData[BUFFER_LEN] = ...;
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    // Initialize DMA peripheral
    DMA_Init(DMA0);

    // Initialize SPI peripheral
    SPI_MasterGetDefaultConfig(&masterConfig);
    masterConfig.sselNum = SPI_SSEL;
    SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);

    // Enable DMA channels connected to SPI0 Tx/SPI0 Rx request lines
    DMA_EnableChannel(SPI0, SPI_MASTER_TX_CHANNEL);
    DMA_EnableChannel(SPI0, SPI_MASTER_RX_CHANNEL);

    // Set DMA channels priority
    DMA_SetChannelPriority(SPI0, SPI_MASTER_TX_CHANNEL,
        kDMA_ChannelPriority3);
    DMA_SetChannelPriority(SPI0, SPI_MASTER_RX_CHANNEL,
        kDMA_ChannelPriority2);

    // Creates the DMA handle.
    DMA_CreateHandle(&masterTxHandle, SPI0, SPI_MASTER_TX_CHANNEL);
    DMA_CreateHandle(&masterRxHandle, SPI0, SPI_MASTER_RX_CHANNEL);

    // Create SPI DMA handle
    SPI_MasterTransferCreateHandleDMA(SPI0, spiHandle, SPI_UserCallback,
        NULL, &g_spiTxDmaHandle, &g_spiRxDmaHandle);

    // Prepares to send.
    xfer.txData = sendData;
}

```

```
xfer.rxData = receiveBuff;
xfer.dataSize = sizeof(sendData);

// Sends out.
SPI_MasterTransferDMA(SPI0, &spiHandle, &xfer);

// Waits for send to complete.
while (!isFinished)
{
}

// ...
}
```

## Modules

- SPI CMSIS driver
- SPI DMA Driver
- SPI Driver
- SPI FreeRTOS driver

## 15.3 SPI Driver

### 15.3.1 Overview

This section describes the programming interface of the SPI DMA driver.

## Files

- file [fsl\\_spi.h](#)

## Data Structures

- struct [spi\\_delay\\_config\\_t](#)  
*SPI delay time configure structure. [More...](#)*
- struct [spi\\_master\\_config\\_t](#)  
*SPI master user configure structure. [More...](#)*
- struct [spi\\_slave\\_config\\_t](#)  
*SPI slave user configure structure. [More...](#)*
- struct [spi\\_transfer\\_t](#)  
*SPI transfer structure. [More...](#)*
- struct [spi\\_half\\_duplex\\_transfer\\_t](#)  
*SPI half-duplex(master only) transfer structure. [More...](#)*
- struct [spi\\_config\\_t](#)  
*Internal configuration structure used in 'spi' and 'spi\_dma' driver. [More...](#)*
- struct [spi\\_master\\_handle\\_t](#)  
*SPI transfer handle structure. [More...](#)*

## Macros

- #define [SPI\\_DUMMYDATA](#) (0xFFU)  
*SPI dummy transfer data, the data is sent while txBuff is NULL.*
- #define [SPI\\_RETRY\\_TIMES](#) 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*

## Typedefs

- typedef spi\_master\_handle\_t [spi\\_slave\\_handle\\_t](#)  
*Slave handle type.*
- typedef void(\* [spi\\_master\\_callback\\_t](#)) (SPI\_Type \*base, spi\_master\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*SPI master callback for finished transmit.*
- typedef void(\* [spi\\_slave\\_callback\\_t](#)) (SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle, [status\\_t](#) status, void \*userData)  
*SPI slave callback for finished transmit.*

- `typedef void(* flexcomm_spi_master_irq_handler_t )(SPI_Type *base, spi_master_handle_t *handle)`  
*Typedef for master interrupt handler.*
- `typedef void(* flexcomm_spi_slave_irq_handler_t )(SPI_Type *base, spi_slave_handle_t *handle)`  
*Typedef for slave interrupt handler.*

## Enumerations

- enum `spi_xfer_option_t` {
   
  `kSPI_FrameDelay` = (SPI\_FIFOWR\_EOF\_MASK),
   
  `kSPI_FrameAssert` = (SPI\_FIFOWR\_EOT\_MASK) }
   
*SPI transfer option.*
- enum `spi_shift_direction_t` {
   
  `kSPI_MsbFirst` = 0U,
   
  `kSPI_LsbFirst` = 1U }
   
*SPI data shifter direction options.*
- enum `spi_clock_polarity_t` {
   
  `kSPI_ClockPolarityActiveHigh` = 0x0U,
   
  `kSPI_ClockPolarityActiveLow` }
   
*SPI clock polarity configuration.*
- enum `spi_clock_phase_t` {
   
  `kSPI_ClockPhaseFirstEdge` = 0x0U,
   
  `kSPI_ClockPhaseSecondEdge` }
   
*SPI clock phase configuration.*
- enum `spi_txfifo_watermark_t` {
   
  `kSPI_TxFifo0` = 0,
   
  `kSPI_TxFifo1` = 1,
   
  `kSPI_TxFifo2` = 2,
   
  `kSPI_TxFifo3` = 3,
   
  `kSPI_TxFifo4` = 4,
   
  `kSPI_TxFifo5` = 5,
   
  `kSPI_TxFifo6` = 6,
   
  `kSPI_TxFifo7` = 7 }
   
*txFIFO watermark values*
- enum `spi_rxfifo_watermark_t` {
   
  `kSPI_RxFifo1` = 0,
   
  `kSPI_RxFifo2` = 1,
   
  `kSPI_RxFifo3` = 2,
   
  `kSPI_RxFifo4` = 3,
   
  `kSPI_RxFifo5` = 4,
   
  `kSPI_RxFifo6` = 5,
   
  `kSPI_RxFifo7` = 6,
   
  `kSPI_RxFifo8` = 7 }
   
*rxFIFO watermark values*
- enum `spi_data_width_t` {

```

kSPI_Data4Bits = 3,
kSPI_Data5Bits = 4,
kSPI_Data6Bits = 5,
kSPI_Data7Bits = 6,
kSPI_Data8Bits = 7,
kSPI_Data9Bits = 8,
kSPI_Data10Bits = 9,
kSPI_Data11Bits = 10,
kSPI_Data12Bits = 11,
kSPI_Data13Bits = 12,
kSPI_Data14Bits = 13,
kSPI_Data15Bits = 14,
kSPI_Data16Bits = 15 }

```

*Transfer data width.*

- enum `spi_ssel_t` {
   
kSPI\_Ssel0 = 0,
   
kSPI\_Ssel1 = 1,
   
kSPI\_Ssel2 = 2,
   
kSPI\_Ssel3 = 3 }

*Slave select.*

- enum `spi_spol_t`
  
*ssel polarity*
- enum {
   
kStatus\_SPI\_Busy = MAKE\_STATUS(kStatusGroup\_LPC\_SPI, 0),
   
kStatus\_SPI\_Idle = MAKE\_STATUS(kStatusGroup\_LPC\_SPI, 1),
   
kStatus\_SPI\_Error = MAKE\_STATUS(kStatusGroup\_LPC\_SPI, 2),
   
kStatus\_SPI\_BaudrateNotSupport,
   
kStatus\_SPI\_Timeout = MAKE\_STATUS(kStatusGroup\_LPC\_SPI, 4) }

*SPI transfer status.*

- enum `_spi_interrupt_enable` {
   
kSPI\_RxLvlIrq = SPI\_FIFOINTENSET\_RXLVL\_MASK,
   
kSPI\_TxLvlIrq = SPI\_FIFOINTENSET\_TXLVL\_MASK }
- enum `_spi_statusflags` {
   
kSPI\_TxEmptyFlag = SPI\_FIFOSTAT\_TXEMPTY\_MASK,
   
kSPI\_TxNotFullFlag = SPI\_FIFOSTAT\_TXNOTFULL\_MASK,
   
kSPI\_RxNotEmptyFlag = SPI\_FIFOSTAT\_RXNOTEEMPTY\_MASK,
   
kSPI\_RxFullFlag = SPI\_FIFOSTAT\_RXFULL\_MASK }

*SPI status flags.*

## Variables

- volatile uint8\_t `s_dummyData` []
   
*Global variable for dummy data value setting.*

## Driver version

- #define `FSL_SPI_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 1)`)  
*SPI driver version.*

### 15.3.2 Data Structure Documentation

#### 15.3.2.1 `struct spi_delay_config_t`

Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maximum value of these delay time is 15.

#### Data Fields

- `uint8_t preDelay`  
*Delay between SSEL assertion and the beginning of transfer.*
- `uint8_t postDelay`  
*Delay between the end of transfer and SSEL deassertion.*
- `uint8_t frameDelay`  
*Delay between frame to frame.*
- `uint8_t transferDelay`  
*Delay between transfer to transfer.*

#### Field Documentation

- (1) `uint8_t spi_delay_config_t::preDelay`
- (2) `uint8_t spi_delay_config_t::postDelay`
- (3) `uint8_t spi_delay_config_t::frameDelay`
- (4) `uint8_t spi_delay_config_t::transferDelay`

#### 15.3.2.2 `struct spi_master_config_t`

#### Data Fields

- `bool enableLoopback`  
*Enable loopback for test purpose.*
- `bool enableMaster`  
*Enable SPI at initialization time.*
- `spi_clock_polarity_t polarity`  
*Clock polarity.*
- `spi_clock_phase_t phase`  
*Clock phase.*
- `spi_shift_direction_t direction`  
*MSB or LSB.*
- `uint32_t baudRate_Bps`

- **spi\_data\_width\_t dataWidth**  
*Width of the data.*
- **spi\_ssel\_t sselNum**  
*Slave select number.*
- **spi\_spol\_t sselPol**  
*Configure active CS polarity.*
- **uint8\_t txWatermark**  
*txFIFO watermark*
- **uint8\_t rxWatermark**  
*rxFIFO watermark*
- **spi\_delay\_config\_t delayConfig**  
*Delay configuration.*

## Field Documentation

### (1) spi\_delay\_config\_t spi\_master\_config\_t::delayConfig

#### 15.3.2.3 struct spi\_slave\_config\_t

## Data Fields

- **bool enableSlave**  
*Enable SPI at initialization time.*
- **spi\_clock\_polarity\_t polarity**  
*Clock polarity.*
- **spi\_clock\_phase\_t phase**  
*Clock phase.*
- **spi\_shift\_direction\_t direction**  
*MSB or LSB.*
- **spi\_data\_width\_t dataWidth**  
*Width of the data.*
- **spi\_spol\_t sselPol**  
*Configure active CS polarity.*
- **uint8\_t txWatermark**  
*txFIFO watermark*
- **uint8\_t rxWatermark**  
*rxFIFO watermark*

#### 15.3.2.4 struct spi\_transfer\_t

## Data Fields

- **uint8\_t \* txData**  
*Send buffer.*
- **uint8\_t \* rxData**  
*Receive buffer.*
- **uint32\_t configFlags**  
*Additional option to control transfer, [spi\\_xfer\\_option\\_t](#).*
- **size\_t dataSize**

*Transfer bytes.*

### Field Documentation

(1) `uint32_t spi_transfer_t::configFlags`

**15.3.2.5 struct spi\_half\_duplex\_transfer\_t**

### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `size_t txDataSize`  
*Transfer bytes for transmit.*
- `size_t rxDataSize`  
*Transfer bytes.*
- `uint32_t configFlags`  
*Transfer configuration flags, [spi\\_xfer\\_option\\_t](#).*
- `bool isPcsAssertInTransfer`  
*If PCS pin keep assert between transmit and receive.*
- `bool isTransmitFirst`  
*True for transmit first and false for receive first.*

### Field Documentation

(1) `uint32_t spi_half_duplex_transfer_t::configFlags`

(2) `bool spi_half_duplex_transfer_t::isPcsAssertInTransfer`

true for assert and false for deassert.

(3) `bool spi_half_duplex_transfer_t::isTransmitFirst`

**15.3.2.6 struct spi\_config\_t**

**15.3.2.7 struct \_spi\_master\_handle**

Master handle type.

### Data Fields

- `uint8_t *volatile txData`  
*Transfer buffer.*
- `uint8_t *volatile rxData`  
*Receive buffer.*
- `volatile size_t txRemainingBytes`  
*Number of data to be transmitted [in bytes].*
- `volatile size_t rxRemainingBytes`

- **volatile int8\_t toReceiveCount**  
*Number of data to be received [in bytes].*
- **size\_t totalByteCount**  
*The number of data expected to receive in data width.*
- **volatile uint32\_t state**  
*A number of transfer bytes.*
- **spi\_master\_callback\_t callback**  
*SPI internal state.*
- **void \*userData**  
*SPI callback.*
- **uint8\_t dataWidth**  
*Callback parameter.*
- **uint8\_t sselNum**  
*Width of the data [Valid values: 1 to 16].*
- **uint32\_t configFlags**  
*Slave select number to be asserted when transferring data [Valid values: 0 to 3].*
- **uint8\_t txWatermark**  
*Additional option to control transfer.*
- **uint8\_t rxWatermark**  
*txFIFO watermark*
- **uint8\_t rxWatermark**  
*rxFIFO watermark*

## Field Documentation

### (1) volatile int8\_t spi\_master\_handle\_t::toReceiveCount

Since the received count and sent count should be the same to complete the transfer, if the sent count is x and the received count is y, toReceiveCount is x-y.

## 15.3.3 Macro Definition Documentation

### 15.3.3.1 #define FSL\_SPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 1))

### 15.3.3.2 #define SPI\_DUMMYDATA (0xFFU)

### 15.3.3.3 #define SPI\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/

## 15.3.4 Typedef Documentation

### 15.3.4.1 typedef void(\* flexcomm\_spi\_master\_irq\_handler\_t)(SPI\_Type \*base, spi\_master\_handle\_t \*handle)

### 15.3.4.2 typedef void(\* flexcomm\_spi\_slave\_irq\_handler\_t)(SPI\_Type \*base, spi\_slave\_handle\_t \*handle)

### 15.3.5 Enumeration Type Documentation

#### 15.3.5.1 enum spi\_xfer\_option\_t

Enumerator

**kSPI\_FrameDelay** A delay may be inserted, defined in the DLY register.

**kSPI\_FrameAssert** SSEL will be deasserted at the end of a transfer.

#### 15.3.5.2 enum spi\_shift\_direction\_t

Enumerator

**kSPI\_MsbFirst** Data transfers start with most significant bit.

**kSPI\_LsbFirst** Data transfers start with least significant bit.

#### 15.3.5.3 enum spi\_clock\_polarity\_t

Enumerator

**kSPI\_ClockPolarityActiveHigh** Active-high SPI clock (idles low).

**kSPI\_ClockPolarityActiveLow** Active-low SPI clock (idles high).

#### 15.3.5.4 enum spi\_clock\_phase\_t

Enumerator

**kSPI\_ClockPhaseFirstEdge** First edge on SCK occurs at the middle of the first cycle of a data transfer.

**kSPI\_ClockPhaseSecondEdge** First edge on SCK occurs at the start of the first cycle of a data transfer.

#### 15.3.5.5 enum spi\_txfifo\_watermark\_t

Enumerator

**kSPI\_TxFifo0** SPI tx watermark is empty.

**kSPI\_TxFifo1** SPI tx watermark at 1 item.

**kSPI\_TxFifo2** SPI tx watermark at 2 items.

**kSPI\_TxFifo3** SPI tx watermark at 3 items.

**kSPI\_TxFifo4** SPI tx watermark at 4 items.

**kSPI\_TxFifo5** SPI tx watermark at 5 items.

**kSPI\_TxFifo6** SPI tx watermark at 6 items.

**kSPI\_TxFifo7** SPI tx watermark at 7 items.

### 15.3.5.6 enum spi\_rx\_fifo\_watermark\_t

Enumerator

- kSPI\_RxFifo1* SPI rx watermark at 1 item.
- kSPI\_RxFifo2* SPI rx watermark at 2 items.
- kSPI\_RxFifo3* SPI rx watermark at 3 items.
- kSPI\_RxFifo4* SPI rx watermark at 4 items.
- kSPI\_RxFifo5* SPI rx watermark at 5 items.
- kSPI\_RxFifo6* SPI rx watermark at 6 items.
- kSPI\_RxFifo7* SPI rx watermark at 7 items.
- kSPI\_RxFifo8* SPI rx watermark at 8 items.

### 15.3.5.7 enum spi\_data\_width\_t

Enumerator

- kSPI\_Data4Bits* 4 bits data width
- kSPI\_Data5Bits* 5 bits data width
- kSPI\_Data6Bits* 6 bits data width
- kSPI\_Data7Bits* 7 bits data width
- kSPI\_Data8Bits* 8 bits data width
- kSPI\_Data9Bits* 9 bits data width
- kSPI\_Data10Bits* 10 bits data width
- kSPI\_Data11Bits* 11 bits data width
- kSPI\_Data12Bits* 12 bits data width
- kSPI\_Data13Bits* 13 bits data width
- kSPI\_Data14Bits* 14 bits data width
- kSPI\_Data15Bits* 15 bits data width
- kSPI\_Data16Bits* 16 bits data width

### 15.3.5.8 enum spi\_ssel\_t

Enumerator

- kSPI\_Ssel0* Slave select 0.
- kSPI\_Ssel1* Slave select 1.
- kSPI\_Ssel2* Slave select 2.
- kSPI\_Ssel3* Slave select 3.

### 15.3.5.9 anonymous enum

Enumerator

- kStatus\_SPI\_Busy* SPI bus is busy.

*kStatus\_SPI\_Idle* SPI is idle.

*kStatus\_SPI\_Error* SPI error.

*kStatus\_SPI\_BaudrateNotSupport* Baudrate is not support in current clock source.

*kStatus\_SPI\_Timeout* SPI timeout polling status flags.

### 15.3.5.10 enum \_spi\_interrupt\_enable

Enumerator

*kSPI\_RxLvlIrq* Rx level interrupt.

*kSPI\_TxLvlIrq* Tx level interrupt.

### 15.3.5.11 enum \_spi\_statusflags

Enumerator

*kSPI\_TxEmptyFlag* txFifo is empty

*kSPI\_TxNotFullFlag* txFifo is not full

*kSPI\_RxNotEmptyFlag* rxFIFO is not empty

*kSPI\_RxFullFlag* rxFIFO is full

## 15.3.6 Variable Documentation

### 15.3.6.1 volatile uint8\_t s\_dummyData[]

## 15.4 SPI DMA Driver

### 15.4.1 Overview

This section describes the programming interface of the SPI DMA driver.

### Files

- file [fsl\\_spi\\_dma.h](#)

### Data Structures

- struct [spi\\_dma\\_handle\\_t](#)  
*SPI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

### Typedefs

- typedef void(\* [spi\\_dma\\_callback\\_t](#))([SPI\\_Type](#) \*base, [spi\\_dma\\_handle\\_t](#) \*handle, [status\\_t](#) status, void \*userData)  
*SPI DMA callback called at the end of transfer.*

### Driver version

- #define [FSL\\_SPI\\_DMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 2))  
*SPI DMA driver version 2.1.1.*

### DMA Transactional

- [status\\_t SPI\\_MasterTransferCreateHandleDMA](#) ([SPI\\_Type](#) \*base, [spi\\_dma\\_handle\\_t](#) \*handle, [spi\\_dma\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*txHandle, [dma\\_handle\\_t](#) \*rxHandle)  
*Initialize the SPI master DMA handle.*
- [status\\_t SPI\\_MasterTransferDMA](#) ([SPI\\_Type](#) \*base, [spi\\_dma\\_handle\\_t](#) \*handle, [spi\\_transfer\\_t](#) \*xfer)  
*Perform a non-blocking SPI transfer using DMA.*
- [status\\_t SPI\\_MasterHalfDuplexTransferDMA](#) ([SPI\\_Type](#) \*base, [spi\\_dma\\_handle\\_t](#) \*handle, [spi\\_half\\_duplex\\_transfer\\_t](#) \*xfer)  
*Transfers a block of data using a DMA method.*
- static [status\\_t SPI\\_SlaveTransferCreateHandleDMA](#) ([SPI\\_Type](#) \*base, [spi\\_dma\\_handle\\_t](#) \*handle, [spi\\_dma\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*txHandle, [dma\\_handle\\_t](#) \*rxHandle)  
*Initialize the SPI slave DMA handle.*
- static [status\\_t SPI\\_SlaveTransferDMA](#) ([SPI\\_Type](#) \*base, [spi\\_dma\\_handle\\_t](#) \*handle, [spi\\_transfer\\_t](#) \*xfer)  
*Perform a non-blocking SPI transfer using DMA.*

- void **SPI\_MasterTransferAbortDMA** (SPI\_Type \*base, spi\_dma\_handle\_t \*handle)  
*Abort a SPI transfer using DMA.*
- status\_t **SPI\_MasterTransferGetCountDMA** (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the master DMA transfer remaining bytes.*
- static void **SPI\_SlaveTransferAbortDMA** (SPI\_Type \*base, spi\_dma\_handle\_t \*handle)  
*Abort a SPI transfer using DMA.*
- static status\_t **SPI\_SlaveTransferGetCountDMA** (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the slave DMA transfer remaining bytes.*

## 15.4.2 Data Structure Documentation

### 15.4.2.1 struct \_spi\_dma\_handle

#### Data Fields

- volatile bool **txInProgress**  
*Send transfer finished.*
- volatile bool **rxInProgress**  
*Receive transfer finished.*
- **dma\_handle\_t \* txHandle**  
*DMA handler for SPI send.*
- **dma\_handle\_t \* rxHandle**  
*DMA handler for SPI receive.*
- uint8\_t **bytesPerFrame**  
*Bytes in a frame for SPI transfer.*
- **spi\_dma\_callback\_t callback**  
*Callback for SPI DMA transfer.*
- void \* **userData**  
*User Data for SPI DMA callback.*
- uint32\_t **state**  
*Internal state of SPI DMA transfer.*
- size\_t **transferSize**  
*Bytes need to be transfer.*

## 15.4.3 Macro Definition Documentation

### 15.4.3.1 #define FSL\_SPI\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))

## 15.4.4 Typedef Documentation

### 15.4.4.1 **typedef void(\* spi\_dma\_callback\_t)(SPI\_Type \*base, spi\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

## 15.4.5 Function Documentation

**15.4.5.1 status\_t SPI\_MasterTransferCreateHandleDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_dma\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *txHandle*, dma\_handle\_t \* *rxHandle* )**

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI handle pointer.
<i>callback</i>	User callback function called at the end of a transfer.
<i>userData</i>	User data for callback.
<i>txHandle</i>	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
<i>rxHandle</i>	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

**15.4.5.2 status\_t SPI\_MasterTransferDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* )**

Note

This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.
<i>xfer</i>	Pointer to dma transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.

<i>kStatus_SPI_Busy</i>	SPI is not idle, is running another transfer.
-------------------------	---

#### 15.4.5.3 status\_t SPI\_MasterHalfDuplexTransferDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_half\_duplex\_transfer\_t \* *xfer* )

This function using polling way to do the first half transmission and using DMA way to do the second half transmission, the transfer mechanism is half-duplex. When do the second half transmission, code will return right away. When all data is transferred, the callback function is called.

Parameters

<i>base</i>	SPI base pointer
<i>handle</i>	A pointer to the spi_master_dma_handle_t structure which stores the transfer state.
<i>xfer</i>	A pointer to the <a href="#">spi_half_duplex_transfer_t</a> structure.

Returns

status of status\_t.

#### 15.4.5.4 static status\_t SPI\_SlaveTransferCreateHandleDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_dma\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *txHandle*, dma\_handle\_t \* *rxHandle* ) [inline], [static]

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI handle pointer.
<i>callback</i>	User callback function called at the end of a transfer.
<i>userData</i>	User data for callback.
<i>txHandle</i>	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
<i>rxHandle</i>	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

#### 15.4.5.5 static status\_t SPI\_SlaveTransferDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* ) [inline], [static]

## Note

This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

## Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.
<i>xfer</i>	Pointer to dma transfer structure.

## Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_SPI_Busy</i>	SPI is not idle, is running another transfer.

**15.4.5.6 void SPI\_MasterTransferAbortDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle* )**

## Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.

**15.4.5.7 status\_t SPI\_MasterTransferGetCountDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the master DMA transfer remaining bytes.

## Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	A pointer to the spi_dma_handle_t structure which stores the transfer state.
<i>count</i>	A number of bytes transferred by the non-blocking transaction.

## Returns

status of status\_t.

15.4.5.8 **static void SPI\_SlaveTransferAbortDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle* ) [inline], [static]**

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	SPI DMA handle pointer.

#### 15.4.5.9 static status\_t SPI\_SlaveTransferGetCountDMA ( **SPI\_Type** \* *base*, **spi\_dma\_handle\_t** \* *handle*, **size\_t** \* *count* ) [inline], [static]

This function gets the slave DMA transfer remaining bytes.

Parameters

<i>base</i>	SPI peripheral base address.
<i>handle</i>	A pointer to the <b>spi_dma_handle_t</b> structure which stores the transfer state.
<i>count</i>	A number of bytes transferred by the non-blocking transaction.

Returns

status of **status\_t**.

## 15.5 SPI FreeRTOS driver

### 15.5.1 Overview

This section describes the programming interface of the SPI FreeRTOS driver.

### Files

- file [fsl\\_spi\\_freertos.h](#)

### Data Structures

- struct [spi\\_rtos\\_handle\\_t](#)  
*SPI FreeRTOS handle. [More...](#)*

### Driver version

- #define [FSL\\_SPI\\_FREERTOS\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 0))  
*SPI FreeRTOS driver version 2.1.0.*

### SPI RTOS Operation

- [status\\_t SPI\\_RTOS\\_Init](#) ([spi\\_rtos\\_handle\\_t](#) \*handle, [SPI\\_Type](#) \*base, const [spi\\_master\\_config\\_t](#) \*masterConfig, [uint32\\_t](#) srcClock\_Hz)  
*Initializes SPI.*
- [status\\_t SPI\\_RTOS\\_Deinit](#) ([spi\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes the SPI.*
- [status\\_t SPI\\_RTOS\\_Transfer](#) ([spi\\_rtos\\_handle\\_t](#) \*handle, [spi\\_transfer\\_t](#) \*transfer)  
*Performs SPI transfer.*

### 15.5.2 Data Structure Documentation

#### 15.5.2.1 struct spi\_rtos\_handle\_t

##### Data Fields

- [SPI\\_Type](#) \* [base](#)  
*SPI base address.*
- [spi\\_master\\_handle\\_t](#) [drv\\_handle](#)  
*Handle of the underlying driver, treated as opaque by the RTOS layer.*
- [SemaphoreHandle\\_t](#) [mutex](#)  
*Mutex to lock the handle during a trasfer.*
- [SemaphoreHandle\\_t](#) [event](#)

*Semaphore to notify and unblock task when transfer ends.*

### 15.5.3 Macro Definition Documentation

#### 15.5.3.1 #define FSL\_SPI\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

### 15.5.4 Function Documentation

#### 15.5.4.1 status\_t SPI\_RTOS\_Init ( *spi\_rtos\_handle\_t \* handle*, *SPI\_Type \* base*, *const spi\_master\_config\_t \* masterConfig*, *uint32\_t srcClock\_Hz* )

This function initializes the SPI module and related RTOS context.

Parameters

<i>handle</i>	The RTOS SPI handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the SPI instance to initialize.
<i>masterConfig</i>	Configuration structure to set-up SPI in master mode.
<i>srcClock_Hz</i>	Frequency of input clock of the SPI module.

Returns

status of the operation.

#### 15.5.4.2 status\_t SPI\_RTOS\_Deinit ( *spi\_rtos\_handle\_t \* handle* )

This function deinitializes the SPI module and related RTOS context.

Parameters

<i>handle</i>	The RTOS SPI handle.
---------------	----------------------

#### 15.5.4.3 status\_t SPI\_RTOS\_Transfer ( *spi\_rtos\_handle\_t \* handle*, *spi\_transfer\_t \* transfer* )

This function performs an SPI transfer according to data given in the transfer structure.

## Parameters

<i>handle</i>	The RTOS SPI handle.
<i>transfer</i>	Structure specifying the transfer parameters.

## Returns

status of the operation.

## 15.6 SPI CMSIS driver

This section describes the programming interface of the SPI Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

### 15.6.1 Function groups

#### 15.6.1.1 SPI CMSIS GetVersion Operation

This function group will return the SPI CMSIS Driver version to user.

#### 15.6.1.2 SPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

#### 15.6.1.3 SPI CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialized the instance in master mode or slave mode. And this API must be called before you configure an instance or after you Deinit an instance. The right steps to start an instance is that you must initialize the instance which been selected firstly, then you can power on the instance. After these all have been done, you can configure the instance by using control operation. If you want to Uninitialize the instance, you must power off the instance first.

#### 15.6.1.4 SPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

#### 15.6.1.5 SPI CMSIS Status Operation

This function group gets the SPI transfer status.

#### 15.6.1.6 SPI CMSIS Control Operation

This function can configure instance as master mode or slave mode, set baudrate for master mode transfer, get current baudrate of master mode transfer, set transfer data bits and other control command.

## 15.6.2 Typical use case

### 15.6.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*SPI master init*/
DRIVER_MASTER_SPI.Initialize(SPI_MasterSignalEvent_t);
DRIVER_MASTER_SPI.PowerControl(ARM_POWER_FULL);
DRIVER_MASTER_SPI.Control(ARM_SPI_MODE_MASTER, TRANSFER_BAUDRATE);

/* Start master transfer */
DRIVER_MASTER_SPI.Transfer(masterTxData, masterRxData, TRANSFER_SIZE);

/* Master power off */
DRIVER_MASTER_SPI.PowerControl(ARM_POWER_OFF);

/* Master uninitialized */
DRIVER_MASTER_SPI.Uninitialize();
```

### 15.6.2.2 Slave Operation

```
/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*SPI slave init*/
DRIVER_SLAVE_SPI.Initialize(SPI_SlaveSignalEvent_t);
DRIVER_SLAVE_SPI.PowerControl(ARM_POWER_FULL);
DRIVER_SLAVE_SPI.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
DRIVER_SLAVE_SPI.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
DRIVER_SLAVE_SPI.PowerControl(ARM_POWER_OFF);

/* slave uninitialized */
DRIVER_SLAVE_SPI.Uninitialize();
```

# Chapter 16

## USART: Universal Synchronous/Asynchronous Receiver-/Transmitter Driver

### 16.1 Overview

The MCUXpresso SDK provides a peripheral UART driver for the Universal Synchronous Receiver-/Transmitter (USART) module of MCUXpresso SDK devices. Driver does not support synchronous mode.

The USART driver includes two parts: functional APIs and transactional APIs.

Functional APIs are used for USART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the USART peripheral and know how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. USART functional operation groups provide the functional APIs set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `uart_handle_t` as the second parameter. Initialize the handle by calling the [USART\\_TransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer, which means that the functions [USART\\_TransferSendNonBlocking\(\)](#) and [USART\\_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_USART_TxIdle` and `kStatus_USART_RxIdle`.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the [USART\\_TransferCreateHandle\(\)](#). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The [USART\\_TransferReceiveNonBlocking\(\)](#) function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the `kStatus_USART_RxIdle`.

If the receive ring buffer is full, the upper layer is informed through a callback with the `kStatus_USART_RxRingBufferOverrun`. In the callback function, the upper layer reads data out from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code:

```
USART_TransferCreateHandle(USART0, &handle, USART_UserCallback, NULL);
```

In this example, the buffer size is 32, but only 31 bytes are used for saving data.

## 16.2 Typical use case

### 16.2.1 USART Send/receive using a polling method

```

uint8_t ch;
USART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableTx = true;
user_config.enableRx = true;

USART_Init(USART1, &user_config, 120000000U);

while(1)
{
    USART_ReadBlocking(USART1, &ch, 1);
    USART_WriteBlocking(USART1, &ch, 1);
}

```

### 16.2.2 USART Send/receive using an interrupt method

```

uart_handle_t g_usartHandle;
uart_config_t user_config;
uart_transfer_t sendXfer;
uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void USART_UserCallback(uart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);
    USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);

    // Prepare to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData);
    txFinished = false;

    // Send out.
    USART_TransferSendNonBlocking(USART1, &g_usartHandle, &sendXfer);
}

```

```

// Wait send finished.
while (!txFinished)
{
}

// Prepare to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData);
rxFinished = false;

// Receive.
USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer,
                                NULL);

// Wait receive finished.
while (!rxFinished)
{
}

// ...
}

```

### 16.2.3 USART Receive using the ringbuffer feature

```

#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE      32

uart_handle_t g_usartHandle;
uart_config_t user_config;
uart_transfer_t sendXfer;
uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);
    USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);
    USART_TransferStartRingBuffer(USART1, &g_usartHandle, ringBuffer,
                                 RING_BUFFER_SIZE);
    // Now the RX is working in background, receive in to ring buffer.

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
}

```

```

rxFinished = false;

// Receive.
USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer);

if (bytesRead == RX_DATA_SIZE) /* Have read enough data. */
{
    ;
}
else
{
    if (bytesRead) /* Received some data, process first. */
    {
        ;
    }

    // Wait receive finished.
    while (!rxFinished)
    {
    }
}

// ...
}

```

#### 16.2.4 USART Send/Receive using the DMA method

```

uart_handle_t g_usartHandle;
dma_handle_t g_usartTxDmaHandle;
dma_handle_t g_usartRxDmaHandle;
uart_config_t user_config;
uart_transfer_t sendXfer;
uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 12000000U);

    // Set up the DMA
}

```

```

DMA_Init(DMA0);
DMA_EnableChannel(DMA0, USART_TX_DMA_CHANNEL);
DMA_EnableChannel(DMA0, USART_RX_DMA_CHANNEL);

DMA_CreateHandle(&g_usartTxDmaHandle, DMA0, USART_TX_DMA_CHANNEL);
DMA_CreateHandle(&g_usartRxDmaHandle, DMA0, USART_RX_DMA_CHANNEL);

USART_TransferCreateHandleDMA(USART1, &g_usartHandle, USART_UserCallback,
    NULL, &g_usartTxDmaHandle, &g_usartRxDmaHandle);

// Prepare to send.
sendXfer.data = sendData
sendXfer.dataSize = sizeof(sendData);
txFinished = false;

// Send out.
USART_TransferSendDMA(USART1, &g_usartHandle, &sendXfer);

// Wait send finished.
while (!txFinished)
{
}

// Prepare to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData);
rxFinished = false;

// Receive.
USART_TransferReceiveDMA(USART1, &g_usartHandle, &receiveXfer);

// Wait receive finished.
while (!rxFinished)
{
}

// ...
}

```

## Modules

- [USART CMSIS Driver](#)
- [USART DMA Driver](#)
- [USART Driver](#)
- [USART FreeRTOS Driver](#)

## 16.3 USART Driver

### 16.3.1 Overview

#### Data Structures

- struct `usart_config_t`  
*USART configuration structure.* [More...](#)
- struct `usart_transfer_t`  
*USART transfer structure.* [More...](#)
- struct `usart_handle_t`  
*USART handle structure.* [More...](#)

#### Macros

- #define `UART_RETRY_TIMES` 0U  
*Retry times for waiting flag.*

#### Typedefs

- typedef void(\* `usart_transfer_callback_t` )(USART\_Type \*base, usart\_handle\_t \*handle, `status_t` status, void \*userData)  
*USART transfer callback function.*
- typedef void(\* `flexcomm_usart_irq_handler_t` )(USART\_Type \*base, usart\_handle\_t \*handle)  
*Typedef for usart interrupt handler.*

#### Enumerations

- enum {
 `kStatus_USART_TxBusy` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 0),
 `kStatus_USART_RxBusy` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 1),
 `kStatus_USART_TxIdle` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 2),
 `kStatus_USART_RxIdle` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 3),
 `kStatus_USART_TxError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 7),
 `kStatus_USART_RxError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 9),
 `kStatus_USART_RxRingBufferOverrun` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 8),
 `kStatus_USART_NoiseError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 10),
 `kStatus_USART_FramingError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 11),
 `kStatus_USART_ParityError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 12),
 `kStatus_USART_BaudrateNotSupport` }
   
*Error codes for the USART driver.*
- enum `usart_sync_mode_t` {
 `kUSART_SyncModeDisabled` = 0x0U,
 `kUSART_SyncModeSlave` = 0x2U,

- ```

kUSART_SyncModeMaster = 0x3U }

    USART synchronous mode.

• enum usart_parity_mode_t {
    kUSART_ParityDisabled = 0x0U,
    kUSART_ParityEven = 0x2U,
    kUSART_ParityOdd = 0x3U }

    USART parity mode.

• enum usart_stop_bit_count_t {
    kUSART_OneStopBit = 0U,
    kUSART_TwoStopBit = 1U }

    USART stop bit count.

• enum usart_data_len_t {
    kUSART_7BitsPerChar = 0U,
    kUSART_8BitsPerChar = 1U }

    USART data size.

• enum usart_clock_polarity_t {
    kUSART_RxSampleOnFallingEdge = 0x0U,
    kUSART_RxSampleOnRisingEdge = 0x1U }

    USART clock polarity configuration, used in sync mode.

• enum usart_txfifo_watermark_t {
    kUSART_TxFifo0 = 0,
    kUSART_TxFifo1 = 1,
    kUSART_TxFifo2 = 2,
    kUSART_TxFifo3 = 3,
    kUSART_TxFifo4 = 4,
    kUSART_TxFifo5 = 5,
    kUSART_TxFifo6 = 6,
    kUSART_TxFifo7 = 7 }

    txFIFO watermark values

• enum usart_rxfifo_watermark_t {
    kUSART_RxFifo1 = 0,
    kUSART_RxFifo2 = 1,
    kUSART_RxFifo3 = 2,
    kUSART_RxFifo4 = 3,
    kUSART_RxFifo5 = 4,
    kUSART_RxFifo6 = 5,
    kUSART_RxFifo7 = 6,
    kUSART_RxFifo8 = 7 }

    rxFIFO watermark values

• enum _uart_interrupt_enable { ,

```

```

kUSART_TxIdleInterruptEnable = (USART_INTENSET_TXIDLEEN_MASK << 16U),
kUSART_CtsChangeInterruptEnable,
kUSART_RxBreakChangeInterruptEnable,
kUSART_RxStartInterruptEnable = (USART_INTENSET_STARTEN_MASK),
kUSART_FramingErrorInterruptEnable = (USART_INTENSET_FRAMERREN_MASK),
kUSART_ParityErrorInterruptEnable = (USART_INTENSET_PARITYERREN_MASK),
kUSART_NoiseErrorInterruptEnable = (USART_INTENSET_RXNOISEEN_MASK),
kUSART_AutoBaudErrorInterruptEnable = (USART_INTENSET_ABERREN_MASK) }

```

*USART interrupt configuration structure, default settings all disabled.*

- enum `_uart_flags` {
 

```

kUSART_TxError = (USART_FIFOSTAT_TXERR_MASK),
kUSART_RxError = (USART_FIFOSTAT_RXERR_MASK),
kUSART_TxFifoEmptyFlag = (USART_FIFOSTAT_TXEMPTY_MASK),
kUSART_TxFifoNotFullFlag = (USART_FIFOSTAT_TXNOTFULL_MASK),
kUSART_RxFifoNotEmptyFlag = (USART_FIFOSTAT_RXNOTEEMPTY_MASK),
kUSART_RxFifoFullFlag = (USART_FIFOSTAT_RXFULL_MASK),
kUSART_RxIdleFlag = (USART_STAT_RXIDLE_MASK << 16U),
kUSART_TxIdleFlag = (USART_STAT_TXIDLE_MASK << 16U),
kUSART_CtsAssertFlag = (USART_STAT_CTS_MASK << 16U),
kUSART_CtsChangeFlag = (USART_STAT_DELTACTS_MASK << 16U),
kUSART_BreakDetectFlag = (USART_STAT_RXBRK_MASK),
kUSART_BreakDetectChangeFlag = (USART_STAT_DELTARXBRK_MASK),
kUSART_RxStartFlag = (USART_STAT_START_MASK),
kUSART_FramingErrorFlag = (USART_STAT_FRAMERRINT_MASK),
kUSART_ParityErrorFlag = (USART_STAT_PARITYERRINT_MASK),
kUSART_NoiseErrorFlag = (USART_STAT_RXNOISEINT_MASK),
kUSART_AutobaudErrorFlag = (USART_STAT_ABERR_MASK) }

```

*USART status flags.*

## Functions

- `uint32_t USART_GetInstance (USART_Type *base)`  
*Returns instance number for USART peripheral base address.*

## Driver version

- `#define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 8, 0))`  
*USART driver version.*

## Initialization and deinitialization

- `status_t USART_Init (USART_Type *base, const usart_config_t *config, uint32_t srcClock_Hz)`  
*Initializes a USART instance with user configuration structure and peripheral clock.*

- void **USART\_Deinit** (USART\_Type \*base)  
*Deinitializes a USART instance.*
- void **USART\_GetDefaultConfig** (uart\_config\_t \*config)  
*Gets the default configuration structure.*
- status\_t **USART\_SetBaudRate** (USART\_Type \*base, uint32\_t baudrate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the USART instance baud rate.*
- status\_t **USART\_Enable32kMode** (USART\_Type \*base, uint32\_t baudRate\_Bps, bool enableMode32k, uint32\_t srcClock\_Hz)  
*Enable 32 kHz mode which USART uses clock from the RTC oscillator as the clock source.*
- void **USART\_Enable9bitMode** (USART\_Type \*base, bool enable)  
*Enable 9-bit data mode for USART.*
- static void **USART\_SetMatchAddress** (USART\_Type \*base, uint8\_t address)  
*Set the USART slave address.*
- static void **USART\_EnableMatchAddress** (USART\_Type \*base, bool match)  
*Enable the USART match address feature.*

## Status

- static uint32\_t **USART\_GetStatusFlags** (USART\_Type \*base)  
*Get USART status flags.*
- static void **USART\_ClearStatusFlags** (USART\_Type \*base, uint32\_t mask)  
*Clear USART status flags.*

## Interrupts

- static void **USART\_EnableInterrupts** (USART\_Type \*base, uint32\_t mask)  
*Enables USART interrupts according to the provided mask.*
- static void **USART\_DisableInterrupts** (USART\_Type \*base, uint32\_t mask)  
*Disables USART interrupts according to a provided mask.*
- static uint32\_t **USART\_GetEnabledInterrupts** (USART\_Type \*base)  
*Returns enabled USART interrupts.*
- static void **USART\_EnableTxDMA** (USART\_Type \*base, bool enable)  
*Enable DMA for Tx.*
- static void **USART\_EnableRxDMA** (USART\_Type \*base, bool enable)  
*Enable DMA for Rx.*
- static void **USART\_EnableCTS** (USART\_Type \*base, bool enable)  
*Enable CTS.*
- static void **USART\_EnableContinuousSCLK** (USART\_Type \*base, bool enable)  
*Continuous Clock generation.*
- static void **USART\_EnableAutoClearSCLK** (USART\_Type \*base, bool enable)  
*Enable Continuous Clock generation bit auto clear.*
- static void **USART\_SetRx\_fifoWatermark** (USART\_Type \*base, uint8\_t water)  
*Sets the rx FIFO watermark.*
- static void **USART\_SetTx\_fifoWatermark** (USART\_Type \*base, uint8\_t water)  
*Sets the tx FIFO watermark.*

## Bus Operations

- static void **USART\_WriteByte** (USART\_Type \*base, uint8\_t data)  
*Writes to the FIFOWR register.*
- static uint8\_t **USART\_ReadByte** (USART\_Type \*base)  
*Reads the FIFORD register directly.*
- static uint8\_t **USART\_GetRxFifoCount** (USART\_Type \*base)  
*Gets the rx FIFO data count.*
- static uint8\_t **USART\_GetTxFifoCount** (USART\_Type \*base)  
*Gets the tx FIFO data count.*
- void **USART\_SendAddress** (USART\_Type \*base, uint8\_t address)  
*Transmit an address frame in 9-bit data mode.*
- **status\_t USART\_WriteBlocking** (USART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*
- **status\_t USART\_ReadBlocking** (USART\_Type \*base, uint8\_t \*data, size\_t length)  
*Read RX data register using a blocking method.*

## Transactional

- **status\_t USART\_TransferCreateHandle** (USART\_Type \*base, usart\_handle\_t \*handle, **usart\_transfer\_callback\_t** callback, void \*userData)  
*Initializes the USART handle.*
- **status\_t USART\_TransferSendNonBlocking** (USART\_Type \*base, usart\_handle\_t \*handle, **usart\_transfer\_t** \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void **USART\_TransferStartRingBuffer** (USART\_Type \*base, usart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void **USART\_TransferStopRingBuffer** (USART\_Type \*base, usart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- size\_t **USART\_TransferGetRxRingBufferLength** (usart\_handle\_t \*handle)  
*Get the length of received data in RX ring buffer.*
- void **USART\_TransferAbortSend** (USART\_Type \*base, usart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- **status\_t USART\_TransferGetSendCount** (USART\_Type \*base, usart\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been sent out to bus.*
- **status\_t USART\_TransferReceiveNonBlocking** (USART\_Type \*base, usart\_handle\_t \*handle, **usart\_transfer\_t** \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using an interrupt method.*
- void **USART\_TransferAbortReceive** (USART\_Type \*base, usart\_handle\_t \*handle)  
*Aborts the interrupt-driven data receiving.*
- **status\_t USART\_TransferGetReceiveCount** (USART\_Type \*base, usart\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*
- void **USART\_TransferHandleIRQ** (USART\_Type \*base, usart\_handle\_t \*handle)  
*USART IRQ handle function.*

## 16.3.2 Data Structure Documentation

### 16.3.2.1 struct usart\_config\_t

#### Data Fields

- `uint32_t baudRate_Bps`  
*USART baud rate.*
- `usart_parity_mode_t parityMode`  
*Parity mode, disabled (default), even, odd.*
- `usart_stop_bit_count_t stopBitCount`  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- `usart_data_len_t bitCountPerChar`  
*Data length - 7 bit, 8 bit.*
- `bool loopback`  
*Enable peripheral loopback.*
- `bool enableRx`  
*Enable RX.*
- `bool enableTx`  
*Enable TX.*
- `bool enableContinuousSCLK`  
*USART continuous Clock generation enable in synchronous master mode.*
- `bool enableMode32k`  
*USART uses 32 kHz clock from the RTC oscillator as the clock source.*
- `bool enableHardwareFlowControl`  
*Enable hardware control RTS/CTS.*
- `usart_txfifo_watermark_t txWatermark`  
*txFIFO watermark*
- `usart_rxfifo_watermark_t rxWatermark`  
*rxFIFO watermark*
- `usart_sync_mode_t syncMode`  
*Transfer mode select - asynchronous, synchronous master, synchronous slave.*
- `usart_clock_polarity_t clockPolarity`  
*Selects the clock polarity and sampling edge in synchronous mode.*

#### Field Documentation

- (1) `bool usart_config_t::enableContinuousSCLK`
- (2) `bool usart_config_t::enableMode32k`
- (3) `usart_sync_mode_t usart_config_t::syncMode`
- (4) `usart_clock_polarity_t usart_config_t::clockPolarity`

### 16.3.2.2 struct usart\_transfer\_t

#### Data Fields

- `size_t dataSize`

- **uint8\_t \* data**  
*The byte count to be transfer.*
- **uint8\_t \* rxData**  
*The buffer of data to be transfer.*
- **const uint8\_t \* txData**  
*The buffer to receive data.*
- **uint8\_t \* txData**  
*The buffer of data to be sent.*

### Field Documentation

- (1) **uint8\_t\* usart\_transfer\_t::data**
- (2) **uint8\_t\* usart\_transfer\_t::rxData**
- (3) **const uint8\_t\* usart\_transfer\_t::txData**
- (4) **size\_t usart\_transfer\_t::dataSize**

### 16.3.2.3 struct \_usart\_handle

#### Data Fields

- **const uint8\_t \*volatile txData**  
*Address of remaining data to send.*
- **volatile size\_t txDataSize**  
*Size of the remaining data to send.*
- **size\_t txDataSizeAll**  
*Size of the data to send out.*
- **uint8\_t \*volatile rxData**  
*Address of remaining data to receive.*
- **volatile size\_t rxDataSize**  
*Size of the remaining data to receive.*
- **size\_t rxDataSizeAll**  
*Size of the data to receive.*
- **uint8\_t \* rxRingBuffer**  
*Start address of the receiver ring buffer.*
- **size\_t rxRingBufferSize**  
*Size of the ring buffer.*
- **volatile uint16\_t rxRingBufferHead**  
*Index for the driver to store received data into ring buffer.*
- **volatile uint16\_t rxRingBufferTail**  
*Index for the user to get data from the ring buffer.*
- **usart\_transfer\_callback\_t callback**  
*Callback function.*
- **void \* userData**  
*USART callback function parameter.*
- **volatile uint8\_t txState**  
*TX transfer state.*
- **volatile uint8\_t rxState**  
*RX transfer state.*
- **uint8\_t txWatermark**

- *txFIFO watermark*
- `uint8_t rxWatermark`
- *rxFIFO watermark*

### Field Documentation

- (1) `const uint8_t* volatile usart_handle_t::txData`
- (2) `volatile size_t usart_handle_t::txDataSize`
- (3) `size_t usart_handle_t::txDataSizeAll`
- (4) `uint8_t* volatile usart_handle_t::rxData`
- (5) `volatile size_t usart_handle_t::rxDataSize`
- (6) `size_t usart_handle_t::rxDataSizeAll`
- (7) `uint8_t* usart_handle_t::rxRingBuffer`
- (8) `size_t usart_handle_t::rxRingBufferSize`
- (9) `volatile uint16_t usart_handle_t::rxRingBufferHead`
- (10) `volatile uint16_t usart_handle_t::rxRingBufferTail`
- (11) `uart_transfer_callback_t usart_handle_t::callback`
- (12) `void* usart_handle_t::userData`
- (13) `volatile uint8_t usart_handle_t::txState`

### 16.3.3 Macro Definition Documentation

#### 16.3.3.1 #define FSL\_USART\_DRIVER\_VERSION (MAKE\_VERSION(2, 8, 0))

#### 16.3.3.2 #define UART\_RETRY\_TIMES 0U

Defining to zero means to keep waiting for the flag until it is assert/deassert in blocking transfer, otherwise the program will wait until the `UART_RETRY_TIMES` counts down to 0, if the flag still remains unchanged then program will return `kStatus_USART_Timeout`. It is not advised to use this macro in formal application to prevent any hardware error because the actual wait period is affected by the compiler and optimization.

### 16.3.4 Typedef Documentation

**16.3.4.1** `typedef void(* usart_transfer_callback_t)(USART_Type *base, usart_handle_t *handle, status_t status, void *userData)`

**16.3.4.2** `typedef void(* flexcomm_usart_irq_handler_t)(USART_Type *base, usart_handle_t *handle)`

## 16.3.5 Enumeration Type Documentation

### 16.3.5.1 anonymous enum

Enumerator

*kStatus\_USART\_TxBusy* Transmitter is busy.

*kStatus\_USART\_RxBusy* Receiver is busy.

*kStatus\_USART\_TxIdle* USART transmitter is idle.

*kStatus\_USART\_RxIdle* USART receiver is idle.

*kStatus\_USART\_TxError* Error happens on txFIFO.

*kStatus\_USART\_RxError* Error happens on rxFIFO.

*kStatus\_USART\_RxRingBufferOverrun* Error happens on rx ring buffer.

*kStatus\_USART\_NoiseError* USART noise error.

*kStatus\_USART\_FramingError* USART framing error.

*kStatus\_USART\_ParityError* USART parity error.

*kStatus\_USART\_BaudrateNotSupport* Baudrate is not support in current clock source.

### 16.3.5.2 enum usart\_sync\_mode\_t

Enumerator

*kUSART\_SyncModeDisabled* Asynchronous mode.

*kUSART\_SyncModeSlave* Synchronous slave mode.

*kUSART\_SyncModeMaster* Synchronous master mode.

### 16.3.5.3 enum usart\_parity\_mode\_t

Enumerator

*kUSART\_ParityDisabled* Parity disabled.

*kUSART\_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.

*kUSART\_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

#### 16.3.5.4 enum usart\_stop\_bit\_count\_t

Enumerator

*kUSART\_OneStopBit* One stop bit.

*kUSART\_TwoStopBit* Two stop bits.

#### 16.3.5.5 enum usart\_data\_len\_t

Enumerator

*kUSART\_7BitsPerChar* Seven bit mode.

*kUSART\_8BitsPerChar* Eight bit mode.

#### 16.3.5.6 enum usart\_clock\_polarity\_t

Enumerator

*kUSART\_RxSampleOnFallingEdge* Un\_RXD is sampled on the falling edge of SCLK.

*kUSART\_RxSampleOnRisingEdge* Un\_RXD is sampled on the rising edge of SCLK.

#### 16.3.5.7 enum usart\_txfifo\_watermark\_t

Enumerator

*kUSART\_TxFifo0* USART tx watermark is empty.

*kUSART\_TxFifo1* USART tx watermark at 1 item.

*kUSART\_TxFifo2* USART tx watermark at 2 items.

*kUSART\_TxFifo3* USART tx watermark at 3 items.

*kUSART\_TxFifo4* USART tx watermark at 4 items.

*kUSART\_TxFifo5* USART tx watermark at 5 items.

*kUSART\_TxFifo6* USART tx watermark at 6 items.

*kUSART\_TxFifo7* USART tx watermark at 7 items.

#### 16.3.5.8 enum usart\_rxfifo\_watermark\_t

Enumerator

*kUSART\_RxFifo1* USART rx watermark at 1 item.

*kUSART\_RxFifo2* USART rx watermark at 2 items.

*kUSART\_RxFifo3* USART rx watermark at 3 items.

*kUSART\_RxFifo4* USART rx watermark at 4 items.

*kUSART\_RxFifo5* USART rx watermark at 5 items.

- kUSART\_RxFifo6*** USART rx watermark at 6 items.
- kUSART\_RxFifo7*** USART rx watermark at 7 items.
- kUSART\_RxFifo8*** USART rx watermark at 8 items.

### 16.3.5.9 enum \_uart\_interrupt\_enable

Enumerator

- kUSART\_TxIdleInterruptEnable*** Transmitter idle.
- kUSART\_CtsChangeInterruptEnable*** Change in the state of the CTS input.
- kUSART\_RxBreakChangeInterruptEnable*** Break condition asserted or deasserted.
- kUSART\_RxStartInterruptEnable*** Rx start bit detected.
- kUSART\_FramingErrorInterruptEnable*** Framing error detected.
- kUSART\_ParityErrorInterruptEnable*** Parity error detected.
- kUSART\_NoiseErrorInterruptEnable*** Noise error detected.
- kUSART\_AutoBaudErrorInterruptEnable*** Auto baudrate error detected.

### 16.3.5.10 enum \_uart\_flags

This provides constants for the USART status flags for use in the USART functions.

Enumerator

- kUSART\_TxError*** TEERR bit, sets if TX buffer is error.
- kUSART\_RxError*** RXERR bit, sets if RX buffer is error.
- kUSART\_TxFifoEmptyFlag*** TXEMPTY bit, sets if TX buffer is empty.
- kUSART\_TxFifoNotFullFlag*** TXNOTFULL bit, sets if TX buffer is not full.
- kUSART\_RxFifoNotEmptyFlag*** RXNOEMPTY bit, sets if RX buffer is not empty.
- kUSART\_RxFifoFullFlag*** RXFULL bit, sets if RX buffer is full.
- kUSART\_RxIdleFlag*** Receiver idle.
- kUSART\_TxIdleFlag*** Transmitter idle.
- kUSART\_CtsAssertFlag*** CTS signal high.
- kUSART\_CtsChangeFlag*** CTS signal changed interrupt status.
- kUSART\_BreakDetectFlag*** Break detected. Self cleared when rx pin goes high again.
- kUSART\_BreakDetectChangeFlag*** Break detect change interrupt flag. A change in the state of receiver break detection.
- kUSART\_RxStartFlag*** Rx start bit detected interrupt flag.
- kUSART\_FramingErrorFlag*** Framing error interrupt flag.
- kUSART\_ParityErrorFlag*** parity error interrupt flag.
- kUSART\_NoiseErrorFlag*** Noise error interrupt flag.
- kUSART\_AutobaudErrorFlag*** Auto baudrate error interrupt flag, caused by the baudrate counter timeout before the end of start bit.

### 16.3.6 Function Documentation

**16.3.6.1 uint32\_t USART\_GetInstance ( USART\_Type \* *base* )**

**16.3.6.2 status\_t USART\_Init ( USART\_Type \* *base*, const usart\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )**

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the [USART\\_GetDefaultConfig\(\)](#) function. Example below shows how to use this API to configure USART.

```
*  usart_config_t usartConfig;
*  usartConfig.baudRate_Bps = 115200U;
*  usartConfig.parityMode = kUSART_ParityDisabled;
*  usartConfig.stopBitCount = kUSART_OneStopBit;
*  USART_Init(USART1, &usartConfig, 20000000U);
*
```

Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>base</i>        | USART peripheral base address.                   |
| <i>config</i>      | Pointer to user-defined configuration structure. |
| <i>srcClock_Hz</i> | USART clock source frequency in HZ.              |

Return values

|                                         |                                                  |
|-----------------------------------------|--------------------------------------------------|
| <i>kStatus_USART_BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_InvalidArgument</i>          | USART base address is not valid                  |
| <i>kStatus_Success</i>                  | Status USART initialize succeed                  |

**16.3.6.3 void USART\_Deinit ( USART\_Type \* *base* )**

This function waits for TX complete, disables TX and RX, and disables the USART clock.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

**16.3.6.4 void USART\_GetDefaultConfig ( usart\_config\_t \* *config* )**

This function initializes the USART configuration structure to a default value. The default values are: usartConfig->baudRate\_Bps = 115200U; usartConfig->parityMode = kUSART\_ParityDisabled; usart-

```
Config->stopBitCount = kUSART_OneStopBit; usartConfig->bitCountPerChar = kUSART_8BitsPerChar; usartConfig->loopback = false; usartConfig->enableTx = false; usartConfig->enableRx = false;
```

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>config</i> | Pointer to configuration structure. |
|---------------|-------------------------------------|

#### 16.3.6.5 status\_t USART\_SetBaudRate ( USART\_Type \* *base*, uint32\_t *baudrate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the USART\_Init.

```
* USART_SetBaudRate(USART1, 115200U, 20000000U);
*
```

Parameters

|                     |                                     |
|---------------------|-------------------------------------|
| <i>base</i>         | USART peripheral base address.      |
| <i>baudrate_Bps</i> | USART baudrate to be set.           |
| <i>srcClock_Hz</i>  | USART clock source frequency in HZ. |

Return values

|                                          |                                                  |
|------------------------------------------|--------------------------------------------------|
| <i>kStatus_USART_-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                   | Set baudrate succeed.                            |
| <i>kStatus_InvalidArgument</i>           | One or more arguments are invalid.               |

#### 16.3.6.6 status\_t USART\_Enable32kMode ( USART\_Type \* *base*, uint32\_t *baudRate\_Bps*, bool *enableMode32k*, uint32\_t *srcClock\_Hz* )

Please note that in order to use a 32 kHz clock to operate USART properly, the RTC oscillator and its 32 kHz output must be manually enabled by user, by calling RTC\_Init and setting SYSCON\_RTCOSCCTRL\_EN bit to 1. And in 32kHz clocking mode the USART can only work at 9600 baudrate or at the baudrate that 9600 can evenly divide, eg: 4800, 3200.

Parameters

---

|                       |                                         |
|-----------------------|-----------------------------------------|
| <i>base</i>           | USART peripheral base address.          |
| <i>baudRate_Bps</i>   | USART baudrate to be set..              |
| <i>enable-Mode32k</i> | true is 32k mode, false is normal mode. |
| <i>srcClock_Hz</i>    | USART clock source frequency in HZ.     |

Return values

|                                         |                                                  |
|-----------------------------------------|--------------------------------------------------|
| <i>kStatus_USART_BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                  | Set baudrate succeed.                            |
| <i>kStatus_InvalidArgument</i>          | One or more arguments are invalid.               |

#### 16.3.6.7 void USART\_Enable9bitMode ( USART\_Type \* *base*, bool *enable* )

This function set the 9-bit mode for USART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | USART peripheral base address.    |
| <i>enable</i> | true to enable, false to disable. |

#### 16.3.6.8 static void USART\_SetMatchAddress ( USART\_Type \* *base*, uint8\_t *address* ) [inline], [static]

This function configures the address for USART module that works as slave in 9-bit data mode. When the address detection is enabled, the frame it receives with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

Any USART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | USART peripheral base address. |
| <i>address</i> | USART slave address.           |

#### 16.3.6.9 static void USART\_EnableMatchAddress ( USART\_Type \* *base*, bool *match* ) [inline], [static]

Parameters

|              |                                                 |
|--------------|-------------------------------------------------|
| <i>base</i>  | USART peripheral base address.                  |
| <i>match</i> | true to enable match address, false to disable. |

#### 16.3.6.10 static uint32\_t USART\_GetStatusFlags ( USART\_Type \* *base* ) [inline], [static]

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators [\\_uart\\_flags](#). To check a specific status, compare the return value with enumerators in [\\_uart\\_flags](#). For example, to check whether the TX is empty:

```
*     if (kUSART_TxFifoNotFullFlag &
*         USART_GetStatusFlags(USART1))
*     {
*         ...
*     }
```

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

Returns

USART status flags which are ORed by the enumerators in the [\\_uart\\_flags](#).

#### 16.3.6.11 static void USART\_ClearStatusFlags ( USART\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function clear supported USART status flags Flags that can be cleared or set are: kUSART\_TxError kUSART\_RxError For example:

```
*     USART_ClearStatusFlags(USART1, kUSART_TxError |
*                           kUSART_RxError)
*
```

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
| <i>mask</i> | status flags to be cleared.    |

**16.3.6.12 static void USART\_EnableInterrupts ( USART\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_uart\\_interrupt\\_enable](#). For example, to enable TX empty interrupt and RX full interrupt:

```
*     USART_EnableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
    kUSART_RxLevelInterruptEnable);
*
```

## Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | USART peripheral base address.                                                   |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_uart_interrupt_enable</a> . |

**16.3.6.13 static void USART\_DisableInterrupts ( USART\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [\\_uart\\_interrupt\\_enable](#). This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     USART_DisableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
    kUSART_RxLevelInterruptEnable);
*
```

## Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | USART peripheral base address.                                                    |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_uart_interrupt_enable</a> . |

**16.3.6.14 static uint32\_t USART\_GetEnabledInterrupts ( USART\_Type \* *base* )  
[inline], [static]**

This function returns the enabled USART interrupts.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

#### 16.3.6.15 static void USART\_EnableCTS ( USART\_Type \* *base*, bool *enable* ) [inline], [static]

This function will determine whether CTS is used for flow control.

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                            |
| <i>enable</i> | Enable CTS or not, true for enable and false for disable. |

#### 16.3.6.16 static void USART\_EnableContinuousSCLK ( USART\_Type \* *base*, bool *enable* ) [inline], [static]

By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this function, SCLK will run continuously in synchronous mode, allowing characters to be received on Un\_RxD independently from transmission on Un\_TxD).

Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                                         |
| <i>enable</i> | Enable Continuous Clock generation mode or not, true for enable and false for disable. |

#### 16.3.6.17 static void USART\_EnableAutoClearSCLK ( USART\_Type \* *base*, bool *enable* ) [inline], [static]

While enable this function, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                   |
| <i>enable</i> | Enable auto clear or not, true for enable and false for disable. |

#### 16.3.6.18 static void USART\_SetRx\_fifoWatermark ( USART\_Type \* *base*, uint8\_t *water* ) [inline], [static]

Parameters

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | USART peripheral base address. |
| <i>water</i> | Rx FIFO watermark.             |

**16.3.6.19 static void USART\_SetTxFifoWatermark ( USART\_Type \* *base*, uint8\_t *water* ) [inline], [static]**

Parameters

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | USART peripheral base address. |
| <i>water</i> | Tx FIFO watermark.             |

**16.3.6.20 static void USART\_WriteByte ( USART\_Type \* *base*, uint8\_t *data* ) [inline], [static]**

This function writes data to the txFIFO directly. The upper layer must ensure that txFIFO has space for data to write before calling this function.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
| <i>data</i> | The byte to write.             |

**16.3.6.21 static uint8\_t USART\_ReadByte ( USART\_Type \* *base* ) [inline], [static]**

This function reads data from the rxFIFO directly. The upper layer must ensure that the rxFIFO is not empty before calling this function.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

Returns

The byte read from USART data register.

**16.3.6.22 static uint8\_t USART\_GetRxFifoCount ( USART\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

Returns

rx FIFO data count.

### 16.3.6.23 static uint8\_t USART\_GetTxFifoCount ( USART\_Type \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

Returns

tx FIFO data count.

### 16.3.6.24 void USART\_SendAddress ( USART\_Type \* *base*, uint8\_t *address* )

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | USART peripheral base address. |
| <i>address</i> | USART slave address.           |

### 16.3.6.25 status\_t USART\_WriteBlocking ( USART\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | USART peripheral base address.      |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

Return values

|                                |                                         |
|--------------------------------|-----------------------------------------|
| <i>kStatus_USART_Timeout</i>   | Transmission timed out and was aborted. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                       |
| <i>kStatus_Success</i>         | Successfully wrote all data.            |

#### 16.3.6.26 status\_t USART\_ReadBlocking ( USART\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data and read data from the TX register.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                          |
| <i>data</i>   | Start address of the buffer to store the received data. |
| <i>length</i> | Size of the buffer.                                     |

Return values

|                                    |                                                 |
|------------------------------------|-------------------------------------------------|
| <i>kStatus_USART_-FramingError</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_USART_Parity-Error</i>  | Noise error happened while receiving data.      |
| <i>kStatus_USART_Noise-Error</i>   | Framing error happened while receiving data.    |
| <i>kStatus_USART_RxError</i>       | Overflow or underflow rxFIFO happened.          |
| <i>kStatus_USART_Timeout</i>       | Transmission timed out and was aborted.         |
| <i>kStatus_Success</i>             | Successfully received all data.                 |

#### 16.3.6.27 status\_t USART\_TransferCreateHandle ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the USART handle which can be used for other USART transactional APIs. Usually, for a specified USART instance, call this API once to get the initialized handle.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | USART peripheral base address.          |
| <i>handle</i>   | USART handle pointer.                   |
| <i>callback</i> | The callback function.                  |
| <i>userData</i> | The parameter of the callback function. |

#### 16.3.6.28 **status\_t USART\_TransferSendNonBlocking ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer* )**

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the IRQ handler, the USART driver calls the callback function and passes the [kStatus\\_USART\\_TxIdle](#) as status parameter.

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                   |
| <i>handle</i> | USART handle pointer.                                            |
| <i>xfer</i>   | USART transfer structure. See <a href="#">usart_transfer_t</a> . |

Return values

|                                |                                                                                    |
|--------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data transmission.                                          |
| <i>kStatus_USART_TxBusy</i>    | Previous transmission still not finished, data not all written to TX register yet. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                                                  |

#### 16.3.6.29 **void USART\_TransferStartRingBuffer ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )**

This function sets up the RX ring buffer to a specific USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the [USART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if *ringBufferSize* is 32, then only 31 bytes are used for saving data.

Parameters

|                       |                                                                                                  |
|-----------------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>           | USART peripheral base address.                                                                   |
| <i>handle</i>         | USART handle pointer.                                                                            |
| <i>ringBuffer</i>     | Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.                                                                         |

#### 16.3.6.30 void USART\_TransferStopRingBuffer ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

#### 16.3.6.31 size\_t USART\_TransferGetRxRingBufferLength ( usart\_handle\_t \* *handle* )

Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | USART handle pointer. |
|---------------|-----------------------|

Returns

Length of received data in RX ring buffer.

#### 16.3.6.32 void USART\_TransferAbortSend ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are still not sent out.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

**16.3.6.33 status\_t USART\_TransferGetSendCount( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been sent out to bus by interrupt method.

## Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |
| <i>count</i>  | Send bytes count.              |

## Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

### 16.3.6.34 status\_t USART\_TransferReceiveNonBlocking ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter [kStatus\\_USART\\_RxIdle](#). For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the *xfer*->*data* and this function returns with the parameter *receivedBytes* set to 5. For the left 5 bytes, newly arrived data is saved from the *xfer*->*data*[5]. When 5 bytes are received, the USART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the *xfer*->*data*. When all data is received, the upper layer is notified.

## Parameters

|                      |                                                                  |
|----------------------|------------------------------------------------------------------|
| <i>base</i>          | USART peripheral base address.                                   |
| <i>handle</i>        | USART handle pointer.                                            |
| <i>xfer</i>          | USART transfer structure, see <a href="#">usart_transfer_t</a> . |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.                    |

## Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully queue the transfer into transmit queue. |
| <i>kStatus_USART_RxBusy</i>    | Previous receive request is not finished.            |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                    |

### 16.3.6.35 void USART\_TransferAbortReceive ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

### 16.3.6.36 status\_t USART\_TransferGetReceiveCount ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |
| <i>count</i>  | Receive bytes count.           |

Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                               |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

### 16.3.6.37 void USART\_TransferHandleIRQ ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )

This function handles the USART transmit and receive IRQ request.

## Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

## 16.4 USART DMA Driver

### 16.4.1 Overview

#### Files

- file [fsl\\_usart\\_dma.h](#)

#### Data Structures

- struct [usart\\_dma\\_handle\\_t](#)  
*USART DMA handle.* [More...](#)

#### TypeDefs

- typedef void(\* [usart\\_dma\\_transfer\\_callback\\_t](#))(USART\_Type \*base, usart\_dma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*UART transfer callback function.*

#### Driver version

- #define [FSL\\_USART\\_DMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 6, 0))  
*USART dma driver version.*

#### DMA transactional

- [status\\_t USART\\_TransferCreateHandleDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle, [usart\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*txDmaHandle, [dma\\_handle\\_t](#) \*rxDmaHandle)  
*Initializes the USART handle which is used in transactional functions.*
- [status\\_t USART\\_TransferSendDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle, [usart\\_transfer\\_t](#) \*xfer)  
*Sends data using DMA.*
- [status\\_t USART\\_TransferReceiveDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle, [usart\\_transfer\\_t](#) \*xfer)  
*Receives data using DMA.*
- void [USART\\_TransferAbortSendDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle)  
*Aborts the sent data using DMA.*
- void [USART\\_TransferAbortReceiveDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle)  
*Aborts the received data using DMA.*
- [status\\_t USART\\_TransferGetReceiveCountDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*

- **status\_t USART\_TransferGetSendCountDMA** (USART\_Type \*base, usart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been sent.*

## 16.4.2 Data Structure Documentation

### 16.4.2.1 struct\_usart\_dma\_handle

#### Data Fields

- USART\_Type \* **base**  
*UART peripheral base address.*
- usart\_dma\_transfer\_callback\_t **callback**  
*Callback function.*
- void \* **userData**  
*UART callback function parameter.*
- size\_t **rxDataSizeAll**  
*Size of the data to receive.*
- size\_t **txDataSizeAll**  
*Size of the data to send out.*
- dma\_handle\_t \* **txDmaHandle**  
*The DMA TX channel used.*
- dma\_handle\_t \* **rxDmaHandle**  
*The DMA RX channel used.*
- volatile uint8\_t **txState**  
*TX transfer state.*
- volatile uint8\_t **rxState**  
*RX transfer state.*

#### Field Documentation

- (1) USART\_Type\* **usart\_dma\_handle\_t::base**
- (2) usart\_dma\_transfer\_callback\_t **usart\_dma\_handle\_t::callback**
- (3) void\* **usart\_dma\_handle\_t::userData**
- (4) size\_t **usart\_dma\_handle\_t::rxDataSizeAll**
- (5) size\_t **usart\_dma\_handle\_t::txDataSizeAll**
- (6) dma\_handle\_t\* **usart\_dma\_handle\_t::txDmaHandle**
- (7) dma\_handle\_t\* **usart\_dma\_handle\_t::rxDmaHandle**
- (8) volatile uint8\_t **usart\_dma\_handle\_t::txState**

## 16.4.3 Macro Definition Documentation

**16.4.3.1 #define FSL\_USART\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))**

#### 16.4.4 Typedef Documentation

**16.4.4.1 `typedef void(* usart_dma_transfer_callback_t)(USART_Type *base,  
usart_dma_handle_t *handle, status_t status, void *userData)`**

#### 16.4.5 Function Documentation

**16.4.5.1 `status_t USART_TransferCreateHandleDMA ( USART_Type * base,  
usart_dma_handle_t * handle, usart_dma_transfer_callback_t callback, void *  
userData, dma_handle_t * txDmaHandle, dma_handle_t * rxDmaHandle )`**

Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>base</i>        | USART peripheral base address.                        |
| <i>handle</i>      | Pointer to <code>usart_dma_handle_t</code> structure. |
| <i>callback</i>    | Callback function.                                    |
| <i>userData</i>    | User data.                                            |
| <i>txDmaHandle</i> | User-requested DMA handle for TX DMA transfer.        |
| <i>rxDmaHandle</i> | User-requested DMA handle for RX DMA transfer.        |

**16.4.5.2 `status_t USART_TransferSendDMA ( USART_Type * base, usart_dma_handle_t  
* handle, usart_transfer_t * xfer )`**

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                       |
| <i>handle</i> | USART handle pointer.                                                |
| <i>xfer</i>   | USART DMA transfer structure. See <a href="#">usart_transfer_t</a> . |

Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_USART_TxBusy</i>    | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

#### 16.4.5.3 **status\_t USART\_TransferReceiveDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer* )**

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                       |
| <i>handle</i> | Pointer to usart_dma_handle_t structure.                             |
| <i>xfer</i>   | USART DMA transfer structure. See <a href="#">usart_transfer_t</a> . |

Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_USART_RxBusy</i>    | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

#### 16.4.5.4 **void USART\_TransferAbortSendDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle* )**

This function aborts send data using DMA.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | USART peripheral base address           |
| <i>handle</i> | Pointer to usart_dma_handle_t structure |

#### 16.4.5.5 **void USART\_TransferAbortReceiveDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle* )**

This function aborts the received data using DMA.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | USART peripheral base address           |
| <i>handle</i> | Pointer to usart_dma_handle_t structure |

#### 16.4.5.6 status\_t USART\_TransferGetReceiveCountDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |
| <i>count</i>  | Receive bytes count.           |

Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                               |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

#### 16.4.5.7 status\_t USART\_TransferGetSendCountDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been sent.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |
| <i>count</i>  | Sent bytes count.              |

Return values

|                                      |                                               |
|--------------------------------------|-----------------------------------------------|
| <i>kStatus_NoTransferIn-Progress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>       | Parameter is invalid.                         |
| <i>kStatus_Success</i>               | Get successfully through the parameter count; |

## 16.5 USART FreeRTOS Driver

### 16.5.1 Overview

#### Files

- file `fsl_usart_freertos.h`

#### Data Structures

- struct `rtos_usart_config`  
*FLEX USART configuration structure.* [More...](#)
- struct `usart_rtos_handle_t`  
*FLEX USART FreeRTOS handle.* [More...](#)

#### Driver version

- #define `FSL_USART_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 0)`)  
*USART FreeRTOS driver version.*

#### USART RTOS Operation

- int `USART_RTOS_Init` (`usart_rtos_handle_t` \*`handle`, `usart_handle_t` \*`t_handle`, const struct `rtos_usart_config` \*`cfg`)  
*Initializes a USART instance for operation in RTOS.*
- int `USART_RTOS_Deinit` (`usart_rtos_handle_t` \*`handle`)  
*Deinitializes a USART instance for operation.*

#### USART transactional Operation

- int `USART_RTOS_Send` (`usart_rtos_handle_t` \*`handle`, `uint8_t` \*`buffer`, `uint32_t` `length`)  
*Sends data in the background.*
- int `USART_RTOS_Receive` (`usart_rtos_handle_t` \*`handle`, `uint8_t` \*`buffer`, `uint32_t` `length`, `size_t` \*`received`)  
*Receives data.*

### 16.5.2 Data Structure Documentation

#### 16.5.2.1 struct `rtos_usart_config`

##### Data Fields

- `USART_Type` \* `base`

- **USART base address.**
- **uint32\_t sreclk**  
USART source clock in Hz.
- **uint32\_t baudrate**  
Desired communication speed.
- **uart\_parity\_mode\_t parity**  
Parity setting.
- **uart\_stop\_bit\_count\_t stopbits**  
Number of stop bits to use.
- **uint8\_t \* buffer**  
Buffer for background reception.
- **uint32\_t buffer\_size**  
Size of buffer for background reception.

### 16.5.2.2 struct usart\_rtos\_handle\_t

#### Data Fields

- **USART\_Type \* base**  
USART base address.
- **uart\_transfer\_t txTransfer**  
TX transfer structure.
- **uart\_transfer\_t rxTransfer**  
RX transfer structure.
- **SemaphoreHandle\_t rxSemaphore**  
RX semaphore for resource sharing.
- **SemaphoreHandle\_t txSemaphore**  
TX semaphore for resource sharing.
- **EventGroupHandle\_t rxEvent**  
RX completion event.
- **EventGroupHandle\_t txEvent**  
TX completion event.
- **void \* t\_state**  
Transactional state of the underlying driver.

### 16.5.3 Macro Definition Documentation

#### 16.5.3.1 #define FSL\_USART\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))

### 16.5.4 Function Documentation

#### 16.5.4.1 int USART\_RTOs\_Init ( usart\_rtos\_handle\_t \* handle, usart\_handle\_t \* t\_handle, const struct rtos\_usart\_config \* cfg )

Parameters

|                 |                                                                                     |
|-----------------|-------------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS USART handle, the pointer to allocated space for RTOS context.             |
| <i>t_handle</i> | The pointer to allocated space where to store transactional layer internal state.   |
| <i>cfg</i>      | The pointer to the parameters required to configure the USART after initialization. |

Returns

0 succeed, others fail.

#### 16.5.4.2 int USART\_RTOs\_Deinit ( usart\_rtos\_handle\_t \* *handle* )

This function deinitializes the USART module, sets all register values to reset value, and releases the resources.

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS USART handle. |
|---------------|------------------------|

#### 16.5.4.3 int USART\_RTOs\_Send ( usart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS USART handle.         |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 16.5.4.4 int USART\_RTOs\_Receive ( usart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from USART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

## Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS USART handle.                                                           |
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.                                                  |
| <i>received</i> | The pointer to a variable of size_t where the number of received data is filled. |

## 16.6 USART CMSIS Driver

This section describes the programming interface of the USART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The USART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

### 16.6.1 USART Send Methods

#### 16.6.1.1 USART Send/receive using an interrupt method

```
/* USART callback */
void USART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txOnGoing = false;
    }
}
Driver_USART0.Initialize(USART_Callback);
Driver_USART0.PowerControl(ARM_POWER_FULL);
/* Send g_tipString out. */
txOnGoing = true;
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

#### 16.6.1.2 USART Send/Receive using the DMA method

```
/* USART callback */
void USART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txOnGoing = false;
    }
}
Driver_USART0.Initialize(USART_Callback);
DMA_Init(DMA0);
Driver_USART0.PowerControl(ARM_POWER_FULL);

/* Send g_tipString out. */
txOnGoing = true;
```

```
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

# Chapter 17

## GINT: Group GPIO Input Interrupt Driver

### 17.1 Overview

The MCUXpresso SDK provides a driver for the Group GPIO Input Interrupt (GINT).

It can configure one or more pins to generate a group interrupt when the pin conditions are met. The pins do not have to be configured as GPIO pins.

### 17.2 Group GPIO Input Interrupt Driver operation

[GINT\\_SetCtrl\(\)](#) and [GINT\\_ConfigPins\(\)](#) functions configure the pins.

[GINT\\_EnableCallback\(\)](#) function enables the callback functionality. Callback function is called when the pin conditions are met.

### 17.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/gint

#### Files

- file [fsl\\_gint.h](#)

#### Typedefs

- `typedef void(* gint_cb_t )(void)`  
*GINT Callback function.*

#### Enumerations

- `enum gint_comb_t {`  
  `kGINT_CombineOr = 0U,`  
  `kGINT_CombineAnd = 1U }`  
*GINT combine inputs type.*
- `enum gint_trig_t {`  
  `kGINT_TrigEdge = 0U,`  
  `kGINT_TrigLevel = 1U }`  
*GINT trigger type.*

#### Functions

- `void GINT_Init (GINT_Type *base)`  
*Initialize GINT peripheral.*
- `void GINT_SetCtrl (GINT_Type *base, gint_comb_t comb, gint_trig_t trig, gint_cb_t callback)`

- `GINT_Setup` *Setup GINT peripheral control parameters.*
- void `GINT_SetCtrl` (GINT\_Type \*base, `gint_comb_t` \*comb, `gint_trig_t` \*trig, `gint_cb_t` \*callback)  
*Get GINT peripheral control parameters.*
- void `GINT_ConfigPins` (GINT\_Type \*base, `gint_port_t` port, `uint32_t` polarityMask, `uint32_t` enableMask)  
*Configure GINT peripheral pins.*
- void `GINT_GetConfigPins` (GINT\_Type \*base, `gint_port_t` port, `uint32_t` \*polarityMask, `uint32_t` \*enableMask)  
*Get GINT peripheral pin configuration.*
- void `GINT_EnableCallback` (GINT\_Type \*base)  
*Enable callback.*
- void `GINT_DisableCallback` (GINT\_Type \*base)  
*Disable callback.*
- static void `GINT_ClrStatus` (GINT\_Type \*base)  
*Clear GINT status.*
- static `uint32_t` `GINT_GetStatus` (GINT\_Type \*base)  
*Get GINT status.*
- void `GINT_Deinit` (GINT\_Type \*base)  
*Deinitialize GINT peripheral.*

## Driver version

- #define `FSL_GINT_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 0)`)  
*Driver version.*

## 17.4 Macro Definition Documentation

### 17.4.1 #define FSL\_GINT\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

## 17.5 Typedef Documentation

### 17.5.1 `typedef void(* gint_cb_t)(void)`

## 17.6 Enumeration Type Documentation

### 17.6.1 `enum gint_comb_t`

Enumerator

*kGINT\_CombineOr* A grouped interrupt is generated when any one of the enabled inputs is active.

*kGINT\_CombineAnd* A grouped interrupt is generated when all enabled inputs are active.

### 17.6.2 `enum gint_trig_t`

Enumerator

*kGINT\_TrigEdge* Edge triggered based on polarity.

*kGINT\_TrigLevel* Level triggered based on polarity.

## 17.7 Function Documentation

### 17.7.1 void GINT\_Init ( **GINT\_Type** \* *base* )

This function initializes the GINT peripheral and enables the clock.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

### 17.7.2 void GINT\_SetCtrl ( **GINT\_Type** \* *base*, **gint\_comb\_t** *comb*, **gint\_trig\_t** *trig*, **gint\_cb\_t** *callback* )

This function sets the control parameters of GINT peripheral.

Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address of the GINT peripheral.                                                 |
| <i>comb</i>     | Controls if the enabled inputs are logically ORed or ANDed for interrupt generation. |
| <i>trig</i>     | Controls if the enabled inputs are level or edge sensitive based on polarity.        |
| <i>callback</i> | This function is called when configured group interrupt is generated.                |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

### 17.7.3 void GINT\_GetCtrl ( **GINT\_Type** \* *base*, **gint\_comb\_t** \* *comb*, **gint\_trig\_t** \* *trig*, **gint\_cb\_t** \* *callback* )

This function returns the control parameters of GINT peripheral.

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>base</i>     | Base address of the GINT peripheral.  |
| <i>comb</i>     | Pointer to store combine input value. |
| <i>trig</i>     | Pointer to store trigger value.       |
| <i>callback</i> | Pointer to store callback function.   |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

#### 17.7.4 void GINT\_ConfigPins ( GINT\_Type \* *base*, gint\_port\_t *port*, uint32\_t *polarityMask*, uint32\_t *enableMask* )

This function enables and controls the polarity of enabled pin(s) of a given port.

Parameters

|                     |                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | Base address of the GINT peripheral.                                                                                            |
| <i>port</i>         | Port number.                                                                                                                    |
| <i>polarityMask</i> | Each bit position selects the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH. |
| <i>enableMask</i>   | Each bit position selects if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.          |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

#### 17.7.5 void GINT\_GetConfigPins ( GINT\_Type \* *base*, gint\_port\_t *port*, uint32\_t \* *polarityMask*, uint32\_t \* *enableMask* )

This function returns the pin configuration of a given port.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

|                     |                                                                                                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>port</i>         | Port number.                                                                                                                                                          |
| <i>polarityMask</i> | Pointer to store the polarity mask. Each bit position indicates the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH. |
| <i>enableMask</i>   | Pointer to store the enable mask. Each bit position indicates if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.            |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 17.7.6 void GINT\_EnableCallback ( GINT\_Type \* *base* )

This function enables the interrupt for the selected GINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 17.7.7 void GINT\_DisableCallback ( GINT\_Type \* *base* )

This function disables the interrupt for the selected GINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | Base address of the peripheral. |
|-------------|---------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 17.7.8 static void GINT\_ClrStatus ( GINT\_Type \* *base* ) [inline], [static]

This function clears the GINT status bit.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

### 17.7.9 static uint32\_t GINT\_GetStatus ( GINT\_Type \* *base* ) [inline], [static]

This function returns the GINT status.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

Return values

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>status</i> | = 0 No group interrupt request. = 1 Group interrupt request active. |
|---------------|---------------------------------------------------------------------|

### 17.7.10 void GINT\_Deinit ( GINT\_Type \* *base* )

This function disables the GINT clock.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

# Chapter 18

## Hashcrypt: The Cryptographic Accelerator

### 18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Hashcrypt peripheral. The Hashcrypt peripheral provides one or more engines to perform specific symmetric crypto algorithms, including hashing and en/decryption. The cryptographic acceleration is normally used in conjunction with pure-hardware blocks for hashing and symmetric cryptography, thereby providing performance and energy efficiency for a range of cryptographic uses.

Blocking synchronous APIs are provided for selected cryptographic algorithms using Hashcrypt hardware. The driver interface intends to be easily integrated with generic software crypto libraries such as mbed-TLS. The Hashcrypt operations are complete (and results are made available for further usage) when a function returns. When called, these functions do not return until an Hashcrypt operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and also for plaintext or ciphertext data movements. These functions provide typical interface to upper layer or application software. There is one non-blocking function provided for the purpose of background hashing. [HASHCRYPT\\_SHA\\_UpdateNonBlocking\(\)](#) starts hashing of an input message while the CPU can continue executing.

### 18.2 Hashcrypt Driver Initialization and deinitialization

Hashcrypt Driver is initialized by calling the [HASHCRYPT\\_Init\(\)](#) function, it enables clock and disables reset for Hashcrypt peripheral. Hashcrypt Driver is deinitialized by calling the [HASHCRYPT\\_Deinit\(\)](#) function, it disables clock and enables reset.

### 18.3 Comments about API usage in RTOS

Hashcrypt operations provided by this driver are not re-entrant. Thus, application software shall ensure the Hashcrypt module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

### 18.4 Comments about API usage in interrupt handler

APIs can be used from interrupt handler although execution time shall be considered (interrupt latency increases considerably).

### 18.5 Hashcrypt Driver Examples

#### 18.5.1 Simple examples

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/hashcrypt/

## Modules

- Hashcrypt AES
- Hashcrypt Background HASH
- Hashcrypt HASH

## 18.6 Hashcrypt AES

### 18.6.1 Overview

#### Data Structures

- struct `hashcrypt_handle_t`  
*Specify HASHCRYPT's key resource.* [More...](#)

#### Macros

- `#define HASHCRYPT_AES_BLOCK_SIZE 16U`  
*AES block size in bytes.*

#### Enumerations

- enum `hashcrypt_aes_mode_t` {
   
`kHASHCRYPT_AesEcb` = 0U,  
`kHASHCRYPT_AesCbc` = 1U,  
`kHASHCRYPT_AesCtr` = 2U }
   
*AES mode.*
- enum `hashcrypt_aes_keysize_t` {
   
`kHASHCRYPT_Aes128` = 0U,  
`kHASHCRYPT_Aes192` = 1U,  
`kHASHCRYPT_Aes256` = 2U,  
`kHASHCRYPT_InvalidKey` = 3U }
   
*Size of AES key.*
- enum `hashcrypt_key_t` {
   
`kHASHCRYPT_UserKey` = 0xc3c3U,  
`kHASHCRYPT_SecretKey` = 0x3c3cU }
   
*HASHCRYPT key source selection.*

#### Functions

- `status_t HASHCRYPT_AES_SetKey (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *key, size_t keySize)`  
*Set AES key to hashcrypt\_handle\_t struct and optionally to HASHCRYPT.*
- `status_t HASHCRYPT_AES_EncryptEcb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)`  
*Encrypts AES on one or multiple 128-bit block(s).*
- `status_t HASHCRYPT_AES_DecryptEcb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)`  
*Decrypts AES on one or multiple 128-bit block(s).*
- `status_t HASHCRYPT_AES_EncryptCbc (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])`

- *Encrypts AES using CBC block mode.*
- `status_t HASHCRYPT_AES_DecryptCbc` (`HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16]`)  
*Decrypts AES using CBC block mode.*
- `status_t HASHCRYPT_AES_CryptCtr` (`HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *input, uint8_t *output, size_t size, uint8_t counter[HASHCRYPT_AES_BLOCK_SIZE], uint8_t counterlast[HASHCRYPT_AES_BLOCK_SIZE], size_t *szLeft)`  
*Encrypts or decrypts AES using CTR block mode.*
- `status_t HASHCRYPT_AES_CryptOfb` (`HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *input, uint8_t *output, size_t size, const uint8_t iv[HASHCRYPT_AES_BLOCK_SIZE])`  
*Encrypts or decrypts AES using OFB block mode.*
- `status_t HASHCRYPT_AES_EncryptCfb` (`HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16]`)  
*Encrypts AES using CFB block mode.*
- `status_t HASHCRYPT_AES_DecryptCfb` (`HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16]`)  
*Decrypts AES using CFB block mode.*

## 18.6.2 Data Structure Documentation

### 18.6.2.1 struct \_hashcrypt\_handle

#### Data Fields

- `uint32_t keyWord [8]`  
*Copy of user key (set by `HASHCRYPT_AES_SetKey()`).*
- `hashcrypt_key_t keyType`  
*For operations with key (such as AES encryption/decryption), specify key type.*

#### Field Documentation

- (1) `uint32_t hashcrypt_handle_t::keyWord[8]`
- (2) `hashcrypt_key_t hashcrypt_handle_t::keyType`

## 18.6.3 Enumeration Type Documentation

### 18.6.3.1 enum hashcrypt\_aes\_mode\_t

Enumerator

- kHASHCRYPT\_AesEcb*** AES ECB mode.
- kHASHCRYPT\_AesCbc*** AES CBC mode.
- kHASHCRYPT\_AesCtr*** AES CTR mode.

### 18.6.3.2 enum hashcrypt\_aes\_keysize\_t

Enumerator

*kHASHCRYPT\_Aes128* AES 128 bit key.  
*kHASHCRYPT\_Aes192* AES 192 bit key.  
*kHASHCRYPT\_Aes256* AES 256 bit key.  
*kHASHCRYPT\_InvalidKey* AES invalid key.

### 18.6.3.3 enum hashcrypt\_key\_t

Enumerator

*kHASHCRYPT\_UserKey* HASHCRYPT user key.  
*kHASHCRYPT\_SecretKey* HASHCRYPT secret key (dedicated hw bus from PUF)

## 18.6.4 Function Documentation

### 18.6.4.1 status\_t HASHCRYPT\_AES\_SetKey ( HASHCRYPT\_Type \* *base*, hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *key*, size\_t *keySize* )

Sets the AES key for encryption/decryption with the hashcrypt\_handle\_t structure. The hashcrypt\_handle\_t input argument specifies key source.

Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>base</i>    | HASHCRYPT peripheral base address.               |
| <i>handle</i>  | Handle used for the request.                     |
| <i>key</i>     | 0-mod-4 aligned pointer to AES key.              |
| <i>keySize</i> | AES key size in bytes. Shall equal 16, 24 or 32. |

Returns

status from set key operation

### 18.6.4.2 status\_t HASHCRYPT\_AES\_EncryptEcb ( HASHCRYPT\_Type \* *base*, hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, size\_t *size* )

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | HASHCRYPT peripheral base address                                     |
|     | <i>handle</i>     | Handle used for this request.                                         |
|     | <i>plaintext</i>  | Input plain text to encrypt                                           |
| out | <i>ciphertext</i> | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |

Returns

Status from encrypt operation

**18.6.4.3 status\_t HASHCRYPT\_AES\_DecryptEcb ( HASHCRYPT\_Type \* *base*, hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, size\_t *size* )**

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | HASHCRYPT peripheral base address                                     |
|     | <i>handle</i>     | Handle used for this request.                                         |
|     | <i>ciphertext</i> | Input plain text to encrypt                                           |
| out | <i>plaintext</i>  | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |

Returns

Status from decrypt operation

**18.6.4.4 status\_t HASHCRYPT\_AES\_EncryptCbc ( HASHCRYPT\_Type \* *base*, hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, size\_t *size*, const uint8\_t *iv[16]* )**

Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | HASHCRYPT peripheral base address                                     |
|     | <i>handle</i>     | Handle used for this request.                                         |
|     | <i>plaintext</i>  | Input plain text to encrypt                                           |
| out | <i>ciphertext</i> | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input initial vector to combine with the first input block.           |

Returns

Status from encrypt operation

#### 18.6.4.5 status\_t HASHCRYPT\_AES\_DecryptCbc ( HASHCRYPT\_Type \* *base*, hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, size\_t *size*, const uint8\_t *iv[16]* )

Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | HASHCRYPT peripheral base address                                     |
|     | <i>handle</i>     | Handle used for this request.                                         |
|     | <i>ciphertext</i> | Input cipher text to decrypt                                          |
| out | <i>plaintext</i>  | Output plain text                                                     |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input initial vector to combine with the first input block.           |

Returns

Status from decrypt operation

#### 18.6.4.6 status\_t HASHCRYPT\_AES\_CryptCtr ( HASHCRYPT\_Type \* *base*, hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *input*, uint8\_t \* *output*, size\_t *size*, uint8\_t *counter[HASHCRYPT\_AES\_BLOCK\_SIZE]*, uint8\_t *counterlast[HASHCRYPT\_AES\_BLOCK\_SIZE]*, size\_t \* *szLeft* )

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

## Parameters

|         |                    |                                                                                                                                |
|---------|--------------------|--------------------------------------------------------------------------------------------------------------------------------|
|         | <i>base</i>        | HASHCRYPT peripheral base address                                                                                              |
|         | <i>handle</i>      | Handle used for this request.                                                                                                  |
|         | <i>input</i>       | Input data for CTR block mode                                                                                                  |
| out     | <i>output</i>      | Output data for CTR block mode                                                                                                 |
|         | <i>size</i>        | Size of input and output data in bytes                                                                                         |
| in, out | <i>counter</i>     | Input counter (updates on return)                                                                                              |
| out     | <i>counterlast</i> | Output cipher of last counter, for chained CTR calls (statefull encryption). NULL can be passed if chained calls are not used. |
| out     | <i>szLeft</i>      | Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.                  |

## Returns

Status from encrypt operation

**18.6.4.7 status\_t HASHCRYPT\_AES\_CryptOfb ( HASHCRYPT\_Type \* *base*, hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *input*, uint8\_t \* *output*, size\_t *size*, const uint8\_t *iv*[HASHCRYPT\_AES\_BLOCK\_SIZE] )**

Encrypts or decrypts AES using OFB block mode. AES OFB mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

## Parameters

|     |               |                                                             |
|-----|---------------|-------------------------------------------------------------|
|     | <i>base</i>   | HASHCRYPT peripheral base address                           |
|     | <i>handle</i> | Handle used for this request.                               |
|     | <i>input</i>  | Input data for OFB block mode                               |
| out | <i>output</i> | Output data for OFB block mode                              |
|     | <i>size</i>   | Size of input and output data in bytes                      |
|     | <i>iv</i>     | Input initial vector to combine with the first input block. |

## Returns

Status from encrypt operation

18.6.4.8 status\_t HASHCRYPT\_AES\_EncryptCfb ( HASHCRYPT\_Type \* *base*,  
hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*,  
size\_t *size*, const uint8\_t *iv[16]* )

Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | HASHCRYPT peripheral base address                                     |
|     | <i>handle</i>     | Handle used for this request.                                         |
|     | <i>plaintext</i>  | Input plain text to encrypt                                           |
| out | <i>ciphertext</i> | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input initial vector to combine with the first input block.           |

Returns

Status from encrypt operation

#### 18.6.4.9 status\_t HASHCRYPT\_AES\_DecryptCfb ( HASHCRYPT\_Type \* *base*, hashcrypt\_handle\_t \* *handle*, const uint8\_t \* *ciphertext*, uint8\_t \* *plaintext*, size\_t *size*, const uint8\_t *iv[16]* )

Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | HASHCRYPT peripheral base address                                     |
|     | <i>handle</i>     | Handle used for this request.                                         |
|     | <i>ciphertext</i> | Input cipher text to decrypt                                          |
| out | <i>plaintext</i>  | Output plaintext text                                                 |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input initial vector to combine with the first input block.           |

Returns

Status from encrypt operation

## 18.7 Hashcrypt HASH

### 18.7.1 Overview

#### Data Structures

- struct `hashcrypt_hash_ctx_t`  
*Storage type used to save hash context. [More...](#)*

#### Macros

- `#define HASHCRYPT_HASH_CTX_SIZE 22`  
*HASHCRYPT HASH Context size.*

#### Typedefs

- `typedef void(* hashcrypt_callback_t )(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, status_t status, void *userData)`  
*HASHCRYPT background hash callback function.*

#### Functions

- `status_t HASHCRYPT_SHA (HASHCRYPT_Type *base, hashcrypt_algo_t algo, const uint8_t *input, size_t inputSize, uint8_t *output, size_t *outputSize)`  
*Create HASH on given data.*
- `status_t HASHCRYPT_SHA_Init (HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, hashcrypt_algo_t algo)`  
*Initialize HASH context.*
- `status_t HASHCRYPT_SHA_Update (HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, const uint8_t *input, size_t inputSize)`  
*Add data to current HASH.*
- `status_t HASHCRYPT_SHA_Finish (HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, uint8_t *output, size_t *outputSize)`  
*Finalize hashing.*

### 18.7.2 Data Structure Documentation

#### 18.7.2.1 struct hashcrypt\_hash\_ctx\_t

##### Data Fields

- `uint32_t x [HASHCRYPT_HASH_CTX_SIZE]`  
*storage*

### 18.7.3 Macro Definition Documentation

#### 18.7.3.1 #define HASHCRYPT\_HASH\_CTX\_SIZE 22

### 18.7.4 Typedef Documentation

#### 18.7.4.1 `typedef void(* hashcrypt_callback_t)(HASHCRYPT_Type *base, hashcrypt_hash_ctx_t *ctx, status_t status, void *userData)`

### 18.7.5 Function Documentation

#### 18.7.5.1 `status_t HASHCRYPT_SHA ( HASHCRYPT_Type * base, hashcrypt_algo_t algo, const uint8_t * input, size_t inputSize, uint8_t * output, size_t * outputSize )`

Perform the full SHA in one function call. The function is blocking.

Parameters

|     |                   |                                                               |
|-----|-------------------|---------------------------------------------------------------|
|     | <i>base</i>       | HASHCRYPT peripheral base address                             |
|     | <i>algo</i>       | Underlaying algorithm to use for hash computation.            |
|     | <i>input</i>      | Input data                                                    |
|     | <i>inputSize</i>  | Size of input data in bytes                                   |
| out | <i>output</i>     | Output hash data                                              |
| out | <i>outputSize</i> | Output parameter storing the size of the output hash in bytes |

Returns

Status of the one call hash operation.

#### 18.7.5.2 `status_t HASHCRYPT_SHA_Init ( HASHCRYPT_Type * base, hashcrypt_hash_ctx_t * ctx, hashcrypt_algo_t algo )`

This function initializes the HASH.

Parameters

|     |             |                                                    |
|-----|-------------|----------------------------------------------------|
|     | <i>base</i> | HASHCRYPT peripheral base address                  |
| out | <i>ctx</i>  | Output hash context                                |
|     | <i>algo</i> | Underlaying algorithm to use for hash computation. |

Returns

Status of initialization

#### 18.7.5.3 status\_t HASHCRYPT\_SHA\_Update ( **HASHCRYPT\_Type \* base,** **hashcrypt\_hash\_ctx\_t \* ctx, const uint8\_t \* input, size\_t inputSize** )

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus\_Success, the running hash has been updated (HASHCRYPT has processed the input data), so the memory at *input* pointer can be released back to system. The HASHCRYPT context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

|         |                  |                                   |
|---------|------------------|-----------------------------------|
|         | <i>base</i>      | HASHCRYPT peripheral base address |
| in, out | <i>ctx</i>       | HASH context                      |
|         | <i>input</i>     | Input data                        |
|         | <i>inputSize</i> | Size of input data in bytes       |

Returns

Status of the hash update operation

#### 18.7.5.4 status\_t HASHCRYPT\_SHA\_Finish ( **HASHCRYPT\_Type \* base,** **hashcrypt\_hash\_ctx\_t \* ctx, uint8\_t \* output, size\_t \* outputSize** )

Outputs the final hash (computed by HASHCRYPT\_HASH\_Update()) and erases the context.

Parameters

---

|        |                   |                                                                                                                                                                            |
|--------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | <i>base</i>       | HASHCRYPT peripheral base address                                                                                                                                          |
| in,out | <i>ctx</i>        | Input hash context                                                                                                                                                         |
| out    | <i>output</i>     | Output hash data                                                                                                                                                           |
| in,out | <i>outputSize</i> | Optional parameter (can be passed as NULL). On function entry, it specifies the size of output[] buffer. On function return, it stores the number of updated output bytes. |

Returns

Status of the hash finish operation

## 18.8 Hashcrypt Background HASH

### 18.8.1 Overview

#### Functions

- void [HASHCRYPT\\_SHA\\_SetCallback](#) (HASHCRYPT\_Type \*base, hashcrypt\_hash\_ctx\_t \*ctx, hashcrypt\_callback\_t callback, void \*userData)
   
*Initializes the HASHCRYPT handle for background hashing.*
- status\_t [HASHCRYPT\\_SHA\\_UpdateNonBlocking](#) (HASHCRYPT\_Type \*base, hashcrypt\_hash\_ctx\_t \*ctx, const uint8\_t \*input, size\_t inputSize)
   
*Create running hash on given data.*

### 18.8.2 Function Documentation

#### 18.8.2.1 void HASHCRYPT\_SHA\_SetCallback ( HASHCRYPT\_Type \* *base*, hashcrypt\_hash\_ctx\_t \* *ctx*, hashcrypt\_callback\_t *callback*, void \* *userData* )

This function initializes the hash context for background hashing (Non-blocking) APIs. This is less typical interface to hash function, but can be used for parallel processing, when main CPU has something else to do. Example is digital signature RSASSA-PKCS1-V1\_5-VERIFY((n,e),M,S) algorithm, where background hashing of M can be started, then CPU can compute  $S^e \bmod n$  (in parallel with background hashing) and once the digest becomes available, CPU can proceed to comparison of EM with EM'.

##### Parameters

|     |                 |                                                                                                  |
|-----|-----------------|--------------------------------------------------------------------------------------------------|
|     | <i>base</i>     | HASHCRYPT peripheral base address.                                                               |
| out | <i>ctx</i>      | Hash context.                                                                                    |
|     | <i>callback</i> | Callback function.                                                                               |
|     | <i>userData</i> | User data (to be passed as an argument to callback function, once callback is invoked from isr). |

#### 18.8.2.2 status\_t HASHCRYPT\_SHA\_UpdateNonBlocking ( HASHCRYPT\_Type \* *base*, hashcrypt\_hash\_ctx\_t \* *ctx*, const uint8\_t \* *input*, size\_t *inputSize* )

Configures the HASHCRYPT to compute new running hash as AHB master and returns immediately. HASHCRYPT AHB Master mode supports only aligned *input* address and can be called only once per continuous block of data. Every call to this function must be preceded with [HASHCRYPT\\_SHA\\_Init\(\)](#) and finished with [HASHCRYPT\\_SHA\\_Finish\(\)](#). Once callback function is invoked by HASHCRYPT isr, it should set a flag for the main application to finalize the hashing (padding) and to read out the final digest by calling [HASHCRYPT\\_SHA\\_Finish\(\)](#).

## Parameters

|                  |                                                                                                         |
|------------------|---------------------------------------------------------------------------------------------------------|
| <i>base</i>      | HASHCRYPT peripheral base address                                                                       |
| <i>ctx</i>       | Specifies callback. Last incomplete 512-bit block of the input is copied into clear buffer for padding. |
| <i>input</i>     | 32-bit word aligned pointer to Input data.                                                              |
| <i>inputSize</i> | Size of input data in bytes (must be word aligned)                                                      |

## Returns

Status of the hash update operation.

# Chapter 19

## IAP: In Application Programming Driver

### 19.1 Overview

The MCUXpresso SDK provides a driver for the In Application Programming (IAP).

It provides a set of functions to call the on-chip in application programming interface. User code executing from on-chip RAM can call these function to read information like part id, read and write flash, read and write ffr.

### 19.2 In Application Programming operation

`FLASH_Init()` Initializes the global flash properties structure members

`FLASH_Erase()` Erases the flash sectors encompassed by parameters passed into function

`FLASH_Program()` Programs flash with data at locations passed in through parameters

`FLASH_VerifyErase()` Verifies an erasure of the desired flash area has been erased

`FLASH_VerifyProgram()` Verifies programming of the desired flash area has been programmed

`FLASHGetProperty()` Returns the desired flash property.

`FFR_Init()` Generic APIs for FFR

`FFR_Deinit()` Generic APIs for FFR

`FFR_CustomerPagesInit()` APIs to access CFPA pages

`FFR_InfieldPageWrite()` APIs to access CFPA pages

`FFR_GetCustomerInfieldData()` APIs to access CMPA pages

`FFR_GetCustomerData()` Read data stored in 'Customer Factory CFG Page'

`FFR_KeystoreWrite()` Read data stored in 'Customer Factory CFG Page'

`FFR_KeystoreGetAC()` Read data stored in 'Customer Factory CFG Page'

`FFR_KeystoreGetKC()` Read data stored in 'Customer Factory CFG Page'

`FFR_GetUUID()` Read data stored in 'NXP Manufacuring Programmed CFG Page'

`FFR_GetManufactureData()` Read data stored in 'NXP Manufacuring Programmed CFG Page'

`kb_init()` Initialize ROM API for a given operation

`kb_deinit()` Cleans up the ROM API context

`kb_execute()` Perform the operation configured during init

`skboot_authenticate()` Authenticate entry function with ARENA allocator init

`HASH_IRQHandler()` Interface for image authentication API

`kb_init()` Initialize ROM API for a given operation  
`kb_deinit()` Cleans up the ROM API context  
`kb_execute()` Perform the operation configured during init  
`skboot_authenticate()` Authenticate entry function with ARENA allocator init  
`HASH_IRQHandler()` Interface for image authentication API

## 19.3 Typical use case

### 19.3.1 IAP Basic Operations

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/iap1

## Modules

- `IAP_FFR` Driver
- `IAP_KBP` Driver

## Files

- file `fsl_iap.h`

## Data Structures

- struct `flash_ecc_log_t`  
*Flash ECC log info.* [More...](#)
- struct `flash_mode_config_t`  
*Flash controller parameter config.* [More...](#)
- struct `flash_ffr_config_t`  
*Flash controller parameter config.* [More...](#)
- struct `flash_config_t`  
*Flash driver state information.* [More...](#)

## Enumerations

- enum `flash_property_tag_t` {
   
`kFLASH_PropertyPflashSectorSize` = 0x00U,  
`kFLASH_PropertyPflashTotalSize` = 0x01U,  
`kFLASH_PropertyPflashBlockSize` = 0x02U,  
`kFLASH_PropertyPflashBlockCount` = 0x03U,  
`kFLASH_PropertyPflashBlockBaseAddr` = 0x04U,  
`kFLASH_PropertyPflashPageSize` = 0x30U,  
`kFLASH_PropertyPflashSystemFreq` = 0x31U,  
`kFLASH_PropertyFfrSectorSize` = 0x40U,  
`kFLASH_PropertyFfrTotalSize` = 0x41U,  
`kFLASH_PropertyFfrBlockBaseAddr` = 0x42U,  
`kFLASH_PropertyFfrPageSize` = 0x43U }

- Enumeration for various flash properties.
- enum `_flash_max_erase_page_value` { `kFLASH_MaxPagesToErase` = 100U }
- Enumeration for flash max pages to erase.
- enum `_flash_alignment_property` {
   
`kFLASH_AlignmentUnitVerifyErase` = 4,
   
`kFLASH_AlignmentUnitProgram` = 512,
   
`kFLASH_AlignmentUnitSingleWordRead` = 16 }
- Enumeration for flash alignment property.
- enum `_flash_read_ecc_option` { , `kFLASH_ReadWithEccOff` = 1 }
- Enumeration for flash read ecc option.
- enum `_flash_read_margin_option` {
   
`kFLASH_ReadMarginNormal` = 0,
   
`kFLASH_ReadMarginVsProgram` = 1,
   
`kFLASH_ReadMarginVsErase` = 2,
   
`kFLASH_ReadMarginIllegalBitCombination` = 3 }
- Enumeration for flash read margin option.
- enum `_flash_read_dmacc_option` {
   
`kFLASH_ReadDmaccDisabled` = 0,
   
`kFLASH_ReadDmaccEnabled` = 1 }
- Enumeration for flash read dmacc option.
- enum `_flash_ramp_control_option` {
   
`kFLASH_RampControlDivisionFactorReserved` = 0,
   
`kFLASH_RampControlDivisionFactor256` = 1,
   
`kFLASH_RampControlDivisionFactor128` = 2,
   
`kFLASH_RampControlDivisionFactor64` = 3 }
- Enumeration for flash ramp control option.

## Functions

- `status_t FLASH_Read (flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t lengthInBytes)`  
Reads flash at locations passed in through parameters.

## Flash version

- enum `_flash_driver_version_constants` {
   
`kFLASH_DriverVersionName` = 'F',
   
`kFLASH_DriverVersionMajor` = 2,
   
`kFLASH_DriverVersionMinor` = 1,
   
`kFLASH_DriverVersionBugfix` = 3 }
   
Flash driver version for ROM.
- `#define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))`  
Constructs the version number for drivers.
- `#define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))`  
Flash driver version for SDK.

## Flash configuration

- `#define FSL_FEATURE_FLASH_IP_IS_C040HD_ATFC (1)`

- Flash IP Type.*
- #define **FSL\_FEATURE\_FLASH\_IP\_IS\_C040HD\_FC** (0)

## Flash status

- enum **\_flash\_status** {
   
 kStatus\_FLASH\_Success = MAKE\_STATUS(kStatusGroupGeneric, 0),
   
 kStatus\_FLASH\_InvalidArgument = MAKE\_STATUS(kStatusGroupGeneric, 4),
   
 kStatus\_FLASH\_SizeError = MAKE\_STATUS(kStatusGroupFlashDriver, 0),
   
 kStatus\_FLASH\_AlignmentError,
   
 kStatus\_FLASH\_AddressError = MAKE\_STATUS(kStatusGroupFlashDriver, 2),
   
 kStatus\_FLASH\_AccessError,
   
 kStatus\_FLASH\_ProtectionViolation,
   
 kStatus\_FLASH\_CommandFailure,
   
 kStatus\_FLASH\_UnknownProperty = MAKE\_STATUS(kStatusGroupFlashDriver, 6),
   
 kStatus\_FLASH\_EraseKeyError = MAKE\_STATUS(kStatusGroupFlashDriver, 7),
   
 kStatus\_FLASH\_RegionExecuteOnly,
   
 kStatus\_FLASH\_ExecuteInRamFunctionNotReady,
   
 kStatus\_FLASH\_CommandNotSupported = MAKE\_STATUS(kStatusGroupFlashDriver, 11),
   
 kStatus\_FLASH\_ReadOnlyProperty = MAKE\_STATUS(kStatusGroupFlashDriver, 12),
   
 kStatus\_FLASH\_InvalidPropertyValue,
   
 kStatus\_FLASH\_InvalidSpeculationOption,
   
 kStatus\_FLASH\_EccError,
   
 kStatus\_FLASH\_CompareError,
   
 kStatus\_FLASH\_RegulationLoss = MAKE\_STATUS(kStatusGroupFlashDriver, 0x12),
   
 kStatus\_FLASH\_InvalidWaitStateCycles,
   
 kStatus\_FLASH\_OutOfDateCfpaPage,
   
 kStatus\_FLASH\_BankIfrPageData = MAKE\_STATUS(kStatusGroupFlashDriver, 0x21),
   
 kStatus\_FLASH\_EncryptedRegionsEraseNotDoneAtOnce,
   
 kStatus\_FLASH\_ProgramVerificationNotAllowed,
   
 kStatus\_FLASH\_HashCheckError,
   
 kStatus\_FLASH\_SealedFfrRegion = MAKE\_STATUS(kStatusGroupFlashDriver, 0x25),
   
 kStatus\_FLASH\_FfrRegionWriteBroken,
   
 kStatus\_FLASH\_NmpaAccessNotAllowed,
   
 kStatus\_FLASH\_CmpaCfgDirectEraseNotAllowed,
   
 kStatus\_FLASH\_FfrBankIsLocked = MAKE\_STATUS(kStatusGroupFlashDriver, 0x29) }
- Flash driver status codes.*
- #define **kStatusGroupGeneric** 0
- Flash driver status group.*
- #define **kStatusGroupFlashDriver** 1
  - #define **MAKE\_STATUS**(group, code) (((group)\*100) + (code))
- Constructs a status code value from a group and a code number.*

## Flash API key

- enum **\_flash\_driver\_api\_keys** { **kFLASH\_ApiEraseKey** = FOUR\_CHAR\_CODE('l', 'f', 'e', 'k') }
- Enumeration for Flash driver API keys.*

- #define **FOUR\_CHAR\_CODE**(a, b, c, d) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))  
*Constructs the four character code for the Flash driver API key.*

## Initialization

- **status\_t FLASH\_Init (flash\_config\_t \*config)**  
*Initializes the global flash properties structure members.*

## Erasing

- **status\_t FLASH\_Erase (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)**  
*Erases the flash sectors encompassed by parameters passed into function.*

## Programming

- **status\_t FLASH\_Program (flash\_config\_t \*config, uint32\_t start, uint8\_t \*src, uint32\_t lengthInBytes)**  
*Programs flash with data at locations passed in through parameters.*

## Verification

- **status\_t FLASH\_VerifyErase (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes)**  
*Verifies an erasure of the desired flash area at a specified margin level.*
- **status\_t FLASH\_VerifyProgram (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, const uint8\_t \*expectedData, uint32\_t \*failedAddress, uint32\_t \*failedData)**  
*Verifies programming of the desired flash area at a specified margin level.*

## Properties

- **status\_t FLASHGetProperty (flash\_config\_t \*config, flash\_property\_tag\_t whichProperty, uint32\_t \*value)**  
*Returns the desired flash property.*

## 19.4 Data Structure Documentation

### 19.4.1 struct flash\_ecc\_log\_t

### 19.4.2 struct flash\_mode\_config\_t

### 19.4.3 struct flash\_ffr\_config\_t

### 19.4.4 struct flash\_config\_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

## Data Fields

- `uint32_t PFlashBlockBase`  
*A base address of the first PFlash block.*
- `uint32_t PFlashTotalSize`  
*The size of the combined PFlash block.*
- `uint32_t PFlashBlockCount`  
*A number of PFlash blocks.*
- `uint32_t PFlashPageSize`  
*The size in bytes of a page of PFlash.*
- `uint32_t PFlashSectorSize`  
*The size in bytes of a sector of PFlash.*

## Field Documentation

- (1) `uint32_t flash_config_t::PFlashTotalSize`
- (2) `uint32_t flash_config_t::PFlashBlockCount`
- (3) `uint32_t flash_config_t::PFlashPageSize`
- (4) `uint32_t flash_config_t::PFlashSectorSize`

## 19.5 Macro Definition Documentation

19.5.1 `#define MAKE_VERSION( major, minor, bugfix ) (((major) << 16) | ((minor) << 8) | (bugfix))`

19.5.2 `#define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))`

Version 2.1.4.

19.5.3 `#define FSL_FEATURE_FLASH_IP_IS_C040HD_ATFC (1)`

19.5.4 `#define kStatusGroupGeneric 0`

19.5.5 `#define MAKE_STATUS( group, code ) (((group)*100) + (code))`

19.5.6 `#define FOUR_CHAR_CODE( a, b, c, d ) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))`

## 19.6 Enumeration Type Documentation

### 19.6.1 enum \_flash\_driver\_version\_constants

Enumerator

- kFLASH\_DriverVersionName*** Flash driver version name.
- kFLASH\_DriverVersionMajor*** Major flash driver version.
- kFLASH\_DriverVersionMinor*** Minor flash driver version.
- kFLASH\_DriverVersionBugfix*** Bugfix for flash driver version.

### 19.6.2 enum \_flash\_status

Enumerator

- kStatus\_FLASH\_Success*** API is executed successfully.
- kStatus\_FLASH\_InvalidArgument*** Invalid argument.
- kStatus\_FLASH\_SizeError*** Error size.
- kStatus\_FLASH\_AlignmentError*** Parameter is not aligned with the specified baseline.
- kStatus\_FLASH\_AddressError*** Address is out of range.
- kStatus\_FLASH\_AccessError*** Invalid instruction codes and out-of bound addresses.
- kStatus\_FLASH\_ProtectionViolation*** The program/erase operation is requested to execute on protected areas.
- kStatus\_FLASH\_CommandFailure*** Run-time error during command execution.
- kStatus\_FLASH\_UnknownProperty*** Unknown property.
- kStatus\_FLASH\_EraseKeyError*** API erase key is invalid.
- kStatus\_FLASH\_RegionExecuteOnly*** The current region is execute-only.
- kStatus\_FLASH\_ExecuteInRamFunctionNotReady*** Execute-in-RAM function is not available.
- kStatus\_FLASH\_CommandNotSupported*** Flash API is not supported.
- kStatus\_FLASH\_ReadOnlyProperty*** The flash property is read-only.
- kStatus\_FLASH\_InvalidPropertyValue*** The flash property value is out of range.
- kStatus\_FLASH\_InvalidSpeculationOption*** The option of flash prefetch speculation is invalid.
- kStatus\_FLASH\_EccError*** A correctable or uncorrectable error during command execution.
- kStatus\_FLASH\_CompareError*** Destination and source memory contents do not match.
- kStatus\_FLASH\_RegulationLoss*** A loss of regulation during read.
- kStatus\_FLASH\_InvalidWaitStateCycles*** The wait state cycle set to r/w mode is invalid.
- kStatus\_FLASH\_OutOfDateCfpapage*** CFPA page version is out of date.
- kStatus\_FLASH\_BankIfrPageData*** Blank page cannot be read.
- kStatus\_FLASH\_EncryptedRegionsEraseNotDoneAtOnce*** Encrypted flash subregions are not erased at once.
- kStatus\_FLASH\_ProgramVerificationNotAllowed*** Program verification is not allowed when the encryption is enabled.
- kStatus\_FLASH\_HashCheckError*** Hash check of page data is failed.
- kStatus\_FLASH\_SealedFfrRegion*** The FFR region is sealed.
- kStatus\_FLASH\_FfrRegionWriteBroken*** The FFR Spec region is not allowed to be written discontinuously.

***kStatus\_FLASH\_NmpaAccessNotAllowed*** The NMPA region is not allowed to be read/written/erased.

***kStatus\_FLASH\_CmpaCfgDirectEraseNotAllowed*** The CMPA Cfg region is not allowed to be erased directly.

***kStatus\_FLASH\_FfrBankIsLocked*** The FFR bank region is locked.

### 19.6.3 enum \_flash\_driver\_api\_keys

Note

The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Enumerator

***kFLASH\_ApiEraseKey*** Key value used to validate all flash erase APIs.

### 19.6.4 enum flash\_property\_tag\_t

Enumerator

***kFLASH\_PropertyPflashSectorSize*** Pflash sector size property.

***kFLASH\_PropertyPflashTotalSize*** Pflash total size property.

***kFLASH\_PropertyPflashBlockSize*** Pflash block size property.

***kFLASH\_PropertyPflashBlockCount*** Pflash block count property.

***kFLASH\_PropertyPflashBlockBaseAddr*** Pflash block base address property.

***kFLASH\_PropertyPflashPageSize*** Pflash page size property.

***kFLASH\_PropertyPflashSystemFreq*** System Frequency System Frequency.

***kFLASH\_PropertyFfrSectorSize*** FFR sector size property.

***kFLASH\_PropertyFfrTotalSize*** FFR total size property.

***kFLASH\_PropertyFfrBlockBaseAddr*** FFR block base address property.

***kFLASH\_PropertyFfrPageSize*** FFR page size property.

### 19.6.5 enum \_flash\_max\_erase\_page\_value

Enumerator

***kFLASH\_MaxPagesToErase*** The max value in pages to erase.

### 19.6.6 enum \_flash\_alignment\_property

Enumerator

***kFLASH\_AlignmentUnitVerifyErase*** The alignment unit in bytes used for verify erase operation.

***kFLASH\_AlignmentUnitProgram*** The alignment unit in bytes used for program operation.

***kFLASH\_AlignmentUnitSingleWordRead*** The alignment unit in bytes used for verify program operation. The alignment unit in bytes used for SingleWordRead command.

### 19.6.7 enum \_flash\_read\_ecc\_option

Enumerator

***kFLASH\_ReadWithEccOff*** ECC is on.

### 19.6.8 enum \_flash\_read\_margin\_option

Enumerator

***kFLASH\_ReadMarginNormal*** Normal read.

***kFLASH\_ReadMarginVsProgram*** Margin vs. program

***kFLASH\_ReadMarginVsErase*** Margin vs. erase

***kFLASH\_ReadMarginIllegalBitCombination*** Illegal bit combination.

### 19.6.9 enum \_flash\_read\_dmacc\_option

Enumerator

***kFLASH\_ReadDmaccDisabled*** Memory word.

***kFLASH\_ReadDmaccEnabled*** DMACC word.

### 19.6.10 enum \_flash\_ramp\_control\_option

Enumerator

***kFLASH\_RampControlDivisionFactorReserved*** Reserved.

***kFLASH\_RampControlDivisionFactor256*** clk48mhz / 256 = 187.5KHz

***kFLASH\_RampControlDivisionFactor128*** clk48mhz / 128 = 375KHz

***kFLASH\_RampControlDivisionFactor64*** clk48mhz / 64 = 750KHz

## 19.7 Function Documentation

### 19.7.1 `status_t FLASH_Init( flash_config_t * config )`

This function checks and initializes the Flash module for the other Flash APIs.

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>config</i> | Pointer to the storage for the driver runtime state. |
|---------------|------------------------------------------------------|

Return values

|                                          |                                                                |
|------------------------------------------|----------------------------------------------------------------|
| <i>kStatus_FLASH_Success</i>             | API was executed successfully.                                 |
| <i>kStatus_FLASH_InvalidArgument</i>     | An invalid argument is provided.                               |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                   |
| <i>kStatus_FLASH_CommandNotSupported</i> | Flash API is not supported.                                    |
| <i>kStatus_FLASH_EccError</i>            | A correctable or uncorrectable error during command execution. |

### 19.7.2 status\_t FLASH\_Erase ( *flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key* )

This function erases the appropriate number of flash sectors based on the desired start address and length.

Parameters

|                      |                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------|
| <i>config</i>        | The pointer to the storage for the driver runtime state.                                                   |
| <i>start</i>         | The start address of the desired flash memory to be erased. The start address need to be 512bytes-aligned. |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be erased. Must be 512bytes-aligned.               |
| <i>key</i>           | The value used to validate all flash erase APIs.                                                           |

Return values

|                              |                                                                                                                                                |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FLASH_Success</i> | API was executed successfully; the appropriate number of flash sectors based on the desired start address and length were erased successfully. |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|

|                                          |                                                                |
|------------------------------------------|----------------------------------------------------------------|
| <i>kStatus_FLASH_InvalidArgument</i>     | An invalid argument is provided.                               |
| <i>kStatus_FLASH_AlignmentError</i>      | The parameter is not aligned with the specified baseline.      |
| <i>kStatus_FLASH_AddressError</i>        | The address is out of range.                                   |
| <i>kStatus_FLASH_EraseKeyError</i>       | The API erase key is invalid.                                  |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                   |
| <i>kStatus_FLASH_CommandNotSupported</i> | Flash API is not supported.                                    |
| <i>kStatus_FLASH_EccError</i>            | A correctable or uncorrectable error during command execution. |

### 19.7.3 **status\_t FLASH\_Program ( flash\_config\_t \* config, uint32\_t start, uint8\_t \* src, uint32\_t lengthInBytes )**

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

|                      |                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------|
| <i>config</i>        | A pointer to the storage for the driver runtime state.                                            |
| <i>start</i>         | The start address of the desired flash memory to be programmed. Must be 512bytes-aligned.         |
| <i>src</i>           | A pointer to the source buffer of data that is to be programmed into the flash.                   |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words), to be programmed. Must be 512bytes-aligned. |

Return values

|                              |                                                                                                                                    |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FLASH_Success</i> | API was executed successfully; the desired data were programmed successfully into flash based on desired start address and length. |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------|

|                                          |                                                                |
|------------------------------------------|----------------------------------------------------------------|
| <i>kStatus_FLASH_InvalidArgument</i>     | An invalid argument is provided.                               |
| <i>kStatus_FLASH_AlignmentError</i>      | Parameter is not aligned with the specified baseline.          |
| <i>kStatus_FLASH_AddressError</i>        | Address is out of range.                                       |
| <i>kStatus_FLASH_AccessError</i>         | Invalid instruction codes and out-of bounds addresses.         |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                   |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                   |
| <i>kStatus_FLASH_CommandNotSupported</i> | Flash API is not supported.                                    |
| <i>kStatus_FLASH_EccError</i>            | A correctable or uncorrectable error during command execution. |

#### 19.7.4 **status\_t FLASH\_Read ( flash\_config\_t \* config, uint32\_t start, uint8\_t \* dest, uint32\_t lengthInBytes )**

This function read the flash memory from a given flash area as determined by the start address and the length.

Parameters

|                      |                                                                         |
|----------------------|-------------------------------------------------------------------------|
| <i>config</i>        | A pointer to the storage for the driver runtime state.                  |
| <i>start</i>         | The start address of the desired flash memory to be read.               |
| <i>dest</i>          | A pointer to the dest buffer of data that is to be read from the flash. |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words), to be read.       |

Return values

|                                      |                                  |
|--------------------------------------|----------------------------------|
| <i>kStatus_FLASH_Success</i>         | API was executed successfully.   |
| <i>kStatus_FLASH_InvalidArgument</i> | An invalid argument is provided. |

|                                          |                                                                |
|------------------------------------------|----------------------------------------------------------------|
| <i>kStatus_FLASH_AlignmentError</i>      | Parameter is not aligned with the specified baseline.          |
| <i>kStatus_FLASH_AddressError</i>        | Address is out of range.                                       |
| <i>kStatus_FLASH_AccessError</i>         | Invalid instruction codes and out-of bounds addresses.         |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                   |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                   |
| <i>kStatus_FLASH_CommandNotSupported</i> | Flash API is not supported.                                    |
| <i>kStatus_FLASH_EccError</i>            | A correctable or uncorrectable error during command execution. |

### 19.7.5 **status\_t FLASH\_VerifyErase ( flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes )**

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

|                      |                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------|
| <i>config</i>        | A pointer to the storage for the driver runtime state.                                                       |
| <i>start</i>         | The start address of the desired flash memory to be verified. The start address need to be 512bytes-aligned. |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words), to be verified. Must be 512bytes-aligned.              |

Return values

|                                      |                                                                            |
|--------------------------------------|----------------------------------------------------------------------------|
| <i>kStatus_FLASH_Success</i>         | API was executed successfully; the specified FLASH region has been erased. |
| <i>kStatus_FLASH_InvalidArgument</i> | An invalid argument is provided.                                           |

|                                          |                                                                |
|------------------------------------------|----------------------------------------------------------------|
| <i>kStatus_FLASH_AlignmentError</i>      | Parameter is not aligned with specified baseline.              |
| <i>kStatus_FLASH_AddressError</i>        | Address is out of range.                                       |
| <i>kStatus_FLASH_AccessError</i>         | Invalid instruction codes and out-of bounds addresses.         |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                   |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                   |
| <i>kStatus_FLASH_CommandNotSupported</i> | Flash API is not supported.                                    |
| <i>kStatus_FLASH_EccError</i>            | A correctable or uncorrectable error during command execution. |

#### 19.7.6 **status\_t FLASH\_VerifyProgram ( flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, const uint8\_t \* expectedData, uint32\_t \* failedAddress, uint32\_t \* failedData )**

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

|                      |                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>config</i>        | A pointer to the storage for the driver runtime state.                                                                                                              |
| <i>start</i>         | The start address of the desired flash memory to be verified. need be 512bytes-aligned.                                                                             |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words), to be verified. need be 512bytes-aligned.                                                                     |
| <i>expectedData</i>  | A pointer to the expected data that is to be verified against.                                                                                                      |
| <i>failedAddress</i> | A pointer to the returned failing address.                                                                                                                          |
| <i>failedData</i>    | A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure. |

Return values

|                                          |                                                                                                                |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FLASH_Success</i>             | API was executed successfully; the desired data have been successfully programmed into specified FLASH region. |
| <i>kStatus_FLASH_InvalidArgument</i>     | An invalid argument is provided.                                                                               |
| <i>kStatus_FLASH_AlignmentError</i>      | Parameter is not aligned with specified baseline.                                                              |
| <i>kStatus_FLASH_AddressError</i>        | Address is out of range.                                                                                       |
| <i>kStatus_FLASH_AccessError</i>         | Invalid instruction codes and out-of bounds addresses.                                                         |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                                                                   |
| <i>kStatus_FLASH_CommandFailure</i>      | Run-time error during the command execution.                                                                   |
| <i>kStatus_FLASH_CommandNotSupported</i> | Flash API is not supported.                                                                                    |
| <i>kStatus_FLASH_EccError</i>            | A correctable or uncorrectable error during command execution.                                                 |

### 19.7.7 **status\_t FLASHGetProperty ( flash\_config\_t \* config, flash\_property\_tag\_t whichProperty, uint32\_t \* value )**

Parameters

|                      |                                                                               |
|----------------------|-------------------------------------------------------------------------------|
| <i>config</i>        | A pointer to the storage for the driver runtime state.                        |
| <i>whichProperty</i> | The desired property from the list of properties in enum flash_property_tag_t |
| <i>value</i>         | A pointer to the value returned for the desired flash property.               |

Return values

|                                      |                                                                        |
|--------------------------------------|------------------------------------------------------------------------|
| <i>kStatus_FLASH_Success</i>         | API was executed successfully; the flash property was stored to value. |
| <i>kStatus_FLASH_InvalidArgument</i> | An invalid argument is provided.                                       |

|                                       |                          |
|---------------------------------------|--------------------------|
| <i>kStatus_FLASH_-UnknownProperty</i> | An unknown property tag. |
|---------------------------------------|--------------------------|

## 19.8 IAP\_FFR Driver

### 19.8.1 Overview

#### Files

- file `fsl_iap_ffr.h`

#### Macros

- `#define ALIGN_DOWN(x, a) ((x) & (uint32_t)(-((int32_t)(a))))`  
*Alignment(down) utility.*
- `#define ALIGN_UP(x, a) (-((int32_t)((uint32_t)(-((int32_t)(x))) & (uint32_t)(-((int32_t)(a))))))`  
*Alignment(up) utility.*

#### Enumerations

- `enum _flash_ffr_page_offset {`  
`kFfrPageOffset_CFPA = 0,`  
`kFfrPageOffset_CFPA_Scratch = 0,`  
`kFfrPageOffset_CFPA_Cfg = 1,`  
`kFfrPageOffset_CFPA_CfgPong = 2,`  
`kFfrPageOffset_CMPA = 3,`  
`kFfrPageOffset_CMPA_Cfg = 3,`  
`kFfrPageOffset_CMPA_Key = 4,`  
`kFfrPageOffset_NMPA = 7,`  
`kFfrPageOffset_NMPA_Romcp = 7,`  
`kFfrPageOffset_NMPA_Repair = 9,`  
`kFfrPageOffset_NMPA_Cfg = 15,`  
`kFfrPageOffset_NMPA_End = 16 }`  
*flash ffr page offset.*
- `enum _flash_ffr_page_num {`  
`kFfrPageNum_CFPA = 3,`  
`kFfrPageNum_CMPA = 4,`  
`kFfrPageNum_NMPA = 10 }`  
*flash ffr page number.*

#### Flash IFR version

- `#define FSL_FLASH_IFR_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`  
*Flash IFR driver version for SDK.*

## FFR APIs

- `status_t FFR_Init (flash_config_t *config)`  
*Initializes the global FFR properties structure members.*
- `status_t FFR_Lock_All (flash_config_t *config)`  
*Enable firewall for all flash banks.*
- `status_t FFR_InfieldPageWrite (flash_config_t *config, uint8_t *page_data, uint32_t valid_len)`  
*APIs to access CFPA pages.*
- `status_t FFR_GetCustomerInfieldData (flash_config_t *config, uint8_t *pData, uint32_t offset, uint32_t len)`  
*APIs to access CFPA pages.*
- `status_t FFR_CustFactoryPageWrite (flash_config_t *config, uint8_t *page_data, bool seal_part)`  
*APIs to access CMPA pages.*
- `status_t FFR_GetCustomerData (flash_config_t *config, uint8_t *pData, uint32_t offset, uint32_t len)`  
*APIs to access CMPA page.*
- `status_t FFR_GetUUID (flash_config_t *config, uint8_t *uuid)`  
*APIs to access CMPA page.*
- `status_t FFR_KeystoreWrite (flash_config_t *config, ffr_key_store_t *pKeyStore)`  
*This routine writes the 3 pages allocated for Key store data.,*
- `status_t FFR_KeystoreGetAC (flash_config_t *config, uint8_t *pActivationCode)`  
*Get/Read Key store code routines.*
- `status_t FFR_KeystoreGetKC (flash_config_t *config, uint8_t *pKeyCode, ffr_key_type_t keyIndex)`  
*Get/Read Key store code routines.*

### 19.8.2 Macro Definition Documentation

#### 19.8.2.1 `#define FSL_FLASH_IFR_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

Version 2.1.0.

#### 19.8.2.2 `#define ALIGN_DOWN( x, a ) ((x) & (uint32_t)(-((int32_t)(a))))`

#### 19.8.2.3 `#define ALIGN_UP( x, a ) (-((int32_t)((uint32_t)(-((int32_t)(x))) & (uint32_t)(-((int32_t)(a))))))`

### 19.8.3 Enumeration Type Documentation

#### 19.8.3.1 `enum _flash_ffr_page_offset`

Enumerator

*kFfrPageOffset\_CFPA* Customer In-Field programmed area.

*kFfrPageOffset\_CFPA\_Scratch* CFPA Scratch page.

*kFfrPageOffset\_CFPA\_Cfg* CFPA Configuration area (Ping page)

*kFfrPageOffset\_CFPA\_CfgPong* Same as CFPA page (Pong page)  
*kFfrPageOffset\_CMPA* Customer Manufacturing programmed area.  
*kFfrPageOffset\_CMPA\_Cfg* CMPA Configuration area (Part of CMPA)  
*kFfrPageOffset\_CMPA\_Key* Key Store area (Part of CMPA)  
*kFfrPageOffset\_NMPA* NXP Manufacturing programmed area.  
*kFfrPageOffset\_NMPA\_Romcp* ROM patch area (Part of NMPA)  
*kFfrPageOffset\_NMPA\_Repair* Repair area (Part of NMPA)  
*kFfrPageOffset\_NMPA\_Cfg* NMPA configuration area (Part of NMPA)  
*kFfrPageOffset\_NMPA\_End* Reserved (Part of NMPA)

### 19.8.3.2 enum \_flash\_ffr\_page\_num

Enumerator

*kFfrPageNum\_CFPA* Customer In-Field programmed area.  
*kFfrPageNum\_CMPA* Customer Manufacturing programmed area.  
*kFfrPageNum\_NMPA* NXP Manufacturing programmed area.

## 19.8.4 Function Documentation

### 19.8.4.1 status\_t FFR\_Init ( flash\_config\_t \* config )

Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>config</i> | A pointer to the storage for the driver runtime state. |
|---------------|--------------------------------------------------------|

Return values

|                                      |                                  |
|--------------------------------------|----------------------------------|
| <i>kStatus_FLASH_Success</i>         | API was executed successfully.   |
| <i>kStatus_FLASH_InvalidArgument</i> | An invalid argument is provided. |

### 19.8.4.2 status\_t FFR\_Lock\_All ( flash\_config\_t \* config )

CFPA, CMPA, and NMPA flash areas region will be locked, After this function executed; Unless the board is reset again.

Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>config</i> | A pointer to the storage for the driver runtime state. |
|---------------|--------------------------------------------------------|

Return values

|                                       |                                  |
|---------------------------------------|----------------------------------|
| <i>kStatus_FLASH_Success</i>          | An invalid argument is provided. |
| <i>kStatus_FLASH_Invalid-Argument</i> | An invalid argument is provided. |

#### 19.8.4.3 status\_t FFR\_InfieldPageWrite ( *flash\_config\_t \* config, uint8\_t \* page\_data, uint32\_t valid\_len* )

This routine will erase CFPA and program the CFPA page with passed data.

Parameters

|                  |                                                                                |
|------------------|--------------------------------------------------------------------------------|
| <i>config</i>    | A pointer to the storage for the driver runtime state.                         |
| <i>page_data</i> | A pointer to the source buffer of data that is to be programmed into the CFPA. |
| <i>valid_len</i> | The length, given in bytes, to be programmed.                                  |

Return values

|                                         |                                                             |
|-----------------------------------------|-------------------------------------------------------------|
| <i>kStatus_FLASH_Success</i>            | The desire page-data were programed successfully into CFPA. |
| <i>kStatus_FLASH_Invalid-Argument</i>   | An invalid argument is provided.                            |
| <i>kStatus_FTFx_Address-Error</i>       | Address is out of range.                                    |
| <i>kStatus_FLASH_Ffr-BankIsLocked</i>   | The CFPA was locked.                                        |
| <i>kStatus_FLASH_OutOf-DateCfpaPage</i> | It is not newest CFPA page.                                 |

#### 19.8.4.4 status\_t FFR\_GetCustomerInfieldData ( *flash\_config\_t \* config, uint8\_t \* pData, uint32\_t offset, uint32\_t len* )

Generic read function, used by customer to read data stored in 'Customer In-field Page'.

## Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>config</i> | A pointer to the storage for the driver runtime state.                                 |
| <i>pData</i>  | A pointer to the dest buffer of data that is to be read from 'Customer In-field Page'. |
| <i>offset</i> | An offset from the 'Customer In-field Page' start address.                             |
| <i>len</i>    | The length, given in bytes, to be read.                                                |

## Return values

|                                       |                                         |
|---------------------------------------|-----------------------------------------|
| <i>kStatus_FLASH_Success</i>          | Get data from 'Customer In-field Page'. |
| <i>kStatus_FLASH_Invalid-Argument</i> | An invalid argument is provided.        |
| <i>kStatus_FTFx_Address-Error</i>     | Address is out of range.                |
| <i>kStatus_FLASH_-CommandFailure</i>  | access error.                           |

**19.8.4.5 status\_t FFR\_CustFactoryPageWrite ( flash\_config\_t \* *config*, uint8\_t \* *page\_data*, bool *seal\_part* )**

This routine will erase "customer factory page" and program the page with passed data. If 'seal\_part' parameter is TRUE then the routine will compute SHA256 hash of the page contents and then programs the pages. 1.During development customer code uses this API with 'seal\_part' set to FALSE. 2.During manufacturing this parameter should be set to TRUE to seal the part from further modifications 3.This routine checks if the page is sealed or not. A page is said to be sealed if the SHA256 value in the page has non-zero value. On boot ROM locks the firewall for the region if hash is programmed anyways. So, write/erase commands will fail eventually.

## Parameters

|                  |                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------|
| <i>config</i>    | A pointer to the storage for the driver runtime state.                                            |
| <i>page_data</i> | A pointer to the source buffer of data that is to be programmed into the "customer factory page". |
| <i>seal_part</i> | Set fasle for During development customer code.                                                   |

## Return values

|                                       |                                                             |
|---------------------------------------|-------------------------------------------------------------|
| <i>kStatus_FLASH_Success</i>          | The desire page-data were programed successfully into CMPA. |
| <i>kStatus_FLASH_Invalid-Argument</i> | Parameter is not aligned with the specified baseline.       |
| <i>kStatus_FTFx_Address-Error</i>     | Address is out of range.                                    |
| <i>kStatus_FLASH_-CommandFailure</i>  | access error.                                               |

#### 19.8.4.6 status\_t FFR\_GetCustomerData ( *flash\_config\_t \* config, uint8\_t \* pData, uint32\_t offset, uint32\_t len* )

Read data stored in 'Customer Factory CFG Page'.

Parameters

|               |                                                                                             |
|---------------|---------------------------------------------------------------------------------------------|
| <i>config</i> | A pointer to the storage for the driver runtime state.                                      |
| <i>pData</i>  | A pointer to the dest buffer of data that is to be read from the Customer Factory CFG Page. |
| <i>offset</i> | Address offset relative to the CMPA area.                                                   |
| <i>len</i>    | The length, given in bytes to be read.                                                      |

Return values

|                                       |                                                       |
|---------------------------------------|-------------------------------------------------------|
| <i>kStatus_FLASH_Success</i>          | Get data from 'Customer Factory CFG Page'.            |
| <i>kStatus_FLASH_Invalid-Argument</i> | Parameter is not aligned with the specified baseline. |
| <i>kStatus_FTFx_Address-Error</i>     | Address is out of range.                              |
| <i>kStatus_FLASH_-CommandFailure</i>  | access error.                                         |

#### 19.8.4.7 status\_t FFR\_GetUUID ( *flash\_config\_t \* config, uint8\_t \* uuid* )

1.SW should use this API routine to get the UUID of the chip. 2.Calling routine should pass a pointer to buffer which can hold 128-bit value.

#### **19.8.4.8 status\_t FFR\_KeystoreWrite ( flash\_config\_t \* config, ffr\_key\_store\_t \* pKeyStore )**

- 1.Used during manufacturing. Should write pages when 'customer factory page' is not in sealed state.
- 2.Optional routines to set individual data members (activation code, key codes etc) to construct the key store structure in RAM before committing it to IFR/FFR.

Parameters

|                  |                                                                                                         |
|------------------|---------------------------------------------------------------------------------------------------------|
| <i>config</i>    | A pointer to the storage for the driver runtime state.                                                  |
| <i>pKeyStore</i> | A Pointer to the 3 pages allocated for Key store data. that will be written to 'customer factory page'. |

Return values

|                                      |                                                       |
|--------------------------------------|-------------------------------------------------------|
| <i>kStatus_FLASH_Success</i>         | The key were programed successfully into FFR.         |
| <i>kStatus_FLASH_InvalidArgument</i> | Parameter is not aligned with the specified baseline. |
| <i>kStatus_FTFx_AddressError</i>     | Address is out of range.                              |
| <i>kStatus_FLASH_CommandFailure</i>  | access error.                                         |

#### **19.8.4.9 status\_t FFR\_KeystoreGetAC ( flash\_config\_t \* config, uint8\_t \* pActivationCode )**

1. Calling code should pass buffer pointer which can hold activation code 1192 bytes.
2. Check if flash aperture is small or regular and read the data appropriately.

#### **19.8.4.10 status\_t FFR\_KeystoreGetKC ( flash\_config\_t \* config, uint8\_t \* pKeyCode, ffr\_key\_type\_t keyIndex )**

1. Calling code should pass buffer pointer which can hold key code 52 bytes.
2. Check if flash aperture is small or regular and read the data appropriately.
3. keyIndex specifies which key code is read.

## 19.9 IAP\_KBP Driver

### 19.9.1 Overview

#### Data Structures

- struct [kb\\_region\\_t](#)  
*Memory region definition.* [More...](#)
- struct [kb\\_load\\_sb\\_t](#)  
*User-provided options passed into `kb_init()`.* [More...](#)
- struct [memory\\_region\\_interface\\_t](#)  
*Interface to memory operations for one region of memory.* [More...](#)
- struct [memory\\_map\\_entry\\_t](#)  
*Structure of a memory map entry.* [More...](#)

#### Macros

- #define [kStatusGroup\\_RomApi](#) (108U)  
*ROM API status group number.*

#### Enumerations

- enum {
 [kStatus\\_RomApiExecuteCompleted](#) = kStatus\_Success,  
[kStatus\\_RomApiNeedMoreData](#),  
[kStatus\\_RomApiBufferSizeNotEnough](#),  
[kStatus\\_RomApiInvalidBuffer](#) }
   
*ROM API status codes.*
- enum [kb\\_operation\\_t](#) {
 [kRomAuthenticateImage](#) = 1,  
[kRomLoadImage](#) = 2 }
   
*Details of the operation to be performed by the ROM.*
- enum [\\_kb\\_security\\_profile](#)
  
*Security constraint flags, Security profile flags.*

#### Functions

- [status\\_t kb\\_init](#) (kb\_session\_ref\_t \*\*session, const kb\_options\_t \*options)  
*Initialize ROM API for a given operation.*
- [status\\_t kb\\_deinit](#) (kb\_session\_ref\_t \*session)  
*Cleans up the ROM API context.*
- [status\\_t kb\\_execute](#) (kb\_session\_ref\_t \*session, const uint8\_t \*data, uint32\_t dataLength)  
*Perform the operation configured during init.*

## 19.9.2 Data Structure Documentation

### 19.9.2.1 struct kb\_region\_t

### 19.9.2.2 struct kb\_load\_sb\_t

The buffer field is a pointer to memory provided by the caller for use by Kboot during execution of the operation. Minimum size is the size of each certificate in the chain plus 432 bytes additional per certificate.

The profile field is a mask that specifies which features are required in the SB file or image being processed. This includes the minimum AES and RSA key sizes. See the \_kb\_security\_profile enum for profile mask constants. The image being loaded or authenticated must match the profile or an error will be returned.

minBuildNumber is an optional field that can be used to prevent version rollback. The API will check the build number of the image, and if it is less than minBuildNumber will fail with an error.

maxImageLength is used to verify the offsetToCertificateBlockHeaderInBytes value at the beginning of a signed image. It should be set to the length of the SB file. If verifying an image in flash, it can be set to the internal flash size or a large number like 0x10000000.

userRHK can optionally be used by the user to override the RHK in IFR. If userRHK is not NULL, it points to a 32-byte array containing the SHA-256 of the root certificate's RSA public key.

The regions field points to an array of memory regions that the SB file being loaded is allowed to access. If regions is NULL, then all memory is accessible by the SB file. This feature is required to prevent a malicious image from erasing good code or RAM contents while it is being loaded, only for us to find that the image is inauthentic when we hit the end of the section.

overrideSBBootSectionID lets the caller override the default section of the SB file that is processed during a kKbootLoadSB operation. By default, the section specified in the firstBootableSectionID field of the SB header is loaded. If overrideSBBootSectionID is non-zero, then the section with the given ID will be loaded instead.

The userSBKEK field lets a user provide their own AES-256 key for unwrapping keys in an SB file during the kKbootLoadSB operation. userSBKEK should point to a 32-byte AES-256 key. If userSBKEK is N-ULL then the IFR SBKEK will be used. After [kb\\_init\(\)](#) returns, the caller should zero out the data pointed to by userSBKEK, as the API will have installed the key in the CAU3.

### 19.9.2.3 struct memory\_region\_interface\_t

#### 19.9.2.4 struct memory\_map\_entry\_t

### 19.9.3 Enumeration Type Documentation

#### 19.9.3.1 anonymous enum

Enumerator

*kStatus\_RomApiExecuteCompleted* ROM successfully process the whole sb file/boot image.

*kStatus\_RomApiNeedMoreData* ROM needs more data to continue processing the boot image.

*kStatus\_RomApiBufferSizeNotEnough* The user buffer is not enough for use by Kboot during execution of the operation.

*kStatus\_RomApiInvalidBuffer* The user buffer is not ok for sbloader or authentication.

#### 19.9.3.2 enum kb\_operation\_t

The [kRomAuthenticateImage](#) operation requires the entire signed image to be available to the application.

Enumerator

*kRomAuthenticateImage* Authenticate a signed image.

*kRomLoadImage* Load SB file.

### 19.9.4 Function Documentation

#### 19.9.4.1 status\_t kb\_init ( kb\_session\_ref\_t \*\* session, const kb\_options\_t \* options )

Init the ROM API based on the options provided by the application in the second argument. Every call to rom\_init() should be paired with a call to rom\_deinit().

Return values

|                                                   |                                                                                   |
|---------------------------------------------------|-----------------------------------------------------------------------------------|
| <a href="#">kStatus_Success</a>                   | API was executed successfully.                                                    |
| <a href="#">kStatus_InvalidArgument</a>           | An invalid argument is provided.                                                  |
| <a href="#">kStatus_RomApiBufferSizeNotEnough</a> | The user buffer is not enough for use by Kboot during execution of the operation. |

|                                             |                                                                              |
|---------------------------------------------|------------------------------------------------------------------------------|
| <i>kStatus_RomApiInvalid-Buffer</i>         | The user buffer is not ok for sbloader or authentication.                    |
| <i>kStatus_SKBOOT_Fail</i>                  | Return the failed status of secure boot.                                     |
| <i>kStatus_SKBOOT_KeystoreMarkerInvalid</i> | The key code for the particular PRINCE region is not present in the keystore |
| <i>kStatus_SKBOOT_Success</i>               | Return the successful status of secure boot.                                 |

#### 19.9.4.2 **status\_t kb\_deinit ( kb\_session\_ref\_t \* session )**

After this call, the context parameter can be reused for another operation by calling rom\_init() again.

Return values

|                        |                               |
|------------------------|-------------------------------|
| <i>kStatus_Success</i> | API was executed successfully |
|------------------------|-------------------------------|

#### 19.9.4.3 **status\_t kb\_execute ( kb\_session\_ref\_t \* session, const uint8\_t \* data, uint32\_t dataLength )**

This application must call this API repeatedly, passing in sequential chunks of data from the boot image (SB file) that is to be processed. The ROM will perform the selected operation on this data and return. The application may call this function with as much or as little data as it wishes, which can be used to select the granularity of time given to the application in between executing the operation.

Parameters

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <i>session</i>    | Current ROM context pointer.                                      |
| <i>data</i>       | Buffer of boot image data provided to the ROM by the application. |
| <i>dataLength</i> | Length in bytes of the data in the buffer provided to the ROM.    |

Return values

|                                        |                                                          |
|----------------------------------------|----------------------------------------------------------|
| <i>kStatus_Success</i>                 | ROM successfully process the part of sb file/boot image. |
| <i>kStatus_RomApiExecute-Completed</i> | ROM successfully process the whole sb file/boot image.   |

|                                          |                                                                                       |
|------------------------------------------|---------------------------------------------------------------------------------------|
| <i>kStatus_Fail</i>                      | An error occurred while executing the operation.                                      |
| <i>kStatus_RomApiNeedMoreData</i>        | No error occurred, but the ROM needs more data to continue processing the boot image. |
| <i>kStatus_RomApiBufferSizeNotEnough</i> | user buffer is not enough for use by Kboot during execution of the operation.         |

# Chapter 20

## INPUTMUX: Input Multiplexing Driver

### 20.1 Overview

The MCUXpresso SDK provides a driver for the Input multiplexing (INPUTMUX).

It configures the inputs to the pin interrupt block, DMA trigger, and frequency measure function. Once configured, the clock is not needed for the inputmux.

### 20.2 Input Multiplexing Driver operation

INPUTMUX\_AttachSignal function configures the specified input

### 20.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/inputmux

## Files

- file [fsl\\_inputmux.h](#)
- file [fsl\\_inputmux\\_connections.h](#)

## Functions

- void [INPUTMUX\\_Init](#) (INPUTMUX\_Type \*base)  
*Initialize INPUTMUX peripheral.*
- void [INPUTMUX\\_AttachSignal](#) (INPUTMUX\_Type \*base, uint32\_t index, [inputmux\\_connection\\_t](#) connection)  
*Attaches a signal.*
- void [INPUTMUX\\_Deinit](#) (INPUTMUX\_Type \*base)  
*Deinitialize INPUTMUX peripheral.*

## Input multiplexing connections

- enum [inputmux\\_connection\\_t](#) {  
    kINPUTMUX\_Sct0In0ToSct0 = 0U + (SCT0\_INMUX0 << PMUX\_SHIFT),  
    kINPUTMUX\_Sai1RxBclkToSct0 = 76U + (SCT0\_INMUX0 << PMUX\_SHIFT),  
    kINPUTMUX\_Sai1RxSyncOutToTimer0Captsel = 64U + (TIMER0CAPTSEL0 << PMUX\_SHIFT),  
    kINPUTMUX\_Sai1RxSyncOutToTimer1Captsel = 64U + (TIMER1CAPTSEL0 << PMUX\_SHIFT),  
    kINPUTMUX\_Sai1RxSyncOutToTimer2Captsel = 64U + (TIMER2CAPTSEL0 << PMUX\_SHIFT),  
    kINPUTMUX\_Sai1RxSyncOutToTimer3Captsel = 64U + (TIMER3CAPTSEL0 << PMUX\_SHIFT)

```

FT),
kINPUTMUX_Sai1RxSyncOutToTimer4Captsel = 64U + (TIMER4CAPTSEL0 << PMUX_SHIFT),
kINPUTMUX_Sai1RxSyncOutToTimer0Trigger = 64U + (TIMER0TRIGIN << PMUX_SHIFT),
kINPUTMUX_Sai1RxSyncOutToTimer1Trigger = 64U + (TIMER1TRIGIN << PMUX_SHIFT),
kINPUTMUX_Sai1RxSyncOutToTimer2Trigger = 64U + (TIMER2TRIGIN << PMUX_SHIFT),
kINPUTMUX_Sai1RxSyncOutToTimer3Trigger = 64U + (TIMER3TRIGIN << PMUX_SHIFT),
kINPUTMUX_Sai1RxSyncOutToTimer4Trigger = 64U + (TIMER4TRIGIN << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToSmartDma = 70U + (SMARTDMAARCHB_INMUX0 << PMUX_SHIFT),
kINPUTMUX_GpioPort1Pin31ToPintsel = 63U + (PINTSEL0 << PMUX_SHIFT),
kINPUTMUX_EvtgOut1AToFreqmeasRef = 9u + (FREQMEAS_REF_REG << PMUX_SHIFT)
,
kINPUTMUX_EvtgOut1AToFreqmeasTar = 9u + (FREQMEAS_TAR_REG << PMUX_SHIFT)
,
kINPUTMUX_Gpio3PinEventTrig1ToCmp0Trigger = 40U + (CMP0_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToCmp1Trigger = 40U + (CMP1_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToCmp2Trigger = 40U + (CMP2_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_WuuToAdc0Trigger = 65U + (ADC0_TRIG0 << PMUX_SHIFT),
kINPUTMUX_WuuToAdc1Trigger = 65U + (ADC1_TRIG0 << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToDac0Trigger = 31U + (DAC0_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToDac1Trigger = 31U + (DAC1_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToDac2Trigger = 31U + (DAC2_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc0Trigger = 51U + (ENC0_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc0Home = 51U + (ENC0_HOME_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc0Index = 51U + (ENC0_INDEX_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc0Phaseb = 51U + (ENC0_PHASEB_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc0Phasea = 51U + (ENC0_PHASEA_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc1Trigger = 51U + (ENC1_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc1Home = 51U + (ENC1_HOME_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc1Index = 51U + (ENC1_INDEX_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc1Phaseb = 51U + (ENC1_PHASEB_REG << PMUX_SHIFT),
kINPUTMUX_TrigIn9ToEnc1Phasea = 51U + (ENC1_PHASEA_REG << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm0ExtSync = 60U + (FlexPWM0_SM0_EXTSYNC_REG << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm1ExtSync = 60U + (FlexPWM0_SM1_EXTSYNC_REG << PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm2ExtSync = 60U + (FlexPWM0_SM2_EXTSYNC_REG << PMUX_SHIFT)

```

```

YNC_REG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm3ExtSync = 60U + (FlexPWM0_SM3_EXTS-
YNC_REG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm0Exta = 60U + (FlexPWM0_SM0_EXTA_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm1Exta = 60U + (FlexPWM0_SM1_EXTA_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm2Exta = 60U + (FlexPWM0_SM2_EXTA_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Sm3Exta = 60U + (FlexPWM0_SM3_EXTA_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0ExtForce = 60U + (FlexPWM0_EXTFORCE_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault0 = 60U + (FlexPWM0_FAULT0_REG <<
PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault1 = 60U + (FlexPWM0_FAULT1_REG <<
PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault2 = 60U + (FlexPWM0_FAULT2_REG <<
PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm0Fault3 = 60U + (FlexPWM0_FAULT3_REG <<
PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm0ExtSync = 60U + (FlexPWM1_SM0_EXTS-
YNC_REG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm1ExtSync = 60U + (FlexPWM1_SM1_EXTS-
YNC_REG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm2ExtSync = 60U + (FlexPWM1_SM2_EXTS-
YNC_REG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm3ExtSync = 60U + (FlexPWM1_SM3_EXTS-
YNC_REG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm0Exta = 60U + (FlexPWM1_SM0_EXTA_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm1Exta = 60U + (FlexPWM1_SM1_EXTA_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm2Exta = 60U + (FlexPWM1_SM2_EXTA_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Sm3Exta = 60U + (FlexPWM1_SM3_EXTA_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1ExtForce = 60U + (FlexPWM1_EXTFORCE_R-
EG << PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault0 = 60U + (FlexPWM1_FAULT0_REG <<
PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault1 = 60U + (FlexPWM1_FAULT1_REG <<
PMUX_SHIFT) ,
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault2 = 60U + (FlexPWM1_FAULT2_REG <<

```

```

PMUX_SHIFT),
kINPUTMUX_Gpio3PinEventTrig1ToFlexPwm1Fault3 = 60U + (FlexPWM1_FAULT3_REG <<
PMUX_SHIFT),
kINPUTMUX_ExttrigIn1ToPwm0ExtClk = 5U + (PWM0_EXT_CLK_REG << PMUX_SHIFT),
kINPUTMUX_ExttrigIn1ToPwm1ExtClk = 5U + (PWM1_EXT_CLK_REG << PMUX_SHIFT),
kINPUTMUX_TmprOut1ToEvtgTrigger = 57U + (EVTG_TRIG0_REG << PMUX_SHIFT),
kINPUTMUX_Lpflexcomm9Trig3ToUsbfsTrigger = 9U + (USBFS_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_Lptmr1ToTsiTrigger = 1U + (TSI_TRIG_REG << PMUX_SHIFT),
kINPUTMUX_EnetPpsOut0ToExtTrigger = 47U + (EXT_TRIG0_REG << PMUX_SHIFT),
kINPUTMUX_WuuToSincFilterChTrigger = 56U + (SINC_FILTER_CH0_REG << PMUX_SH-
IFT),
kINPUTMUX_FlexioCh7ToOpamp0Trigger = 54U + (OPAMP0_TRIG_REG << PMUX_SHIFT)
,
kINPUTMUX_FlexioCh7ToOpamp1Trigger = 54U + (OPAMP1_TRIG_REG << PMUX_SHIFT)
,
kINPUTMUX_FlexioCh7ToOpamp2Trigger = 54U + (OPAMP2_TRIG_REG << PMUX_SHIFT)
,
kINPUTMUX_WuuToFlexcomm0Trigger = 42U + (FLEXCOMM0_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm1Trigger = 42U + (FLEXCOMM1_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm2Trigger = 42U + (FLEXCOMM2_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm3Trigger = 42U + (FLEXCOMM3_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm4Trigger = 42U + (FLEXCOMM4_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm5Trigger = 42U + (FLEXCOMM5_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm6Trigger = 42U + (FLEXCOMM6_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm7Trigger = 42U + (FLEXCOMM7_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm8Trigger = 42U + (FLEXCOMM8_TRIG_REG << PMUX_SH-
IFT),
kINPUTMUX_WuuToFlexcomm9Trigger = 42U + (FLEXCOMM9_TRIG_REG << PMUX_SH-
IFT) }

```

*INPUTMUX connections type.*

- enum `inputmux_signal_t`

```

kINPUTMUX_FlexSpi0RxToDma0Ch1Ena = 1U + (DMA0_REQ_ENABLE0_REG << ENA_S-
HIFT),
kINPUTMUX_Evtg0Out0AToDma0Ch31Ena = 31U + (DMA0_REQ_ENABLE0_REG << EN-
A_SHIFT),
kINPUTMUX_FlexIO0ShiftRegister2RequestToDma0Ch63Ena = 31U + (DMA0_REQ_ENABL-

```

```

E1_REG << ENA_SHIFT) ,
kINPUTMUX_I3c0RxToDma0Ch95Ena = 31U + (DMA0_REQ_ENABLE2_REG << ENA_SH-
IFT) ,
kINPUTMUX_Tsi0OutOfRangeToDma0Ch121Ena = 25U + (DMA0_REQ_ENABLE3_REG <<
ENA_SHIFT) ,
kINPUTMUX_Evtg0Out0AToDma1Ch31Ena = 31U + (DMA1_REQ_ENABLE0_REG << EN-
A_SHIFT) ,
kINPUTMUX_FlexIO0ShiftRegister2RequestToDma1Ch63Ena = 31U + (DMA1_REQ_ENABL-
E1_REG << ENA_SHIFT) ,
kINPUTMUX_I3c0RxToDma1Ch95Ena = 31U + (DMA1_REQ_ENABLE2_REG << ENA_SH-
IFT) }

```

*INPUTMUX signal enable/disable type.*

- #define **SCT0\_INMUX0** 0x00U

*Periphinmux IDs.*

- #define **TIMER0CAPTSEL0** 0x20U
- #define **TIMER0TRIGIN** 0x30U
- #define **TIMER1CAPTSEL0** 0x40U
- #define **TIMER1TRIGIN** 0x50U
- #define **TIMER2CAPTSEL0** 0x60U
- #define **TIMER2TRIGIN** 0x70U
- #define **SMARTDMAARCHB\_INMUX0** 0xA0U
- #define **PINTSEL0** 0xC0U
- #define **FREQMEAS\_REF\_REG** 0x180U
- #define **FREQMEAS\_TAR\_REG** 0x184U
- #define **TIMER3CAPTSEL0** 0x1A0U
- #define **TIMER3TRIGIN** 0x1B0U
- #define **TIMER4CAPTSEL0** 0x1C0U
- #define **TIMER4TRIGIN** 0x1D0U
- #define **CMP0\_TRIG\_REG** 0x260U
- #define **ADC0\_TRIG0** 0x280U
- #define **ADC1\_TRIG0** 0x2C0U
- #define **DAC0\_TRIG\_REG** 0x300U
- #define **DAC1\_TRIG\_REG** 0x320U
- #define **DAC2\_TRIG\_REG** 0x340U
- #define **ENC0\_TRIG\_REG** 0x360U
- #define **ENC0\_HOME\_REG** 0x364U
- #define **ENC0\_INDEX\_REG** 0x368U
- #define **ENC0\_PHASEB\_REG** 0x36CU
- #define **ENC0\_PHASEA\_REG** 0x370U
- #define **ENC1\_TRIG\_REG** 0x380U
- #define **ENC1\_HOME\_REG** 0x384U
- #define **ENC1\_INDEX\_REG** 0x388U
- #define **ENC1\_PHASEB\_REG** 0x38CU
- #define **ENC1\_PHASEA\_REG** 0x390U
- #define **FlexPWM0\_SM0\_EXTSYNC\_REG** 0x3A0U
- #define **FlexPWM0\_SM1\_EXTSYNC\_REG** 0x3A4U
- #define **FlexPWM0\_SM2\_EXTSYNC\_REG** 0x3A8U
- #define **FlexPWM0\_SM3\_EXTSYNC\_REG** 0x3ACU
- #define **FlexPWM0\_SM0\_EXTA\_REG** 0x3B0U
- #define **FlexPWM0\_SM1\_EXTA\_REG** 0x3B4U
- #define **FlexPWM0\_SM2\_EXTA\_REG** 0x3B8U
- #define **FlexPWM0\_SM3\_EXTA\_REG** 0x3BCU
- #define **FlexPWM0\_EXTFORCE\_REG** 0x3C0U

- #define **FlexPWM0\_FAULT0\_REG** 0x3C4U
- #define **FlexPWM0\_FAULT1\_REG** 0x3C8U
- #define **FlexPWM0\_FAULT2\_REG** 0x3CCU
- #define **FlexPWM0\_FAULT3\_REG** 0x3D0U
- #define **FlexPWM1\_SM0\_EXTSYNC\_REG** 0x3E0U
- #define **FlexPWM1\_SM1\_EXTSYNC\_REG** 0x3E4U
- #define **FlexPWM1\_SM2\_EXTSYNC\_REG** 0x3E8U
- #define **FlexPWM1\_SM3\_EXTSYNC\_REG** 0x3ECU
- #define **FlexPWM1\_SM0\_EXTA\_REG** 0x3F0U
- #define **FlexPWM1\_SM1\_EXTA\_REG** 0x3F4U
- #define **FlexPWM1\_SM2\_EXTA\_REG** 0x3F8U
- #define **FlexPWM1\_SM3\_EXTA\_REG** 0x3FCU
- #define **FlexPWM1\_EXTFORCE\_REG** 0x400U
- #define **FlexPWM1\_FAULT0\_REG** 0x404U
- #define **FlexPWM1\_FAULT1\_REG** 0x408U
- #define **FlexPWM1\_FAULT2\_REG** 0x40CU
- #define **FlexPWM1\_FAULT3\_REG** 0x410U
- #define **PWM0\_EXT\_CLK\_REG** 0x420U
- #define **PWM1\_EXT\_CLK\_REG** 0x424U
- #define **EVTG\_TRIG0\_REG** 0x440U
- #define **USBFS\_TRIG\_REG** 0x480U
- #define **TSI\_TRIG\_REG** 0x4A0U
- #define **EXT\_TRIG0\_REG** 0x4C0U
- #define **CMP1\_TRIG\_REG** 0x4E0U
- #define **CMP2\_TRIG\_REG** 0x500U
- #define **SINC\_FILTER\_CH0\_REG** 0x520U
- #define **OPAMP0\_TRIG\_REG** 0x580U
- #define **OPAMP1\_TRIG\_REG** 0x584U
- #define **OPAMP2\_TRIG\_REG** 0x588U
- #define **FLEXCOMM0\_TRIG\_REG** 0x5A0U
- #define **FLEXCOMM1\_TRIG\_REG** 0x5C0U
- #define **FLEXCOMM2\_TRIG\_REG** 0x5E0U
- #define **FLEXCOMM3\_TRIG\_REG** 0x600U
- #define **FLEXCOMM4\_TRIG\_REG** 0x620U
- #define **FLEXCOMM5\_TRIG\_REG** 0x640U
- #define **FLEXCOMM6\_TRIG\_REG** 0x660U
- #define **FLEXCOMM7\_TRIG\_REG** 0x680U
- #define **FLEXCOMM8\_TRIG\_REG** 0x6A0U
- #define **FLEXCOMM9\_TRIG\_REG** 0x6C0U
- #define **FLEXIO\_TRIG0\_REG** 0x6E0U
- #define **DMA0\_REQ\_ENABLE0\_REG** 0x700U
- #define **DMA0\_REQ\_ENABLE1\_REG** 0x710U
- #define **DMA0\_REQ\_ENABLE2\_REG** 0x720U
- #define **DMA0\_REQ\_ENABLE3\_REG** 0x730U
- #define **DMA1\_REQ\_ENABLE0\_REG** 0x780U
- #define **DMA1\_REQ\_ENABLE1\_REG** 0x790U
- #define **DMA1\_REQ\_ENABLE2\_REG** 0x7A0U
- #define **DMA1\_REQ\_ENABLE3\_REG** 0x7B0U
- #define **ENA\_SHIFT\_8U**
- #define **PMUX\_SHIFT\_20U**

## Driver version

- #define **FSL\_INPUTMUX\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 5))  
*Group interrupt driver version for SDK.*

## 20.4 Enumeration Type Documentation

### 20.4.1 enum inputmux\_connection\_t

Enumerator

***kINPUTMUX\_Sct0In0ToSct0*** SCT0 INMUX.  
***kINPUTMUX\_Sai1RxBclkToSct0*** TIMER0 CAPTSEL.  
***kINPUTMUX\_Sai1RxSyncOutToTimer0Captsel*** TIMER1 CAPTSEL.  
***kINPUTMUX\_Sai1RxSyncOutToTimer1Captsel*** TIMER2 CAPTSEL.  
***kINPUTMUX\_Sai1RxSyncOutToTimer2Captsel*** TIMER3 CAPTSEL.  
***kINPUTMUX\_Sai1RxSyncOutToTimer3Captsel*** Timer4 CAPTSEL.  
***kINPUTMUX\_Sai1RxSyncOutToTimer4Captsel*** TIMER0 Trigger.  
***kINPUTMUX\_Sai1RxSyncOutToTimer0Trigger*** TIMER1 Trigger.  
***kINPUTMUX\_Sai1RxSyncOutToTimer1Trigger*** TIMER2 Trigger.  
***kINPUTMUX\_Sai1RxSyncOutToTimer2Trigger*** TIMER3 Trigger.  
***kINPUTMUX\_Sai1RxSyncOutToTimer3Trigger*** TIMER4 Trigger.  
***kINPUTMUX\_Sai1RxSyncOutToTimer4Trigger*** SMARTDMA arch B inputs.  
***kINPUTMUX\_Gpio3PinEventTrig1ToSmartDma*** Pin interrupt select.  
***kINPUTMUX\_GpioPort1Pin31ToPintsel*** Selection for frequency measurement reference clock.  
***kINPUTMUX\_EvtgOut1AToFreqmeasRef*** Selection for frequency measurement target clock.  
***kINPUTMUX\_EvtgOut1AToFreqmeasTar*** Cmp0 Trigger.  
***kINPUTMUX\_Gpio3PinEventTrig1ToCmp0Trigger*** Cmp1 Trigger.  
***kINPUTMUX\_Gpio3PinEventTrig1ToCmp1Trigger*** Cmp2 Trigger.  
***kINPUTMUX\_Gpio3PinEventTrig1ToCmp2Trigger*** Adc0 Trigger.  
***kINPUTMUX\_WuuToAdc0Trigger*** Adc1 Trigger.  
***kINPUTMUX\_WuuToAdc1Trigger*** Dac0 Trigger.  
***kINPUTMUX\_Gpio3PinEventTrig1ToDac0Trigger*** Dac1 Trigger.  
***kINPUTMUX\_Gpio3PinEventTrig1ToDac1Trigger*** Dac2 Trigger.  
***kINPUTMUX\_Gpio3PinEventTrig1ToDac2Trigger*** ENC0 Trigger Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc0Trigger*** ENC0 Home Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc0Home*** ENC0 Index Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc0Index*** ENC0 Phaseb Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc0Phaseb*** ENC0 Phasea Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc0Phasea*** ENC1 Trigger Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc1Trigger*** ENC1 Home Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc1Home*** ENC1 Index Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc1Index*** ENC1 Phaseb Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc1Phaseb*** ENC1 Phasea Input Connections.  
***kINPUTMUX\_TrigIn9ToEnc1Phasea*** FlexPWM0\_SM0\_EXTSYNC input trigger connections.  
***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Sm0ExtSync*** FlexPWM0\_SM1\_EXTSYNC  
input trigger connections.  
***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Sm1ExtSync*** FlexPWM0\_SM2\_EXTSYNC2  
input trigger connections.  
***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Sm2ExtSync*** FlexPWM0\_SM3\_EXTSYNC  
input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Sm3ExtSync*** FlexPWM0\_SM0\_EXTA input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Sm0Exta*** FlexPWM0\_SM1\_EXTA input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Sm1Exta*** FlexPWM0\_SM2\_EXTA input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Sm2Exta*** FlexPWM0\_SM3\_EXTA input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Sm3Exta*** FlexPWM0\_EXTFORCE input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0ExtForce*** FlexPWM0\_FAULT0 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Fault0*** FlexPWM0\_FAULT1 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Fault1*** FlexPWM0\_FAULT2 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Fault2*** FlexPWM0\_FAULT3 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm0Fault3*** FlexPWM1\_SM0\_EXTSYNC input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Sm0ExtSync*** FlexPWM1\_SM1\_EXTSYNC input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Sm1ExtSync*** FlexPWM1\_SM2\_EXTSYNC2 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Sm2ExtSync*** FlexPWM1\_SM3\_EXTSYNC input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Sm3ExtSync*** FlexPWM1\_SM0\_EXTA input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Sm0Exta*** FlexPWM1\_SM1\_EXTA input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Sm1Exta*** FlexPWM1\_SM2\_EXTA input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Sm3Exta*** FlexPWM1\_SM3\_EXTA input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Sm3ExtSync*** FlexPWM1\_EXTFORCE input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1ExtForce*** FlexPWM1\_FAULT0 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Fault0*** FlexPWM1\_FAULT1 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Fault1*** FlexPWM1\_FAULT2 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Fault2*** FlexPWM1\_FAULT3 input trigger connections.

***kINPUTMUX\_Gpio3PinEventTrig1ToFlexPwm1Fault3*** PWM0 external clock trigger.

*kINPUTMUX\_ExtrigIn1ToPwm0ExtClk* PWM1 external clock trigger.  
*kINPUTMUX\_ExtrigIn1ToPwm1ExtClk* EVTG trigger input connections.  
*kINPUTMUX\_TmprOut1ToEvtgTrigger* USB-FS trigger input connections.  
*kINPUTMUX\_Lpflexcomm9Trig3ToUsbfsTrigger* TSI trigger input connections.  
*kINPUTMUX\_Lptmr1ToTsiTrigger* EXT trigger connections.  
*kINPUTMUX\_EnetPpsOut0ToExtTrigger* SINC Filter channel trigger input connections.  
*kINPUTMUX\_WuuToSincFilterChTrigger* OPAMP0 trigger input connections.  
*kINPUTMUX\_FlexioCh7ToOpamp0Trigger* OPAMP1 trigger input connections.  
*kINPUTMUX\_FlexioCh7ToOpamp1Trigger* OPAMP2 trigger input connections.  
*kINPUTMUX\_FlexioCh7ToOpamp2Trigger* FLEXCOMM0 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm0Trigger* FLEXCOMM1 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm1Trigger* FLEXCOMM2 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm2Trigger* FLEXCOMM3 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm3Trigger* FLEXCOMM4 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm4Trigger* FLEXCOMM5 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm5Trigger* FLEXCOMM6 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm6Trigger* FLEXCOMM7 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm7Trigger* FLEXCOMM8 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm8Trigger* FLEXCOMM9 trigger input connections.  
*kINPUTMUX\_WuuToFlexcomm9Trigger* FlexIO trigger input connections.

## 20.4.2 enum inputmux\_signal\_t

Enumerator

*kINPUTMUX\_FlexSpi0RxToDma0Ch1Ena* DMA0 REQ ENABLE0 signal.  
*kINPUTMUX\_Evtg0Out0AToDma0Ch31Ena* DMA0 REQ ENABLE1 signal.  
*kINPUTMUX\_FlexIO0ShiftRegister2RequestToDma0Ch63Ena* DMA0 REQ ENABLE2 signal.  
*kINPUTMUX\_I3c0RxToDma0Ch95Ena* DMA0 REQ ENABLE3 signal.  
*kINPUTMUX\_Tsi0OutOfRangeToDma0Ch121Ena* DMA1 REQ ENABLE0 signal.  
*kINPUTMUX\_Evtg0Out0AToDma1Ch31Ena* DMA1 REQ ENABLE1 signal.  
*kINPUTMUX\_FlexIO0ShiftRegister2RequestToDma1Ch63Ena* DMA1 REQ ENABLE2 signal.  
*kINPUTMUX\_I3c0RxToDma1Ch95Ena* DMA1 REQ ENABLE3 signal.

## 20.5 Function Documentation

### 20.5.1 void INPUTMUX\_Init( INPUTMUX\_Type \* *base* )

This function enables the INPUTMUX clock.

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | Base address of the INPUTMUX peripheral. |
|-------------|------------------------------------------|

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

### 20.5.2 void INPUTMUX\_AttachSignal ( INPUTMUX\_Type \* *base*, uint32\_t *index*, inputmux\_connection\_t *connection* )

This function gates the INPUTPMUX clock.

Parameters

|                   |                                                 |
|-------------------|-------------------------------------------------|
| <i>base</i>       | Base address of the INPUTMUX peripheral.        |
| <i>index</i>      | Destination peripheral to attach the signal to. |
| <i>connection</i> | Selects connection.                             |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

### 20.5.3 void INPUTMUX\_Deinit ( INPUTMUX\_Type \* *base* )

This function disables the INPUTMUX clock.

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | Base address of the INPUTMUX peripheral. |
|-------------|------------------------------------------|

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

# Chapter 21

## LPADC: 12-bit SAR Analog-to-Digital Converter Driver

### 21.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit SAR Analog-to-Digital Converter (LP-ADC) module of MCUXpresso SDK devices.

### 21.2 Typical use case

#### 21.2.1 Polling Configuration

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/lpadc

#### 21.2.2 Interrupt Configuration

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/lpadc

### Files

- file [fsl\\_lpadc.h](#)

### Data Structures

- struct [lpadc\\_config\\_t](#)  
*LPADC global configuration. [More...](#)*
- struct [lpadc\\_conv\\_command\\_config\\_t](#)  
*Define structure to keep the configuration for conversion command. [More...](#)*
- struct [lpadc\\_conv\\_trigger\\_config\\_t](#)  
*Define structure to keep the configuration for conversion trigger. [More...](#)*
- struct [lpadc\\_conv\\_result\\_t](#)  
*Define the structure to keep the conversion result. [More...](#)*

### Macros

- #define [LPADC\\_GET\\_ACTIVE\\_COMMAND\\_STATUS](#)(statusVal) ((statusVal & ADC\_STAT\_CMDACT\_MASK) >> ADC\_STAT\_CMDACT\_SHIFT)  
*Define the MACRO function to get command status from status value.*
- #define [LPADC\\_GET\\_ACTIVE\\_TRIGGER\\_STATUS](#)(statusVal) ((statusVal & ADC\_STAT\_TRGACT\_MASK) >> ADC\_STAT\_TRGACT\_SHIFT)  
*Define the MACRO function to get trigger status from status value.*

## Enumerations

- enum `_lpadc_status_flags` {
   
  `kLPADC_ResultFIFO0OverflowFlag` = ADC\_STAT\_FOF0\_MASK,
   
  `kLPADC_ResultFIFO0ReadyFlag` = ADC\_STAT\_RDY0\_MASK,
   
  `kLPADC_ResultFIFO1OverflowFlag` = ADC\_STAT\_FOF1\_MASK,
   
  `kLPADC_ResultFIFO1ReadyFlag` = ADC\_STAT\_RDY1\_MASK }
   
    *Define hardware flags of the module.*
- enum `_lpadc_interrupt_enable` {
   
  `kLPADC_ResultFIFO0OverflowInterruptEnable` = ADC\_IE\_FOFIE0\_MASK,
   
  `kLPADC_FIFO0WatermarkInterruptEnable` = ADC\_IE\_FWMIE0\_MASK,
   
  `kLPADC_ResultFIFO1OverflowInterruptEnable` = ADC\_IE\_FOFIE1\_MASK,
   
  `kLPADC_FIFO1WatermarkInterruptEnable` = ADC\_IE\_FWMIE1\_MASK,
   
  `kLPADC_TriggerExceptionInterruptEnable` = ADC\_IE\_TEXC\_IE\_MASK,
   
  `kLPADC_Trigger0CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 0UL),
   
  `kLPADC_Trigger1CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 1UL),
   
  `kLPADC_Trigger2CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 2UL),
   
  `kLPADC_Trigger3CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 3UL),
   
  `kLPADC_Trigger4CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 4UL),
   
  `kLPADC_Trigger5CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 5UL),
   
  `kLPADC_Trigger6CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 6UL),
   
  `kLPADC_Trigger7CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 7UL),
   
  `kLPADC_Trigger8CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 8UL),
   
  `kLPADC_Trigger9CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 9UL),
   
  `kLPADC_Trigger10CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 10UL),
   
  `kLPADC_Trigger11CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 11UL),
   
  `kLPADC_Trigger12CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 12UL),
   
  `kLPADC_Trigger13CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 13UL),
   
  `kLPADC_Trigger14CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 14UL),
   
  `kLPADC_Trigger15CompletionInterruptEnable` = ADC\_IE\_TCOMP\_IE(1UL << 15UL) }
   
    *Define interrupt switchers of the module.*
- enum `_lpadc_trigger_status_flags` {

```

kLPADC_Trigger0InterruptedFlag = 1UL << 0UL,
kLPADC_Trigger1InterruptedFlag = 1UL << 1UL,
kLPADC_Trigger2InterruptedFlag = 1UL << 2UL,
kLPADC_Trigger3InterruptedFlag = 1UL << 3UL,
kLPADC_Trigger4InterruptedFlag = 1UL << 4UL,
kLPADC_Trigger5InterruptedFlag = 1UL << 5UL,
kLPADC_Trigger6InterruptedFlag = 1UL << 6UL,
kLPADC_Trigger7InterruptedFlag = 1UL << 7UL,
kLPADC_Trigger8InterruptedFlag = 1UL << 8UL,
kLPADC_Trigger9InterruptedFlag = 1UL << 9UL,
kLPADC_Trigger10InterruptedFlag = 1UL << 10UL,
kLPADC_Trigger11InterruptedFlag = 1UL << 11UL,
kLPADC_Trigger12InterruptedFlag = 1UL << 12UL,
kLPADC_Trigger13InterruptedFlag = 1UL << 13UL,
kLPADC_Trigger14InterruptedFlag = 1UL << 14UL,
kLPADC_Trigger15InterruptedFlag = 1UL << 15UL,
kLPADC_Trigger0CompletedFlag = 1UL << 16UL,
kLPADC_Trigger1CompletedFlag = 1UL << 17UL,
kLPADC_Trigger2CompletedFlag = 1UL << 18UL,
kLPADC_Trigger3CompletedFlag = 1UL << 19UL,
kLPADC_Trigger4CompletedFlag = 1UL << 20UL,
kLPADC_Trigger5CompletedFlag = 1UL << 21UL,
kLPADC_Trigger6CompletedFlag = 1UL << 22UL,
kLPADC_Trigger7CompletedFlag = 1UL << 23UL,
kLPADC_Trigger8CompletedFlag = 1UL << 24UL,
kLPADC_Trigger9CompletedFlag = 1UL << 25UL,
kLPADC_Trigger10CompletedFlag = 1UL << 26UL,
kLPADC_Trigger11CompletedFlag = 1UL << 27UL,
kLPADC_Trigger12CompletedFlag = 1UL << 28UL,
kLPADC_Trigger13CompletedFlag = 1UL << 29UL,
kLPADC_Trigger14CompletedFlag = 1UL << 30UL,
kLPADC_Trigger15CompletedFlag = 1UL << 31UL }

```

*The enumerator of lpadc trigger status flags, including interrupted flags and completed flags.*

- enum `lpadc_sample_scale_mode_t` {
   
    `kLPADC_SamplePartScale,`
  
    `kLPADC_SampleFullScale = 1U }`

*Define enumeration of sample scale mode.*
- enum `lpadc_sample_channel_mode_t` {
   
    `kLPADC_SampleChannelSingleEndSideA = 0U,`
  
    `kLPADC_SampleChannelSingleEndSideB = 1U,`
  
    `kLPADC_SampleChannelDiffBothSide = 2U,`
  
    `kLPADC_SampleChannelDualSingleEndBothSide }`

*Define enumeration of channel sample mode.*
- enum `lpadc_hardware_average_mode_t` {

```
kLPADC_HardwareAverageCount1 = 0U,
kLPADC_HardwareAverageCount2 = 1U,
kLPADC_HardwareAverageCount4 = 2U,
kLPADC_HardwareAverageCount8 = 3U,
kLPADC_HardwareAverageCount16 = 4U,
kLPADC_HardwareAverageCount32 = 5U,
kLPADC_HardwareAverageCount64 = 6U,
kLPADC_HardwareAverageCount128 = 7U,
kLPADC_HardwareAverageCount256 = 8U,
kLPADC_HardwareAverageCount512 = 9U,
kLPADC_HardwareAverageCount1024 = 10U }
```

*Define enumeration of hardware average selection.*

- enum `lpadc_sample_time_mode_t` {
 

```
kLPADC_SampleTimeADCK3 = 0U,
kLPADC_SampleTimeADCK5 = 1U,
kLPADC_SampleTimeADCK7 = 2U,
kLPADC_SampleTimeADCK11 = 3U,
kLPADC_SampleTimeADCK19 = 4U,
kLPADC_SampleTimeADCK35 = 5U,
kLPADC_SampleTimeADCK67 = 6U,
kLPADC_SampleTimeADCK131 = 7U }
```

*Define enumeration of sample time selection.*

- enum `lpadc_hardware_compare_mode_t` {
 

```
kLPADC_HardwareCompareDisabled = 0U,
kLPADC_HardwareCompareStoreOnTrue = 2U,
kLPADC_HardwareCompareRepeatUntilTrue = 3U }
```

*Define enumeration of hardware compare mode.*

- enum `lpadc_conversion_resolution_mode_t` {
 

```
kLPADC_ConversionResolutionStandard = 0U,
kLPADC_ConversionResolutionHigh = 1U }
```

*Define enumeration of conversion resolution mode.*

- enum `lpadc_conversion_average_mode_t` {
 

```
kLPADC_ConversionAverage1 = 0U,
kLPADC_ConversionAverage2 = 1U,
kLPADC_ConversionAverage4 = 2U,
kLPADC_ConversionAverage8 = 3U,
kLPADC_ConversionAverage16 = 4U,
kLPADC_ConversionAverage32 = 5U,
kLPADC_ConversionAverage64 = 6U,
kLPADC_ConversionAverage128 = 7U,
kLPADC_ConversionAverage256 = 8U,
kLPADC_ConversionAverage512 = 9U,
kLPADC_ConversionAverage1024 = 10U }
```

*Define enumeration of conversion averages mode.*

- enum `lpadc_reference_voltage_source_t` {

```

kLPADC_ReferenceVoltageAlt1 = 0U,
kLPADC_ReferenceVoltageAlt2 = 1U,
kLPADC_ReferenceVoltageAlt3 = 2U }

Define enumeration of reference voltage source.
• enum lpadc_power_level_mode_t {
    kLPADC_PowerLevelAlt1 = 0U,
    kLPADC_PowerLevelAlt2 = 1U,
    kLPADC_PowerLevelAlt3 = 2U,
    kLPADC_PowerLevelAlt4 = 3U }

Define enumeration of power configuration.
• enum lpadc_trigger_priority_policy_t {
    kLPADC_TriggerPriorityPreemptImmediately = 0U,
    kLPADC_TriggerPriorityPreemptSoftly = 1U,
    kLPADC_TriggerPriorityPreemptSubsequently = 2U }

Define enumeration of trigger priority policy.

```

## Driver version

- #define FSL\_LPADC\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))  
*LPADC driver version 2.6.0.*

## Initialization & de-initialization.

- void LPADC\_Init (ADC\_Type \*base, const lpadc\_config\_t \*config)  
*Initializes the LPADC module.*
- void LPADC\_GetDefaultConfig (lpadc\_config\_t \*config)  
*Gets an available pre-defined settings for initial configuration.*
- void LPADC\_Deinit (ADC\_Type \*base)  
*De-initializes the LPADC module.*
- static void LPADC\_Enable (ADC\_Type \*base, bool enable)  
*Switch on/off the LPADC module.*
- static void LPADC\_DoResetFIFO0 (ADC\_Type \*base)  
*Do reset the conversion FIFO0.*
- static void LPADC\_DoResetFIFO1 (ADC\_Type \*base)  
*Do reset the conversion FIFO1.*
- static void LPADC\_DoResetConfig (ADC\_Type \*base)  
*Do reset the module's configuration.*

## Status

- static uint32\_t LPADC\_GetStatusFlags (ADC\_Type \*base)  
*Get status flags.*
- static void LPADC\_ClearStatusFlags (ADC\_Type \*base, uint32\_t mask)  
*Clear status flags.*
- static uint32\_t LPADC\_GetTriggerStatusFlags (ADC\_Type \*base)  
*Get trigger status flags to indicate which trigger sequences have been completed or interrupted by a high priority trigger exception.*
- static void LPADC\_ClearTriggerStatusFlags (ADC\_Type \*base, uint32\_t mask)  
*Clear trigger status flags.*

## Interrupts

- static void **LPADC\_EnableInterrupts** (ADC\_Type \*base, uint32\_t mask)  
*Enable interrupts.*
- static void **LPADC\_DisableInterrupts** (ADC\_Type \*base, uint32\_t mask)  
*Disable interrupts.*

## DMA Control

- static void **LPADC\_EnableFIFO0WatermarkDMA** (ADC\_Type \*base, bool enable)  
*Switch on/off the DMA trigger for FIFO0 watermark event.*
- static void **LPADC\_EnableFIFO1WatermarkDMA** (ADC\_Type \*base, bool enable)  
*Switch on/off the DMA trigger for FIFO1 watermark event.*

## Trigger and conversion with FIFO.

- static uint32\_t **LPADC\_GetConvResultCount** (ADC\_Type \*base, uint8\_t index)  
*Get the count of result kept in conversion FIFO.*
- bool **LPADC\_GetConvResult** (ADC\_Type \*base, lpadc\_conv\_result\_t \*result, uint8\_t index)  
*brief Get the result in conversion FIFO.*
- void **LPADC\_SetConvTriggerConfig** (ADC\_Type \*base, uint32\_t triggerId, const lpadc\_conv\_trigger\_config\_t \*config)  
*Configure the conversion trigger source.*
- void **LPADC\_GetDefaultConvTriggerConfig** (lpadc\_conv\_trigger\_config\_t \*config)  
*Gets an available pre-defined settings for trigger's configuration.*
- static void **LPADC\_DoSoftwareTrigger** (ADC\_Type \*base, uint32\_t triggerIdMask)  
*Do software trigger to conversion command.*
- void **LPADC\_SetConvCommandConfig** (ADC\_Type \*base, uint32\_t commandId, const lpadc\_conv\_command\_config\_t \*config)  
*Configure conversion command.*
- void **LPADC\_GetDefaultConvCommandConfig** (lpadc\_conv\_command\_config\_t \*config)  
*Gets an available pre-defined settings for conversion command's configuration.*
- static void **LPADC\_SetOffsetValue** (ADC\_Type \*base, uint32\_t valueA, uint32\_t valueB)  
*Set proper offset value to trim ADC.*
- static void **LPADC\_EnableOffsetCalibration** (ADC\_Type \*base, bool enable)  
*Enable the offset calibration function.*
- void **LPADC\_DoOffsetCalibration** (ADC\_Type \*base)  
*Do offset calibration.*
- void **LPADC\_DoAutoCalibration** (ADC\_Type \*base)  
*brief Do auto calibration.*

## 21.3 Data Structure Documentation

### 21.3.1 struct lpadc\_config\_t

This structure would used to keep the settings for initialization.

## Data Fields

- bool **enableInDozeMode**

- *Control system transition to Stop and Wait power modes while ADC is converting.*
- **lpadc\_conversion\_average\_mode\_t conversionAverageMode**  
*Auto-Calibration Averages.*
- **bool enableAnalogPreliminary**  
*ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).*
- **uint32\_t powerUpDelay**  
*When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize.*
- **lpadc\_reference\_voltage\_source\_t referenceVoltageSource**  
*Selects the voltage reference high used for conversions.*
- **lpadc\_power\_level\_mode\_t powerLevelMode**  
*Power Configuration Selection.*
- **lpadc\_trigger\_priority\_policy\_t triggerPriorityPolicy**  
*Control how higher priority triggers are handled, see to lpadc\_trigger\_priority\_policy\_t.*
- **bool enableConvPause**  
*Enables the ADC pausing function.*
- **uint32\_t convPauseDelay**  
*Controls the duration of pausing during command execution sequencing.*
- **uint32\_t FIFO0Watermark**  
*FIFO0Watermark is a programmable threshold setting.*
- **uint32\_t FIFO1Watermark**  
*FIFO1Watermark is a programmable threshold setting.*

## Field Documentation

(1) **bool lpadc\_config\_t::enableInDozeMode**

When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

(2) **lpadc\_conversion\_average\_mode\_t lpadc\_config\_t::conversionAverageMode**

(3) **bool lpadc\_config\_t::enableAnalogPreliminary**

(4) **uint32\_t lpadc\_config\_t::powerUpDelay**

The startup delay count of (powerUpDelay \* 4) ADCK cycles must result in a longer delay than the analog startup time.

(5) **lpadc\_reference\_voltage\_source\_t lpadc\_config\_t::referenceVoltageSource**

(6) **lpadc\_power\_level\_mode\_t lpadc\_config\_t::powerLevelMode**

(7) **lpadc\_trigger\_priority\_policy\_t lpadc\_config\_t::triggerPriorityPolicy**

**(8) bool lpadc\_config\_t::enableConvPause**

When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in "Compare Until True" configuration.

**(9) uint32\_t lpadc\_config\_t::convPauseDelay**

The pause delay is a count of (convPauseDelay\*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

**(10) uint32\_t lpadc\_config\_t::FIFO0Watermark**

When the number of datawords stored in the ADC Result FIFO0 is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

**(11) uint32\_t lpadc\_config\_t::FIFO1Watermark**

When the number of datawords stored in the ADC Result FIFO1 is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

### 21.3.2 struct lpadc\_conv\_command\_config\_t

#### Data Fields

- [lpadc\\_sample\\_channel\\_mode\\_t sampleChannelMode](#)  
*Channel sample mode.*
- [uint32\\_t channelNumber](#)  
*Channel number, select the channel or channel pair.*
- [uint32\\_t channelBNumber](#)  
*Alternate Channel B number, select the channel.*
- [uint32\\_t chainedNextCommandNumber](#)  
*Selects the next command to be executed after this command completes.*
- [bool enableAutoChannelIncrement](#)  
*Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.*
- [uint32\\_t loopCount](#)  
*Selects how many times this command executes before finish and transition to the next command or Idle state.*
- [lpadc\\_hardware\\_average\\_mode\\_t hardwareAverageMode](#)  
*Hardware average selection.*
- [lpadc\\_sample\\_time\\_mode\\_t sampleTimeMode](#)  
*Sample time selection.*
- [lpadc\\_hardware\\_compare\\_mode\\_t hardwareCompareMode](#)  
*Hardware compare selection.*
- [uint32\\_t hardwareCompareValueHigh](#)  
*Compare Value High.*

- `uint32_t hardwareCompareValueLow`  
*Compare Value Low.*
- `lpadc_conversion_resolution_mode_t conversionResolutionMode`  
*Conversion resolution mode.*
- `bool enableWaitTrigger`  
*Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.*

**Field Documentation**(1) `lpadc_sample_channel_mode_t lpadc_conv_command_config_t::sampleChannelMode`(2) `uint32_t lpadc_conv_command_config_t::channelNumber`(3) `uint32_t lpadc_conv_command_config_t::channelBNumber`(4) `uint32_t lpadc_conv_command_config_t::chainedNextCommandNumber`

1-15 is available, 0 is to terminate the chain after this command.

(5) `bool lpadc_conv_command_config_t::enableAutoChannelIncrement`(6) `uint32_t lpadc_conv_command_config_t::loopCount`

Command executes LOOP+1 times. 0-15 is available.

(7) `lpadc_hardware_average_mode_t lpadc_conv_command_config_t::hardwareAverageMode`(8) `lpadc_sample_time_mode_t lpadc_conv_command_config_t::sampleTimeMode`(9) `lpadc_hardware_compare_mode_t lpadc_conv_command_config_t::hardwareCompareMode`(10) `uint32_t lpadc_conv_command_config_t::hardwareCompareValueHigh`

The available value range is in 16-bit.

(11) `uint32_t lpadc_conv_command_config_t::hardwareCompareValueLow`

The available value range is in 16-bit.

(12) `lpadc_conversion_resolution_mode_t lpadc_conv_command_config_t::conversion-ResolutionMode`(13) `bool lpadc_conv_command_config_t::enableWaitTrigger`**21.3.3 struct lpadc\_conv\_trigger\_config\_t****Data Fields**

- `uint32_t targetCommandId`

- **uint32\_t delayPower**  
*Select the trigger delay duration to wait at the start of servicing a trigger event.*
- **uint32\_t priority**  
*Sets the priority of the associated trigger source.*
- **bool enableHardwareTrigger**  
*Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not.*

### Field Documentation

(1) **uint32\_t lpadc\_conv\_trigger\_config\_t::targetCommandId**

(2) **uint32\_t lpadc\_conv\_trigger\_config\_t::delayPower**

When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is  $2^{\text{delayPower}}$  ADCK cycles. The available value range is 4-bit.

(3) **uint32\_t lpadc\_conv\_trigger\_config\_t::priority**

If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

(4) **bool lpadc\_conv\_trigger\_config\_t::enableHardwareTrigger**

The software trigger is always available.

### 21.3.4 struct lpadc\_conv\_result\_t

#### Data Fields

- **uint32\_t commandIdSource**  
*Indicate the command buffer being executed that generated this result.*
- **uint32\_t loopCountIndex**  
*Indicate the loop count value during command execution that generated this result.*
- **uint32\_t triggerIdSource**  
*Indicate the trigger source that initiated a conversion and generated this result.*
- **uint16\_t convValue**  
*Data result.*

### Field Documentation

(1) **uint32\_t lpadc\_conv\_result\_t::commandIdSource**

(2) **uint32\_t lpadc\_conv\_result\_t::loopCountIndex**

(3) **uint32\_t lpadc\_conv\_result\_t::triggerIdSource**

(4) **uint16\_t lpadc\_conv\_result\_t::convValue**

## 21.4 Macro Definition Documentation

**21.4.1 #define FSL\_LPADC\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))**

**21.4.2 #define LPADC\_GET\_ACTIVE\_COMMAND\_STATUS( *statusVal* ) ((*statusVal* & ADC\_STAT\_CMDACT\_MASK) >> ADC\_STAT\_CMDACT\_SHIFT)**

The *statusVal* is the return value from [LPADC\\_GetStatusFlags\(\)](#).

**21.4.3 #define LPADC\_GET\_ACTIVE\_TRIGGER\_STATUS( *statusVal* ) ((*statusVal* & ADC\_STAT\_TRGACT\_MASK) >> ADC\_STAT\_TRGACT\_SHIFT)**

The *statusVal* is the return value from [LPADC\\_GetStatusFlags\(\)](#).

## 21.5 Enumeration Type Documentation

### 21.5.1 enum \_lpadc\_status\_flags

Enumerator

***kLPADC\_ResultFIFO0OverflowFlag*** Indicates that more data has been written to the Result FIFO 0 than it can hold.

***kLPADC\_ResultFIFO0ReadyFlag*** Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

***kLPADC\_ResultFIFO1OverflowFlag*** Indicates that more data has been written to the Result FIFO 1 than it can hold.

***kLPADC\_ResultFIFO1ReadyFlag*** Indicates when the number of valid datawords in the result FIFO 1 is greater than the setting watermark level.

### 21.5.2 enum \_lpadc\_interrupt\_enable

Enumerator

***kLPADC\_ResultFIFO0OverflowInterruptEnable*** Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

***kLPADC\_FIFO0WatermarkInterruptEnable*** Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

***kLPADC\_ResultFIFO1OverflowInterruptEnable*** Configures ADC to generate overflow interrupt requests when FOF1 flag is asserted.

***kLPADC\_FIFO1WatermarkInterruptEnable*** Configures ADC to generate watermark interrupt requests when RDY1 flag is asserted.

***kLPADC\_TriggerExceptionInterruptEnable*** Configures ADC to generate trigger exception interrupt.

- kLPADC\_Trigger0CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 0 completion.
- kLPADC\_Trigger1CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 1 completion.
- kLPADC\_Trigger2CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 2 completion.
- kLPADC\_Trigger3CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 3 completion.
- kLPADC\_Trigger4CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 4 completion.
- kLPADC\_Trigger5CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 5 completion.
- kLPADC\_Trigger6CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 6 completion.
- kLPADC\_Trigger7CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 7 completion.
- kLPADC\_Trigger8CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 8 completion.
- kLPADC\_Trigger9CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 9 completion.
- kLPADC\_Trigger10CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 10 completion.
- kLPADC\_Trigger11CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 11 completion.
- kLPADC\_Trigger12CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 12 completion.
- kLPADC\_Trigger13CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 13 completion.
- kLPADC\_Trigger14CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 14 completion.
- kLPADC\_Trigger15CompletionInterruptEnable*** Configures ADC to generate interrupt when trigger 15 completion.

### 21.5.3 enum \_lpadc\_trigger\_status\_flags

Enumerator

- kLPADC\_Trigger0InterruptedFlag*** Trigger 0 is interrupted by a high priority exception.
- kLPADC\_Trigger1InterruptedFlag*** Trigger 1 is interrupted by a high priority exception.
- kLPADC\_Trigger2InterruptedFlag*** Trigger 2 is interrupted by a high priority exception.
- kLPADC\_Trigger3InterruptedFlag*** Trigger 3 is interrupted by a high priority exception.
- kLPADC\_Trigger4InterruptedFlag*** Trigger 4 is interrupted by a high priority exception.
- kLPADC\_Trigger5InterruptedFlag*** Trigger 5 is interrupted by a high priority exception.
- kLPADC\_Trigger6InterruptedFlag*** Trigger 6 is interrupted by a high priority exception.

|                                        |                                                                           |
|----------------------------------------|---------------------------------------------------------------------------|
| <i>kLPADC_Trigger7InterruptedFlag</i>  | Trigger 7 is interrupted by a high priority exception.                    |
| <i>kLPADC_Trigger8InterruptedFlag</i>  | Trigger 8 is interrupted by a high priority exception.                    |
| <i>kLPADC_Trigger9InterruptedFlag</i>  | Trigger 9 is interrupted by a high priority exception.                    |
| <i>kLPADC_Trigger10InterruptedFlag</i> | Trigger 10 is interrupted by a high priority exception.                   |
| <i>kLPADC_Trigger11InterruptedFlag</i> | Trigger 11 is interrupted by a high priority exception.                   |
| <i>kLPADC_Trigger12InterruptedFlag</i> | Trigger 12 is interrupted by a high priority exception.                   |
| <i>kLPADC_Trigger13InterruptedFlag</i> | Trigger 13 is interrupted by a high priority exception.                   |
| <i>kLPADC_Trigger14InterruptedFlag</i> | Trigger 14 is interrupted by a high priority exception.                   |
| <i>kLPADC_Trigger15InterruptedFlag</i> | Trigger 15 is interrupted by a high priority exception.                   |
| <i>kLPADC_Trigger0CompletedFlag</i>    | Trigger 0 is completed and trigger 0 has enabled completion interrupts.   |
| <i>kLPADC_Trigger1CompletedFlag</i>    | Trigger 1 is completed and trigger 1 has enabled completion interrupts.   |
| <i>kLPADC_Trigger2CompletedFlag</i>    | Trigger 2 is completed and trigger 2 has enabled completion interrupts.   |
| <i>kLPADC_Trigger3CompletedFlag</i>    | Trigger 3 is completed and trigger 3 has enabled completion interrupts.   |
| <i>kLPADC_Trigger4CompletedFlag</i>    | Trigger 4 is completed and trigger 4 has enabled completion interrupts.   |
| <i>kLPADC_Trigger5CompletedFlag</i>    | Trigger 5 is completed and trigger 5 has enabled completion interrupts.   |
| <i>kLPADC_Trigger6CompletedFlag</i>    | Trigger 6 is completed and trigger 6 has enabled completion interrupts.   |
| <i>kLPADC_Trigger7CompletedFlag</i>    | Trigger 7 is completed and trigger 7 has enabled completion interrupts.   |
| <i>kLPADC_Trigger8CompletedFlag</i>    | Trigger 8 is completed and trigger 8 has enabled completion interrupts.   |
| <i>kLPADC_Trigger9CompletedFlag</i>    | Trigger 9 is completed and trigger 9 has enabled completion interrupts.   |
| <i>kLPADC_Trigger10CompletedFlag</i>   | Trigger 10 is completed and trigger 10 has enabled completion interrupts. |
| <i>kLPADC_Trigger11CompletedFlag</i>   | Trigger 11 is completed and trigger 11 has enabled completion interrupts. |
| <i>kLPADC_Trigger12CompletedFlag</i>   | Trigger 12 is completed and trigger 12 has enabled completion interrupts. |
| <i>kLPADC_Trigger13CompletedFlag</i>   | Trigger 13 is completed and trigger 13 has enabled completion interrupts. |
| <i>kLPADC_Trigger14CompletedFlag</i>   | Trigger 14 is completed and trigger 14 has enabled completion interrupts. |
| <i>kLPADC_Trigger15CompletedFlag</i>   | Trigger 15 is completed and trigger 15 has enabled completion interrupts. |

### 21.5.4 enum lpadc\_sample\_scale\_mode\_t

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results in a voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Enumerator

- kLPADC\_SamplePartScale*** Use divided input voltage signal. (For scale select, please refer to the reference manual).
- kLPADC\_SampleFullScale*** Full scale (Factor of 1).

### 21.5.5 enum lpadc\_sample\_channel\_mode\_t

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Enumerator

- kLPADC\_SampleChannelSingleEndSideA*** Single end mode, using side A.
- kLPADC\_SampleChannelSingleEndSideB*** Single end mode, using side B.
- kLPADC\_SampleChannelDiffBothSide*** Differential mode, using A and B.
- kLPADC\_SampleChannelDualSingleEndBothSide*** Dual-Single-Ended Mode. Both A side and B side channels are converted independently.

### 21.5.6 enum lpadc\_hardware\_average\_mode\_t

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Enumerator

- kLPADC\_HardwareAverageCount1*** Single conversion.
- kLPADC\_HardwareAverageCount2*** 2 conversions averaged.
- kLPADC\_HardwareAverageCount4*** 4 conversions averaged.
- kLPADC\_HardwareAverageCount8*** 8 conversions averaged.
- kLPADC\_HardwareAverageCount16*** 16 conversions averaged.
- kLPADC\_HardwareAverageCount32*** 32 conversions averaged.
- kLPADC\_HardwareAverageCount64*** 64 conversions averaged.
- kLPADC\_HardwareAverageCount128*** 128 conversions averaged.
- kLPADC\_HardwareAverageCount256*** 256 conversions averaged.
- kLPADC\_HardwareAverageCount512*** 512 conversions averaged.
- kLPADC\_HardwareAverageCount1024*** 1024 conversions averaged.

### 21.5.7 enum lpadc\_sample\_time\_mode\_t

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Enumerator

- kLPADC\_SampleTimeADCK3* 3 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK5* 5 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK7* 7 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK11* 11 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK19* 19 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK35* 35 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK67* 69 ADCK cycles total sample time.
- kLPADC\_SampleTimeADCK131* 131 ADCK cycles total sample time.

### 21.5.8 enum lpadc\_hardware\_compare\_mode\_t

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Enumerator

- kLPADC\_HardwareCompareDisabled* Compare disabled.
- kLPADC\_HardwareCompareStoreOnTrue* Compare enabled. Store on true.
- kLPADC\_HardwareCompareRepeatUntilTrue* Compare enabled. Repeat channel acquisition until true.

### 21.5.9 enum lpadc\_conversion\_resolution\_mode\_t

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to [lpadc\\_sample\\_channel\\_mode\\_t](#)

Enumerator

- kLPADC\_ConversionResolutionStandard* Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.
- kLPADC\_ConversionResolutionHigh* High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

**21.5.10 enum lpadc\_conversion\_average\_mode\_t**

Configure the conversion average number for auto-calibration.

Enumerator

- kLPADC\_ConversionAverage1*** Single conversion.
- kLPADC\_ConversionAverage2*** 2 conversions averaged.
- kLPADC\_ConversionAverage4*** 4 conversions averaged.
- kLPADC\_ConversionAverage8*** 8 conversions averaged.
- kLPADC\_ConversionAverage16*** 16 conversions averaged.
- kLPADC\_ConversionAverage32*** 32 conversions averaged.
- kLPADC\_ConversionAverage64*** 64 conversions averaged.
- kLPADC\_ConversionAverage128*** 128 conversions averaged.
- kLPADC\_ConversionAverage256*** 256 conversions averaged.
- kLPADC\_ConversionAverage512*** 512 conversions averaged.
- kLPADC\_ConversionAverage1024*** 1024 conversions averaged.

**21.5.11 enum lpadc\_reference\_voltage\_source\_t**

For detail information, need to check the SoC's specification.

Enumerator

- kLPADC\_ReferenceVoltageAlt1*** Option 1 setting.
- kLPADC\_ReferenceVoltageAlt2*** Option 2 setting.
- kLPADC\_ReferenceVoltageAlt3*** Option 3 setting.

**21.5.12 enum lpadc\_power\_level\_mode\_t**

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Enumerator

- kLPADC\_PowerLevelAlt1*** Lowest power setting.
- kLPADC\_PowerLevelAlt2*** Next lowest power setting.
- kLPADC\_PowerLevelAlt3*** ...
- kLPADC\_PowerLevelAlt4*** Highest power setting.

### 21.5.13 enum lpadc\_trigger\_priority\_policy\_t

This selection controls how higher priority triggers are handled.

Enumerator

**kLPADC\_TriggerPriorityPreemptImmediately** If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started.

**kLPADC\_TriggerPriorityPreemptSoftly** If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated.

**kLPADC\_TriggerPriorityPreemptSubsequently** If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger.

## 21.6 Function Documentation

### 21.6.1 void LPADC\_Init ( ADC\_Type \* *base*, const lpadc\_config\_t \* *config* )

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | LPADC peripheral base address.                            |
| <i>config</i> | Pointer to configuration structure. See "lpadc_config_t". |

### 21.6.2 void LPADC\_GetDefaultConfig ( lpadc\_config\_t \* *config* )

This function initializes the converter configuration structure with an available settings. The default values are:

```
* config->enableInDozeMode      = true;
* config->enableAnalogPreliminary = false;
* config->powerUpDelay        = 0x80;
* config->referenceVoltageSource = kLPADC_ReferenceVoltageAlt1;
* config->powerLevelMode       = kLPADC_PowerLevelAlt1;
* config->triggerPriorityPolicy = kLPADC_TriggerPriorityPreemptImmediately
;
* config->enableConvPause      = false;
* config->convPauseDelay       = 0U;
* config->FIFOWatermark       = 0U;
*
```

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>config</i> | Pointer to configuration structure. |
|---------------|-------------------------------------|

### 21.6.3 void LPADC\_Deinit ( ADC\_Type \* *base* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | LPADC peripheral base address. |
|-------------|--------------------------------|

### 21.6.4 static void LPADC\_Enable ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>enable</i> | switcher to the module.        |

### 21.6.5 static void LPADC\_DoResetFIFO0 ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | LPADC peripheral base address. |
|-------------|--------------------------------|

### 21.6.6 static void LPADC\_DoResetFIFO1 ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | LPADC peripheral base address. |
|-------------|--------------------------------|

**21.6.7 static void LPADC\_DoResetConfig ( ADC\_Type \* *base* ) [inline],  
[static]**

Reset all ADC internal logic and registers, except the Control Register (ADCx\_CTRL).

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | LPADC peripheral base address. |
|-------------|--------------------------------|

### 21.6.8 static uint32\_t LPADC\_GetStatusFlags ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | LPADC peripheral base address. |
|-------------|--------------------------------|

Returns

status flags' mask. See to [\\_lpadc\\_status\\_flags](#).

### 21.6.9 static void LPADC\_ClearStatusFlags ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Only the flags can be cleared by writing ADCx\_STATUS register would be cleared by this API.

Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                   |
| <i>mask</i> | Mask value for flags to be cleared. See to <a href="#">_lpadc_status_flags</a> . |

### 21.6.10 static uint32\_t LPADC\_GetTriggerStatusFlags ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | LPADC peripheral base address. |
|-------------|--------------------------------|

Returns

The OR'ed value of [\\_lpadc\\_trigger\\_status\\_flags](#).

### 21.6.11 static void LPADC\_ClearTriggerStatusFlags ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                                                             |
| <i>mask</i> | The mask of trigger status flags to be cleared, should be the OR'ed value of <a href="#">_lpadc_trigger_status_flags</a> . |

**21.6.12 static void LPADC\_EnableInterrupts ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                    |
| <i>mask</i> | Mask value for interrupt events. See to <a href="#">_lpadc_interrupt_enable</a> . |

**21.6.13 static void LPADC\_DisableInterrupts ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                    |
| <i>mask</i> | Mask value for interrupt events. See to <a href="#">_lpadc_interrupt_enable</a> . |

**21.6.14 static void LPADC\_EnableFIFO0WatermarkDMA ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>enable</i> | Switcher to the event.         |

**21.6.15 static void LPADC\_EnableFIFO1WatermarkDMA ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>enable</i> | Switcher to the event.         |

### 21.6.16 static uint32\_t LPADC\_GetConvResultCount ( ADC\_Type \* *base*, uint8\_t *index* ) [inline], [static]

Parameters

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | LPADC peripheral base address. |
| <i>index</i> | Result FIFO index.             |

Returns

The count of result kept in conversion FIFO.

### 21.6.17 bool LPADC\_GetConvResult ( ADC\_Type \* *base*, lpadc\_conv\_result\_t \* *result*, uint8\_t *index* )

param *base* LPADC peripheral base address. param *result* Pointer to structure variable that keeps the conversion result in conversion FIFO. param *index* Result FIFO index.

return Status whether FIFO entry is valid.

### 21.6.18 void LPADC\_SetConvTriggerConfig ( ADC\_Type \* *base*, uint32\_t *triggerId*, const lpadc\_conv\_trigger\_config\_t \* *config* )

Each programmable trigger can launch the conversion command in command buffer.

Parameters

|                  |                                                                      |
|------------------|----------------------------------------------------------------------|
| <i>base</i>      | LPADC peripheral base address.                                       |
| <i>triggerId</i> | ID for each trigger. Typically, the available value range is from 0. |

|               |                                                                                          |
|---------------|------------------------------------------------------------------------------------------|
| <i>config</i> | Pointer to configuration structure. See to <a href="#">lpadc_conv_trigger_config_t</a> . |
|---------------|------------------------------------------------------------------------------------------|

### 21.6.19 void LPADC\_GetDefaultConvTriggerConfig ( *lpadc\_conv\_trigger\_config\_t* \* *config* )

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
* config->commandIdSource      = 0U;
* config->loopCountIndex      = 0U;
* config->triggerIdSource     = 0U;
* config->enableHardwareTrigger = false;
*
```

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>config</i> | Pointer to configuration structure. |
|---------------|-------------------------------------|

### 21.6.20 static void LPADC\_DoSoftwareTrigger ( *ADC\_Type* \* *base*, *uint32\_t* *triggerIdMask* ) [inline], [static]

Parameters

|                      |                                                                 |
|----------------------|-----------------------------------------------------------------|
| <i>base</i>          | LPADC peripheral base address.                                  |
| <i>triggerIdMask</i> | Mask value for software trigger indexes, which count from zero. |

### 21.6.21 void LPADC\_SetConvCommandConfig ( *ADC\_Type* \* *base*, *uint32\_t* *commandId*, const *lpadc\_conv\_command\_config\_t* \* *config* )

Parameters

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| <i>base</i>      | LPADC peripheral base address.                                                           |
| <i>commandId</i> | ID for command in command buffer. Typically, the available value range is 1 - 15.        |
| <i>config</i>    | Pointer to configuration structure. See to <a href="#">lpadc_conv_command_config_t</a> . |

### 21.6.22 void LPADC\_GetDefaultConvCommandConfig ( *Ipadc\_conv\_command\_config\_t* \* *config* )

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
* config->sampleScaleMode          = kLPADC_SampleFullScale;
* config->channelBScaleMode       = kLPADC_SampleFullScale;
* config->channelSampleMode        = kLPADC_SampleChannelSingleEndSideA
* ;
* config->channelNumber           = OU;
* config->chainedNextCmdNumber     = OU;
* config->enableAutoChannelIncrement = false;
* config->loopCount                = OU;
* config->hardwareAverageMode      = kLPADC_HardwareAverageCount1;
* config->sampleTimeMode           = kLPADC_SampleTimeADCK3;
* config->hardwareCompareMode      = kLPADC_HardwareCompareDisabled;
* config->hardwareCompareValueHigh = OU;
* config->hardwareCompareValueLow  = OU;
* config->conversionResolutionMode = kLPADC_ConversionResolutionStandard
* ;
* config->enableWaitTrigger        = false;
* config->enableChannelB           = false;
*
```

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>config</i> | Pointer to configuration structure. |
|---------------|-------------------------------------|

### 21.6.23 static void LPADC\_SetOffsetValue ( *ADC\_Type* \* *base*, *uint32\_t* *valueA*, *uint32\_t* *valueB* ) [inline], [static]

Set the offset trim value for offset calibration manually.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>valueA</i> | Setting offset value A.        |
| <i>valueB</i> | Setting offset value B.        |

Note

In normal adc sequence, the values are automatically calculated by LPADC\_EnableOffsetCalibration.

### 21.6.24 static void LPADC\_EnableOffsetCalibration ( *ADC\_Type* \* *base*, *bool* *enable* ) [inline], [static]

Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>base</i>   | LPADC peripheral base address.        |
| <i>enable</i> | switcher to the calibration function. |

### 21.6.25 void LPADC\_DoOffsetCalibration ( ADC\_Type \* *base* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | LPADC peripheral base address. |
|-------------|--------------------------------|

### 21.6.26 void LPADC\_DoAutoCalibration ( ADC\_Type \* *base* )

param base LPADC peripheral base address.

# Chapter 22

## CRC: Cyclic Redundancy Check Driver

### 22.1 Overview

MCUXpresso SDK provides a peripheral driver for the Cyclic Redundancy Check (CRC) module of MCUXpresso SDK devices.

The cyclic redundancy check (CRC) module generates 16/32-bit CRC code for error detection. The CRC module provides three variants of polynomials, a programmable seed, and other parameters required to implement a 16-bit or 32-bit CRC standard.

### 22.2 CRC Driver Initialization and Configuration

[CRC\\_Init\(\)](#) function enables the clock for the CRC module in the LPC SYSCON block and fully (re-)configures the CRC module according to configuration structure. It also starts checksum computation by writing the seed.

The seed member of the configuration structure is the initial checksum for which new data can be added to. When starting new checksum computation, the seed should be set to the initial checksum per the CRC protocol specification. For continued checksum operation, the seed should be set to the intermediate checksum value as obtained from previous calls to [CRC\\_GetConfig\(\)](#) function. After [CRC\\_Init\(\)](#), one or multiple [CRC\\_WriteData\(\)](#) calls follow to update checksum with data, then [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) follows to read the result. [CRC\\_Init\(\)](#) can be called as many times as required, which allows for runtime changes of the CRC protocol.

[CRC\\_GetDefaultConfig\(\)](#) function can be used to set the module configuration structure with parameters for CRC-16/CCITT-FALSE protocol.

[CRC\\_Deinit\(\)](#) function disables clock to the CRC module.

[CRC\\_Reset\(\)](#) performs hardware reset of the CRC module.

### 22.3 CRC Write Data

The [CRC\\_WriteData\(\)](#) function is used to add data to actual CRC. Internally it tries to use 32-bit reads and writes for all aligned data in the user buffer and it uses 8-bit reads and writes for all unaligned data in the user buffer. This function can update CRC with user supplied data chunks of arbitrary size, so one can update CRC byte by byte or with all bytes at once. Prior call of CRC configuration function [CRC\\_Init\(\)](#) fully specifies the CRC module configuration for [CRC\\_WriteData\(\)](#) call.

[CRC\\_WriteSeed\(\)](#) Write seed (initial checksum) to CRC module.

### 22.4 CRC Get Checksum

The [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) function is used to read the CRC module checksum register. The bit reverse and 1's complement operations are already applied to the result if previously

configured. Use [CRC\\_GetConfig\(\)](#) function to get the actual checksum without bit reverse and 1's complement applied so it can be used as seed when resuming calculation later.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / [CRC\\_Get16bitResult\(\)](#) to get final checksum.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / ... / [CRC\\_WriteData\(\)](#) / [CRC\\_Get16bitResult\(\)](#) to get final checksum.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / [CRC\\_GetConfig\(\)](#) to get intermediate checksum to be used as seed value in future.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / ... / [CRC\\_WriteData\(\)](#) / [CRC\\_GetConfig\(\)](#) to get intermediate checksum.

## 22.5 Comments about API usage in RTOS

If multiple RTOS tasks share the CRC module to compute checksums with different data and/or protocols, the following needs to be implemented by the user:

The triplets

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) or [CRC\\_GetConfig\(\)](#)

Should be protected by RTOS mutex to protect CRC module against concurrent accesses from different tasks. For example: Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/crcRefer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/crc

### Files

- file [fsl\\_crc.h](#)

### Data Structures

- struct [crc\\_config\\_t](#)  
*CRC protocol configuration.* [More...](#)

### Macros

- #define [CRC\\_DRIVER\\_USE\\_CRC16\\_CCITT\\_FALSE\\_AS\\_DEFAULT](#) 1  
*Default configuration structure filled by [CRC\\_GetDefaultConfig\(\)](#).*

### Enumerations

- enum [crc\\_polynomial\\_t](#) {  
    kCRC\_Polynomial\_CRC\_CCITT = 0U,  
    kCRC\_Polynomial\_CRC\_16 = 1U,  
    kCRC\_Polynomial\_CRC\_32 = 2U }  
*CRC polynomials to use.*

## Functions

- void **CRC\_Init** (CRC\_Type \*base, const **crc\_config\_t** \*config)  
*Enables and configures the CRC peripheral module.*
- static void **CRC\_Deinit** (CRC\_Type \*base)  
*Disables the CRC peripheral module.*
- void **CRC\_Reset** (CRC\_Type \*base)  
*resets CRC peripheral module.*
- void **CRC\_WriteSeed** (CRC\_Type \*base, uint32\_t seed)  
*Write seed to CRC peripheral module.*
- void **CRC\_GetDefaultConfig** (**crc\_config\_t** \*config)  
*Loads default values to CRC protocol configuration structure.*
- void **CRC\_GetConfig** (CRC\_Type \*base, **crc\_config\_t** \*config)  
*Loads actual values configured in CRC peripheral to CRC protocol configuration structure.*
- void **CRC\_WriteData** (CRC\_Type \*base, const uint8\_t \*data, size\_t dataSize)  
*Writes data to the CRC module.*
- static uint32\_t **CRC\_Get32bitResult** (CRC\_Type \*base)  
*Reads 32-bit checksum from the CRC module.*
- static uint16\_t **CRC\_Get16bitResult** (CRC\_Type \*base)  
*Reads 16-bit checksum from the CRC module.*

## Driver version

- #define **FSL\_CRC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 1))  
*CRC driver version.*

## 22.6 Data Structure Documentation

### 22.6.1 struct **crc\_config\_t**

This structure holds the configuration for the CRC protocol.

## Data Fields

- **crc\_polynomial\_t polynomial**  
*CRC polynomial.*
- bool **reverseIn**  
*Reverse bits on input.*
- bool **complementIn**  
*Perform 1's complement on input.*
- bool **reverseOut**  
*Reverse bits on output.*
- bool **complementOut**  
*Perform 1's complement on output.*
- uint32\_t **seed**  
*Starting checksum value.*

### Field Documentation

- (1) `crc_polynomial_t crc_config_t::polynomial`
- (2) `bool crc_config_t::reverseln`
- (3) `bool crc_config_t::complementIn`
- (4) `bool crc_config_t::reverseOut`
- (5) `bool crc_config_t::complementOut`
- (6) `uint32_t crc_config_t::seed`

## 22.7 Macro Definition Documentation

### 22.7.1 #define FSL\_CRC\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.0.0
  - initial version
- Version 2.0.1
  - add explicit type cast when writing to WR\_DATA
- Version 2.0.2
  - Fix MISRA issue
- Version 2.1.0
  - Add CRC\_WriteSeed function
- Version 2.1.1
  - Fix MISRA issue

### 22.7.2 #define CRC\_DRIVER\_USE\_CRC16\_CCITT\_FALSE\_AS\_DEFAULT 1

Uses CRC-16/CCITT-FALSE as default.

## 22.8 Enumeration Type Documentation

### 22.8.1 enum crc\_polynomial\_t

Enumerator

*kCRC\_Polynomial\_CRC\_CCITT*  $x^{16}+x^{12}+x^5+1$

*kCRC\_Polynomial\_CRC\_16*  $x^{16}+x^{15}+x^2+1$

*kCRC\_Polynomial\_CRC\_32*  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

## 22.9 Function Documentation

### 22.9.1 void CRC\_Init ( **CRC\_Type** \* *base*, **const crc\_config\_t** \* *config* )

This function enables the CRC peripheral clock in the LPC SYSCON block. It also configures the CRC engine and starts checksum computation by writing the seed.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | CRC peripheral address.             |
| <i>config</i> | CRC module configuration structure. |

## 22.9.2 static void CRC\_Deinit ( **CRC\_Type** \* *base* ) [inline], [static]

This function disables the CRC peripheral clock in the LPC SYSCON block.

Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | CRC peripheral address. |
|-------------|-------------------------|

## 22.9.3 void CRC\_Reset ( **CRC\_Type** \* *base* )

Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | CRC peripheral address. |
|-------------|-------------------------|

## 22.9.4 void CRC\_WriteSeed ( **CRC\_Type** \* *base*, **uint32\_t** *seed* )

Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | CRC peripheral address. |
| <i>seed</i> | CRC Seed value.         |

## 22.9.5 void CRC\_GetDefaultConfig ( **crc\_config\_t** \* *config* )

Loads default values to CRC protocol configuration structure. The default values are:

```
* config->polynomial = kCRC_Polynomial_CRC_CCITT;
* config->reverseIn = false;
* config->complementIn = false;
* config->reverseOut = false;
* config->complementOut = false;
* config->seed = 0xFFFFU;
*
```

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>config</i> | CRC protocol configuration structure |
|---------------|--------------------------------------|

## 22.9.6 void CRC\_GetConfig ( **CRC\_Type** \* *base*, **crc\_config\_t** \* *config* )

The values, including seed, can be used to resume CRC calculation later.

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | CRC peripheral address.              |
| <i>config</i> | CRC protocol configuration structure |

## 22.9.7 void CRC\_WriteData ( **CRC\_Type** \* *base*, **const uint8\_t** \* *data*, **size\_t** *dataSize* )

Writes input data buffer bytes to CRC data register.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | CRC peripheral address.                 |
| <i>data</i>     | Input data stream, MSByte in data[0].   |
| <i>dataSize</i> | Size of the input data buffer in bytes. |

## 22.9.8 static uint32\_t CRC\_Get32bitResult ( **CRC\_Type** \* *base* ) [inline], [static]

Reads CRC data register.

Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | CRC peripheral address. |
|-------------|-------------------------|

Returns

final 32-bit checksum, after configured bit reverse and complement operations.

**22.9.9 static uint16\_t CRC\_Get16bitResult ( CRC\_Type \* *base* ) [inline],  
[static]**

Reads CRC data register.

### Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | CRC peripheral address. |
|-------------|-------------------------|

### Returns

final 16-bit checksum, after configured bit reverse and complement operations.

# Chapter 23

## DMA: Direct Memory Access Controller Driver

### 23.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Direct Memory Access (DMA) of MCUXpresso SDK devices.

### 23.2 Typical use case

#### 23.2.1 DMA Operation

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/dma

### Files

- file [fsl\\_dma.h](#)

### Data Structures

- struct [dma\\_descriptor\\_t](#)  
*DMA descriptor structure. [More...](#)*
- struct [dma\\_xfercfg\\_t](#)  
*DMA transfer configuration. [More...](#)*
- struct [dma\\_channel\\_trigger\\_t](#)  
*DMA channel trigger. [More...](#)*
- struct [dma\\_channel\\_config\\_t](#)  
*DMA channel trigger. [More...](#)*
- struct [dma\\_transfer\\_config\\_t](#)  
*DMA transfer configuration. [More...](#)*
- struct [dma\\_handle\\_t](#)  
*DMA transfer handle structure. [More...](#)*

### Macros

- #define [DMA\\_MAX\\_TRANSFER\\_COUNT](#) 0x400U  
*DMA max transfer size.*
- #define [FSL\\_FEATURE\\_DMA\\_LINK\\_DESCRIPTOR\\_ALIGN\\_SIZE](#) (16U)  
*DMA channel numbers.*
- #define [DMA\\_ALLOCATE\\_HEAD\\_DESCRIPTOR](#)(name, number) SDK\_ALIGN([dma\\_descriptor\\_t](#) name[number], FSL\_FEATURE\_DMA\_DESCRIPTOR\_ALIGN\_SIZE)  
*DMA head descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.*
- #define [DMA\\_ALLOCATE\\_HEAD\\_DESCRIPTOR\\_AT\\_NONCACHEABLE](#)(name, number) AT\_NONCACHEABLE\_SECTION\_ALIGN([dma\\_descriptor\\_t](#) name[number], FSL\_FEATURE\_DMA\_DESCRIPTOR\_ALIGN\_SIZE)

*DMA head descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.*

- #define **DMA\_ALLOCATE\_LINK\_DESCRIPTOR**(name, number) SDK\_ALIGN(dma\_descriptor\_t name[number], FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SIZE)
 

*DMA link descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.*
- #define **DMA\_ALLOCATE\_LINK\_DESCRIPTOR\_AT\_NONCACHEABLE**(name, number) AT\_NONCACHEABLE\_SECTION\_ALIGN(dma\_descriptor\_t name[number], FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SIZE)
 

*DMA link descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.*
- #define **DMA\_ALLOCATE\_DATA\_TRANSFER\_BUFFER**(name, width) SDK\_ALIGN(name, width)
 

*DMA transfer buffer address need to align with the transfer width.*
- #define **DMA\_COMMON\_REG\_GET**(base, channel, reg) (((volatile uint32\_t \*)(&((base)->COMMON[0].reg)))[DMA\_CHANNEL\_GROUP(channel)])
 

*DMA linked descriptor address algin size.*
- #define **DMA\_DESCRIPTOR\_END\_ADDRESS**(start, inc, bytes, width) ((uint32\_t \*)((uint32\_t)(start) + (inc) \* (bytes) - (inc) \* (width)))
 

*DMA descriptor end address calculate.*

## Typedefs

- typedef void(\* **dma\_callback** )(struct \_dma\_handle \*handle, void \*userData, bool transferDone, uint32\_t intmode)
 

*Define Callback function for DMA.*

## Enumerations

- enum { **kStatus\_DMA\_Busy** = MAKE\_STATUS(kStatusGroup\_DMA, 0) }
 

*\_dma\_transfer\_status DMA transfer status*
- enum {
 **kDMA\_AddressInterleave0xWidth** = 0U,
 **kDMA\_AddressInterleave1xWidth** = 1U,
 **kDMA\_AddressInterleave2xWidth** = 2U,
 **kDMA\_AddressInterleave4xWidth** = 4U }
 

*\_dma\_addr\_interleave\_size dma address interleave size*
- enum {
 **kDMA\_Transfer8BitWidth** = 1U,
 **kDMA\_Transfer16BitWidth** = 2U,
 **kDMA\_Transfer32BitWidth** = 4U }
 

*\_dma\_transfer\_width dma transfer width*
- enum **dma\_priority\_t** {

- ```

kDMA_ChannelPriority0 = 0,
kDMA_ChannelPriority1,
kDMA_ChannelPriority2,
kDMA_ChannelPriority3,
kDMA_ChannelPriority4,
kDMA_ChannelPriority5,
kDMA_ChannelPriority6,
kDMA_ChannelPriority7 }

    DMA channel priority.
• enum dma_irq_t {
    kDMA_IntA,
    kDMA_IntB,
    kDMA_IntError }

    DMA interrupt flags.
• enum dma_trigger_type_t {
    kDMA_NoTrigger = 0,
    kDMA_LowLevelTrigger = DMA_CHANNEL_CFG_HWTRIGEN(1) | DMA_CHANNEL_CFG_
    _TRIGTYPE(1),
    kDMA_HighLevelTrigger,
    kDMA_FallingEdgeTrigger = DMA_CHANNEL_CFG_HWTRIGEN(1),
    kDMA_RisingEdgeTrigger }

    DMA trigger type.
• enum {
    kDMA_BurstSize1 = 0U,
    kDMA_BurstSize2 = 1U,
    kDMA_BurstSize4 = 2U,
    kDMA_BurstSize8 = 3U,
    kDMA_BurstSize16 = 4U,
    kDMA_BurstSize32 = 5U,
    kDMA_BurstSize64 = 6U,
    kDMA_BurstSize128 = 7U,
    kDMA_BurstSize256 = 8U,
    kDMA_BurstSize512 = 9U,
    kDMA_BurstSize1024 = 10U }
        _dma_burst_size DMA burst size
• enum dma_trigger_burst_t {

```

```

kDMA_SingleTransfer = 0,
kDMA_LevelBurstTransfer = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer1 = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer2,
kDMA_EdgeBurstTransfer4,
kDMA_EdgeBurstTransfer8,
kDMA_EdgeBurstTransfer16,
kDMA_EdgeBurstTransfer32,
kDMA_EdgeBurstTransfer64,
kDMA_EdgeBurstTransfer128,
kDMA_EdgeBurstTransfer256,
kDMA_EdgeBurstTransfer512,
kDMA_EdgeBurstTransfer1024 }

```

*DMA trigger burst.*

- enum `dma_burst_wrap_t` {
 kDMA\_NoWrap = 0,
 kDMA\_SrcWrap = DMA\_CHANNEL\_CFG\_SRCBURSTWRAP(1),
 kDMA\_DstWrap = DMA\_CHANNEL\_CFG\_DSTBURSTWRAP(1),
 kDMA\_SrcAndDstWrap }

*DMA burst wrapping.*

- enum `dma_transfer_type_t` {
 kDMA\_MemoryToMemory = 0x0U,
 kDMA\_PeripheralToMemory,
 kDMA\_MemoryToPeripheral,
 kDMA\_StaticToStatic }

*DMA transfer type.*

## Driver version

- #define `FSL_DMA_DRIVER_VERSION` (MAKE\_VERSION(2, 5, 0))  
*DMA driver version.*

## DMA initialization and De-initialization

- void `DMA_Init` (DMA\_Type \*base)  
*Initializes DMA peripheral.*
- void `DMA_Deinit` (DMA\_Type \*base)  
*Deinitializes DMA peripheral.*
- void `DMA_InstallDescriptorMemory` (DMA\_Type \*base, void \*addr)  
*Install DMA descriptor memory.*

## DMA Channel Operation

- static bool `DMA_ChannelIsActive` (DMA\_Type \*base, uint32\_t channel)  
*Return whether DMA channel is processing transfer.*
- static bool `DMA_ChannelIsBusy` (DMA\_Type \*base, uint32\_t channel)  
*Return whether DMA channel is busy.*
- static void `DMA_EnableChannelInterrupts` (DMA\_Type \*base, uint32\_t channel)

- static void **DMA\_DisableChannelInterrupts** (DMA\_Type \*base, uint32\_t channel)
 

*Disables the interrupt source for the DMA transfer.*
- static void **DMA\_EnableChannel** (DMA\_Type \*base, uint32\_t channel)
 

*Enable DMA channel.*
- static void **DMA\_DisableChannel** (DMA\_Type \*base, uint32\_t channel)
 

*Disable DMA channel.*
- static void **DMA\_EnableChannelPeriphRq** (DMA\_Type \*base, uint32\_t channel)
 

*Set PERIPHREQEN of channel configuration register.*
- static void **DMA\_DisableChannelPeriphRq** (DMA\_Type \*base, uint32\_t channel)
 

*Get PERIPHREQEN value of channel configuration register.*
- void **DMA\_ConfigureChannelTrigger** (DMA\_Type \*base, uint32\_t channel, **dma\_channel\_trigger\_t** \*trigger)
 

*Set trigger settings of DMA channel.*
- void **DMA\_SetChannelConfig** (DMA\_Type \*base, uint32\_t channel, **dma\_channel\_trigger\_t** \*trigger, bool isPeriph)
 

*set channel config.*
- static uint32\_t **DMA\_SetChannelXferConfig** (bool reload, bool clrTrig, bool intA, bool intB, uint8\_t width, uint8\_t srcInc, uint8\_t dstInc, uint32\_t bytes)
 

*DMA channel xfer transfer configurations.*
- uint32\_t **DMA\_GetRemainingBytes** (DMA\_Type \*base, uint32\_t channel)
 

*Gets the remaining bytes of the current DMA descriptor transfer.*
- static void **DMA\_SetChannelPriority** (DMA\_Type \*base, uint32\_t channel, **dma\_priority\_t** priority)
 

*Set priority of channel configuration register.*
- static **dma\_priority\_t DMA\_GetChannelPriority** (DMA\_Type \*base, uint32\_t channel)
 

*Get priority of channel configuration register.*
- static void **DMA\_SetChannelConfigValid** (DMA\_Type \*base, uint32\_t channel)
 

*Set channel configuration valid.*
- static void **DMA\_DoChannelSoftwareTrigger** (DMA\_Type \*base, uint32\_t channel)
 

*Do software trigger for the channel.*
- static void **DMA\_LoadChannelTransferConfig** (DMA\_Type \*base, uint32\_t channel, uint32\_t xfer)
 

*Load channel transfer configurations.*
- void **DMA\_CreateDescriptor** (**dma\_descriptor\_t** \*desc, **dma\_xfercfg\_t** \*xfercfg, void \*srcAddr, void \*dstAddr, void \*nextDesc)
 

*Create application specific DMA descriptor to be used in a chain in transfer.*
- void **DMA\_SetupDescriptor** (**dma\_descriptor\_t** \*desc, uint32\_t xfercfg, void \*srcStartAddr, void \*dstStartAddr, void \*nextDesc)
 

*setup dma descriptor*
- void **DMA\_SetupChannelDescriptor** (**dma\_descriptor\_t** \*desc, uint32\_t xfercfg, void \*srcStartAddr, void \*dstStartAddr, void \*nextDesc, **dma\_burst\_wrap\_t** wrapType, uint32\_t burstSize)
 

*setup dma channel descriptor*
- void **DMA\_LoadChannelDescriptor** (DMA\_Type \*base, uint32\_t channel, **dma\_descriptor\_t** \*descriptor)
 

*load channel transfer descriptor.*

## DMA Transactional Operation

- void **DMA\_AbortTransfer** (**dma\_handle\_t** \*handle)
 

*Abort running transfer by handle.*
- void **DMA\_CreateHandle** (**dma\_handle\_t** \*handle, DMA\_Type \*base, uint32\_t channel)

- Creates the DMA handle.
- void **DMA\_SetCallback** (**dma\_handle\_t** \*handle, **dma\_callback** callback, void \*userData)
  - Installs a callback function for the DMA transfer.
- void **DMA\_PrepTransfer** (**dma\_transfer\_config\_t** \*config, void \*srcAddr, void \*dstAddr, uint32\_t byteWidth, uint32\_t transferBytes, **dma\_transfer\_type\_t** type, void \*nextDesc)
  - Prepares the DMA transfer structure.
- void **DMA\_PrepChannelTransfer** (**dma\_channel\_config\_t** \*config, void \*srcStartAddr, void \*dstStartAddr, uint32\_t xferCfg, **dma\_transfer\_type\_t** type, **dma\_channel\_trigger\_t** \*trigger, void \*nextDesc)
  - Prepare channel transfer configurations.
- **status\_t DMA\_SubmitTransfer** (**dma\_handle\_t** \*handle, **dma\_transfer\_config\_t** \*config)
  - Submits the DMA transfer request.
- void **DMA\_SubmitChannelTransferParameter** (**dma\_handle\_t** \*handle, uint32\_t xferCfg, void \*srcStartAddr, void \*dstStartAddr, void \*nextDesc)
  - Submit channel transfer parameter directly.
- void **DMA\_SubmitChannelDescriptor** (**dma\_handle\_t** \*handle, **dma\_descriptor\_t** \*descriptor)
  - Submit channel descriptor.
- **status\_t DMA\_SubmitChannelTransfer** (**dma\_handle\_t** \*handle, **dma\_channel\_config\_t** \*config)
  - Submits the DMA channel transfer request.
- void **DMA\_StartTransfer** (**dma\_handle\_t** \*handle)
  - DMA start transfer.
- void **DMA\_IRQHandle** (**DMA\_Type** \*base)
  - DMA IRQ handler for descriptor transfer complete.

## 23.3 Data Structure Documentation

### 23.3.1 struct **dma\_descriptor\_t**

#### Data Fields

- volatile uint32\_t **xfercfg**
  - Transfer configuration.
- void \* **srcEndAddr**
  - Last source address of DMA transfer.
- void \* **dstEndAddr**
  - Last destination address of DMA transfer.
- void \* **linkToNextDesc**
  - Address of next DMA descriptor in chain.

### 23.3.2 struct **dma\_xfercfg\_t**

#### Data Fields

- bool **valid**
  - Descriptor is ready to transfer.
- bool **reload**
  - Reload channel configuration register after current descriptor is exhausted.
- bool **swtrig**

- *Perform software trigger.*
- bool `clrtrig`  
*Clear trigger.*
- bool `intA`  
*Raises IRQ when transfer is done and set IRQA status register flag.*
- bool `intB`  
*Raises IRQ when transfer is done and set IRQB status register flag.*
- uint8\_t `byteWidth`  
*Byte width of data to transfer.*
- uint8\_t `srcInc`  
*Increment source address by 'srcInc' x 'byteWidth'.*
- uint8\_t `dstInc`  
*Increment destination address by 'dstInc' x 'byteWidth'.*
- uint16\_t `transferCount`  
*Number of transfers.*

## Field Documentation

(1) **bool dma\_xfercfg\_t::swtrig**

Transfer if fired when 'valid' is set

### 23.3.3 struct dma\_channel\_trigger\_t

## Data Fields

- `dma_trigger_type_t type`  
*Select hardware trigger as edge triggered or level triggered.*
- `dma_trigger_burst_t burst`  
*Select whether hardware triggers cause a single or burst transfer.*
- `dma_burst_wrap_t wrap`  
*Select wrap type, source wrap or dest wrap, or both.*

## Field Documentation

(1) `dma_trigger_type_t dma_channel_trigger_t::type`

(2) `dma_trigger_burst_t dma_channel_trigger_t::burst`

(3) `dma_burst_wrap_t dma_channel_trigger_t::wrap`

### 23.3.4 struct dma\_channel\_config\_t

## Data Fields

- void \* `srcStartAddr`  
*Source data address.*
- void \* `dstStartAddr`

- *Destination data address.*  
void \* **nextDesc**
- *Chain custom descriptor.*  
uint32\_t **xferCfg**
- *channel transfer configurations*  
**dma\_channel\_trigger\_t** \* **trigger**
- *DMA trigger type.*  
bool **isPeriph**
- *select the request type*

### 23.3.5 struct dma\_transfer\_config\_t

#### Data Fields

- uint8\_t \* **srcAddr**  
*Source data address.*
- uint8\_t \* **dstAddr**  
*Destination data address.*
- uint8\_t \* **nextDesc**
- *Chain custom descriptor.*  
**dma\_xfercfg\_t** **xfercfg**
- *Transfer options.*  
bool **isPeriph**
- *DMA transfer is driven by peripheral.*

### 23.3.6 struct dma\_handle\_t

#### Data Fields

- **dma\_callback** **callback**  
*Callback function.*
- void \* **userData**  
*Callback function parameter.*
- DMA\_Type \* **base**  
*DMA peripheral base address.*
- uint8\_t **channel**  
*DMA channel number.*

#### Field Documentation

##### (1) **dma\_callback** **dma\_handle\_t::callback**

Invoked when transfer of descriptor with interrupt flag finishes

## 23.4 Macro Definition Documentation

**23.4.1 #define FSL\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 0))**

Version 2.5.0.

**23.4.2 #define FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SIZE (16U)**

DMA head link descriptor table align size

**23.4.3 #define DMA\_ALLOCATE\_HEAD\_DESCRIPTOR( *name*,  
*number* ) SDK\_ALIGN(dma\_descriptor\_t *name[number]*,  
FSL\_FEATURE\_DMA\_DESCRIPTOR\_ALIGN\_SIZE)**

Parameters

<i>name</i>	Allocate descriptor name.
<i>number</i>	Number of descriptor to be allocated.

**23.4.4 #define DMA\_ALLOCATE\_HEAD\_DESCRIPTOR\_AT\_NONCACHEABLE(  
*name*, *number* ) AT\_NONCACHEABLE\_SECTION\_ALIGN(dma\_descriptor\_t  
*name[number]*, FSL\_FEATURE\_DMA\_DESCRIPTOR\_ALIGN\_SIZE)**

Parameters

<i>name</i>	Allocate descriptor name.
<i>number</i>	Number of descriptor to be allocated.

**23.4.5 #define DMA\_ALLOCATE\_LINK\_DESCRIPTOR( *name*,  
*number* ) SDK\_ALIGN(dma\_descriptor\_t *name[number]*,  
FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SIZE)**

Parameters

<i>name</i>	Allocate descriptor name.
<i>number</i>	Number of descriptor to be allocated.

**23.4.6 #define DMA\_ALLOCATE\_LINK\_DESCRIPTOR\_AT\_NONCACHEABLE(  
*name*, *number*) AT\_NONCACHEABLE\_SECTION\_ALIGN(dma\_descriptor\_t  
*name[number]*, FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SI-  
ZE)**

Parameters

<i>name</i>	Allocate descriptor name.
<i>number</i>	Number of descriptor to be allocated.

**23.4.7 #define DMA\_DESCRIPTOR\_END\_ADDRESS( *start*, *inc*, *bytes*, *width*  
) ((uint32\_t \*)((uint32\_t)(*start*) + (*inc*) \* (*bytes*) - (*inc*) \* (*width*)))**

Parameters

<i>start</i>	start address
<i>inc</i>	address interleave size
<i>bytes</i>	transfer bytes
<i>width</i>	transfer width

## 23.5 Typedef Documentation

**23.5.1 typedef void(\* dma\_callback)(struct \_dma\_handle \*handle, void \*userData,  
bool transferDone, uint32\_t intmode)**

## 23.6 Enumeration Type Documentation

### 23.6.1 anonymous enum

Enumerator

***kStatus\_DMA\_Busy*** Channel is busy and can't handle the transfer request.

### 23.6.2 anonymous enum

Enumerator

<i>kDMA_AddressInterleave0xWidth</i>	dma source/destination address no interleave
<i>kDMA_AddressInterleave1xWidth</i>	dma source/destination address interleave 1xwidth
<i>kDMA_AddressInterleave2xWidth</i>	dma source/destination address interleave 2xwidth
<i>kDMA_AddressInterleave4xWidth</i>	dma source/destination address interleave 3xwidth

### 23.6.3 anonymous enum

Enumerator

<i>kDMA_Transfer8BitWidth</i>	dma channel transfer bit width is 8 bit
<i>kDMA_Transfer16BitWidth</i>	dma channel transfer bit width is 16 bit
<i>kDMA_Transfer32BitWidth</i>	dma channel transfer bit width is 32 bit

### 23.6.4 enum dma\_priority\_t

Enumerator

<i>kDMA_ChannelPriority0</i>	Highest channel priority - priority 0.
<i>kDMA_ChannelPriority1</i>	Channel priority 1.
<i>kDMA_ChannelPriority2</i>	Channel priority 2.
<i>kDMA_ChannelPriority3</i>	Channel priority 3.
<i>kDMA_ChannelPriority4</i>	Channel priority 4.
<i>kDMA_ChannelPriority5</i>	Channel priority 5.
<i>kDMA_ChannelPriority6</i>	Channel priority 6.
<i>kDMA_ChannelPriority7</i>	Lowest channel priority - priority 7.

### 23.6.5 enum dma\_irq\_t

Enumerator

<i>kDMA_IntA</i>	DMA interrupt flag A.
<i>kDMA_IntB</i>	DMA interrupt flag B.
<i>kDMA_IntError</i>	DMA interrupt flag error.

### 23.6.6 enum dma\_trigger\_type\_t

Enumerator

<i>kDMA_NoTrigger</i>	Trigger is disabled.
-----------------------	----------------------

**kDMA\_LowLevelTrigger** Low level active trigger.  
**kDMA\_HighLevelTrigger** High level active trigger.  
**kDMA\_FallingEdgeTrigger** Falling edge active trigger.  
**kDMA\_RisingEdgeTrigger** Rising edge active trigger.

### 23.6.7 anonymous enum

Enumerator

**kDMA\_BurstSize1** burst size 1 transfer  
**kDMA\_BurstSize2** burst size 2 transfer  
**kDMA\_BurstSize4** burst size 4 transfer  
**kDMA\_BurstSize8** burst size 8 transfer  
**kDMA\_BurstSize16** burst size 16 transfer  
**kDMA\_BurstSize32** burst size 32 transfer  
**kDMA\_BurstSize64** burst size 64 transfer  
**kDMA\_BurstSize128** burst size 128 transfer  
**kDMA\_BurstSize256** burst size 256 transfer  
**kDMA\_BurstSize512** burst size 512 transfer  
**kDMA\_BurstSize1024** burst size 1024 transfer

### 23.6.8 enum dma\_trigger\_burst\_t

Enumerator

**kDMA\_SingleTransfer** Single transfer.  
**kDMA\_LevelBurstTransfer** Burst transfer driven by level trigger.  
**kDMA\_EdgeBurstTransfer1** Perform 1 transfer by edge trigger.  
**kDMA\_EdgeBurstTransfer2** Perform 2 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer4** Perform 4 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer8** Perform 8 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer16** Perform 16 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer32** Perform 32 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer64** Perform 64 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer128** Perform 128 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer256** Perform 256 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer512** Perform 512 transfers by edge trigger.  
**kDMA\_EdgeBurstTransfer1024** Perform 1024 transfers by edge trigger.

### 23.6.9 enum dma\_burst\_wrap\_t

Enumerator

*kDMA\_NoWrap* Wrapping is disabled.

*kDMA\_SrcWrap* Wrapping is enabled for source.

*kDMA\_DstWrap* Wrapping is enabled for destination.

*kDMA\_SrcAndDstWrap* Wrapping is enabled for source and destination.

### 23.6.10 enum dma\_transfer\_type\_t

Enumerator

*kDMA\_MemoryToMemory* Transfer from memory to memory (increment source and destination)

*kDMA\_PeripheralToMemory* Transfer from peripheral to memory (increment only destination)

*kDMA\_MemoryToPeripheral* Transfer from memory to peripheral (increment only source)

*kDMA\_StaticToStatic* Peripheral to static memory (do not increment source or destination)

## 23.7 Function Documentation

### 23.7.1 void DMA\_Init ( DMA\_Type \* *base* )

This function enable the DMA clock, set descriptor table and enable DMA peripheral.

Parameters

<i>base</i>	DMA peripheral base address.
-------------	------------------------------

### 23.7.2 void DMA\_Deinit ( DMA\_Type \* *base* )

This function gates the DMA clock.

Parameters

<i>base</i>	DMA peripheral base address.
-------------	------------------------------

### 23.7.3 void DMA\_InstallDescriptorMemory ( DMA\_Type \* *base*, void \* *addr* )

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, although current DMA driver has a default DMA descriptor buffer, but it support one DMA descriptor for one channel only.

Parameters

<i>base</i>	DMA base address.
<i>addr</i>	DMA descriptor address

#### 23.7.4 static bool DMA\_ChannelsActive ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

Returns

True for active state, false otherwise.

#### 23.7.5 static bool DMA\_ChannelsBusy ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

Returns

True for busy state, false otherwise.

#### 23.7.6 static void DMA\_EnableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

### 23.7.7 static void DMA\_DisableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

### 23.7.8 static void DMA\_EnableChannel ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

### 23.7.9 static void DMA\_DisableChannel ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

### 23.7.10 static void DMA\_EnableChannelPeriphRq ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

### 23.7.11 static void DMA\_DisableChannelPeriphRq ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

Returns

True for enabled PeriphRq, false for disabled.

### 23.7.12 void DMA\_ConfigureChannelTrigger ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_channel\_trigger\_t \* *trigger* )

**Deprecated** Do not use this function. It has been superceded by [DMA\\_SetChannelConfig](#).

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>trigger</i>	trigger configuration.

### 23.7.13 void DMA\_SetChannelConfig ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_channel\_trigger\_t \* *trigger*, bool *isPeriph* )

This function provide a interface to configure channel configuration reisters.

Parameters

<i>base</i>	DMA base address.
<i>channel</i>	DMA channel number.
<i>trigger</i>	channel configurations structure.
<i>isPeriph</i>	true is periph request, false is not.

**23.7.14 static uint32\_t DMA\_SetChannelXferConfig ( bool *reload*, bool *clrTrig*,  
bool *intA*, bool *intB*, uint8\_t *width*, uint8\_t *srcInc*, uint8\_t *dstInc*, uint32\_t  
*bytes* ) [inline], [static]**

Parameters

<i>reload</i>	true is reload link descriptor after current exhaust, false is not
<i>clrTrig</i>	true is clear trigger status, wait software trigger, false is not
<i>intA</i>	enable interruptA
<i>intB</i>	enable interruptB
<i>width</i>	transfer width
<i>srcInc</i>	source address interleave size
<i>dstInc</i>	destination address interleave size
<i>bytes</i>	transfer bytes

Returns

The value of xfer config

**23.7.15 uint32\_t DMA\_GetRemainingBytes ( DMA\_Type \* *base*, uint32\_t *channel* )**

Parameters

<i>base</i>	DMA peripheral base address.
-------------	------------------------------

<i>channel</i>	DMA channel number.
----------------	---------------------

Returns

The number of bytes which have not been transferred yet.

### 23.7.16 static void DMA\_SetChannelPriority ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_priority\_t *priority* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>priority</i>	Channel priority value.

### 23.7.17 static dma\_priority\_t DMA\_GetChannelPriority ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

Returns

Channel priority value.

### 23.7.18 static void DMA\_SetChannelConfigValid ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

**23.7.19 static void DMA\_DoChannelSoftwareTrigger ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

**23.7.20 static void DMA\_LoadChannelTransferConfig ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *xfer* ) [inline], [static]**

Parameters

<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.
<i>xfer</i>	transfer configurations.

**23.7.21 void DMA\_CreateDescriptor ( dma\_descriptor\_t \* *desc*, dma\_xfercfg\_t \* *xfercfg*, void \* *srcAddr*, void \* *dstAddr*, void \* *nextDesc* )**

**Deprecated** Do not use this function. It has been superceded by [DMA\\_SetupDescriptor](#).

Parameters

<i>desc</i>	DMA descriptor address.
<i>xfercfg</i>	Transfer configuration for DMA descriptor.
<i>srcAddr</i>	Address of last item to transmit

<i>dstAddr</i>	Address of last item to receive.
<i>nextDesc</i>	Address of next descriptor in chain.

### 23.7.22 void DMA\_SetupDescriptor ( *dma\_descriptor\_t \* desc, uint32\_t xfercfg, void \* srcStartAddr, void \* dstStartAddr, void \* nextDesc* )

Note: This function do not support configure wrap descriptor.

Parameters

<i>desc</i>	DMA descriptor address.
<i>xfercfg</i>	Transfer configuration for DMA descriptor.
<i>srcStartAddr</i>	Start address of source address.
<i>dstStartAddr</i>	Start address of destination address.
<i>nextDesc</i>	Address of next descriptor in chain.

### 23.7.23 void DMA\_SetupChannelDescriptor ( *dma\_descriptor\_t \* desc, uint32\_t xfercfg, void \* srcStartAddr, void \* dstStartAddr, void \* nextDesc, dma\_burst\_wrap\_t wrapType, uint32\_t burstSize* )

Note: This function support configure wrap descriptor.

Parameters

<i>desc</i>	DMA descriptor address.
<i>xfercfg</i>	Transfer configuration for DMA descriptor.
<i>srcStartAddr</i>	Start address of source address.
<i>dstStartAddr</i>	Start address of destination address.
<i>nextDesc</i>	Address of next descriptor in chain.
<i>wrapType</i>	burst wrap type.
<i>burstSize</i>	burst size, reference _dma_burst_size.

### 23.7.24 void DMA\_LoadChannelDescriptor ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_descriptor\_t \* *descriptor* )

This function can be used to load descriptor to driver internal channel descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the polling transfer, application can allocate a local descriptor memory table to prepare a descriptor firstly and then call this api to load the configured descriptor to driver descriptor table.

```
* DMA_Init(DMA0);
* DMA_EnableChannel(DMA0, DEMO_DMA_CHANNEL);
* DMA_SetupDescriptor(desc, xferCfg, s_srcBuffer, &s_destBuffer[0], NULL);
* DMA_LoadChannelDescriptor(DMA0, DEMO_DMA_CHANNEL, (
    dma_descriptor_t *)desc);
* DMA_DoChannelSoftwareTrigger(DMA0, DEMO_DMA_CHANNEL);
* while(DMA_ChannelIsBusy(DMA0, DEMO_DMA_CHANNEL))
* {}
*
```

Parameters

<i>base</i>	DMA base address.
<i>channel</i>	DMA channel.
<i>descriptor</i>	configured DMA descriptor.

### 23.7.25 void DMA\_AbortTransfer ( dma\_handle\_t \* *handle* )

This function aborts DMA transfer specified by handle.

Parameters

<i>handle</i>	DMA handle pointer.
---------------	---------------------

### 23.7.26 void DMA\_CreateHandle ( dma\_handle\_t \* *handle*, DMA\_Type \* *base*, uint32\_t *channel* )

This function is called if using transaction API for DMA. This function initializes the internal state of DMA handle.

Parameters

<i>handle</i>	DMA handle pointer. The DMA handle stores callback function and parameters.
<i>base</i>	DMA peripheral base address.
<i>channel</i>	DMA channel number.

### 23.7.27 void DMA\_SetCallback ( *dma\_handle\_t \* handle*, *dma\_callback callback*, *void \* userData* )

This callback is called in DMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

<i>handle</i>	DMA handle pointer.
<i>callback</i>	DMA callback function pointer.
<i>userData</i>	Parameter for callback function.

### 23.7.28 void DMA\_PrepTransfer ( *dma\_transfer\_config\_t \* config*, *void \* srcAddr*, *void \* dstAddr*, *uint32\_t byteWidth*, *uint32\_t transferBytes*, *dma\_transfer\_type\_t type*, *void \* nextDesc* )

**Deprecated** Do not use this function. It has been superceded by [DMA\\_PrepChannelTransfer](#). This function prepares the transfer configuration structure according to the user input.

Parameters

<i>config</i>	The user configuration structure of type <i>dma_transfer_t</i> .
<i>srcAddr</i>	DMA transfer source address.
<i>dstAddr</i>	DMA transfer destination address.
<i>byteWidth</i>	DMA transfer destination address width(bytes).
<i>transferBytes</i>	DMA transfer bytes to be transferred.
<i>type</i>	DMA transfer type.
<i>nextDesc</i>	Chain custom descriptor to transfer.

Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, so the source address must be 4 bytes aligned, or it shall result in source address error(SAE).

23.7.29 **void DMA\_PrepChannelTransfer ( *dma\_channel\_config\_t \* config, void \* srcStartAddr, void \* dstStartAddr, uint32\_t xferCfg, dma\_transfer\_type\_t type, dma\_channel\_trigger\_t \* trigger, void \* nextDesc* )**

This function used to prepare channel transfer configurations.

Parameters

<i>config</i>	Pointer to DMA channel transfer configuration structure.
<i>srcStartAddr</i>	source start address.
<i>dstStartAddr</i>	destination start address.
<i>xferCfg</i>	xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
<i>type</i>	transfer type.
<i>trigger</i>	DMA channel trigger configurations.
<i>nextDesc</i>	address of next descriptor.

### 23.7.30 status\_t DMA\_SubmitTransfer ( *dma\_handle\_t \* handle*, *dma\_transfer\_config\_t \* config* )

**Deprecated** Do not use this function. It has been superceded by [DMA\\_SubmitChannelTransfer](#).

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

<i>handle</i>	DMA handle pointer.
<i>config</i>	Pointer to DMA transfer configuration structure.

Return values

<i>kStatus_DMA_Success</i>	It means submit transfer request succeed.
<i>kStatus_DMA_QueueFull</i>	It means TCD queue is full. Submit transfer request is not allowed.
<i>kStatus_DMA_Busy</i>	It means the given channel is busy, need to submit request later.

### 23.7.31 void DMA\_SubmitChannelTransferParameter ( *dma\_handle\_t \* handle*, *uint32\_t xferCfg*, *void \* srcStartAddr*, *void \* dstStartAddr*, *void \* nextDesc* )

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

- for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle, DMA_CHANNEL_XFER(reload,
    clrTrig, intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, NULL);
DMA_StartTransfer(handle)
*
```

- for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[3]);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
DMA_SetupDescriptor(nextDesc2, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle, DMA_CHANNEL_XFER(reload,
clrTrig, intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, nextDesc0);
DMA_StartTransfer(handle);
*
```

#### Parameters

<i>handle</i>	Pointer to DMA handle.
<i>xferCfg</i>	xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
<i>srcStartAddr</i>	source start address.
<i>dstStartAddr</i>	destination start address.
<i>nextDesc</i>	address of next descriptor.

### 23.7.32 void DMA\_SubmitChannelDescriptor ( **dma\_handle\_t \* handle,** **dma\_descriptor\_t \* descriptor** )

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, this functiono is typical for the ping pong case:

- for the ping pong case, application should responsible for the descriptor, for example, application should prepare two descriptor table with macro.

```

define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[2]);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
    srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
    srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelDescriptor(handle, nextDesc0);
DMA_StartTransfer(handle);

*

```

## Parameters

<i>handle</i>	Pointer to DMA handle.
<i>descriptor</i>	descriptor to submit.

### 23.7.33 status\_t DMA\_SubmitChannelTransfer ( **dma\_handle\_t \* handle,** **dma\_channel\_config\_t \* config** )

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time. It is used for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```

DMA_CreateHandle(handle, base, channel)
DMA_PreparesChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
    trigger,NULL);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)

*

```

2. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```

define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);
DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
    srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
    srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
DMA_SetupDescriptor(nextDesc2, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
    srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
DMA_CreateHandle(handle, base, channel)
DMA_PreparesChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
    trigger,nextDesc0);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)

*

```

3. for the ping pong case, application should responsible for link descriptor, for example, application should prepare two descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```

define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
    srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width,
    srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
DMA_CreateHandle(handle, base, channel)
DMA_PrepChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
    trigger,nextDesc0);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
*

```

#### Parameters

<i>handle</i>	DMA handle pointer.
<i>config</i>	Pointer to DMA transfer configuration structure.

#### Return values

<i>kStatus_DMA_Success</i>	It means submit transfer request succeed.
<i>kStatus_DMA_QueueFull</i>	It means TCD queue is full. Submit transfer request is not allowed.
<i>kStatus_DMA_Busy</i>	It means the given channel is busy, need to submit request later.

### 23.7.34 void DMA\_StartTransfer ( **dma\_handle\_t \* handle** )

This function enables the channel request. User can call this function after submitting the transfer request It will trigger transfer start with software trigger only when hardware trigger is not used.

#### Parameters

<i>handle</i>	DMA handle pointer.
---------------	---------------------

### 23.7.35 void DMA\_IRQHandler ( **DMA\_Type \* base** )

This function clears the channel major interrupt flag and call the callback function if it is not NULL.

### Parameters

<i>base</i>	DMA base address.
-------------	-------------------

# Chapter 24

## GPIO: General Purpose I/O

### 24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Purpose I/O (GPIO) module of MCUXpresso SDK devices.

### 24.2 Function groups

#### 24.2.1 Initialization and deinitialization

The function [GPIO\\_PinInit\(\)](#) initializes the GPIO with specified configuration.

#### 24.2.2 Pin manipulation

The function [GPIO\\_PinWrite\(\)](#) set output state of selected GPIO pin. The function [GPIO\\_PinRead\(\)](#) read input value of selected GPIO pin.

#### 24.2.3 Port manipulation

The function [GPIO\\_PortSet\(\)](#) sets the output level of selected GPIO pins to the logic 1. The function [GPIO\\_PortClear\(\)](#) sets the output level of selected GPIO pins to the logic 0. The function [GPIO\\_PortToggle\(\)](#) reverse the output level of selected GPIO pins. The function [GPIO\\_PortRead\(\)](#) read input value of selected port.

#### 24.2.4 Port masking

The function [GPIO\\_PortMaskedSet\(\)](#) set port mask, only pins masked by 0 will be enabled in following functions. The function [GPIO\\_PortMaskedWrite\(\)](#) sets the state of selected GPIO port, only pins masked by 0 will be affected. The function [GPIO\\_PortMaskedRead\(\)](#) reads the state of selected GPIO port, only pins masked by 0 are enabled for read, pins masked by 1 are read as 0.

### 24.3 Typical use case

Example use of GPIO API. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/gpio

## Files

- file [fsl\\_gpio.h](#)

## Data Structures

- struct [gpio\\_pin\\_config\\_t](#)  
*The GPIO pin configuration structure. [More...](#)*

## Enumerations

- enum [gpio\\_pin\\_direction\\_t](#) {
   
   [kGPIO\\_DigitalInput](#) = 0U,
   
   [kGPIO\\_DigitalOutput](#) = 1U
 }
   
*LPC GPIO direction definition.*

## Functions

- static void [GPIO\\_PortSet](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_PortClear](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_PortToggle](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Reverses current output logic of the multiple GPIO pins.*

## Driver version

- #define [FSL\\_GPIO\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 7))  
*LPC GPIO driver version.*

## GPIO Configuration

- void [GPIO\\_PortInit](#) (GPIO\_Type \*base, uint32\_t port)  
*Initializes the GPIO peripheral.*
- void [GPIO\\_PinInit](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin, const [gpio\\_pin\\_config\\_t](#) \*config)  
*Initializes a GPIO pin used by the board.*

## GPIO Output Operations

- static void [GPIO\\_PinWrite](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin, uint8\_t output)  
*Sets the output level of the one GPIO pin to the logic 1 or 0.*

## GPIO Input Operations

- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin)  
*Reads the current input value of the GPIO PIN.*

## 24.4 Data Structure Documentation

### 24.4.1 struct gpio\_pin\_config\_t

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

#### Data Fields

- `gpio_pin_direction_t pinDirection`  
*GPIO direction, input or output.*
- `uint8_t outputLogic`  
*Set default output logic, no use in input.*

## 24.5 Macro Definition Documentation

### 24.5.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 7))

## 24.6 Enumeration Type Documentation

### 24.6.1 enum gpio\_pin\_direction\_t

Enumerator

`kGPIO_DigitalInput` Set current pin as digital input.

`kGPIO_DigitalOutput` Set current pin as digital output.

## 24.7 Function Documentation

### 24.7.1 void GPIO\_PortInit ( `GPIO_Type * base`, `uint32_t port` )

This function ungates the GPIO clock.

Parameters

<code>base</code>	GPIO peripheral base pointer.
<code>port</code>	GPIO port number.

### 24.7.2 void GPIO\_PinInit ( `GPIO_Type * base`, `uint32_t port`, `uint32_t pin`, `const gpio_pin_config_t * config` )

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the `GPIO_PinInit()` function.

This is an example to define an input pin or output pin configuration:

```

* Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalOutput,
*   0,
* }
*

```

## Parameters

<i>base</i>	GPIO peripheral base pointer(Typically GPIO)
<i>port</i>	GPIO port number
<i>pin</i>	GPIO pin number
<i>config</i>	GPIO pin configuration pointer

#### 24.7.3 static void GPIO\_PinWrite ( `GPIO_Type` \* *base*, `uint32_t` *port*, `uint32_t` *pin*, `uint8_t` *output* ) [inline], [static]

## Parameters

<i>base</i>	GPIO peripheral base pointer(Typically GPIO)
<i>port</i>	GPIO port number
<i>pin</i>	GPIO pin number
<i>output</i>	GPIO pin output logic level. <ul style="list-style-type: none"><li>• 0: corresponding pin output low-logic level.</li><li>• 1: corresponding pin output high-logic level.</li></ul>

#### 24.7.4 static `uint32_t` GPIO\_PinRead ( `GPIO_Type` \* *base*, `uint32_t` *port*, `uint32_t` *pin* ) [inline], [static]

## Parameters

<i>base</i>	GPIO peripheral base pointer(Typically GPIO)
<i>port</i>	GPIO port number
<i>pin</i>	GPIO pin number

Return values

<i>GPIO</i>	port input value <ul style="list-style-type: none"> <li>• 0: corresponding pin input low-logic level.</li> <li>• 1: corresponding pin input high-logic level.</li> </ul>
-------------	--

#### 24.7.5 static void GPIO\_PortSet ( **GPIO\_Type** \* *base*, **uint32\_t** *port*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer(Typically GPIO)
<i>port</i>	GPIO port number
<i>mask</i>	GPIO pin number macro

#### 24.7.6 static void GPIO\_PortClear ( **GPIO\_Type** \* *base*, **uint32\_t** *port*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	GPIO peripheral base pointer(Typically GPIO)
<i>port</i>	GPIO port number
<i>mask</i>	GPIO pin number macro

#### 24.7.7 static void GPIO\_PortToggle ( **GPIO\_Type** \* *base*, **uint32\_t** *port*, **uint32\_t** *mask* ) [inline], [static]

## Parameters

<i>base</i>	GPIO peripheral base pointer(Typically GPIO)
<i>port</i>	GPIO port number
<i>mask</i>	GPIO pin number macro

# Chapter 25

## IOCON: I/O pin configuration

### 25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the I/O pin configuration (IOCON) module of MCUXpresso SDK devices.

### 25.2 Function groups

#### 25.2.1 Pin mux set

The function `IOCONPinMuxSet()` set pinmux for single pin according to selected configuration.

#### 25.2.2 Pin mux set

The function `IOCON_SetPinMuxing()` set pinmux for group of pins according to selected configuration.

### 25.3 Typical use case

Example use of IOCON API to selection of GPIO mode. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/iocon

## Files

- file `fsl_iocon.h`

## Data Structures

- struct `iocon_group_t`  
*Array of IOCON pin definitions passed to `IOCON_SetPinMuxing()` must be in this format. [More...](#)*

## Macros

- `#define IOCON_FUNC0 0x0`  
*IOCON function and mode selection definitions.*
- `#define IOCON_FUNC1 0x1`  
*Selects pin function 1.*
- `#define IOCON_FUNC2 0x2`  
*Selects pin function 2.*
- `#define IOCON_FUNC3 0x3`  
*Selects pin function 3.*
- `#define IOCON_FUNC4 0x4`  
*Selects pin function 4.*

- #define **IOCON\_FUNC5** 0x5  
*Selects pin function 5.*
- #define **IOCON\_FUNC6** 0x6  
*Selects pin function 6.*
- #define **IOCON\_FUNC7** 0x7  
*Selects pin function 7.*

## Functions

- \_\_STATIC\_INLINE void **IOCON\_PinMuxSet** (IOCON\_Type \*base, uint8\_t port, uint8\_t pin, uint32\_t modefunc)  
*Sets I/O Control pin mux.*
- \_\_STATIC\_INLINE void **IOCON\_SetPinMuxing** (IOCON\_Type \*base, const **iocon\_group\_t** \*pin-Array, uint32\_t arrayLength)  
*Set all I/O Control pin muxing.*

## Driver version

- #define **FSL\_IOCON\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 2, 0))  
*IOCON driver version.*

## 25.4 Data Structure Documentation

### 25.4.1 struct **iocon\_group\_t**

## 25.5 Macro Definition Documentation

### 25.5.1 #define **FSL\_IOCON\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 2, 0))

### 25.5.2 #define **IOCON\_FUNC0** 0x0

Note

See the User Manual for specific modes and functions supported by the various pins.*Selects pin function 0*

## 25.6 Function Documentation

### 25.6.1 \_\_STATIC\_INLINE void **IOCON\_PinMuxSet** ( **IOCON\_Type** \* *base*, **uint8\_t** *port*, **uint8\_t** *pin*, **uint32\_t** *modefunc* )

Parameters

---

<i>base</i>	: The base of IOCON peripheral on the chip
<i>port</i>	: GPIO port to mux
<i>pin</i>	: GPIO pin to mux
<i>modefunc</i>	: OR'ed values of type IOCON_*

Returns

Nothing

### 25.6.2 **`__STATIC_INLINE void IOCON_SetPinMuxing ( IOCON_Type * base, const iocon_group_t * pinArray, uint32_t arrayLength )`**

Parameters

<i>base</i>	: The base of IOCON peripheral on the chip
<i>pinArray</i>	: Pointer to array of pin mux selections
<i>arrayLength</i>	: Number of entries in pinArray

Returns

Nothing

# Chapter 26

## RTC: Real Time Clock

### 26.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC).

### 26.2 Function groups

The RTC driver supports operating the module as a time counter.

#### 26.2.1 Initialization and deinitialization

The function [RTC\\_Init\(\)](#) initializes the RTC with specified configurations. The function [RTC\\_GetDefaultConfig\(\)](#) gets the default configurations.

The function [RTC\\_Deinit\(\)](#) disables the RTC timer and disables the module clock.

#### 26.2.2 Set & Get Datetime

The function [RTC\\_SetDatetime\(\)](#) sets the timer period in seconds. User passes in the details in date & time format by using the below data structure.

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/rtc

The function [RTC\\_GetDatetime\(\)](#) reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

#### 26.2.3 Set & Get Alarm

The function [RTC\\_SetAlarm\(\)](#) sets the alarm time period in seconds. User passes in the details in date & time format by using the datetime data structure.

The function [RTC\\_GetAlarm\(\)](#) reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

#### 26.2.4 Start & Stop timer

The function [RTC\\_StartTimer\(\)](#) starts the RTC time counter.

The function [RTC\\_StopTimer\(\)](#) stops the RTC time counter.

## 26.2.5 Status

Provides functions to get and clear the RTC status.

## 26.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

## 26.2.7 High resolution timer

Provides functions to enable high resolution timer and set and get the wake time.

## 26.3 Typical use case

### 26.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/rtc

## Files

- file [fsl\\_rtc.h](#)

## Data Structures

- struct [rtc\\_datetime\\_t](#)  
*Structure is used to hold the date and time. [More...](#)*

## Enumerations

- enum [rtc\\_interrupt\\_enable\\_t](#) {
   
 kRTC\_AlarmInterruptEnable = RTC\_CTRL\_ALARMDPD\_EN\_MASK,
   
 kRTC\_WakeupInterruptEnable = RTC\_CTRL\_WAKEDPD\_EN\_MASK
 }
   
*List of RTC interrupts.*
- enum [rtc\\_status\\_flags\\_t](#) {
   
 kRTC\_AlarmFlag = RTC\_CTRL\_ALARM1HZ\_MASK,
   
 kRTC\_WakeupFlag = RTC\_CTRL\_WAKE1KHZ\_MASK
 }
   
*List of RTC flags.*

## Functions

- static void [RTC\\_SetSecondsTimerMatch](#) (RTC\_Type \*base, uint32\_t matchValue)
   
*Set the RTC seconds timer (1HZ) MATCH value.*
- static uint32\_t [RTC\\_GetSecondsTimerMatch](#) (RTC\_Type \*base)
   
*Read actual RTC seconds timer (1HZ) MATCH value.*

- static void **RTC\_SetSecondsTimerCount** (RTC\_Type \*base, uint32\_t countValue)  
*Set the RTC seconds timer (1HZ) COUNT value.*
- static uint32\_t **RTC\_GetSecondsTimerCount** (RTC\_Type \*base)  
*Read the actual RTC seconds timer (1HZ) COUNT value.*
- static void **RTC\_SetWakeupCount** (RTC\_Type \*base, uint16\_t wakeupValue)  
*Enable the RTC wake-up timer (1KHZ) and set countdown value to the RTC WAKE register.*
- static uint16\_t **RTC\_GetWakeupCount** (RTC\_Type \*base)  
*Read the actual value from the WAKE register value in RTC wake-up timer (1KHZ)*
- static void **RTC\_Reset** (RTC\_Type \*base)  
*Perform a software reset on the RTC module.*

## Driver version

- #define **FSL\_RTC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 3))  
*Version 2.1.3.*

## Initialization and deinitialization

- void **RTC\_Init** (RTC\_Type \*base)  
*Un-gate the RTC clock and enable the RTC oscillator.*
- static void **RTC\_Deinit** (RTC\_Type \*base)  
*Stop the timer and gate the RTC clock.*

## Current Time & Alarm

- status\_t **RTC\_SetDatetime** (RTC\_Type \*base, const **rtc\_datetime\_t** \*datetime)  
*Set the RTC date and time according to the given time structure.*
- void **RTC\_GetDatetime** (RTC\_Type \*base, **rtc\_datetime\_t** \*datetime)  
*Get the RTC time and stores it in the given time structure.*
- status\_t **RTC\_SetAlarm** (RTC\_Type \*base, const **rtc\_datetime\_t** \*alarmTime)  
*Set the RTC alarm time.*
- void **RTC\_GetAlarm** (RTC\_Type \*base, **rtc\_datetime\_t** \*datetime)  
*Return the RTC alarm time.*

## RTC wake-up timer (1KHZ) Enable

- static void **RTC\_EnableWakeupTimer** (RTC\_Type \*base, bool enable)  
*Enable the RTC wake-up timer (1KHZ).*
- static uint32\_t **RTC\_GetEnabledWakeupTimer** (RTC\_Type \*base)  
*Get the enabled status of the RTC wake-up timer (1KHZ).*

## Interrupt Interface

- static void **RTC\_EnableWakeUpTimerInterruptFromDPD** (RTC\_Type \*base, bool enable)  
*Enable the wake-up timer interrupt from deep power down mode.*
- static void **RTC\_EnableAlarmTimerInterruptFromDPD** (RTC\_Type \*base, bool enable)  
*Enable the alarm timer interrupt from deep power down mode.*
- static void **RTC\_EnableInterrupts** (RTC\_Type \*base, uint32\_t mask)  
*Enables the selected RTC interrupts.*
- static void **RTC\_DisableInterrupts** (RTC\_Type \*base, uint32\_t mask)

- Disables the selected RTC interrupts.
- static uint32\_t [RTC\\_GetEnabledInterrupts](#) (RTC\_Type \*base)  
Get the enabled RTC interrupts.

## Status Interface

- static uint32\_t [RTC\\_GetStatusFlags](#) (RTC\_Type \*base)  
Get the RTC status flags.
- static void [RTC\\_ClearStatusFlags](#) (RTC\_Type \*base, uint32\_t mask)  
Clear the RTC status flags.

## Timer Enable

- static void [RTC\\_EnableTimer](#) (RTC\_Type \*base, bool enable)  
Enable the RTC timer counter.
- static void [RTC\\_StartTimer](#) (RTC\_Type \*base)  
Starts the RTC time counter.
- static void [RTC\\_StopTimer](#) (RTC\_Type \*base)  
Stops the RTC time counter.

## 26.4 Data Structure Documentation

### 26.4.1 struct rtc\_datetime\_t

#### Data Fields

- uint16\_t [year](#)  
Range from 1970 to 2099.
- uint8\_t [month](#)  
Range from 1 to 12.
- uint8\_t [day](#)  
Range from 1 to 31 (depending on month).
- uint8\_t [hour](#)  
Range from 0 to 23.
- uint8\_t [minute](#)  
Range from 0 to 59.
- uint8\_t [second](#)  
Range from 0 to 59.

#### Field Documentation

- (1) [uint16\\_t rtc\\_datetime\\_t::year](#)
- (2) [uint8\\_t rtc\\_datetime\\_t::month](#)
- (3) [uint8\\_t rtc\\_datetime\\_t::day](#)
- (4) [uint8\\_t rtc\\_datetime\\_t::hour](#)
- (5) [uint8\\_t rtc\\_datetime\\_t::minute](#)

(6) `uint8_t rtc_datetime_t::second`

## 26.5 Enumeration Type Documentation

### 26.5.1 enum rtc\_interrupt\_enable\_t

Enumerator

*kRTC\_AlarmInterruptEnable* Alarm interrupt.  
*kRTC\_WakeupInterruptEnable* Wake-up interrupt.

### 26.5.2 enum rtc\_status\_flags\_t

Enumerator

*kRTC\_AlarmFlag* Alarm flag.  
*kRTC\_WakeupFlag* 1kHz wake-up timer flag

## 26.6 Function Documentation

### 26.6.1 void RTC\_Init ( RTC\_Type \* *base* )

Note

This API should be called at the beginning of the application using the RTC driver.

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

### 26.6.2 static void RTC\_Deinit ( RTC\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

### 26.6.3 status\_t RTC\_SetDatetime ( RTC\_Type \* *base*, const rtc\_datetime\_t \* *datetime* )

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to structure where the date and time details to set are stored

Returns

kStatus\_Success: Success in setting the time and starting the RTC  
kStatus\_InvalidArgument: Error because the datetime format is incorrect

#### 26.6.4 void RTC\_SetDatetime ( RTC\_Type \* *base*, rtc\_datetime\_t \* *datetime* )

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to structure where the date and time details are stored.

#### 26.6.5 status\_t RTC\_SetAlarm ( RTC\_Type \* *base*, const rtc\_datetime\_t \* *alarmTime* )

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

<i>base</i>	RTC peripheral base address
<i>alarmTime</i>	Pointer to structure where the alarm time is stored.

Returns

kStatus\_Success: success in setting the RTC alarm  
kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
kStatus\_Fail: Error because the alarm time has already passed

#### 26.6.6 void RTC\_GetAlarm ( RTC\_Type \* *base*, rtc\_datetime\_t \* *datetime* )

Parameters

<i>base</i>	RTC peripheral base address
<i>datetime</i>	Pointer to structure where the alarm date and time details are stored.

### 26.6.7 static void RTC\_EnableWakeupTimer ( RTC\_Type \* *base*, bool *enable* ) [inline], [static]

After calling this function, the RTC driver will use/un-use the RTC wake-up (1KHZ) at the same time.

Parameters

<i>base</i>	RTC peripheral base address
<i>enable</i>	Use/Un-use the RTC wake-up timer. <ul style="list-style-type: none"> <li>• true: Use RTC wake-up timer at the same time.</li> <li>• false: Un-use RTC wake-up timer, RTC only use the normal seconds timer by default.</li> </ul>

### 26.6.8 static uint32\_t RTC\_GetEnabledWakeupTimer ( RTC\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

Returns

The enabled status of RTC wake-up timer (1KHZ).

### 26.6.9 static void RTC\_SetSecondsTimerMatch ( RTC\_Type \* *base*, uint32\_t *matchValue* ) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
<i>matchValue</i>	The value to be set into the RTC MATCH register

**26.6.10 static uint32\_t RTC\_GetSecondsTimerMatch ( RTC\_Type \* *base* )  
[inline], [static]**

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

Returns

The actual RTC seconds timer (1HZ) MATCH value.

**26.6.11 static void RTC\_SetSecondsTimerCount ( RTC\_Type \* *base*, uint32\_t  
*countValue* ) [inline], [static]**

Parameters

<i>base</i>	RTC peripheral base address
<i>countValue</i>	The value to be loaded into the RTC COUNT register

**26.6.12 static uint32\_t RTC\_GetSecondsTimerCount ( RTC\_Type \* *base* )  
[inline], [static]**

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

Returns

The actual RTC seconds timer (1HZ) COUNT value.

**26.6.13 static void RTC\_SetWakeupCount ( RTC\_Type \* *base*, uint16\_t  
*wakeupValue* ) [inline], [static]**

Parameters

<i>base</i>	RTC peripheral base address
<i>wakeupValue</i>	The value to be loaded into the WAKE register in RTC wake-up timer (1KHZ).

#### 26.6.14 static uint16\_t RTC\_GetWakeupCount ( RTC\_Type \* *base* ) [inline], [static]

Read the WAKE register twice and compare the result, if the value match, the time can be used.

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

Returns

The actual value of the WAKE register value in RTC wake-up timer (1KHZ).

#### 26.6.15 static void RTC\_EnableWakeUpTimerInterruptFromDPD ( RTC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
<i>enable</i>	Enable/Disable wake-up timer interrupt from deep power down mode. <ul style="list-style-type: none"> <li>• true: Enable wake-up timer interrupt from deep power down mode.</li> <li>• false: Disable wake-up timer interrupt from deep power down mode.</li> </ul>

#### 26.6.16 static void RTC\_EnableAlarmTimerInterruptFromDPD ( RTC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
<i>enable</i>	Enable/Disable alarm timer interrupt from deep power down mode. <ul style="list-style-type: none"> <li>• true: Enable alarm timer interrupt from deep power down mode.</li> <li>• false: Disable alarm timer interrupt from deep power down mode.</li> </ul>

### 26.6.17 static void RTC\_EnableInterrupts ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

**Deprecated** Do not use this function. It has been superceded by [RTC\\_EnableAlarmTimerInterruptFromDPD](#) and [RTC\\_EnableWakeUpTimerInterruptFromDPD](#)

Parameters

<i>base</i>	RTC peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">rtc_interrupt_enable_t</a>

### 26.6.18 static void RTC\_DisableInterrupts ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

**Deprecated** Do not use this function. It has been superceded by [RTC\\_EnableAlarmTimerInterruptFromDPD](#) and [RTC\\_EnableWakeUpTimerInterruptFromDPD](#)

Parameters

<i>base</i>	RTC peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">rtc_interrupt_enable_t</a>

### 26.6.19 static uint32\_t RTC\_GetEnabledInterrupts ( RTC\_Type \* *base* ) [inline], [static]

**Deprecated** Do not use this function. It will be deleted in next release version.

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [rtc\\_interrupt\\_enable\\_t](#)

### 26.6.20 static uint32\_t RTC\_GetStatusFlags ( RTC\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

Returns

The status flags. This is the logical OR of members of the enumeration [rtc\\_status\\_flags\\_t](#)

### 26.6.21 static void RTC\_ClearStatusFlags ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	RTC peripheral base address
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration <a href="#">rtc_status_flags_t</a>

### 26.6.22 static void RTC\_EnableTimer ( RTC\_Type \* *base*, bool *enable* ) [inline], [static]

After calling this function, the RTC inner counter increments once a second when only using the RTC seconds timer (1hz), while the RTC inner wake-up timer countdown once a millisecond when using RTC wake-up timer (1KHZ) at the same time. RTC timer contain two timers, one is the RTC normal seconds timer, the other one is the RTC wake-up timer, the RTC enable bit is the master switch for the whole RTC timer, so user can use the RTC seconds (1HZ) timer independently, but they can't use the RTC wake-up timer (1KHZ) independently.

Parameters

<i>base</i>	RTC peripheral base address
<i>enable</i>	Enable/Disable RTC Timer counter. <ul style="list-style-type: none"> <li>• true: Enable RTC Timer counter.</li> <li>• false: Disable RTC Timer counter.</li> </ul>

### 26.6.23 static void RTC\_StartTimer( RTC\_Type \* *base* ) [inline], [static]

**Deprecated** Do not use this function. It has been superceded by [RTC\\_EnableTimer](#)

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

### 26.6.24 static void RTC\_StopTimer( RTC\_Type \* *base* ) [inline], [static]

**Deprecated** Do not use this function. It has been superceded by [RTC\\_EnableTimer](#)

RTC's seconds register can be written to only when the timer is stopped.

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

### 26.6.25 static void RTC\_Reset( RTC\_Type \* *base* ) [inline], [static]

This resets all RTC registers to their reset value. The bit is cleared by software explicitly clearing it.

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

# Chapter 27

## Mailbox

### 27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Mailbox module of MCUXpresso SDK devices.

The mailbox driver API provide:

- init/deinit: `MAILBOX_Init()`/`MAILBOX_Deinit()`
- read/write from/to mailbox register: `MAILBOX_SetValue()`/`MAILBOX_GetValue()`
- set/clear mailbox register bits: `MAILBOX_SetValueBits()`/`MAILBOX_ClearValueBits()`
- get/set mutex: `MAILBOX_GetMutex()`/`MAILBOX_SetMutex()`

### 27.2 Typical use case

**Example of code on primary core, which cause interrupt on secondary core by writing to mailbox register.**

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/mailbox

**Example of code on secondary core to handle interrupt from primary core.**

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/mailbox

**Example of code to get/set mutex.**

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/mailbox

### Files

- file `fsl_mailbox.h`

### Functions

- static uint32\_t `MAILBOX_GetMutex` (MAILBOX\_Type \*base)  
*Get MUTEX state and lock mutex.*
- static void `MAILBOX_SetMutex` (MAILBOX\_Type \*base)  
*Set MUTEX state.*

## Driver version

- #define **FSL\_MAILBOX\_DRIVER\_VERSION** (MAKE\_VERSION(2, 3, 0))  
*MAILBOX driver version 2.3.0.*

## MAILBOX initialization

CPU ID.

- static void **MAILBOX\_Init** (MAILBOX\_Type \*base)  
*Initializes the MAILBOX module.*
- static void **MAILBOX\_Deinit** (MAILBOX\_Type \*base)  
*De-initializes the MAILBOX module.*

### 27.3 Macro Definition Documentation

#### 27.3.1 #define FSL\_MAILBOX\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 0))

### 27.4 Function Documentation

#### 27.4.1 static void MAILBOX\_Init ( **MAILBOX\_Type \* base** ) [inline], [static]

This function enables the MAILBOX clock only.

Parameters

<i>base</i>	MAILBOX peripheral base address.
-------------	----------------------------------

#### 27.4.2 static void MAILBOX\_Deinit ( **MAILBOX\_Type \* base** ) [inline], [static]

This function disables the MAILBOX clock only.

Parameters

<i>base</i>	MAILBOX peripheral base address.
-------------	----------------------------------

#### 27.4.3 static uint32\_t MAILBOX\_GetMutex ( **MAILBOX\_Type \* base** ) [inline], [static]

Parameters

<i>base</i>	MAILBOX peripheral base address.
-------------	----------------------------------

Returns

See note

Note

Returns '1' if the mutex was taken or '0' if another resources has the mutex locked. Once a mutex is taken, it can be returned with the [MAILBOX\\_SetMutex\(\)](#) function.

#### 27.4.4 static void MAILBOX\_SetMutex ( MAILBOX\_Type \* *base* ) [inline], [static]

Parameters

<i>base</i>	MAILBOX peripheral base address.
-------------	----------------------------------

Note

Sets mutex state to '1' and allows other resources to get the mutex.

# Chapter 28

## MRT: Multi-Rate Timer

### 28.1 Overview

The MCUXpresso SDK provides a driver for the Multi-Rate Timer (MRT) of MCUXpresso SDK devices.

### 28.2 Function groups

The MRT driver supports operating the module as a time counter.

#### 28.2.1 Initialization and deinitialization

The function [MRT\\_Init\(\)](#) initializes the MRT with specified configurations. The function [MRT\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the MRT operating mode.

The function [MRT\\_Deinit\(\)](#) stops the MRT timers and disables the module clock.

#### 28.2.2 Timer period Operations

The function [MRT\\_UpdateTimerPeriod\(\)](#) is used to update the timer period in units of count. The new value is immediately loaded or will be loaded at the end of the current time interval.

The function [MRT\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. The user can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds

#### 28.2.3 Start and Stop timer operations

The function [MRT\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value, counts down to 0 and depending on the timer mode it either loads the respective start value again or stop. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function [MRT\\_StopTimer\(\)](#) stops the timer counting.

## 28.2.4 Get and release channel

These functions can be used to reserve and release a channel. The function [MRT\\_GetIdleChannel\(\)](#) finds the available channel. This function returns the lowest available channel number. The function [MRT\\_ReleaseChannel\(\)](#) release the channel when the timer is using the multi-task mode. In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use.

## 28.2.5 Status

Provides functions to get and clear the PIT status.

## 28.2.6 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 28.3 Typical use case

### 28.3.1 MRT tick example

Updates the MRT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/mrt

## Files

- file [fsl\\_mrt.h](#)

## Data Structures

- struct [mrt\\_config\\_t](#)  
*MRT configuration structure.* [More...](#)

## Enumerations

- enum [mrt\\_chnl\\_t](#) {
   
kMRT\_Channel\_0 = 0U,
   
kMRT\_Channel\_1,
   
kMRT\_Channel\_2,
   
kMRT\_Channel\_3 }
   
*List of MRT channels.*
- enum [mrt\\_timer\\_mode\\_t](#) {
   
kMRT\_RepeatMode = (0 << MRT\_CHANNEL\_CTRL\_MODE\_SHIFT),
   
kMRT\_OneShotMode = (1 << MRT\_CHANNEL\_CTRL\_MODE\_SHIFT),
   
kMRT\_OneShotStallMode = (2 << MRT\_CHANNEL\_CTRL\_MODE\_SHIFT) }
   
*List of MRT timer modes.*

- enum `mrt_interrupt_enable_t` { `kMRT_TimerInterruptEnable` = MRT\_CHANNEL\_CTRL\_INTERRUPT\_MASK }

*List of MRT interrupts.*

- enum `mrt_status_flags_t` {  
`kMRT_TimerInterruptFlag` = MRT\_CHANNEL\_STAT\_INFLAG\_MASK,  
`kMRT_TimerRunFlag` = MRT\_CHANNEL\_STAT\_RUN\_MASK }

*List of MRT status flags.*

## Driver version

- #define `FSL_MRT_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 3))  
*Version 2.0.3.*

## Initialization and deinitialization

- void `MRT_Init` (MRT\_Type \*base, const `mrt_config_t` \*config)  
*Ungates the MRT clock and configures the peripheral for basic operation.*
- void `MRT_Deinit` (MRT\_Type \*base)  
*Gate the MRT clock.*
- static void `MRT_GetDefaultConfig` (`mrt_config_t` \*config)  
*Fill in the MRT config struct with the default settings.*
- static void `MRT_SetupChannelMode` (MRT\_Type \*base, `mrt_chnl_t` channel, const `mrt_timer_mode_t` mode)  
*Sets up an MRT channel mode.*

## Interrupt Interface

- static void `MRT_EnableInterrupts` (MRT\_Type \*base, `mrt_chnl_t` channel, uint32\_t mask)  
*Enables the MRT interrupt.*
- static void `MRT_DisableInterrupts` (MRT\_Type \*base, `mrt_chnl_t` channel, uint32\_t mask)  
*Disables the selected MRT interrupt.*
- static uint32\_t `MRT_GetEnabledInterrupts` (MRT\_Type \*base, `mrt_chnl_t` channel)  
*Gets the enabled MRT interrupts.*

## Status Interface

- static uint32\_t `MRT_GetStatusFlags` (MRT\_Type \*base, `mrt_chnl_t` channel)  
*Gets the MRT status flags.*
- static void `MRT_ClearStatusFlags` (MRT\_Type \*base, `mrt_chnl_t` channel, uint32\_t mask)  
*Clears the MRT status flags.*

## Read and Write the timer period

- void `MRT_UpdateTimerPeriod` (MRT\_Type \*base, `mrt_chnl_t` channel, uint32\_t count, bool immediateLoad)  
*Used to update the timer period in units of count.*
- static uint32\_t `MRT_GetCurrentTimerCount` (MRT\_Type \*base, `mrt_chnl_t` channel)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [MRT\\_StartTimer](#) (MRT\_Type \*base, [mrt\\_chnl\\_t](#) channel, uint32\_t count)  
*Starts the timer counting.*
- static void [MRT\\_StopTimer](#) (MRT\_Type \*base, [mrt\\_chnl\\_t](#) channel)  
*Stops the timer counting.*

## Get & release channel

- static uint32\_t [MRT\\_GetIdleChannel](#) (MRT\_Type \*base)  
*Find the available channel.*
- static void [MRT\\_ReleaseChannel](#) (MRT\_Type \*base, [mrt\\_chnl\\_t](#) channel)  
*Release the channel when the timer is using the multi-task mode.*

## 28.4 Data Structure Documentation

### 28.4.1 struct mrt\_config\_t

This structure holds the configuration settings for the MRT peripheral. To initialize this structure to reasonable defaults, call the [MRT\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

## Data Fields

- bool [enableMultiTask](#)  
*true: Timers run in multi-task mode; false: Timers run in hardware status mode*

## 28.5 Enumeration Type Documentation

### 28.5.1 enum mrt\_chnl\_t

Enumerator

- kMRT\_Channel\_0*** MRT channel number 0.
- kMRT\_Channel\_1*** MRT channel number 1.
- kMRT\_Channel\_2*** MRT channel number 2.
- kMRT\_Channel\_3*** MRT channel number 3.

### 28.5.2 enum mrt\_timer\_mode\_t

Enumerator

- kMRT\_RepeatMode*** Repeat Interrupt mode.
- kMRT\_OneShotMode*** One-shot Interrupt mode.
- kMRT\_OneShotStallMode*** One-shot stall mode.

### 28.5.3 enum mrt\_interrupt\_enable\_t

Enumerator

*kMRT\_TimerInterruptEnable* Timer interrupt enable.

### 28.5.4 enum mrt\_status\_flags\_t

Enumerator

*kMRT\_TimerInterruptFlag* Timer interrupt flag.

*kMRT\_TimerRunFlag* Indicates state of the timer.

## 28.6 Function Documentation

### 28.6.1 void MRT\_Init ( MRT\_Type \* *base*, const mrt\_config\_t \* *config* )

Note

This API should be called at the beginning of the application using the MRT driver.

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>config</i>	Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MOD-CFG register, param config is useless.

### 28.6.2 void MRT\_Deinit ( MRT\_Type \* *base* )

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
-------------	--

### 28.6.3 static void MRT\_GetDefaultConfig ( mrt\_config\_t \* *config* ) [inline], [static]

The default values are:

```
*     config->enableMultiTask = false;
*
```

Parameters

<i>config</i>	Pointer to user's MRT config structure.
---------------	---

**28.6.4 static void MRT\_SetupChannelMode ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, const mrt\_timer\_mode\_t *mode* ) [inline], [static]**

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Channel that is being configured.
<i>mode</i>	Timer mode to use for the channel.

**28.6.5 static void MRT\_EnableInterrupts ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">mrt_interrupt_enable_t</a>

**28.6.6 static void MRT\_DisableInterrupts ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">mrt_interrupt_enable_t</a>

**28.6.7 static uint32\_t MRT\_GetEnabledInterrupts ( MRT\_Type \* *base*, mrt\_chnl\_t *channel* ) [inline], [static]**

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [mrt\\_interrupt\\_enable\\_t](#)

#### 28.6.8 static uint32\_t MRT\_GetStatusFlags ( MRT\_Type \* *base*, mrt\_chnl\_t *channel* ) [inline], [static]

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration [mrt\\_status\\_flags\\_t](#)

#### 28.6.9 static void MRT\_ClearStatusFlags ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration <a href="#">mrt_status_flags_t</a>

#### 28.6.10 void MRT\_UpdateTimerPeriod ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, uint32\_t *count*, bool *immediateLoad* )

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

## Note

User can call the utility macros provided in `fsl_common.h` to convert to ticks

## Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number
<i>count</i>	Timer period in units of ticks
<i>immediateLoad</i>	true: Load the new value immediately into the TIMER register; false: Load the new value at the end of current timer interval

### 28.6.11 static uint32\_t MRT\_GetCurrentTimerCount ( `MRT_Type * base`, `mrt_chnl_t channel` ) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

## Note

User can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec

## Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number

## Returns

Current timer counting value in ticks

### 28.6.12 static void MRT\_StartTimer ( `MRT_Type * base`, `mrt_chnl_t channel`, `uint32_t count` ) [inline], [static]

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

## Note

User can call the utility macros provided in `fsl_common.h` to convert to ticks

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number.
<i>count</i>	Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature.

### 28.6.13 static void MRT\_StopTimer ( MRT\_Type \* *base*, mrt\_chnl\_t *channel* ) [inline], [static]

This function stops the timer from counting.

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number.

### 28.6.14 static uint32\_t MRT\_GetIdleChannel ( MRT\_Type \* *base* ) [inline], [static]

This function returns the lowest available channel number.

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
-------------	--

### 28.6.15 static void MRT\_ReleaseChannel ( MRT\_Type \* *base*, mrt\_chnl\_t *channel* ) [inline], [static]

In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use. The user can hold on to a channel acquired by calling [MRT\\_GetIdleChannel\(\)](#) for as long as it is needed and release it by calling this function. This removes the need to ask for an available channel for every use.

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
<i>channel</i>	Timer channel number.

# Chapter 29

## OSTIMER: OS Event Timer Driver

### 29.1 Overview

The MCUXpresso SDK provides a peripheral driver for the OSTIMER module of MCUXpresso SDK devices. OSTIMER driver is created to help user to operate the OSTIMER module. The OSTIMER timer can be used as a low power timer. The APIs can be used to enable the OSTIMER module, initialize it and set the match time, get the current timer count. And the raw value in OS timer register is gray-code type, so both decimal and gray-code format API were added for users. OSTIMER can be used as a wake up source from low power mode.

### 29.2 Function groups

The OSTIMER driver supports operating the module as a time counter.

#### 29.2.1 Initialization and deinitialization

The [OSTIMER\\_Init\(\)](#) function will initialize the OSTIMER and enable the clock for OSTIMER. The [OSTIMER\\_Deinit\(\)](#) function will shut down the bus clock of OSTIMER.

#### 29.2.2 OSTIMER status

The function [OSTIMER\\_GetStatusFlags\(\)](#) will get the current status flag of OSTIMER. The function [OSTIMER\\_ClearStatusFlag\(\)](#) will help clear the status flags.

#### 29.2.3 OSTIMER set match value

For OSTIMER, allow users set the match in two ways, set match value with raw data(gray code) and set the match value with common data(decimal format). [OSTIMER\\_SetMatchRawValue\(\)](#) is used with gray code and [OSTIMER\\_SetMatchValue\(\)](#) is used together with decimal data.

#### 29.2.4 OSTIMER get timer count

The OSTIMER driver allow users to get the timer count in two ways, getting the gray code value by using [OSTIMER\\_GetCaptureRawValue\(\)](#) and getting the decimal data by using [OSTIMER\\_GetCurrentTimerValue\(\)](#).

## 29.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/ostimer/

## Files

- file `fsl_ostimer.h`

## Typedefs

- typedef void(\* `ostimer_callback_t`)(void)  
*ostimer callback function.*

## Enumerations

- enum `_ostimer_flags` { `kOSTIMER_MatchInterruptFlag` = (OSTIMER\_OSEVENT\_CTRL\_OSTIMER\_INTRFLAG\_MASK) }  
*OSTIMER status flags.*

## Driver version

- #define `FSL_OSTIMER_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 0)`)  
*OSTIMER driver version.*

## Initialization and deinitialization

- void `OSTIMER_Init` (OSTIMER\_Type \*base)  
*Initializes an OSTIMER by turning its bus clock on.*
- void `OSTIMER_Deinit` (OSTIMER\_Type \*base)  
*Deinitializes a OSTIMER instance.*
- `uint64_t OSTIMER_GrayToDecimal` (`uint64_t gray`)  
*Translate the value from gray-code to decimal.*
- `static uint64_t OSTIMER_DecimalToGray` (`uint64_t dec`)  
*Translate the value from decimal to gray-code.*
- `uint32_t OSTIMER_GetStatusFlags` (OSTIMER\_Type \*base)  
*Get OSTIMER status Flags.*
- `void OSTIMER_ClearStatusFlags` (OSTIMER\_Type \*base, `uint32_t mask`)  
*Clear Status Interrupt Flags.*
- `status_t OSTIMER_SetMatchRawValue` (OSTIMER\_Type \*base, `uint64_t count`, `ostimer_callback_t cb`)  
*Set the match raw value for OSTIMER.*
- `status_t OSTIMER_SetMatchValue` (OSTIMER\_Type \*base, `uint64_t count`, `ostimer_callback_t cb`)  
*Set the match value for OSTIMER.*
- `static void OSTIMER_SetMatchRegister` (OSTIMER\_Type \*base, `uint64_t value`)  
*Set value to OSTIMER MATCH register directly.*
- `static void OSTIMER_EnableMatchInterrupt` (OSTIMER\_Type \*base)  
*Enable the OSTIMER counter match interrupt.*
- `static void OSTIMER_DisableMatchInterrupt` (OSTIMER\_Type \*base)  
*Disable the OSTIMER counter match interrupt.*

- static uint64\_t **OSTIMER\_GetCurrentTimerRawValue** (OSTIMER\_Type \*base)  
*Get current timer raw count value from OSTIMER.*
- uint64\_t **OSTIMER\_GetCurrentTimerValue** (OSTIMER\_Type \*base)  
*Get current timer count value from OSTIMER.*
- static uint64\_t **OSTIMER\_GetCaptureRawValue** (OSTIMER\_Type \*base)  
*Get the capture value from OSTIMER.*
- uint64\_t **OSTIMER\_GetCaptureValue** (OSTIMER\_Type \*base)  
*Get the capture value from OSTIMER.*
- void **OSTIMER\_HandleIRQ** (OSTIMER\_Type \*base, **ostimer\_callback\_t** cb)  
*OS timer interrupt Service Handler.*

## 29.4 Macro Definition Documentation

### 29.4.1 #define FSL\_OSTIMER\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))

## 29.5 TypeDef Documentation

### 29.5.1 typedef void(\* ostimer\_callback\_t)(void)

## 29.6 Enumeration Type Documentation

### 29.6.1 enum \_ostimer\_flags

Enumerator

*kOSTIMER\_MatchInterruptFlag* Match interrupt flag bit, sets if the match value was reached.

## 29.7 Function Documentation

### 29.7.1 void OSTIMER\_Init ( OSTIMER\_Type \* *base* )

### 29.7.2 void OSTIMER\_Deinit ( OSTIMER\_Type \* *base* )

This function shuts down OSTIMER bus clock

Parameters

<i>base</i>	OSTIMER peripheral base address.
-------------	----------------------------------

### 29.7.3 uint64\_t OSTIMER\_GrayToDecimal ( uint64\_t *gray* )

Parameters

<i>gray</i>	The gray value input.
-------------	-----------------------

Returns

The decimal value.

#### 29.7.4 static uint64\_t OSTIMER\_DecimalToGray ( uint64\_t *dec* ) [inline], [static]

Parameters

<i>dec</i>	The decimal value.
------------	--------------------

Returns

The gray code of the input value.

#### 29.7.5 uint32\_t OSTIMER\_GetStatusFlags ( OSTIMER\_Type \* *base* )

This returns the status flag. Currently, only match interrupt flag can be got.

Parameters

<i>base</i>	OSTIMER peripheral base address.
-------------	----------------------------------

Returns

status register value

#### 29.7.6 void OSTIMER\_ClearStatusFlags ( OSTIMER\_Type \* *base*, uint32\_t *mask* )

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

<i>base</i>	OSTIMER peripheral base address.
<i>mask</i>	Clear bit mask.

Returns

none

### 29.7.7 status\_t OSTIMER\_SetMatchRawValue ( OSTIMER\_Type \* *base*, uint64\_t *count*, ostimer\_callback\_t *cb* )

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format is gray-code, if decimal data was desired, please using [OSTIMER\\_SetMatchValue\(\)](#).

Parameters

<i>base</i>	OSTIMER peripheral base address.
<i>count</i>	OSTIMER timer match value.(Value is gray-code format)
<i>cb</i>	OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

<i>kStatus_Success</i>	- Set match raw value and enable interrupt Successfully.
<i>kStatus_Fail</i>	- Set match raw value fail.

### 29.7.8 status\_t OSTIMER\_SetMatchValue ( OSTIMER\_Type \* *base*, uint64\_t *count*, ostimer\_callback\_t *cb* )

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

Parameters

<i>base</i>	OSTIMER peripheral base address.
<i>count</i>	OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code internally.)
<i>cb</i>	OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

<i>kStatus_Success</i>	- Set match value and enable interrupt Successfully.
<i>kStatus_Fail</i>	- Set match value fail.

### 29.7.9 static void OSTIMER\_SetMatchRegister ( **OSTIMER\_Type** \* *base*, **uint64\_t** *value* ) [inline], [static]

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function [OSTIMER\\_DecimalToGray](#) could convert decimal value to gray code.

Parameters

<i>base</i>	OSTIMER peripheral base address.
<i>count</i>	OSTIMER timer match value (Value is gray-code format).

### 29.7.10 static void OSTIMER\_EnableMatchInterrupt ( **OSTIMER\_Type** \* *base* ) [inline], [static]

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

<i>base</i>	OSTIMER peripheral base address.
-------------	----------------------------------

### 29.7.11 static void OSTIMER\_DisableMatchInterrupt ( **OSTIMER\_Type** \* *base* ) [inline], [static]

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

<i>base</i>	OSTIMER peripheral base address.
-------------	----------------------------------

### 29.7.12 static uint64\_t OSTIMER\_GetCurrentTimerRawValue ( OSTIMER\_Type \* *base* ) [inline], [static]

This function will get a gray code type timer count value from OS timer register. The raw value of timer count is gray code format.

Parameters

<i>base</i>	OSTIMER peripheral base address.
-------------	----------------------------------

Returns

Raw value of OSTIMER, gray code format.

### 29.7.13 uint64\_t OSTIMER\_GetCurrentTimerValue ( OSTIMER\_Type \* *base* )

This function will get a decimal timer count value. The RAW value of timer count is gray code format, will be translated to decimal data internally.

Parameters

<i>base</i>	OSTIMER peripheral base address.
-------------	----------------------------------

Returns

Value of OSTIMER which will be formated to decimal value.

### 29.7.14 static uint64\_t OSTIMER\_GetCaptureRawValue ( OSTIMER\_Type \* *base* ) [inline], [static]

This function will get a captured gray-code value from OSTIMER. The Raw value of timer capture is gray code format.

Parameters

<i>base</i>	OSTIMER peripheral base address.
-------------	----------------------------------

Returns

Raw value of capture register, data format is gray code.

### 29.7.15 `uint64_t OSTIMER_GetCaptureValue ( OSTIMER_Type * base )`

This function will get a capture decimal-value from OSTIMER. The RAW value of timer capture is gray code format, will be translated to decimal data internally.

Parameters

<i>base</i>	OSTIMER peripheral base address.
-------------	----------------------------------

Returns

Value of capture register, data format is decimal.

### 29.7.16 `void OSTIMER_HandleIRQ ( OSTIMER_Type * base, ostimer_callback_t cb )`

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in [OSTIMER\\_SetMatchValue\(\)](#)). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

<i>base</i>	OS timer peripheral base address.
<i>cb</i>	callback scheduled for this instance of OS timer

Returns

none

# Chapter 30

## PINT: Pin Interrupt and Pattern Match Driver

### 30.1 Overview

The MCUXpresso SDK provides a driver for the Pin Interrupt and Pattern match (PINT).

It can configure one or more pins to generate a pin interrupt when the pin or pattern match conditions are met. The pins do not have to be configured as gpio pins however they must be connected to PINT via INPUTMUX. Only the pin interrupt or pattern match function can be active for interrupt generation. If the pin interrupt function is enabled then the pattern match function can be used for wakeup via RXEV.

### 30.2 Pin Interrupt and Pattern match Driver operation

[PINT\\_PinInterruptConfig\(\)](#) function configures the pins for pin interrupt.

[PINT\\_PatternMatchConfig\(\)](#) function configures the pins for pattern match.

#### 30.2.1 Pin Interrupt use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/pint

#### 30.2.2 Pattern match use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/pint

### Files

- file [fsl\\_pint.h](#)

### TypeDefs

- `typedef void(* pint_cb_t )(pint_pin_int_t pinr, uint32_t pmatch_status)`  
*PINT Callback function.*

### Enumerations

- `enum pint_pin_enable_t {`  
  `kPINT_PinIntEnableNone = 0U,`  
  `kPINT_PinIntEnableRiseEdge = PINT_PIN_RISE_EDGE,`  
  `kPINT_PinIntEnableFallEdge = PINT_PIN_FALL_EDGE,`  
  `kPINT_PinIntEnableBothEdges = PINT_PIN_BOTH_EDGE,`  
  `kPINT_PinIntEnableLowLevel = PINT_PIN_LOW_LEVEL,`  
  `kPINT_PinIntEnableHighLevel = PINT_PIN_HIGH_LEVEL }`

*PINT Pin Interrupt enable type.*

- enum `pint_pin_int_t` {
   
    `kPINT_PinInt0` = 0U,  
`kPINT_PinInt1` = 1U,  
`kPINT_PinInt2` = 2U,  
`kPINT_PinInt3` = 3U,  
`kPINT_PinInt4` = 4U,  
`kPINT_PinInt5` = 5U,  
`kPINT_PinInt6` = 6U,  
`kPINT_PinInt7` = 7U }

*PINT Pin Interrupt type.*

- enum `pint_pmatch_input_src_t` {
   
    `kPINT_PatternMatchInp0Src` = 0U,  
`kPINT_PatternMatchInp1Src` = 1U,  
`kPINT_PatternMatchInp2Src` = 2U,  
`kPINT_PatternMatchInp3Src` = 3U,  
`kPINT_PatternMatchInp4Src` = 4U,  
`kPINT_PatternMatchInp5Src` = 5U,  
`kPINT_PatternMatchInp6Src` = 6U,  
`kPINT_PatternMatchInp7Src` = 7U,  
`kPINT_SecPatternMatchInp0Src` = 0U,  
`kPINT_SecPatternMatchInp1Src` = 1U }

*PINT Pattern Match bit slice input source type.*

- enum `pint_pmatch_bslice_t` {
   
    `kPINT_PatternMatchBSlice0` = 0U,  
`kPINT_PatternMatchBSlice1` = 1U,  
`kPINT_PatternMatchBSlice2` = 2U,  
`kPINT_PatternMatchBSlice3` = 3U,  
`kPINT_PatternMatchBSlice4` = 4U,  
`kPINT_PatternMatchBSlice5` = 5U,  
`kPINT_PatternMatchBSlice6` = 6U,  
`kPINT_PatternMatchBSlice7` = 7U }

*PINT Pattern Match bit slice type.*

- enum `pint_pmatch_bslice_cfg_t` {
   
    `kPINT_PatternMatchAlways` = 0U,  
`kPINT_PatternMatchStickyRise` = 1U,  
`kPINT_PatternMatchStickyFall` = 2U,  
`kPINT_PatternMatchStickyBothEdges` = 3U,  
`kPINT_PatternMatchHigh` = 4U,  
`kPINT_PatternMatchLow` = 5U,  
`kPINT_PatternMatchNever` = 6U,  
`kPINT_PatternMatchBothEdges` = 7U }

*PINT Pattern Match configuration type.*

## Functions

- void **PINT\_Init** (PINT\_Type \*base)  
*Initialize PINT peripheral.*
- void **PINT\_PinInterruptConfig** (PINT\_Type \*base, pint\_pin\_int\_t intr, pint\_pin\_enable\_t enable, pint\_cb\_t callback)  
*Configure PINT peripheral pin interrupt.*
- void **PINT\_PinInterruptGetConfig** (PINT\_Type \*base, pint\_pin\_int\_t pintr, pint\_pin\_enable\_t \*enable, pint\_cb\_t \*callback)  
*Get PINT peripheral pin interrupt configuration.*
- void **PINT\_PinInterruptClrStatus** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.*
- static uint32\_t **PINT\_PinInterruptGetStatus** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Get Selected pin interrupt status.*
- void **PINT\_PinInterruptClrStatusAll** (PINT\_Type \*base)  
*Clear all pin interrupts status only when pins were triggered by edge-sensitive.*
- static uint32\_t **PINT\_PinInterruptGetStatusAll** (PINT\_Type \*base)  
*Get all pin interrupts status.*
- static void **PINT\_PinInterruptClrFallFlag** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Clear Selected pin interrupt fall flag.*
- static uint32\_t **PINT\_PinInterruptGetFallFlag** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Get selected pin interrupt fall flag.*
- static void **PINT\_PinInterruptClrFallFlagAll** (PINT\_Type \*base)  
*Clear all pin interrupt fall flags.*
- static uint32\_t **PINT\_PinInterruptGetFallFlagAll** (PINT\_Type \*base)  
*Get all pin interrupt fall flags.*
- static void **PINT\_PinInterruptClrRiseFlag** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Clear Selected pin interrupt rise flag.*
- static uint32\_t **PINT\_PinInterruptGetRiseFlag** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Get selected pin interrupt rise flag.*
- static void **PINT\_PinInterruptClrRiseFlagAll** (PINT\_Type \*base)  
*Clear all pin interrupt rise flags.*
- static uint32\_t **PINT\_PinInterruptGetRiseFlagAll** (PINT\_Type \*base)  
*Get all pin interrupt rise flags.*
- void **PINT\_PatternMatchConfig** (PINT\_Type \*base, pint\_pmatch\_bslice\_t bslice, pint\_pmatch\_cfg\_t \*cfg)  
*Configure PINT pattern match.*
- void **PINT\_PatternMatchGetConfig** (PINT\_Type \*base, pint\_pmatch\_bslice\_t bslice, pint\_pmatch\_cfg\_t \*cfg)  
*Get PINT pattern match configuration.*
- static uint32\_t **PINT\_PatternMatchGetStatus** (PINT\_Type \*base, pint\_pmatch\_bslice\_t bslice)  
*Get pattern match bit slice status.*
- static uint32\_t **PINT\_PatternMatchGetStatusAll** (PINT\_Type \*base)  
*Get status of all pattern match bit slices.*
- uint32\_t **PINT\_PatternMatchResetDetectLogic** (PINT\_Type \*base)  
*Reset pattern match detection logic.*
- static void **PINT\_PatternMatchEnable** (PINT\_Type \*base)  
*Enable pattern match function.*
- static void **PINT\_PatternMatchDisable** (PINT\_Type \*base)  
*Disable pattern match function.*
- static void **PINT\_PatternMatchEnableRXEV** (PINT\_Type \*base)

- static void **PINT\_PatternMatchDisableRXEV** (PINT\_Type \*base)
  - Enable RXEV output.*
  - Disable RXEV output.*
- void **PINT\_EnableCallback** (PINT\_Type \*base)
  - Enable callback.*
- void **PINT\_DisableCallback** (PINT\_Type \*base)
  - Disable callback.*
- void **PINT\_Deinit** (PINT\_Type \*base)
  - Deinitialize PINT peripheral.*
- void **PINT\_EnableCallbackByIndex** (PINT\_Type \*base, **pint\_pin\_int\_t** pintIdx)
  - enable callback by pin index.*
- void **PINT\_DisableCallbackByIndex** (PINT\_Type \*base, **pint\_pin\_int\_t** pintIdx)
  - disable callback by pin index.*

## Driver version

- #define **FSL\_PINT\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 1, 11))

## 30.3 Typedef Documentation

### 30.3.1 **typedef void(\* pint\_cb\_t)(pint\_pin\_int\_t pintr, uint32\_t pmatch\_status)**

## 30.4 Enumeration Type Documentation

### 30.4.1 **enum pint\_pin\_enable\_t**

Enumerator

- kPINT\_PinIntEnableNone** Do not generate Pin Interrupt.
- kPINT\_PinIntEnableRiseEdge** Generate Pin Interrupt on rising edge.
- kPINT\_PinIntEnableFallEdge** Generate Pin Interrupt on falling edge.
- kPINT\_PinIntEnableBothEdges** Generate Pin Interrupt on both edges.
- kPINT\_PinIntEnableLowLevel** Generate Pin Interrupt on low level.
- kPINT\_PinIntEnableHighLevel** Generate Pin Interrupt on high level.

### 30.4.2 **enum pint\_pin\_int\_t**

Enumerator

- kPINT\_PinInt0** Pin Interrupt 0.
- kPINT\_PinInt1** Pin Interrupt 1.
- kPINT\_PinInt2** Pin Interrupt 2.
- kPINT\_PinInt3** Pin Interrupt 3.
- kPINT\_PinInt4** Pin Interrupt 4.
- kPINT\_PinInt5** Pin Interrupt 5.
- kPINT\_PinInt6** Pin Interrupt 6.
- kPINT\_PinInt7** Pin Interrupt 7.

### 30.4.3 enum pint\_pmatch\_input\_src\_t

Enumerator

<i>kPINT_PatternMatchInp0Src</i>	Input source 0.
<i>kPINT_PatternMatchInp1Src</i>	Input source 1.
<i>kPINT_PatternMatchInp2Src</i>	Input source 2.
<i>kPINT_PatternMatchInp3Src</i>	Input source 3.
<i>kPINT_PatternMatchInp4Src</i>	Input source 4.
<i>kPINT_PatternMatchInp5Src</i>	Input source 5.
<i>kPINT_PatternMatchInp6Src</i>	Input source 6.
<i>kPINT_PatternMatchInp7Src</i>	Input source 7.
<i>kPINT_SecPatternMatchInp0Src</i>	Input source 0.
<i>kPINT_SecPatternMatchInp1Src</i>	Input source 1.

### 30.4.4 enum pint\_pmatch\_bslice\_t

Enumerator

<i>kPINT_PatternMatchBSlice0</i>	Bit slice 0.
<i>kPINT_PatternMatchBSlice1</i>	Bit slice 1.
<i>kPINT_PatternMatchBSlice2</i>	Bit slice 2.
<i>kPINT_PatternMatchBSlice3</i>	Bit slice 3.
<i>kPINT_PatternMatchBSlice4</i>	Bit slice 4.
<i>kPINT_PatternMatchBSlice5</i>	Bit slice 5.
<i>kPINT_PatternMatchBSlice6</i>	Bit slice 6.
<i>kPINT_PatternMatchBSlice7</i>	Bit slice 7.

### 30.4.5 enum pint\_pmatch\_bslice\_cfg\_t

Enumerator

<i>kPINT_PatternMatchAlways</i>	Always Contributes to product term match.
<i>kPINT_PatternMatchStickyRise</i>	Sticky Rising edge.
<i>kPINT_PatternMatchStickyFall</i>	Sticky Falling edge.
<i>kPINT_PatternMatchStickyBothEdges</i>	Sticky Rising or Falling edge.
<i>kPINT_PatternMatchHigh</i>	High level.
<i>kPINT_PatternMatchLow</i>	Low level.
<i>kPINT_PatternMatchNever</i>	Never contributes to product term match.
<i>kPINT_PatternMatchBothEdges</i>	Either rising or falling edge.

## 30.5 Function Documentation

### 30.5.1 void PINT\_Init( PINT\_Type \* *base* )

This function initializes the PINT peripheral and enables the clock.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.2 void PINT\_PinInterruptConfig ( PINT\_Type \* *base*, pint\_pin\_int\_t *intr*, pint\_pin\_enable\_t *enable*, pint\_cb\_t *callback* )

This function configures a given pin interrupt.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>intr</i>	Pin interrupt.
<i>enable</i>	Selects detection logic.
<i>callback</i>	Callback.

Return values

<i>None.</i>
--------------

### 30.5.3 void PINT\_PinInterruptGetConfig ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr*, pint\_pin\_enable\_t \* *enable*, pint\_cb\_t \* *callback* )

This function returns the configuration of a given pin interrupt.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>pintr</i>	Pin interrupt.
<i>enable</i>	Pointer to store the detection logic.

<i>callback</i>	Callback.
-----------------	-----------

Return values

<i>None.</i>
--------------

### 30.5.4 void PINT\_PinInterruptClrStatus ( **PINT\_Type** \* *base*, **pint\_pin\_int\_t** *pintr* )

This function clears the selected pin interrupt status.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>pintr</i>	Pin interrupt.

Return values

<i>None.</i>
--------------

### 30.5.5 static uint32\_t PINT\_PinInterruptGetStatus ( **PINT\_Type** \* *base*, **pint\_pin\_int\_t** *pintr* ) [inline], [static]

This function returns the selected pin interrupt status.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>pintr</i>	Pin interrupt.

Return values

<i>status</i>	= 0 No pin interrupt request. = 1 Selected Pin interrupt request active.
---------------	--

### 30.5.6 void PINT\_PinInterruptClrStatusAll ( **PINT\_Type** \* *base* )

This function clears the status of all pin interrupts.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.7 static uint32\_t PINT\_PinInterruptGetStatusAll ( **PINT\_Type** \* *base* ) [**inline**], [**static**]

This function returns the status of all pin interrupts.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>status</i>	Each bit position indicates the status of corresponding pin interrupt. = 0 No pin interrupt request. = 1 Pin interrupt request active.
---------------	---

### 30.5.8 static void PINT\_PinInterruptClrFallFlag ( **PINT\_Type** \* *base*, **pint\_pin\_int\_t** *pintr* ) [**inline**], [**static**]

This function clears the selected pin interrupt fall flag.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>pintr</i>	Pin interrupt.

Return values

<i>None.</i>
--------------

### 30.5.9 static uint32\_t PINT\_PinInterruptGetFallFlag ( **PINT\_Type** \* *base*, **pint\_pin\_int\_t** *pintr* ) [**inline**], [**static**]

This function returns the selected pin interrupt fall flag.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>pintr</i>	Pin interrupt.

Return values

<i>flag</i>	= 0 Falling edge has not been detected. = 1 Falling edge has been detected.
-------------	---

### 30.5.10 static void PINT\_PinInterruptClrFallFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function clears the fall flag for all pin interrupts.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>	
--------------	--

### 30.5.11 static uint32\_t PINT\_PinInterruptGetFallFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the fall flag of all pin interrupts.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>flags</i>	Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.
--------------	--

**30.5.12 static void PINT\_PinInterruptClrRiseFlag ( PINT\_Type \* *base*,  
pint\_pin\_int\_t *pintr* ) [inline], [static]**

This function clears the selected pin interrupt rise flag.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>pintr</i>	Pin interrupt.

Return values

<i>None.</i>
--------------

### 30.5.13 static uint32\_t PINT\_PinInterruptGetRiseFlag ( **PINT\_Type** \* *base*, **pint\_pin\_int\_t** *pintr* ) [inline], [static]

This function returns the selected pin interrupt rise flag.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>pintr</i>	Pin interrupt.

Return values

<i>flag</i>	= 0 Rising edge has not been detected. = 1 Rising edge has been detected.
-------------	---

### 30.5.14 static void PINT\_PinInterruptClrRiseFlagAll ( **PINT\_Type** \* *base* ) [inline], [static]

This function clears the rise flag for all pin interrupts.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.15 static uint32\_t PINT\_PinInterruptGetRiseFlagAll ( **PINT\_Type** \* *base* ) [inline], [static]

This function returns the rise flag of all pin interrupts.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>flags</i>	Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.
--------------	---

### 30.5.16 void PINT\_PatternMatchConfig ( PINT\_Type \* *base*, pint\_pmatch\_bslice\_t *bslice*, pint\_pmatch\_cfg\_t \* *cfg* )

This function configures a given pattern match bit slice.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>bslice</i>	Pattern match bit slice number.
<i>cfg</i>	Pointer to bit slice configuration.

Return values

<i>None.</i>
--------------

### 30.5.17 void PINT\_PatternMatchGetConfig ( PINT\_Type \* *base*, pint\_pmatch\_bslice\_t *bslice*, pint\_pmatch\_cfg\_t \* *cfg* )

This function returns the configuration of a given pattern match bit slice.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>bslice</i>	Pattern match bit slice number.
<i>cfg</i>	Pointer to bit slice configuration.

Return values

<i>None.</i>
--------------

### 30.5.18 static uint32\_t PINT\_PatternMatchGetStatus ( PINT\_Type \* *base*,                   pint\_pmatch\_bslice\_t *bslice* ) [inline], [static]

This function returns the status of selected bit slice.

Parameters

<i>base</i>	Base address of the PINT peripheral.
<i>bslice</i>	Pattern match bit slice number.

Return values

<i>status</i>	= 0 Match has not been detected. = 1 Match has been detected.
---------------	---

### 30.5.19 static uint32\_t PINT\_PatternMatchGetStatusAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the status of all bit slices.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>status</i>	Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected.
---------------	---

### 30.5.20 uint32\_t PINT\_PatternMatchResetDetectLogic ( PINT\_Type \* *base* )

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>pmstatus</i>	Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected.
-----------------	--

### 30.5.21 static void PINT\_PatternMatchEnable ( PINT\_Type \* *base* ) [inline], [static]

This function enables the pattern match function.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.22 static void PINT\_PatternMatchDisable ( PINT\_Type \* *base* ) [inline], [static]

This function disables the pattern match function.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.23 static void PINT\_PatternMatchEnableRXEV ( PINT\_Type \* *base* ) [inline], [static]

This function enables the pattern match RXEV output.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.24 static void PINT\_PatternMatchDisableRXEV ( PINT\_Type \* *base* ) [inline], [static]

This function disables the pattern match RXEV output.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.25 void PINT\_EnableCallback ( PINT\_Type \* *base* )

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.26 void PINT\_DisableCallback ( PINT\_Type \* *base* )

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

<i>base</i>	Base address of the peripheral.
-------------	---------------------------------

Return values

<i>None.</i>
--------------

### 30.5.27 void PINT\_Deinit ( PINT\_Type \* *base* )

This function disables the PINT clock.

Parameters

<i>base</i>	Base address of the PINT peripheral.
-------------	--------------------------------------

Return values

<i>None.</i>
--------------

### 30.5.28 void PINT\_EnableCallbackByIndex ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintIdx* )

This function enables callback by pin index instead of enabling all pins.

Parameters

<i>base</i>	Base address of the peripheral.
<i>pintIdx</i>	pin index.

Return values

<i>None.</i>
--------------

### 30.5.29 void PINT\_DisableCallbackByIndex ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintIdx* )

This function disables callback by pin index instead of disabling all pins.

## Parameters

<i>base</i>	Base address of the peripheral.
<i>pintIdx</i>	pin index.

## Return values

<i>None.</i>	
--------------	--

# Chapter 31

## PLU: Programmable Logic Unit

### 31.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Programmable Logic Unit module of MCUXpresso SDK devices.

### 31.2 Function groups

The PLU driver supports the creation of small combinatorial and/or sequential logic networks including simple state machines.

#### 31.2.1 Initialization and de-initialization

The function [PLU\\_Init\(\)](#) enables the PLU clock and reset the module.

The function [PIT\\_Deinit\(\)](#) gates the PLU clock.

#### 31.2.2 Set input/output source and Truth Table

The function [PLU\\_SetLutInputSource\(\)](#) sets the input source for the LUT element.

The function [PLU\\_SetOutputSource\(\)](#) sets output source of the PLU module.

The function [PLU\\_SetLutTruthTable\(\)](#) sets the truth table for the LUT element.

#### 31.2.3 Read current Output State

The function [PLU\\_ReadOutputState\(\)](#) reads the current state of the 8 designated PLU Outputs.

#### 31.2.4 Wake-up/Interrupt Control

The function [PLU\\_EnableWakeIntRequest\(\)](#) enables the wake-up/interrupt request on a PLU output pin with a optional configuration to eliminate the glitches. The function [PLU\\_GetDefaultWakeIntConfig\(\)](#) gets the default configuration which can be used in a case with a given PLU\_CLKIN.

The function [PLU\\_LatchInterrupt\(\)](#) latches the interrupt and it can be cleared by function [PLU\\_ClearLatchedInterrupt\(\)](#).

### 31.3 Typical use case

#### 31.3.1 PLU combination example

Create a simple combinatorial logic network to control the LED. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/plu/combination

### Data Structures

- struct `plu_wakeint_config_t`  
*Wake configuration.* [More...](#)

### Enumerations

- enum `plu_lut_index_t` {
   
kPLU\_LUT\_0 = 0U,  
 kPLU\_LUT\_1 = 1U,  
 kPLU\_LUT\_2 = 2U,  
 kPLU\_LUT\_3 = 3U,  
 kPLU\_LUT\_4 = 4U,  
 kPLU\_LUT\_5 = 5U,  
 kPLU\_LUT\_6 = 6U,  
 kPLU\_LUT\_7 = 7U,  
 kPLU\_LUT\_8 = 8U,  
 kPLU\_LUT\_9 = 9U,  
 kPLU\_LUT\_10 = 10U,  
 kPLU\_LUT\_11 = 11U,  
 kPLU\_LUT\_12 = 12U,  
 kPLU\_LUT\_13 = 13U,  
 kPLU\_LUT\_14 = 14U,  
 kPLU\_LUT\_15 = 15U,  
 kPLU\_LUT\_16 = 16U,  
 kPLU\_LUT\_17 = 17U,  
 kPLU\_LUT\_18 = 18U,  
 kPLU\_LUT\_19 = 19U,  
 kPLU\_LUT\_20 = 20U,  
 kPLU\_LUT\_21 = 21U,  
 kPLU\_LUT\_22 = 22U,  
 kPLU\_LUT\_23 = 23U,  
 kPLU\_LUT\_24 = 24U,  
 kPLU\_LUT\_25 = 25U }

*Index of LUT.*

- enum `plu_lut_in_index_t` {
   
kPLU\_LUT\_IN\_0 = 0U,  
 kPLU\_LUT\_IN\_1 = 1U,  
 kPLU\_LUT\_IN\_2 = 2U,  
 kPLU\_LUT\_IN\_3 = 3U,

```
kPLU_LUT_IN_4 = 4U }
```

*Inputs of LUT.*

- enum `plu_lut_input_source_t` {
 

```
kPLU_LUT_IN_SRC_PLU_IN_0 = 0U,
kPLU_LUT_IN_SRC_PLU_IN_1 = 1U,
kPLU_LUT_IN_SRC_PLU_IN_2 = 2U,
kPLU_LUT_IN_SRC_PLU_IN_3 = 3U,
kPLU_LUT_IN_SRC_PLU_IN_4 = 4U,
kPLU_LUT_IN_SRC_PLU_IN_5 = 5U,
kPLU_LUT_IN_SRC_LUT_OUT_0 = 6U,
kPLU_LUT_IN_SRC_LUT_OUT_1 = 7U,
kPLU_LUT_IN_SRC_LUT_OUT_2 = 8U,
kPLU_LUT_IN_SRC_LUT_OUT_3 = 9U,
kPLU_LUT_IN_SRC_LUT_OUT_4 = 10U,
kPLU_LUT_IN_SRC_LUT_OUT_5 = 11U,
kPLU_LUT_IN_SRC_LUT_OUT_6 = 12U,
kPLU_LUT_IN_SRC_LUT_OUT_7 = 13U,
kPLU_LUT_IN_SRC_LUT_OUT_8 = 14U,
kPLU_LUT_IN_SRC_LUT_OUT_9 = 15U,
kPLU_LUT_IN_SRC_LUT_OUT_10 = 16U,
kPLU_LUT_IN_SRC_LUT_OUT_11 = 17U,
kPLU_LUT_IN_SRC_LUT_OUT_12 = 18U,
kPLU_LUT_IN_SRC_LUT_OUT_13 = 19U,
kPLU_LUT_IN_SRC_LUT_OUT_14 = 20U,
kPLU_LUT_IN_SRC_LUT_OUT_15 = 21U,
kPLU_LUT_IN_SRC_LUT_OUT_16 = 22U,
kPLU_LUT_IN_SRC_LUT_OUT_17 = 23U,
kPLU_LUT_IN_SRC_LUT_OUT_18 = 24U,
kPLU_LUT_IN_SRC_LUT_OUT_19 = 25U,
kPLU_LUT_IN_SRC_LUT_OUT_20 = 26U,
kPLU_LUT_IN_SRC_LUT_OUT_21 = 27U,
kPLU_LUT_IN_SRC_LUT_OUT_22 = 28U,
kPLU_LUT_IN_SRC_LUT_OUT_23 = 29U,
kPLU_LUT_IN_SRC_LUT_OUT_24 = 30U,
kPLU_LUT_IN_SRC_LUT_OUT_25 = 31U,
kPLU_LUT_IN_SRC_FLIPFLOP_0 = 32U,
kPLU_LUT_IN_SRC_FLIPFLOP_1 = 33U,
kPLU_LUT_IN_SRC_FLIPFLOP_2 = 34U,
kPLU_LUT_IN_SRC_FLIPFLOP_3 = 35U }
```

*Available sources of LUT input.*

- enum `plu_output_index_t` {

```
kPLU_OUTPUT_0 = 0U,
kPLU_OUTPUT_1 = 1U,
kPLU_OUTPUT_2 = 2U,
kPLU_OUTPUT_3 = 3U,
kPLU_OUTPUT_4 = 4U,
kPLU_OUTPUT_5 = 5U,
kPLU_OUTPUT_6 = 6U,
kPLU_OUTPUT_7 = 7U }
```

*PLU output multiplexer registers.*

- enum `plu_output_source_t` {
 

```
kPLU_OUT_SRC_LUT_0 = 0U,
kPLU_OUT_SRC_LUT_1 = 1U,
kPLU_OUT_SRC_LUT_2 = 2U,
kPLU_OUT_SRC_LUT_3 = 3U,
kPLU_OUT_SRC_LUT_4 = 4U,
kPLU_OUT_SRC_LUT_5 = 5U,
kPLU_OUT_SRC_LUT_6 = 6U,
kPLU_OUT_SRC_LUT_7 = 7U,
kPLU_OUT_SRC_LUT_8 = 8U,
kPLU_OUT_SRC_LUT_9 = 9U,
kPLU_OUT_SRC_LUT_10 = 10U,
kPLU_OUT_SRC_LUT_11 = 11U,
kPLU_OUT_SRC_LUT_12 = 12U,
kPLU_OUT_SRC_LUT_13 = 13U,
kPLU_OUT_SRC_LUT_14 = 14U,
kPLU_OUT_SRC_LUT_15 = 15U,
kPLU_OUT_SRC_LUT_16 = 16U,
kPLU_OUT_SRC_LUT_17 = 17U,
kPLU_OUT_SRC_LUT_18 = 18U,
kPLU_OUT_SRC_LUT_19 = 19U,
kPLU_OUT_SRC_LUT_20 = 20U,
kPLU_OUT_SRC_LUT_21 = 21U,
kPLU_OUT_SRC_LUT_22 = 22U,
kPLU_OUT_SRC_LUT_23 = 23U,
kPLU_OUT_SRC_LUT_24 = 24U,
kPLU_OUT_SRC_LUT_25 = 25U,
kPLU_OUT_SRC_FLIPFLOP_0 = 26U,
kPLU_OUT_SRC_FLIPFLOP_1 = 27U,
kPLU_OUT_SRC_FLIPFLOP_2 = 28U,
kPLU_OUT_SRC_FLIPFLOP_3 = 29U }
```

*Available sources of PLU output.*

- enum `_plu_interrupt_mask` {

```
kPLU_OUTPUT_0_INTERRUPT_MASK = 1 << 0,
kPLU_OUTPUT_1_INTERRUPT_MASK = 1 << 1,
kPLU_OUTPUT_2_INTERRUPT_MASK = 1 << 2,
kPLU_OUTPUT_3_INTERRUPT_MASK = 1 << 3,
kPLU_OUTPUT_4_INTERRUPT_MASK = 1 << 4,
kPLU_OUTPUT_5_INTERRUPT_MASK = 1 << 5,
kPLU_OUTPUT_6_INTERRUPT_MASK = 1 << 6,
kPLU_OUTPUT_7_INTERRUPT_MASK = 1 << 7 }
```

*The enumerator of PLU Interrupt.*

- enum `plu_wakeint_filter_mode_t` {
   
kPLU\_WAKEINT\_FILTER\_MODE\_BYPASS = 0U,
 kPLU\_WAKEINT\_FILTER\_MODE\_1\_CLK\_PERIOD = 1U,
 kPLU\_WAKEINT\_FILTER\_MODE\_2\_CLK\_PERIOD = 2U,
 kPLU\_WAKEINT\_FILTER\_MODE\_3\_CLK\_PERIOD = 3U }

*Control input of the PLU, add filtering for glitch.*

- enum `plu_wakeint_filter_clock_source_t` {
   
kPLU\_WAKEINT\_FILTER\_CLK\_SRC\_1MHZ\_LPOSC = 0U,
 kPLU\_WAKEINT\_FILTER\_CLK\_SRC\_12MHZ\_FRO = 1U,
 kPLU\_WAKEINT\_FILTER\_CLK\_SRC\_ALT = 2U }

*Clock source for filter mode.*

## Driver version

- #define `FSL_PLU_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 1))  
*Version 2.2.1.*

## Initialization and deinitialization

- void `PLU_Init` (PLU\_Type \*base)  
*Enable the PLU clock and reset the module.*
- void `PLU_Deinit` (PLU\_Type \*base)  
*Gate the PLU clock.*

## Set input/output source and Truth Table

- static void `PLU_SetLutInputSource` (PLU\_Type \*base, `plu_lut_index_t` lutIndex, `plu_lut_in_index_t` lutInIndex, `plu_lut_input_source_t` inputSrc)  
*Set Input source of LUT.*
- static void `PLU_SetOutputSource` (PLU\_Type \*base, `plu_output_index_t` outputIndex, `plu_output_source_t` outputSrc)  
*Set Output source of PLU.*
- static void `PLU_SetLutTruthTable` (PLU\_Type \*base, `plu_lut_index_t` lutIndex, uint32\_t truthTable)  
*Set Truth Table of LUT.*

## Read current Output State

- static uint32\_t `PLU_ReadOutputState` (PLU\_Type \*base)  
*Read the current state of the 8 designated PLU Outputs.*

## Wake-up/Interrupt Control

- void [PLU\\_GetDefaultWakeIntConfig](#) ([plu\\_wakeint\\_config\\_t](#) \*config)  
*Gets an available pre-defined settings for wakeup/interrupt control.*
- void [PLU\\_EnableWakeIntRequest](#) ([PLU\\_Type](#) \*base, [uint32\\_t](#) interruptMask, const [plu\\_wakeint\\_config\\_t](#) \*config)  
*Enable PLU outputs wakeup/interrupt request.*
- static void [PLU\\_LatchInterrupt](#) ([PLU\\_Type](#) \*base)  
*Latch an interrupt.*
- void [PLU\\_ClearLatchedInterrupt](#) ([PLU\\_Type](#) \*base)  
*Clear the latched interrupt.*

## 31.4 Data Structure Documentation

### 31.4.1 struct plu\_wakeint\_config\_t

#### Data Fields

- [plu\\_wakeint\\_filter\\_mode\\_t](#) filterMode  
*Filter Mode.*
- [plu\\_wakeint\\_filter\\_clock\\_source\\_t](#) clockSource  
*The clock source for filter mode.*

#### Field Documentation

- (1) [plu\\_wakeint\\_filter\\_mode\\_t plu\\_wakeint\\_config\\_t::filterMode](#)
- (2) [plu\\_wakeint\\_filter\\_clock\\_source\\_t plu\\_wakeint\\_config\\_t::clockSource](#)

## 31.5 Enumeration Type Documentation

### 31.5.1 enum plu\_lut\_index\_t

Enumerator

- kPLU\_LUT\_0*** 5-input Look-up Table 0
- kPLU\_LUT\_1*** 5-input Look-up Table 1
- kPLU\_LUT\_2*** 5-input Look-up Table 2
- kPLU\_LUT\_3*** 5-input Look-up Table 3
- kPLU\_LUT\_4*** 5-input Look-up Table 4
- kPLU\_LUT\_5*** 5-input Look-up Table 5
- kPLU\_LUT\_6*** 5-input Look-up Table 6
- kPLU\_LUT\_7*** 5-input Look-up Table 7
- kPLU\_LUT\_8*** 5-input Look-up Table 8
- kPLU\_LUT\_9*** 5-input Look-up Table 9
- kPLU\_LUT\_10*** 5-input Look-up Table 10
- kPLU\_LUT\_11*** 5-input Look-up Table 11
- kPLU\_LUT\_12*** 5-input Look-up Table 12
- kPLU\_LUT\_13*** 5-input Look-up Table 13

<i>kPLU_LUT_14</i>	5-input Look-up Table 14
<i>kPLU_LUT_15</i>	5-input Look-up Table 15
<i>kPLU_LUT_16</i>	5-input Look-up Table 16
<i>kPLU_LUT_17</i>	5-input Look-up Table 17
<i>kPLU_LUT_18</i>	5-input Look-up Table 18
<i>kPLU_LUT_19</i>	5-input Look-up Table 19
<i>kPLU_LUT_20</i>	5-input Look-up Table 20
<i>kPLU_LUT_21</i>	5-input Look-up Table 21
<i>kPLU_LUT_22</i>	5-input Look-up Table 22
<i>kPLU_LUT_23</i>	5-input Look-up Table 23
<i>kPLU_LUT_24</i>	5-input Look-up Table 24
<i>kPLU_LUT_25</i>	5-input Look-up Table 25

### 31.5.2 enum plu\_lut\_in\_index\_t

5 input present for each LUT.

Enumerator

<i>kPLU_LUT_IN_0</i>	LUT input 0.
<i>kPLU_LUT_IN_1</i>	LUT input 1.
<i>kPLU_LUT_IN_2</i>	LUT input 2.
<i>kPLU_LUT_IN_3</i>	LUT input 3.
<i>kPLU_LUT_IN_4</i>	LUT input 4.

### 31.5.3 enum plu\_lut\_input\_source\_t

Enumerator

<i>kPLU_LUT_IN_SRC_PLU_IN_0</i>	Select PLU input 0 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_PLU_IN_1</i>	Select PLU input 1 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_PLU_IN_2</i>	Select PLU input 2 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_PLU_IN_3</i>	Select PLU input 3 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_PLU_IN_4</i>	Select PLU input 4 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_PLU_IN_5</i>	Select PLU input 5 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_LUT_OUT_0</i>	Select LUT output 0 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_LUT_OUT_1</i>	Select LUT output 1 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_LUT_OUT_2</i>	Select LUT output 2 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_LUT_OUT_3</i>	Select LUT output 3 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_LUT_OUT_4</i>	Select LUT output 4 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_LUT_OUT_5</i>	Select LUT output 5 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_LUT_OUT_6</i>	Select LUT output 6 to be connected to LUTn Input x.
<i>kPLU_LUT_IN_SRC_LUT_OUT_7</i>	Select LUT output 7 to be connected to LUTn Input x.

***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_8*** Select LUT output 8 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_9*** Select LUT output 9 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_10*** Select LUT output 10 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_11*** Select LUT output 11 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_12*** Select LUT output 12 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_13*** Select LUT output 13 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_14*** Select LUT output 14 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_15*** Select LUT output 15 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_16*** Select LUT output 16 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_17*** Select LUT output 17 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_18*** Select LUT output 18 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_19*** Select LUT output 19 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_20*** Select LUT output 20 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_21*** Select LUT output 21 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_22*** Select LUT output 22 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_23*** Select LUT output 23 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_24*** Select LUT output 24 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_25*** Select LUT output 25 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_FLIPFLOP\_0*** Select Flip-Flops state 0 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_FLIPFLOP\_1*** Select Flip-Flops state 1 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_FLIPFLOP\_2*** Select Flip-Flops state 2 to be connected to LUTn Input x.  
***kPLU\_LUT\_IN\_SRC\_FLIPFLOP\_3*** Select Flip-Flops state 3 to be connected to LUTn Input x.

### 31.5.4 enum plu\_output\_index\_t

Enumerator

***kPLU\_OUTPUT\_0*** PLU OUTPUT 0.  
***kPLU\_OUTPUT\_1*** PLU OUTPUT 1.  
***kPLU\_OUTPUT\_2*** PLU OUTPUT 2.  
***kPLU\_OUTPUT\_3*** PLU OUTPUT 3.  
***kPLU\_OUTPUT\_4*** PLU OUTPUT 4.  
***kPLU\_OUTPUT\_5*** PLU OUTPUT 5.  
***kPLU\_OUTPUT\_6*** PLU OUTPUT 6.  
***kPLU\_OUTPUT\_7*** PLU OUTPUT 7.

### 31.5.5 enum plu\_output\_source\_t

Enumerator

***kPLU\_OUT\_SRC\_LUT\_0*** Select LUT0 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_1*** Select LUT1 output to be connected to PLU output.

<i>kPLU_OUT_SRC_LUT_2</i>	Select LUT2 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_3</i>	Select LUT3 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_4</i>	Select LUT4 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_5</i>	Select LUT5 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_6</i>	Select LUT6 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_7</i>	Select LUT7 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_8</i>	Select LUT8 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_9</i>	Select LUT9 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_10</i>	Select LUT10 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_11</i>	Select LUT11 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_12</i>	Select LUT12 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_13</i>	Select LUT13 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_14</i>	Select LUT14 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_15</i>	Select LUT15 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_16</i>	Select LUT16 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_17</i>	Select LUT17 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_18</i>	Select LUT18 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_19</i>	Select LUT19 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_20</i>	Select LUT20 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_21</i>	Select LUT21 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_22</i>	Select LUT22 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_23</i>	Select LUT23 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_24</i>	Select LUT24 output to be connected to PLU output.
<i>kPLU_OUT_SRC_LUT_25</i>	Select LUT25 output to be connected to PLU output.
<i>kPLU_OUT_SRC_FLIPFLOP_0</i>	Select Flip-Flops state(0) to be connected to PLU output.
<i>kPLU_OUT_SRC_FLIPFLOP_1</i>	Select Flip-Flops state(1) to be connected to PLU output.
<i>kPLU_OUT_SRC_FLIPFLOP_2</i>	Select Flip-Flops state(2) to be connected to PLU output.
<i>kPLU_OUT_SRC_FLIPFLOP_3</i>	Select Flip-Flops state(3) to be connected to PLU output.

### 31.5.6 enum \_plu\_interrupt\_mask

Enumerator

<i>kPLU_OUTPUT_0_INTERRUPT_MASK</i>	Select PLU output 0 contribute to interrupt/wake-up generation.
<i>kPLU_OUTPUT_1_INTERRUPT_MASK</i>	Select PLU output 1 contribute to interrupt/wake-up generation.
<i>kPLU_OUTPUT_2_INTERRUPT_MASK</i>	Select PLU output 2 contribute to interrupt/wake-up generation.
<i>kPLU_OUTPUT_3_INTERRUPT_MASK</i>	Select PLU output 3 contribute to interrupt/wake-up generation.
<i>kPLU_OUTPUT_4_INTERRUPT_MASK</i>	Select PLU output 4 contribute to interrupt/wake-up generation.

***kPLU\_OUTPUT\_5\_INTERRUPT\_MASK*** Select PLU output 5 contribute to interrupt/wake-up generation.

***kPLU\_OUTPUT\_6\_INTERRUPT\_MASK*** Select PLU output 6 contribute to interrupt/wake-up generation.

***kPLU\_OUTPUT\_7\_INTERRUPT\_MASK*** Select PLU output 7 contribute to interrupt/wake-up generation.

### 31.5.7 enum plu\_wakeint\_filter\_mode\_t

Enumerator

***kPLU\_WAKEINT\_FILTER\_MODE\_BYPASS*** Select Bypass mode.

***kPLU\_WAKEINT\_FILTER\_MODE\_1\_CLK\_PERIOD*** Filter 1 clock period.

***kPLU\_WAKEINT\_FILTER\_MODE\_2\_CLK\_PERIOD*** Filter 2 clock period.

***kPLU\_WAKEINT\_FILTER\_MODE\_3\_CLK\_PERIOD*** Filter 3 clock period.

### 31.5.8 enum plu\_wakeint\_filter\_clock\_source\_t

Enumerator

***kPLU\_WAKEINT\_FILTER\_CLK\_SRC\_1MHZ\_LPOSC*** Select the 1MHz low-power oscillator as the filter clock.

***kPLU\_WAKEINT\_FILTER\_CLK\_SRC\_12MHZ\_FRO*** Select the 12MHz FRO as the filer clock.

***kPLU\_WAKEINT\_FILTER\_CLK\_SRC\_ALT*** Select a third clock source.

## 31.6 Function Documentation

### 31.6.1 void PLU\_Init ( PLU\_Type \* *base* )

Note

This API should be called at the beginning of the application using the PLU driver.

Parameters

<i>base</i>	PLU peripheral base address
-------------	-----------------------------

### 31.6.2 void PLU\_Deinit ( PLU\_Type \* *base* )

Parameters

<i>base</i>	PLU peripheral base address
-------------	-----------------------------

### 31.6.3 static void PLU\_SetLutInputSource ( **PLU\_Type** \* *base*, **plu\_lut\_index\_t** *lutIndex*, **plu\_lut\_in\_index\_t** *lutInIndex*, **plu\_lut\_input\_source\_t** *inputSrc* ) [inline], [static]

Note: An external clock must be applied to the PLU\_CLKIN input when using FFs. For each LUT, the slot associated with the output from LUTn itself is tied low.

Parameters

<i>base</i>	PLU peripheral base address.
<i>lutIndex</i>	LUT index (see <a href="#">plu_lut_index_t</a> typedef enumeration).
<i>lutInIndex</i>	LUT input index (see <a href="#">plu_lut_in_index_t</a> typedef enumeration).
<i>inputSrc</i>	LUT input source (see <a href="#">plu_lut_input_source_t</a> typedef enumeration).

### 31.6.4 static void PLU\_SetOutputSource ( **PLU\_Type** \* *base*, **plu\_output\_index\_t** *outputIndex*, **plu\_output\_source\_t** *outputSrc* ) [inline], [static]

Note: An external clock must be applied to the PLU\_CLKIN input when using FFs.

Parameters

<i>base</i>	PLU peripheral base address.
<i>outputIndex</i>	PLU output index (see <a href="#">plu_output_index_t</a> typedef enumeration).
<i>outputSrc</i>	PLU output source (see <a href="#">plu_output_source_t</a> typedef enumeration).

### 31.6.5 static void PLU\_SetLutTruthTable ( **PLU\_Type** \* *base*, **plu\_lut\_index\_t** *lutIndex*, **uint32\_t** *truthTable* ) [inline], [static]

Parameters

<i>base</i>	PLU peripheral base address.
<i>lutIndex</i>	LUT index (see <a href="#">plu_lut_index_t</a> typedef enumeration).
<i>truthTable</i>	Truth Table value.

### 31.6.6 static uint32\_t PLU\_ReadOutputState ( PLU\_Type \* *base* ) [inline], [static]

Note: The PLU bus clock must be re-enabled prior to reading the Outpus Register if PLU bus clock is shut-off.

Parameters

<i>base</i>	PLU peripheral base address.
-------------	------------------------------

Returns

Current PLU output state value.

### 31.6.7 void PLU\_GetDefaultWakeIntConfig ( plu\_wakeint\_config\_t \* *config* )

This function initializes the initial configuration structure with an available settings. The default values are:

```
*     config->filterMode = kPLU_WAKEINT_FILTER_MODE_BYPASS;
*     config->clockSource = kPLU_WAKEINT_FILTER_CLK_SRC_1MHZ_LPOSC;
*
```

Parameters

<i>config</i>	Pointer to configuration structure.
---------------	-------------------------------------

### 31.6.8 void PLU\_EnableWakeIntRequest ( PLU\_Type \* *base*, uint32\_t *interruptMask*, const plu\_wakeint\_config\_t \* *config* )

This function enables Any of the eight selected PLU outputs to contribute to an asynchronous wake-up or an interrupt request.

Note: If a PLU\_CLKIN is provided, the raw wake-up/interrupt request will be set on the rising-edge of the PLU\_CLKIN whenever the raw request signal is high. This registered signal will be glitch-free and just use the default wakeint config by [PLU\\_GetDefaultWakeIntConfig\(\)](#). If not, have to specify the filter

mode and clock source to eliminate the glitches caused by long and widely disparate delays through the network of LUTs making up the PLU. This way may increase power consumption in low-power operating modes and inject delay before the wake-up/interrupt request is generated.

Parameters

<i>base</i>	PLU peripheral base address.
<i>interruptMask</i>	PLU interrupt mask (see <a href="#">_plu_interrupt_mask</a> enumeration).
<i>config</i>	Pointer to configuration structure (see <a href="#">plu_wakeint_config_t</a> typedef enumeration)

### 31.6.9 static void PLU\_LatchInterrupt ( PLU\_Type \* *base* ) [inline], [static]

This function latches the interrupt and then it can be cleared with [PLU\\_ClearLatchedInterrupt\(\)](#).

Note: This mode is not compatible with use of the glitch filter. If this bit is set, the FILTER MODE should be set to kPLU\_WAKEINT\_FILTER\_MODE\_BYPASS (Bypass Mode) and PLU\_CLKIN should be provided. If this bit is set, the wake-up/interrupt request will be set on the rising-edge of PLU\_CLKIN whenever the raw wake-up/interrupt signal is high. The request must be cleared by software.

Parameters

<i>base</i>	PLU peripheral base address.
-------------	------------------------------

### 31.6.10 void PLU\_ClearLatchedInterrupt ( PLU\_Type \* *base* )

This function clears the wake-up/interrupt request flag latched by [PLU\\_LatchInterrupt\(\)](#)

Note: It is not necessary for the PLU bus clock to be enabled in order to write-to or read-back this bit.

Parameters

<i>base</i>	PLU peripheral base address.
-------------	------------------------------

## Chapter 32

# POWERQUAD: PowerQuad hardware accelerator

### 32.1 Overview

The MCUXpresso SDK provides driver for the PowerQuad module of MCUXpresso SDK devices.

The PowerQuad hardware accelerator for (fixed/floating point/matrix operation) DSP functions is that idea is to replace some of the CMSIS DSP functionality with the hardware features provided by this IP.

PowerQuad driver provides the following CMSIS DSP compatible functions:

- Matrix functions

```
arm_mat_add_q15  
arm_mat_add_q31  
arm_mat_add_f32  
arm_mat_sub_q15  
arm_mat_sub_q31  
arm_mat_sub_f32  
arm_mat_mult_q15  
arm_mat_mult_q31  
arm_mat_mult_f32  
arm_mat_inverse_f32  
arm_mat_trans_q15  
arm_mat_trans_q31  
arm_mat_trans_f32  
arm_mat_scale_q15  
arm_mat_scale_q31  
arm_mat_scale_f32
```

- Math functions

```
arm_sqrt_q15  
arm_sqrt_q31  
arm_sin_q15  
arm_sin_q31  
arm_sin_f32  
arm_cos_q15  
arm_cos_q31  
arm_cos_f32
```

- Filter functions

```
arm_fir_q15  
arm_fir_q31  
arm_fir_f32  
arm_conv_q15  
arm_conv_q31  
arm_conv_f32  
arm_correlate_q15  
arm_correlate_q31  
arm_correlate_f32
```

- Transform functions

```
arm_rfft_q15  
arm_rfft_q31  
arm_cfft_q15
```

```
arm_cfft_q31
arm_ifft_q15
arm_ifft_q31
arm_dct4_q15
arm_dct4_q31
```

## Note

## CMSIS DSP compatible functions limitations

1. PowerQuad FFT engine only looks at the bottom 27 bits of the input word, down scale the input data to avoid saturation.
2. When use arm\_fir\_q15/arm\_fir\_q31/arm\_fir\_f32 for incremental, the new data should follow the old data. For example the array for input data is inputData[], and the array for output data is outputData[]. The first 32 input data is saved in inputData[0:31], after calling arm\_fir\_xxx(&fir, inputData, outputData, 32), the output data is saved in outputData[0:31]. The new input data must be saved from inputData[32], then call arm\_fir\_xxx(&fir, &inputData[32], &outputData[32], 32) for incremental calculation.

The PowerQuad consists of several internal computation engines: Transform engine, Transcendental function engine, Trigonometry function engine, Dual biquad IIR filter engine, Matrix accelerator engine, FIR filter engine, CORDIC engine.

For low level APIs, all function APIs, except using coprocessor instruction and arctan/arctanh API, need to calling wait done API to wait for calculation complete.

## 32.2 Function groups

### 32.2.1 POWERQUAD functional Operation

This group implements the POWERQUAD functional API.

#### Data Structures

- struct [pq\\_prescale\\_t](#)  
*Coprocessor prescale. [More...](#)*
- struct [pq\\_config\\_t](#)  
*powerquad data structure format [More...](#)*
- struct [pq\\_biquad\\_param\\_t](#)  
*Struct to save biquad parameters. [More...](#)*
- struct [pq\\_biquad\\_state\\_t](#)  
*Struct to save biquad state. [More...](#)*
- struct [pq\\_biquad\\_cascade\\_df2\\_instance](#)  
*Instance structure for the direct form II Biquad cascade filter. [More...](#)*
- union [pq\\_float\\_t](#)  
*Conversion between integer and float type. [More...](#)*

## Macros

- #define **PQ\_LN\_INF** PQ\_LN, 1, PQ\_TRANS  
*Parameter used for vector ln(x)*
- #define **PQ\_INV\_INF** PQ\_INV, 0, PQ\_TRANS  
*Parameter used for vector 1/x.*
- #define **PQ\_SQRT\_INF** PQ\_SQRT, 0, PQ\_TRANS  
*Parameter used for vector sqrt(x)*
- #define **PQ\_ISQRT\_INF** PQ\_INVSQRT, 0, PQ\_TRANS  
*Parameter used for vector 1/sqrt(x)*
- #define **PQETOX\_INF** PQETOX, 0, PQ\_TRANS  
*Parameter used for vector e^x.*
- #define **PQETONX\_INF** PQETONX, 0, PQ\_TRANS  
*Parameter used for vector e^(-x)*
- #define **PQ\_SIN\_INF** PQ\_SIN, 1, PQ\_TRIG  
*Parameter used for vector sin(x)*
- #define **PQ\_COS\_INF** PQ\_COS, 1, PQ\_TRIG  
*Parameter used for vector cos(x)*
- #define **PQ\_Initiate\_Vector\_Func**(pSrc, pDst)  
*Start 32-bit data vector calculation.*
- #define **PQ\_End\_Vector\_Func()** \_\_asm volatile("POP {r2-r7}")  
*End vector calculation.*
- #define **PQ\_StartVector**(PSRC, PDST, LENGTH)  
*Start 32-bit data vector calculation.*
- #define **PQ\_StartVectorFixed16**(PSRC, PDST, LENGTH)  
*Start 16-bit data vector calculation.*
- #define **PQ\_StartVectorQ15**(PSRC, PDST, LENGTH)  
*Start Q15-bit data vector calculation.*
- #define **PQ\_EndVector()** \_\_asm volatile("POP {r3-r10} \n")  
*End vector calculation.*
- #define **PQ\_Vector8F32**(BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)  
*Float data vector calculation.*
- #define **PQ\_Vector8Fixed32**(BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)  
*Fixed 32bits data vector calculation.*
- #define **PQ\_Vector8Fixed16**(BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)  
*Fixed 32bits data vector calculation.*
- #define **PQ\_Vector8Q15**(BATCH\_OPCODE, DOUBLE\_READ\_ADDERS, BATCH\_MACHINE)  
*Q15 data vector calculation.*
- #define **PQ\_DF2\_Vector8\_FP**(middle, last)  
*Float data vector biquad direct form II calculation.*
- #define **PQ\_DF2\_Vector8\_FX**(middle, last)  
*Fixed data vector biquad direct form II calculation.*
- #define **PQ\_Vector8BiquadDf2F32()**  
*Float data vector biquad direct form II calculation.*
- #define **PQ\_Vector8BiquadDf2Fixed32()**  
*Fixed 32-bit data vector biquad direct form II calculation.*
- #define **PQ\_Vector8BiquadDf2Fixed16()**  
*Fixed 16-bit data vector biquad direct form II calculation.*
- #define **PQ\_DF2\_Cascade\_Vector8\_FP**(middle, last)  
*Float data vector direct form II biquad cascade filter.*

- #define **PQ\_DF2\_Cascade\_Vector8\_FX**(middle, last)  
*Fixed data vector direct form II biquad cascade filter.*
- #define **PQ\_Vector8BiquadDf2CascadeF32()**  
*Float data vector direct form II biquad cascade filter.*
- #define **PQ\_Vector8BiquadDf2CascadeFixed32()**  
*Fixed 32-bit data vector direct form II biquad cascade filter.*
- #define **PQ\_Vector8BiquadDf2CascadeFixed16()**  
*Fixed 16-bit data vector direct form II biquad cascade filter.*
- #define **POWERQUAD\_MAKE\_MATRIX\_LEN**(mat1Row, mat1Col, mat2Col) (((uint32\_t)(mat1Row) << 0U) | ((uint32\_t)(mat1Col) << 8U) | ((uint32\_t)(mat2Col) << 16U))  
*Make the length used for matrix functions.*
- #define **PQ\_Q31\_2\_FLOAT**(x) (((float)(x)) / 2147483648.0f)  
*Convert Q31 to float.*
- #define **PQ\_Q15\_2\_FLOAT**(x) (((float)(x)) / 32768.0f)  
*Convert Q15 to float.*

## Enumerations

- enum **pq\_computationengine\_t** {
   
  kPQ\_CP\_PQ = 0,  
  kPQ\_CP\_mtx = 1,  
  kPQ\_CP\_FFT = 2,  
  kPQ\_CP\_FIR = 3,  
  kPQ\_CP\_CORDIC = 5 }
   
*powerquad computation engine*
- enum **pq\_format\_t** {
   
  kPQ\_16Bit = 0,  
  kPQ\_32Bit = 1,  
  kPQ\_Float = 2 }
   
*powerquad data structure format type*
- enum **pq\_cordic\_iter\_t** {
   
  kPQ\_Iteration\_8 = 0,  
  kPQ\_Iteration\_16,  
  kPQ\_Iteration\_24 }
   
*CORDIC iteration.*

## Driver version

- #define **FSL\_POWERQUAD\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 0))  
*Version.*

## POWERQUAD functional Operation

- void **PQ\_GetDefaultConfig** (**pq\_config\_t** \*config)  
*Get default configuration.*
- void **PQ\_SetConfig** (POWERQUAD\_Type \*base, const **pq\_config\_t** \*config)  
*Set configuration with format/prescale.*
- static void **PQ\_SetCoprocessorScaler** (POWERQUAD\_Type \*base, const **pq\_prescale\_t** \*prescale)

- set coprocessor scaler for coprocessor instructions, this function is used to set output saturation and scaling for input/output.*
- void [PQ\\_Init](#) (POWERQUAD\_Type \*base)  
*Initializes the POWERQUAD module.*
  - void [PQ\\_Deinit](#) (POWERQUAD\_Type \*base)  
*De-initializes the POWERQUAD module.*
  - void [PQ\\_SetFormat](#) (POWERQUAD\_Type \*base, pq\_computationengine\_t engine, pq\_format\_t format)  
*Set format for non-coprocessor instructions.*
  - static void [PQ\\_WaitDone](#) (POWERQUAD\_Type \*base)  
*Wait for the completion.*
  - static void [PQ\\_LnF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point natural log.*
  - static void [PQ\\_InvF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point reciprocal.*
  - static void [PQ\\_SqrtF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point square-root.*
  - static void [PQ\\_InvSqrtF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point inverse square-root.*
  - static void [PQ\\_EtoxF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point natural exponent.*
  - static void [PQ\\_EtonxF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point natural exponent with negative parameter.*
  - static void [PQ\\_SinF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point sine.*
  - static void [PQ\\_CosF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point cosine.*
  - static void [PQ\\_BiquadF32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point biquad.*
  - static void [PQ\\_DivF32](#) (float \*x1, float \*x2, float \*pDst)  
*Processing function for the floating-point division.*
  - static void [PQ\\_Biquad1F32](#) (float \*pSrc, float \*pDst)  
*Processing function for the floating-point biquad.*
  - static int32\_t [PQ\\_LnFixed](#) (int32\_t val)  
*Processing function for the fixed natural log.*
  - static int32\_t [PQ\\_InvFixed](#) (int32\_t val)  
*Processing function for the fixed reciprocal.*
  - static uint32\_t [PQ\\_SqrtFixed](#) (uint32\_t val)  
*Processing function for the fixed square-root.*
  - static int32\_t [PQ\\_InvSqrtFixed](#) (int32\_t val)  
*Processing function for the fixed inverse square-root.*
  - static int32\_t [PQ\\_EtoxFixed](#) (int32\_t val)  
*Processing function for the Fixed natural exponent.*
  - static int32\_t [PQ\\_EtonxFixed](#) (int32\_t val)  
*Processing function for the fixed natural exponent with negative parameter.*
  - static int32\_t [PQ\\_SinQ31](#) (int32\_t val)  
*Processing function for the fixed sine.*
  - static int16\_t [PQ\\_SinQ15](#) (int16\_t val)  
*Processing function for the fixed sine.*
  - static int32\_t [PQ\\_CosQ31](#) (int32\_t val)  
*Processing function for the fixed cosine.*

- static int16\_t **PQ\_CosQ15** (int16\_t val)  
*Processing function for the fixed sine.*
- static int32\_t **PQ\_BiquadFixed** (int32\_t val)  
*Processing function for the fixed biquad.*
- void **PQ\_VectorLnF32** (float \*pSrc, float \*pDst, int32\_t length)  
*Processing function for the floating-point vectorised natural log.*
- void **PQ\_VectorInvF32** (float \*pSrc, float \*pDst, int32\_t length)  
*Processing function for the floating-point vectorised reciprocal.*
- void **PQ\_VectorSqrtF32** (float \*pSrc, float \*pDst, int32\_t length)  
*Processing function for the floating-point vectorised square-root.*
- void **PQ\_VectorInvSqrtF32** (float \*pSrc, float \*pDst, int32\_t length)  
*Processing function for the floating-point vectorised inverse square-root.*
- void **PQ\_VectorEtoxF32** (float \*pSrc, float \*pDst, int32\_t length)  
*Processing function for the floating-point vectorised natural exponent.*
- void **PQ\_VectorEtonxF32** (float \*pSrc, float \*pDst, int32\_t length)  
*Processing function for the floating-point vectorised natural exponent with negative parameter.*
- void **PQ\_VectorSinF32** (float \*pSrc, float \*pDst, int32\_t length)  
*Processing function for the floating-point vectorised sine.*
- void **PQ\_VectorCosF32** (float \*pSrc, float \*pDst, int32\_t length)  
*Processing function for the floating-point vectorised cosine.*
- void **PQ\_VectorLnFixed32** (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)  
*Processing function for the Q31 vectorised natural log.*
- void **PQ\_VectorInvFixed32** (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)  
*Processing function for the Q31 vectorised reciprocal.*
- void **PQ\_VectorSqrtFixed32** (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)  
*Processing function for the 32-bit integer vectorised square-root.*
- void **PQ\_VectorInvSqrtFixed32** (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)  
*Processing function for the 32-bit integer vectorised inverse square-root.*
- void **PQ\_VectorEtoxFixed32** (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)  
*Processing function for the 32-bit integer vectorised natural exponent.*
- void **PQ\_VectorEtonxFixed32** (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)  
*Processing function for the 32-bit integer vectorised natural exponent with negative parameter.*
- void **PQ\_VectorSinQ15** (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)  
*Processing function for the Q15 vectorised sine.*
- void **PQ\_VectorCosQ15** (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)  
*Processing function for the Q15 vectorised cosine.*
- void **PQ\_VectorSinQ31** (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)  
*Processing function for the Q31 vectorised sine.*
- void **PQ\_VectorCosQ31** (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)  
*Processing function for the Q31 vectorised cosine.*
- void **PQ\_VectorLnFixed16** (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)  
*Processing function for the 16-bit integer vectorised natural log.*
- void **PQ\_VectorInvFixed16** (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)  
*Processing function for the 16-bit integer vectorised reciprocal.*
- void **PQ\_VectorSqrtFixed16** (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)  
*Processing function for the 16-bit integer vectorised square-root.*
- void **PQ\_VectorInvSqrtFixed16** (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)  
*Processing function for the 16-bit integer vectorised inverse square-root.*
- void **PQ\_VectorEtoxFixed16** (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)  
*Processing function for the 16-bit integer vectorised natural exponent.*
- void **PQ\_VectorEtonxFixed16** (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)

- void [PQ\\_VectorBiquadDf2F32](#) (float \*pSrc, float \*pDst, int32\_t length)
 

*Processing function for the 16-bit integer vectorised natural exponent with negative parameter.*
- void [PQ\\_VectorBiquadDf2Fixed32](#) (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)
 

*Processing function for the floating-point vectorised biquad direct form II.*
- void [PQ\\_VectorBiquadDf2Fixed16](#) (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)
 

*Processing function for the 32-bit integer vectorised biquad direct form II.*
- void [PQ\\_VectorBiquadCascadeDf2F32](#) (float \*pSrc, float \*pDst, int32\_t length)
 

*Processing function for the 16-bit integer vectorised biquad direct form II.*
- void [PQ\\_VectorBiquadCascadeDf2Fixed32](#) (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)
 

*Processing function for the floating-point vectorised biquad direct form II.*
- void [PQ\\_VectorBiquadCascadeDf2Fixed16](#) (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)
 

*Processing function for the 32-bit integer vectorised biquad direct form II.*
- int32\_t [PQ\\_ArctanFixed](#) (POWERQUAD\_Type \*base, int32\_t x, int32\_t y, [pq\\_cordic\\_iter\\_t](#) iteration)
 

*Processing function for the 16-bit integer vectorised biquad direct form II.*
- static int32\_t [PQ\\_ArctanhFixed](#) (POWERQUAD\_Type \*base, int32\_t x, int32\_t y, [pq\\_cordic\\_iter\\_t](#) iteration)
 

*Processing function for the fixed inverse trigonometric.*
- void [PQ\\_TransformCFFT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
 

*Processing function for the fixed inverse trigonometric.*
- void [PQ\\_TransformRFFT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
 

*Processing function for the fixed biquad.*
- void [PQ\\_TransformIFFT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
 

*Processing function for the complex FFT.*
- void [PQ\\_TransformCDCT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
 

*Processing function for the real FFT.*
- void [PQ\\_TransformIDCT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
 

*Processing function for the inverse complex FFT.*
- void [PQ\\_TransformRDCT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
 

*Processing function for the complex DCT.*
- void [PQ\\_TransformIDCT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
 

*Processing function for the real DCT.*
- void [PQ\\_BiquadBackUpInternalState](#) (POWERQUAD\_Type \*base, int32\_t biquad\_num, [pq\\_biquad\\_state\\_t](#) \*state)
 

*Processing function for the inverse complex DCT.*
- void [PQ\\_BiquadRestoreInternalState](#) (POWERQUAD\_Type \*base, int32\_t biquad\_num, [pq\\_biquad\\_state\\_t](#) \*state)
 

*Processing function for backup biquad context.*
- void [PQ\\_BiquadCascadeDf2Init](#) ([pq\\_biquad\\_cascade\\_df2\\_instance](#) \*S, uint8\_t numStages, [pq\\_biquad\\_state\\_t](#) \*pState)
 

*Processing function for restore biquad context.*
- void [PQ\\_BiquadCascadeDf2F32](#) (const [pq\\_biquad\\_cascade\\_df2\\_instance](#) \*S, float \*pSrc, float \*pDst, uint32\_t blockSize)
 

*Initialization function for the direct form II Biquad cascade filter.*

- Processing function for the floating-point direct form II Biquad cascade filter.
- void **PQ\_BiquadCascadeDf2Fixed32** (const **pq\_biquad\_cascade\_df2\_instance** \*S, int32\_t \*pSrc, int32\_t \*pDst, uint32\_t blockSize)
  - Processing function for the Q31 direct form II Biquad cascade filter.
- void **PQ\_BiquadCascadeDf2Fixed16** (const **pq\_biquad\_cascade\_df2\_instance** \*S, int16\_t \*pSrc, int16\_t \*pDst, uint32\_t blockSize)
  - Processing function for the Q15 direct form II Biquad cascade filter.
- void **PQ\_FIR** (POWERQUAD\_Type \*base, const void \*pAData, int32\_t ALength, const void \*pBData, int32\_t BLength, void \*pResult, uint32\_t opType)
  - Processing function for the FIR.
- void **PQ\_FIRIncrement** (POWERQUAD\_Type \*base, int32\_t ALength, int32\_t BLength, int32\_t xOffset)
  - Processing function for the incremental FIR.
- void **PQ\_MatrixAddition** (POWERQUAD\_Type \*base, uint32\_t length, void \*pAData, void \*pBData, void \*pResult)
  - Processing function for the matrix addition.
- void **PQ\_MatrixSubtraction** (POWERQUAD\_Type \*base, uint32\_t length, void \*pAData, void \*pBData, void \*pResult)
  - Processing function for the matrix subtraction.
- void **PQ\_MatrixMultiplication** (POWERQUAD\_Type \*base, uint32\_t length, void \*pAData, void \*pBData, void \*pResult)
  - Processing function for the matrix multiplication.
- void **PQ\_MatrixProduct** (POWERQUAD\_Type \*base, uint32\_t length, void \*pAData, void \*pBData, void \*pResult)
  - Processing function for the matrix product.
- void **PQ\_VectorDotProduct** (POWERQUAD\_Type \*base, uint32\_t length, void \*pAData, void \*pBData, void \*pResult)
  - Processing function for the vector dot product.
- void **PQ\_MatrixInversion** (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pTmpData, void \*pResult)
  - Processing function for the matrix inverse.
- void **PQ\_MatrixTranspose** (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
  - Processing function for the matrix transpose.
- void **PQ\_MatrixScale** (POWERQUAD\_Type \*base, uint32\_t length, float misc, const void \*pData, void \*pResult)
  - Processing function for the matrix scale.

## 32.3 Data Structure Documentation

### 32.3.1 struct pq\_prescale\_t

#### Data Fields

- int8\_t **inputPrescale**
  - Input prescale.*
- int8\_t **outputPrescale**
  - Output prescale.*
- int8\_t **outputSaturate**

*Output saturate at n bits, for example 0x11 is 8 bit space, the value will be truncated at +127 or -128.*

### Field Documentation

- (1) `int8_t pq_prescale_t::inputPrescale`
- (2) `int8_t pq_prescale_t::outputPrescale`
- (3) `int8_t pq_prescale_t::outputSaturate`

### 32.3.2 struct pq\_config\_t

#### Data Fields

- `pq_format_t inputAFormat`  
*Input A format.*
- `int8_t inputAPrescale`  
*Input A prescale, for example 1.5 can be  $1.5 * 2^n$  if you scale by 'shifting' ('scaling' by a factor of n).*
- `pq_format_t inputBFormat`  
*Input B format.*
- `int8_t inputBPrescale`  
*Input B prescale.*
- `pq_format_t outputFormat`  
*Out format.*
- `int8_t outputPrescale`  
*Out prescale.*
- `pq_format_t tmpFormat`  
*Temp format.*
- `int8_t tmpPrescale`  
*Temp prescale.*
- `pq_format_t machineFormat`  
*Machine format.*
- `uint32_t * tmpBase`  
*Tmp base address.*

### Field Documentation

- (1) `pq_format_t pq_config_t::inputAFormat`
- (2) `int8_t pq_config_t::inputAPrescale`
- (3) `pq_format_t pq_config_t::inputBFormat`
- (4) `int8_t pq_config_t::inputBPrescale`
- (5) `pq_format_t pq_config_t::outputFormat`
- (6) `int8_t pq_config_t::outputPrescale`
- (7) `pq_format_t pq_config_t::tmpFormat`

- (8) `int8_t pq_config_t::tmpPrescale`
- (9) `pq_format_t pq_config_t::machineFormat`
- (10) `uint32_t* pq_config_t::tmpBase`

### 32.3.3 struct pq\_biquad\_param\_t

#### Data Fields

- float `v_n_1`  
*v[n-1], set to 0 when initialization.*
- float `v_n`  
*v[n], set to 0 when initialization.*
- float `a_1`  
*a[1]*
- float `a_2`  
*a[2]*
- float `b_0`  
*b[0]*
- float `b_1`  
*b[1]*
- float `b_2`  
*b[2]*

#### Field Documentation

- (1) `float pq_biquad_param_t::v_n_1`
- (2) `float pq_biquad_param_t::v_n`

### 32.3.4 struct pq\_biquad\_state\_t

#### Data Fields

- `pq_biquad_param_t param`  
*Filter parameter.*
- `uint32_t compreg`  
*Internal register, set to 0 when initialization.*

#### Field Documentation

- (1) `pq_biquad_param_t pq_biquad_state_t::param`
- (2) `uint32_t pq_biquad_state_t::compreg`

### 32.3.5 struct pq\_biquad\_cascade\_df2\_instance

#### Data Fields

- uint8\_t numStages
- pq\_biquad\_state\_t \* pState

#### Field Documentation

(1) uint8\_t pq\_biquad\_cascade\_df2\_instance::numStages

Number of 2nd order stages in the filter.

(2) pq\_biquad\_state\_t\* pq\_biquad\_cascade\_df2\_instance::pState

Points to the array of state coefficients.

### 32.3.6 union pq\_float\_t

#### Data Fields

- float floatX  
*Float type.*
- uint32\_t integerX  
*Unsigned interger type.*

#### Field Documentation

(1) float pq\_float\_t::floatX

(2) uint32\_t pq\_float\_t::integerX

## 32.4 Macro Definition Documentation

### 32.4.1 #define FSL\_POWERQUAD\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

### 32.4.2 #define PQ\_Initiate\_Vector\_Func( pSrc, pDst )

#### Value:

```
__asm volatile(
    "MOV r0, %[psrc]          \
     "MOV r1, %[pdst]          \
     "PUSH {r2-r7}             \
     "LDRD r2,r3,[r0],#8      \
     [pdst] "r"(pDst)         \
     : "r0", "r1")
```

Start the vector calculation, the input data could be float, int32\_t or Q31.

Parameters

<i>pSrc</i>	Pointer to the source data.
<i>pDst</i>	Pointer to the destination data.

### 32.4.3 #define PQ\_End\_Vector\_Func( ) \_\_asm volatile("POP {r2-r7}")

This function should be called after vector calculation.

### 32.4.4 #define PQ\_StartVector( PSRC, PDST, LENGTH )

**Value:**

```
__asm volatile(
    "MOV r0, %[psrc]          \
     "MOV r1, %[pdst]          \
     "MOV r2, %[length]         \
     "PUSH {r3-r10}            \
     "MOV r3, #0                \
     "MOV r10, #0               \
     "LDRD r4,r5,[r0],#8       \
     "[pdst] "r"(PDST), [length] "r"(LENGTH) \
     : "r0", "r1", "r2")
```

Start the vector calculation, the input data could be float, int32\_t or Q31.

Parameters

<i>PSRC</i>	Pointer to the source data.
<i>PDST</i>	Pointer to the destination data.
<i>LENGTH</i>	Number of the data, must be multiple of 8.

### 32.4.5 #define PQ\_StartVectorFixed16( PSRC, PDST, LENGTH )

**Value:**

```
__asm volatile(
    "MOV r0, %[psrc]          \
     "MOV r1, %[pdst]          \
     "MOV r2, %[length]         \
     "PUSH {r3-r10}            \
     "MOV r3, #0                \
     "LDRSH r4,[r0],#2          \
     "LDRSH r5,[r0],#2          \
     "[pdst] "r"(PDST), [length] "r"(LENGTH) \
     : "r0", "r1", "r2")
```

Start the vector calculation, the input data could be int16\_t. This function should be use with [PQ\\_Vector8-Fixed16](#).

Parameters

<i>PSRC</i>	Pointer to the source data.
<i>PDST</i>	Pointer to the destination data.
<i>LENGTH</i>	Number of the data, must be multiple of 8.

### 32.4.6 #define PQ\_StartVectorQ15( *PSRC*, *PDST*, *LENGTH* )

**Value:**

```
__asm volatile(
    "MOV r0, %[psrc]          \
     "MOV r1, %[pdst]          \
     "MOV r2, %[length]         \
     "PUSH {r3-r10}            \
     "MOV r3, #0                \
     "LDR r5, [r0], #4          \
     "LSL r4,r5,#16             \
     "BFC r5,#0,#16             \
     %[psrc]      "r"(PSRC),   \
     [pdst]      "r"(PDST),    \
     [length]     "r"(LENGTH)   \
     : "r0", "r1", "r2")
```

Start the vector calculation, the input data could be Q15. This function should be use with [PQ\\_Vector8-Q15](#). This function is dedicate for SinQ15/CosQ15 vector calculation. Because PowerQuad only supports Q31 Sin/Cos fixed function, so the input Q15 data is left shift 16 bits first, after Q31 calculation, the output data is right shift 16 bits.

Parameters

<i>PSRC</i>	Pointer to the source data.
<i>PDST</i>	Pointer to the destination data.
<i>LENGTH</i>	Number of the data, must be multiple of 8.

### 32.4.7 #define PQ\_EndVector( ) \_\_asm volatile("POP {r3-r10}\n")

This function should be called after vector calculation.

### 32.4.8 #define PQ\_Vector8F32( *BATCH\_OPCODE*, *DOUBLE\_READ\_ADDERS*, *BATCH\_MACHINE* )

Float data vector calculation, the input data should be float. The parameter could be PQ\_LN\_INF, PQ\_IN-V\_INF, PQ\_SQRT\_INF, PQ\_ISQRT\_INF, PQETOX\_INF, PQETONX\_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

### **32.4.9 #define PQ\_Vector8Fixed32( *BATCH\_OPCODE*, *DOUBLE\_READ\_ADDERS*, *BATCH\_MACHINE* )**

Float data vector calculation, the input data should be 32-bit integer. The parameter could be PQ\_LN\_INF, PQ\_INV\_INF, PQ\_SQRT\_INF, PQ\_ISQRT\_INF, PQETOX\_INF, PQETONX\_INF. PQ\_SIN\_INF, PQ\_COS\_INF. When this function is used for sin/cos calculation, the input data should be in the format Q1.31. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int32_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

### **32.4.10 #define PQ\_Vector8Fixed16( *BATCH\_OPCODE*, *DOUBLE\_READ\_ADDERS*, *BATCH\_MACHINE* )**

Float data vector calculation, the input data should be 16-bit integer. The parameter could be PQ\_LN\_INF, PQ\_INV\_INF, PQ\_SQRT\_INF, PQ\_ISQRT\_INF, PQETOX\_INF, PQETONX\_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int16_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

### **32.4.11 #define PQ\_Vector8Q15( *BATCH\_OPCODE*, *DOUBLE\_READ\_ADDERS*, *BATCH\_MACHINE* )**

Q15 data vector calculation, this function should only be used for sin/cos Q15 calculation, and the coprocessor output prescaler must be set to 31 before this function. This function loads Q15 data and left shift 16 bits, calculate and right shift 16 bits, then stores to the output array. The input range -1 to 1 means -pi to pi. For example, to calculate sin of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {...}
int16_t output[VECTOR_LEN];
const pq_prescale_t prescale =
{
    .inputPrescale = 0,
    .outputPrescale = 31,
    .outputSaturate = 0
};

PQ_SetCoprocessorScaler(POWERQUAD, const pq_prescale_t *prescale);

PQ_StartVectorQ15(pSrc, pDst, length);
PQ_Vector8Q15(PQ_SQRT_INF);
PQ_EndVector();
```

### 32.4.12 #define PQ\_DF2\_Vector8\_FP( *middle, last* )

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FP(false,false);
PQ_DF2_Vector8_FP(true,true);
PQ_End_Vector_Func();
```

### 32.4.13 #define PQ\_DF2\_Vector8\_FX( *middle, last* )

Biquad filter, the input and output data are fixed data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
```

```

        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FX(false,false);
PQ_DF2_Vector8_FX(true,true);
PQ_End_Vector_Func();

```

### 32.4.14 #define PQ\_Vector8BiquadDf2F32( )

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```

#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2F32();
PQ_EndVector();

```

### 32.4.15 #define PQ\_Vector8BiquadDf2Fixed32( )

Biquad filter, the input and output data are Q31 or 32-bit integer. Biquad side 0 is used. Example:

```

#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

```

```
PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed32();
PQ_EndVector();
```

### 32.4.16 #define PQ\_Vector8BiquadDf2Fixed16( )

Biquad filter, the input and output data are Q15 or 16-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int16_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed16();
PQ_EndVector();
```

### 32.4.17 #define PQ\_DF2\_Cascade\_Vector8\_FP( *middle*, *last* )

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxxx,
```

```

.a_2 = xxxx,
.b_0 = xxxx,
.b_1 = xxxx,
.b_2 = xxxx,
},
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FP(false, false);
PQ_DF2_Cascade_Vector8_FP(true, true);
PQ_End_Vector_Func();

```

### 32.4.18 #define PQ\_DF2\_Cascade\_Vector8\_FX( *middle, last* )

The input and output data are fixed data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FX(false, false);
PQ_DF2_Cascade_Vector8_FX(true, true);
PQ_End_Vector_Func();

```

**32.4.19 #define PQ\_Vector8BiquadDf2CascadeF32( )**

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeF32();
PQ_EndVector();
```

**32.4.20 #define PQ\_Vector8BiquadDf2CascadeFixed32( )**

The input and output data are fixed 32-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};
```

```

pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed32();
PQ_EndVector();

```

### 32.4.21 #define PQ\_Vector8BiquadDf2CascadeFixed16( )

The input and output data are fixed 16-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```

#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxxx,
        .a_2 = xxxx,
        .b_0 = xxxx,
        .b_1 = xxxx,
        .b_2 = xxxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2CascadeFixed16();
PQ_EndVector();

```

**32.4.22 #define POWERQUAD\_MAKE\_MATRIX\_LEN( mat1Row, mat1Col,  
mat2Col ) (((uint32\_t)(mat1Row) << 0U) | ((uint32\_t)(mat1Col) << 8U) |  
((uint32\_t)(mat2Col) << 16U))**

**32.4.23 #define PQ\_Q31\_2\_FLOAT( x ) (((float)(x)) / 2147483648.0f)**

**32.4.24 #define PQ\_Q15\_2\_FLOAT( x ) (((float)(x)) / 32768.0f)**

## 32.5 Enumeration Type Documentation

### 32.5.1 enum pq\_computationengine\_t

Enumerator

*kPQ\_CP\_PQ* Math engine.

*kPQ\_CP\_mtx* Matrix engine.

*kPQ\_CP\_FFT* FFT engine.

*kPQ\_CP\_FIR* FIR engine.

*kPQ\_CP\_CORDIC* CORDIC engine.

### 32.5.2 enum pq\_format\_t

Enumerator

*kPQ\_16Bit* Int16 Fixed point.

*kPQ\_32Bit* Int32 Fixed point.

*kPQ\_Float* Float point.

### 32.5.3 enum pq\_cordic\_iter\_t

Enumerator

*kPQ\_Iteration\_8* Iterate 8 times.

*kPQ\_Iteration\_16* Iterate 16 times.

*kPQ\_Iteration\_24* Iterate 24 times.

## 32.6 Function Documentation

### 32.6.1 void PQ\_GetDefaultConfig ( pq\_config\_t \* config )

This function initializes the POWERQUAD configuration structure to a default value. FORMAT register field definitions Bits[15:8] scaler (for scaled 'q31' formats) Bits[5:4] external format. 00b=q15, 01b=q31,

10b=float Bits[1:0] internal format. 00b=q15, 01b=q31, 10b=float POWERQUAD->INAFORMAT = (config->inputAPrescale << 8U) | (config->inputAFormat << 4U) | config->machineFormat

For all Powerquad operations internal format must be float (with the only exception being the FFT related functions, ie FFT/IFFT/DCT/IDCT which must be set to q31). The default values are: config->inputAFormat = kPQ\_Float; config->inputAPrescale = 0; config->inputBFormat = kPQ\_Float; config->inputBPrescale = 0; config->outputFormat = kPQ\_Float; config->outputPrescale = 0; config->tmpFormat = kPQ\_Float; config->tmpPrescale = 0; config->machineFormat = kPQ\_Float; config->tmpBase = 0xE0000000;

Parameters

<i>config</i>	Pointer to "pq_config_t" structure.
---------------	-------------------------------------

### 32.6.2 void PQ\_SetConfig ( POWERQUAD\_Type \* *base*, const pq\_config\_t \* *config* )

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>config</i>	Pointer to "pq_config_t" structure.

### 32.6.3 static void PQ\_SetCoprocessorScaler ( POWERQUAD\_Type \* *base*, const pq\_prescale\_t \* *prescale* ) [inline], [static]

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>prescale</i>	Pointer to "pq_prescale_t" structure.

### 32.6.4 void PQ\_Init ( POWERQUAD\_Type \* *base* )

Parameters

---

<i>base</i>	POWERQUAD peripheral base address.
-------------	------------------------------------

**32.6.5 void PQ\_Deinit ( POWERQUAD\_Type \* *base* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address.
-------------	------------------------------------

**32.6.6 void PQ\_SetFormat ( POWERQUAD\_Type \* *base*, pq\_computationengine\_t *engine*, pq\_format\_t *format* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>engine</i>	Computation engine
<i>format</i>	Data format

**32.6.7 static void PQ\_WaitDone ( POWERQUAD\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	POWERQUAD peripheral base address
-------------	-----------------------------------

**32.6.8 static void PQ\_LnF32 ( float \* *pSrc*, float \* *pDst* ) [inline], [static]**

Parameters

* <i>pSrc</i>	points to the block of input data. The range of the input value is (0 +INFINITY).
* <i>pDst</i>	points to the block of output data

**32.6.9 static void PQ\_InvF32 ( float \* *pSrc*, float \* *pDst* ) [inline], [static]**

Parameters

<i>*pSrc</i>	points to the block of input data. The range of the input value is non-zero.
<i>*pDst</i>	points to the block of output data

### 32.6.10 static void PQ\_SqrtF32 ( float \* *pSrc*, float \* *pDst* ) [inline], [static]

Parameters

<i>*pSrc</i>	points to the block of input data. The range of the input value is [0 +INFINITY).
<i>*pDst</i>	points to the block of output data

### 32.6.11 static void PQ\_InvSqrtF32 ( float \* *pSrc*, float \* *pDst* ) [inline], [static]

Parameters

<i>*pSrc</i>	points to the block of input data. The range of the input value is (0 +INFINITY).
<i>*pDst</i>	points to the block of output data

### 32.6.12 static void PQ\_EtoxF32 ( float \* *pSrc*, float \* *pDst* ) [inline], [static]

Parameters

<i>*pSrc</i>	points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
<i>*pDst</i>	points to the block of output data

### 32.6.13 static void PQ\_EtonxF32 ( float \* *pSrc*, float \* *pDst* ) [inline], [static]

Parameters

<code>*pSrc</code>	points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
<code>*pDst</code>	points to the block of output data

### 32.6.14 static void PQ\_SinF32 ( `float * pSrc, float * pDst` ) [inline], [static]

Parameters

<code>*pSrc</code>	points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
<code>*pDst</code>	points to the block of output data

### 32.6.15 static void PQ\_CosF32 ( `float * pSrc, float * pDst` ) [inline], [static]

Parameters

<code>*pSrc</code>	points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
<code>*pDst</code>	points to the block of output data

### 32.6.16 static void PQ\_BiquadF32 ( `float * pSrc, float * pDst` ) [inline], [static]

Parameters

<code>*pSrc</code>	points to the block of input data
<code>*pDst</code>	points to the block of output data

### 32.6.17 static void PQ\_DivF32 ( `float * x1, float * x2, float * pDst` ) [inline], [static]

Get  $x1 / x2$ .

Parameters

$x1$	x1
$x2$	x2
$*pDst$	points to the block of output data

**32.6.18 static void PQ\_Biquad1F32 ( float \* *pSrc*, float \* *pDst* ) [inline], [static]**

Parameters

$*pSrc$	points to the block of input data
$*pDst$	points to the block of output data

**32.6.19 static int32\_t PQ\_LnFixed ( int32\_t *val* ) [inline], [static]**

Parameters

<i>val</i>	value to be calculated. The range of the input value is (0 +INFINITY).
------------	--

Returns

returns  $\ln(\text{val})$ .

**32.6.20 static int32\_t PQ\_InvFixed ( int32\_t *val* ) [inline], [static]**

Parameters

<i>val</i>	value to be calculated. The range of the input value is non-zero.
------------	---

Returns

returns  $\text{inv}(\text{val})$ .

**32.6.21 static uint32\_t PQ\_SqrtFixed ( uint32\_t *val* ) [inline], [static]**

Parameters

<i>val</i>	value to be calculated. The range of the input value is [0 +INFINITY).
------------	--

Returns

returns sqrt(val).

### 32.6.22 static int32\_t PQ\_InvSqrtFixed ( int32\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated. The range of the input value is (0 +INFINITY).
------------	--

Returns

returns 1/sqrt(val).

### 32.6.23 static int32\_t PQ\_EtoxFixed ( int32\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated. The range of the input value is (-INFINITY +INFINITY).
------------	--

Returns

returns etox<sup>^</sup>(val).

### 32.6.24 static int32\_t PQ\_EtonxFixed ( int32\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated. The range of the input value is (-INFINITY +INFINITY).
------------	--

Returns

returns etonx<sup>^</sup>(val).

### 32.6.25 static int32\_t PQ\_SinQ31 ( int32\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].
------------	--

Returns

returns sin(val).

### 32.6.26 static int16\_t PQ\_SinQ15 ( int16\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].
------------	--

Returns

returns sin(val).

### 32.6.27 static int32\_t PQ\_CosQ31 ( int32\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].
------------	--

Returns

returns cos(val).

### 32.6.28 static int16\_t PQ\_CosQ15 ( int16\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].
------------	--

Returns

returns cos(val).

### 32.6.29 static int32\_t PQ\_BiquadFixed ( int32\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated
------------	------------------------

Returns

returns biquad(val).

### 32.6.30 void PQ\_VectorLnF32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

* <i>pSrc</i>	points to the block of input data
* <i>pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.31 void PQ\_VectorInvF32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

* <i>pSrc</i>	points to the block of input data
* <i>pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.32 void PQ\_VectorSqrtF32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

* <i>pSrc</i>	points to the block of input data
* <i>pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.33 void PQ\_VectorInvSqrtF32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.34 void PQ\_VectorEtoxF32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.35 void PQ\_VectorEtonxF32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.36 void PQ\_VectorSinF32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.37 void PQ\_VectorCosF32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.38 void PQ\_VectorLnFixed32 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.39 void PQ\_VectorInvFixed32 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.40 void PQ\_VectorSqrtFixed32 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.41 void PQ\_VectorInvSqrtFixed32 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.42 void PQ\_VectorEtoxFixed32 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.43 void PQ\_VectorEtonxFixed32 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.44 void PQ\_VectorSinQ15 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.45 void PQ\_VectorCosQ15 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.46 void PQ\_VectorSinQ31 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.47 void PQ\_VectorCosQ31 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.48 void PQ\_VectorLnFixed16 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.49 void PQ\_VectorInvFixed16 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

**32.6.50 void PQ\_VectorSqrtFixed16 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )**

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

**32.6.51 void PQ\_VectorInvSqrtFixed16 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )**

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

**32.6.52 void PQ\_VectorEtoxFixed16 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )**

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data

<i>length</i>	the block of input data.
---------------	--------------------------

### 32.6.53 void PQ\_VectorEtonxFixed16 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )

Parameters

* <i>pSrc</i>	points to the block of input data
* <i>pDst</i>	points to the block of output data
<i>length</i>	the block of input data.

### 32.6.54 void PQ\_VectorBiquadDf2F32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

* <i>pSrc</i>	points to the block of input data
* <i>pDst</i>	points to the block of output data
<i>length</i>	the block size of input data.

### 32.6.55 void PQ\_VectorBiquadDf2Fixed32 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

* <i>pSrc</i>	points to the block of input data
* <i>pDst</i>	points to the block of output data
<i>length</i>	the block size of input data

### 32.6.56 void PQ\_VectorBiquadDf2Fixed16 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block size of input data

### 32.6.57 void PQ\_VectorBiquadCascadeDf2F32 ( float \* *pSrc*, float \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block size of input data

### 32.6.58 void PQ\_VectorBiquadCascadeDf2Fixed32 ( int32\_t \* *pSrc*, int32\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data
<i>length</i>	the block size of input data

### 32.6.59 void PQ\_VectorBiquadCascadeDf2Fixed16 ( int16\_t \* *pSrc*, int16\_t \* *pDst*, int32\_t *length* )

Parameters

<i>*pSrc</i>	points to the block of input data
<i>*pDst</i>	points to the block of output data

<i>length</i>	the block size of input data
---------------	------------------------------

### 32.6.60 int32\_t PQ\_ArcTanFixed ( POWERQUAD\_Type \* *base*, int32\_t *x*, int32\_t *y*, pq\_cordic\_iter\_t *iteration* )

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>x</i>	value of opposite
<i>y</i>	value of adjacent
<i>iteration</i>	iteration times

Returns

The return value is in the range of -2^27 to 2^27, which means -pi to pi.

Note

The sum of *x* and *y* should not exceed the range of int32\_t.

Larger input number gets higher output accuracy, for example the arctan(0.5), the result of PQ\_ArcTanFixed(POWERQUAD, 100000, 200000, kPQ\_Iteration\_24) is more accurate than PQ\_ArcTanFixed(POWERQUAD, 1, 2, kPQ\_Iteration\_24).

### 32.6.61 int32\_t PQ\_ArctanhFixed ( POWERQUAD\_Type \* *base*, int32\_t *x*, int32\_t *y*, pq\_cordic\_iter\_t *iteration* )

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>x</i>	value of opposite
<i>y</i>	value of adjacent

<i>iteration</i>	iteration times
------------------	-----------------

Returns

The return value is in the range of -2<sup>27</sup> to 2<sup>27</sup>, which means -1 to 1.

Note

The sum of x and y should not exceed the range of int32\_t.

Larger input number gets higher output accuracy, for example the arctanh(0.5), the result of PQ\_ArctanhFixed(POWERQUAD, 100000, 200000, kPQ\_Iteration\_24) is more accurate than PQ\_ArctanhFixed(POWERQUAD, 1, 2, kPQ\_Iteration\_24).

### 32.6.62 static int32\_t PQ\_Biquad1Fixed ( int32\_t val ) [inline], [static]

Parameters

<i>val</i>	value to be calculated
------------	------------------------

Returns

returns biquad(val).

### 32.6.63 void PQ\_TransformCFFT ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pData*, void \* *pResult* )

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	number of input samples
<i>pData</i>	input data
<i>pResult</i>	output data.

### 32.6.64 void PQ\_TransformRFFT ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pData*, void \* *pResult* )

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	number of input samples
<i>pData</i>	input data
<i>pResult</i>	output data.

**32.6.65 void PQ\_TransformIFFT ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	number of input samples
<i>pData</i>	input data
<i>pResult</i>	output data.

**32.6.66 void PQ\_TransformCDCT ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	number of input samples
<i>pData</i>	input data
<i>pResult</i>	output data.

**32.6.67 void PQ\_TransformRDCT ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	number of input samples
<i>pData</i>	input data
<i>pResult</i>	output data.

**32.6.68 void PQ\_TransformIDCT ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	number of input samples
<i>pData</i>	input data
<i>pResult</i>	output data.

**32.6.69 void PQ\_BiquadBackUpInternalState ( POWERQUAD\_Type \* *base*, int32\_t *biquad\_num*, pq\_biquad\_state\_t \* *state* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>biquad_num</i>	biquad side
<i>state</i>	point to states.

**32.6.70 void PQ\_BiquadRestoreInternalState ( POWERQUAD\_Type \* *base*, int32\_t *biquad\_num*, pq\_biquad\_state\_t \* *state* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>biquad_num</i>	biquad side
<i>state</i>	point to states.

32.6.71 void PQ\_BiquadCascadeDf2Init ( pq\_biquad\_cascade\_df2\_instance \* **S**,  
                  uint8\_t *numStages*, pq\_biquad\_state\_t \* *pState* )

Parameters

in, out	<i>*S</i>	points to an instance of the filter data structure.
in	<i>numStages</i>	number of 2nd order stages in the filter.
in	<i>*pState</i>	points to the state buffer.

**32.6.72 void PQ\_BiquadCascadeDf2F32 ( const pq\_biquad\_cascade\_df2\_instance \* *S*, float \* *pSrc*, float \* *pDst*, uint32\_t *blockSize* )**

Parameters

in	<i>*S</i>	points to an instance of the filter data structure.
in	<i>*pSrc</i>	points to the block of input data.
out	<i>*pDst</i>	points to the block of output data
in	<i>blockSize</i>	number of samples to process.

**32.6.73 void PQ\_BiquadCascadeDf2Fixed32 ( const pq\_biquad\_cascade\_df2\_instance \* *S*, int32\_t \* *pSrc*, int32\_t \* *pDst*, uint32\_t *blockSize* )**

Parameters

in	<i>*S</i>	points to an instance of the filter data structure.
in	<i>*pSrc</i>	points to the block of input data.
out	<i>*pDst</i>	points to the block of output data
in	<i>blockSize</i>	number of samples to process.

**32.6.74 void PQ\_BiquadCascadeDf2Fixed16 ( const pq\_biquad\_cascade\_df2\_instance \* *S*, int16\_t \* *pSrc*, int16\_t \* *pDst*, uint32\_t *blockSize* )**

## Parameters

in	<i>*S</i>	points to an instance of the filter data structure.
in	<i>*pSrc</i>	points to the block of input data.
out	<i>*pDst</i>	points to the block of output data
in	<i>blockSize</i>	number of samples to process.

**32.6.75 void PQ\_FIR ( POWERQUAD\_Type \* *base*, const void \* *pAData*, int32\_t *ALength*, const void \* *pBData*, int32\_t *BLength*, void \* *pResult*, uint32\_t *opType* )**

## Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>pAData</i>	the first input sequence
<i>ALength</i>	number of the first input sequence
<i>pBData</i>	the second input sequence
<i>BLength</i>	number of the second input sequence
<i>pResult</i>	array for the output data
<i>opType</i>	operation type, could be PQ_FIR_FIR, PQ_FIR_CONVOLUTION, PQ_FIR_CORRELATION.

**32.6.76 void PQ\_FIRIncrement ( POWERQUAD\_Type \* *base*, int32\_t *ALength*, int32\_t *BLength*, int32\_t *xOffset* )**

This function can be used after pq\_fir() for incremental FIR operation when new x data are available

## Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>ALength</i>	number of input samples

<i>BLength</i>	number of taps
<i>xOffset</i>	offset for number of input samples

**32.6.77 void PQ\_MatrixAddition ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pAData*, void \* *pBData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro <a href="#">POWERQUAD_MAKE_MATRIX_LEN</a> .
<i>pAData</i>	input matrix A
<i>pBData</i>	input matrix B
<i>pResult</i>	array for the output data.

**32.6.78 void PQ\_MatrixSubtraction ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pAData*, void \* *pBData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro <a href="#">POWERQUAD_MAKE_MATRIX_LEN</a> .
<i>pAData</i>	input matrix A
<i>pBData</i>	input matrix B
<i>pResult</i>	array for the output data.

**32.6.79 void PQ\_MatrixMultiplication ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pAData*, void \* *pBData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro <a href="#">POWERQUAD_MAKE_MATRIX_LEN</a> .
<i>pAData</i>	input matrix A
<i>pBData</i>	input matrix B
<i>pResult</i>	array for the output data.

**32.6.80 void PQ\_MatrixProduct ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pAData*, void \* *pBData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro <a href="#">POWERQUAD_MAKE_MATRIX_LEN</a> .
<i>pAData</i>	input matrix A
<i>pBData</i>	input matrix B
<i>pResult</i>	array for the output data.

**32.6.81 void PQ\_VectorDotProduct ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pAData*, void \* *pBData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	length of vector
<i>pAData</i>	input vector A
<i>pBData</i>	input vector B
<i>pResult</i>	array for the output data.

32.6.82 void PQ\_MatrixInversion ( POWERQUAD\_Type \* *base*, uint32\_t *length*,  
void \* *pData*, void \* *pTmpData*, void \* *pResult* )

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro <a href="#">POWERQUAD_MAKE_MATRIX_LEN</a> .
<i>pData</i>	input matrix
<i>pTmpData</i>	input temporary matrix, pTmpData length not less than pData lenght and 1024 words is sufficient for the largest supported matrix.
<i>pResult</i>	array for the output data, round down for fixed point.

**32.6.83 void PQ\_MatrixTranspose ( POWERQUAD\_Type \* *base*, uint32\_t *length*, void \* *pData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro <a href="#">POWERQUAD_MAKE_MATRIX_LEN</a> .
<i>pData</i>	input matrix
<i>pResult</i>	array for the output data.

**32.6.84 void PQ\_MatrixScale ( POWERQUAD\_Type \* *base*, uint32\_t *length*, float *misc*, const void \* *pData*, void \* *pResult* )**

Parameters

<i>base</i>	POWERQUAD peripheral base address
<i>length</i>	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro <a href="#">POWERQUAD_MAKE_MATRIX_LEN</a> .
<i>misc</i>	scaling parameters
<i>pData</i>	input matrix
<i>pResult</i>	array for the output data.

# Chapter 33

## PRINCE: PRINCE bus crypto engine

### 33.1 Overview

The MCUXpresso SDK provides a peripheral driver for the PRINCE bus crypto engine module of MCUXpresso SDK devices.

..

This example code shows how to use the PRINCE driver.

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/prince

### Enumerations

- enum `skboot_status_t` {  
    `kStatus_SKBOOT_Success` = 0x5ac3c35au,  
    `kStatus_SKBOOT_Fail` = 0xc35ac35au,  
    `kStatus_SKBOOT_InvalidArgument` = 0xc35a5ac3u,  
    `kStatus_SKBOOT_KeyStoreMarkerInvalid` = 0xc3c35a5au,  
    `kStatus_SKBOOT_HashcryptFinishedWithStatusSuccess`,  
    `kStatus_SKBOOT_HashcryptFinishedWithStatusFail`,  
    `kStatus_SKBOOT_Success` = 0x5ac3c35au,  
    `kStatus_SKBOOT_Fail` = 0xc35ac35au,  
    `kStatus_SKBOOT_InvalidArgument` = 0xc35a5ac3u,  
    `kStatus_SKBOOT_KeyStoreMarkerInvalid` = 0xc3c35a5au }  
        *Secure status enumeration.*
- enum `secure_bool_t` {  
    `kSECURE_TRUE` = 0xc33cc33cU,  
    `kSECURE_FALSE` = 0x5aa55aa5U,  
    `kSECURE_CALLPROTECT_SECURITY_FLAGS` = 0xc33c5aa5U,  
    `kSECURE_CALLPROTECT_IS_APP_READY` = 0x5aa5c33cU,  
    `kSECURE_TRACKER_VERIFIED` = 0x55aacc33U,  
    `kSECURE_TRUE` = 0xc33cc33cU,  
    `kSECURE_FALSE` = 0x5aa55aa5U }  
        *Secure boolean enumeration.*
- enum `prince_region_t` {  
    `kPRINCE_Region0` = 0U,  
    `kPRINCE_Region1` = 1U,  
    `kPRINCE_Region2` = 2U }  
        *Prince region.*
- enum `prince_lock_t` {

```
kPRINCE_Region0Lock = 1U,
kPRINCE_Region1Lock = 2U,
kPRINCE_Region2Lock = 4U,
kPRINCE_MaskLock = 256U }
```

*Prince lock.*

- enum `prince_flags_t` {
   
kPRINCE\_Flag\_None = 0U,
   
kPRINCE\_Flag\_EraseCheck = 1U,
   
kPRINCE\_Flag\_WriteCheck = 2U }

*Prince flag.*

## Functions

- static void `PRINCE_EncryptEnable` (`PRINCE_Type` \*base)  
*Enable data encryption.*
- static void `PRINCE_EncryptDisable` (`PRINCE_Type` \*base)  
*Disable data encryption.*
- static bool `PRINCE_IsEncryptEnable` (`PRINCE_Type` \*base)  
*Is Enable data encryption.*
- static void `PRINCE_SetMask` (`PRINCE_Type` \*base, `uint64_t` mask)  
*Sets PRINCE data mask.*
- static void `PRINCE_SetLock` (`PRINCE_Type` \*base, `uint32_t` lock)  
*Locks access for specified region registers or data mask register.*
- `status_t PRINCE_GenNewIV` (`prince_region_t` region, `uint8_t` \*iv\_code, `bool` store, `flash_config_t` \*flash\_context)  
*Generate new IV code.*
- `status_t PRINCE_LoadIV` (`prince_region_t` region, `uint8_t` \*iv\_code)  
*Load IV code.*
- `status_t PRINCE_SetEncryptForAddressRange` (`prince_region_t` region, `uint32_t` start\_address, `uint32_t` length, `flash_config_t` \*flash\_context, `bool` regenerate\_iv)  
*Allow encryption/decryption for specified address range.*
- `status_t PRINCE_GetRegionSREnable` (`PRINCE_Type` \*base, `prince_region_t` region, `uint32_t` \*sr\_enable)  
*Gets the PRINCE Sub-Region Enable register.*
- `status_t PRINCE_GetRegionBaseAddress` (`PRINCE_Type` \*base, `prince_region_t` region, `uint32_t` \*region\_base\_addr)  
*Gets the PRINCE region base address register.*
- `status_t PRINCE_SetRegionIV` (`PRINCE_Type` \*base, `prince_region_t` region, `const uint8_t` iv[8])  
*Sets the PRINCE region IV.*
- `status_t PRINCE_SetRegionBaseAddress` (`PRINCE_Type` \*base, `prince_region_t` region, `uint32_t` region\_base\_addr)  
*Sets the PRINCE region base address.*
- `status_t PRINCE_SetRegionSREnable` (`PRINCE_Type` \*base, `prince_region_t` region, `uint32_t` sr\_enable)  
*Sets the PRINCE Sub-Region Enable register.*
- `status_t PRINCE_FlashEraseWithChecker` (`flash_config_t` \*config, `uint32_t` start, `uint32_t` length-InBytes, `uint32_t` key)  
*Erases the flash sectors encompassed by parameters passed into function.*
- `status_t PRINCE_FlashProgramWithChecker` (`flash_config_t` \*config, `uint32_t` start, `uint8_t` \*src,

`uint32_t lengthInBytes)`  
*Programs flash with data at locations passed in through parameters.*

## Driver version

- `#define FSL_PRINCE_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`  
*PRINCE driver version 2.5.0.*

## 33.2 Macro Definition Documentation

### 33.2.1 `#define FSL_PRINCE_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`

Current version: 2.5.0

Change log:

- Version 2.0.0
  - Initial version.
- Version 2.1.0
  - Update for the A1 rev. of LPC55Sxx serie.
- Version 2.2.0
  - Add runtime checking of the A0 and A1 rev. of LPC55Sxx serie to support both silicone revisions.
- Version 2.3.0
  - Add support for LPC55S1x and LPC55S2x series
- Version 2.3.0
  - Fix MISRA-2012 issues.
- Version 2.3.1
  - Add support for LPC55S0x series
- Version 2.3.2
  - Fix documentation of enumeration. Extend PRINCE example.
- Version 2.4.0
  - Add support for LPC55S3x series
- Version 2.5.0
  - Add PRINCE\_Config() and PRINCE\_Reconfig() features.

## 33.3 Enumeration Type Documentation

### 33.3.1 `enum skboot_status_t`

Enumerator

`kStatus_SKBOOT_Success` SKBOOT return success status.  
`kStatus_SKBOOT_Fail` SKBOOT return fail status.  
`kStatus_SKBOOT_InvalidArgument` SKBOOT return invalid argument status.  
`kStatus_SKBOOT_KeyStoreMarkerInvalid` SKBOOT return Keystore invalid Marker status.  
`kStatus_SKBOOT_HashcryptFinishedWithStatusSuccess` SKBOOT return Hashcrypt finished with the success status.

*kStatus\_SKBOOT\_HashcryptFinishedWithStatusFail* SKBOOT return Hashcrypt finished with the fail status.

*kStatus\_SKBOOT\_Success* PRINCE Success.

*kStatus\_SKBOOT\_Fail* PRINCE Fail.

*kStatus\_SKBOOT\_InvalidArgument* PRINCE Invalid argument.

*kStatus\_SKBOOT\_KeyStoreMarkerInvalid* PRINCE Invalid marker.

### 33.3.2 enum secure\_bool\_t

Enumerator

*kSECURE\_TRUE* Secure true flag.

*kSECURE\_FALSE* Secure false flag.

*kSECURE\_CALLPROTECT\_SECURITY\_FLAGS* Secure call protect the security flag.

*kSECURE\_CALLPROTECT\_IS\_APP\_READY* Secure call protect the app is ready flag.

*kSECURE\_TRACKER\_VERIFIED* Secure tracker verified flag.

*kSECURE\_TRUE* PRINCE true.

*kSECURE\_FALSE* PRINCE false.

### 33.3.3 enum prince\_region\_t

Enumerator

*kPRINCE\_Region0* PRINCE region 0.

*kPRINCE\_Region1* PRINCE region 1.

*kPRINCE\_Region2* PRINCE region 2.

### 33.3.4 enum prince\_lock\_t

Enumerator

*kPRINCE\_Region0Lock* PRINCE region 0 lock.

*kPRINCE\_Region1Lock* PRINCE region 1 lock.

*kPRINCE\_Region2Lock* PRINCE region 2 lock.

*kPRINCE\_MaskLock* PRINCE mask register lock.

### 33.3.5 enum prince\_flags\_t

Enumerator

*kPRINCE\_Flag\_None* PRINCE Flag None.

*kPRINCE\_Flag\_EraseCheck* PRINCE Flag Erase check.

*kPRINCE\_Flag\_WriteCheck* PRINCE Flag Write check.

## 33.4 Function Documentation

### 33.4.1 static void PRINCE\_EncryptEnable ( PRINCE\_Type \* *base* ) [inline], [static]

This function enables PRINCE on-the-fly data encryption.

Parameters

<i>base</i>	PRINCE peripheral address.
-------------	----------------------------

### 33.4.2 static void PRINCE\_EncryptDisable ( PRINCE\_Type \* *base* ) [inline], [static]

This function disables PRINCE on-the-fly data encryption.

Parameters

<i>base</i>	PRINCE peripheral address.
-------------	----------------------------

### 33.4.3 static bool PRINCE\_IsEncryptEnable ( PRINCE\_Type \* *base* ) [inline], [static]

This function test if PRINCE on-the-fly data encryption is enabled.

Parameters

<i>base</i>	PRINCE peripheral address.
-------------	----------------------------

Returns

true if enabled, false if not

### 33.4.4 static void PRINCE\_SetMask ( PRINCE\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

This function sets the PRINCE mask that is used to mask decrypted data.

Parameters

<i>base</i>	PRINCE peripheral address.
<i>mask</i>	64-bit data mask value.

### 33.4.5 static void PRINCE\_SetLock ( PRINCE\_Type \* *base*, uint32\_t *lock* ) [inline], [static]

This function sets lock on specified region registers or mask register.

Parameters

<i>base</i>	PRINCE peripheral address.
<i>lock</i>	registers to lock. This is a logical OR of members of the enumeration <a href="#">prince_lock_t</a>

### 33.4.6 status\_t PRINCE\_GenNewIV ( prince\_region\_t *region*, uint8\_t \* *iv\_code*, bool *store*, flash\_config\_t \* *flash\_context* )

This function generates new IV code and stores it into the persistent memory. Ensure about 800 bytes free space on the stack when calling this routine with the store parameter set to true!

Parameters

<i>region</i>	PRINCE region index.
<i>iv_code</i>	IV code pointer used for storing the newly generated 52 bytes long IV code.
<i>store</i>	flag to allow storing the newly generated IV code into the persistent memory (FFR).
<i>flash_context</i>	pointer to the flash driver context structure.

Returns

kStatus\_Success upon success

kStatus\_Fail otherwise, kStatus\_Fail is also returned if the key code for the particular PRINCE region is not present in the keystore (though new IV code has been provided)

### 33.4.7 status\_t PRINCE\_LoadIV ( prince\_region\_t *region*, uint8\_t \* *iv\_code* )

This function enables IV code loading into the PRINCE bus encryption engine.

Parameters

<i>region</i>	PRINCE region index.
<i>iv_code</i>	IV code pointer used for passing the IV code.

Returns

kStatus\_Success upon success  
kStatus\_Fail otherwise

### 33.4.8 status\_t PRINCE\_SetEncryptForAddressRange ( *prince\_region\_t region*, *uint32\_t start\_address*, *uint32\_t length*, *flash\_config\_t \* flash\_context*, *bool regenerate\_iv* )

This function sets the encryption/decryption for specified address range. The SR mask value for the selected Prince region is calculated from provided start\_address and length parameters. This calculated value is OR'ed with the actual SR mask value and stored into the PRINCE SR\_ENABLE register and also into the persistent memory (FFR) to be used after the device reset. It is possible to define several nonadjacent encrypted areas within one Prince region when calling this function repeatedly. If the length parameter is set to 0, the SR mask value is set to 0 and thus the encryption/decryption for the whole selected Prince region is disabled. Ensure about 800 bytes free space on the stack when calling this routine!

Parameters

<i>region</i>	PRINCE region index.
<i>start_address</i>	start address of the area to be encrypted/decrypted.
<i>length</i>	length of the area to be encrypted/decrypted.
<i>flash_context</i>	pointer to the flash driver context structure.
<i>regenerate_iv</i>	flag to allow IV code regenerating, storing into the persistent memory (FFR) and loading into the PRINCE engine

Returns

kStatus\_Success upon success  
kStatus\_Fail otherwise

### 33.4.9 status\_t PRINCE\_GetRegionSREnable ( *PRINCE\_Type \* base*, *prince\_region\_t region*, *uint32\_t \* sr\_enable* )

This function gets PRINCE SR\_ENABLE register.

Parameters

<i>base</i>	PRINCE peripheral address.
<i>region</i>	PRINCE region index.
<i>sr_enable</i>	Sub-Region Enable register pointer.

Returns

kStatus\_Success upon success

kStatus\_InvalidArgument

### 33.4.10 status\_t PRINCE\_GetRegionBaseAddress ( PRINCE\_Type \* *base*, prince\_region\_t *region*, uint32\_t \* *region\_base\_addr* )

This function gets PRINCE BASE\_ADDR register.

Parameters

<i>base</i>	PRINCE peripheral address.
<i>region</i>	PRINCE region index.
<i>region_base_addr</i>	Region base address pointer.

Returns

kStatus\_Success upon success

kStatus\_InvalidArgument

### 33.4.11 status\_t PRINCE\_SetRegionIV ( PRINCE\_Type \* *base*, prince\_region\_t *region*, const uint8\_t *iv[8]* )

This function sets specified AES IV for the given region.

Parameters

<i>base</i>	PRINCE peripheral address.
<i>region</i>	Selection of the PRINCE region to be configured.
<i>iv</i>	64-bit AES IV in little-endian byte order.

### 33.4.12 status\_t PRINCE\_SetRegionBaseAddress ( **PRINCE\_Type** \* *base*, **prince\_region\_t** *region*, **uint32\_t** *region\_base\_addr* )

This function configures PRINCE region base address.

Parameters

<i>base</i>	PRINCE peripheral address.
<i>region</i>	Selection of the PRINCE region to be configured.
<i>region_base_addr</i>	Base Address for region.

### 33.4.13 status\_t PRINCE\_SetRegionSREnable ( **PRINCE\_Type** \* *base*, **prince\_region\_t** *region*, **uint32\_t** *sr\_enable* )

This function configures PRINCE SR\_ENABLE register.

Parameters

<i>base</i>	PRINCE peripheral address.
<i>region</i>	Selection of the PRINCE region to be configured.
<i>sr_enable</i>	Sub-Region Enable register value.

### 33.4.14 status\_t PRINCE\_FlashEraseWithChecker ( **flash\_config\_t** \* *config*, **uint32\_t** *start*, **uint32\_t** *lengthInBytes*, **uint32\_t** *key* )

This function erases the appropriate number of flash sectors based on the desired start address and length. It deals with the flash erase function complementary to the standard erase API of the IAP1 driver. This implementation additionally checks if the whole encrypted PRINCE subregions are erased at once to avoid secrets revealing. The checker implementation is limited to one contiguous PRINCE-controlled memory area.

## Parameters

<i>config</i>	The pointer to the flash driver context structure.
<i>start</i>	The start address of the desired flash memory to be erased. The start address needs to be prince-sburegion-aligned.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words) to be erased. Must be prince-sburegion-size-aligned.
<i>key</i>	The value used to validate all flash erase APIs.

## Returns

[kStatus\\_FLASH\\_Success](#) API was executed successfully.  
[kStatus\\_FLASH\\_InvalidArgument](#) An invalid argument is provided.  
[kStatus\\_FLASH\\_AlignmentError](#) The parameter is not aligned with the specified baseline.  
[kStatus\\_FLASH\\_AddressError](#) The address is out of range.  
[kStatus\\_FLASH\\_EraseKeyError](#) The API erase key is invalid.  
[kStatus\\_FLASH\\_CommandFailure](#) Run-time error during the command execution.  
[kStatus\\_FLASH\\_CommandNotSupported](#) Flash API is not supported.  
[kStatus\\_FLASH\\_EccError](#) A correctable or uncorrectable error during command execution.  
[kStatus\\_FLASH\\_EncryptedRegionsEraseNotDoneAtOnce](#) Encrypted flash subregions are not erased at once.

### 33.4.15 `status_t PRINCE_FlashProgramWithChecker ( flash_config_t * config, uint32_t start, uint8_t * src, uint32_t lengthInBytes )`

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length. It deals with the flash program function complementary to the standard program API of the IAP1 driver. This implementation additionally checks if the whole PRINCE subregions are programmed at once to avoid secrets revealing. The checker implementation is limited to one contiguous PRINCE-controlled memory area.

## Parameters

<i>config</i>	The pointer to the flash driver context structure.
<i>start</i>	The start address of the desired flash memory to be programmed. Must be prince-sburegion-aligned.

<i>src</i>	A pointer to the source buffer of data that is to be programmed into the flash.
<i>lengthInBytes</i>	The length, given in bytes (not words or long-words), to be programmed. Must be prince-sburegion-size-aligned.

## Returns

[kStatus\\_FLASH\\_Success](#) API was executed successfully.  
[kStatus\\_FLASH\\_InvalidArgument](#) An invalid argument is provided.  
[kStatus\\_FLASH\\_AlignmentError](#) Parameter is not aligned with the specified baseline.  
[kStatus\\_FLASH\\_AddressError](#) Address is out of range.  
[kStatus\\_FLASH\\_AccessError](#) Invalid instruction codes and out-of bounds addresses.  
[kStatus\\_FLASH\\_CommandFailure](#) Run-time error during the command execution.  
[kStatus\\_FLASH\\_CommandFailure](#) Run-time error during the command execution.  
[kStatus\\_FLASH\\_CommandNotSupported](#) Flash API is not supported.  
[kStatus\\_FLASH\\_EccError](#) A correctable or uncorrectable error during command execution.  
[kStatus\\_FLASH\\_SizeError](#) Encrypted flash subregions are not programmed at once.

# Chapter 34

## PUF: Physical Unclonable Function

### 34.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Physical Unclonable Function (PUF) module of MCUXpresso SDK devices. The PUF controller provides a secure key storage without injecting or provisioning device unique PUF root key.

Blocking synchronous APIs are provided for generating the activation code, intrinsic key generation, storing and reconstructing keys using PUF hardware. The PUF operations are complete (and results are made available for further usage) when a function returns. When called, these functions do not return until an PUF operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

### 34.2 PUF Driver Initialization and deinitialization

PUF Driver is initialized by calling the PUF\_Init() function, it resets the PUF module, enables its clock and enables power to PUF SRAM. PUF Driver is deinitialized by calling the PUF\_Deinit() function, it disables PUF module clock, asserts peripheral reset and disables power to PUF SRAM.

### 34.3 Comments about API usage in RTOS

PUF operations provided by this driver are not re-entrant. Thus, application software shall ensure the PUF module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

### 34.4 Comments about API usage in interrupt handler

All APIs can be used from interrupt handler although execution time shall be considered (interrupt latency of equal and lower priority interrupts increases).

### 34.5 PUF Driver Examples

#### 34.5.1 Simple examples

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/puf

#### Macros

- #define `PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x)` `((160u + (((((x) << 3) + 255u) >> 8) << 8)) >> 3)`  
*Get Key Code size in bytes from key size in bytes at compile time.*

## Enumerations

- enum `puf_key_slot_t` {
   
    `kPUF_KeySlot0` = 0U,  
`kPUF_KeySlot1` = 1U }
   
*PUF key slot.*
- enum
   
*PUF status return codes.*

## Driver version

- #define `FSL_PUF_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 6)`)
   
*PUF driver version.*

## 34.6 Macro Definition Documentation

### 34.6.1 #define FSL\_PUF\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 6))

Version 2.1.6.

Current version: 2.1.6

Change log:

- 2.0.0
  - Initial version.
- 2.0.1
  - Fixed `puf_wait_usec` function optimization issue.
- 2.0.2
  - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.
- 2.0.3
  - Fix MISRA C-2012 issue.
- 2.1.0
  - Align driver with PUF SRAM controller registers on LPCXpresso55s16.
  - Update initialization logic .
- 2.1.1
  - Fix ARMGCC build warning .
- 2.1.2
  - Update: Add automatic big to little endian swap for user (pre-shared) keys destined to secret hardware bus (PUF key index 0).
- 2.1.3
  - Fix MISRA C-2012 issue.
- 2.1.4
  - Replace register `uint32_t ticksCount` with volatile `uint32_t ticksCount` in `puf_wait_usec()` to prevent optimization out delay loop.
- 2.1.5
  - Use common SDK delay in `puf_wait_usec()`

- 2.1.6
  - Changed wait time in PUF\_Init(), when initialization fails it will try PUF\_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.

**34.6.2 #define PUF\_GET\_KEY\_CODE\_SIZE\_FOR\_KEY\_SIZE( *x* ) ((160u + (((((*x*)<<3) + 255u)>>8)<<8))>>3)**

## 34.7 Enumeration Type Documentation

### 34.7.1 enum puf\_key\_slot\_t

Enumerator

*kPUF\_KeySlot0* PUF key slot 0.

*kPUF\_KeySlot1* PUF key slot 1.

### 34.7.2 anonymous enum

# Chapter 35

## RNG: Random Number Generator

### 35.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Random Number Generator module of MCUXpresso SDK devices.

The Random Number Generator is a hardware module that generates 32-bit random numbers. A typical consumer is a pseudo random number generator (PRNG) which can be implemented to achieve both true randomness and cryptographic strength random numbers using the RNG output as its entropy seed. The data generated by a RNG is intended for direct use by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

### 35.2 Get random data from RNG

1. [RNG\\_GetRandomData\(\)](#) function gets random data from the RNG module.

This example code shows how to get 128-bit random data from the RNG driver.

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/rng

### Functions

- void [RNG\\_Init](#) (RNG\_Type \*base)  
*Initializes the RNG.*
- void [RNG\\_Deinit](#) (RNG\_Type \*base)  
*Shuts down the RNG.*
- status\_t [RNG\\_GetRandomData](#) (RNG\_Type \*base, void \*data, size\_t dataSize)  
*Gets random data.*
- static uint32\_t [RNG\\_GetRandomWord](#) (RNG\_Type \*base)  
*Returns random 32-bit number.*

### Driver version

- #define [FSL\\_RNG\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 3))  
*RNG driver version.*

### 35.3 Macro Definition Documentation

#### 35.3.1 #define FSL\_RNG\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))

Version 2.0.3.

Current version: 2.0.3

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Fix MISRA C-2012 issue.
- Version 2.0.2
  - Add RESET\_PeripheralReset function inside RNG\_Init and RNG\_Deinit functions.
- Version 2.0.3
  - Modified RNG\_Init and RNG\_GetRandomData functions, added rng\_accumulateEntropy and rng\_readEntropy functions.
  - These changes are reflecting recommended usage of RNG according to device UM.

## 35.4 Function Documentation

### 35.4.1 void RNG\_Init ( RNG\_Type \* *base* )

This function initializes the RNG. When called, the RNG module and ring oscillator is enabled.

Parameters

<i>base</i>	RNG base address
-------------	------------------

Returns

If successful, returns the kStatus\_RNG\_Success. Otherwise, it returns an error.

### 35.4.2 void RNG\_Deinit ( RNG\_Type \* *base* )

This function shuts down the RNG.

Parameters

<i>base</i>	RNG base address.
-------------	-------------------

### 35.4.3 status\_t RNG\_GetRandomData ( RNG\_Type \* *base*, void \* *data*, size\_t *dataSize* )

This function gets random data from the RNG.

Parameters

<i>base</i>	RNG base address.
<i>data</i>	Pointer address used to store random data.
<i>dataSize</i>	Size of the buffer pointed by the data parameter.

Returns

random data

### 35.4.4 static uint32\_t RNG\_GetRandomWord( RNG\_Type \* *base* ) [inline], [static]

This function gets random number from the RNG.

Parameters

<i>base</i>	RNG base address.
-------------	-------------------

Returns

random number

# Chapter 36

## SCTimer: SCTimer/PWM (SCT)

### 36.1 Overview

The MCUXpresso SDK provides a driver for the SCTimer Module (SCT) of MCUXpresso SDK devices.

### 36.2 Function groups

The SCTimer driver supports the generation of PWM signals. The driver also supports enabling events in various states of the SCTimer and the actions that will be triggered when an event occurs.

#### 36.2.1 Initialization and deinitialization

The function [SCTIMER\\_Init\(\)](#) initializes the SCTimer with specified configurations. The function [SCTIMER\\_GetDefaultConfig\(\)](#) gets the default configurations.

The function [SCTIMER\\_Deinit\(\)](#) halts the SCTimer counter and turns off the module clock.

#### 36.2.2 PWM Operations

The function [SCTIMER\\_SetupPwm\(\)](#) sets up SCTimer channels for PWM output. The function can set up the PWM signal properties duty cycle and level-mode (active low or high) to use. However, the same PWM period and PWM mode (edge or center-aligned) is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 1 and 100.

The function [SCTIMER\\_UpdatePwmDutycycle\(\)](#) updates the PWM signal duty cycle of a particular SCTimer channel.

#### 36.2.3 Status

Provides functions to get and clear the SCTimer status.

#### 36.2.4 Interrupt

Provides functions to enable/disable SCTimer interrupts and get current enabled interrupts.

## 36.3 SCTimer State machine and operations

The SCTimer has 10 states and each state can have a set of events enabled that can trigger a user specified action when the event occurs.

### 36.3.1 SCTimer event operations

The user can create an event and enable it in the current state using the functions [SCTIMER\\_CreateAndScheduleEvent\(\)](#) and [SCTIMER\\_ScheduleEvent\(\)](#). [SCTIMER\\_CreateAndScheduleEvent\(\)](#) creates a new event based on the users preference and enables it in the current state. [SCTIMER\\_ScheduleEvent\(\)](#) enables an event created earlier in the current state.

### 36.3.2 SCTimer state operations

The user can get the current state number by calling [SCTIMER\\_GetCurrentState\(\)](#), they can use this state number to set state transitions when a particular event is triggered.

Once the user has created and enabled events for the current state they can go to the next state by calling the function [SCTIMER\\_IncreaseState\(\)](#). The user can then start creating events to be enabled in this new state.

### 36.3.3 SCTimer action operations

There are a set of functions that decide what action should be taken when an event is triggered. [SCTIMER\\_SetupCaptureAction\(\)](#) sets up which counter to capture and which capture register to read on event trigger. [SCTIMER\\_SetupNextStateAction\(\)](#) sets up which state the SCTimer state machine should transition to on event trigger. [SCTIMER\\_SetupOutputSetAction\(\)](#) sets up which pin to set on event trigger. [SCTIMER\\_SetupOutputClearAction\(\)](#) sets up which pin to clear on event trigger. [SCTIMER\\_SetupOutputToggleAction\(\)](#) sets up which pin to toggle on event trigger. [SCTIMER\\_SetupCounterLimitAction\(\)](#) sets up which counter will be limited on event trigger. [SCTIMER\\_SetupCounterStopAction\(\)](#) sets up which counter will be stopped on event trigger. [SCTIMER\\_SetupCounterStartAction\(\)](#) sets up which counter will be started on event trigger. [SCTIMER\\_SetupCounterHaltAction\(\)](#) sets up which counter will be halted on event trigger. [SCTIMER\\_SetupDmaTriggerAction\(\)](#) sets up which DMA request will be activated on event trigger.

## 36.4 16-bit counter mode

The SCTimer is configurable to run as two 16-bit counters via the enableCounterUnify flag that is available in the configuration structure passed in to the [SCTIMER\\_Init\(\)](#) function.

When operating in 16-bit mode, it is important the user specify the appropriate counter to use when working with the functions: [SCTIMER\\_StartTimer\(\)](#), [SCTIMER\\_StopTimer\(\)](#), [SCTIMER\\_CreateAndScheduleEvent\(\)](#), [SCTIMER\\_SetupCaptureAction\(\)](#), [SCTIMER\\_SetupCounterLimitAction\(\)](#), [SCTIM-](#)

`ER_SetupCounterStopAction()`, `SCTIMER_SetupCounterStartAction()`, and `SCTIMER_SetupCounterHaltAction()`.

## 36.5 Typical use case

### 36.5.1 PWM output

Output a PWM signal on 2 SCTimer channels with different duty cycles. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/sctimer`

## Files

- file `fsl_sctimer.h`

## Data Structures

- struct `sctimer_pwm_signal_param_t`  
*Options to configure a SCTimer PWM signal. [More...](#)*
- struct `sctimer_config_t`  
*SCTimer configuration structure. [More...](#)*

## Typedefs

- typedef void(\* `sctimer_event_callback_t` )(void)  
*SCTimer callback typedef.*

## Enumerations

- enum `sctimer_pwm_mode_t` {
   
  `kSCTIMER_EdgeAlignedPwm` = 0U,
   
  `kSCTIMER_CenterAlignedPwm` }
   
*SCTimer PWM operation modes.*
- enum `sctimer_counter_t` {
   
  `kSCTIMER_Counter_L` = (1U << 0),
   
  `kSCTIMER_Counter_H` = (1U << 1),
   
  `kSCTIMER_Counter_U` = (1U << 2) }
   
*SCTimer counters type.*
- enum `sctimer_input_t` {
   
  `kSCTIMER_Input_0` = 0U,
   
  `kSCTIMER_Input_1`,
   
  `kSCTIMER_Input_2`,
   
  `kSCTIMER_Input_3`,
   
  `kSCTIMER_Input_4`,
   
  `kSCTIMER_Input_5`,
   
  `kSCTIMER_Input_6`,
   
  `kSCTIMER_Input_7` }
   
*List of SCTimer input pins.*

- enum `sctimer_out_t` {
   
kSCTIMER\_Out\_0 = 0U,
   
kSCTIMER\_Out\_1,
   
kSCTIMER\_Out\_2,
   
kSCTIMER\_Out\_3,
   
kSCTIMER\_Out\_4,
   
kSCTIMER\_Out\_5,
   
kSCTIMER\_Out\_6,
   
kSCTIMER\_Out\_7,
   
kSCTIMER\_Out\_8,
   
kSCTIMER\_Out\_9 }

*List of SCTimer output pins.*

- enum `sctimer_pwm_level_select_t` {
   
kSCTIMER\_LowTrue = 0U,
   
kSCTIMER\_HighTrue }
- SCTimer PWM output pulse mode: high-true, low-true or no output.*
- enum `sctimer_clock_mode_t` {
   
kSCTIMER\_System\_ClockMode = 0U,
   
kSCTIMER\_Sampled\_ClockMode,
   
kSCTIMER\_Input\_ClockMode,
   
kSCTIMER\_Asynchronous\_ClockMode }
- SCTimer clock mode options.*

- enum `sctimer_clock_select_t` {
   
kSCTIMER\_Clock\_On\_Rise\_Input\_0 = 0U,
   
kSCTIMER\_Clock\_On\_Fall\_Input\_0,
   
kSCTIMER\_Clock\_On\_Rise\_Input\_1,
   
kSCTIMER\_Clock\_On\_Fall\_Input\_1,
   
kSCTIMER\_Clock\_On\_Rise\_Input\_2,
   
kSCTIMER\_Clock\_On\_Fall\_Input\_2,
   
kSCTIMER\_Clock\_On\_Rise\_Input\_3,
   
kSCTIMER\_Clock\_On\_Fall\_Input\_3,
   
kSCTIMER\_Clock\_On\_Rise\_Input\_4,
   
kSCTIMER\_Clock\_On\_Fall\_Input\_4,
   
kSCTIMER\_Clock\_On\_Rise\_Input\_5,
   
kSCTIMER\_Clock\_On\_Fall\_Input\_5,
   
kSCTIMER\_Clock\_On\_Rise\_Input\_6,
   
kSCTIMER\_Clock\_On\_Fall\_Input\_6,
   
kSCTIMER\_Clock\_On\_Rise\_Input\_7,
   
kSCTIMER\_Clock\_On\_Fall\_Input\_7 }

*SCTimer clock select options.*

- enum `sctimer_conflict_resolution_t` {
   
kSCTIMER\_ResolveNone = 0U,
   
kSCTIMER\_ResolveSet,
   
kSCTIMER\_ResolveClear,
   
kSCTIMER\_ResolveToggle }

*SCTimer output conflict resolution options.*

- enum `sctimer_event_active_direction_t` {
   
  `kSCTIMER_ActiveIndependent` = 0U,
   
  `kSCTIMER_ActiveInCountUp`,
   
  `kSCTIMER_ActiveInCountDown` }

*List of SCTimer event generation active direction when the counters are operating in BIDIR mode.*

- enum `sctimer_event_t`

*List of SCTimer event types.*

- enum `sctimer_interrupt_enable_t` {
   
  `kSCTIMER_Event0InterruptEnable` = (1U << 0),
   
  `kSCTIMER_Event1InterruptEnable` = (1U << 1),
   
  `kSCTIMER_Event2InterruptEnable` = (1U << 2),
   
  `kSCTIMER_Event3InterruptEnable` = (1U << 3),
   
  `kSCTIMER_Event4InterruptEnable` = (1U << 4),
   
  `kSCTIMER_Event5InterruptEnable` = (1U << 5),
   
  `kSCTIMER_Event6InterruptEnable` = (1U << 6),
   
  `kSCTIMER_Event7InterruptEnable` = (1U << 7),
   
  `kSCTIMER_Event8InterruptEnable` = (1U << 8),
   
  `kSCTIMER_Event9InterruptEnable` = (1U << 9),
   
  `kSCTIMER_Event10InterruptEnable` = (1U << 10),
   
  `kSCTIMER_Event11InterruptEnable` = (1U << 11),
   
  `kSCTIMER_Event12InterruptEnable` = (1U << 12) }

*List of SCTimer interrupts.*

- enum `sctimer_status_flags_t` {
   
  `kSCTIMER_Event0Flag` = (1U << 0),
   
  `kSCTIMER_Event1Flag` = (1U << 1),
   
  `kSCTIMER_Event2Flag` = (1U << 2),
   
  `kSCTIMER_Event3Flag` = (1U << 3),
   
  `kSCTIMER_Event4Flag` = (1U << 4),
   
  `kSCTIMER_Event5Flag` = (1U << 5),
   
  `kSCTIMER_Event6Flag` = (1U << 6),
   
  `kSCTIMER_Event7Flag` = (1U << 7),
   
  `kSCTIMER_Event8Flag` = (1U << 8),
   
  `kSCTIMER_Event9Flag` = (1U << 9),
   
  `kSCTIMER_Event10Flag` = (1U << 10),
   
  `kSCTIMER_Event11Flag` = (1U << 11),
   
  `kSCTIMER_Event12Flag` = (1U << 12),
   
  `kSCTIMER_BusErrorLFlag`,
   
  `kSCTIMER_BusErrorHFlag` }

*List of SCTimer flags.*

## Driver version

- #define `FSL_SCTIMER_DRIVER_VERSION` (`MAKE_VERSION(2, 4, 6)`)
- Version.*

## Initialization and deinitialization

- `status_t SCTIMER_Init (SCT_Type *base, const sctimer_config_t *config)`  
*Ungates the SCTimer clock and configures the peripheral for basic operation.*
- `void SCTIMER_Deinit (SCT_Type *base)`  
*Gates the SCTimer clock.*
- `void SCTIMER_GetDefaultConfig (sctimer_config_t *config)`  
*Fills in the SCTimer configuration structure with the default settings.*

## PWM setup operations

- `status_t SCTIMER_SetupPwm (SCT_Type *base, const sctimer_pwm_signal_param_t *pwmParams, sctimer_pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *event)`  
*Configures the PWM signal parameters.*
- `void SCTIMER_UpdatePwmDutyCycle (SCT_Type *base, sctimer_out_t output, uint8_t dutyCyclePercent, uint32_t event)`  
*Updates the duty cycle of an active PWM signal.*

## Interrupt Interface

- `static void SCTIMER_EnableInterrupts (SCT_Type *base, uint32_t mask)`  
*Enables the selected SCTimer interrupts.*
- `static void SCTIMER_DisableInterrupts (SCT_Type *base, uint32_t mask)`  
*Disables the selected SCTimer interrupts.*
- `static uint32_t SCTIMER_GetEnabledInterrupts (SCT_Type *base)`  
*Gets the enabled SCTimer interrupts.*

## Status Interface

- `static uint32_t SCTIMER_GetStatusFlags (SCT_Type *base)`  
*Gets the SCTimer status flags.*
- `static void SCTIMER_ClearStatusFlags (SCT_Type *base, uint32_t mask)`  
*Clears the SCTimer status flags.*

## Counter Start and Stop

- `static void SCTIMER_StartTimer (SCT_Type *base, uint32_t countertoStart)`  
*Starts the SCTimer counter.*
- `static void SCTIMER_StopTimer (SCT_Type *base, uint32_t countertoStop)`  
*Halts the SCTimer counter.*

## Functions to create a new event and manage the state logic

- `status_t SCTIMER_CreateAndScheduleEvent (SCT_Type *base, sctimer_event_t howToMonitor, uint32_t matchValue, uint32_t whichIO, sctimer_counter_t whichCounter, uint32_t *event)`  
*Create an event that is triggered on a match or IO and schedule in current state.*
- `void SCTIMER_ScheduleEvent (SCT_Type *base, uint32_t event)`  
*Enable an event in the current state.*
- `status_t SCTIMER_IncreaseState (SCT_Type *base)`

- `uint32_t SCTIMER_GetCurrentState (SCT_Type *base)`  
*Provides the current state.*
- `static void SCTIMER_SetCounterState (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t state)`  
*Set the counter current state.*
- `static uint16_t SCTIMER_GetCounterState (SCT_Type *base, sctimer_counter_t whichCounter)`  
*Get the counter current state value.*

## Actions to take in response to an event

- `status_t SCTIMER_SetupCaptureAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t *captureRegister, uint32_t event)`  
*Setup capture of the counter value on trigger of a selected event.*
- `void SCTIMER_SetCallback (SCT_Type *base, sctimer_event_callback_t callback, uint32_t event)`  
*Receive notification when the event trigger an interrupt.*
- `static void SCTIMER_SetupStateLdMethodAction (SCT_Type *base, uint32_t event, bool fgLoad)`  
*Change the load method of transition to the specified state.*
- `static void SCTIMER_SetupNextStateActionWithLdMethod (SCT_Type *base, uint32_t nextState, uint32_t event, bool fgLoad)`  
*Transition to the specified state with Load method.*
- `static void SCTIMER_SetupNextStateAction (SCT_Type *base, uint32_t nextState, uint32_t event)`  
*Transition to the specified state.*
- `static void SCTIMER_SetupEventActiveDirection (SCT_Type *base, sctimer_event_active_direction_t activeDirection, uint32_t event)`  
*Setup event active direction when the counters are operating in BIDIR mode.*
- `static void SCTIMER_SetupOutputSetAction (SCT_Type *base, uint32_t whichIO, uint32_t event)`  
*Set the Output.*
- `static void SCTIMER_SetupOutputClearAction (SCT_Type *base, uint32_t whichIO, uint32_t event)`  
*Clear the Output.*
- `void SCTIMER_SetupOutputToggleAction (SCT_Type *base, uint32_t whichIO, uint32_t event)`  
*Toggle the output level.*
- `static void SCTIMER_SetupCounterLimitAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)`  
*Limit the running counter.*
- `static void SCTIMER_SetupCounterStopAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)`  
*Stop the running counter.*
- `static void SCTIMER_SetupCounterStartAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)`  
*Re-start the stopped counter.*
- `static void SCTIMER_SetupCounterHaltAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)`  
*Halt the running counter.*
- `static void SCTIMER_SetupDmaTriggerAction (SCT_Type *base, uint32_t dmaNumber, uint32_t event)`  
*Generate a DMA request.*
- `static void SCTIMER_SetCOUNTValue (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t value)`

- static uint32\_t **SCTIMER\_GetCOUNTValue** (SCT\_Type \*base, **sctimer\_counter\_t** whichCounter)
 

*Set the value of counter.*
- static void **SCTIMER\_SetEventInState** (SCT\_Type \*base, uint32\_t event, uint32\_t state)
 

*Get the value of counter.*
- static void **SCTIMER\_ClearEventInState** (SCT\_Type \*base, uint32\_t event, uint32\_t state)
 

*Set the state mask bit field of EV\_STATE register.*
- static bool **SCTIMER\_GetEventInState** (SCT\_Type \*base, uint32\_t event, uint32\_t state)
 

*Clear the state mask bit field of EV\_STATE register.*
- void **SCTIMER\_EventHandleIRQ** (SCT\_Type \*base)
 

*Get the state mask bit field of EV\_STATE register.*

*SCTimer interrupt handler.*

## 36.6 Data Structure Documentation

### 36.6.1 struct sctimer\_pwm\_signal\_param\_t

#### Data Fields

- **sctimer\_out\_t output**

*The output pin to use to generate the PWM signal.*
- **sctimer\_pwm\_level\_select\_t level**

*PWM output active level select.*
- **uint8\_t dutyCyclePercent**

*PWM pulse width, value should be between 0 to 100 0 = always inactive signal (0% duty cycle) 100 = always active signal (100% duty cycle).*

#### Field Documentation

- (1) **sctimer\_pwm\_level\_select\_t sctimer\_pwm\_signal\_param\_t::level**
- (2) **uint8\_t sctimer\_pwm\_signal\_param\_t::dutyCyclePercent**

### 36.6.2 struct sctimer\_config\_t

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the **SCTMR\_GetDefaultConfig()** function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

#### Data Fields

- **bool enableCounterUnify**

*true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters.*
- **sctimer\_clock\_mode\_t clockMode**

*SCT clock mode value.*
- **sctimer\_clock\_select\_t clockSelect**

*SCT clock select value.*

- bool `enableBidirection_l`  
*true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter*
- bool `enableBidirection_h`  
*true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter*
- uint8\_t `prescale_l`  
*Prescale value to produce the L or unified counter clock.*
- uint8\_t `prescale_h`  
*Prescale value to produce the H counter clock.*
- uint8\_t `outInitState`  
*Defines the initial output value.*
- uint8\_t `inputsync`  
*SCT INSYNC value, INSYNC field in the CONFIG register, from bit9 to bit 16.*

## Field Documentation

### (1) `bool sctimer_config_t::enableCounterUnify`

User can use the 16-bit low counter and the 16-bit high counters at the same time; for Hardware limit, user can not use unified 32-bit counter and any 16-bit low/high counter at the same time.

### (2) `bool sctimer_config_t::enableBidirection_h`

This field is used only if the enableCounterUnify is set to false

### (3) `uint8_t sctimer_config_t::prescale_h`

This field is used only if the enableCounterUnify is set to false

### (4) `uint8_t sctimer_config_t::inputsync`

it is used to define synchronization for input N: bit 9 = input 0 bit 10 = input 1 bit 11 = input 2 bit 12 = input 3 All other bits are reserved (bit13 ~bit 16). How User to set the the value for the member inputsync. IE: delay for input0, and input 1, bypasses for input 2 and input 3 MACRO definition in user level. #define INPUTSYNC0 (0U) #define INPUTSYNC1 (1U) #define INPUTSYNC2 (2U) #define INPUTSYNC3 (3U) User Code. sctimerInfo.inputsync = (1 << INPUTSYNC2) | (1 << INPUTSYNC3);

## 36.7 Typedef Documentation

### 36.7.1 `typedef void(* sctimer_event_callback_t)(void)`

## 36.8 Enumeration Type Documentation

### 36.8.1 `enum sctimer_pwm_mode_t`

Enumerator

***kSCTIMER\_EdgeAlignedPwm*** Edge-aligned PWM.

***kSCTIMER\_CenterAlignedPwm*** Center-aligned PWM.

### 36.8.2 enum sctimer\_counter\_t

Enumerator

*kSCTIMER\_Counter\_L* 16-bit Low counter.  
*kSCTIMER\_Counter\_H* 16-bit High counter.  
*kSCTIMER\_Counter\_U* 32-bit Unified counter.

### 36.8.3 enum sctimer\_input\_t

Enumerator

*kSCTIMER\_Input\_0* SCTIMER input 0.  
*kSCTIMER\_Input\_1* SCTIMER input 1.  
*kSCTIMER\_Input\_2* SCTIMER input 2.  
*kSCTIMER\_Input\_3* SCTIMER input 3.  
*kSCTIMER\_Input\_4* SCTIMER input 4.  
*kSCTIMER\_Input\_5* SCTIMER input 5.  
*kSCTIMER\_Input\_6* SCTIMER input 6.  
*kSCTIMER\_Input\_7* SCTIMER input 7.

### 36.8.4 enum sctimer\_out\_t

Enumerator

*kSCTIMER\_Out\_0* SCTIMER output 0.  
*kSCTIMER\_Out\_1* SCTIMER output 1.  
*kSCTIMER\_Out\_2* SCTIMER output 2.  
*kSCTIMER\_Out\_3* SCTIMER output 3.  
*kSCTIMER\_Out\_4* SCTIMER output 4.  
*kSCTIMER\_Out\_5* SCTIMER output 5.  
*kSCTIMER\_Out\_6* SCTIMER output 6.  
*kSCTIMER\_Out\_7* SCTIMER output 7.  
*kSCTIMER\_Out\_8* SCTIMER output 8.  
*kSCTIMER\_Out\_9* SCTIMER output 9.

### 36.8.5 enum sctimer\_pwm\_level\_select\_t

Enumerator

*kSCTIMER\_LowTrue* Low true pulses.  
*kSCTIMER\_HighTrue* High true pulses.

### 36.8.6 enum sctimer\_clock\_mode\_t

Enumerator

- kSCTIMER\_System\_ClockMode*** System Clock Mode.
- kSCTIMER\_Sampled\_ClockMode*** Sampled System Clock Mode.
- kSCTIMER\_Input\_ClockMode*** SCT Input Clock Mode.
- kSCTIMER\_Asynchronous\_ClockMode*** Asynchronous Mode.

### 36.8.7 enum sctimer\_clock\_select\_t

Enumerator

- kSCTIMER\_Clock\_On\_Rise\_Input\_0*** Rising edges on input 0.
- kSCTIMER\_Clock\_On\_Fall\_Input\_0*** Falling edges on input 0.
- kSCTIMER\_Clock\_On\_Rise\_Input\_1*** Rising edges on input 1.
- kSCTIMER\_Clock\_On\_Fall\_Input\_1*** Falling edges on input 1.
- kSCTIMER\_Clock\_On\_Rise\_Input\_2*** Rising edges on input 2.
- kSCTIMER\_Clock\_On\_Fall\_Input\_2*** Falling edges on input 2.
- kSCTIMER\_Clock\_On\_Rise\_Input\_3*** Rising edges on input 3.
- kSCTIMER\_Clock\_On\_Fall\_Input\_3*** Falling edges on input 3.
- kSCTIMER\_Clock\_On\_Rise\_Input\_4*** Rising edges on input 4.
- kSCTIMER\_Clock\_On\_Fall\_Input\_4*** Falling edges on input 4.
- kSCTIMER\_Clock\_On\_Rise\_Input\_5*** Rising edges on input 5.
- kSCTIMER\_Clock\_On\_Fall\_Input\_5*** Falling edges on input 5.
- kSCTIMER\_Clock\_On\_Rise\_Input\_6*** Rising edges on input 6.
- kSCTIMER\_Clock\_On\_Fall\_Input\_6*** Falling edges on input 6.
- kSCTIMER\_Clock\_On\_Rise\_Input\_7*** Rising edges on input 7.
- kSCTIMER\_Clock\_On\_Fall\_Input\_7*** Falling edges on input 7.

### 36.8.8 enum sctimer\_conflict\_resolution\_t

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

Enumerator

- kSCTIMER\_ResolveNone*** No change.
- kSCTIMER\_ResolveSet*** Set output.
- kSCTIMER\_ResolveClear*** Clear output.
- kSCTIMER\_ResolveToggle*** Toggle output.

### 36.8.9 enum sctimer\_event\_active\_direction\_t

Enumerator

***kSCTIMER\_ActiveIndependent*** This event is triggered regardless of the count direction.

***kSCTIMER\_ActiveInCountUp*** This event is triggered only during up-counting when BIDIR = 1.

***kSCTIMER\_ActiveInCountDown*** This event is triggered only during down-counting when BIDIR = 1.

### 36.8.10 enum sctimer\_interrupt\_enable\_t

Enumerator

***kSCTIMER\_Event0InterruptEnable*** Event 0 interrupt.

***kSCTIMER\_Event1InterruptEnable*** Event 1 interrupt.

***kSCTIMER\_Event2InterruptEnable*** Event 2 interrupt.

***kSCTIMER\_Event3InterruptEnable*** Event 3 interrupt.

***kSCTIMER\_Event4InterruptEnable*** Event 4 interrupt.

***kSCTIMER\_Event5InterruptEnable*** Event 5 interrupt.

***kSCTIMER\_Event6InterruptEnable*** Event 6 interrupt.

***kSCTIMER\_Event7InterruptEnable*** Event 7 interrupt.

***kSCTIMER\_Event8InterruptEnable*** Event 8 interrupt.

***kSCTIMER\_Event9InterruptEnable*** Event 9 interrupt.

***kSCTIMER\_Event10InterruptEnable*** Event 10 interrupt.

***kSCTIMER\_Event11InterruptEnable*** Event 11 interrupt.

***kSCTIMER\_Event12InterruptEnable*** Event 12 interrupt.

### 36.8.11 enum sctimer\_status\_flags\_t

Enumerator

***kSCTIMER\_Event0Flag*** Event 0 Flag.

***kSCTIMER\_Event1Flag*** Event 1 Flag.

***kSCTIMER\_Event2Flag*** Event 2 Flag.

***kSCTIMER\_Event3Flag*** Event 3 Flag.

***kSCTIMER\_Event4Flag*** Event 4 Flag.

***kSCTIMER\_Event5Flag*** Event 5 Flag.

***kSCTIMER\_Event6Flag*** Event 6 Flag.

***kSCTIMER\_Event7Flag*** Event 7 Flag.

***kSCTIMER\_Event8Flag*** Event 8 Flag.

***kSCTIMER\_Event9Flag*** Event 9 Flag.

***kSCTIMER\_Event10Flag*** Event 10 Flag.

***kSCTIMER\_Event11Flag*** Event 11 Flag.

***kSCTIMER\_Event12Flag*** Event 12 Flag.

***kSCTIMER\_BusErrorLFlag*** Bus error due to write when L counter was not halted.

***kSCTIMER\_BusErrorHFlag*** Bus error due to write when H counter was not halted.

## 36.9 Function Documentation

### 36.9.1 status\_t SCTIMER\_Init ( **SCT\_Type** \* *base*, **const sctimer\_config\_t** \* *config* )

Note

This API should be called at the beginning of the application using the SCTimer driver.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>config</i>	Pointer to the user configuration structure.

Returns

**kStatus\_Success** indicates success; Else indicates failure.

### 36.9.2 void SCTIMER\_Deinit ( **SCT\_Type** \* *base* )

Parameters

<i>base</i>	SCTimer peripheral base address
-------------	---------------------------------

### 36.9.3 void SCTIMER\_GetDefaultConfig ( **sctimer\_config\_t** \* *config* )

The default values are:

```
* config->enableCounterUnify = true;
* config->clockMode = kSCTIMER_System_ClockMode;
* config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
* config->enableBidirection_l = false;
* config->enableBidirection_h = false;
* config->prescale_l = 0U;
* config->prescale_h = 0U;
* config->outInitState = 0U;
* config->inputsync = 0xFU;
*
```

## Parameters

<i>config</i>	Pointer to the user configuration structure.
---------------	--

**36.9.4 status\_t SCTIMER\_SetupPwm ( *SCT\_Type* \* *base*, const *sctimer\_pwm\_signal\_param\_t* \* *pwmParams*, *sctimer\_pwm\_mode\_t* *mode*, *uint32\_t* *pwmFreq\_Hz*, *uint32\_t* *srcClock\_Hz*, *uint32\_t* \* *event* )**

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events are enabled for the same state. The state number can be retrieved by calling the function *SCTIMER\_GetCurrentStateNumber()*. The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

## Note

When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's *pwmFreq\_Hz*.

## Parameters

<i>base</i>	SCTimer peripheral base address
<i>pwmParams</i>	PWM parameters to configure the output
<i>mode</i>	PWM operation mode, options available in enumeration <a href="#">sctimer_pwm_mode_t</a>
<i>pwmFreq_Hz</i>	PWM signal frequency in Hz
<i>srcClock_Hz</i>	SCTimer counter clock in Hz
<i>event</i>	Pointer to a variable where the PWM period event number is stored

## Returns

kStatus\_Success on success kStatus\_Fail If we have hit the limit in terms of number of events created or if an incorrect PWM dutycycle is passed in.

**36.9.5 void SCTIMER\_UpdatePwmDutycycle ( *SCT\_Type* \* *base*, *sctimer\_out\_t* *output*, *uint8\_t* *dutyCyclePercent*, *uint32\_t* *event* )**

Before calling this function, the counter is set to operate as one 32-bit counter (unify bit is set to 1).

Parameters

<i>base</i>	SCTimer peripheral base address
<i>output</i>	The output to configure
<i>dutyCycle-Percent</i>	New PWM pulse width; the value should be between 1 to 100
<i>event</i>	Event number associated with this PWM signal. This was returned to the user by the function <a href="#">SCTIMER_SetupPwm()</a> .

### 36.9.6 static void SCTIMER\_EnableInterrupts ( **SCT\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	SCTimer peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">sctimer_interrupt_enable_t</a>

### 36.9.7 static void SCTIMER\_DisableInterrupts ( **SCT\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	SCTimer peripheral base address
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">sctimer_interrupt_enable_t</a>

### 36.9.8 static **uint32\_t** SCTIMER\_GetEnabledInterrupts ( **SCT\_Type** \* *base* ) [inline], [static]

Parameters

<i>base</i>	SCTimer peripheral base address
-------------	---------------------------------

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [sctimer\\_interrupt\\_enable\\_t](#)

### 36.9.9 static uint32\_t SCTIMER\_GetStatusFlags ( **SCT\_Type** \* *base* ) [inline], [static]

Parameters

<i>base</i>	SCTimer peripheral base address
-------------	---------------------------------

Returns

The status flags. This is the logical OR of members of the enumeration [sctimer\\_status\\_flags\\_t](#)

### 36.9.10 static void SCTIMER\_ClearStatusFlags ( **SCT\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	SCTimer peripheral base address
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration <a href="#">sctimer_status_flags_t</a>

### 36.9.11 static void SCTIMER\_StartTimer ( **SCT\_Type** \* *base*, **uint32\_t** *countertoStart* ) [inline], [static]

Note

In 16-bit mode, we can enable both Counter\_L and Counter\_H, In 32-bit mode, we only can select Counter\_U.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>countertoStart</i>	The SCTimer counters to enable. This is a logical OR of members of the enumeration <a href="#">sctimer_counter_t</a> .

### 36.9.12 static void SCTIMER\_StopTimer ( *SCT\_Type* \* *base*, *uint32\_t* *countertoStop* ) [inline], [static]

Parameters

<i>base</i>	SCTimer peripheral base address
<i>countertoStop</i>	The SCTimer counters to stop. This is a logical OR of members of the enumeration <a href="#">sctimer_counter_t</a> .

### 36.9.13 status\_t SCTIMER\_CreateAndScheduleEvent ( *SCT\_Type* \* *base*, *sctimer\_event\_t* *howToMonitor*, *uint32\_t* *matchValue*, *uint32\_t* *whichIO*, *sctimer\_counter\_t* *whichCounter*, *uint32\_t* \* *event* )

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>howToMonitor</i>	Event type; options are available in the enumeration <a href="#">sctimer_interrupt_enable_t</a>
<i>matchValue</i>	The match value that will be programmed to a match register
<i>whichIO</i>	The input or output that will be involved in event triggering. This field is ignored if the event type is "match only"

<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
<i>event</i>	Pointer to a variable where the new event number is stored

Returns

kStatus\_Success on success kStatus\_Error if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

### 36.9.14 void SCTIMER\_ScheduleEvent ( **SCT\_Type \* base, uint32\_t event** )

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function [SCTIMER\\_SetupPwm\(\)](#) or function [SCTIMER\\_CreateAndScheduleEvent\(\)](#).

Parameters

<i>base</i>	SCTimer peripheral base address
<i>event</i>	Event number to enable in the current state

### 36.9.15 status\_t SCTIMER\_IncreaseState ( **SCT\_Type \* base** )

All future events created by calling the function [SCTIMER\\_ScheduleEvent\(\)](#) will be enabled in this new state.

Parameters

<i>base</i>	SCTimer peripheral base address
-------------	---------------------------------

Returns

kStatus\_Success on success kStatus\_Error if we have hit the limit in terms of states used

### 36.9.16 uint32\_t SCTIMER\_GetCurrentState ( **SCT\_Type \* base** )

User can use this to set the next state by calling the function [SCTIMER\\_SetupNextStateAction\(\)](#).

Parameters

<i>base</i>	SCTimer peripheral base address
-------------	---------------------------------

Returns

The current state

### 36.9.17 static void SCTIMER\_SetCounterState ( **SCT\_Type** \* *base*,                   **sctimer\_counter\_t** *whichCounter*, **uint32\_t** *state* ) [inline], [static]

The function is to set the state variable bit field of STATE register. Writing to the STATE\_L, STATE\_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
<i>state</i>	The counter current state number (only support range from 0~31).

### 36.9.18 static uint16\_t SCTIMER\_GetCounterState ( **SCT\_Type** \* *base*,                   **sctimer\_counter\_t** *whichCounter* ) [inline], [static]

The function is to get the state variable bit field of STATE register.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The the counter current state value.

### 36.9.19 status\_t SCTIMER\_SetupCaptureAction ( **SCT\_Type** \* *base*,                   **sctimer\_counter\_t** *whichCounter*, **uint32\_t** \* *captureRegister*, **uint32\_t**                   *event* )

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
<i>captureRegister</i>	Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
<i>event</i>	Event number that will trigger the capture

Returns

kStatus\_Success on success kStatus\_Error if we have hit the limit in terms of number of match/capture registers available

### 36.9.20 void SCTIMER\_SetCallback ( **SCT\_Type \* base**, **sctimer\_event\_callback\_t callback**, **uint32\_t event** )

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

Parameters

<i>base</i>	SCTimer peripheral base address
<i>event</i>	Event number that will trigger the interrupt
<i>callback</i>	Function to invoke when the event is triggered

### 36.9.21 static void SCTIMER\_SetupStateLdMethodAction ( **SCT\_Type \* base**, **uint32\_t event**, **bool fgLoad** ) [inline], [static]

Change the load method of transition, it will be triggered by the event number that is passed in by the user.

Parameters

<i>base</i>	SCTimer peripheral base address
-------------	---------------------------------

<i>event</i>	Event number that will change the method to trigger the state transition
<i>fgLoad</i>	<p>The method to load highest-numbered event occurring for that state to the STATE register.</p> <ul style="list-style-type: none"> <li>• true: Load the STATEV value to STATE when the event occurs to be the next state.</li> <li>• false: Add the STATEV value to STATE when the event occurs to be the next state.</li> </ul>

### 36.9.22 static void SCTIMER\_SetupNextStateActionWithLdMethod ( **SCT\_Type** \* *base*, **uint32\_t** *nextState*, **uint32\_t** *event*, **bool** *fgLoad* ) [inline], [static]

This transition will be triggered by the event number that is passed in by the user, the method decide how to load the highest-numbered event occurring for that state to the STATE register.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>nextState</i>	The next state SCTimer will transition to
<i>event</i>	Event number that will trigger the state transition
<i>fgLoad</i>	<p>The method to load the highest-numbered event occurring for that state to the STATE register.</p> <ul style="list-style-type: none"> <li>• true: Load the STATEV value to STATE when the event occurs to be the next state.</li> <li>• false: Add the STATEV value to STATE when the event occurs to be the next state.</li> </ul>

### 36.9.23 static void SCTIMER\_SetupNextStateAction ( **SCT\_Type** \* *base*, **uint32\_t** *nextState*, **uint32\_t** *event* ) [inline], [static]

**Deprecated** Do not use this function. It has been superceded by [SCTIMER\\_SetupNextStateActionWithLdMethod](#)

This transition will be triggered by the event number that is passed in by the user.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>nextState</i>	The next state SCTimer will transition to
<i>event</i>	Event number that will trigger the state transition

**36.9.24 static void SCTIMER\_SetupEventActiveDirection ( *SCT\_Type* \* *base*, *sctimer\_event\_active\_direction\_t* *activeDirection*, *uint32\_t* *event* ) [inline], [static]**

Parameters

<i>base</i>	SCTimer peripheral base address
<i>activeDirection</i>	Event generation active direction, see <a href="#">sctimer_event_active_direction_t</a> .
<i>event</i>	Event number that need setup the active direction.

**36.9.25 static void SCTIMER\_SetupOutputSetAction ( *SCT\_Type* \* *base*, *uint32\_t* *whichIO*, *uint32\_t* *event* ) [inline], [static]**

This output will be set when the event number that is passed in by the user is triggered.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichIO</i>	The output to set
<i>event</i>	Event number that will trigger the output change

**36.9.26 static void SCTIMER\_SetupOutputClearAction ( *SCT\_Type* \* *base*, *uint32\_t* *whichIO*, *uint32\_t* *event* ) [inline], [static]**

This output will be cleared when the event number that is passed in by the user is triggered.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichIO</i>	The output to clear
<i>event</i>	Event number that will trigger the output change

### 36.9.27 void SCTIMER\_SetupOutputToggleAction ( *SCT\_Type \* base*, *uint32\_t whichIO*, *uint32\_t event* )

This change in the output level is triggered by the event number that is passed in by the user.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichIO</i>	The output to toggle
<i>event</i>	Event number that will trigger the output change

### 36.9.28 static void SCTIMER\_SetupCounterLimitAction ( *SCT\_Type \* base*, *sctimer\_counter\_t whichCounter*, *uint32\_t event* ) [inline], [static]

The counter is limited when the event number that is passed in by the user is triggered.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
<i>event</i>	Event number that will trigger the counter to be limited

### 36.9.29 static void SCTIMER\_SetupCounterStopAction ( *SCT\_Type \* base*, *sctimer\_counter\_t whichCounter*, *uint32\_t event* ) [inline], [static]

The counter is stopped when the event number that is passed in by the user is triggered.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
<i>event</i>	Event number that will trigger the counter to be stopped

### 36.9.30 static void SCTIMER\_SetupCounterStartAction ( *SCT\_Type* \* *base*, *sctimer\_counter\_t* *whichCounter*, *uint32\_t* *event* ) [inline], [static]

The counter will re-start when the event number that is passed in by the user is triggered.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
<i>event</i>	Event number that will trigger the counter to re-start

### 36.9.31 static void SCTIMER\_SetupCounterHaltAction ( *SCT\_Type* \* *base*, *sctimer\_counter\_t* *whichCounter*, *uint32\_t* *event* ) [inline], [static]

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the [SCTIMER\\_StartTimer\(\)](#) function.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
<i>event</i>	Event number that will trigger the counter to be halted

### 36.9.32 static void SCTIMER\_SetupDmaTriggerAction ( *SCT\_Type* \* *base*, *uint32\_t* *dmaNumber*, *uint32\_t* *event* ) [inline], [static]

DMA request will be triggered by the event number that is passed in by the user.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>dmaNumber</i>	The DMA request to generate
<i>event</i>	Event number that will trigger the DMA request

**36.9.33 static void SCTIMER\_SetCOUNTValue ( **SCT\_Type** \* *base*,  
                  sctimer\_counter\_t *whichCounter*, uint32\_t *value* ) [inline], [static]**

The function is to set the value of Count register, Writing to the COUNT\_L, COUNT\_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
<i>value</i>	the counter value update to the COUNT register.

### 36.9.34 static uint32\_t SCTIMER\_GetCOUNTValue ( *SCT\_Type \* base*, *sctimer\_counter\_t whichCounter* ) [inline], [static]

The function is to read the value of Count register, software can read the counter registers at any time..

Parameters

<i>base</i>	SCTimer peripheral base address
<i>whichCounter</i>	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The value of counter selected.

### 36.9.35 static void SCTIMER\_SetEventInState ( *SCT\_Type \* base*, *uint32\_t event*, *uint32\_t state* ) [inline], [static]

Parameters

<i>base</i>	SCTimer peripheral base address
<i>event</i>	The EV_STATE register be set.
<i>state</i>	The state value in which the event is enabled to occur.

### 36.9.36 static void SCTIMER\_ClearEventInState ( *SCT\_Type \* base*, *uint32\_t event*, *uint32\_t state* ) [inline], [static]

Parameters

<i>base</i>	SCTimer peripheral base address
<i>event</i>	The EV_STATE register be clear.
<i>state</i>	The state value in which the event is disabled to occur.

### 36.9.37 static bool SCTIMER\_GetEventInState ( **SCT\_Type** \* *base*, **uint32\_t** *event*, **uint32\_t** *state* ) [inline], [static]

Note

This function is to check whether the event is enabled in a specific state.

Parameters

<i>base</i>	SCTimer peripheral base address
<i>event</i>	The EV_STATE register be read.
<i>state</i>	The state value.

Returns

The the state mask bit field of EV\_STATE register.

- true: The event is enable in state.
- false: The event is disable in state.

### 36.9.38 void SCTIMER\_EventHandleIRQ ( **SCT\_Type** \* *base* )

Parameters

<i>base</i>	SCTimer peripheral base address.
-------------	----------------------------------

# Chapter 37

## SDIF: SD/MMC/SDIO card interface

### 37.1 Overview

The MCUXpresso SDK provides a peripheral driver for the SD/MMC/SDIO card interface (sdif) module of MCUXpresso SDK devices.

### 37.2 Typical use case

#### 37.2.1 sdif Operation

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/sdif

## Data Structures

- struct `sdif_dma_descriptor_t`  
*define the internal DMA descriptor [More...](#)*
- struct `sdif_dma_config_t`  
*Defines the internal DMA configure structure. [More...](#)*
- struct `sdif_data_t`  
*Card data descriptor. [More...](#)*
- struct `sdif_command_t`  
*Card command descriptor. [More...](#)*
- struct `sdif_transfer_t`  
*Transfer state. [More...](#)*
- struct `sdif_config_t`  
*Data structure to initialize the sdif. [More...](#)*
- struct `sdif_capability_t`  
*SDIF capability information. [More...](#)*
- struct `sdif_transfer_callback_t`  
*sdif callback functions. [More...](#)*
- struct `sdif_handle_t`  
*sdif handle [More...](#)*
- struct `sdif_host_t`  
*sdif host descriptor [More...](#)*

## Macros

- #define `SDIF_CLOCK_RANGE_NEED_DELAY` (50000000U)  
*SDIOCLKCTRL setting Below clock delay setting should depend on specific platform, so it can be redefined when timing mismatch issue occur.*
- #define `SDIF_HIGHSPEED_SAMPLE_DELAY` (12U)  
*High speed mode clk\_sample fixed delay.*
- #define `SDIF_HIGHSPEED_DRV_DELAY` (31U)  
*High speed mode clk\_drv fixed delay.*

- #define **SDIF\_HIGHSPEED\_SAMPLE\_PHASE\_SHIFT** (0U)  
*High speed mode clk\_sample phase shift.*
- #define **SDIF\_HIGHSPEED\_DRV\_PHASE\_SHIFT** (1U) /\* 90 degrees clk\_drv phase delay \*/  
*High speed mode clk\_drv phase shift.*
- #define **SDIF\_DEFAULT\_MODE\_SAMPLE\_DELAY** (12U)  
*default mode sample fixed delay*
- #define **SDIF\_DEFAULT\_MODE\_DRV\_DELAY** (31U)  
 *$31 * 250\text{ps} = 7.75\text{ns}$*
- #define **SDIF\_INTERNAL\_DMA\_ADDR\_ALIGN** (4U)  
*SDIF internal DMA descriptor address and the data buffer address align.*
- #define **SDIF\_DMA\_DESCRIPTOR\_DISABLE\_COMPLETE\_INT\_FLAG** (1UL << 1U)  
*SDIF DMA descriptor flag.*

## Typedefs

- typedef **status\_t**(\* **sdif\_transfer\_function\_t** )(SDIF\_Type \*base, **sdif\_transfer\_t** \*content)  
*sdif transfer function.*

## Enumerations

- enum {
 **kStatus\_SDIF\_DescriptorBufferLenError** = MAKE\_STATUS(kStatusGroup\_SDIF, 0U),
 **kStatus\_SDIF\_InvalidArgument** = MAKE\_STATUS(kStatusGroup\_SDIF, 1U),
 **kStatus\_SDIF\_SyncCmdTimeout** = MAKE\_STATUS(kStatusGroup\_SDIF, 2U),
 **kStatus\_SDIF\_SendCmdFail** = MAKE\_STATUS(kStatusGroup\_SDIF, 3U),
 **kStatus\_SDIF\_SendCmdErrorBufferFull**,
 **kStatus\_SDIF\_DMATransferFailWithFBE**,
 **kStatus\_SDIF\_DMATransferDescriptorUnavailable**,
 **kStatus\_SDIF\_DataTransferFail** = MAKE\_STATUS(kStatusGroup\_SDIF, 6U),
 **kStatus\_SDIF\_ResponseError** = MAKE\_STATUS(kStatusGroup\_SDIF, 7U),
 **kStatus\_SDIF\_DMAAddrNotAlign** = MAKE\_STATUS(kStatusGroup\_SDIF, 8U),
 **kStatus\_SDIF\_BusyTransferring** = MAKE\_STATUS(kStatusGroup\_SDIF, 9U),
 **kStatus\_SDIF\_DataTransferSuccess** = MAKE\_STATUS(kStatusGroup\_SDIF, 10U),
 **kStatus\_SDIF\_SendCmdSuccess** = MAKE\_STATUS(kStatusGroup\_SDIF, 11U) }
   
*\_sdif\_status SDIF status*
- enum {
 **kSDIF\_SupportHighSpeedFlag** = 0x1U,
 **kSDIF\_SupportDmaFlag** = 0x2U,
 **kSDIF\_SupportSuspendResumeFlag** = 0x4U,
 **kSDIF\_SupportV330Flag** = 0x8U,
 **kSDIF\_Support4BitFlag** = 0x10U,
 **kSDIF\_Support8BitFlag** = 0x20U }
   
*\_sdif\_capability\_flag Host controller capabilities flag mask*
- enum {
 **kSDIF\_ResetController** = SDIF\_CTRL\_CONTROLLER\_RESET\_MASK,
 **kSDIF\_ResetFIFO** = SDIF\_CTRL\_FIFO\_RESET\_MASK,
 **kSDIF\_ResetDMAInterface** = SDIF\_CTRL\_DMA\_RESET\_MASK,
 **kSDIF\_ResetAll** }

- *\_sdif\_reset\_type define the reset type*
- enum `sdif_bus_width_t` {
   
    `kSDIF_Bus1BitWidth` = 0U,  
`kSDIF_Bus4BitWidth` = 1U,  
`kSDIF_Bus8BitWidth` = 2U }
   
*define the card bus width type*
- enum {
   
    `kSDIF_CmdResponseExpect` = SDIF\_CMD\_RESPONSE\_EXPECT\_MASK,  
`kSDIF_CmdResponseLengthLong` = SDIF\_CMD\_RESPONSE\_LENGTH\_MASK,  
`kSDIF_CmdCheckResponseCRC` = SDIF\_CMD\_CHECK\_RESPONSE\_CRC\_MASK,  
`kSDIF_DataExpect` = SDIF\_CMD\_DATA\_EXPECTED\_MASK,  
`kSDIF_DataWriteToCard` = SDIF\_CMD\_READ\_WRITE\_MASK,  
`kSDIFDataStreamTransfer` = SDIF\_CMD\_TRANSFER\_MODE\_MASK,  
`kSDIF_DataTransferAutoStop` = SDIF\_CMD\_SEND\_AUTO\_STOP\_MASK,  
`kSDIF_WaitPreTransferComplete`,  
`kSDIF_TransferStopAbort` = SDIF\_CMD\_STOP\_ABORT\_CMD\_MASK,  
`kSDIF_SendInitialization`,  
`kSDIF_CmdUpdateClockRegisterOnly`,  
`kSDIF_CmdtoReadCEATADevice` = SDIF\_CMD\_READ\_CEATA\_DEVICE\_MASK,  
`kSDIF_CmdExpectCCS` = SDIF\_CMD\_CCS\_EXPECTED\_MASK,  
`kSDIF_BootModeEnable` = SDIF\_CMD\_ENABLE\_BOOT\_MASK,  
`kSDIF_BootModeExpectAck` = SDIF\_CMD\_EXPECT\_BOOT\_ACK\_MASK,  
`kSDIF_BootModeDisable` = SDIF\_CMD\_DISABLE\_BOOT\_MASK,  
`kSDIF_BootModeAlternate` = SDIF\_CMD\_BOOT\_MODE\_MASK,  
`kSDIF_CmdVoltageSwitch` = SDIF\_CMD\_VOLT\_SWITCH\_MASK,  
`kSDIF_CmdDataUseHoldReg` = SDIF\_CMD\_USE\_HOLD\_REG\_MASK }
- *\_sdif\_command\_flags define the command flags*
- enum {
   
    `kCARD_CommandTypeNormal` = 0U,  
`kCARD_CommandTypeSuspend` = 1U,  
`kCARD_CommandTypeResume` = 2U,  
`kCARD_CommandTypeAbort` = 3U }
   
*\_sdif\_command\_type The command type*
- enum {
   
    `kCARD_ResponseNone` = 0U,  
`kCARD_ResponseR1` = 1U,  
`kCARD_ResponseR1b` = 2U,  
`kCARD_ResponseR2` = 3U,  
`kCARD_ResponseR3` = 4U,  
`kCARD_ResponseR4` = 5U,  
`kCARD_ResponseR5` = 6U,  
`kCARD_ResponseR5b` = 7U,  
`kCARD_ResponseR6` = 8U,  
`kCARD_ResponseR7` = 9U }
   
*\_sdif\_response\_type The command response type.*
- enum {

```

kSDIF_CardDetect = SDIF_INTMASK_CDET_MASK,
kSDIF_ResponseError = SDIF_INTMASK_RE_MASK,
kSDIF_CommandDone = SDIF_INTMASK_CDONE_MASK,
kSDIF_DataTransferOver = SDIF_INTMASK_DTO_MASK,
kSDIF_WriteFIFORequest = SDIF_INTMASK_TXDR_MASK,
kSDIF_ReadFIFORequest = SDIF_INTMASK_RXDR_MASK,
kSDIF_ResponseCRCError = SDIF_INTMASK_RCRC_MASK,
kSDIF_DataCRCError = SDIF_INTMASK_DCRC_MASK,
kSDIF_ResponseTimeout = SDIF_INTMASK_RTO_MASK,
kSDIF_DataReadTimeout = SDIF_INTMASK_DRTO_MASK,
kSDIF_DataStarvationByHostTimeout = SDIF_INTMASK_HTO_MASK,
kSDIF_FIFOError = SDIF_INTMASK_FRUN_MASK,
kSDIF_HardwareLockError = SDIF_INTMASK_HLE_MASK,
kSDIF_DataStartBitError = SDIF_INTMASK_SBE_MASK,
kSDIF_AutoCmdDone = SDIF_INTMASK_ACD_MASK,
kSDIF_DataEndBitError = SDIF_INTMASK_EBE_MASK,
kSDIF_SDIOInterrupt = SDIF_INTMASK_SDIO_INT_MASK_MASK,
kSDIF_CommandTransferStatus,
kSDIF_DataTransferStatus ,
kSDIF_AllInterruptStatus = 0x1FFFFU }

    _sdif_interrupt_mask define the interrupt mask flags
• enum {
    kSDIF_DMATransFinishOneDescriptor = SDIF_IDSTS_TI_MASK,
    kSDIF_DMARecvFinishOneDescriptor = SDIF_IDSTS_RI_MASK,
    kSDIF_DMAFatalBusError = SDIF_IDSTS_FBE_MASK,
    kSDIF_DMADescriptorUnavailable = SDIF_IDSTS_DU_MASK,
    kSDIF_DMACardErrorSummary = SDIF_IDSTS_CES_MASK,
    kSDIF_NormalInterruptSummary = SDIF_IDSTS_NIS_MASK,
    kSDIF_AbnormalInterruptSummary = SDIF_IDSTS_AIS_MASK }

    _sdif_dma_status define the internal DMA status flags
• enum {
    kSDIF_DisableCompleteInterrupt = 0x2U,
    kSDIF_DMADescriptorDataBufferEnd = 0x4U,
    kSDIF_DMADescriptorDataBufferStart = 0x8U,
    kSDIF_DMASecondAddrChained = 0x10U,
    kSDIF_DMADescriptorEnd = 0x20U,
    kSDIF_DMADescriptorOwnByDMA = (int)0x80000000 }

    _sdif_dma_descriptor_flag define the internal DMA descriptor flag
• enum sdif_dma_mode_t
    define the internal DMA mode

```

## Functions

- void **SDIF\_Init** (SDIF\_Type \*base, **sdif\_config\_t** \*config)
   
*SDIF module initialization function.*
- void **SDIF\_Deinit** (SDIF\_Type \*base)

- **`SDIF module deinit function.`**
- `bool SDIF_SendCardActive (SDIF_Type *base, uint32_t timeout)`  
*SDIF send initialize 80 clocks for SD card after initial.*
- `static void SDIF_EnableCardClock (SDIF_Type *base, bool enable)`  
*SDIF module enable/disable card clock.*
- `static void SDIF_EnableLowPowerMode (SDIF_Type *base, bool enable)`  
*SDIF module enable/disable module disable the card clock to enter low power mode when card is idle,for SDIF cards, if interrupts must be detected, clock should not be stopped.*
- `static void SDIF_EnableCardPower (SDIF_Type *base, bool enable)`  
*enable/disable the card power.*
- `void SDIF_SetCardBusWidth (SDIF_Type *base, sdif_bus_width_t type)`  
*set card data bus width*
- `static uint32_t SDIF_DetectCardInsert (SDIF_Type *base, bool data3)`  
*SDIF module detect card insert status function.*
- `uint32_t SDIF_SetCardClock (SDIF_Type *base, uint32_t srcClock_Hz, uint32_t target_HZ)`  
*Sets the card bus clock frequency.*
- `bool SDIF_Reset (SDIF_Type *base, uint32_t mask, uint32_t timeout)`  
*reset the different block of the interface.*
- `static uint32_t SDIF_GetCardWriteProtect (SDIF_Type *base)`  
*get the card write protect status*
- `static void SDIF AssertHardwareReset (SDIF_Type *base)`  
*toggle state on hardware reset PIN This is used which card has a reset PIN typically.*
- `status_t SDIF_SendCommand (SDIF_Type *base, sdif_command_t *cmd, uint32_t timeout)`  
*send command to the card*
- `static void SDIF_EnableGlobalInterrupt (SDIF_Type *base, bool enable)`  
*SDIF enable/disable global interrupt.*
- `static void SDIF_EnableInterrupt (SDIF_Type *base, uint32_t mask)`  
*SDIF enable interrupt.*
- `static void SDIF_DisableInterrupt (SDIF_Type *base, uint32_t mask)`  
*SDIF disable interrupt.*
- `static uint32_t SDIF_GetInterruptStatus (SDIF_Type *base)`  
*SDIF get interrupt status.*
- `static uint32_t SDIF_GetEnabledInterruptStatus (SDIF_Type *base)`  
*SDIF get enabled interrupt status.*
- `static void SDIF_ClearInterruptStatus (SDIF_Type *base, uint32_t mask)`  
*SDIF clear interrupt status.*
- `void SDIF_TransferCreateHandle (SDIF_Type *base, sdif_handle_t *handle, sdif_transfer_callback_t *callback, void *userData)`  
*Creates the SDIF handle.*
- `static void SDIF_EnableDmaInterrupt (SDIF_Type *base, uint32_t mask)`  
*SDIF enable DMA interrupt.*
- `static void SDIF_DisableDmaInterrupt (SDIF_Type *base, uint32_t mask)`  
*SDIF disable DMA interrupt.*
- `static uint32_t SDIF_GetInternalDMAStatus (SDIF_Type *base)`  
*SDIF get internal DMA status.*
- `static uint32_t SDIF_GetEnabledDMAInterruptStatus (SDIF_Type *base)`  
*SDIF get enabled internal DMA interrupt status.*
- `static void SDIF_ClearInternalDMAStatus (SDIF_Type *base, uint32_t mask)`  
*SDIF clear internal DMA status.*
- `status_t SDIF_InternalDMAConfig (SDIF_Type *base, sdif_dma_config_t *config, const uint32_t *data, uint32_t dataSize)`

- static void **SDIF\_EnableInternalDMA** (SDIF\_Type \*base, bool enable)  
*SDIF internal DMA enable.*
- static void **SDIF\_SendReadWait** (SDIF\_Type \*base)  
*SDIF send read wait to SDIF card function.*
- bool **SDIF\_AbortReadData** (SDIF\_Type \*base, uint32\_t timeout)  
*SDIF abort the read data when SDIF card is in suspend state Once assert this bit,data state machine will be reset which is waiting for the next blocking data,used in SDIO card suspend sequence,should call after suspend cmd send.*
- static void **SDIF\_EnableCEATAInterrupt** (SDIF\_Type \*base, bool enable)  
*SDIF enable/disable CE-ATA card interrupt this bit should set together with the card register.*
- status\_t **SDIF\_TransferNonBlocking** (SDIF\_Type \*base, sdif\_handle\_t \*handle, sdif\_dma\_config\_t \*dmaConfig, sdif\_transfer\_t \*transfer)  
*SDIF transfer function data/cmd in a non-blocking way this API should be use in interrupt mode, when use this API user must call SDIF\_TransferCreateHandle first, all status check through interrupt.*
- status\_t **SDIF\_TransferBlocking** (SDIF\_Type \*base, sdif\_dma\_config\_t \*dmaConfig, sdif\_transfer\_t \*transfer)  
*SDIF transfer function data/cmd in a blocking way.*
- status\_t **SDIF\_ReleaseDMADescriptor** (SDIF\_Type \*base, sdif\_dma\_config\_t \*dmaConfig)  
*SDIF release the DMA descriptor to DMA engine this function should be called when DMA descriptor unavailable status occurs.*
- void **SDIF\_GetCapability** (SDIF\_Type \*base, sdif\_capability\_t \*capability)  
*SDIF return the controller capability.*
- static uint32\_t **SDIF\_GetControllerStatus** (SDIF\_Type \*base)  
*SDIF return the controller status.*
- static void **SDIF\_SendCCSD** (SDIF\_Type \*base, bool withAutoStop)  
*SDIF send command complete signal disable to CE-ATA card.*
- void **SDIF\_ConfigClockDelay** (uint32\_t target\_HZ, uint32\_t divider)  
*SDIF config the clock delay This function is used to config the cclk\_in delay to sample and driver the data ,should meet the min setup time and hold time, and user need to config this parameter according to your board setting.*

## Driver version

- #define **FSL\_SDIF\_DRIVER\_VERSION** (MAKE\_VERSION(2U, 1U, 0U))  
*Driver version 2.0.15.*

## 37.3 Data Structure Documentation

### 37.3.1 struct sdif\_dma\_descriptor\_t

#### Data Fields

- uint32\_t **dmaDesAttribute**  
*internal DMA attribute control and status*
- uint32\_t **dmaDataBufferSize**  
*internal DMA transfer buffer size control*
- const uint32\_t \* **dmaDataBufferAddr0**  
*internal DMA buffer 0 addr ,the buffer size must be 32bit aligned*

- const uint32\_t \* **dmaDataBufferAddr1**  
*internal DMA buffer 1 addr ,the buffer size must be 32bit aligned*

### 37.3.2 struct sdif\_dma\_config\_t

#### Data Fields

- bool **enableFixBurstLen**  
*fix burst len enable/disable flag,When set, the AHB will use only SINGLE, INCR4, INCR8 or INCR16 during start of normal burst transfers.*
- **sdif\_dma\_mode\_t mode**  
*define the DMA mode*
- uint8\_t **dmaDesSkipLen**  
*define the descriptor skip length ,the length between two descriptor this field is special for dual DMA mode*
- uint32\_t \* **dmaDesBufferStartAddr**  
*internal DMA descriptor start address*
- uint32\_t **dmaDesBufferLen**  
*internal DMA buffer descriptor buffer len ,user need to pay attention to the dma descriptor buffer length if it is bigger enough for your transfer*

#### Field Documentation

##### (1) bool **sdif\_dma\_config\_t::enableFixBurstLen**

When reset, the AHB will use SINGLE and INCR burst transfer operations

### 37.3.3 struct sdif\_data\_t

#### Data Fields

- bool **streamTransfer**  
*indicate this is a stream data transfer command*
- bool **enableAutoCommand12**  
*indicate if auto stop will send when data transfer over*
- bool **enableIgnoreError**  
*indicate if enable ignore error when transfer data*
- size\_t **blockSize**  
*Block size, take care when configure this parameter.*
- uint32\_t **blockCount**  
*Block count.*
- uint32\_t \* **rxData**  
*data buffer to receive*
- const uint32\_t \* **txData**  
*data buffer to transfer*

### 37.3.4 struct sdif\_command\_t

Define card command-related attribute.

#### Data Fields

- `uint32_t index`  
*Command index.*
- `uint32_t argument`  
*Command argument.*
- `uint32_t response [4U]`  
*Response for this command.*
- `uint32_t type`  
*define the command type*
- `uint32_t responseType`  
*Command response type.*
- `uint32_t flags`  
*Cmd flags.*
- `uint32_t responseErrorFlags`  
*response error flags, need to check the flags when receive the cmd response*

### 37.3.5 struct sdif\_transfer\_t

#### Data Fields

- `sdif_data_t * data`  
*Data to transfer.*
- `sdif_command_t * command`  
*Command to send.*

### 37.3.6 struct sdif\_config\_t

#### Data Fields

- `uint8_t responseTimeout`  
*command response timeout value*
- `uint32_t cardDetDebounce_Clock`  
*define the debounce clock count which will used in card detect logic,typical value is 5-25ms*
- `uint32_t dataTimeout`  
*data timeout value*

### 37.3.7 struct sdif\_capability\_t

Defines a structure to get the capability information of SDIF.

#### Data Fields

- `uint32_t sdVersion`  
*support SD card/sdio version*
- `uint32_t mmcVersion`  
*support emmc card version*
- `uint32_t maxBlockLength`  
*Maximum block length unitized as byte.*
- `uint32_t maxBlockCount`  
*Maximum byte count can be transferred.*
- `uint32_t flags`  
*Capability flags to indicate the support information.*

### 37.3.8 struct sdif\_transfer\_callback\_t

#### Data Fields

- `void(* cardInserted )(SDIF_Type *base, void *userData)`  
*card insert call back*
- `void(* cardRemoved )(SDIF_Type *base, void *userData)`  
*card remove call back*
- `void(* SDIOInterrupt )(SDIF_Type *base, void *userData)`  
*SDIO card interrupt occurs.*
- `void(* DMADesUnavailable )(SDIF_Type *base, void *userData)`  
*DMA descriptor unavailable.*
- `void(* CommandReload )(SDIF_Type *base, void *userData)`  
*command buffer full, need re-load*
- `void(* TransferComplete )(SDIF_Type *base, void *handle, status_t status, void *userData)`  
*Transfer complete callback.*

### 37.3.9 struct sdif\_handle\_t

Defines the structure to save the sdif state information and callback function. The detail interrupt status when send command or transfer data can be obtained from interruptFlags field by using mask defined in sdif\_interrupt\_flag\_t;

Note

All the fields except interruptFlags and transferredWords must be allocated by the user.

## Data Fields

- `sdif_data_t` \*volatile `data`  
*Data to transfer.*
- `sdif_command_t` \*volatile `command`  
*Command to send.*
- volatile `uint32_t` `transferredWords`  
*Words transferred by polling way.*
- `sdif_transfer_callback_t` `callback`  
*Callback function.*
- void \* `userData`  
*Parameter for transfer complete callback.*

### 37.3.10 struct `sdif_host_t`

## Data Fields

- `SDIF_Type` \* `base`  
*sdif peripheral base address*
- `uint32_t` `sourceClock_Hz`  
*sdif source clock frequency united in Hz*
- `sdif_config_t` `config`  
*sdif configuration*
- `sdif_transfer_function_t` `transfer`  
*sdif transfer function*
- `sdif_capability_t` `capability`  
*sdif capability information*

## 37.4 Macro Definition Documentation

### 37.4.1 #define `FSL_SDIF_DRIVER_VERSION` (`MAKE_VERSION(2U, 1U, 0U)`)

### 37.4.2 #define `SDIF_CLOCK_RANGE_NEED_DELAY` (`50000000U`)

Such as: response error/CRC error and so on

clock range value which need to add delay to avoid timing issue

### 37.4.3 #define `SDIF_HIGHSPEED_SAMPLE_DELAY` (`12U`)

$12 \times 250\text{ps} = 3\text{ns}$

### 37.4.4 #define `SDIF_HIGHSPEED_DRV_DELAY` (`31U`)

$31 \times 250\text{ps} = 7.75\text{ns}$

### 37.4.5 #define SDIF\_DEFAULT\_MODE\_SAMPLE\_DELAY (12U)

$12 * 250\text{ps} = 3\text{ns}$

## 37.5 Typedef Documentation

### 37.5.1 `typedef status_t(* sdif_transfer_function_t)(SDIF_Type *base, sdif_transfer_t *content)`

## 37.6 Enumeration Type Documentation

### 37.6.1 anonymous enum

Enumerator

*kStatus\_SDIF\_DescriptorBufferLenError* Set DMA descriptor failed.  
*kStatus\_SDIF\_InvalidArgument* invalid argument status  
*kStatus\_SDIF\_SyncCmdTimeout* sync command to CIU timeout status  
*kStatus\_SDIF\_SendCmdFail* send command to card fail  
*kStatus\_SDIF\_SendCmdErrorBufferFull* send command to card fail, due to command buffer full  
 user need to resend this command  
*kStatus\_SDIF\_DMATransferFailWithFBE* DMA transfer data fail with fatal bus error , to do with  
 this error :issue a hard reset/controller reset.  
*kStatus\_SDIF\_DMATransferDescriptorUnavailable* DMA descriptor unavailable.  
*kStatus\_SDIF\_DataTransferFail* transfer data fail  
*kStatus\_SDIF\_ResponseError* response error  
*kStatus\_SDIF\_DMAAddrNotAlign* DMA address not align.  
*kStatus\_SDIF\_BusyTransferring* SDIF transfer busy status.  
*kStatus\_SDIF\_DataTransferSuccess* transfer data success  
*kStatus\_SDIF\_SendCmdSuccess* transfer command success

### 37.6.2 anonymous enum

Enumerator

*kSDIF\_SupportHighSpeedFlag* Support high-speed.  
*kSDIF\_SupportDmaFlag* Support DMA.  
*kSDIF\_SupportSuspendResumeFlag* Support suspend/resume.  
*kSDIF\_SupportV330Flag* Support voltage 3.3V.  
*kSDIF\_Support4BitFlag* Support 4 bit mode.  
*kSDIF\_Support8BitFlag* Support 8 bit mode.

### 37.6.3 anonymous enum

Enumerator

***kSDIF\_ResetController*** reset controller,will reset: BIU/CIU interface CIU and state machine,ABORT\_READ\_DATA,SEND\_IRQ\_RESPONSE and READ\_WAIT bits of control register,START\_CMD bit of the command register  
***kSDIF\_ResetFIFO*** reset data FIFO  
***kSDIF\_ResetDMAInterface*** reset DMA interface  
***kSDIF\_ResetAll*** reset all

### 37.6.4 enum sdif\_bus\_width\_t

Enumerator

***kSDIF\_Bus1BitWidth*** 1bit bus width, 1bit mode and 4bit mode share one register bit  
***kSDIF\_Bus4BitWidth*** 4bit mode mask  
***kSDIF\_Bus8BitWidth*** support 8 bit mode

### 37.6.5 anonymous enum

Enumerator

***kSDIF\_CmdResponseExpect*** command request response  
***kSDIF\_CmdResponseLengthLong*** command response length long  
***kSDIF\_CmdCheckResponseCRC*** request check command response CRC  
***kSDIF\_DataExpect*** request data transfer,either read/write  
***kSDIF\_DataWriteToCard*** data transfer direction  
***kSDIF\_DataStreamTransfer*** data transfer mode :stream/block transfer command  
***kSDIF\_DataTransferAutoStop*** data transfer with auto stop at the end of  
***kSDIF\_WaitPreTransferComplete*** wait pre transfer complete before sending this cmd  
***kSDIF\_TransferStopAbort*** when host issue stop or abort cmd to stop data transfer ,this bit should set so that cmd/data state-machines of CIU can return to idle correctly  
***kSDIF\_SendInitialization*** send initialization 80 clocks for SD card after power on  
***kSDIF\_CmdUpdateClockRegisterOnly*** send cmd update the CIU clock register only  
***kSDIF\_CmdtoReadCEATADevice*** host is perform read access to CE-ATA device  
***kSDIF\_CmdExpectCCS*** command expect command completion signal signal  
***kSDIF\_BootModeEnable*** this bit should only be set for mandatory boot mode  
***kSDIF\_BootModeExpectAck*** boot mode expect ack  
***kSDIF\_BootModeDisable*** when software set this bit along with START\_CMD, CIU terminates the boot operation  
***kSDIF\_BootModeAlternate*** select boot mode ,alternate or mandatory  
***kSDIF\_CmdVoltageSwitch*** this bit set for CMD11 only  
***kSDIF\_CmdDataUseHoldReg*** cmd and data send to card through the HOLD register

### 37.6.6 anonymous enum

Enumerator

*kCARD\_CommandTypeNormal* Normal command.  
*kCARD\_CommandTypeSuspend* Suspend command.  
*kCARD\_CommandTypeResume* Resume command.  
*kCARD\_CommandTypeAbort* Abort command.

### 37.6.7 anonymous enum

Define the command response type from card to host controller.

Enumerator

*kCARD\_ResponseNone* Response type: none.  
*kCARD\_ResponseR1* Response type: R1.  
*kCARD\_ResponseR1b* Response type: R1b.  
*kCARD\_ResponseR2* Response type: R2.  
*kCARD\_ResponseR3* Response type: R3.  
*kCARD\_ResponseR4* Response type: R4.  
*kCARD\_ResponseR5* Response type: R5.  
*kCARD\_ResponseR5b* Response type: R5b.  
*kCARD\_ResponseR6* Response type: R6.  
*kCARD\_ResponseR7* Response type: R7.

### 37.6.8 anonymous enum

Enumerator

*kSDIF\_CardDetect* mask for card detect  
*kSDIF\_ResponseError* command response error  
*kSDIF\_CommandDone* command transfer over  
*kSDIF\_DataTransferOver* data transfer over flag  
*kSDIF\_WriteFIFORequest* write FIFO request  
*kSDIF\_ReadFIFORequest* read FIFO request  
*kSDIF\_ResponseCRCError* response CRC error  
*kSDIF\_DataCRCError* data CRC error  
*kSDIF\_ResponseTimeout* response timeout  
*kSDIF\_DataReadTimeout* read data timeout  
*kSDIF\_DataStarvationByHostTimeout* data starvation by host time out  
*kSDIF\_FIFOError* indicate the FIFO under run or overrun error  
*kSDIF\_HardwareLockError* hardware lock write error  
*kSDIF\_DataStartBitError* start bit error  
*kSDIF\_AutoCmdDone* indicate the auto command done

*kSDIF\_DataEndBitError* end bit error  
*kSDIF\_SDIOInterrupt* interrupt from the SDIO card  
*kSDIF\_CommandTransferStatus* command transfer status collection  
*kSDIF\_DataTransferStatus* data transfer status collection  
*kSDIF\_AllInterruptStatus* all interrupt mask

### 37.6.9 anonymous enum

Enumerator

*kSDIF\_DMATransFinishOneDescriptor* DMA transfer finished for one DMA descriptor.  
*kSDIF\_DMARcvFinishOneDescriptor* DMA receive finished for one DMA descriptor.  
*kSDIF\_DMAFatalBusError* DMA fatal bus error.  
*kSDIF\_DMADescriptorUnavailable* DMA descriptor unavailable.  
*kSDIF\_DMACardErrorSummary* card error summary  
*kSDIF\_NormalInterruptSummary* normal interrupt summary  
*kSDIF\_AbnormalInterruptSummary* abnormal interrupt summary

### 37.6.10 anonymous enum

**Deprecated** Do not use this enum anymore, please use SDIF\_DMA\_DESCRIPTOR\_XXX\_FLAG instead.

Enumerator

*kSDIF\_DisableCompleteInterrupt* disable the complete interrupt flag for the ends in the buffer pointed to by this descriptor  
*kSDIF\_DMADescriptorDataBufferEnd* indicate this descriptor contain the last data buffer of data  
*kSDIF\_DMADescriptorDataBufferStart* indicate this descriptor contain the first data buffer of data,if first buffer size is 0,next descriptor contain the begin of the data  
*kSDIF\_DMASEcondAddrChained* indicate that the second addr in the descriptor is the next descriptor addr not the data buffer  
*kSDIF\_DMADescriptorEnd* indicate that the descriptor list reached its final descriptor  
*kSDIF\_DMADescriptorOwnByDMA* indicate the descriptor is own by SD/MMC DMA

## 37.7 Function Documentation

### 37.7.1 void SDIF\_Init ( *SDIF\_Type* \* *base*, *sdif\_config\_t* \* *config* )

Configures the SDIF according to the user configuration.

Parameters

<i>base</i>	SDIF peripheral base address.
<i>config</i>	SDIF configuration information.

### 37.7.2 void SDIF\_Deinit ( SDIF\_Type \* *base* )

user should call this function follow with IP reset

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

### 37.7.3 bool SDIF\_SendCardActive ( SDIF\_Type \* *base*, uint32\_t *timeout* )

Parameters

<i>base</i>	SDIF peripheral base address.
<i>timeout</i>	timeout value

### 37.7.4 static void SDIF\_EnableCardClock ( SDIF\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	SDIF peripheral base address.
<i>enable</i>	enable/disable flag

### 37.7.5 static void SDIF\_EnableLowPowerMode ( SDIF\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	SDIF peripheral base address.
<i>enable</i>	enable/disable flag

### 37.7.6 static void SDIF\_EnableCardPower ( **SDIF\_Type** \* *base*, **bool** *enable* ) [inline], [static]

once turn power on, software should wait for regulator/switch ramp-up time before trying to initialize card.

Parameters

<i>base</i>	SDIF peripheral base address.
<i>enable</i>	enable/disable flag.

### 37.7.7 void SDIF\_SetCardBusWidth ( **SDIF\_Type** \* *base*, **sdif\_bus\_width\_t** *type* )

Parameters

<i>base</i>	SDIF peripheral base address.
<i>type</i>	bus width type

### 37.7.8 static uint32\_t SDIF\_DetectCardInsert ( **SDIF\_Type** \* *base*, **bool** *data3* ) [inline], [static]

Parameters

<i>base</i>	SDIF peripheral base address.
<i>data3</i>	indicate use data3 as card insert detect pin

Return values

<i>I</i>	card is inserted 0 card is removed
----------	------------------------------------

### 37.7.9 uint32\_t SDIF\_SetCardClock ( **SDIF\_Type** \* *base*, **uint32\_t** *srcClock\_Hz*, **uint32\_t** *target\_HZ* )

Parameters

<i>base</i>	SDIF peripheral base address.
<i>srcClock_Hz</i>	SDIF source clock frequency united in Hz.
<i>target_HZ</i>	card bus clock frequency united in Hz.

Returns

The nearest frequency of busClock\_Hz configured to SD bus.

### 37.7.10 **bool SDIF\_Reset ( SDIF\_Type \* *base*, uint32\_t *mask*, uint32\_t *timeout* )**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>mask</i>	indicate which block to reset.
<i>timeout</i>	timeout value, set to wait the bit self clear

Returns

reset result.

### 37.7.11 **static uint32\_t SDIF\_GetCardWriteProtect ( SDIF\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

### 37.7.12 **static void SDIF\_AssertHardwareReset ( SDIF\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

### 37.7.13 **status\_t SDIF\_SendCommand ( SDIF\_Type \* *base*, sdif\_command\_t \* *cmd*, uint32\_t *timeout* )**

This api include polling the status of the bit START\_COMMAND, if 0 used as timeout value, then this function will return directly without polling the START\_CMD status.

Parameters

<i>base</i>	SDIF peripheral base address.
<i>cmd</i>	configuration collection
<i>timeout</i>	the timeout value of polling START_CMD auto clear status.

Returns

command execute status

### 37.7.14 **static void SDIF\_EnableGlobalInterrupt ( SDIF\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>enable</i>	enable/disable flag

### 37.7.15 **static void SDIF\_EnableInterrupt ( SDIF\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>mask</i>	mask

**37.7.16 static void SDIF\_DisableInterrupt ( SDIF\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>mask</i>	mask

**37.7.17 static uint32\_t SDIF\_GetInterruptStatus ( SDIF\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

**37.7.18 static uint32\_t SDIF\_GetEnabledInterruptStatus ( SDIF\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

**37.7.19 static void SDIF\_ClearInterruptStatus ( SDIF\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

<i>mask</i>	mask to clear
-------------	---------------

### 37.7.20 void SDIF\_TransferCreateHandle ( **SDIF\_Type** \* *base*, **sdif\_handle\_t** \* *handle*, **sdif\_transfer\_callback\_t** \* *callback*, **void** \* *userData* )

register call back function for interrupt and enable the interrupt

Parameters

<i>base</i>	SDIF peripheral base address.
<i>handle</i>	SDIF handle pointer.
<i>callback</i>	Structure pointer to contain all callback functions.
<i>userData</i>	Callback function parameter.

### 37.7.21 static void SDIF\_EnableDmaInterrupt ( **SDIF\_Type** \* *base*, **uint32\_t** *mask* ) [**inline**], [**static**]

Parameters

<i>base</i>	SDIF peripheral base address.
<i>mask</i>	mask to set

### 37.7.22 static void SDIF\_DisableDmaInterrupt ( **SDIF\_Type** \* *base*, **uint32\_t** *mask* ) [**inline**], [**static**]

Parameters

<i>base</i>	SDIF peripheral base address.
<i>mask</i>	mask to clear

### 37.7.23 static **uint32\_t** SDIF\_GetInternalDMAStatus ( **SDIF\_Type** \* *base* ) [**inline**], [**static**]

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

Returns

the internal DMA status register

### 37.7.24 static uint32\_t SDIF\_GetEnabledDMAInterruptStatus ( **SDIF\_Type** \* *base* ) [inline], [static]

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

Returns

the internal DMA status register

### 37.7.25 static void SDIF\_ClearInternalDMAStatus ( **SDIF\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

Parameters

<i>base</i>	SDIF peripheral base address.
<i>mask</i>	mask to clear

### 37.7.26 **status\_t** SDIF\_InternalDMAConfig ( **SDIF\_Type** \* *base*, **sdif\_dma\_config\_t** \* *config*, **const uint32\_t** \* *data*, **uint32\_t** *dataSize* )

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

<i>config</i>	DMA configuration collection
<i>data</i>	buffer pointer
<i>dataSize</i>	buffer size

**37.7.27 static void SDIF\_EnableInternalDMA ( SDIF\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>enable</i>	internal DMA enable or disable flag.

**37.7.28 static void SDIF\_SendReadWait ( SDIF\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

**37.7.29 bool SDIF\_AbortReadData ( SDIF\_Type \* *base*, uint32\_t *timeout* )**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>timeout</i>	timeout value to wait this bit self clear which indicate the data machine reset to idle

**37.7.30 static void SDIF\_EnableCEATAInterrupt ( SDIF\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>enable</i>	enable/disable flag

### 37.7.31 status\_t SDIF\_TransferNonBlocking ( **SDIF\_Type** \* *base*, **sdif\_handle\_t** \* *handle*, **sdif\_dma\_config\_t** \* *dmaConfig*, **sdif\_transfer\_t** \* *transfer* )

Parameters

<i>base</i>	SDIF peripheral base address.
<i>handle</i>	handle
<i>dmaConfig</i>	config structure This parameter can be config as: 1. NULL In this condition, polling transfer mode is selected 2. available DMA config In this condition, DMA transfer mode is selected
<i>transfer</i>	transfer configuration collection

### 37.7.32 status\_t SDIF\_TransferBlocking ( **SDIF\_Type** \* *base*, **sdif\_dma\_config\_t** \* *dmaConfig*, **sdif\_transfer\_t** \* *transfer* )

Parameters

<i>base</i>	SDIF peripheral base address.
<i>dmaConfig</i>	config structure 1. NULL In this condition, polling transfer mode is selected 2. available DMA config In this condition, DMA transfer mode is selected
<i>transfer</i>	transfer configuration collection

### 37.7.33 status\_t SDIF\_ReleaseDMADescriptor ( **SDIF\_Type** \* *base*, **sdif\_dma\_config\_t** \* *dmaConfig* )

Parameters

<i>base</i>	SDIF peripheral base address.
<i>dmaConfig</i>	DMA config pointer

**37.7.34 void SDIF\_GetCapability ( SDIF\_Type \* *base*, sdif\_capability\_t \* *capability* )**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>capability</i>	capability pointer

**37.7.35 static uint32\_t SDIF\_GetControllerStatus ( SDIF\_Type \* *base* )  
[inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
-------------	-------------------------------

**37.7.36 static void SDIF\_SendCCSD ( SDIF\_Type \* *base*, bool *withAutoStop* )  
[inline], [static]**

Parameters

<i>base</i>	SDIF peripheral base address.
<i>withAutoStop</i>	auto stop flag

**37.7.37 void SDIF\_ConfigClockDelay ( uint32\_t *target\_HZ*, uint32\_t *divider* )**

Parameters

## Function Documentation

<i>target_HZ</i>	freq work mode
<i>divider</i>	not used in this function anymore, use DELAY value instead of phase directly.

# Chapter 38

## SYSCTL: I2S bridging and signal sharing Configuration

### 38.1 Overview

The MCUXpresso SDK provides a peripheral driver for the SYSCTL module of MCUXpresso SDK devices. For further details, see the corresponding chapter.

### Files

- file [fsl\\_sysctl.h](#)
- file [fsl\\_sysctl.h](#)

### Enumerations

- enum [\\_sysctl\\_share\\_set\\_index](#) {  
    kSYSCTL\_ShareSet0 = 0,  
    kSYSCTL\_ShareSet1 = 1 }  
*SYSCTL share set.*
- enum [sysctl\\_fcctrlsel\\_signal\\_t](#) {  
    kSYSCTL\_FlexcommSignalSCK = SYSCTL\_FCCTRLSEL\_SCKINSEL\_SHIFT,  
    kSYSCTL\_FlexcommSignalWS = SYSCTL\_FCCTRLSEL\_WSINSEL\_SHIFT,  
    kSYSCTL\_FlexcommSignalDataIn = SYSCTL\_FCCTRLSEL\_DATAINSEL\_SHIFT,  
    kSYSCTL\_FlexcommSignalDataOut = SYSCTL\_FCCTRLSEL\_DATAOUTSEL\_SHIFT }  
*SYSCTL flexcomm signal.*
- enum [\\_sysctl\\_share\\_src](#) {  
    kSYSCTL\_Flexcomm0 = 0,  
    kSYSCTL\_Flexcomm1 = 1,  
    kSYSCTL\_Flexcomm2 = 2,  
    kSYSCTL\_Flexcomm4 = 4,  
    kSYSCTL\_Flexcomm5 = 5,  
    kSYSCTL\_Flexcomm6 = 6,  
    kSYSCTL\_Flexcomm7 = 7 }  
*SYSCTL flexcomm index.*
- enum [\\_sysctl\\_dataout\\_mask](#) {  
    kSYSCTL\_Flexcomm0DataOut = SYSCTL\_SHAREDCTRLSET\_FC0DATAOUTEN\_MASK,  
    kSYSCTL\_Flexcomm1DataOut = SYSCTL\_SHAREDCTRLSET\_FC1DATAOUTEN\_MASK,  
    kSYSCTL\_Flexcomm2DataOut = SYSCTL\_SHAREDCTRLSET\_FC2DATAOUTEN\_MASK,  
    kSYSCTL\_Flexcomm4DataOut = SYSCTL\_SHAREDCTRLSET\_FC4DATAOUTEN\_MASK,  
    kSYSCTL\_Flexcomm5DataOut = SYSCTL\_SHAREDCTRLSET\_FC5DATAOUTEN\_MASK,  
    kSYSCTL\_Flexcomm6DataOut = SYSCTL\_SHAREDCTRLSET\_FC6DATAOUTEN\_MASK,  
    kSYSCTL\_Flexcomm7DataOut = SYSCTL\_SHAREDCTRLSET\_FC7DATAOUTEN\_MASK }  
*SYSCTL shared data out mask.*

- enum `sysctl_sharedctrlset_signal_t` {
   
  `kSYSCTL_SharedCtrlSignalSCK` = `SYSCTL_SHAREDCTRLSET_SHAREDSCKSEL_SHIFT`,
   
  `kSYSCTL_SharedCtrlSignalWS` = `SYSCTL_SHAREDCTRLSET_SHAREDWSSEL_SHIFT`,
   
  `kSYSCTL_SharedCtrlSignalDataIn` = `SYSCTL_SHAREDCTRLSET_SHAREDDATASEL_SHIFT`,
   
  `kSYSCTL_SharedCtrlSignalDataOut` = `SYSCTL_SHAREDCTRLSET_FC0DATAOUTEN_SHIFT`
  
}
- SYSCTL flexcomm signal.*

## Driver version

- #define `FSL_SYSCTL_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 5)`)
   
*Group sysctl driver version for SDK.*

## Initialization and deinitialization

- void `SYSCTL_Init` (`SYSCTL_Type *base`)
   
*SYSCTL initial.*
- void `SYSCTL_Deinit` (`SYSCTL_Type *base`)
   
*SYSCTL\_deinit.*

## SYSCTL share signal configure

- void `SYSCTL_SetFlexcommShareSet` (`SYSCTL_Type *base, uint32_t flexCommIndex, uint32_t sckSet, uint32_t wsSet, uint32_t dataInSet, uint32_t dataOutSet`)
   
*SYSCTL share set configure for flexcomm.*
- void `SYSCTL_SetShareSet` (`SYSCTL_Type *base, uint32_t flexCommIndex, sysctl_fcctrlsel_signal_t signal, uint32_t set`)
   
*SYSCTL share set configure for separate signal.*
- void `SYSCTL_SetShareSetSrc` (`SYSCTL_Type *base, uint32_t setIndex, uint32_t sckShareSrc, uint32_t wsShareSrc, uint32_t dataInShareSrc, uint32_t dataOutShareSrc`)
   
*SYSCTL share set source configure.*
- void `SYSCTL_SetShareSignalSrc` (`SYSCTL_Type *base, uint32_t setIndex, sysctl_sharedctrlset_signal_t signal, uint32_t shareSrc`)
   
*SYSCTL sck source configure.*

## 38.2 Macro Definition Documentation

### 38.2.1 #define FSL\_SYSCTL\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 5))

Version 2.0.5.

## 38.3 Enumeration Type Documentation

### 38.3.1 enum \_sysctl\_share\_set\_index

Enumerator

`kSYSCTL_ShareSet0` share set 0

*kSYSCTL\_ShareSet1* share set 1

### 38.3.2 enum sysctl\_fcctrlsel\_signal\_t

Enumerator

*kSYSCTL\_FlexcommSignalSCK* SCK signal.  
*kSYSCTL\_FlexcommSignalWS* WS signal.  
*kSYSCTL\_FlexcommSignalDataIn* Data in signal.  
*kSYSCTL\_FlexcommSignalDataOut* Data out signal.

### 38.3.3 enum \_sysctl\_share\_src

Enumerator

*kSYSCTL\_Flexcomm0* share set 0  
*kSYSCTL\_Flexcomm1* share set 1  
*kSYSCTL\_Flexcomm2* share set 2  
*kSYSCTL\_Flexcomm4* share set 4  
*kSYSCTL\_Flexcomm5* share set 5  
*kSYSCTL\_Flexcomm6* share set 6  
*kSYSCTL\_Flexcomm7* share set 7

### 38.3.4 enum \_sysctl\_dataout\_mask

Enumerator

*kSYSCTL\_Flexcomm0DataOut* share set 0  
*kSYSCTL\_Flexcomm1DataOut* share set 1  
*kSYSCTL\_Flexcomm2DataOut* share set 2  
*kSYSCTL\_Flexcomm4DataOut* share set 4  
*kSYSCTL\_Flexcomm5DataOut* share set 5  
*kSYSCTL\_Flexcomm6DataOut* share set 6  
*kSYSCTL\_Flexcomm7DataOut* share set 7

### 38.3.5 enum sysctl\_sharedctrlset\_signal\_t

Enumerator

*kSYSCTL\_SharedCtrlSignalSCK* SCK signal.

*kSYSCTL\_SharedCtrlSignalWS* WS signal.  
*kSYSCTL\_SharedCtrlSignalDataIn* Data in signal.  
*kSYSCTL\_SharedCtrlSignalDataOut* Data out signal.

## 38.4 Function Documentation

### 38.4.1 void SYSCTL\_Init ( **SYSCTL\_Type** \* *base* )

Parameters

<i>base</i>	Base address of the SYSCTL peripheral.
-------------	--

### 38.4.2 void SYSCTL\_Deinit ( **SYSCTL\_Type** \* *base* )

Parameters

<i>base</i>	Base address of the SYSCTL peripheral.
-------------	--

### 38.4.3 void SYSCTL\_SetFlexcommShareSet ( **SYSCTL\_Type** \* *base*, **uint32\_t** *flexCommIndex*, **uint32\_t** *sckSet*, **uint32\_t** *wsSet*, **uint32\_t** *dataInSet*, **uint32\_t** *dataOutSet* )

Parameters

<i>base</i>	Base address of the SYSCTL peripheral.
<i>flexCommIndex</i>	index of flexcomm, reference _sysctl_share_src
<i>sckSet</i>	share set for sck,reference _sysctl_share_set_index
<i>wsSet</i>	share set for ws, reference _sysctl_share_set_index
<i>dataInSet</i>	share set for data in, reference _sysctl_share_set_index
<i>dataOutSet</i>	share set for data out, reference _sysctl_dataout_mask

### 38.4.4 void SYSCTL\_SetShareSet ( **SYSCTL\_Type** \* *base*, **uint32\_t** *flexCommIndex*, **sysctl\_fcctrlsel\_signal\_t** *signal*, **uint32\_t** *set* )

Parameters

<i>base</i>	Base address of the SYSCTL peripheral
<i>flexCommIndex</i>	index of flexcomm,reference _sysctl_share_src
<i>signal</i>	FCCTRLSEL signal shift
<i>set</i>	share set for sck, reference _sysctl_share_set_index

**38.4.5 void SYSCTL\_SetShareSetSrc ( SYSCTL\_Type \* *base*, uint32\_t *setIndex*, uint32\_t *sckShareSrc*, uint32\_t *wsShareSrc*, uint32\_t *dataInShareSrc*, uint32\_t *dataOutShareSrc* )**

Parameters

<i>base</i>	Base address of the SYSCTL peripheral
<i>setIndex</i>	index of share set, reference _sysctl_share_set_index
<i>sckShareSrc</i>	sck source for this share set,reference _sysctl_share_src
<i>wsShareSrc</i>	ws source for this share set,reference _sysctl_share_src
<i>dataInShareSrc</i>	data in source for this share set,reference _sysctl_share_src
<i>dataOutShare-Src</i>	data out source for this share set,reference _sysctl_dataout_mask

**38.4.6 void SYSCTL\_SetShareSignalSrc ( SYSCTL\_Type \* *base*, uint32\_t *setIndex*, sysctl\_sharedctrlset\_signal\_t *signal*, uint32\_t *shareSrc* )**

Parameters

<i>base</i>	Base address of the SYSCTL peripheral
<i>setIndex</i>	index of share set, reference _sysctl_share_set_index
<i>signal</i>	FCCTRLSEL signal shift
<i>shareSrc</i>	sck source fro this share set,reference _sysctl_share_src

# Chapter 39

## UTICK: MictoTick Timer Driver

### 39.1 Overview

The MCUXpresso SDK provides a peripheral driver for the UTICK module of MCUXpresso SDK devices. UTICK driver is created to help user to operate the UTICK module. The UTICK timer can be used as a low power timer. The APIs can be used to enable the UTICK module, initialize it and set the time. UTICK can be used as a wake up source from low power mode.

### 39.2 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/utick

### Files

- file [fsl\\_utick.h](#)

### TypeDefs

- `typedef void(* utick_callback_t )(void)`  
*UTICK callback function.*

### Enumerations

- `enum utick_mode_t {`  
    `kUTICK_Onetime = 0x0U,`  
    `kUTICK_Repeat = 0x1U }`  
*UTICK timer operational mode.*

### Driver version

- `#define FSL_UTICK_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))`  
*UTICK driver version 2.0.5.*

### Initialization and deinitialization

- `void UTICK_Init (UTICK_Type *base)`  
*Initializes an UTICK by turning its bus clock on.*
- `void UTICK_Deinit (UTICK_Type *base)`  
*Deinitializes a UTICK instance.*
- `uint32_t UTICK_GetStatusFlags (UTICK_Type *base)`  
*Get Status Flags.*
- `void UTICK_ClearStatusFlags (UTICK_Type *base)`  
*Clear Status Interrupt Flags.*

- void [UTICK\\_SetTick](#) (UTICK\_Type \*base, utick\_mode\_t mode, uint32\_t count, utick\_callback\_t cb)  
*Starts UTICK.*
- void [UTICK\\_HandleIRQ](#) (UTICK\_Type \*base, utick\_callback\_t cb)  
*UTICK Interrupt Service Handler.*

### 39.3 Macro Definition Documentation

#### 39.3.1 #define FSL\_UTICK\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 5))

### 39.4 Typedef Documentation

#### 39.4.1 typedef void(\* utick\_callback\_t)(void)

### 39.5 Enumeration Type Documentation

#### 39.5.1 enum utick\_mode\_t

Enumerator

*kUTICK\_Onetime* Trigger once.

*kUTICK\_Repeat* Trigger repeatedly.

### 39.6 Function Documentation

#### 39.6.1 void UTICK\_Init ( UTICK\_Type \* *base* )

#### 39.6.2 void UTICK\_Deinit ( UTICK\_Type \* *base* )

This function shuts down Utick bus clock

Parameters

<i>base</i>	UTICK peripheral base address.
-------------	--------------------------------

#### 39.6.3 uint32\_t UTICK\_GetStatusFlags ( UTICK\_Type \* *base* )

This returns the status flag

Parameters

<i>base</i>	UTICK peripheral base address.
-------------	--------------------------------

Returns

status register value

### 39.6.4 void UTICK\_ClearStatusFlags ( UTICK\_Type \* *base* )

This clears intr status flag

Parameters

<i>base</i>	UTICK peripheral base address.
-------------	--------------------------------

Returns

none

### 39.6.5 void UTICK\_SetTick ( UTICK\_Type \* *base*, utick\_mode\_t *mode*, uint32\_t *count*, utick\_callback\_t *cb* )

This function starts a repeat/onetime countdown with an optional callback

Parameters

<i>base</i>	UTICK peripheral base address.
<i>mode</i>	UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
<i>count</i>	UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
<i>cb</i>	UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

Returns

none

### 39.6.6 void UTICK\_HandleIRQ ( UTICK\_Type \* *base*, utick\_callback\_t *cb* )

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in [UTICK\\_SetTick\(\)](#)). if no user callback is scheduled, the interrupt will simply be cleared.

## Parameters

<i>base</i>	UTICK peripheral base address.
<i>cb</i>	callback scheduled for this instance of UTICK

## Returns

none

# Chapter 40

## WWDT: Windowed Watchdog Timer Driver

### 40.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

### 40.2 Function groups

#### 40.2.1 Initialization and deinitialization

The function `WWDT_Init()` initializes the watchdog timer with specified configurations. The configurations include timeout value and whether to enable watchdog after init. The function `WWDT_GetDefaultConfig()` gets the default configurations.

The function `WWDT_Deinit()` disables the watchdog and the module clock.

#### 40.2.2 Status

Provides functions to get and clear the WWDT status.

#### 40.2.3 Interrupt

Provides functions to enable/disable WWDT interrupts and get current enabled interrupts.

#### 40.2.4 Watch dog Refresh

The function `WWDT_Refresh()` feeds the WWDT.

### 40.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/wwdt

### Files

- file `fsl_wwdt.h`

### Data Structures

- struct `wwdt_config_t`  
*Describes WWDT configuration structure.* [More...](#)

## Enumerations

- enum `_wwdt_status_flags_t` {
   
`kWWDT_TimeoutFlag` = `WWDT_MOD_WDTOF_MASK`,
   
`kWWDT_WarningFlag` = `WWDT_MOD_WDINT_MASK` }
   
*WWDT status flags.*

## Driver version

- `#define FSL_WWDT_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))`
  
*Defines WWDT driver version.*

## Refresh sequence

- `#define WWDT_FIRST_WORD_OF_REFRESH (0xAAU)`
  
*First word of refresh sequence.*
- `#define WWDT_SECOND_WORD_OF_REFRESH (0x55U)`
  
*Second word of refresh sequence.*

## WWDT Initialization and De-initialization

- `void WWDT_GetDefaultConfig (wwdt_config_t *config)`
  
*Initializes WWDT configure structure.*
- `void WWDT_Init (WWDT_Type *base, const wwdt_config_t *config)`
  
*Initializes the WWDT.*
- `void WWDT_Deinit (WWDT_Type *base)`
  
*Shuts down the WWDT.*

## WWDT Functional Operation

- `static void WWDT_Enable (WWDT_Type *base)`
  
*Enables the WWDT module.*
- `static void WWDT_Disable (WWDT_Type *base)`
  
*Disables the WWDT module.*
- `static uint32_t WWDT_GetStatusFlags (WWDT_Type *base)`
  
*Gets all WWDT status flags.*
- `void WWDT_ClearStatusFlags (WWDT_Type *base, uint32_t mask)`
  
*Clear WWDT flag.*
- `static void WWDT_SetWarningValue (WWDT_Type *base, uint32_t warningValue)`
  
*Set the WWDT warning value.*
- `static void WWDT_SetTimeoutValue (WWDT_Type *base, uint32_t timeoutCount)`
  
*Set the WWDT timeout value.*
- `static void WWDT_SetWindowValue (WWDT_Type *base, uint32_t windowValue)`
  
*Sets the WWDT window value.*
- `void WWDT_Refresh (WWDT_Type *base)`
  
*Refreshes the WWDT timer.*

## 40.4 Data Structure Documentation

#### 40.4.1 struct wwdt\_config\_t

##### Data Fields

- bool `enableWwdt`  
*Enables or disables WWDT.*
- bool `enableWatchdogReset`  
*true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset*
- bool `enableWatchdogProtect`  
*true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time*
- bool `enableLockOscillator`  
*true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator*
- uint32\_t `windowValue`  
*Window value, set this to 0xFFFF if windowing is not in effect.*
- uint32\_t `timeoutValue`  
*Timeout value.*
- uint32\_t `warningValue`  
*Watchdog time counter value that will generate a warning interrupt.*
- uint32\_t `clockFreq_Hz`  
*Watchdog clock source frequency.*

##### Field Documentation

###### (1) uint32\_t wwdt\_config\_t::warningValue

Set this to 0 for no warning

###### (2) uint32\_t wwdt\_config\_t::clockFreq\_Hz

#### 40.5 Macro Definition Documentation

##### 40.5.1 #define FSL\_WWDT\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 9))

#### 40.6 Enumeration Type Documentation

##### 40.6.1 enum \_wwdt\_status\_flags\_t

This structure contains the WWDT status flags for use in the WWDT functions.

Enumerator

*kWWDT\_TimeoutFlag* Time-out flag, set when the timer times out.

*kWWDT\_WarningFlag* Warning interrupt flag, set when timer is below the value WDWARNINT.

#### 40.7 Function Documentation

### 40.7.1 void WWDT\_GetDefaultConfig ( `wwdt_config_t * config` )

This function initializes the WWDT configure structure to default value. The default value are:

```
* config->enableWwdt = true;
* config->enableWatchdogReset = false;
* config->enableWatchdogProtect = false;
* config->enableLockOscillator = false;
* config->windowValue = 0xFFFFFFFU;
* config->timeoutValue = 0xFFFFFFFU;
* config->warningValue = 0;
*
```

Parameters

<i>config</i>	Pointer to WWDT config structure.
---------------	-----------------------------------

See Also

[wwdt\\_config\\_t](#)

### 40.7.2 void WWDT\_Init ( `WWDT_Type * base, const wwdt_config_t * config` )

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
* wwdt_config_t config;
* WWDT_GetDefaultConfig(&config);
* config.timeoutValue = 0x7ffU;
* WWDT_Init(wwdt_base,&config);
*
```

Parameters

<i>base</i>	WWDT peripheral base address
<i>config</i>	The configuration of WWDT

### 40.7.3 void WWDT\_Deinit ( `WWDT_Type * base` )

This function shuts down the WWDT.

Parameters

<i>base</i>	WWDT peripheral base address
-------------	------------------------------

#### 40.7.4 static void WWDT\_Enable( WWDT\_Type \* *base* ) [inline], [static]

This function write value into WWDT\_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

<i>base</i>	WWDT peripheral base address
-------------	------------------------------

#### 40.7.5 static void WWDT\_Disable( WWDT\_Type \* *base* ) [inline], [static]

**Deprecated** Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT\_MOD register to disable the WWDT.

Parameters

<i>base</i>	WWDT peripheral base address
-------------	------------------------------

#### 40.7.6 static uint32\_t WWDT\_GetStatusFlags( WWDT\_Type \* *base* ) [inline], [static]

This function gets all status flags.

Example for getting Timeout Flag:

```
*     uint32_t status;
*     status = WWDT_GetStatusFlags(wwdt_base) &
*             kWWDT_TimeoutFlag;
*
```

Parameters

<i>base</i>	WWDT peripheral base address
-------------	------------------------------

Returns

The status flags. This is the logical OR of members of the enumeration [\\_wwdt\\_status\\_flags\\_t](#)

#### 40.7.7 void WWDT\_ClearStatusFlags ( **WWDT\_Type** \* *base*, **uint32\_t** *mask* )

This function clears WWDT status flag.

Example for clearing warning flag:

```
*     WWDT_ClearStatusFlags (wwdt_base, kWWDT_WarningFlag);
*
```

Parameters

<i>base</i>	WWDT peripheral base address
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration <a href="#">_wwdt_status_flags_t</a>

#### 40.7.8 static void WWDT\_SetWarningValue ( **WWDT\_Type** \* *base*, **uint32\_t** *warningValue* ) [inline], [static]

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

<i>base</i>	WWDT peripheral base address
<i>warningValue</i>	WWDT warning value.

#### 40.7.9 static void WWDT\_SetTimeoutValue ( **WWDT\_Type** \* *base*, **uint32\_t** *timeoutCount* ) [inline], [static]

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC

register. Thus the minimum time-out interval is TWDCLK\*256\*4. If enableWatchdogProtect flag is true in `wwdt_config_t` config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

<i>base</i>	WWDT peripheral base address
<i>timeoutCount</i>	WWDT timeout value, count of WWDT clock tick.

#### 40.7.10 static void WWDT\_SetWindowValue ( WWDT\_Type \* *base*, uint32\_t *windowValue* ) [inline], [static]

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set *windowValue* to 0xFFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

<i>base</i>	WWDT peripheral base address
<i>windowValue</i>	WWDT window value.

#### 40.7.11 void WWDT\_Refresh ( WWDT\_Type \* *base* )

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

<i>base</i>	WWDT peripheral base address
-------------	------------------------------

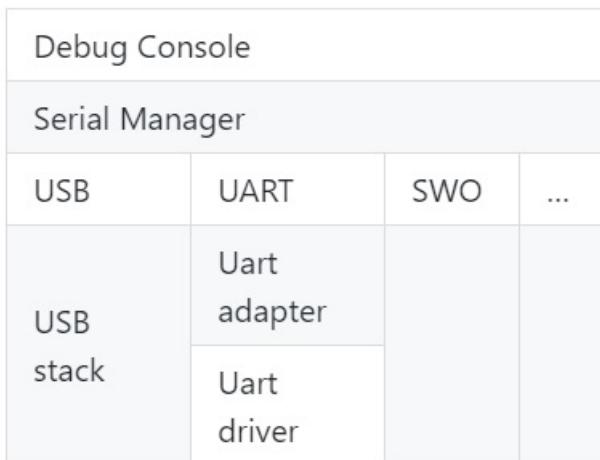
# Chapter 41

## Debug Console

### 41.1 Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data. The below picture shows the layout of debug console.



**Debug console overview**

### 41.2 Function groups

#### 41.2.1 Initialization

To initialize the debug console, call the [DbgConsole\\_Init\(\)](#) function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate,  
                         serial_port_type_t device, uint32_t clkSrcFreq);
```

Select the supported debug console hardware device type, such as

```
typedef enum _serial_port_type  
{  
    kSerialPort_Uart = 1U,  
    kSerialPort_UsbCdc,  
    kSerialPort_Swo,  
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. This example shows how to call the [DbgConsole\\_Init\(\)](#) given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUDRATE, BOARD_DEBUG_UART_TYPE,
                 BOARD_DEBUG_UART_CLK_FREQ);
```

#### 41.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	Description
.number	For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

length	Description
Do not support	

specifier	Description
d or i	Signed decimal integer
f	Decimal floating point
F	Decimal floating point capital letters
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer capital letters
o	Signed octal
b	Binary value
p	Pointer address
u	Unsigned decimal integer
c	Character
s	String of characters
n	Nothing printed

- Support a format specifier for SCANF following this prototype " %[\*][width][length]specifier", which is explained below

*	Description
An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument.	

width	Description
This specifies the maximum number of characters to be read in the current reading operation.	

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

```

version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */

```

### 41.2.3 SDK\_DEBUGCONSOLE and SDK\_DEBUGCONSOLE\_UART

There are two macros `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` added to configure `PRINTF` and low level output peripheral.

- The macro `SDK_DEBUGCONSOLE` is used for frontend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The function can be set by the macro `SDK_DEBUGCONSOLE`.
- The macro `SDK_DEBUGCONSOLE_UART` is used for backend. It is used to decide whether provide low level IO implementation to toolchain printf and scanf. For example, within MCUXpresso, if the macro `SDK_DEBUGCONSOLE_UART` is defined, `_sys_write` and `_sys_read` will be used when `_REDLIB_` is defined; `_write` and `_read` will be used in other cases. The macro does not specifically refer to the peripheral "UART". It refers to the external peripheral similar to UART, like as USB CDC, UART, SWO, etc. So if the macro `SDK_DEBUGCONSOLE_UART` is not defined when tool-chain printf is calling, the semihosting will be used.

The following matrix show the effects of `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` on `PRINTF` and `printf`. The green mark is the default setting of the debug console.

<code>SDK_DEBUGCONSOLE</code>	<code>SDK_DEBUGCONSOLE_UART</code>	<code>PRINTF</code>	<code>printf</code>
<code>DEBUGCONSOLE_- REDIRECT_TO_SDK</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_- REDIRECT_TO_SDK</code>	undefined	Low level peripheral*	semihost
<code>DEBUGCONSOLE_- REDIRECT_TO_TO- OLCHAIN</code>	defined	Low level peripheral*	Low level peripheral
<code>DEBUGCONSOLE_- REDIRECT_TO_TO- OLCHAIN</code>	undefined	semihost	semihost
<code>DEBUGCONSOLE_- DISABLE</code>	defined	No output	Low level peripheral
<code>DEBUGCONSOLE_- DISABLE</code>	undefined	No output	semihost

- \* the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

### 41.3 Typical use case

#### Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

#### Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

#### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

#### Print out failure messages using MCUXpresso SDK \_\_assert\_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
          , line, func);
    for (;;) {}
}
```

#### Note:

To use 'printf' and 'scanf' for GNUC Base, add file '**fsl\_sbrk.c**' in path: ..\{package}\devices\{subset}\utilities\fsl\_sbrk.c to your project.

## Modules

- [SWO](#)
- [Semihosting](#)

## Macros

- `#define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U`  
*Definition select redirect toolchain printf, scanf to uart or not.*
- `#define DEBUGCONSOLE_REDIRECT_TO_SDK 1U`  
*Select SDK version printf, scanf.*
- `#define DEBUGCONSOLE_DISABLE 2U`  
*Disable debugconsole function.*
- `#define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK`  
*Definition to select sdk or toolchain printf, scanf.*
- `#define PRINTF DbgConsole_Printf`  
*Definition to select redirect toolchain printf, scanf to uart or not.*

## Variables

- `serial_handle_t g_serialHandle`  
*serial manager handle*

## Initialization

- `status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)`  
*Initializes the peripheral used for debug messages.*
- `status_t DbgConsole_Deinit (void)`  
*De-initializes the peripheral used for debug messages.*
- `status_t DbgConsole_EnterLowpower (void)`  
*Prepares to enter low power consumption.*
- `status_t DbgConsole_ExitLowpower (void)`  
*Restores from low power consumption.*
- `int DbgConsole_Printf (const char *fmt_s,...)`  
*Writes formatted output to the standard output stream.*
- `int DbgConsole_Vprintf (const char *fmt_s, va_list formatStringArg)`  
*Writes formatted output to the standard output stream.*
- `int DbgConsole_Putchar (int ch)`  
*Writes a character to stdout.*
- `int DbgConsole_Scanf (char *fmt_s,...)`  
*Reads formatted data from the standard input stream.*
- `int DbgConsole_Getchar (void)`  
*Reads a character from standard input.*
- `int DbgConsole_BlockingPrintf (const char *fmt_s,...)`  
*Writes formatted output to the standard output stream with the blocking mode.*
- `int DbgConsole_BlockingVprintf (const char *fmt_s, va_list formatStringArg)`  
*Writes formatted output to the standard output stream with the blocking mode.*
- `status_t DbgConsole_Flush (void)`  
*Debug console flush.*

## 41.4 Macro Definition Documentation

### 41.4.1 #define DEBUGCONSOLE\_REDIRECT\_TO\_TOOLCHAIN 0U

Select toolchain printf and scanf.

### 41.4.2 #define DEBUGCONSOLE\_REDIRECT\_TO\_SDK 1U

### 41.4.3 #define DEBUGCONSOLE\_DISABLE 2U

### 41.4.4 #define SDK\_DEBUGCONSOLE DEBUGCONSOLE\_REDIRECT\_TO\_SDK

The macro only support to be redefined in project setting.

### 41.4.5 #define PRINTF DbgConsole\_Printf

if SDK\_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK\_DEBUGCONSOLE defined to 1,it represents select SDK version printf, scanf. if SDK\_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

## 41.5 Function Documentation

### 41.5.1 status\_t DbgConsole\_Init ( uint8\_t instance, uint32\_t baudRate, serial\_port\_type\_t device, uint32\_t clkSrcFreq )

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

<i>instance</i>	The instance of the module.If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UART adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART peripheral is LPUART1. If the uart_adapter.c is added to the current project, the UART peripheral is UART1.
-----------------	---

<i>baudRate</i>	The desired baud rate in bits per second.
<i>device</i>	Low level device type for the debug console, can be one of the following. <ul style="list-style-type: none"><li>• kSerialPort_Uart,</li><li>• kSerialPort_UsbCdc</li></ul>
<i>clkSrcFreq</i>	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

<i>kStatus_Success</i>	Execution successfully
------------------------	------------------------

#### 41.5.2 status\_t DbgConsole\_Deinit( void )

Call this function to disable debug log messages to be output via the specified peripheral initialized by the serial manager module.

Returns

Indicates whether de-initialization was successful or not.

#### 41.5.3 status\_t DbgConsole\_EnterLowpower( void )

This function is used to prepare to enter low power consumption.

Returns

Indicates whether de-initialization was successful or not.

#### 41.5.4 status\_t DbgConsole\_ExitLowpower( void )

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

#### 41.5.5 int DbgConsole\_Printf( const char \* *fmt\_s*, ... )

Call this function to write a formatted output to the standard output stream.

Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

Returns

Returns the number of characters printed or a negative value if an error occurs.

#### 41.5.6 int DbgConsole\_Vprintf ( const char \* *fmt\_s*, va\_list *formatStringArg* )

Call this function to write a formatted output to the standard output stream.

Parameters

<i>fmt_s</i>	Format control string.
<i>formatString-Arg</i>	Format arguments.

Returns

Returns the number of characters printed or a negative value if an error occurs.

#### 41.5.7 int DbgConsole\_Putchar ( int *ch* )

Call this function to write a character to stdout.

Parameters

<i>ch</i>	Character to be written.
-----------	--------------------------

Returns

Returns the character written.

#### 41.5.8 int DbgConsole\_Scanf ( char \* *fmt\_s*, ... )

Call this function to read formatted data from the standard input stream.

## Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole\_TryGetchar to get the input char.

## Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

## Returns

Returns the number of fields successfully converted and assigned.

**41.5.9 int DbgConsole\_Getchar ( void )**

Call this function to read a character from standard input.

## Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole\_TryGetchar to get the input char.

## Returns

Returns the character read.

**41.5.10 int DbgConsole\_BlockingPrintf ( const char \* *fmt\_s*, ... )**

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set or not. The function could be used in system ISR mode with DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set.

Parameters

<i>fmt_s</i>	Format control string.
--------------	------------------------

Returns

Returns the number of characters printed or a negative value if an error occurs.

#### 41.5.11 int DbgConsole\_BlockingVprintf ( const char \* *fmt\_s*, va\_list *formatStringArg* )

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set or not. The function could be used in system ISR mode with DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set.

Parameters

<i>fmt_s</i>	Format control string.
<i>formatString-Arg</i>	Format arguments.

Returns

Returns the number of characters printed or a negative value if an error occurs.

#### 41.5.12 status\_t DbgConsole\_Flush ( void )

Call this function to wait the tx buffer empty. If interrupt transfer is using, make sure the global IRQ is enable before call this function This function should be called when 1, before enter power down mode 2, log is required to print to terminal immediately

Returns

Indicates whether wait idle was successful or not.

## 41.6 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

### 41.6.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the `SDK_DEBUGCONSOLE` is `DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN`.

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

#### Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

#### Step 3: Starting semihosting

1. Choose "Semihosting\_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting. Please Make sure the `SDK_DEBUGCONSOLE_UART` is not defined in project settings.
4. Start the project by choosing Project>Download and Debug.
5. Choose View>Terminal I/O to display the output from the I/O operations.

### 41.6.2 Guide Semihosting for Keil µVision

**NOTE:** Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

### 41.6.3 Guide Semihosting for MCUXpresso IDE

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK\_DEBUGCONSOLE=0, if set SDK\_DEBUGCONSOLE=1, the log will be redirect to the UART.

#### Step 2: Building the project

1. Compile and link the project.

#### Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

### 41.6.4 Guide Semihosting for ARMGCC

#### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telet.
  - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

#### Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__stack_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__stack_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__heap_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__heap_size__=0x2000")
```

## Step 2: Building the project

1. Change "CMakeLists.txt":

```
Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=nano.specs")"
to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=rdimon.specs")"
```

### Replace paragraph

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
```

### To

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
```

### Remove

```
target_link_libraries(semihosting_ARMGCC.elf debug nosys)
```

2. Run "build\_debug.bat" to build project

### Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

2. After the setting, press "enter". The PuTTY window now shows the printf() output.

## 41.7 SWO

Serial wire output is a mechanism for ARM targets to output signal from core through a single pin. Some IDEs also support SWO, such IAR and KEIL, both input and output are supported, see below for details.

### 41.7.1 Guide SWO for SDK

**NOTE:** After the setting both "printf" and "PRINTF" are available for debugging, JlinkSWOViewer can be used to capture the output log.

#### Step 1: Setting up the environment

1. Define SERIAL\_PORT\_TYPE\_SWO in your project settings.
2. Prepare code, the port and baudrate can be decided by application, clkSrcFreq should be mcu core clock frequency:

```
DbgConsole_Init(instance, baudRate, kSerialPort_Swo, clkSrcFreq);
```

3. Use PRINTF or printf to print some thing in application.

#### Step 2: Building the project

#### Step 3: Download and run project

##### 41.7.1.1 Guide SWO for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

#### Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK, and click the Activate button to enable trace data collection.
6. There are three cases for this SDK\_DEBUGCONSOLE\_UART whether or not defined. a: if use uppercase PRINTF to output log, The SDK\_DEBUGCONSOLE\_UART defined or not defined will not effect debug function. b: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to zero, then debug function ok. c: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to one, then debug function ok.

**NOTE:** Case a or c only apply at example which enable swo function, the SDK\_DEBUGCONSOLE\_UART definition in fsl\_debug\_console.h. For case a and c, Do and not do the above third step will be not affect function.

1. Start the project by choosing Project>Download and Debug.

### **Step 2: Building the project**

### **Step 3: Starting swo**

1. Download and debug application.
2. Choose View -> Terminal I/O to display the output from the I/O operations.
3. Run application.

## **41.7.2 Guide SWO for Keil µVision**

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

### **Step 1: Setting up the environment**

1. There are three cases for this SDK\_DEBUGCONSOLE\_UART whether or not defined. a: if use uppercase PRINTF to output log, the SDK\_DEBUGCONSOLE\_UART definition does not affect the functionality and skip the second step directly. b: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to zero, then start the second step. c: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to one, then skip the second step directly.

**NOTE:** Case a or c only apply at example which enable swo function, the SDK\_DEBUGCONSOLE\_UART definition in fsl\_debug\_console.h.

1. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
2. Open Project>Options for target or using Alt+F7 or click.
3. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
4. Select “Debug” tab and choose Port:SW, then select “Trace” tab, choose “Enable” and click OK, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

### **Step 3: Building the project**

1. Compile and link the project by choosing Project>Build Target or using F7.

### **Step 4: Run the project**

1. Choose “Debug” on menu bar or Ctrl F5.
2. In menu bar, choose “Serial Window” and click to “Debug (printf) Viewer”.
3. Run line by line to see result in Console Window.

### 41.7.3 Guide SWO for MCUXpresso IDE

**NOTE:** MCUX support SWO for LPC-Link2 debug probe only.

### 41.7.4 Guide SWO for ARMGCC

**NOTE:** ARMGCC has no library support SWO.

# Chapter 42

## Notification Framework

### 42.1 Overview

This section describes the programming interface of the Notifier driver.

### 42.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.  
The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *notify, void *data)
{
    status_t ret = kStatus_Success;

    ...
    ...

    return ret;
}
// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *
    userData)
```

```

{
    ...
    ...
    ...
}

...
...
...
...
...
...
// Main function.
int main(void)
{
    // Define a notifier handle.
    notifier_handle_t powerModeHandle;

    // Callback configuration.
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
        kNOTIFIER_CallbackBeforeAfter,
        (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    // Power mode configurations.
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    // Definition of a transition to and out the power modes.
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    // Create Notifier handle.
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
        APP_PowerModeSwitch, NULL);
    ...

    ...
    // Power mode switch.
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
        kNOTIFIER_PolicyAgreement);
}

```

## Data Structures

- struct [notifier\\_notification\\_block\\_t](#)  
*notification block passed to the registered callback function.* [More...](#)
- struct [notifier\\_callback\\_config\\_t](#)  
*Callback configuration structure.* [More...](#)
- struct [notifier\\_handle\\_t](#)  
*Notifier handle structure.* [More...](#)

## Typedefs

- [typedef void notifier\\_user\\_config\\_t](#)  
*Notifier user configuration type.*
- [typedef status\\_t\(\\* notifier\\_user\\_function\\_t \)\(notifier\\_user\\_config\\_t \\*targetConfig, void \\*userData\)](#)

- *Notifier user function prototype Use this function to execute specific operations in configuration switch.*  
**typedef status\_t(\* notifier\_callback\_t )(notifier\_notification\_block\_t \*notify, void \*data)**  
*Callback prototype.*

## Enumerations

- **enum \_notifier\_status {**  
**kStatus\_NOTIFIER\_ErrorNotificationBefore,**  
**kStatus\_NOTIFIER\_ErrorNotificationAfter }**  
*Notifier error codes.*
- **enum notifier\_policy\_t {**  
**kNOTIFIER\_PolicyAgreement,**  
**kNOTIFIER\_PolicyForcible }**  
*Notifier policies.*
- **enum notifier\_notification\_type\_t {**  
**kNOTIFIER\_NotifyRecover = 0x00U,**  
**kNOTIFIER\_NotifyBefore = 0x01U,**  
**kNOTIFIER\_NotifyAfter = 0x02U }**  
*Notification type.*
- **enum notifier\_callback\_type\_t {**  
**kNOTIFIER\_CallbackBefore = 0x01U,**  
**kNOTIFIER\_CallbackAfter = 0x02U,**  
**kNOTIFIER\_CallbackBeforeAfter = 0x03U }**  
*The callback type, which indicates kinds of notification the callback handles.*

## Functions

- **status\_t NOTIFIER\_CreateHandle (notifier\_handle\_t \*notifierHandle, notifier\_user\_config\_t \*\*configs, uint8\_t configsNumber, notifier\_callback\_config\_t \*callbacks, uint8\_t callbacksNumber, notifier\_user\_function\_t userFunction, void \*userData)**  
*Creates a Notifier handle.*
- **status\_t NOTIFIER\_SwitchConfig (notifier\_handle\_t \*notifierHandle, uint8\_t configIndex, notifier\_policy\_t policy)**  
*Switches the configuration according to a pre-defined structure.*
- **uint8\_t NOTIFIER\_GetErrorCallbackIndex (notifier\_handle\_t \*notifierHandle)**  
*This function returns the last failed notification callback.*

## 42.3 Data Structure Documentation

### 42.3.1 struct notifier\_notification\_block\_t

#### Data Fields

- **notifier\_user\_config\_t \* targetConfig**  
*Pointer to target configuration.*
- **notifier\_policy\_t policy**  
*Configure transition policy.*
- **notifier\_notification\_type\_t notifyType**

*Configure notification type.*

#### Field Documentation

- (1) **notifier\_user\_config\_t\* notifier\_notification\_block\_t::targetConfig**
- (2) **notifier\_policy\_t notifier\_notification\_block\_t::policy**
- (3) **notifier\_notification\_type\_t notifier\_notification\_block\_t::notifyType**

### 42.3.2 struct notifier\_callback\_config\_t

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. callback - pointer to the callback function callbackType - specifies when the callback is called callbackData - pointer to the data passed to the callback.

#### Data Fields

- **notifier\_callback\_t callback**  
*Pointer to the callback function.*
- **notifier\_callback\_type\_t callbackType**  
*Callback type.*
- **void \* callbackData**  
*Pointer to the data passed to the callback.*

#### Field Documentation

- (1) **notifier\_callback\_t notifier\_callback\_config\_t::callback**
- (2) **notifier\_callback\_type\_t notifier\_callback\_config\_t::callbackType**
- (3) **void\* notifier\_callback\_config\_t::callbackData**

### 42.3.3 struct notifier\_handle\_t

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. [NOTIFIER\\_CreateHandle\(\)](#) must be called to initialize this handle.

#### Data Fields

- **notifier\_user\_config\_t \*\* configsTable**  
*Pointer to configure table.*
- **uint8\_t configsNumber**  
*Number of configurations.*

- `notifier_callback_config_t * callbacksTable`  
*Pointer to callback table.*
- `uint8_t callbacksNumber`  
*Maximum number of callback configurations.*
- `uint8_t errorCallbackIndex`  
*Index of callback returns error.*
- `uint8_t currentConfigIndex`  
*Index of current configuration.*
- `notifier_user_function_t userFunction`  
*User function.*
- `void * userData`  
*User data passed to user function.*

## Field Documentation

- (1) `notifier_user_config_t** notifier_handle_t::configsTable`
- (2) `uint8_t notifier_handle_t::configsNumber`
- (3) `notifier_callback_config_t* notifier_handle_t::callbacksTable`
- (4) `uint8_t notifier_handle_t::callbacksNumber`
- (5) `uint8_t notifier_handle_t::errorCallbackIndex`
- (6) `uint8_t notifier_handle_t::currentConfigIndex`
- (7) `notifier_user_function_t notifier_handle_t::userFunction`
- (8) `void* notifier_handle_t::userData`

## 42.4 Typedef Documentation

### 42.4.1 `typedef void notifier_user_config_t`

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

### 42.4.2 `typedef status_t(* notifier_user_function_t)(notifier_user_config_t *targetConfig, void *userData)`

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, `NOTIFIER_SwitchConfig()` exits.

Parameters

<i>targetConfig</i>	target Configuration.
<i>userData</i>	Refers to other specific data passed to user function.

Returns

An error code or kStatus\_Success.

#### 42.4.3 **typedef status\_t(\* notifier\_callback\_t)(notifier\_notification\_block\_t \*notify, void \*data)**

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the [notifier\\_callback\\_config\\_t](#) callback configuration structure. Depending on callback type, function of this prototype is called (see [NOTIFIER\\_SwitchConfig\(\)](#)) before configuration switch, after it or in both use cases to notify about the switch progress (see [notifier\\_callback\\_type\\_t](#)). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see [notifier\\_notification\\_block\\_t](#)) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see [notifier\\_policy\\_t](#)), the callback may deny the execution of the user function by returning an error code different than kStatus\_Success (see [NOTIFIER\\_SwitchConfig\(\)](#)).

Parameters

<i>notify</i>	Notification block.
<i>data</i>	Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

Returns

An error code or kStatus\_Success.

### 42.5 Enumeration Type Documentation

#### 42.5.1 enum \_notifier\_status

Used as return value of Notifier functions.

Enumerator

***kStatus\_NOTIFIER\_ErrorNotificationBefore*** An error occurs during send "BEFORE" notification.

***kStatus\_NOTIFIER\_ErrorNotificationAfter*** An error occurs during send "AFTER" notification.

### 42.5.2 enum notifier\_policy\_t

Defines whether the user function execution is forced or not. For `kNOTIFIER_PolicyForcible`, the user function is executed regardless of the callback results, while `kNOTIFIER_PolicyAgreement` policy is used to exit `NOTIFIER_SwitchConfig()` when any of the callbacks returns error code. See also `NOTIFIER_SwitchConfig()` description.

Enumerator

***kNOTIFIER\_PolicyAgreement*** `NOTIFIER_SwitchConfig()` method is exited when any of the callbacks returns error code.

***kNOTIFIER\_PolicyForcible*** The user function is executed regardless of the results.

### 42.5.3 enum notifier\_notification\_type\_t

Used to notify registered callbacks

Enumerator

***kNOTIFIER\_NotifyRecover*** Notify IP to recover to previous work state.

***kNOTIFIER\_NotifyBefore*** Notify IP that configuration setting is going to change.

***kNOTIFIER\_NotifyAfter*** Notify IP that configuration setting has been changed.

### 42.5.4 enum notifier\_callback\_type\_t

Used in the callback configuration structure (`notifier_callback_config_t`) to specify when the registered callback is called during configuration switch initiated by the `NOTIFIER_SwitchConfig()`. Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect `NOTIFIER_SwitchConfig()` execution. See the `NOTIFIER_SwitchConfig()` and `notifier_policy_t` documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

***kNOTIFIER\_CallbackBefore*** Callback handles BEFORE notification.

***kNOTIFIER\_CallbackAfter*** Callback handles AFTER notification.

***kNOTIFIER\_CallbackBeforeAfter*** Callback handles BEFORE and AFTER notification.

## 42.6 Function Documentation

42.6.1 **status\_t NOTIFIER\_CreateHandle ( notifier\_handle\_t \* *notifierHandle*,  
notifier\_user\_config\_t \*\* *configs*, uint8\_t *configsNumber*, notifier\_callback-  
\_config\_t \* *callbacks*, uint8\_t *callbacksNumber*, notifier\_user\_function\_t  
*userFunction*, void \* *userData* )**

## Parameters

<i>notifierHandle</i>	A pointer to the notifier handle.
<i>configs</i>	A pointer to an array with references to all configurations which is handled by the Notifier.
<i>configsNumber</i>	Number of configurations. Size of the configuration array.
<i>callbacks</i>	A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value.
<i>callbacks-Number</i>	Number of registered callbacks. Size of the callbacks array.
<i>userFunction</i>	User function.
<i>userData</i>	User data passed to user function.

## Returns

An error Code or kStatus\_Success.

#### 42.6.2 **status\_t NOTIFIER\_SwitchConfig ( notifier\_handle\_t \* *notifierHandle*, uint8\_t *configIndex*, notifier\_policy\_t *policy* )**

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER\_PolicyForcible) or exited (kNOTIFIER\_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER\_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when NOTIFIER\_SwitchConfig() exits.

## Parameters

<i>notifierHandle</i>	pointer to notifier handle
<i>configIndex</i>	Index of the target configuration.
<i>policy</i>	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus\_Success.

#### 42.6.3 `uint8_t NOTIFIER_GetErrorCallbackIndex ( notifier_handle_t *notifierHandle )`

This function returns an index of the last callback that failed during the configuration switch while the last [NOTIFIER\\_SwitchConfig\(\)](#) was called. If the last [NOTIFIER\\_SwitchConfig\(\)](#) call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

<i>notifierHandle</i>	Pointer to the notifier handle
-----------------------	--------------------------------

Returns

Callback Index of the last failed callback or value equal to callbacks count.

# Chapter 43

## Shell

### 43.1 Overview

This section describes the programming interface of the Shell middleware.

Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

### 43.2 Function groups

#### 43.2.1 Initialization

To initialize the Shell middleware, call the `SHELL_Init()` function with these parameters. This function automatically enables the middleware.

```
shell_status_t SHELL_Init(shell_handle_t shellHandle,  
    serial_handle_t serialHandle, char *prompt);
```

Then, after the initialization was successful, call a command to control MCUs.

This example shows how to call the `SHELL_Init()` given the user configuration structure.

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
```

#### 43.2.2 Advanced Feature

- Support to get a character from standard input devices.

```
static shell_status_t SHELL_GetChar(shell_context_handle_t *shellContextHandle, uint8_t *ch);
```

Commands	Description
help	List all the registered commands.
exit	Exit program.

#### 43.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");  
SHELL_Task(s_shellHandle);
```

## Data Structures

- struct `shell_command_t`  
*User command data configuration structure. More...*

## Macros

- #define `SHELL_NON_BLOCKING_MODE` SERIAL\_MANAGER\_NON\_BLOCKING\_MODE  
*Whether use non-blocking mode.*
- #define `SHELL_AUTO_COMPLETE` (1U)  
*Macro to set on/off auto-complete feature.*
- #define `SHELL_BUFFER_SIZE` (64U)  
*Macro to set console buffer size.*
- #define `SHELL_MAX_ARGS` (8U)  
*Macro to set maximum arguments in command.*
- #define `SHELL_HISTORY_COUNT` (3U)  
*Macro to set maximum count of history commands.*
- #define `SHELL_IGNORE_PARAMETER_COUNT` (0xFF)  
*Macro to bypass arguments check.*
- #define `SHELL_HANDLE_SIZE`  
*The handle size of the shell module.*
- #define `SHELL_USE_COMMON_TASK` (0U)  
*Macro to determine whether use common task.*
- #define `SHELL_TASK_PRIORITY` (2U)  
*Macro to set shell task priority.*
- #define `SHELL_TASK_STACK_SIZE` (1000U)  
*Macro to set shell task stack size.*
- #define `SHELL_HANDLE_DEFINE`(name) uint32\_t name[((`SHELL_HANDLE_SIZE` + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]  
*Defines the shell handle.*
- #define `SHELL_COMMAND_DEFINE`(command, descriptor, callback, paramInt)  
*Defines the shell command structure.*
- #define `SHELL_COMMAND`(command) &g\_shellCommand##command  
*Gets the shell command pointer.*

## Typedefs

- typedef void \* `shell_handle_t`  
*The handle of the shell module.*
- typedef `shell_status_t`(\* `cmd_function_t` )(`shell_handle_t` shellHandle, int32\_t argc, char \*\*argv)  
*User command function prototype.*

## Enumerations

- enum `shell_status_t` {
   
`kStatus_SHELL_Success` = kStatus\_Success,
   
`kStatus_SHELL_Error` = MAKE\_STATUS(kStatusGroup\_SHELL, 1),
   
`kStatus_SHELL_OpenWriteHandleFailed` = MAKE\_STATUS(kStatusGroup\_SHELL, 2),
   
`kStatus_SHELL_OpenReadHandleFailed` = MAKE\_STATUS(kStatusGroup\_SHELL, 3),
   
`kStatus_SHELL_RetUsage` = MAKE\_STATUS(kStatusGroup\_SHELL, 4) }
   
*Shell status.*

## Shell functional operation

- `shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char *prompt)`  
*Initializes the shell module.*
- `shell_status_t SHELL_RegisterCommand (shell_handle_t shellHandle, shell_command_t *shellCommand)`  
*Registers the shell command.*
- `shell_status_t SHELL_UnregisterCommand (shell_command_t *shellCommand)`  
*Unregisters the shell command.*
- `shell_status_t SHELL_Write (shell_handle_t shellHandle, const char *buffer, uint32_t length)`  
*Sends data to the shell output stream.*
- `int SHELL_Printf (shell_handle_t shellHandle, const char *formatString,...)`  
*Writes formatted output to the shell output stream.*
- `shell_status_t SHELL_WriteSynchronization (shell_handle_t shellHandle, const char *buffer, uint32_t length)`  
*Sends data to the shell output stream with OS synchronization.*
- `int SHELL_PrintfSynchronization (shell_handle_t shellHandle, const char *formatString,...)`  
*Writes formatted output to the shell output stream with OS synchronization.*
- `void SHELL_ChangePrompt (shell_handle_t shellHandle, char *prompt)`  
*Change shell prompt.*
- `void SHELL_PrintPrompt (shell_handle_t shellHandle)`  
*Print shell prompt.*
- `void SHELL_Task (shell_handle_t shellHandle)`  
*The task function for Shell.*
- `static bool SHELL_checkRunningInIsr (void)`  
*Check if code is running in ISR.*

## 43.3 Data Structure Documentation

### 43.3.1 struct shell\_command\_t

#### Data Fields

- `const char * pcCommand`  
*The command that is executed.*
- `char * pcHelpString`  
*String that describes how to use the command.*
- `const cmd_function_t pFuncCallBack`  
*A pointer to the callback function that returns the output generated by the command.*
- `uint8_t cExpectedNumberOfParameters`  
*Commands expect a fixed number of parameters, which may be zero.*
- `list_element_t link`  
*link of the element*

#### Field Documentation

##### (1) `const char* shell_command_t::pcCommand`

For example "help". It must be all lower case.

**(2) `char* shell_command_t::pcHelpString`**

It should start with the command itself, and end with "\r\n". For example "help: Returns a list of all the commands\r\n".

**(3) `const cmd_function_t shell_command_t::pFuncCallBack`****(4) `uint8_t shell_command_t::cExpectedNumberOfParameters`**

## 43.4 Macro Definition Documentation

### 43.4.1 `#define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE`

### 43.4.2 `#define SHELL_AUTO_COMPLETE (1U)`

### 43.4.3 `#define SHELL_BUFFER_SIZE (64U)`

### 43.4.4 `#define SHELL_MAX_ARGS (8U)`

### 43.4.5 `#define SHELL_HISTORY_COUNT (3U)`

### 43.4.6 `#define SHELL_HANDLE_SIZE`

**Value:**

```
(160U + SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE +
    SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + \
    SERIAL_MANAGER_WRITE_HANDLE_SIZE)
```

It is the sum of the SHELL\_HISTORY\_COUNT \* SHELL\_BUFFER\_SIZE + SHELL\_BUFFER\_SIZE + SERIAL\_MANAGER\_READ\_HANDLE\_SIZE + SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE

### 43.4.7 `#define SHELL_USE_COMMON_TASK (0U)`

### 43.4.8 `#define SHELL_TASK_PRIORITY (2U)`

### 43.4.9 `#define SHELL_TASK_STACK_SIZE (1000U)`

#### 43.4.10 #define SHELL\_HANDLE\_DEFINE( *name* ) uint32\_t *name*[(**SHELL\_HANDLE\_SIZE** + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t)]

This macro is used to define a 4 byte aligned shell handle. Then use "(shell\_handle\_t)*name*" to get the shell handle.

The macro should be global and could be optional. You could also define shell handle by yourself.

This is an example,

```
* SHELL_HANDLE_DEFINE(shellHandle);
*
```

Parameters

<i>name</i>	The name string of the shell handle.
-------------	--------------------------------------

#### 43.4.11 #define SHELL\_COMMAND\_DEFINE( *command*, *descriptor*, *callback*, *paramCount* )

**Value:**

```
\shell_command_t g_shellCommand##command = {
    (#command), (descriptor), (callback), (paramCount), {0},      \
}
```

This macro is used to define the shell command structure [shell\\_command\\_t](#). And then uses the macro SHELL\_COMMAND to get the command structure pointer. The macro should not be used in any function.

This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

<i>command</i>	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".
----------------	--

<i>descriptor</i>	The description of the command is used for showing the command usage when "help" is typing.
<i>callback</i>	The callback of the command is used to handle the command line when the input command is matched.
<i>paramCount</i>	The max parameter count of the current command.

#### 43.4.12 #define SHELL\_COMMAND( *command* ) &g\_shellCommand##*command*

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL\_COMMAND\_DEFINE is used.

Parameters

<i>command</i>	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".
----------------	--

### 43.5 Typedef Documentation

#### 43.5.1 **typedef shell\_status\_t(\* cmd\_function\_t)(shell\_handle\_t shellHandle, int32\_t argc, char \*\*argv)**

### 43.6 Enumeration Type Documentation

#### 43.6.1 enum shell\_status\_t

Enumerator

*kStatus\_SHELL\_Success* Success.

*kStatus\_SHELL\_Error* Failed.

*kStatus\_SHELL\_OpenWriteHandleFailed* Open write handle failed.

*kStatus\_SHELL\_OpenReadHandleFailed* Open read handle failed.

*kStatus\_SHELL\_RetUsage* RetUsage for print cmd usage.

### 43.7 Function Documentation

#### 43.7.1 **shell\_status\_t SHELL\_Init ( shell\_handle\_t shellHandle, serial\_handle\_t serialHandle, char \* prompt )**

This function must be called before calling all other Shell functions. Call operation the Shell commands with user-defined settings. The example below shows how to set up the Shell and how to call the SHELL\_Init function by passing in these parameters. This is an example.

```
* static SHELL_HANDLE_DEFINE(s_shellHandle);
```

```
*   SHELL_Init((shell_handle_t)s_shellHandle, (
    serial_handle_t)s_serialHandle, "Test@SHELL>");  
*
```

## Parameters

<i>shellHandle</i>	Pointer to point to a memory space of size <b>SHELL_HANDLE_SIZE</b> allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <b>SHELL_HANDLE_DEFINE(shellHandle)</b> ; or <code>uint32_t shellHandle[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<i>serialHandle</i>	The serial manager module handle pointer.
<i>prompt</i>	The string prompt pointer of Shell. Only the global variable can be passed.

## Return values

<i>kStatus_SHELL_Success</i>	The shell initialization succeed.
<i>kStatus_SHELL_Error</i>	An error occurred when the shell is initialized.
<i>kStatus_SHELL_OpenWriteHandleFailed</i>	Open the write handle failed.
<i>kStatus_SHELL_OpenReadHandleFailed</i>	Open the read handle failed.

**43.7.2 shell\_status\_t SHELL\_RegisterCommand ( shell\_handle\_t *shellHandle*, shell\_command\_t \* *shellCommand* )**

This function is used to register the shell command by using the command configuration shell\_command\_config\_t. This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

## Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>shellCommand</i>	The command element.

## Return values

<i>kStatus_SHELL_Success</i>	Successfully register the command.
<i>kStatus_SHELL_Error</i>	An error occurred.

### 43.7.3 shell\_status\_t SHELL\_UnregisterCommand ( shell\_command\_t \* *shellCommand* )

This function is used to unregister the shell command.

Parameters

<i>shellCommand</i>	The command element.
---------------------	----------------------

Return values

<i>kStatus_SHELL_Success</i>	Successfully unregister the command.
------------------------------	--------------------------------------

### 43.7.4 shell\_status\_t SHELL\_Write ( shell\_handle\_t *shellHandle*, const char \* *buffer*, uint32\_t *length* )

This function is used to send data to the shell output stream.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>buffer</i>	Start address of the data to write.
<i>length</i>	Length of the data to write.

Return values

<i>kStatus_SHELL_Success</i>	Successfully send data.
<i>kStatus_SHELL_Error</i>	An error occurred.

### 43.7.5 int SHELL\_Printf ( shell\_handle\_t *shellHandle*, const char \* *formatString*, ... )

Call this function to write a formatted output to the shell output stream.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>formatString</i>	Format string.

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 43.7.6 **shell\_status\_t SHELL\_WriteSynchronization ( shell\_handle\_t *shellHandle*, const char \* *buffer*, uint32\_t *length* )**

This function is used to send data to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>buffer</i>	Start address of the data to write.
<i>length</i>	Length of the data to write.

Return values

<i>kStatus_SHELL_Success</i>	Successfully send data.
<i>kStatus_SHELL_Error</i>	An error occurred.

### 43.7.7 **int SHELL\_PrintfSynchronization ( shell\_handle\_t *shellHandle*, const char \* *formatString*, ... )**

Call this function to write a formatted output to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

<i>formatString</i>	Format string.
---------------------	----------------

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 43.7.8 void SHELL\_ChangePrompt ( shell\_handle\_t *shellHandle*, char \* *prompt* )

Call this function to change shell prompt.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>prompt</i>	The string which will be used for command prompt

Returns

NULL.

### 43.7.9 void SHELL\_PrintPrompt ( shell\_handle\_t *shellHandle* )

Call this function to print shell prompt.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

Returns

NULL.

### 43.7.10 void SHELL\_Task ( shell\_handle\_t *shellHandle* )

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------

#### **43.7.11 static bool SHELL\_checkRunningInIsr( void ) [inline], [static]**

This function is used to check if code running in ISR.

Return values

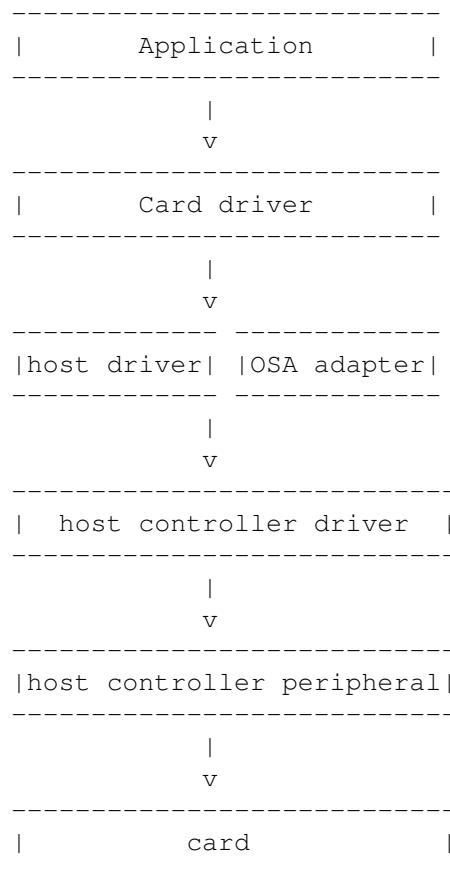
<i>TRUE</i>	if code runing in ISR.
-------------	------------------------

# Chapter 44

## Cards: Secure Digital Card/Embedded MultiMedia Card/SD-IO Card

### 44.1 Overview

The MCUXpresso SDK provides drivers to access the Secure Digital Card(up to v3.0), Embedded Multi-Media Card(up to v5.0) and sdio card(up to v3.0) based on the SDHC/USDHC/SDIF driver. Here is a simple block diagram about the drivers:



### Modules

- [MMC Card Driver](#)
- [SD Card Driver](#)
- [SDIO Card Driver](#)
- [SDMMC Common](#)
- [SDMMC HOST Driver](#)
- [SDMMC OSA](#)

## 44.2 SDIO Card Driver

### 44.2.1 Overview

The SDIO card driver provide card initialization/IO direct and extend command interface.

### 44.2.2 SDIO CARD Operation

#### error log support

Not supported yet.

#### User configurable

#### Board dependency

#### Mutual exclusive access support for RTOS

SDIO driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
SDIO_Deinit(card); /* This function will destroy the created mutex */
SDIO_Init(card);
```

#### Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/sdmmc\_examples/

#### Data Structures

- struct [sdio\\_card\\_t](#)  
*SDIO card state. More...*

#### Macros

- #define [FSL\\_SDIO\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2U, 4U, 1U)) /\*2.4.1\*/
*Middleware version.*
- #define [FSL\\_SDIO\\_MAX\\_IO\\_NUMS](#) (7U)
*sdio device support maximum IO number*

## Typedefs

- `typedef void(* sdio_io_irq_handler_t )(sdio_card_t *card, uint32_t func)`  
*sdio io handler*

## Enumerations

- `enum sdio_io_direction_t {  
 kSDIO_IORead = 0U,  
 kSDIO_IOWrite = 1U }`  
*sdio io read/write direction*

## Initialization and deinitialization

- `status_t SDIO_Init (sdio_card_t *card)`  
*SDIO card init function.*
- `void SDIO_Deinit (sdio_card_t *card)`  
*SDIO card\_deinit, include card and host\_deinit.*
- `status_t SDIO_CardInit (sdio_card_t *card)`  
*Initializes the card.*
- `void SDIO_CardDeinit (sdio_card_t *card)`  
*Deinitializes the card.*
- `status_t SDIO_HostInit (sdio_card_t *card)`  
*initialize the host.*
- `void SDIO_HostDeinit (sdio_card_t *card)`  
*Deinitializes the host.*
- `void SDIO_HostDoReset (sdio_card_t *card)`  
*reset the host.*
- `void SDIO_SetCardPower (sdio_card_t *card, bool enable)`  
*set card power.*
- `status_t SDIO_CardInActive (sdio_card_t *card)`  
*set SDIO card to inactive state*
- `status_t SDIO_GetCardCapability (sdio_card_t *card, sdio_func_num_t func)`  
*get SDIO card capability*
- `status_t SDIO_SetBlockSize (sdio_card_t *card, sdio_func_num_t func, uint32_t blockSize)`  
*set SDIO card block size*
- `status_t SDIO_CardReset (sdio_card_t *card)`  
*set SDIO card reset*
- `status_t SDIO_SetDataBusWidth (sdio_card_t *card, sdio_bus_width_t busWidth)`  
*set SDIO card data bus width*
- `status_t SDIO_SwitchToHighSpeed (sdio_card_t *card)`  
*switch the card to high speed*
- `status_t SDIO_ReadCIS (sdio_card_t *card, sdio_func_num_t func, const uint32_t *tupleList, uint32_t tupleNum)`  
*read SDIO card CIS for each function*
- `status_t SDIO_PollingCardInsert (sdio_card_t *card, uint32_t status)`  
*sdio wait card detect function.*
- `bool SDIO_IsCardPresent (sdio_card_t *card)`

*sdio card present check function.*

## IO operations

- `status_t SDIO_IO_Write_Direct` (`sdio_card_t *card`, `sdio_func_num_t func`, `uint32_t regAddr`, `uint8_t *data`, `bool raw`)
 

*IO direct write transfer function.*
- `status_t SDIO_IO_Read_Direct` (`sdio_card_t *card`, `sdio_func_num_t func`, `uint32_t regAddr`, `uint8_t *data`)
 

*IO direct read transfer function.*
- `status_t SDIO_IO_RW_Direct` (`sdio_card_t *card`, `sdio_io_direction_t direction`, `sdio_func_num_t func`, `uint32_t regAddr`, `uint8_t dataIn`, `uint8_t *dataOut`)
 

*IO direct read/write transfer function.*
- `status_t SDIO_IO_Write_Extended` (`sdio_card_t *card`, `sdio_func_num_t func`, `uint32_t regAddr`, `uint8_t *buffer`, `uint32_t count`, `uint32_t flags`)
 

*IO extended write transfer function.*
- `status_t SDIO_IO_Read_Extended` (`sdio_card_t *card`, `sdio_func_num_t func`, `uint32_t regAddr`, `uint8_t *buffer`, `uint32_t count`, `uint32_t flags`)
 

*IO extended read transfer function.*
- `status_t SDIO_EnableIOInterrupt` (`sdio_card_t *card`, `sdio_func_num_t func`, `bool enable`)
 

*enable IO interrupt*
- `status_t SDIO_EnableIO` (`sdio_card_t *card`, `sdio_func_num_t func`, `bool enable`)
 

*enable IO and wait IO ready*
- `status_t SDIO_SelectIO` (`sdio_card_t *card`, `sdio_func_num_t func`)
 

*select IO*
- `status_t SDIO_AbortIO` (`sdio_card_t *card`, `sdio_func_num_t func`)
 

*Abort IO transfer.*
- `status_t SDIO_SetDriverStrength` (`sdio_card_t *card`, `sd_driver_strength_t driverStrength`)
 

*Set driver strength.*
- `status_t SDIO_EnableAsyncInterrupt` (`sdio_card_t *card`, `bool enable`)
 

*Enable/Disable Async interrupt.*
- `status_t SDIO_GetPendingInterrupt` (`sdio_card_t *card`, `uint8_t *pendingInt`)
 

*Get pending interrupt.*
- `status_t SDIO_IO_Transfer` (`sdio_card_t *card`, `sdio_command_t cmd`, `uint32_t argument`, `uint32_t blockSize`, `uint8_t *txData`, `uint8_t *rxData`, `uint16_t dataSize`, `uint32_t *response`)
 

*sdio card io transfer function.*
- `void SDIO_SetIOIRQHandler` (`sdio_card_t *card`, `sdio_func_num_t func`, `sdio_io_irq_handler_t handler`)
 

*sdio set io IRQ handler.*
- `status_t SDIO_HandlePendingIOInterrupt` (`sdio_card_t *card`)
 

*sdio card io pending interrupt handle function.*

### 44.2.3 Data Structure Documentation

#### 44.2.3.1 struct \_sdio\_card

sdio card descriptor

Define the card structure including the necessary fields to identify and describe the card.

## Data Fields

- `sdmmchost_t * host`  
*Host information.*
- `sdio_usr_param_t usrParam`  
*user parameter*
- `bool noInternalAlign`  
*use this flag to disable sdmmc align.*
- `uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]`  
*internal buffer*
- `bool isHostReady`  
*use this flag to indicate if need host re-init or not*
- `bool memPresentFlag`  
*indicate if memory present*
- `uint32_t busClock_Hz`  
*SD bus clock frequency united in Hz.*
- `uint32_t relativeAddress`  
*Relative address of the card.*
- `uint8_t sdVersion`  
*SD version.*
- `sd_timing_mode_t currentTiming`  
*current timing mode*
- `sd_driver_strength_t driverStrength`  
*driver strength*
- `sd_max_current_t maxCurrent`  
*card current limit*
- `sdmmc_operation_voltage_t operationVoltage`  
*card operation voltage*
- `uint8_t sdioVersion`  
*SDIO version.*
- `uint8_t cccrVersioin`  
*CCCR version.*
- `uint8_t ioTotalNumber`  
*total number of IO function*
- `uint32_t cccrflags`  
*Flags in \_sd\_card\_flag.*
- `uint32_t io0blockSize`  
*record the io0 block size*
- `uint32_t ocr`  
*Raw OCR content, only 24bit available for SDIO card.*
- `uint32_t commonCISPointer`  
*point to common CIS*
- `sdio_common_cis_t commonCIS`  
*CIS table.*
- `sdio_fbr_t ioFBR [FSL_SDIO_MAX_IO_NUMS]`  
*FBR table.*
- `sdio_func_cis_t funcCIS [FSL_SDIO_MAX_IO_NUMS]`  
*function CIS table*
- `sdio_io_irq_handler_t ioIRQHandler [FSL_SDIO_MAX_IO_NUMS]`

- *io IRQ handler*
- `uint8_t ioIntIndex`  
*used to record current enabled io interrupt index*
- `uint8_t ioIntNums`  
*used to record total enabled io interrupt numbers*
- `sdmmc_osa_mutex_t lock`  
*card access lock*

## Field Documentation

### (1) `bool sdio_card_t::noInternalAlign`

If disable, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

## 44.2.4 Macro Definition Documentation

### 44.2.4.1 `#define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 1U)) /*2.4.1*/`

## 44.2.5 Enumeration Type Documentation

### 44.2.5.1 `enum sdio_io_direction_t`

Enumerator

- `kSDIO_IORead` io read
- `kSDIO_IOWrite` io write

## 44.2.6 Function Documentation

### 44.2.6.1 `status_t SDIO_Init( sdio_card_t * card )`

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDIO_Deinit(card);
* SDIO_Init(card);
*
```

Parameters

---

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Go-IdleFailed</i>	
<i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</i>	
<i>kStatus_SDMMC_SDIO-InvalidCard</i>	
<i>kStatus_SDMMC_SDIO-InvalidVoltage</i>	
<i>kStatus_SDMMC_Send-RelativeAddressFailed</i>	
<i>kStatus_SDMMC_Select-CardFailed</i>	
<i>kStatus_SDMMC_SDIO-SwitchHighSpeedFail</i>	
<i>kStatus_SDMMC_SDIO-ReadCISFail</i>	
<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.2 void SDIO\_Deinit ( **sdio\_card\_t** \* *card* )

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.2.6.3 status\_t SDIO\_CardInit ( **sdio\_card\_t** \* *card* )

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return *kStatus\_SDMMC\_HostNotReady*.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDIO_CardDeinit(card);
* SDIO_CardInit(card);
*
```

## Parameters

<i>card</i>	Card descriptor.
-------------	------------------

## Return values

<i>kStatus_SDMMC_Host-NotReady</i>	host is not ready.
<i>kStatus_SDMMC_Go-IdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_Not-SupportYet</i>	Card not support.
<i>kStatus_SDMMC_Send-OperationCondition-Failed</i>	Send operation condition failed.
<i>kStatus_SDMMC_All-SendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_Send-RelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_Send-CsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_Select-CardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_Send-ScrFailed</i>	Send SCR failed.
<i>kStatus_SDMMC_SetBus-WidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_Switch-HighSpeedFailed</i>	Switch high speed failed.

<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

#### 44.2.6.4 void SDIO\_CardDeinit ( *sdio\_card\_t* \* *card* )

This function deinitializes the specific card.

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.2.6.5 status\_t SDIO\_HostInit ( *sdio\_card\_t* \* *card* )

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.2.6.6 void SDIO\_HostDeinit ( *sdio\_card\_t* \* *card* )

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.2.6.7 void SDIO\_HostDoReset ( *sdio\_card\_t* \* *card* )

This function reset the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.2.6.8 void SDIO\_SetCardPower ( *sdio\_card\_t* \* *card*, *bool enable* )

The power off operation depend on host or the user define power on function.

Parameters

<i>card</i>	card descriptor.
<i>enable</i>	true is power on, false is power off.

#### 44.2.6.9 status\_t SDIO\_CardInActive ( **sdio\_card\_t \* card** )

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.10 status\_t SDIO\_GetCardCapability ( **sdio\_card\_t \* card, sdio\_func\_num\_t func** )

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.11 status\_t SDIO\_SetBlockSize ( **sdio\_card\_t \* card, sdio\_func\_num\_t func, uint32\_t blockSize** )

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	io number
<i>blockSize</i>	block size

Return values

<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	
<i>kStatus_SDMMC_SDIO_InvalidArgument</i>	
<i>kStatus_Success</i>	

#### 44.2.6.12 status\_t SDIO\_CardReset ( *sdio\_card\_t \* card* )

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.13 status\_t SDIO\_SetDataBusWidth ( *sdio\_card\_t \* card, sdio\_bus\_width\_t busWidth* )

Parameters

<i>card</i>	Card descriptor.
<i>busWidth</i>	bus width

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.14 status\_t SDIO\_SwitchToHighSpeed ( *sdio\_card\_t \* card* )

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_SDMMC_SDIO_-SwitchHighSpeedFail</i>	
<i>kStatus_Success</i>	

#### 44.2.6.15 status\_t SDIO\_ReadCIS ( *sdio\_card\_t \* card, sdio\_func\_num\_t func, const uint32\_t \* tupleList, uint32\_t tupleNum* )

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	io number
<i>tupleList</i>	code list
<i>tupleNum</i>	code number

Return values

<i>kStatus_SDMMC_SDIO_-ReadCISFail</i>	
<i>kStatus_SDMMC_-TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.16 status\_t SDIO\_PollingCardInsert ( *sdio\_card\_t \* card, uint32\_t status* )

Detect card through GPIO, CD, DATA3.

Parameters

<i>card</i>	card descriptor.
<i>status</i>	detect status, kSD_Inserted or kSD_Removed.

#### 44.2.6.17 bool SDIO\_IsCardPresent ( *sdio\_card\_t \* card* )

Parameters

<i>card</i>	card descriptor.
-------------	------------------

#### 44.2.6.18 status\_t SDIO\_IO\_Write\_Direct ( *sdio\_card\_t \* card*, *sdio\_func\_num\_t func*, *uint32\_t regAddr*, *uint8\_t \* data*, *bool raw* )

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>data</i>	the data pointer to write
<i>raw</i>	flag, indicate read after write or write only

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.19 status\_t SDIO\_IO\_Read\_Direct ( *sdio\_card\_t \* card*, *sdio\_func\_num\_t func*, *uint32\_t regAddr*, *uint8\_t \* data* )

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>data</i>	pointer to read

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.20 status\_t SDIO\_IO\_RW\_Direct ( *sdio\_card\_t \* card*, *sdio\_io\_direction\_t direction*, *sdio\_func\_num\_t func*, *uint32\_t regAddr*, *uint8\_t dataIn*, *uint8\_t \* dataOut* )

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>direction</i>	io access direction, please reference <i>sdio_io_direction_t</i> .
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>dataIn</i>	data to write
<i>dataOut</i>	data pointer for readback data, support both for read and write, when application want readback the data after write command, <i>dataOut</i> should not be NULL.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.21 status\_t SDIO\_IO\_Write\_Extended ( *sdio\_card\_t \* card*, *sdio\_func\_num\_t func*, *uint32\_t regAddr*, *uint8\_t \* buffer*, *uint32\_t count*, *uint32\_t flags* )

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>buffer</i>	data buffer to write
<i>count</i>	data count
<i>flags</i>	write flags

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_SDMMC_SDIO_InvalidArgument</i>	
<i>kStatus_Success</i>	

#### 44.2.6.22 **status\_t SDIO\_IO\_Read\_Extended ( *sdio\_card\_t \* card, sdio\_func\_num\_t func, uint32\_t regAddr, uint8\_t \* buffer, uint32\_t count, uint32\_t flags* )**

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>regAddr</i>	register address
<i>buffer</i>	data buffer to read
<i>count</i>	data count
<i>flags</i>	write flags

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_SDMMC_SDIO_InvalidArgument</i>	
<i>kStatus_Success</i>	

#### 44.2.6.23 **status\_t SDIO\_EnableOIInterrupt ( *sdio\_card\_t \* card, sdio\_func\_num\_t func, bool enable* )**

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>enable</i>	enable/disable flag

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.24 status\_t SDIO\_EnableIO ( ***sdio\_card\_t \* card, sdio\_func\_num\_t func, bool enable*** )

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number
<i>enable</i>	enable/disable flag

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.25 status\_t SDIO\_SelectIO ( ***sdio\_card\_t \* card, sdio\_func\_num\_t func*** )

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

44.2.6.26 status\_t SDIO\_AbortIO ( *sdio\_card\_t \* card*, *sdio\_func\_num\_t func* )

Parameters

<i>card</i>	Card descriptor.
<i>func</i>	IO number

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.27 status\_t SDIO\_SetDriverStrength ( **sdio\_card\_t \* card, sd\_driver\_strength\_t driverStrength** )

Parameters

<i>card</i>	Card descriptor.
<i>driverStrength</i>	target driver strength.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

#### 44.2.6.28 status\_t SDIO\_EnableAsyncInterrupt ( **sdio\_card\_t \* card, bool enable** )

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	true is enable, false is disable.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
-------------------------------------	--

<i>kStatus_Success</i>
------------------------

**44.2.6.29 status\_t SDIO\_GetPendingInterrupt ( *sdio\_card\_t \* card, uint8\_t \* pendingInt* )**

Parameters

<i>card</i>	Card descriptor.
<i>pendingInt</i>	pointer store pending interrupt

Return values

<i>kStatus_SDMMC_TransferFailed</i>	
<i>kStatus_Success</i>	

**44.2.6.30 status\_t SDIO\_IO\_Transfer ( *sdio\_card\_t \* card, sdio\_command\_t cmd, uint32\_t argument, uint32\_t blockSize, uint8\_t \* txData, uint8\_t \* rxData, uint16\_t dataSize, uint32\_t \* response* )**

This function can be used for transfer direct/extend command. Please pay attention to the non-align data buffer address transfer, if data buffer address can not meet host controller internal DMA requirement, sdio driver will try to use internal align buffer if data size is not bigger than internal buffer size, Align address transfer always can get a better performance, so if application want sdio driver make sure buffer address align,

Please note it is a thread safe function.

Parameters

<i>card</i>	card descriptor.
<i>cmd</i>	command to transfer
<i>argument</i>	argument to transfer
<i>blockSize</i>	used for block mode.
<i>txData</i>	tx buffer pointer or NULL

<i>rxData</i>	rx buffer pointer or NULL
<i>dataSize</i>	transfer data size
<i>response</i>	reponse pointer, if application want read response back, please set it to a NON-NULL pointer.

#### 44.2.6.31 void SDIO\_SetIOIRQHandler ( *sdio\_card\_t* \* *card*, *sdio\_func\_num\_t* *func*, *sdio\_io\_irq\_handler\_t* *handler* )

Parameters

<i>card</i>	card descriptor.
<i>func</i>	function io number.
<i>handler</i>	io IRQ handler.

#### 44.2.6.32 status\_t SDIO\_HandlePendingIOInterrupt ( *sdio\_card\_t* \* *card* )

This function is used to handle the pending io interrupt. To reigster a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, true);
* SDIO_SetIOIRQHandler(card, 0, func0_handler);
*
```

call it in interrupt callback

```
* SDIO_HandlePendingIOInterrupt(card);
*
```

To releae a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, false);
* SDIO_SetIOIRQHandler(card, 0, NULL);
*
```

Parameters

<i>card</i>	card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_- TransferFailed</i>	
<i>kStatus_Success</i>	

## 44.3 SD Card Driver

### 44.3.1 Overview

The SDCARD driver provide card initialization/read/write/erase interface.

### 44.3.2 SD CARD Operation

#### error log support

Lots of error log has been added to sd relate functions, if error occurs during initial/read/write, please enable the error log print functionality with `#define SDMMC_ENABLE_LOG_PRINT 1` And rerun the project then user can check what kind of error happened.

#### User configurable

```
typedef struct _sd_card
{
    sdmmchost_t *host;
    sd_usr_param_t usrParam;
    bool isHostReady;
    bool noInternalAlign;
    uint32_t busClock_Hz;
    uint32_t relativeAddress;
    uint32_t version;
    uint32_t flags;
    uint8_t internalBuffer[FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE];
    uint32_t ocr;
    sd_cid_t cid;
    sd_csd_t csd;
    sd_scr_t scr;
    sd_status_t stat;
    uint32_t blockCount;
    uint32_t blockSize;
    sd_timing_mode_t currentTiming;
    sd_driver_strength_t driverStrength;
    sd_max_current_t maxCurrent;
    sdmmc_operation_voltage_t operationVoltage;
    sdmmc_osa_mutex_t lock;
} sd_card_t;
```

Part of The variables above is user configurable,

##### 1. host

Application need to provide host controller base address and the host's source clock frequency, etc.  
For example:

```
/* allocate dma descriptor buffer for host controller */
s_host.dmaDesBuffer
s_host.dmaDesBufferWordsNum
/* */
((sd_card_t *)card)->host
((sd_card_t *)card)->host->hostController.base
((sd_card_t *)card)->host->hostController.sourceClock_Hz

/* allocate resource for sdmmc osa layer */
((sd_card_t *)card)->host->hostEvent = &s_event;
```

## 2. sdcard\_usr\_param\_t usrParam

```
/* board layer configuration register */
((sd_card_t *)card)->usrParam.cd      = &s_cd;
((sd_card_t *)card)->usrParam.pwr      = BOARD_SDCardPowerControl;
((sd_card_t *)card)->usrParam.ioStrength = BOARD_SD_Pin_Config;
((sd_card_t *)card)->usrParam.ioVoltage  = &s_ioVoltage;
((sd_card_t *)card)->usrParam.maxFreq    = BOARD_SDMMC_SD_HOST_SUPPORT_SDR104_FREQ;

a. cd-which allow application define the card insert/remove callback function, redefine the card detect
   timeout ms and also allow application determine how to detect card.
b. pwr-which allow application redefine the card power on/off function.
c. ioStrength-which is used to switch the signal pin configurations include driver strength/speed mode
   dynamically for different timing(SDR/HS timing) mode, reference the function defined sdmmc_config.c
d. ioVoltage-which allow application register io voltage switch function instead of using the function host
   driver provided for SDR/HS200/HS400 timing.
e. maxFreq-which allow application set the maximum bus clock that the board support.
```

### 1. bool noInternalAlign

Sdmmc include an address align internal buffer(to use host controller internal DMA), to improve read/write performance while application cannot make sure the data address used to read/write is align, set it to true will achieve a better performance.

### 2. sd\_timing\_mode\_t currentTiming

It is used to indicate the currentTiming the card is working on, however sdmmc also support preset timing mode, then sdmmc will try to switch to this timing first, if failed, a valid timing will switch to automatically. Generally, user may not set this variable if you don't know what kind of timing the card support, sdmmc will switch to the highest timing which the card support.

### 3. sd\_driver\_strength\_t driverStrength

Choose a valid card driver strength if application required and call SD\_SetDriverStrength in application.

### 4. sd\_max\_current\_t maxCurrent

Choose a valid card current if application required and call SD\_SetMaxCurrent in application.

## Mutual exclusive access support for RTOS

SDCARD driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
SD_Deinit(card); /* This function will destroy the created mutex */
SD_Init(card);
```

## Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/sdmmc\_examples/

## Data Structures

- struct `sd_card_t`  
*SD card state. More...*

## Macros

- #define **FSL\_SD\_DRIVER\_VERSION** (MAKE\_VERSION(2U, 4U, 1U)) /\*2.4.1\*/
 *Driver version.*

## Enumerations

- enum {
 **kSD\_SupportHighCapacityFlag** = (1U << 1U),
 **kSD\_Support4BitWidthFlag** = (1U << 2U),
 **kSD\_SupportSdhcFlag** = (1U << 3U),
 **kSD\_SupportSdxcFlag** = (1U << 4U),
 **kSD\_SupportVoltage180v** = (1U << 5U),
 **kSD\_SupportSetBlockCountCmd** = (1U << 6U),
 **kSD\_SupportSpeedClassControlCmd** = (1U << 7U) }
- SD card flags.*

## SDCARD Function

- **status\_t SD\_Init (sd\_card\_t \*card)**  
*Initializes the card on a specific host controller.*
- **void SD\_Deinit (sd\_card\_t \*card)**  
*Deinitializes the card.*
- **status\_t SD\_CardInit (sd\_card\_t \*card)**  
*Initializes the card.*
- **void SD\_CardDeinit (sd\_card\_t \*card)**  
*Deinitializes the card.*
- **status\_t SD\_HostInit (sd\_card\_t \*card)**  
*initialize the host.*
- **void SD\_HostDeinit (sd\_card\_t \*card)**  
*Deinitializes the host.*
- **void SD\_HostDoReset (sd\_card\_t \*card)**  
*reset the host.*
- **void SD\_SetCardPower (sd\_card\_t \*card, bool enable)**  
*set card power.*
- **status\_t SD\_PollingCardInsert (sd\_card\_t \*card, uint32\_t status)**  
*sd wait card detect function.*
- **bool SD\_IsCardPresent (sd\_card\_t \*card)**  
*sd card present check function.*
- **bool SD\_CheckReadOnly (sd\_card\_t \*card)**  
*Checks whether the card is write-protected.*
- **status\_t SD\_SelectCard (sd\_card\_t \*card, bool isSelected)**  
*Send SELECT\_CARD command to set the card to be transfer state or not.*
- **status\_t SD\_ReadStatus (sd\_card\_t \*card)**  
*Send ACMD13 to get the card current status.*
- **status\_t SD\_ReadBlocks (sd\_card\_t \*card, uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)**

- *Reads blocks from the specific card.*  
**status\_t SD\_WriteBlocks** (*sd\_card\_t* \*card, const *uint8\_t* \*buffer, *uint32\_t* startBlock, *uint32\_t* blockCount)
  - Writes blocks of data to the specific card.*
- *Erase blocks of the specific card.*  
**status\_t SD\_EraseBlocks** (*sd\_card\_t* \*card, *uint32\_t* startBlock, *uint32\_t* blockCount)
  - Select card driver strength select card driver strength*
- *select max current select max operation current*  
**status\_t SD\_SetDriverStrength** (*sd\_card\_t* \*card, *sd\_driver\_strength\_t* driverStrength)
  - select card driver strength select card driver strength*
- *Polling card idle status.*  
**status\_t SD\_PollingCardStatusBusy** (*sd\_card\_t* \*card, *uint32\_t* timeoutMs)
  - Polling card idle status.*

### 44.3.3 Data Structure Documentation

#### 44.3.3.1 struct sd\_card\_t

Define the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- **sdmmchost\_t \* host**  
*Host configuration.*
- **sd\_usr\_param\_t usrParam**  
*User parameter.*
- **bool isHostReady**  
*use this flag to indicate if need host re-init or not*
- **bool noInternalAlign**  
*used to enable/disable the functionality of the exchange buffer*
- **uint32\_t busClock\_Hz**  
*SD bus clock frequency unitized in Hz.*
- **uint32\_t relativeAddress**  
*Relative address of the card.*
- **uint32\_t version**  
*Card version.*
- **uint32\_t flags**  
*Flags in \_sd\_card\_flag.*
- **uint8\_t internalBuffer [FSL\_SDMMC\_CARD\_INTERNAL\_BUFFER\_SIZE]**  
*internal buffer*
- **uint32\_t ocr**  
*Raw OCR content.*
- **sd\_cid\_t cid**  
*CID.*
- **sd\_csd\_t csd**  
*CSD.*
- **sd\_scr\_t scr**  
*SCR.*
- **sd\_status\_t stat**

- *sd 512 bit status*
- **uint32\_t blockCount**  
*Card total block number.*
- **uint32\_t blockSize**  
*Card block size.*
- **sd\_timing\_mode\_t currentTiming**  
*current timing mode*
- **sd\_driver\_strength\_t driverStrength**  
*driver strength*
- **sd\_max\_current\_t maxCurrent**  
*card current limit*
- **sdmmc\_operation\_voltage\_t operationVoltage**  
*card operation voltage*
- **sdmmc\_osu\_mutex\_t lock**  
*card access lock*

#### 44.3.4 Macro Definition Documentation

44.3.4.1 #define FSL\_SD\_DRIVER\_VERSION (MAKE\_VERSION(2U, 4U, 1U)) /\*2.4.1\*/

#### 44.3.5 Enumeration Type Documentation

##### 44.3.5.1 anonymous enum

Enumerator

- kSD\_SupportHighCapacityFlag** Support high capacity.
- kSD\_Support4BitWidthFlag** Support 4-bit data width.
- kSD\_SupportSdhcFlag** Card is SDHC.
- kSD\_SupportSdxcFlag** Card is SDXC.
- kSD\_SupportVoltage180v** card support 1.8v voltage
- kSD\_SupportSetBlockCountCmd** card support cmd23 flag
- kSD\_SupportSpeedClassControlCmd** card support speed class control flag

#### 44.3.6 Function Documentation

##### 44.3.6.1 status\_t SD\_Init ( sd\_card\_t \* card )

This function initializes the card on a specific host controller, it is consist of host init, card detect, card init function, however user can ignore this high level function, instead of use the low level function, such as SD\_CardInit, SD\_HostInit, SD\_CardDetect.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

\* SD\_Deinit (card);

```
* SD_Init(card);
*
```

## Parameters

<i>card</i>	Card descriptor.
-------------	------------------

## Return values

<i>kStatus_SDMMC_HostNotReady</i>	host is not ready.
<i>kStatus_SDMMC_GoIdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_NotSupportYet</i>	Card not support.
<i>kStatus_SDMMC_HandShakeOperationConditionFailed</i>	Send operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_SendRelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_SelectCardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_SendScrFailed</i>	Send SCR failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_SwitchBusTimingFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Set card block size failed.

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

#### 44.3.6.2 void SD\_Deinit ( *sd\_card\_t \* card* )

This function deinitializes the specific card and host. Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.3.6.3 status\_t SD\_CardInit ( *sd\_card\_t \* card* )

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return *kStatus\_SDMMC\_HostNotReady*.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SD_CardDeinit(card);
* SD_CardInit(card);
*
```

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	host is not ready.
<i>kStatus_SDMMC_Go-IdleFailed</i>	Go idle failed.
<i>kStatus_SDMMC_Not-SupportYet</i>	Card not support.
<i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</i>	Send operation condition failed.

<i>kStatus_SDMMC_AllSendCidFailed</i>	Send CID failed.
<i>kStatus_SDMMC_SendRelativeAddressFailed</i>	Send relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Send CSD failed.
<i>kStatus_SDMMC_SelectCardFailed</i>	Send SELECT_CARD command failed.
<i>kStatus_SDMMC_SendScrFailed</i>	Send SCR failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Set bus width failed.
<i>kStatus_SDMMC_SwitchBusTimingFailed</i>	Switch high speed failed.
<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Set card block size failed.
<i>kStatus_Success</i>	Operate successfully.

#### 44.3.6.4 void SD\_CardDeinit( *sd\_card\_t* \* *card* )

This function deinitializes the specific card. Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.3.6.5 status\_t SD\_HostInit( *sd\_card\_t* \* *card* )

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.3.6.6 void SD\_HostDeinit( *sd\_card\_t* \* *card* )

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.3.6.7 void SD\_HostDoReset ( *sd\_card\_t* \* *card* )

This function reset the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.3.6.8 void SD\_SetCardPower ( *sd\_card\_t* \* *card*, *bool* *enable* )

The power off operation depend on host or the user define power on function.

Parameters

<i>card</i>	card descriptor.
<i>enable</i>	true is power on, false is power off.

#### 44.3.6.9 *status\_t* SD\_PollingCardInsert ( *sd\_card\_t* \* *card*, *uint32\_t* *status* )

Detect card through GPIO, CD, DATA3.

Parameters

<i>card</i>	card descriptor.
<i>status</i>	detect status, kSD_Inserted or kSD_Removed.

#### 44.3.6.10 *bool* SD\_IsCardPresent ( *sd\_card\_t* \* *card* )

Parameters

<i>card</i>	card descriptor.
-------------	------------------

#### 44.3.6.11 *bool* SD\_CheckReadOnly ( *sd\_card\_t* \* *card* )

This function checks if the card is write-protected via the CSD register.

Parameters

<i>card</i>	The specific card.
-------------	--------------------

Return values

<i>true</i>	Card is read only.
<i>false</i>	Card isn't read only.

#### 44.3.6.12 status\_t SD\_SelectCard ( *sd\_card\_t \* card, bool isSelected* )

Parameters

<i>card</i>	Card descriptor.
<i>isSelected</i>	True to set the card into transfer state, false to disselect.

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

#### 44.3.6.13 status\_t SD\_ReadStatus ( *sd\_card\_t \* card* )

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_SendApplicationCommandFailed</i>	send application command failed.

<i>kStatus_Success</i>	Operate successfully.
------------------------	-----------------------

#### 44.3.6.14 status\_t SD\_ReadBlocks ( *sd\_card\_t \* card*, *uint8\_t \* buffer*, *uint32\_t startBlock*, *uint32\_t blockCount* )

This function reads blocks from the specific card with default block size defined by the SDHC\_CARD\_DEFAULT\_BLOCK\_SIZE.

Please note it is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save the data read from card.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to read.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Not-SupportYet</i>	Not support now.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Stop-TransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

#### 44.3.6.15 status\_t SD\_WriteBlocks ( *sd\_card\_t \* card*, *const uint8\_t \* buffer*, *uint32\_t startBlock*, *uint32\_t blockCount* )

This function writes blocks to the specific card with default block size 512 bytes.

Please note,

1. It is a thread safe function.
2. It is a sync write function which means that the card status may still busy after the function return.

Application can call function SD\_PollingCardStatusBusy to wait card status idle after the write operation.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer holding the data to be written to the card.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to write.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Not-SupportYet</i>	Not support now.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Stop-TransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operate successfully.

#### 44.3.6.16 **status\_t SD\_EraseBlocks ( sd\_card\_t \* card, uint32\_t startBlock, uint32\_t blockCount )**

This function erases blocks of the specific card with default block size 512 bytes.

Please note,

1. It is a thread safe function.
2. It is a async erase function which means that the card status may still busy after the function return.  
Application can call function SD\_PollingCardStatusBusy to wait card status idle after the erase operation.

Parameters

<i>card</i>	Card descriptor.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to erase.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	Send status failed.
<i>kStatus_Success</i>	Operate successfully.

#### 44.3.6.17 **status\_t SD\_SetDriverStrength ( *sd\_card\_t \* card, sd\_driver\_strength\_t driverStrength* )**

Parameters

<i>card</i>	Card descriptor.
<i>driverStrength</i>	Driver strength

#### 44.3.6.18 **status\_t SD\_SetMaxCurrent ( *sd\_card\_t \* card, sd\_max\_current\_t maxCurrent* )**

Parameters

<i>card</i>	Card descriptor.
<i>maxCurrent</i>	Max current

#### 44.3.6.19 **status\_t SD\_PollingCardStatusBusy ( *sd\_card\_t \* card, uint32\_t timeoutMs* )**

This function can be used to polling the status from busy to Idle, the function will return if the card status idle or timeout.

## Parameters

<i>card</i>	Card descriptor.
<i>timeoutMs</i>	polling card status timeout value.

## Return values

<i>kStatus_Success</i>	Operate successfully.
<i>kStatus_SDMMC_Wait-WriteCompleteFailed</i>	CMD13 transfer failed.
<i>kStatus_SDMMC_-PollingCardIdle-Failed,polling</i>	card DAT0 idle failed.

## 44.4 MMC Card Driver

### 44.4.1 Overview

The MMCCARD driver provide card initialization/read/write/erase interface.

### 44.4.2 MMC CARD Operation

#### error log support

Not support yet

#### User configurable

#### Board dependency

#### Mutual exclusive access support for RTOS

MMCCARD driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization.

```
MMC_Deinit(card); /* This function will destroy the created mutex */
MMC_Init(card);
```

#### Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/sdmmc\_examples/

## Data Structures

- struct [mmc\\_usr\\_param\\_t](#)  
*card user parameter [More...](#)*
- struct [mmc\\_card\\_t](#)  
*mmc card state [More...](#)*

## Macros

- #define [FSL\\_MMC\\_DRIVER\\_VERSION](#)([MAKE\\_VERSION](#)(2U, 5U, 0U)) /\*2.5.0\*/
*Middleware mmc version.*

## Typedefs

- `typedef void(* mmc_io_strength_t )(uint32_t busFreq)`  
*card io strength control*

## Enumerations

- `enum {`  
 `kMMC_SupportHighSpeed26MHZFlag = (1U << 0U),`  
 `kMMC_SupportHighSpeed52MHZFlag = (1U << 1U),`  
 `kMMC_SupportHighSpeedDDR52MHZ180V300VFlag = (1 << 2U),`  
 `kMMC_SupportHighSpeedDDR52MHZ120VFlag = (1 << 3U),`  
 `kMMC_SupportHS200200MHZ180VFlag = (1 << 4U),`  
 `kMMC_SupportHS200200MHZ120VFlag = (1 << 5U),`  
 `kMMC_SupportHS400DDR200MHZ180VFlag = (1 << 6U),`  
 `kMMC_SupportHS400DDR200MHZ120VFlag = (1 << 7U),`  
 `kMMC_SupportHighCapacityFlag = (1U << 8U),`  
 `kMMC_SupportAlternateBootFlag = (1U << 9U),`  
 `kMMC_SupportDDRBootFlag = (1U << 10U),`  
 `kMMC_SupportHighSpeedBootFlag = (1U << 11U),`  
 `kMMC_SupportEnhanceHS400StrobeFlag = (1U << 12U) }`  
*MMC card flags.*
- `enum mmc_sleep_awake_t {`  
 `kMMC_Sleep = 1U,`  
 `kMMC_Awake = 0U }`  
*mmccard sleep/awake state*

## MMCCARD Function

- `status_t MMC_Init (mmc_card_t *card)`  
*Initializes the MMC card and host.*
- `void MMC_Deinit (mmc_card_t *card)`  
*Deinitializes the card and host.*
- `status_t MMC_CardInit (mmc_card_t *card)`  
*Initializes the card.*
- `void MMC_CardDeinit (mmc_card_t *card)`  
*Deinitializes the card.*
- `status_t MMC_HostInit (mmc_card_t *card)`  
*initialize the host.*
- `void MMC_HostDeinit (mmc_card_t *card)`  
*Deinitializes the host.*
- `void MMC_HostDoReset (mmc_card_t *card)`  
*Resets the host.*
- `void MMC_HostReset (SDMMCHOST_CONFIG *host)`  
*Resets the host.*
- `void MMC_SetCardPower (mmc_card_t *card, bool enable)`

- **bool MMC\_CheckReadOnly (mmc\_card\_t \*card)**  
*Checks if the card is read-only.*
- **status\_t MMC\_ReadBlocks (mmc\_card\_t \*card, uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)**  
*Reads data blocks from the card.*
- **status\_t MMC\_WriteBlocks (mmc\_card\_t \*card, const uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)**  
*Writes data blocks to the card.*
- **status\_t MMC\_EraseGroups (mmc\_card\_t \*card, uint32\_t startGroup, uint32\_t endGroup)**  
*Erases groups of the card.*
- **status\_t MMC\_SelectPartition (mmc\_card\_t \*card, mmc\_access\_partition\_t partitionNumber)**  
*Selects the partition to access.*
- **status\_t MMC\_SetBootConfig (mmc\_card\_t \*card, const mmc\_boot\_config\_t \*config)**  
*Configures the boot activity of the card.*
- **status\_t MMC\_StartBoot (mmc\_card\_t \*card, const mmc\_boot\_config\_t \*mmcConfig, uint8\_t \*buffer, sdmmchost\_boot\_config\_t \*hostConfig)**  
*MMC card start boot.*
- **status\_t MMC\_SetBootConfigWP (mmc\_card\_t \*card, uint8\_t wp)**  
*MMC card set boot configuration write protect.*
- **status\_t MMC\_ReadBootData (mmc\_card\_t \*card, uint8\_t \*buffer, sdmmchost\_boot\_config\_t \*hostConfig)**  
*MMC card continuous read boot data.*
- **status\_t MMC\_StopBoot (mmc\_card\_t \*card, uint32\_t bootMode)**  
*MMC card stop boot mode.*
- **status\_t MMC\_SetBootPartitionWP (mmc\_card\_t \*card, mmc\_boot\_partition\_wp\_t bootPartitionWP)**  
*MMC card set boot partition write protect.*
- **status\_t MMC\_EnableCacheControl (mmc\_card\_t \*card, bool enable)**  
*MMC card cache control function.*
- **status\_t MMC\_FlushCache (mmc\_card\_t \*card)**  
*MMC card cache flush function.*
- **status\_t MMC\_SetSleepAwake (mmc\_card\_t \*card, mmc\_sleep\_awake\_t state)**  
*MMC sets card sleep awake state.*
- **status\_t MMC\_PollingCardStatusBusy (mmc\_card\_t \*card, bool checkStatus, uint32\_t timeoutMs)**  
*Polling card idle status.*

#### 44.4.3 Data Structure Documentation

##### 44.4.3.1 struct mmc\_usr\_param\_t

###### Data Fields

- **mmc\_io\_strength\_t ioStrength**  
*switch sd io strength*
- **uint32\_t maxFreq**  
*board support maximum frequency*
- **uint32\_t capability**  
*board capability flag*

#### 44.4.3.2 struct mmc\_card\_t

Defines the card structure including the necessary fields to identify and describe the card.

##### Data Fields

- `sdmmchost_t * host`  
*Host information.*
- `mmc_usr_param_t usrParam`  
*user parameter*
- `bool isHostReady`  
*Use this flag to indicate if host re-init needed or not.*
- `bool noInternalAlign`  
*Use this flag to disable sdmmc align.*
- `uint32_t busClock_Hz`  
*MMC bus clock united in Hz.*
- `uint32_t relativeAddress`  
*Relative address of the card.*
- `bool enablePreDefinedBlockCount`  
*Enable PRE-DEFINED block count when read/write.*
- `uint32_t flags`  
*Capability flag in `_mmc_card_flag`.*
- `uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]`  
*raw buffer used for mmc driver internal*
- `uint32_t ocr`  
*Raw OCR content.*
- `mmc_cid_t cid`  
*CID.*
- `mmc_csd_t csd`  
*CSD.*
- `mmc_extended_csd_t extendedCsd`  
*Extended CSD.*
- `uint32_t blockSize`  
*Card block size.*
- `uint32_t userPartitionBlocks`  
*Card total block number in user partition.*
- `uint32_t bootPartitionBlocks`  
*Boot partition size united as block size.*
- `uint32_t eraseGroupBlocks`  
*Erase group size united as block size.*
- `mmc_access_partition_t currentPartition`  
*Current access partition.*
- `mmc_voltage_window_t hostVoltageWindowVCCQ`  
*application must set this value according to board specific*
- `mmc_voltage_window_t hostVoltageWindowVCC`  
*application must set this value according to board specific*
- `mmc_high_speed_timing_t busTiming`  
*indicates the current work timing mode*
- `mmc_data_bus_width_t busWidth`  
*indicates the current work bus width*

- `sdmmc_osa_mutex_t lock`  
*card access lock*

## Field Documentation

### (1) `bool mmc_card_t::noInternalAlign`

If disabled, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are aligned to low level driver.

## 44.4.4 Macro Definition Documentation

### 44.4.4.1 `#define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 5U, 0U)) /*2.5.0*/`

## 44.4.5 Enumeration Type Documentation

### 44.4.5.1 anonymous enum

Enumerator

*kMMC\_SupportHighSpeed26MHZFlag* Support high speed 26MHZ.  
*kMMC\_SupportHighSpeed52MHZFlag* Support high speed 52MHZ.  
*kMMC\_SupportHighSpeedDDR52MHZ180V300VFlag* ddr 52MHZ 1.8V or 3.0V  
*kMMC\_SupportHighSpeedDDR52MHZ120VFlag* DDR 52MHZ 1.2V.  
*kMMC\_SupportHS200200MHZ180VFlag* HS200 ,200MHZ,1.8V.  
*kMMC\_SupportHS200200MHZ120VFlag* HS200, 200MHZ, 1.2V.  
*kMMC\_SupportHS400DDR200MHZ180VFlag* HS400, DDR, 200MHZ,1.8V.  
*kMMC\_SupportHS400DDR200MHZ120VFlag* HS400, DDR, 200MHZ,1.2V.  
*kMMC\_SupportHighCapacityFlag* Support high capacity.  
*kMMC\_SupportAlternateBootFlag* Support alternate boot.  
*kMMC\_SupportDDRBootFlag* support DDR boot flag  
*kMMC\_SupportHighSpeedBootFlag* support high speed boot flag  
*kMMC\_SupportEnhanceHS400StrobeFlag* support enhance HS400 strobe

### 44.4.5.2 `enum mmc_sleep_awake_t`

Enumerator

*kMMC\_Sleep* MMC card sleep.  
*kMMC\_Awake* MMC card awake.

## 44.4.6 Function Documentation

#### 44.4.6.1 status\_t MMC\_Init( mmc\_card\_t \* *card* )

## Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex to be created redundantly, application must follow bellow sequence for card re-initialization:

```
MMC_Deinit(card);
MMC_Init(card);
*
```

## Return values

<i>kStatus_SDMMC_HostNotReady</i>	Host is not ready.
<i>kStatus_SDMMC_GoIdleFailed</i>	Going idle failed.
<i>kStatus_SDMMC_HandShakeOperationConditionFailed</i>	Sending operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Sending CID failed.
<i>kStatus_SDMMC_SetRelativeAddressFailed</i>	Setgng relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Sending CSD failed.
<i>kStatus_SDMMC_CardNotSupport</i>	Card not support.
<i>kStatus_SDMMC_SelectCardFailed</i>	Sending SELECT_CARD command failed.
<i>kStatus_SDMMC_SendExtendedCsdFailed</i>	Sending EXT_CSD failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Setting bus width failed.

<i>kStatus_SDMMC_Switch-BusTimingFailed</i>	Switching high speed failed.
<i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>	Setting card block size failed.
<i>kStatus_SDMMC_Set-PowerClassFail</i>	Setting card power class failed.
<i>kStatus_Success</i>	Operation succeeded.

#### 44.4.6.2 void MMC\_Deinit ( mmc\_card\_t \* *card* )

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.4.6.3 status\_t MMC\_CardInit ( mmc\_card\_t \* *card* )

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex to be created redundantly, application must follow bellow sequence for card re-initialization:

```
MMC_CardDeinit(card);
MMC_CardInit(card);
*
```

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>kStatus_SDMMC_Host-NotReady</i>	Host is not ready.
------------------------------------	--------------------

<i>kStatus_SDMMC_GoIdleFailed</i>	Going idle failed.
<i>kStatus_SDMMC_HandShakeOperationConditionFailed</i>	Sending operation condition failed.
<i>kStatus_SDMMC_AllSendCidFailed</i>	Sending CID failed.
<i>kStatus_SDMMC_SetRelativeAddressFailed</i>	Setting relative address failed.
<i>kStatus_SDMMC_SendCsdFailed</i>	Sending CSD failed.
<i>kStatus_SDMMC_CardNotSupport</i>	Card not support.
<i>kStatus_SDMMC_SelectCardFailed</i>	Sending SELECT_CARD command failed.
<i>kStatus_SDMMC_SendExtendedCsdFailed</i>	Sending EXT_CSD failed.
<i>kStatus_SDMMC_SetDataBusWidthFailed</i>	Setting bus width failed.
<i>kStatus_SDMMC_SwitchBusTimingFailed</i>	Switching high speed failed.
<i>kStatus_SDMMC_SetCardBlockSizeFailed</i>	Setting card block size failed.
<i>kStatus_SDMMC_SetPowerClassFail</i>	Setting card power class failed.
<i>kStatus_Success</i>	Operation succeeded.

#### 44.4.6.4 void MMC\_CardDeinit ( mmc\_card\_t \* **card** )

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.4.6.5 status\_t MMC\_HostInit ( mmc\_card\_t \* *card* )

This function deinitializes the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.4.6.6 void MMC\_HostDeinit ( mmc\_card\_t \* *card* )

This function deinitializes the host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.4.6.7 void MMC\_HostDoReset ( mmc\_card\_t \* *card* )

This function resets the specific host.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.4.6.8 void MMC\_HostReset ( SDMMCHOST\_CONFIG \* *host* )

**Deprecated** Do not use this function. It has been superceded by [MMC\\_HostDoReset](#). This function resets the specific host.

Parameters

<i>host</i>	Host descriptor.
-------------	------------------

#### 44.4.6.9 void MMC\_SetCardPower ( mmc\_card\_t \* *card*, bool *enable* )

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	True is powering on, false is powering off.

#### 44.4.6.10 bool MMC\_CheckReadOnly ( mmc\_card\_t \* *card* )

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

Return values

<i>true</i>	Card is read only.
<i>false</i>	Card isn't read only.

#### 44.4.6.11 status\_t MMC\_ReadBlocks ( mmc\_card\_t \* *card*, uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save data.
<i>startBlock</i>	The start block index.
<i>blockCount</i>	The number of blocks to read.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_Card-NotSupport</i>	Card not support.

<i>kStatus_SDMMC_SetBlockCountFailed</i>	Setting block count failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_StopTransmissionFailed</i>	Stopping transmission failed.
<i>kStatus_Success</i>	Operation succeeded.

#### 44.4.6.12 **status\_t MMC\_WriteBlocks ( mmc\_card\_t \* card, const uint8\_t \* buffer, uint32\_t startBlock, uint32\_t blockCount )**

Note

1. It is a thread safe function.
2. It is an async write function which means that the card status may still be busy after the function returns. Application can call function MMC\_PollingCardStatusBusy to wait for the card status to be idle after the write operation.

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	The buffer to save data blocks.
<i>startBlock</i>	Start block number to write.
<i>blockCount</i>	Block count.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_NotSupportYet</i>	Not support now.
<i>kStatus_SDMMC_SetBlockCountFailed</i>	Setting block count failed.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Sending status failed.

<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_StopTransmissionFailed</i>	Stop transmission failed.
<i>kStatus_Success</i>	Operation succeeded.

#### 44.4.6.13 status\_t MMC\_EraseGroups ( *mmc\_card\_t \* card*, *uint32\_t startGroup*, *uint32\_t endGroup* )

The erase command is best used to erase the entire device or a partition. Erase group is the smallest erase unit in MMC card. The erase range is [startGroup, endGroup].

Note

1. It is a thread safe function.
2. This function always polls card busy status according to the timeout value defined in the card register after all the erase command sent out.

Parameters

<i>card</i>	Card descriptor.
<i>startGroup</i>	Start group number.
<i>endGroup</i>	End group number.

Return values

<i>kStatus_InvalidArgument</i>	Invalid argument.
<i>kStatus_SDMMC_WaitWriteCompleteFailed</i>	Send status failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operation succeeded.

#### 44.4.6.14 status\_t MMC\_SelectPartition ( *mmc\_card\_t \* card*, *mmc\_access\_partition\_t partitionNumber* )

Note

It is a thread safe function.

Parameters

<i>card</i>	Card descriptor.
<i>partition-Number</i>	The partition number.

Return values

<i>kStatus_SDMMC_ConfigureExtendedCsdFailed</i>	Configuring EXT_CSD failed.
<i>kStatus_Success</i>	Operation succeeded.

#### 44.4.6.15 status\_t MMC\_SetBootConfig ( mmc\_card\_t \* *card*, const mmc\_boot\_config\_t \* *config* )

Parameters

<i>card</i>	Card descriptor.
<i>config</i>	Boot configuration structure.

Return values

<i>kStatus_SDMMC_NotSupportYet</i>	Not support now.
<i>kStatus_SDMMC_ConfigureExtendedCsdFailed</i>	Configuring EXT_CSD failed.
<i>kStatus_SDMMC_ConfigureBootFailed</i>	Configuring boot failed.
<i>kStatus_Success</i>	Operation succeeded.

#### 44.4.6.16 status\_t MMC\_StartBoot ( mmc\_card\_t \* *card*, const mmc\_boot\_config\_t \* *mmcConfig*, uint8\_t \* *buffer*, sdmmchost\_boot\_config\_t \* *hostConfig* )

Parameters

<i>card</i>	Card descriptor.
<i>mmcConfig</i>	The mmc Boot configuration structure.
<i>buffer</i>	Address to receive data.
<i>hostConfig</i>	Host boot configurations.

Return values

<i>kStatus_Fail</i>	Failed.
<i>kStatus_SDMMC_TransferFailed</i>	Transfer failed.
<i>kStatus_SDMMC_Go_IdleFailed</i>	Resetting card failed.
<i>kStatus_Success</i>	Operation succeeded.

#### 44.4.6.17 status\_t MMC\_SetBootConfigWP ( *mmc\_card\_t \* card, uint8\_t wp* )

Parameters

<i>card</i>	Card descriptor.
<i>wp</i>	Write protect value.

#### 44.4.6.18 status\_t MMC\_ReadBootData ( *mmc\_card\_t \* card, uint8\_t \* buffer, sdmmchost\_boot\_config\_t \* hostConfig* )

Parameters

<i>card</i>	Card descriptor.
<i>buffer</i>	Buffer address.
<i>hostConfig</i>	Host boot configurations.

#### 44.4.6.19 status\_t MMC\_StopBoot ( *mmc\_card\_t \* card, uint32\_t bootMode* )

Parameters

<i>card</i>	Card descriptor.
<i>bootMode</i>	Boot mode.

#### 44.4.6.20 status\_t MMC\_SetBootPartitionWP ( *mmc\_card\_t \* card*, *mmc\_boot\_partition\_wp\_t bootPartitionWP* )

Parameters

<i>card</i>	Card descriptor.
<i>bootPartition-WP</i>	Boot partition write protect value.

#### 44.4.6.21 status\_t MMC\_EnableCacheControl ( *mmc\_card\_t \* card*, *bool enable* )

The mmc device's cache is enabled by the driver by default. The cache should in typical case reduce the access time (compared to an access to the main nonvolatile storage) for both write and read.

Parameters

<i>card</i>	Card descriptor.
<i>enable</i>	True is enabling the cache, false is disabling the cache.

#### 44.4.6.22 status\_t MMC\_FlushCache ( *mmc\_card\_t \* card* )

A Flush operation refers to the requirement, from the host to the device, to write the cached data to the nonvolatile memory. Prior to a flush, the device may autonomously write data to the nonvolatile memory, but after the flush operation all data in the volatile area must be written to nonvolatile memory. There is no requirement for flush due to switching between the partitions. (Note: This also implies that the cache data shall not be lost when switching between partitions). Cached data may be lost in SLEEP state, so host should flush the cache before placing the device into SLEEP state.

Parameters

<i>card</i>	Card descriptor.
-------------	------------------

#### 44.4.6.23 status\_t MMC\_SetSleepAwake ( mmc\_card\_t \* card, mmc\_sleep\_awake\_t state )

The Sleep/Awake command is used to initiate the state transition between Standby state and Sleep state. The memory device indicates the transition phase busy by pulling down the DAT0 line. The Sleep/Standby state is reached when the memory device stops pulling down the DAT0 line, then the function returns.

Parameters

<i>card</i>	Card descriptor.
<i>state</i>	The sleep/awake command argument, refer to <a href="#">mmc_sleep_awake_t</a> .

Return values

<i>kStatus_SDMMC_NotSupportYet</i>	Indicates the memory device doesn't support the Sleep/Awake command.
<i>kStatus_SDMMC_TransferFailed</i>	Indicates command transferred fail.
<i>kStatus_SDMMC_PollingCardIdleFailed</i>	Indicates polling DAT0 busy timeout.
<i>kStatus_SDMMC_DeselectCardFailed</i>	Indicates deselect card command failed.
<i>kStatus_SDMMC_SelectCardFailed</i>	Indicates select card command failed.
<i>kStatus_Success</i>	Indicates the card state switched successfully.

#### 44.4.6.24 status\_t MMC\_PollingCardStatusBusy ( mmc\_card\_t \* card, bool checkStatus, uint32\_t timeoutMs )

This function can be used to poll the status from busy to idle, the function will return with the card status being idle or timeout or command failed.

Parameters

<i>card</i>	Card descriptor.
<i>checkStatus</i>	True is send CMD and read DAT0 status to check card status, false is read DAT0 status only.
<i>timeoutMs</i>	Polling card status timeout value.

Return values

<i>kStatus_SDMMC_Card-StatusIdle</i>	Card is idle.
<i>kStatus_SDMMC_Card-StatusBusy</i>	Card is busy.
<i>kStatus_SDMMC_TransferFailed</i>	Command transfer failed.
<i>kStatus_SDMMC_SwitchFailed</i>	Status command reports switch error.

## 44.5 SDMMC HOST Driver

### 44.5.1 Overview

The host adapter driver provide adapter for blocking/non\_blocking mode.

### Modules

- [SDIF HOST adapter Driver](#)

## 44.6 SDMMC OSA

### 44.6.1 Overview

The sdmmc osa adapter provide interface of os adapter.

### Data Structures

- struct `sdmmc_osa_event_t`  
*sdmmc osa event* [More...](#)
- struct `sdmmc_osa_mutex_t`  
*sdmmc osa mutex* [More...](#)

### Macros

- #define `SDMMC_OSA_EVENT_TRANSFER_CMD_SUCCESS` (1UL << 0U)  
*transfer event*
- #define `SDMMC_OSA_EVENT_CARD_INSERTED` (1UL << 8U)  
*card detect event, start from index 8*
- #define `SDMMC_OSA_POLLING_EVENT_BY_SEMPHORE` 1  
*enable semaphore by default*

### sdmmc osa Function

- void `SDMMC_OSAInit` (void)  
*Initialize OSA.*
- `status_t SDMMC_OSAEventCreate` (void \*eventHandle)  
*OSA Create event.*
- `status_t SDMMC_OSAEventWait` (void \*eventHandle, uint32\_t eventType, uint32\_t timeoutMilliseconds, uint32\_t \*event)  
*Wait event.*
- `status_t SDMMC_OSAEventSet` (void \*eventHandle, uint32\_t eventType)  
*set event.*
- `status_t SDMMC_OSAEventGet` (void \*eventHandle, uint32\_t eventType, uint32\_t \*flag)  
*Get event flag.*
- `status_t SDMMC_OSAEventClear` (void \*eventHandle, uint32\_t eventType)  
*clear event flag.*
- `status_t SDMMC_OSAEventDestroy` (void \*eventHandle)  
*Delete event.*
- `status_t SDMMC_OSAMutexCreate` (void \*mutexHandle)  
*Create a mutex.*
- `status_t SDMMC_OSAMutexLock` (void \*mutexHandle, uint32\_t millisec)  
*set event.*
- `status_t SDMMC_OSAMutexUnlock` (void \*mutexHandle)  
*Get event flag.*
- `status_t SDMMC_OSAMutexDestroy` (void \*mutexHandle)  
*Delete mutex.*

- void **SDMMC\_OSA****Delay** (uint32\_t milliseconds)  
*sdmmc delay.*
- uint32\_t **SDMMC\_OSA****DelayUs** (uint32\_t microseconds)  
*sdmmc delay us.*

## 44.6.2 Data Structure Documentation

### 44.6.2.1 struct sdmmc\_osa\_event\_t

### 44.6.2.2 struct sdmmc\_osa\_mutex\_t

## 44.6.3 Function Documentation

### 44.6.3.1 status\_t **SDMMC\_OSAEventCreate** ( void \* *eventHandle* )

Parameters

<i>eventHandle</i>	event handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

### 44.6.3.2 status\_t **SDMMC\_OSAEventWait** ( void \* *eventHandle*, uint32\_t *eventType*, uint32\_t *timeoutMilliseconds*, uint32\_t \* *event* )

Parameters

<i>eventHandle</i>	The event type
<i>eventType</i>	Timeout time in milliseconds.
<i>timeout-Milliseconds</i>	timeout value in ms.
<i>event</i>	event flags.

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

#### 44.6.3.3 status\_t SDMMC\_OSAEventSet ( void \* *eventHandle*, uint32\_t *eventType* )

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	The event type

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

#### 44.6.3.4 status\_t SDMMC\_OSAEventGet ( void \* *eventHandle*, uint32\_t *eventType*, uint32\_t \* *flag* )

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	event type.
<i>flag</i>	pointer to store event value.

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

#### 44.6.3.5 status\_t SDMMC\_OSAEventClear ( void \* *eventHandle*, uint32\_t *eventType* )

Parameters

<i>eventHandle</i>	event handle.
<i>eventType</i>	The event type

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

#### 44.6.3.6 status\_t SDMMC\_OSAEventDestroy ( void \* *eventHandle* )

Parameters

<i>eventHandle</i>	The event handle.
--------------------	-------------------

#### 44.6.3.7 status\_t SDMMC\_OSAMutexCreate ( void \* *mutexHandle* )

Parameters

<i>mutexHandle</i>	mutex handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

#### 44.6.3.8 status\_t SDMMC\_OSAMutexLock ( void \* *mutexHandle*, uint32\_t *millisec* )

Parameters

<i>mutexHandle</i>	mutex handle.
<i>millisec</i>	The maximum number of milliseconds to wait for the mutex. If the mutex is locked, Pass the value osaWaitForever_c will wait indefinitely, pass 0 will return KOSA_StatusTimeout immediately.

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

#### 44.6.3.9 status\_t SDMMC\_OSAMutexUnlock ( void \* *mutexHandle* )

Parameters

<i>mutexHandle</i>	mutex handle.
--------------------	---------------

Return values

<i>kStatus_Fail</i>	or <i>kStatus_Success</i> .
---------------------	-----------------------------

#### 44.6.3.10 **status\_t SDMMC\_OSAMutexDestroy( void \* *mutexHandle* )**

Parameters

<i>mutexHandle</i>	The mutex handle.
--------------------	-------------------

#### 44.6.3.11 **void SDMMC\_OSADelay( uint32\_t *milliseconds* )**

Parameters

<i>milliseconds</i>	time to delay
---------------------	---------------

#### 44.6.3.12 **uint32\_t SDMMC\_OSADelayUs( uint32\_t *microseconds* )**

Parameters

<i>microseconds</i>	time to delay
---------------------	---------------

Returns

actual delayed microseconds

## 44.6.4 SDIF HOST adapter Driver

### 44.6.4.1 Overview

The SDIF host adapter driver provide adapter for blocking/non\_blocking mode.

### Data Structures

- struct `sdmmchost_t`  
*sdmmc host handler* [More...](#)

### Macros

- #define `FSL_SDMMC_HOST_ADAPTER_VERSION` (`MAKE_VERSION(2U, 4U, 0U)`) /\*2.4.-0\*/
   
*Middleware adapter version.*
- #define `SDMMCHOST_SUPPORT_HIGH_SPEED` (1U)
   
*host capability*
- #define `SDMMCHOST_INSTANCE_SUPPORT_8_BIT_WIDTH`(host) 1U
   
*sdmmc host instance capability*
- #define `SDMMCHOST_DMA_DESCRIPTOR_BUFFER_ALIGN_SIZE` (4U)
   
*SDMMC host dma descriptor buffer address align size.*
- #define `SDMMCHOST_RESET_TIMEOUT_VALUE` (1000000U)
   
*SDMMC host reset timoue value.*

### Typedefs

- typedef `sdif_transfer_t sdmmchost_transfer_t`
  
*sdmmc host transfer function*

## Enumerations

- enum {
   
kSDMMCHOST\_SupportHighSpeed = 1U << 0U,  
 kSDMMCHOST\_SupportSuspendResume = 1U << 1U,  
 kSDMMCHOST\_SupportVoltage3v3 = 1U << 2U,  
 kSDMMCHOST\_SupportVoltage3v0 = 1U << 3U,  
 kSDMMCHOST\_SupportVoltage1v8 = 1U << 4U,  
 kSDMMCHOST\_SupportVoltage1v2 = 1U << 5U,  
 kSDMMCHOST\_Support4BitDataWidth = 1U << 6U,  
 kSDMMCHOST\_Support8BitDataWidth = 1U << 7U,  
 kSDMMCHOST\_SupportDDRMode = 1U << 8U,  
 kSDMMCHOST\_SupportDetectCardByData3 = 1U << 9U,  
 kSDMMCHOST\_SupportDetectCardByCD = 1U << 10U,  
 kSDMMCHOST\_SupportAutoCmd12 = 1U << 11U,  
 kSDMMCHOST\_SupportSDR104 = 1U << 12U,  
 kSDMMCHOST\_SupportSDR50 = 1U << 13U,  
 kSDMMCHOST\_SupportHS200 = 1U << 14U,  
 kSDMMCHOST\_SupportHS400 = 1U << 15U }
   
*sdmmc host capability*
- enum \_sdmmchost\_endian\_mode {
   
kSDMMCHOST\_EndianModeBig = 0U,  
 kSDMMCHOST\_EndianModeHalfWordBig = 1U,  
 kSDMMCHOST\_EndianModeLittle = 2U }
   
*host Endian mode corresponding to driver define*

## SDIF host controller function

- void **SDMMCHOST\_SetCardBusWidth** (**sdmmchost\_t** \*host, **uint32\_t** dataBusWidth)
   
*set data bus width.*
- static void **SDMMCHOST\_SendCardActive** (**sdmmchost\_t** \*host)
   
*Send initialization active 80 clocks to card.*
- static **uint32\_t SDMMCHOST\_SetCardClock** (**sdmmchost\_t** \*host, **uint32\_t** targetClock)
   
*Set card bus clock.*
- static **bool SDMMCHOST\_IsCardBusy** (**sdmmchost\_t** \*host)
   
*check card status by DATA0.*
- static **status\_t SDMMCHOST\_StartBoot** (**sdmmchost\_t** \*host, **sdmmchost\_boot\_config\_t** \*hostConfig, **sdmmchost\_cmd\_t** \*cmd, **uint8\_t** \*buffer)
   
*start read boot data.*
- static **status\_t SDMMCHOST\_ReadBootData** (**sdmmchost\_t** \*host, **sdmmchost\_boot\_config\_t** \*hostConfig, **uint8\_t** \*buffer)
   
*read boot data.*
- static void **SDMMCHOST\_EnableBoot** (**sdmmchost\_t** \*host, **bool** enable)
   
*enable boot mode.*
- static void **SDMMCHOST\_EnableCardInt** (**sdmmchost\_t** \*host, **bool** enable)
   
*enable card interrupt.*
- **status\_t SDMMCHOST\_CardIntInit** (**sdmmchost\_t** \*host, **void** \*sdioInt)

- *card interrupt function.*
- **status\_t SDMMCHOST\_CardDetectInit** (*sdmmchost\_t* \*host, void \*cd)
  - card detect init function.*
- **status\_t SDMMCHOST\_PollingCardDetectStatus** (*sdmmchost\_t* \*host, *uint32\_t* waitCardStatus, *uint32\_t* timeout)
  - Detect card insert, only need for SD cases.*
- **uint32\_t SDMMCHOST\_CardDetectStatus** (*sdmmchost\_t* \*host)
  - card detect status.*
- **status\_t SDMMCHOST\_Init** (*sdmmchost\_t* \*host)
  - Init host controller.*
- **void SDMMCHOST\_Deinit** (*sdmmchost\_t* \*host)
  - Deinit host controller.*
- **void SDMMCHOST\_SetCardPower** (*sdmmchost\_t* \*host, bool enable)
  - host power off card function.*
- **status\_t SDMMCHOST\_TransferFunction** (*sdmmchost\_t* \*host, *sdmmchost\_transfer\_t* \*content)
  - host transfer function.*
- **void SDMMCHOST\_Reset** (*sdmmchost\_t* \*host)
  - host reset function.*
- **static void SDMMCHOST\_SwitchToVoltage** (*sdmmchost\_t* \*host, *uint32\_t* voltage)
  - switch to voltage.*
- **static status\_t SDMMCHOST\_ExecuteTuning** (*sdmmchost\_t* \*host, *uint32\_t* tuningCmd, *uint32\_t* \*revBuf, *uint32\_t* blockSize)
  - sdmmc host execute tuning.*
- **static void SDMMCHOST\_EnableDDRMode** (*sdmmchost\_t* \*host, bool enable, *uint32\_t* nibblePos)
  - enable DDR mode.*
- **static void SDMMCHOST\_EnableHS400Mode** (*sdmmchost\_t* \*host, bool enable)
  - enable HS400 mode.*
- **static void SDMMCHOST\_EnableStrobeDll** (*sdmmchost\_t* \*host, bool enable)
  - enable STROBE DLL.*
- **static uint32\_t SDMMCHOST\_GetSignalLineStatus** (*sdmmchost\_t* \*host, *uint32\_t* signalLine)
  - Get signal line status.*
- **static void SDMMCHOST\_ForceClockOn** (*sdmmchost\_t* \*host, bool enable)
  - force card clock on.*
- **void SDMMCHOST\_ConvertDataToLittleEndian** (*sdmmchost\_t* \*host, *uint32\_t* \*data, *uint32\_t* wordSize, *uint32\_t* format)
  - sdmmc host convert data sequence to little endian sequence*

#### 44.6.4.2 Data Structure Documentation

##### 44.6.4.2.1 struct sdmmchost\_t

###### Data Fields

- **sdif\_host\_t hostController**
  - host configuration*
- **uint8\_t hostPort**
  - host port number, used for one instance support two card*
- **void \*dmaDesBuffer**

- *DMA descriptor buffer address.*
- `uint32_t dmaDesBufferWordsNum`  
*DMA descriptor buffer size in byte.*
- `sdif_handle_t handle`  
*host controller handler*
- `uint32_t capability`  
*host controller capability*
- `uint32_t maxBlockCount`  
*host controller maximum block count*
- `uint32_t maxBlockSize`  
*host controller maximum block size*
- `sdmmc_osa_event_t hostEvent`  
*host event handler*
- `void *cd`  
*card detect*
- `void *cardInt`  
*call back function for card interrupt*
- `sdmmc_osa_mutex_t lock`  
*host access lock*

#### 44.6.4.3 Macro Definition Documentation

**44.6.4.3.1 #define FSL\_SDMMC\_HOST\_ADAPTER\_VERSION (MAKE\_VERSION(2U, 4U, 0U))  
/\*2.4.0\*/**

#### 44.6.4.4 Enumeration Type Documentation

##### 44.6.4.4.1 anonymous enum

Enumerator

*kSDMMCHOST\_SupportHighSpeed* high speed capability  
*kSDMMCHOST\_SupportSuspendResume* suspend resume capability  
*kSDMMCHOST\_SupportVoltage3v3* 3V3 capability  
*kSDMMCHOST\_SupportVoltage3v0* 3V0 capability  
*kSDMMCHOST\_SupportVoltage1v8* 1V8 capability  
*kSDMMCHOST\_SupportVoltage1v2* 1V2 capability  
*kSDMMCHOST\_Support4BitDataWidth* 4 bit data width capability  
*kSDMMCHOST\_Support8BitDataWidth* 8 bit data width capability  
*kSDMMCHOST\_SupportDDRMode* DDR mode capability.  
*kSDMMCHOST\_SupportDetectCardByData3* data3 detect card capability  
*kSDMMCHOST\_SupportDetectCardByCD* CD detect card capability.  
*kSDMMCHOST\_SupportAutoCmd12* auto command 12 capability  
*kSDMMCHOST\_SupportSDR104* SDR104 capability.  
*kSDMMCHOST\_SupportSDR50* SDR50 capability.  
*kSDMMCHOST\_SupportHS200* HS200 capability.  
*kSDMMCHOST\_SupportHS400* HS400 capability.

#### 44.6.4.4.2 enum \_sdmmchost\_endian\_mode

Enumerator

*kSDMMCHOST\_EndianModeBig* Big endian mode.

*kSDMMCHOST\_EndianModeHalfWordBig* Half word big endian mode.

*kSDMMCHOST\_EndianModeLittle* Little endian mode.

#### 44.6.4.5 Function Documentation

##### 44.6.4.5.1 void SDMMCHOST\_SetCardBusWidth ( *sdmmchost\_t \* host, uint32\_t dataBusWidth* )

Parameters

<i>host</i>	host handler
<i>dataBusWidth</i>	data bus width

##### 44.6.4.5.2 static void SDMMCHOST\_SendCardActive ( *sdmmchost\_t \* host* ) [inline], [static]

Parameters

<i>host</i>	host handler
-------------	--------------

##### 44.6.4.5.3 static uint32\_t SDMMCHOST\_SetCardClock ( *sdmmchost\_t \* host, uint32\_t targetClock* ) [inline], [static]

Parameters

<i>host</i>	host handler
<i>targetClock</i>	target clock frequency

Return values

<i>actual</i>	clock frequency can be reach.
---------------	-------------------------------

##### 44.6.4.5.4 static bool SDMMCHOST\_IsCardBusy ( *sdmmchost\_t \* host* ) [inline], [static]

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>true</i>	is busy, false is idle.
-------------	-------------------------

**44.6.4.5.5 static status\_t SDMMCHOST\_StartBoot ( *sdmmchost\_t \* host, sdmmchost\_boot\_config\_t \* hostConfig, sdmmchost\_cmd\_t \* cmd, uint8\_t \* buffer* ) [inline], [static]**

Parameters

<i>host</i>	host handler
<i>hostConfig</i>	boot configuration
<i>cmd</i>	boot command
<i>buffer</i>	buffer address

**44.6.4.5.6 static status\_t SDMMCHOST\_ReadBootData ( *sdmmchost\_t \* host, sdmmchost\_boot\_config\_t \* hostConfig, uint8\_t \* buffer* ) [inline], [static]**

Parameters

<i>host</i>	host handler
<i>hostConfig</i>	boot configuration
<i>buffer</i>	buffer address

**44.6.4.5.7 static void SDMMCHOST\_EnableBoot ( *sdmmchost\_t \* host, bool enable* ) [inline], [static]**

Parameters

<i>host</i>	host handler
-------------	--------------

<i>enable</i>	true is enable, false is disable
---------------	----------------------------------

**44.6.4.5.8 static void SDMMCHOST\_EnableCardInt ( *sdmmchost\_t* \* *host*, *bool enable* )  
[inline], [static]**

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

**44.6.4.5.9 status\_t SDMMCHOST\_CardIntInit ( *sdmmchost\_t* \* *host*, *void* \* *sdioInt* )**

Parameters

<i>host</i>	host handler
<i>sdioInt</i>	card interrupt configuration

**44.6.4.5.10 status\_t SDMMCHOST\_CardDetectInit ( *sdmmchost\_t* \* *host*, *void* \* *cd* )**

Parameters

<i>host</i>	host handler
<i>cd</i>	card detect configuration

**44.6.4.5.11 status\_t SDMMCHOST\_PollingCardDetectStatus ( *sdmmchost\_t* \* *host*, *uint32\_t*  
*waitCardStatus*, *uint32\_t* *timeout* )**

Parameters

<i>host</i>	host handler
<i>waitCardStatus</i>	status which user want to wait
<i>timeout</i>	wait time out.

Return values

<i>kStatus_Success</i>	detect card insert
<i>kStatus_Fail</i>	card insert event fail

#### 44.6.4.5.12 uint32\_t SDMMCHOST\_CardDetectStatus ( *sdmmchost\_t \* host* )

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>kSD_Inserted</i> , <i>kSD_Removed</i>	
--	--

#### 44.6.4.5.13 status\_t SDMMCHOST\_Init ( *sdmmchost\_t \* host* )

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDMMCHOST_Deinit(host);
* SDMMCHOST_Init(host);
*
```

Parameters

<i>host</i>	host handler
-------------	--------------

Return values

<i>kStatus_Success</i>	host init success
<i>kStatus_Fail</i>	event fail

#### 44.6.4.5.14 void SDMMCHOST\_Deinit ( *sdmmchost\_t \* host* )

Please note it is a thread safe function.

Parameters

<i>host</i>	host handler
-------------	--------------

#### 44.6.4.5.15 void SDMMCHOST\_SetCardPower ( *sdmmchost\_t \* host, bool enable* )

Parameters

<i>host</i>	host handler
<i>enable</i>	true is power on, false is power down.

#### 44.6.4.5.16 status\_t SDMMCHOST\_TransferFunction ( *sdmmchost\_t \* host, sdmmchost\_transfer\_t \* content* )

Please note it is a thread safe function.

Parameters

<i>host</i>	host handler
<i>content</i>	transfer content.

#### 44.6.4.5.17 void SDMMCHOST\_Reset ( *sdmmchost\_t \* host* )

Please note it is a thread safe function.

Parameters

<i>host</i>	host handler
-------------	--------------

#### 44.6.4.5.18 static void SDMMCHOST\_SwitchToVoltage ( *sdmmchost\_t \* host, uint32\_t voltage* ) [inline], [static]

Parameters

<i>host</i>	host handler
-------------	--------------

<i>voltage</i>	switch to voltage level.
----------------	--------------------------

**44.6.4.5.19 static status\_t SDMMCHOST\_ExecuteTuning ( *sdmmchost\_t \* host, uint32\_t tuningCmd, uint32\_t \* revBuf, uint32\_t blockSize* ) [inline], [static]**

Parameters

<i>host</i>	host handler
<i>tuningCmd</i>	tuning command.
<i>revBuf</i>	receive buffer pointer
<i>blockSize</i>	tuning data block size.

**44.6.4.5.20 static void SDMMCHOST\_EnableDDRMode ( *sdmmchost\_t \* host, bool enable, uint32\_t nibblePos* ) [inline], [static]**

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.
<i>nibblePos</i>	nibble position indication. 0- the sequence is 'odd high nibble -> even high nibble -> odd low nibble -> even low nibble'; 1- the sequence is 'odd high nibble -> odd low nibble -> even high nibble -> even low nibble'.

**44.6.4.5.21 static void SDMMCHOST\_EnableHS400Mode ( *sdmmchost\_t \* host, bool enable* ) [inline], [static]**

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

**44.6.4.5.22 static void SDMMCHOST\_EnableStrobeDII ( *sdmmchost\_t \* host, bool enable* ) [inline], [static]**

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

**44.6.4.5.23 static uint32\_t SDMMCHOST\_GetSignalLineStatus ( *sdmmchost\_t \* host, uint32\_t signalLine* ) [inline], [static]**

Parameters

<i>host</i>	host handler
<i>signalLine</i>	signal line type, reference _sdmmc_signal_line

**44.6.4.5.24 static void SDMMCHOST\_ForceClockOn ( *sdmmchost\_t \* host, bool enable* ) [inline], [static]**

Parameters

<i>host</i>	host handler
<i>enable</i>	true is enable, false is disable.

**44.6.4.5.25 void SDMMCHOST\_ConvertDataToLittleEndian ( *sdmmchost\_t \* host, uint32\_t \* data, uint32\_t wordSize, uint32\_t format* )**

Parameters

<i>host</i>	host handler.
<i>data</i>	data buffer address.
<i>wordSize</i>	data buffer size in word.
<i>format</i>	data packet format.

## 44.7 SDMMC Common

### 44.7.1 Overview

The sdmmc common function and definition.

## Data Structures

- struct `sd_detect_card_t`  
`sd card detect` [More...](#)
- struct `sd_io_voltage_t`  
`io voltage control configuration` [More...](#)
- struct `sd_usr_param_t`  
`sdcard user parameter` [More...](#)
- struct `sdio_card_int_t`  
`card interrupt application callback` [More...](#)
- struct `sdio_usr_param_t`  
`sdio user parameter` [More...](#)
- struct `sdio_fbr_t`  
`sdio card FBR register` [More...](#)
- struct `sdio_common_cis_t`  
`sdio card common CIS` [More...](#)
- struct `sdio_func_cis_t`  
`sdio card function CIS` [More...](#)
- struct `sd_status_t`  
`SD card status.` [More...](#)
- struct `sd_cid_t`  
`SD card CID register.` [More...](#)
- struct `sd_csd_t`  
`SD card CSD register.` [More...](#)
- struct `sd_scr_t`  
`SD card SCR register.` [More...](#)
- struct `mmc_cid_t`  
`MMC card CID register.` [More...](#)
- struct `mmc_csd_t`  
`MMC card CSD register.` [More...](#)
- struct `mmc_extended_csd_t`  
`MMC card Extended CSD register (unit: byte).` [More...](#)
- struct `mmc_extended_csd_config_t`  
`MMC Extended CSD configuration.` [More...](#)
- struct `mmc_boot_config_t`  
`MMC card boot configuration definition.` [More...](#)

## Macros

- #define `SWAP_WORD_BYTSEQUENCE(x)` (`__REV(x)`)  
`Reverse byte sequence in uint32_t.`
- #define `SWAP_HALF_WROD_BYTSEQUENCE(x)` (`__REV16(x)`)

- Reverse byte sequence for each half word in `uint32_t`.
- `#define FSL_SDMMC_MAX_VOLTAGE_RETRIES` (1000U)  
*Maximum loop count to check the card operation voltage range.*
- `#define FSL_SDMMC_MAX_CMD_RETRIES` (10U)  
*Maximum loop count to send the cmd.*
- `#define FSL_SDMMC_DEFAULT_BLOCK_SIZE` (512U)  
*Default block size.*
- `#define SDMMC_DATA_BUFFER_ALIGN_CACHE` `sizeof(uint32_t)`  
*make sure the internal buffer address is cache align*
- `#define FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE` (`FSL_SDMMC_DEFAULT_BLOCK_SIZE + SDMMC_DATA_BUFFER_ALIGN_CACHE`)  
*sdmmc card internal buffer size*
- `#define FSL_SDMMC_CARD_MAX_BUS_FREQ`(max, target) ((max) == 0U ? (target) : ((max) > (target) ? (target) : (max)))  
*get maximum freq*
- `#define SDMMC_LOG`(format,...)  
*SD/MMC error log.*
- `#define SDMMC_CLOCK_400KHZ` (400000U)  
*SD/MMC card initialization clock frequency.*
- `#define SD_CLOCK_25MHZ` (25000000U)  
*SD card bus frequency 1 in high-speed mode.*
- `#define SD_CLOCK_50MHZ` (50000000U)  
*SD card bus frequency 2 in high-speed mode.*
- `#define SD_CLOCK_100MHZ` (100000000U)  
*SD card bus frequency in SDR50 mode.*
- `#define SD_CLOCK_208MHZ` (208000000U)  
*SD card bus frequency in SDR104 mode.*
- `#define MMC_CLOCK_26MHZ` (26000000U)  
*MMC card bus frequency 1 in high-speed mode.*
- `#define MMC_CLOCK_52MHZ` (52000000U)  
*MMC card bus frequency 2 in high-speed mode.*
- `#define MMC_CLOCK_DDR52` (52000000U)  
*MMC card bus frequency in high-speed DDR52 mode.*
- `#define MMC_CLOCK_HS200` (200000000U)  
*MMC card bus frequency in high-speed HS200 mode.*
- `#define MMC_CLOCK_HS400` (400000000U)  
*MMC card bus frequency in high-speed HS400 mode.*
- `#define SDMMC_MASK`(bit) (`1UL << (bit)`)  
*mask convert*
- `#define SDMMC_R1_ALL_ERROR_FLAG`  
*R1 all the error flag.*
- `#define SDMMC_R1_CURRENT_STATE`(x) (((x)&0x00001E00U) >> 9U)  
*R1: current state.*
- `#define SDSPI_R7_VERSION_SHIFT` (28U)  
*The bit mask for COMMAND VERSION field in R7.*
- `#define SDSPI_R7_VERSION_MASK` (0xFU)  
*The bit mask for COMMAND VERSION field in R7.*
- `#define SDSPI_R7_VOLTAGE_SHIFT` (8U)  
*The bit shift for VOLTAGE ACCEPTED field in R7.*
- `#define SDSPI_R7_VOLTAGE_MASK` (0xFU)  
*The bit mask for VOLTAGE ACCEPTED field in R7.*

- #define **SDSPI\_R7\_VOLTAGE\_27\_36\_MASK** (0x1U << SDSPI\_R7\_VOLTAGE\_SHIFT)
 

*The bit mask for VOLTAGE 2.7V to 3.6V field in R7.*
- #define **SDSPI\_R7\_ECHO\_SHIFT** (0U)
 

*The bit shift for ECHO field in R7.*
- #define **SDSPI\_R7\_ECHO\_MASK** (0xFFU)
 

*The bit mask for ECHO field in R7.*
- #define **SDSPI\_DATA\_ERROR\_TOKEN\_MASK** (0xFU)
 

*Data error token mask.*
- #define **SDSPI\_DATA\_RESPONSE\_TOKEN\_MASK** (0x1FU)
 

*Mask for data response bits.*
- #define **SDIO\_CCCR\_REG\_NUMBER** (0x16U)
 

*sdio card cccr register number*
- #define **SDIO\_IO\_READY\_TIMEOUT\_UNIT** (10U)
 

*sdio IO ready timeout steps*
- #define **SDIO\_CMD\_ARGUMENT\_RW\_POS** (31U)
 

*read/write flag position*
- #define **SDIO\_CMD\_ARGUMENT\_FUNC\_NUM\_POS** (28U)
 

*function number position*
- #define **SDIO\_DIRECT\_CMD\_ARGUMENT\_RAW\_POS** (27U)
 

*direct raw flag position*
- #define **SDIO\_CMD\_ARGUMENT\_REG\_ADDR\_POS** (9U)
 

*direct reg addr position*
- #define **SDIO\_CMD\_ARGUMENT\_REG\_ADDR\_MASK** (0x1FFFFU)
 

*direct reg addr mask*
- #define **SDIO\_DIRECT\_CMD\_DATA\_MASK** (0xFFU)
 

*data mask*
- #define **SDIO\_EXTEND\_CMD\_ARGUMENT\_BLOCK\_MODE\_POS** (27U)
 

*extended command argument block mode bit position*
- #define **SDIO\_EXTEND\_CMD\_ARGUMENT\_OP\_CODE\_POS** (26U)
 

*extended command argument OP Code bit position*
- #define **SDIO\_EXTEND\_CMD\_BLOCK\_MODE\_MASK** (0x08000000U)
 

*block mode mask*
- #define **SDIO\_EXTEND\_CMD\_OP\_CODE\_MASK** (0x04000000U)
 

*op code mask*
- #define **SDIO\_EXTEND\_CMD\_COUNT\_MASK** (0x1FFU)
 

*byte/block count mask*
- #define **SDIO\_MAX\_BLOCK\_SIZE** (2048U)
 

*max block size*
- #define **SDIO\_FBR\_BASE**(x) ((x)\*0x100U)
 

*function basic register*
- #define **SDIO\_TPL\_CODE\_END** (0xFFU)
 

*tuple end*
- #define **SDIO\_TPL\_CODE\_MANIFID** (0x20U)
 

*manufacturer ID*
- #define **SDIO\_TPL\_CODE\_FUNCID** (0x21U)
 

*function ID*
- #define **SDIO\_TPL\_CODE\_FUNCE** (0x22U)
 

*function extension tuple*
- #define **SDIO\_OCR\_VOLTAGE\_WINDOW\_MASK** (0xFFFFU << 8U)
 

*sdio ocr voltage window mask*
- #define **SDIO\_OCR\_IO\_NUM\_MASK** (7U << kSDIO\_OcrIONumber)

- `sdio ocr reigster IO NUMBER mask`
- `#define SDIO_CCCR_SUPPORT_HIGHSPEED (1UL << 9U)`  
*UHS timing mode flag.*
- `#define SDIO_CCCR_DRIVER_TYPE_MASK (3U << 4U)`  
*Driver type flag.*
- `#define SDIO_CCCR_ASYNC_INT_MASK (1U)`  
*async interrupt flag*
- `#define SDIO_CCCR_SUPPORT_8BIT_BUS (1UL << 18U)`  
*8 bit data bus flag*
- `#define MMC_OCR_V170TO195_SHIFT (7U)`  
*The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.*
- `#define MMC_OCR_V170TO195_MASK (0x00000080U)`  
*The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.*
- `#define MMC_OCR_V200TO260_SHIFT (8U)`  
*The bit shift for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.*
- `#define MMC_OCR_V200TO260_MASK (0x00007F00U)`  
*The bit mask for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.*
- `#define MMC_OCR_V270TO360_SHIFT (15U)`  
*The bit shift for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.*
- `#define MMC_OCR_V270TO360_MASK (0x00FF8000U)`  
*The bit mask for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.*
- `#define MMC_OCR_ACCESS_MODE_SHIFT (29U)`  
*The bit shift for ACCESS MODE field in OCR.*
- `#define MMC_OCR_ACCESS_MODE_MASK (0x60000000U)`  
*The bit mask for ACCESS MODE field in OCR.*
- `#define MMC_OCR_BUSY_SHIFT (31U)`  
*The bit shift for BUSY field in OCR.*
- `#define MMC_OCR_BUSY_MASK (1U << MMC_OCR_BUSY_SHIFT)`  
*The bit mask for BUSY field in OCR.*
- `#define MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT (0U)`  
*The bit shift for FREQUENCY UNIT field in TRANSFER SPEED(TRAN-SPEED in Extended CSD)*
- `#define MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK (0x07U)`  
*The bit mask for FRQEUNCY UNIT in TRANSFER SPEED.*
- `#define MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT (3U)`  
*The bit shift for MULTIPLIER field in TRANSFER SPEED.*
- `#define MMC_TRANSFER_SPEED_MULTIPLIER_MASK (0x78U)`  
*The bit mask for MULTIPLIER field in TRANSFER SPEED.*
- `#define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(CSD) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)`  
*Read the value of FREQUENCY UNIT in TRANSFER SPEED.*
- `#define READ_MMC_TRANSFER_SPEED_MULTIPLIER(CSD) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)`  
*Read the value of MULTIPLIER filed in TRANSFER SPEED.*
- `#define MMC_POWER_CLASS_4BIT_MASK (0x0FU)`  
*The power class value bit mask when bus in 4 bit mode.*
- `#define MMC_POWER_CLASS_8BIT_MASK (0xF0U)`  
*The power class current value bit mask when bus in 8 bit mode.*
- `#define MMC_CACHE_CONTROL_ENABLE (1U)`  
*mmc cache control enable*

- #define **MMC\_CACHE\_TRIGGER\_FLUSH** (1U)  
*mmc cache flush*
- #define **MMC\_DATA\_BUS\_WIDTH\_TYPE\_NUMBER** (3U)  
*The number of data bus width type.*
- #define **MMC\_PARTITION\_CONFIG\_PARTITION\_ACCESS\_SHIFT** (0U)  
*The bit shift for PARTITION ACCESS filed in BOOT CONFIG (BOOT\_CONFIG in Extend CSD)*
- #define **MMC\_PARTITION\_CONFIG\_PARTITION\_ACCESS\_MASK** (0x00000007U)  
*The bit mask for PARTITION ACCESS field in BOOT CONFIG.*
- #define **MMC\_PARTITION\_CONFIG\_PARTITION\_ENABLE\_SHIFT** (3U)  
*The bit shift for PARTITION ENABLE field in BOOT CONFIG.*
- #define **MMC\_PARTITION\_CONFIG\_PARTITION\_ENABLE\_MASK** (0x00000038U)  
*The bit mask for PARTITION ENABLE field in BOOT CONFIG.*
- #define **MMC\_PARTITION\_CONFIG\_BOOT\_ACK\_SHIFT** (6U)  
*The bit shift for ACK field in BOOT CONFIG.*
- #define **MMC\_PARTITION\_CONFIG\_BOOT\_ACK\_MASK** (0x00000040U)  
*The bit mask for ACK field in BOOT CONFIG.*
- #define **MMC\_BOOT\_BUS\_CONDITION\_BUS\_WIDTH\_SHIFT** (0U)  
*The bit shift for BOOT BUS WIDTH field in BOOT CONFIG.*
- #define **MMC\_BOOT\_BUS\_CONDITION\_BUS\_WIDTH\_MASK** (3U)  
*The bit mask for BOOT BUS WIDTH field in BOOT CONFIG.*
- #define **MMC\_BOOT\_BUS\_CONDITION\_RESET\_BUS\_CONDITION\_SHIFT** (2U)  
*The bit shift for BOOT BUS WIDTH RESET field in BOOT CONFIG.*
- #define **MMC\_BOOT\_BUS\_CONDITION\_RESET\_BUS\_CONDITION\_MASK** (4U)  
*The bit mask for BOOT BUS WIDTH RESET field in BOOT CONFIG.*
- #define **MMC\_BOOT\_BUS\_CONDITION\_BOOT\_MODE\_SHIFT** (3U)  
*The bit shift for BOOT MODE field in BOOT CONFIG.*
- #define **MMC\_BOOT\_BUS\_CONDITION\_BOOT\_MODE\_MASK** (0x18U)  
*The bit mask for BOOT MODE field in BOOT CONFIG.*
- #define **MMC\_EXTENDED\_CSD\_BYTES** (512U)  
*The length of Extended CSD register, unit as bytes.*
- #define **MMC\_DEFAULT\_RELATIVE\_ADDRESS** (2UL)  
*MMC card default relative address.*
- #define **SD\_PRODUCT\_NAME\_BYTES** (5U)  
*SD card product name length united as bytes.*
- #define **SD\_AU\_START\_VALUE** (1U)  
*SD AU start value.*
- #define **SD\_UHS\_AU\_START\_VALUE** (7U)  
*SD UHS AU start value.*
- #define **SD\_TRANSFER\_SPEED\_RATE\_UNIT\_SHIFT** (0U)  
*The bit shift for RATE UNIT field in TRANSFER SPEED.*
- #define **SD\_TRANSFER\_SPEED\_RATE\_UNIT\_MASK** (0x07U)  
*The bit mask for RATE UNIT field in TRANSFER SPEED.*
- #define **SD\_TRANSFER\_SPEED\_TIME\_VALUE\_SHIFT** (2U)  
*The bit shift for TIME VALUE field in TRANSFER SPEED.*
- #define **SD\_TRANSFER\_SPEED\_TIME\_VALUE\_MASK** (0x78U)  
*The bit mask for TIME VALUE field in TRANSFER SPEED.*
- #define **SD\_RD\_TRANSFER\_SPEED\_RATE\_UNIT(x)** (((x.transferSpeed) & **SD\_TRANSFER\_SPEED\_RATE\_UNIT\_MASK**) >> **SD\_TRANSFER\_SPEED\_RATE\_UNIT\_SHIFT**)  
*Read the value of FREQUENCY UNIT in TRANSFER SPEED field.*
- #define **SD\_RD\_TRANSFER\_SPEED\_TIME\_VALUE(x)** (((x.transferSpeed) & **SD\_TRANSFER\_SPEED\_TIME\_VALUE\_MASK**) >> **SD\_TRANSFER\_SPEED\_TIME\_VALUE\_SHIFT**)

- *Read the value of TIME VALUE in TRANSFER SPEED field.*  
• #define **MMC\_PRODUCT\_NAME\_BYTES** (6U)  
*MMC card product name length united as bytes.*
- #define **MMC\_SWITCH\_COMMAND\_SET\_SHIFT** (0U)  
*The bit shift for COMMAND SET field in SWITCH command.*
- #define **MMC\_SWITCH\_COMMAND\_SET\_MASK** (0x00000007U)  
*The bit mask for COMMAND set field in SWITCH command.*
- #define **MMC\_SWITCH\_VALUE\_SHIFT** (8U)  
*The bit shift for VALUE field in SWITCH command.*
- #define **MMC\_SWITCH\_VALUE\_MASK** (0x0000FF00U)  
*The bit mask for VALUE field in SWITCH command.*
- #define **MMC\_SWITCH\_BYTE\_INDEX\_SHIFT** (16U)  
*The bit shift for BYTE INDEX field in SWITCH command.*
- #define **MMC\_SWITCH\_BYTE\_INDEX\_MASK** (0x00FF0000U)  
*The bit mask for BYTE INDEX field in SWITCH command.*
- #define **MMC\_SWITCH\_ACCESS\_MODE\_SHIFT** (24U)  
*The bit shift for ACCESS MODE field in SWITCH command.*
- #define **MMC\_SWITCH\_ACCESS\_MODE\_MASK** (0x03000000U)  
*The bit mask for ACCESS MODE field in SWITCH command.*

## Typedefs

- typedef void(\* **sd\_cd\_t** )(bool isInserted, void \*userData)  
*card detect application callback definition*
- typedef bool(\* **sd\_cd\_status\_t** )(void)  
*card detect status*
- typedef void(\* **sd\_io\_voltage\_func\_t** )(sdmmc\_operation\_voltage\_t voltage)  
*card switch voltage function pointer*
- typedef void(\* **sd\_pwr\_t** )(bool enable)  
*card power control function pointer*
- typedef void(\* **sd\_io\_strength\_t** )(uint32\_t busFreq)  
*card io strength control*
- typedef void(\* **sdio\_int\_t** )(void \*userData)  
*card interrupt function pointer*

## Enumerations

- enum {
   
kStatus\_SDMMC\_NotSupportYet = MAKE\_STATUS(kStatusGroup\_SDMMC, 0U),
   
kStatus\_SDMMC\_TransferFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 1U),
   
kStatus\_SDMMC\_SetCardBlockSizeFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 2U),
   
kStatus\_SDMMC\_HostNotSupport = MAKE\_STATUS(kStatusGroup\_SDMMC, 3U),
   
kStatus\_SDMMC\_CardNotSupport = MAKE\_STATUS(kStatusGroup\_SDMMC, 4U),
   
kStatus\_SDMMC\_AllSendCidFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 5U),
   
kStatus\_SDMMC\_SendRelativeAddressFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 6U),
   
kStatus\_SDMMC\_SendCsdFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 7U),
   
kStatus\_SDMMC\_SelectCardFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 8U),
   
kStatus\_SDMMC\_SendScrFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 9U),
   
kStatus\_SDMMC\_SetDataBusWidthFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 10U),
   
kStatus\_SDMMC\_GoIdleFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 11U),
   
kStatus\_SDMMC\_HandShakeOperationConditionFailed,
   
kStatus\_SDMMC\_SendApplicationCommandFailed,
   
kStatus\_SDMMC\_SwitchFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 14U),
   
kStatus\_SDMMC\_StopTransmissionFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 15U),
   
kStatus\_SDMMC\_WaitWriteCompleteFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 16U),
   
kStatus\_SDMMC\_SetBlockCountFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 17U),
   
kStatus\_SDMMC\_SetRelativeAddressFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 18U),
   
kStatus\_SDMMC\_SwitchBusTimingFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 19U),
   
kStatus\_SDMMC\_SendExtendedCsdFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 20U),
   
kStatus\_SDMMC\_ConfigureBootFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 21U),
   
kStatus\_SDMMC\_ConfigureExtendedCsdFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 22-U),
   
kStatus\_SDMMC\_EnableHighCapacityEraseFailed,
   
kStatus\_SDMMC\_SendTestPatternFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 24U),
   
kStatus\_SDMMC\_ReceiveTestPatternFailed = MAKE\_STATUS(kStatusGroup\_SDMMC, 25U),
   
kStatus\_SDMMC\_SDIO\_ResponseError = MAKE\_STATUS(kStatusGroup\_SDMMC, 26U),
   
kStatus\_SDMMC\_SDIO\_InvalidArgument,
   
kStatus\_SDMMC\_SDIO\_SendOperationConditionFail,
   
kStatus\_SDMMC\_InvalidVoltage = MAKE\_STATUS(kStatusGroup\_SDMMC, 29U),
   
kStatus\_SDMMC\_SDIO\_SwitchHighSpeedFail = MAKE\_STATUS(kStatusGroup\_SDMMC, 30-U),
   
kStatus\_SDMMC\_SDIO\_ReadCISFail = MAKE\_STATUS(kStatusGroup\_SDMMC, 31U),
   
kStatus\_SDMMC\_SDIO\_InvalidCard = MAKE\_STATUS(kStatusGroup\_SDMMC, 32U),
   
kStatus\_SDMMC\_TuningFail = MAKE\_STATUS(kStatusGroup\_SDMMC, 33U),
   
kStatus\_SDMMC\_SwitchVoltageFail = MAKE\_STATUS(kStatusGroup\_SDMMC, 34U),
   
kStatus\_SDMMC\_SwitchVoltage18VFail33VSuccess = MAKE\_STATUS(kStatusGroup\_SDMM-

C, 35U),

```
kStatus_SDMMC_ReTuningRequest = MAKE_STATUS(kStatusGroup_SDMMC, 36U),
kStatus_SDMMC_SetDriverStrengthFail = MAKE_STATUS(kStatusGroup_SDMMC, 37U),
kStatus_SDMMC_SetPowerClassFail = MAKE_STATUS(kStatusGroup_SDMMC, 38U),
kStatus_SDMMC_HostNotReady = MAKE_STATUS(kStatusGroup_SDMMC, 39U),
kStatus_SDMMC_CardDetectFailed = MAKE_STATUS(kStatusGroup_SDMMC, 40U),
kStatus_SDMMC_AuSizeNotSetProperly = MAKE_STATUS(kStatusGroup_SDMMC, 41U),
kStatus_SDMMC_PollingCardIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 42U),
kStatus_SDMMC_DeselectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 43U),
kStatus_SDMMC_CardStatusIdle = MAKE_STATUS(kStatusGroup_SDMMC, 44U),
kStatus_SDMMC_CardStatusBusy = MAKE_STATUS(kStatusGroup_SDMMC, 45U),
kStatus_SDMMC_CardInitFailed = MAKE_STATUS(kStatusGroup_SDMMC, 46U) }
```

*SD/MMC card API's running status.*

- enum {
   
kSDMMC\_SignalLineCmd = 1U,
   
kSDMMC\_SignalLineData0 = 2U,
   
kSDMMC\_SignalLineData1 = 4U,
   
kSDMMC\_SignalLineData2 = 8U,
   
kSDMMC\_SignalLineData3 = 16U,
   
kSDMMC\_SignalLineData4 = 32U,
   
kSDMMC\_SignalLineData5 = 64U,
   
kSDMMC\_SignalLineData6 = 128U,
   
kSDMMC\_SignalLineData7 = 256U }

*sdmmc signal line*

- enum **sdmmc\_operation\_voltage\_t** {
   
kSDMMC\_OperationVoltageNone = 0U,
   
kSDMMC\_OperationVoltage330V = 1U,
   
kSDMMC\_OperationVoltage300V = 2U,
   
kSDMMC\_OperationVoltage180V = 3U }

*card operation voltage*

- enum {
   
kSDMMC\_BusWidth1Bit = 0U,
   
kSDMMC\_BusWidth4Bit = 1U,
   
kSDMMC\_BusWidth8Bit = 2U }

*card bus width*

- enum { **kSDMMC\_Support8BitWidth** = 1U }

*sdmmc capability flag*

- enum {
   
kSDMMC\_DataPacketFormatLSBFirst,
   
kSDMMC\_DataPacketFormatMSBFirst }

*@ brief sdmmc data packet format*

- enum **sd\_detect\_card\_type\_t** {
   
kSD\_DetectCardByGpioCD,
   
kSD\_DetectCardByHostCD,
   
kSD\_DetectCardByHostDATA3 }

*sd card detect type*

- enum {
   
  kSD\_Inserted = 1U,
   
  kSD\_Removed = 0U }
   
    *@ brief SD card detect status*
- enum {
   
  kSD\_DAT3PullDown = 0U,
   
  kSD\_DAT3PullUp = 1U }
   
    *@ brief SD card detect status*
- enum **sd\_io\_voltage\_ctrl\_type\_t** {
   
  kSD\_IOVoltageCtrlNotSupport = 0U,
   
  kSD\_IOVoltageCtrlByGpio = 2U }
   
    *io voltage control type*
- enum {
   
  kSDMMC\_R1OutOfRangeFlag = 31,
   
  kSDMMC\_R1AddressErrorFlag = 30,
   
  kSDMMC\_R1BlockLengthErrorFlag = 29,
   
  kSDMMC\_R1EraseSequenceErrorFlag = 28,
   
  kSDMMC\_R1EraseParameterErrorFlag = 27,
   
  kSDMMC\_R1WriteProtectViolationFlag = 26,
   
  kSDMMC\_R1CardIsLockedFlag = 25,
   
  kSDMMC\_R1LockUnlockFailedFlag = 24,
   
  kSDMMC\_R1CommandCrcErrorFlag = 23,
   
  kSDMMC\_R1IllegalCommandFlag = 22,
   
  kSDMMC\_R1CardEccFailedFlag = 21,
   
  kSDMMC\_R1CardControllerErrorFlag = 20,
   
  kSDMMC\_R1ErrorFlag = 19,
   
  kSDMMC\_R1CidCsdOverwriteFlag = 16,
   
  kSDMMC\_R1WriteProtectEraseSkipFlag = 15,
   
  kSDMMC\_R1CardEccDisabledFlag = 14,
   
  kSDMMC\_R1EraseResetFlag = 13,
   
  kSDMMC\_R1ReadyForDataFlag = 8,
   
  kSDMMC\_R1SwitchErrorFlag = 7,
   
  kSDMMC\_R1ApplicationCommandFlag = 5,
   
  kSDMMC\_R1AuthenticationSequenceErrorFlag = 3 }
   
    *Card status bit in R1.*
- enum **sdmmc\_r1\_current\_state\_t** {
   
  kSDMMC\_R1StateIdle = 0U,
   
  kSDMMC\_R1StateReady = 1U,
   
  kSDMMC\_R1StateIdentify = 2U,
   
  kSDMMC\_R1StateStandby = 3U,
   
  kSDMMC\_R1StateTransfer = 4U,
   
  kSDMMC\_R1StateSendData = 5U,
   
  kSDMMC\_R1StateReceiveData = 6U,
   
  kSDMMC\_R1StateProgram = 7U,
   
  kSDMMC\_R1StateDisconnect = 8U }
   
    *CURRENT\_STATE filed in R1.*

- enum {
   
  kSDSPI\_R1InIdleStateFlag = (1U << 0U),
   
  kSDSPI\_R1EraseResetFlag = (1U << 1U),
   
  kSDSPI\_R1IllegalCommandFlag = (1U << 2U),
   
  kSDSPI\_R1CommandCrcErrorFlag = (1U << 3U),
   
  kSDSPI\_R1EraseSequenceErrorFlag = (1U << 4U),
   
  kSDSPI\_R1AddressErrorFlag = (1U << 5U),
   
  kSDSPI\_R1ParameterErrorFlag = (1U << 6U) }

*Error bit in SPI mode R1.*

- enum {
   
  kSDSPI\_R2CardLockedFlag = (1U << 0U),
   
  kSDSPI\_R2WriteProtectEraseSkip = (1U << 1U),
   
  kSDSPI\_R2LockUnlockFailed = (1U << 1U),
   
  kSDSPI\_R2ErrorFlag = (1U << 2U),
   
  kSDSPI\_R2CardControllerErrorFlag = (1U << 3U),
   
  kSDSPI\_R2CardEccFailedFlag = (1U << 4U),
   
  kSDSPI\_R2WriteProtectViolationFlag = (1U << 5U),
   
  kSDSPI\_R2EraseParameterErrorFlag = (1U << 6U),
   
  kSDSPI\_R2OutOfRangeFlag = (1U << 7U),
   
  kSDSPI\_R2CsdOverwriteFlag = (1U << 7U) }

*Error bit in SPI mode R2.*

- enum {
   
  kSDSPI\_DataErrorTokenError = (1U << 0U),
   
  kSDSPI\_DataErrorTokenCardControllerError = (1U << 1U),
   
  kSDSPI\_DataErrorTokenCardEccFailed = (1U << 2U),
   
  kSDSPI\_DataErrorTokenOutOfRange = (1U << 3U) }

*Data Error Token mask bit.*

- enum **sdspi\_data\_token\_t** {
   
  kSDSPI\_DataTokenBlockRead = 0xFEU,
   
  kSDSPI\_DataTokenSingleBlockWrite = 0xFEU,
   
  kSDSPI\_DataTokenMultipleBlockWrite = 0xFCU,
   
  kSDSPI\_DataTokenStopTransfer = 0xFDU }

*Data Token.*

- enum **sdspi\_data\_response\_token\_t** {
   
  kSDSPI\_DataResponseTokenAccepted = 0x05U,
   
  kSDSPI\_DataResponseTokenCrcError = 0x0BU,
   
  kSDSPI\_DataResponseTokenWriteError = 0x0DU }

*Data Response Token.*

- enum **sd\_command\_t** {
   
  kSD\_SendRelativeAddress = 3U,
   
  kSD\_Switch = 6U,
   
  kSD\_SendInterfaceCondition = 8U,
   
  kSD\_VoltageSwitch = 11U,
   
  kSD\_SpeedClassControl = 20U,
   
  kSD\_EraseWriteBlockStart = 32U,
   
  kSD\_EraseWriteBlockEnd = 33U,

- ```
kSD_SendTuningBlock = 19U }
    SD card individual commands.
```
- enum `sdspi_command_t` { `kSDSPI_CommandCrc` = 59U }
 SDSPI individual commands.
  - enum `sd_application_command_t` {
 `kSD_ApplicationSetBusWdith` = 6U,
 `kSD_ApplicationStatus` = 13U,
 `kSD_ApplicationSendNumberWriteBlocks` = 22U,
 `kSD_ApplicationSetWriteBlockEraseCount` = 23U,
 `kSD_ApplicationSendOperationCondition` = 41U,
 `kSD_ApplicationSetClearCardDetect` = 42U,
 `kSD_ApplicationSendScr` = 51U }

 SD card individual application commands.
  - enum {
 `kSDMMC_CommandClassBasic` = (1U << 0U),
 `kSDMMC_CommandClassBlockRead` = (1U << 2U),
 `kSDMMC_CommandClassBlockWrite` = (1U << 4U),
 `kSDMMC_CommandClassErase` = (1U << 5U),
 `kSDMMC_CommandClassWriteProtect` = (1U << 6U),
 `kSDMMC_CommandClassLockCard` = (1U << 7U),
 `kSDMMC_CommandClassApplicationSpecific` = (1U << 8U),
 `kSDMMC_CommandClassInputOutputMode` = (1U << 9U),
 `kSDMMC_CommandClassSwitch` = (1U << 10U) }

 SD card command class.
  - enum {
 `kSD_OcrPowerUpBusyFlag` = 31,
 `kSD_OcrHostCapacitySupportFlag` = 30,
 `kSD_OcrCardCapacitySupportFlag` = `kSD_OcrHostCapacitySupportFlag`,
 `kSD_OcrSwitch18RequestFlag` = 24,
 `kSD_OcrSwitch18AcceptFlag` = `kSD_OcrSwitch18RequestFlag`,
 `kSD_OcrVdd27_28Flag` = 15,
 `kSD_OcrVdd28_29Flag` = 16,
 `kSD_OcrVdd29_30Flag` = 17,
 `kSD_OcrVdd30_31Flag` = 18,
 `kSD_OcrVdd31_32Flag` = 19,
 `kSD_OcrVdd32_33Flag` = 20,
 `kSD_OcrVdd33_34Flag` = 21,
 `kSD_OcrVdd34_35Flag` = 22,
 `kSD_OcrVdd35_36Flag` = 23 }

 OCR register in SD card.
  - enum {
 `kSD_SpecificationVersion1_0` = (1U << 0U),
 `kSD_SpecificationVersion1_1` = (1U << 1U),
 `kSD_SpecificationVersion2_0` = (1U << 2U),
 `kSD_SpecificationVersion3_0` = (1U << 3U) }

 SD card specification version number.

- enum `sd_switch_mode_t` {
   
  `kSD_SwitchCheck` = 0U,
   
  `kSD_SwitchSet` = 1U }
   
    *SD card switch mode.*
- enum {
   
  `kSD_CsdReadBlockPartialFlag` = (1U << 0U),
   
  `kSD_CsdWriteBlockMisalignFlag` = (1U << 1U),
   
  `kSD_CsdReadBlockMisalignFlag` = (1U << 2U),
   
  `kSD_CsdDsrImplementedFlag` = (1U << 3U),
   
  `kSD_CsdEraseBlockEnabledFlag` = (1U << 4U),
   
  `kSD_CsdWriteProtectGroupEnabledFlag` = (1U << 5U),
   
  `kSD_CsdWriteBlockPartialFlag` = (1U << 6U),
   
  `kSD_CsdFileFormatGroupFlag` = (1U << 7U),
   
  `kSD_CsdCopyFlag` = (1U << 8U),
   
  `kSD_CsdPermanentWriteProtectFlag` = (1U << 9U),
   
  `kSD_CsdTemporaryWriteProtectFlag` = (1U << 10U) }
   
    *SD card CSD register flags.*
- enum {
   
  `kSD_ScrDataStatusAfterErase` = (1U << 0U),
   
  `kSD_ScrSdSpecification3` = (1U << 1U) }
   
    *SD card SCR register flags.*
- enum {
   
  `kSD_FunctionSDR12Deafult` = 0U,
   
  `kSD_FunctionSDR25HighSpeed` = 1U,
   
  `kSD_FunctionSDR50` = 2U,
   
  `kSD_FunctionSDR104` = 3U,
   
  `kSD_FunctionDDR50` = 4U }
   
    *SD timing function number.*
- enum {
   
  `kSD_GroupTimingMode` = 0U,
   
  `kSD_GroupCommandSystem` = 1U,
   
  `kSD_GroupDriverStrength` = 2U,
   
  `kSD_GroupCurrentLimit` = 3U }  
*SD group number.*
- enum `sd_timing_mode_t` {
   
  `kSD_TimingSDR12DefaultMode` = 0U,
   
  `kSD_TimingSDR25HighSpeedMode` = 1U,
   
  `kSD_TimingSDR50Mode` = 2U,
   
  `kSD_TimingSDR104Mode` = 3U,
   
  `kSD_TimingDDR50Mode` = 4U }  
*SD card timing mode flags.*
- enum `sd_driver_strength_t` {
   
  `kSD_DriverStrengthTypeB` = 0U,
   
  `kSD_DriverStrengthTypeA` = 1U,
   
  `kSD_DriverStrengthTypeC` = 2U,
   
  `kSD_DriverStrengthTypeD` = 3U }

- *SD card driver strength.*
- enum `sd_max_current_t` {
   
    `kSD_CurrentLimit200MA` = 0U,
   
    `kSD_CurrentLimit400MA` = 1U,
   
    `kSD_CurrentLimit600MA` = 2U,
   
    `kSD_CurrentLimit800MA` = 3U }
- *SD card current limit.*
- enum `sdmmc_command_t` {
   
    `kSDMMC_GoIdleState` = 0U,
   
    `kSDMMC_AllSendCid` = 2U,
   
    `kSDMMC_SetDsr` = 4U,
   
    `kSDMMC_SelectCard` = 7U,
   
    `kSDMMC_SendCsd` = 9U,
   
    `kSDMMC_SendCid` = 10U,
   
    `kSDMMC_StopTransmission` = 12U,
   
    `kSDMMC_SendStatus` = 13U,
   
    `kSDMMC_GoInactiveState` = 15U,
   
    `kSDMMC_SetBlockLength` = 16U,
   
    `kSDMMC_ReadSingleBlock` = 17U,
   
    `kSDMMC_ReadMultipleBlock` = 18U,
   
    `kSDMMC_SetBlockCount` = 23U,
   
    `kSDMMC_WriteSingleBlock` = 24U,
   
    `kSDMMC_WriteMultipleBlock` = 25U,
   
    `kSDMMC_ProgramCsd` = 27U,
   
    `kSDMMC_SetWriteProtect` = 28U,
   
    `kSDMMC_ClearWriteProtect` = 29U,
   
    `kSDMMC_SendWriteProtect` = 30U,
   
    `kSDMMC_Erase` = 38U,
   
    `kSDMMC_LockUnlock` = 42U,
   
    `kSDMMC_ApplicationCommand` = 55U,
   
    `kSDMMC_GeneralCommand` = 56U,
   
    `kSDMMC_ReadOcr` = 58U }
- *SD/MMC card common commands.*
- enum {

```

kSDIO_RegCCCRSdioVer = 0x00U,
kSDIO_RegSDVersion = 0x01U,
kSDIO_RegIOEnable = 0x02U,
kSDIO_RegIOReady = 0x03U,
kSDIO_RegIOIntEnable = 0x04U,
kSDIO_RegIOIntPending = 0x05U,
kSDIO_RegIOAbort = 0x06U,
kSDIO_RegBusInterface = 0x07U,
kSDIO_RegCardCapability = 0x08U,
kSDIO_RegCommonCISPointer = 0x09U,
kSDIO_RegBusSuspend = 0x0C,
kSDIO_RegFunctionSelect = 0x0DU,
kSDIO_RegExecutionFlag = 0x0EU,
kSDIO_RegReadyFlag = 0x0FU,
kSDIO_RegFN0BlockSizeLow = 0x10U,
kSDIO_RegFN0BlockSizeHigh = 0x11U,
kSDIO_RegPowerControl = 0x12U,
kSDIO_RegBusSpeed = 0x13U,
kSDIO_RegUHSITimingSupport = 0x14U,
kSDIO_RegDriverStrength = 0x15U,
kSDIO_RegInterruptExtension = 0x16U }

    sdio card cccr register addr
• enum sdio_command_t {
    kSDIO_SendRelativeAddress = 3U,
    kSDIO_SendOperationCondition = 5U,
    kSDIO_SendInterfaceCondition = 8U,
    kSDIO_RWIODirect = 52U,
    kSDIO_RWIOExtended = 53U }

    sdio card individual commands
• enum sdio_func_num_t {
    kSDIO_FunctionNum0,
    kSDIO_FunctionNum1,
    kSDIO_FunctionNum2,
    kSDIO_FunctionNum3,
    kSDIO_FunctionNum4,
    kSDIO_FunctionNum5,
    kSDIO_FunctionNum6,
    kSDIO_FunctionNum7,
    kSDIO_FunctionMemory }

    sdio card individual commands
• enum {

```

- ```

kSDIO_StatusCmdCRCError = 0x8000U,
kSDIO_StatusIllegalCmd = 0x4000U,
kSDIO_StatusR6Error = 0x2000U,
kSDIO_StatusError = 0x0800U,
kSDIO_StatusFunctionNumError = 0x0200U,
kSDIO_StatusOutOfRange = 0x0100U }
    
```

*sdio command response flag*
- enum {

```

kSDIO_OcrPowerUpBusyFlag = 31,
kSDIO_OcrIONumber = 28,
kSDIO_OcrMemPresent = 27,
kSDIO_OcrVdd20_21Flag = 8,
kSDIO_OcrVdd21_22Flag = 9,
kSDIO_OcrVdd22_23Flag = 10,
kSDIO_OcrVdd23_24Flag = 11,
kSDIO_OcrVdd24_25Flag = 12,
kSDIO_OcrVdd25_26Flag = 13,
kSDIO_OcrVdd26_27Flag = 14,
kSDIO_OcrVdd27_28Flag = 15,
kSDIO_OcrVdd28_29Flag = 16,
kSDIO_OcrVdd29_30Flag = 17,
kSDIO_OcrVdd30_31Flag = 18,
kSDIO_OcrVdd31_32Flag = 19,
kSDIO_OcrVdd32_33Flag = 20,
kSDIO_OcrVdd33_34Flag = 21,
kSDIO_OcrVdd34_35Flag = 22,
kSDIO_OcrVdd35_36Flag = 23 }
    
```

*sdio operation condition flag*
- enum {

```

kSDIO_CCCRSupportDirectCmdDuringDataTrans = (1UL << 0U),
kSDIO_CCCRSupportMultiBlock = (1UL << 1U),
kSDIO_CCCRSupportReadWait = (1UL << 2U),
kSDIO_CCCRSupportSuspendResume = (1UL << 3U),
kSDIO_CCCRSupportIntDuring4BitDataTrans = (1UL << 4U),
kSDIO_CCCRSupportLowSpeed1Bit = (1UL << 6U),
kSDIO_CCCRSupportLowSpeed4Bit = (1UL << 7U),
kSDIO_CCCRSupportMasterPowerControl = (1UL << 8U),
kSDIO_CCCRSupportHighSpeed = (1UL << 9U),
kSDIO_CCCRSupportContinuousSPIInt = (1UL << 10U) }
    
```

*sdio capability flag*
- enum {

```

kSDIO_FBRSupportCSA = (1U << 0U),
kSDIO_FBRSupportPowerSelection = (1U << 1U) }
    
```

*sdio fbr flag*
- enum **sdio\_bus\_width\_t** {

- ```

kSDIO_DataBus1Bit = 0x00U,
kSDIO_DataBus4Bit = 0X02U,
kSDIO_DataBus8Bit = 0X03U }
    
```

*sdio bus width*
- enum `mmc_command_t` {
 

```

kMMC_SendOperationCondition = 1U,
kMMC_SetRelativeAddress = 3U,
kMMC_SleepAwake = 5U,
kMMC_Switch = 6U,
kMMC_SendExtendedCsd = 8U,
kMMC_ReadDataUntilStop = 11U,
kMMC_BusTestRead = 14U,
kMMC_SendingBusTest = 19U,
kMMC_WriteDataUntilStop = 20U,
kMMC_SendTuningBlock = 21U,
kMMC_ProgramCid = 26U,
kMMC_EraseGroupStart = 35U,
kMMC_EraseGroupEnd = 36U,
kMMC_FastInputOutput = 39U,
kMMC_GoInterruptState = 40U }
    
```

*MMC card individual commands.*
- enum `mmc_classified_voltage_t` {
 

```

kMMC_ClassifiedVoltageHigh = 0U,
kMMC_ClassifiedVoltageDual = 1U }
    
```

*MMC card classified as voltage range.*
- enum `mmc_classified_density_t` { `kMMC_ClassifiedDensityWithin2GB` = 0U }
- enum `mmc_access_mode_t` {
 

```

kMMC_AccessModeByte = 0U,
kMMC_AccessModeSector = 2U }
    
```

*MMC card access mode(Access mode in OCR).*
- enum `mmc_voltage_window_t` {
 

```

kMMC_VoltageWindowNone = 0U,
kMMC_VoltageWindow120 = 0x01U,
kMMC_VoltageWindow170to195 = 0x02U,
kMMC_VoltageWindows270to360 = 0x1FFU }
    
```

*MMC card voltage window(VDD voltage window in OCR).*
- enum `mmc_csd_structure_version_t` {
 

```

kMMC_CsdStrucureVersion10 = 0U,
kMMC_CsdStrucureVersion11 = 1U,
kMMC_CsdStrucureVersion12 = 2U,
kMMC_CsdStrucureVersionInExtcsd = 3U }
    
```

*CSD structure version(CSD\_STRUCTURE in CSD).*
- enum `mmc_specification_version_t` { }

```
kMMC_SpecificationVersion0 = 0U,
kMMC_SpecificationVersion1 = 1U,
kMMC_SpecificationVersion2 = 2U,
kMMC_SpecificationVersion3 = 3U,
kMMC_SpecificationVersion4 = 4U }
```

*MMC card specification version(SPEC\_VERS in CSD).*

- enum {
   
kMMC\_ExtendedCsdRevision10 = 0U,
 kMMC\_ExtendedCsdRevision11 = 1U,
 kMMC\_ExtendedCsdRevision12 = 2U,
 kMMC\_ExtendedCsdRevision13 = 3U,
 kMMC\_ExtendedCsdRevision14 = 4U,
 kMMC\_ExtendedCsdRevision15 = 5U,
 kMMC\_ExtendedCsdRevision16 = 6U,
 kMMC\_ExtendedCsdRevision17 = 7U }

*MMC card Extended CSD fix version(EXT\_CSD\_REV in Extended CSD)*

- enum **mmc\_command\_set\_t** {
   
kMMC\_CommandSetStandard = 0U,
 kMMC\_CommandSet1 = 1U,
 kMMC\_CommandSet2 = 2U,
 kMMC\_CommandSet3 = 3U,
 kMMC\_CommandSet4 = 4U }

*MMC card command set(COMMAND\_SET in Extended CSD)*

- enum {
   
kMMC\_SupportAlternateBoot = 1U,
 kMMC\_SupportDDRBoot = 2U,
 kMMC\_SupportHighSpeedBoot = 4U }
- boot support(BOOT\_INFO in Extended CSD)*
- enum **mmc\_high\_speed\_timing\_t** {
   
kMMC\_HighSpeedTimingNone = 0U,
 kMMC\_HighSpeedTiming = 1U,
 kMMC\_HighSpeed200Timing = 2U,
 kMMC\_HighSpeed400Timing = 3U,
 kMMC\_EnhanceHighSpeed400Timing = 4U }

*MMC card high-speed timing(HS\_TIMING in Extended CSD)*

- enum **mmc\_data\_bus\_width\_t** {
   
kMMC\_DataBusWidth1bit = 0U,
 kMMC\_DataBusWidth4bit = 1U,
 kMMC\_DataBusWidth8bit = 2U,
 kMMC\_DataBusWidth4bitDDR = 5U,
 kMMC\_DataBusWidth8bitDDR = 6U,
 kMMC\_DataBusWidth8bitDDRSTROBE = 0x86U }

*MMC card data bus width(BUS\_WIDTH in Extended CSD)*

- enum **mmc\_boot\_partition\_enable\_t** {

```
kMMC_BootPartitionEnableNot = 0U,
kMMC_BootPartitionEnablePartition1 = 1U,
kMMC_BootPartitionEnablePartition2 = 2U,
kMMC_BootPartitionEnableUserAera = 7U }
```

*MMC card boot partition enabled(BOOT\_PARTITION\_ENABLE in Extended CSD)*

- enum `mmc_boot_timing_mode_t` {
 

```
kMMC_BootModeSDRWithDefaultTiming = 0U,
kMMC_BootModeSDRWithHighSpeedTiming = 1U,
kMMC_BootModeDDRTiming = 2U }
```

*boot mode configuration Note: HS200 & HS400 is not support during BOOT operation.*
- enum `mmc_boot_partition_wp_t` {
 

```
kMMC_BootPartitionWPDisable = 0x50U,
kMMC_BootPartitionPwrWPToBothPartition,
kMMC_BootPartitionPermWPToBothPartition = 0x04U,
kMMC_BootPartitionPwrWPToPartition1 = (1U << 7U) | 1U,
kMMC_BootPartitionPwrWPToPartition2 = (1U << 7U) | 3U,
kMMC_BootPartitionPermWPToPartition1,
kMMC_BootPartitionPermWPToPartition2,
kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2,
kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 }
```

*MMC card boot partition write protect configurations All the bits in BOOT\_WP register, except the two R/W bits B\_PERM\_WP\_DIS and B\_PERM\_WP\_EN, shall only be written once per power cycle. The protection mode intended for both boot areas will be set with a single write.*

- enum {
 

```
kMMC_BootPartitionNotProtected = 0U,
kMMC_BootPartitionPwrProtected = 1U,
kMMC_BootPartitionPermProtected = 2U }
```

*MMC card boot partition write protect status.*

- enum `mmc_access_partition_t` {
 

```
kMMC_AccessPartitionUserAera = 0U,
kMMC_AccessPartitionBoot1 = 1U,
kMMC_AccessPartitionBoot2 = 2U,
kMMC_AccessRPMB = 3U,
kMMC_AccessGeneralPurposePartition1 = 4U,
kMMC_AccessGeneralPurposePartition2 = 5U,
kMMC_AccessGeneralPurposePartition3 = 6U,
kMMC_AccessGeneralPurposePartition4 = 7U }
```

*MMC card partition to be accessed(BOOT\_PARTITION\_ACCESS in Extended CSD)*
- enum {

```

kMMC_CsdReadBlockPartialFlag = (1U << 0U),
kMMC_CsdWriteBlockMisalignFlag = (1U << 1U),
kMMC_CsdReadBlockMisalignFlag = (1U << 2U),
kMMC_CsdDsrImplementedFlag = (1U << 3U),
kMMC_CsdWriteProtectGroupEnabledFlag = (1U << 4U),
kMMC_CsdWriteBlockPartialFlag = (1U << 5U),
kMMC_ContentProtectApplicationFlag = (1U << 6U),
kMMC_CsdFileFormatGroupFlag = (1U << 7U),
kMMC_CsdCopyFlag = (1U << 8U),
kMMC_CsdPermanentWriteProtectFlag = (1U << 9U),
kMMC_CsdTemporaryWriteProtectFlag = (1U << 10U) }

```

*MMC card CSD register flags.*

- enum `mmc_extended_csd_access_mode_t` {
   
kMMC\_ExtendedCsdAccessModeCommandSet = 0U,
   
kMMC\_ExtendedCsdAccessModeSetBits = 1U,
   
kMMC\_ExtendedCsdAccessModeClearBits = 2U,
   
kMMC\_ExtendedCsdAccessModeWriteBits = 3U }

*Extended CSD register access mode(Access mode in CMD6).*

- enum `mmc_extended_csd_index_t` {
   
kMMC\_ExtendedCsdIndexFlushCache = 32U,
   
kMMC\_ExtendedCsdIndexCacheControl = 33U,
   
kMMC\_ExtendedCsdIndexBootPartitionWP = 173U,
   
kMMC\_ExtendedCsdIndexEraseGroupDefinition = 175U,
   
kMMC\_ExtendedCsdIndexBootBusConditions = 177U,
   
kMMC\_ExtendedCsdIndexBootConfigWP = 178U,
   
kMMC\_ExtendedCsdIndexPartitionConfig = 179U,
   
kMMC\_ExtendedCsdIndexBusWidth = 183U,
   
kMMC\_ExtendedCsdIndexHighSpeedTiming = 185U,
   
kMMC\_ExtendedCsdIndexPowerClass = 187U,
   
kMMC\_ExtendedCsdIndexCommandSet = 191U }

*EXT CSD byte index.*

- enum {
   
kMMC\_DriverStrength0 = 0U,
   
kMMC\_DriverStrength1 = 1U,
   
kMMC\_DriverStrength2 = 2U,
   
kMMC\_DriverStrength3 = 3U,
   
kMMC\_DriverStrength4 = 4U }
- *mmc driver strength*
- enum `mmc_extended_csd_flags_t` {
   
kMMC\_ExtCsdExtPartitionSupport = (1 << 0U),
   
kMMC\_ExtCsdEnhancePartitionSupport = (1 << 1U),
   
kMMC\_ExtCsdPartitioningSupport = (1 << 2U),
   
kMMC\_ExtCsdPrgCIDCSDInDDRModeSupport = (1 << 3U),
   
kMMC\_ExtCsdBKOpsSupport = (1 << 4U),
   
kMMC\_ExtCsdDataTagSupport = (1 << 5U),
   
kMMC\_ExtCsdModeOperationCodeSupport = (1 << 6U) }

- enum `mmc_boot_mode_t` {
   
    `kMMC_BootModeNormal` = 0U,
   
    `kMMC_BootModeAlternative` = 1U }
   
    *MMC card boot mode.*

## common function

tuning pattern

- `status_t SDMMC_SelectCard (sdmmchost_t *host, uint32_t relativeAddress, bool isSelected)`
  
    *Selects the card to put it into transfer state.*
- `status_t SDMMC_SendApplicationCommand (sdmmchost_t *host, uint32_t relativeAddress)`
  
    *Sends an application command.*
- `status_t SDMMC_SetBlockCount (sdmmchost_t *host, uint32_t blockCount)`
  
    *Sets the block count.*
- `status_t SDMMC_GoIdle (sdmmchost_t *host)`
  
    *Sets the card to be idle state.*
- `status_t SDMMC_SetBlockSize (sdmmchost_t *host, uint32_t blockSize)`
  
    *Sets data block size.*
- `status_t SDMMC_SetCardInactive (sdmmchost_t *host)`
  
    *Sets card to inactive status.*

### 44.7.2 Data Structure Documentation

#### 44.7.2.1 struct sd\_detect\_card\_t

##### Data Fields

- `sd_detect_card_type_t type`
  
    *card detect type*
- `uint32_t cdDebounce_ms`
  
    *card detect debounce delay ms*
- `sd_cd_t callback`
  
    *card inserted callback which is meaningful for interrupt case*
- `sd_cd_status_t cardDetected`
  
    *used to check sd cd status when card detect through GPIO*
- `sd_dat3_pull_t dat3PullFunc`
  
    *function pointer of DATA3 pull up/down*
- `void * userData`
  
    *user data*

#### 44.7.2.2 struct sd\_io\_voltage\_t

##### Data Fields

- `sd_io_voltage_ctrl_type_t type`

- **sd\_io\_voltage\_func\_t func**  
*io voltage switch function*

#### 44.7.2.3 struct sd\_usr\_param\_t

##### Data Fields

- **sd\_pwr\_t pwr**  
*power control configuration pointer*
- **uint32\_t powerOnDelayMS**  
*power on delay time*
- **uint32\_t powerOffDelayMS**  
*power off delay time*
- **sd\_io\_strength\_t ioStrength**  
*switch sd io strength*
- **sd\_io\_voltage\_t \* ioVoltage**  
*switch io voltage*
- **sd\_detect\_card\_t \* cd**  
*card detect*
- **uint32\_t maxFreq**  
*board support maximum frequency*
- **uint32\_t capability**  
*board capability flag*

#### 44.7.2.4 struct sdio\_card\_int\_t

##### Data Fields

- **void \* userData**  
*user data*
- **sdio\_int\_t cardInterrupt**  
*card int call back*

#### 44.7.2.5 struct sdio\_usr\_param\_t

##### Data Fields

- **sd\_pwr\_t pwr**  
*power control configuration pointer*
- **uint32\_t powerOnDelayMS**  
*power on delay time*
- **uint32\_t powerOffDelayMS**  
*power off delay time*
- **sd\_io\_strength\_t ioStrength**  
*switch sd io strength*
- **sd\_io\_voltage\_t \* ioVoltage**  
*switch io voltage*

- `sd_detect_card_t * cd`  
*card detect*
- `sdio_card_int_t * sdioInt`  
*card int*
- `uint32_t maxFreq`  
*board support maximum frequency*
- `uint32_t capability`  
*board capability flag*

#### 44.7.2.6 struct `sdio_fbr_t`

##### Data Fields

- `uint8_t flags`  
*current io flags*
- `uint8_t ioStdFunctionCode`  
*current io standard function code*
- `uint8_t ioExtFunctionCode`  
*current io extended function code*
- `uint32_t ioPointerToCIS`  
*current io pointer to CIS*
- `uint32_t ioPointerToCSA`  
*current io pointer to CSA*
- `uint16_t ioBlockSize`  
*current io block size*

#### 44.7.2.7 struct `sdio_common_cis_t`

##### Data Fields

- `uint16_t mID`  
*manufacturer code*
- `uint16_t mInfo`  
*manufacturer information*
- `uint8_t funcID`  
*function ID*
- `uint16_t fn0MaxBlkSize`  
*function 0 max block size*
- `uint8_t maxTransSpeed`  
*max data transfer speed for all function*

#### 44.7.2.8 struct `sdio_func_cis_t`

##### Data Fields

- `uint8_t funcID`  
*function ID*
- `uint8_t funcInfo`

- *function info*
- **uint8\_t ioVersion**  
*level of application specification this io support*
- **uint32\_t cardPSN**  
*product serial number*
- **uint32\_t ioCSASize**  
*available CSA size for io*
- **uint8\_t ioCSAProperty**  
*CSA property.*
- **uint16\_t ioMaxBlockSize**  
*io max transfer data size*
- **uint32\_t ioOCR**  
*io ioeration condition*
- **uint8\_t ioOPMinPwr**  
*min current in operation mode*
- **uint8\_t ioOPAvgPwr**  
*average current in operation mode*
- **uint8\_t ioOPMaxPwr**  
*max current in operation mode*
- **uint8\_t ioSBMinPwr**  
*min current in standby mode*
- **uint8\_t ioSBAvgPwr**  
*average current in standby mode*
- **uint8\_t ioSBMaxPwr**  
*max current in standby mode*
- **uint16\_t ioMinBandWidth**  
*io min transfer bandwidth*
- **uint16\_t ioOptimumBandWidth**  
*io optimum transfer bandwidth*
- **uint16\_t ioReadyTimeout**  
*timeout value from enalbe to ready*
- **uint16\_t ioHighCurrentAvgCurrent**  
*the average peak current (mA)*  
*when IO operating in high current mode*
- **uint16\_t ioHighCurrentMaxCurrent**  
*the max peak current (mA)*  
*when IO operating in high current mode*
- **uint16\_t ioLowCurrentAvgCurrent**  
*the average peak current (mA)*  
*when IO operating in lower current mode*
- **uint16\_t ioLowCurrentMaxCurrent**  
*the max peak current (mA)*  
*when IO operating in lower current mode*

#### 44.7.2.9 struct sd\_status\_t

##### Data Fields

- **uint8\_t busWidth**  
*current buswidth*

- `uint8_t secureMode`  
*secured mode*
- `uint16_t cardType`  
*sdcard type*
- `uint32_t protectedSize`  
*size of protected area*
- `uint8_t speedClass`  
*speed class of card*
- `uint8_t performanceMove`  
*Performance of move indicated by 1[MB/S]step.*
- `uint8_t auSize`  
*size of AU*
- `uint16_t eraseSize`  
*number of AUs to be erased at a time*
- `uint8_t eraseTimeout`  
*timeout value for erasing areas specified by UNIT OF ERASE AU*
- `uint8_t eraseOffset`  
*fixed offset value added to erase time*
- `uint8_t uhsSpeedGrade`  
*speed grade for UHS mode*
- `uint8_t uhsAuSize`  
*size of AU for UHS mode*

#### 44.7.2.10 struct sd\_cid\_t

##### Data Fields

- `uint8_t manufacturerID`  
*Manufacturer ID [127:120].*
- `uint16_t applicationID`  
*OEM/Application ID [119:104].*
- `uint8_t productName [SD_PRODUCT_NAME_BYTES]`  
*Product name [103:64].*
- `uint8_t productVersion`  
*Product revision [63:56].*
- `uint32_t productSerialNumber`  
*Product serial number [55:24].*
- `uint16_t manufacturerData`  
*Manufacturing date [19:8].*

#### 44.7.2.11 struct sd\_csd\_t

##### Data Fields

- `uint8_t csdStructure`  
*CSD structure [127:126].*
- `uint8_t dataReadAccessTime1`  
*Data read access-time-1 [119:112].*
- `uint8_t dataReadAccessTime2`

- **uint8\_t transferSpeed**  
*Maximum data transfer rate [103:96].*
- **uint16\_t cardCommandClass**  
*Card command classes [95:84].*
- **uint8\_t readBlockLength**  
*Maximum read data block length [83:80].*
- **uint16\_t flags**  
*Flags in \_sd\_csd\_flag.*
- **uint32\_t deviceSize**  
*Device size [73:62].*
- **uint8\_t readCurrentVddMin**  
*Maximum read current at VDD min [61:59].*
- **uint8\_t readCurrentVddMax**  
*Maximum read current at VDD max [58:56].*
- **uint8\_t writeCurrentVddMin**  
*Maximum write current at VDD min [55:53].*
- **uint8\_t writeCurrentVddMax**  
*Maximum write current at VDD max [52:50].*
- **uint8\_t deviceSizeMultiplier**  
*Device size multiplier [49:47].*
- **uint8\_t eraseSectorSize**  
*Erase sector size [45:39].*
- **uint8\_t writeProtectGroupSize**  
*Write protect group size [38:32].*
- **uint8\_t writeSpeedFactor**  
*Write speed factor [28:26].*
- **uint8\_t writeBlockLength**  
*Maximum write data block length [25:22].*
- **uint8\_t fileFormat**  
*File format [11:10].*

#### 44.7.2.12 struct sd\_scr\_t

##### Data Fields

- **uint8\_t scrStructure**  
*SCR Structure [63:60].*
- **uint8\_t sdSpecification**  
*SD memory card specification version [59:56].*
- **uint16\_t flags**  
*SCR flags in \_sd\_scr\_flag.*
- **uint8\_t sdSecurity**  
*Security specification supported [54:52].*
- **uint8\_t sdBusWidths**  
*Data bus widths supported [51:48].*
- **uint8\_t extendedSecurity**  
*Extended security support [46:43].*
- **uint8\_t commandSupport**  
*Command support bits [33:32] 33-support CMD23, 32-support cmd20.*

- `uint32_t reservedForManufacturer`  
*reserved for manufacturer usage [31:0]*

#### 44.7.2.13 struct mmc\_cid\_t

##### Data Fields

- `uint8_t manufacturerID`  
*Manufacturer ID.*
- `uint16_t applicationID`  
*OEM/Application ID.*
- `uint8_t productName [MMC_PRODUCT_NAME_BYTES]`  
*Product name.*
- `uint8_t productVersion`  
*Product revision.*
- `uint32_t productSerialNumber`  
*Product serial number.*
- `uint8_t manufacturerData`  
*Manufacturing date.*

#### 44.7.2.14 struct mmc\_csd\_t

##### Data Fields

- `uint8_t csdStructureVersion`  
*CSD structure [127:126].*
- `uint8_t systemSpecificationVersion`  
*System specification version [125:122].*
- `uint8_t dataReadAccessTime1`  
*Data read access-time 1 [119:112].*
- `uint8_t dataReadAccessTime2`  
*Data read access-time 2 in CLOCK cycles (NSAC\*100) [111:104].*
- `uint8_t transferSpeed`  
*Max.*
- `uint16_t cardCommandClass`  
*card command classes [95:84]*
- `uint8_t readBlockLength`  
*Max.*
- `uint16_t flags`  
*Contain flags in \_mmc\_csd\_flag.*
- `uint16_t deviceSize`  
*Device size [73:62].*
- `uint8_t readCurrentVddMin`  
*Max.*
- `uint8_t readCurrentVddMax`  
*Max.*
- `uint8_t writeCurrentVddMin`  
*Max.*
- `uint8_t writeCurrentVddMax`

- **uint8\_t deviceSizeMultiplier**  
*Device size multiplier [49:47].*
- **uint8\_t eraseGroupSize**  
*Erase group size [46:42].*
- **uint8\_t eraseGroupSizeMultiplier**  
*Erase group size multiplier [41:37].*
- **uint8\_t writeProtectGroupSize**  
*Write protect group size [36:32].*
- **uint8\_t defaultEcc**  
*Manufacturer default ECC [30:29].*
- **uint8\_t writeSpeedFactor**  
*Write speed factor [28:26].*
- **uint8\_t maxWriteBlockLength**  
*Max.*
- **uint8\_t fileFormat**  
*File format [11:10].*
- **uint8\_t eccCode**  
*ECC code [9:8].*

## Field Documentation

(1) **uint8\_t mmc\_csd\_t::transferSpeed**

bus clock frequency [103:96]

(2) **uint8\_t mmc\_csd\_t::readBlockLength**

read data block length [83:80]

(3) **uint8\_t mmc\_csd\_t::readCurrentVddMin**

read current @ VDD min [61:59]

(4) **uint8\_t mmc\_csd\_t::readCurrentVddMax**

read current @ VDD max [58:56]

(5) **uint8\_t mmc\_csd\_t::writeCurrentVddMin**

write current @ VDD min [55:53]

(6) **uint8\_t mmc\_csd\_t::writeCurrentVddMax**

write current @ VDD max [52:50]

(7) **uint8\_t mmc\_csd\_t::maxWriteBlockLength**

write data block length [25:22]

#### 44.7.2.15 struct mmc\_extended\_csd\_t

##### Data Fields

- uint8\_t **cacheCtrl**  
*< secure removal type[16]*
- uint8\_t **partitionAttribute**  
*< power off notification[34]*
- uint8\_t **userWP**  
*< max enhance area size [159-157]*
- uint8\_t **bootPartitionWP**  
*boot write protect register[173]*
- uint8\_t **bootWPStatus**  
*boot write protect status register[174]*
- uint8\_t **highDensityEraseGroupDefinition**  
*High-density erase group definition [175].*
- uint8\_t **bootDataBusConditions**  
*Boot bus conditions [177].*
- uint8\_t **bootConfigProtect**  
*Boot config protection [178].*
- uint8\_t **partitionConfig**  
*Boot configuration [179].*
- uint8\_t **eraseMemoryContent**  
*Erased memory content [181].*
- uint8\_t **dataBusWidth**  
*Data bus width mode [183].*
- uint8\_t **highSpeedTiming**  
*High-speed interface timing [185].*
- uint8\_t **powerClass**  
*Power class [187].*
- uint8\_t **commandSetRevision**  
*Command set revision [189].*
- uint8\_t **commandSet**  
*Command set [191].*
- uint8\_t **extendecCsdVersion**  
*Extended CSD revision [192].*
- uint8\_t **csdStructureVersion**  
*CSD structure version [194].*
- uint8\_t **cardType**  
*Card Type [196].*
- uint8\_t **ioDriverStrength**  
*IO driver strength [197].*
- uint8\_t **partitionSwitchTimeout**  
*< out of interrupt busy timing [198]*
- uint8\_t **powerClass52MHz195V**  
*Power Class for 52MHz @ 1.95V [200].*
- uint8\_t **powerClass26MHz195V**  
*Power Class for 26MHz @ 1.95V [201].*
- uint8\_t **powerClass52MHz360V**  
*Power Class for 52MHz @ 3.6V [202].*
- uint8\_t **powerClass26MHz360V**

- **Power Class for 26MHz @ 3.6V [203].**
- **uint8\_t minimumReadPerformance4Bit26MHz**  
*Minimum Read Performance for 4bit at 26MHz [205].*
- **uint8\_t minimumWritePerformance4Bit26MHz**  
*Minimum Write Performance for 4bit at 26MHz [206].*
- **uint8\_t minimumReadPerformance8Bit26MHz4Bit52MHz**  
*Minimum read Performance for 8bit at 26MHz/4bit @ 52MHz [207].*
- **uint8\_t minimumWritePerformance8Bit26MHz4Bit52MHz**  
*Minimum Write Performance for 8bit at 26MHz/4bit @ 52MHz [208].*
- **uint8\_t minimumReadPerformance8Bit52MHz**  
*Minimum Read Performance for 8bit at 52MHz [209].*
- **uint8\_t minimumWritePerformance8Bit52MHz**  
*Minimum Write Performance for 8bit at 52MHz [210].*
- **uint32\_t sectorCount**  
*Sector Count [215:212].*
- **uint8\_t sleepAwakeTimeout**  
*< sleep notification timeout [216]*
- **uint8\_t sleepCurrentVCCQ**  
*< Production state awareness timeout [218]*
- **uint8\_t sleepCurrentVCC**  
*Sleep current (VCC) [220].*
- **uint8\_t highCapacityWriteProtectGroupSize**  
*High-capacity write protect group size [221].*
- **uint8\_t reliableWriteSectorCount**  
*Reliable write sector count [222].*
- **uint8\_t highCapacityEraseTimeout**  
*High-capacity erase timeout [223].*
- **uint8\_t highCapacityEraseUnitSize**  
*High-capacity erase unit size [224].*
- **uint8\_t accessSize**  
*Access size [225].*
- **uint8\_t minReadPerformance8bitAt52MHZDDR**  
*< secure trim multiplier[229]*
- **uint8\_t minWritePerformance8bitAt52MHZDDR**  
*Minimum write performance for 8bit at DDR 52MHZ[235].*
- **uint8\_t powerClass200MHZVCCQ130VVCC360V**  
*power class for 200MHZ, at VCCQ= 1.3V,VCC=3.6V[236]*
- **uint8\_t powerClass200MHZVCCQ195VVCC360V**  
*power class for 200MHZ, at VCCQ= 1.95V,VCC=3.6V[237]*
- **uint8\_t powerClass52MHZDDR195V**  
*power class for 52MHZ,DDR at Vcc 1.95V[238]*
- **uint8\_t powerClass52MHZDDR360V**  
*power class for 52MHZ,DDR at Vcc 3.6V[239]*
- **uint32\_t genericCMD6Timeout**  
*< 1st initialization time after partitioning[241]*
- **uint32\_t cacheSize**  
*cache size[252-249]*
- **uint8\_t powerClass200MHZDDR360V**  
*power class for 200MHZ, DDR at VCC=2.6V[253]*
- **uint8\_t extPartitionSupport**  
*< fw VERSION [261-254]*

- `uint8_t supportedCommandSet`  
*< large unit size[495]*

### Field Documentation

(1) `uint8_t mmc_extended_csd_t::cacheCtrl`

< product state awareness enablement[17]  
< max preload data size[21-18]  
< pre-load data size[25-22]  
< FFU status [26]  
< mode operation code[29]  
< mode config [30] control to turn on/off cache[33]

(2) `uint8_t mmc_extended_csd_t::partitionAttribute`

< packed cmd fail index [35]  
< packed cmd status[36]  
< context configuration[51-37]  
< extended partitions attribut[53-52]  
< exception events status[55-54]  
< exception events control[57-56]  
< number of group to be released[58]  
< class 6 command control[59]  
< 1st initialization after disabling sector size emu[60]  
< sector size[61]  
< sector size emulation[62]  
< native sector size[63]  
< period wakeup [131]  
< package case temperature is controlled[132]  
< production state awareness[133]  
< enhanced user data start addr [139-136]  
< enhanced user data area size[142-140]  
< general purpose partition size[154-143] partition attribute [156]

(3) `uint8_t mmc_extended_csd_t::userWP`

< HPI management [161]

< write reliability parameter register[166]  
< write reliability setting register[167]  
< RPMB size multi [168]  
< FW configuration[169] user write protect register[171]

**(4) uint8\_t mmc\_extended\_csd\_t::partitionSwitchTimeout**

partition switch timing [199]

**(5) uint8\_t mmc\_extended\_csd\_t::sleepAwakeTimeout**

Sleep/awake timeout [217]

**(6) uint8\_t mmc\_extended\_csd\_t::sleepCurrentVCCQ**

Sleep current (VCCQ) [219]

**(7) uint8\_t mmc\_extended\_csd\_t::minReadPerformance8bitAt52MHZDDR**

< secure erase multiplier[230]  
< secure feature support[231]  
< trim multiplier[232] Minimum read performance for 8bit at DDR 52MHZ[234]

**(8) uint32\_t mmc\_extended\_csd\_t::genericCMD6Timeout**

< correct prg sectors number[245-242]  
< background operations status[246]  
< power off notification timeout[247] generic CMD6 timeout[248]

**(9) uint8\_t mmc\_extended\_csd\_t::extPartitionSupport**

< device version[263-262]  
< optimal trim size[264]  
< optimal write size[265]  
< optimal read size[266]  
< pre EOL information[267]  
< device life time estimation typeA[268]  
< device life time estimation typeB[269]  
< number of FW sectors correctly programmed[305-302]  
< FFU argument[490-487]  
< operation code timeout[491]

< support mode [493] extended partition attribute support[494]

**(10) uint8\_t mmc\_extended\_csd\_t::supportedCommandSet**

< context management capability[496]

< tag resource size[497]

< tag unit size[498]

< max packed write cmd[500]

< max packed read cmd[501]

< HPI feature[503] Supported Command Sets [504]

#### 44.7.2.16 struct mmc\_extended\_csd\_config\_t

##### Data Fields

- **mmc\_command\_set\_t commandSet**  
*Command set.*
- **uint8\_t ByteValue**  
*The value to set.*
- **uint8\_t ByteIndex**  
*The byte index in Extended CSD(mmc\_extended\_csd\_index\_t)*
- **mmc\_extended\_csd\_access\_mode\_t accessMode**  
*Access mode.*

#### 44.7.2.17 struct mmc\_boot\_config\_t

##### Data Fields

- **mmc\_boot\_mode\_t bootMode**  
*mmc boot mode*
- **bool enableBootAck**  
*Enable boot ACK.*
- **mmc\_boot\_partition\_enable\_t bootPartition**  
*Boot partition.*
- **mmc\_boot\_timing\_mode\_t bootTimingMode**  
*boot mode*
- **mmc\_data\_bus\_width\_t bootDataBusWidth**  
*Boot data bus width.*
- **bool retainBootbusCondition**  
*If retain boot bus width and boot mode conditions.*
- **bool pwrBootConfigProtection**  
*Disable the change of boot configuration register bits from at this point until next power cycle or next H/W reset operation*
- **bool premBootConfigProtection**  
*Disable the change of boot configuration register bits permanently.*
- **mmc\_boot\_partition\_wp\_t bootPartitionWP**

*boot partition write protect configurations*

### 44.7.3 Macro Definition Documentation

44.7.3.1 `#define SDMMC_LOG( format, ... )`

44.7.3.2 `#define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT( CSD ) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)`

44.7.3.3 `#define READ_MMC_TRANSFER_SPEED_MULTIPLIER( CSD ) (((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)`

44.7.3.4 `#define MMC_EXTENDED_CSD_BYTES (512U)`

44.7.3.5 `#define SD_PRODUCT_NAME_BYTES (5U)`

44.7.3.6 `#define MMC_PRODUCT_NAME_BYTES (6U)`

44.7.3.7 `#define MMC_SWITCH_COMMAND_SET_SHIFT (0U)`

44.7.3.8 `#define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)`

### 44.7.4 Enumeration Type Documentation

44.7.4.1 anonymous enum

Enumerator

`kStatus_SDMMC_NotSupportYet` Haven't supported.

`kStatus_SDMMC_TransferFailed` Send command failed.

`kStatus_SDMMC_SetCardBlockSizeFailed` Set block size failed.

`kStatus_SDMMC_HostNotSupport` Host doesn't support.

`kStatus_SDMMC_CardNotSupport` Card doesn't support.

`kStatus_SDMMC_AllSendCidFailed` Send CID failed.

`kStatus_SDMMC_SendRelativeAddressFailed` Send relative address failed.

`kStatus_SDMMC_SendCsdFailed` Send CSD failed.

`kStatus_SDMMC_SelectCardFailed` Select card failed.

`kStatus_SDMMC_SendScrFailed` Send SCR failed.

`kStatus_SDMMC_SetDataBusWidthFailed` Set bus width failed.

`kStatus_SDMMC_GoIdleFailed` Go idle failed.

`kStatus_SDMMC_HandShakeOperationConditionFailed` Send Operation Condition failed.

`kStatus_SDMMC_SendApplicationCommandFailed` Send application command failed.

*kStatus\_SDMMC\_SwitchFailed* Switch command failed.  
*kStatus\_SDMMC\_StopTransmissionFailed* Stop transmission failed.  
*kStatus\_SDMMC\_WaitWriteCompleteFailed* Wait write complete failed.  
*kStatus\_SDMMC\_SetBlockCountFailed* Set block count failed.  
*kStatus\_SDMMC\_SetRelativeAddressFailed* Set relative address failed.  
*kStatus\_SDMMC\_SwitchBusTimingFailed* Switch high speed failed.  
*kStatus\_SDMMC\_SendExtendedCsdFailed* Send EXT\_CSD failed.  
*kStatus\_SDMMC\_ConfigureBootFailed* Configure boot failed.  
*kStatus\_SDMMC\_ConfigureExtendedCsdFailed* Configure EXT\_CSD failed.  
*kStatus\_SDMMC\_EnableHighCapacityEraseFailed* Enable high capacity erase failed.  
*kStatus\_SDMMC\_SendTestPatternFailed* Send test pattern failed.  
*kStatus\_SDMMC\_ReceiveTestPatternFailed* Receive test pattern failed.  
*kStatus\_SDMMC\_SDIO\_ResponseError* sdio response error  
*kStatus\_SDMMC\_SDIO\_InvalidArgument* sdio invalid argument response error  
*kStatus\_SDMMC\_SDIO\_SendOperationConditionFail* sdio send operation condition fail  
*kStatus\_SDMMC\_InvalidVoltage* invalid voltage  
*kStatus\_SDMMC\_SDIO\_SwitchHighSpeedFail* switch to high speed fail  
*kStatus\_SDMMC\_SDIO\_ReadCISFail* read CIS fail  
*kStatus\_SDMMC\_SDIO\_InvalidCard* invalid SDIO card  
*kStatus\_SDMMC\_TuningFail* tuning fail  
*kStatus\_SDMMC\_SwitchVoltageFail* switch voltage fail  
*kStatus\_SDMMC\_SwitchVoltage18VFail33VSuccess* switch voltage fail  
*kStatus\_SDMMC\_ReTuningRequest* retuning request  
*kStatus\_SDMMC\_SetDriverStrengthFail* set driver strength fail  
*kStatus\_SDMMC\_SetPowerClassFail* set power class fail  
*kStatus\_SDMMC\_HostNotReady* host controller not ready  
*kStatus\_SDMMC\_CardDetectFailed* card detect failed  
*kStatus\_SDMMC\_AuSizeNotSetProperly* AU size not set properly.  
*kStatus\_SDMMC\_PollingCardIdleFailed* polling card idle status failed  
*kStatus\_SDMMC\_DeselectCardFailed* deselect card failed  
*kStatus\_SDMMC\_CardStatusIdle* card idle  
*kStatus\_SDMMC\_CardStatusBusy* card busy  
*kStatus\_SDMMC\_CardInitFailed* card init failed

#### 44.7.4.2 anonymous enum

Enumerator

|                               |           |
|-------------------------------|-----------|
| <i>kSDMMC_SignalLineCmd</i>   | cmd line  |
| <i>kSDMMC_SignalLineData0</i> | data line |
| <i>kSDMMC_SignalLineData1</i> | data line |
| <i>kSDMMC_SignalLineData2</i> | data line |
| <i>kSDMMC_SignalLineData3</i> | data line |
| <i>kSDMMC_SignalLineData4</i> | data line |
| <i>kSDMMC_SignalLineData5</i> | data line |

*kSDMMC\_SignalLineData6* data line  
*kSDMMC\_SignalLineData7* data line

#### 44.7.4.3 enum sdmmc\_operation\_voltage\_t

Enumerator

*kSDMMC\_OperationVoltageNone* indicate current voltage setting is not setting by suser  
*kSDMMC\_OperationVoltage330V* card operation voltage around 3.3v  
*kSDMMC\_OperationVoltage300V* card operation voltage around 3.0v  
*kSDMMC\_OperationVoltage180V* card operation voltage around 1.8v

#### 44.7.4.4 anonymous enum

Enumerator

*kSDMMC\_BusWidth1Bit* card bus 1 width  
*kSDMMC\_BusWidth4Bit* card bus 4 width  
*kSDMMC\_BusWidth8Bit* card bus 8 width

#### 44.7.4.5 anonymous enum

Enumerator

*kSDMMC\_Support8BitWidth* 8 bit data width capability

#### 44.7.4.6 anonymous enum

Enumerator

*kSDMMC\_DataPacketFormatLSBFirst* usual data packet format LSB first, MSB last  
*kSDMMC\_DataPacketFormatMSBFirst* Wide width data packet format MSB first, LSB last.

#### 44.7.4.7 enum sd\_detect\_card\_type\_t

Enumerator

*kSD\_DetectCardByGpioCD* sd card detect by CD pin through GPIO  
*kSD\_DetectCardByHostCD* sd card detect by CD pin through host  
*kSD\_DetectCardByHostDATA3* sd card detect by DAT3 pin through host

#### 44.7.4.8 anonymous enum

Enumerator

*kSD\_Inserted* card is inserted  
*kSD\_Removed* card is removed

#### 44.7.4.9 anonymous enum

Enumerator

*kSD\_DAT3PullDown* data3 pull down  
*kSD\_DAT3PullUp* data3 pull up

#### 44.7.4.10 enum sd\_io\_voltage\_ctrl\_type\_t

Enumerator

*kSD\_IOVoltageCtrlNotSupport* io voltage control not support  
*kSD\_IOVoltageCtrlByGpio* io voltage control by gpio

#### 44.7.4.11 anonymous enum

Enumerator

*kSDMMC\_R1OutOfRangeFlag* Out of range status bit.  
*kSDMMC\_R1AddressErrorFlag* Address error status bit.  
*kSDMMC\_R1BlockLengthErrorFlag* Block length error status bit.  
*kSDMMC\_R1EraseSequenceErrorFlag* Erase sequence error status bit.  
*kSDMMC\_R1EraseParameterErrorFlag* Erase parameter error status bit.  
*kSDMMC\_R1WriteProtectViolationFlag* Write protection violation status bit.  
*kSDMMC\_R1CardIsLockedFlag* Card locked status bit.  
*kSDMMC\_R1LockUnlockFailedFlag* lock/unlock error status bit  
*kSDMMC\_R1CommandCrcErrorFlag* CRC error status bit.  
*kSDMMC\_R1IllegalCommandFlag* Illegal command status bit.  
*kSDMMC\_R1CardEccFailedFlag* Card ecc error status bit.  
*kSDMMC\_R1CardControllerErrorFlag* Internal card controller error status bit.  
*kSDMMC\_R1ErrorFlag* A general or an unknown error status bit.  
*kSDMMC\_R1CidCsdOverwriteFlag* Cid/csd overwrite status bit.  
*kSDMMC\_R1WriteProtectEraseSkipFlag* Write protection erase skip status bit.  
*kSDMMC\_R1CardEccDisabledFlag* Card ecc disabled status bit.  
*kSDMMC\_R1EraseResetFlag* Erase reset status bit.  
*kSDMMC\_R1ReadyForDataFlag* Ready for data status bit.  
*kSDMMC\_R1SwitchErrorFlag* Switch error status bit.  
*kSDMMC\_R1ApplicationCommandFlag* Application command enabled status bit.

*kSDMMC\_R1AuthenticationSequenceErrorFlag* error in the sequence of authentication process

#### 44.7.4.12 enum sdmmc\_r1\_current\_state\_t

Enumerator

*kSDMMC\_R1StateIdle* R1: current state: idle.  
*kSDMMC\_R1StateReady* R1: current state: ready.  
*kSDMMC\_R1StateIdentify* R1: current state: identification.  
*kSDMMC\_R1StateStandby* R1: current state: standby.  
*kSDMMC\_R1StateTransfer* R1: current state: transfer.  
*kSDMMC\_R1StateSendData* R1: current state: sending data.  
*kSDMMC\_R1StateReceiveData* R1: current state: receiving data.  
*kSDMMC\_R1StateProgram* R1: current state: programming.  
*kSDMMC\_R1StateDisconnect* R1: current state: disconnect.

#### 44.7.4.13 anonymous enum

Enumerator

*kSDSPI\_R1InIdleStateFlag* In idle state.  
*kSDSPI\_R1EraseResetFlag* Erase reset.  
*kSDSPI\_R1IllegalCommandFlag* Illegal command.  
*kSDSPI\_R1CommandCrcErrorFlag* Com crc error.  
*kSDSPI\_R1EraseSequenceErrorFlag* Erase sequence error.  
*kSDSPI\_R1AddressErrorFlag* Address error.  
*kSDSPI\_R1ParameterErrorFlag* Parameter error.

#### 44.7.4.14 anonymous enum

Enumerator

*kSDSPI\_R2CardLockedFlag* Card is locked.  
*kSDSPI\_R2WriteProtectEraseSkip* Write protect erase skip.  
*kSDSPI\_R2LockUnlockFailed* Lock/unlock command failed.  
*kSDSPI\_R2ErrorFlag* Unknown error.  
*kSDSPI\_R2CardControllerErrorFlag* Card controller error.  
*kSDSPI\_R2CardEccFailedFlag* Card ecc failed.  
*kSDSPI\_R2WriteProtectViolationFlag* Write protect violation.  
*kSDSPI\_R2EraseParameterErrorFlag* Erase parameter error.  
*kSDSPI\_R2OutOfRangeFlag* Out of range.  
*kSDSPI\_R2CsdOverwriteFlag* CSD overwrite.

**44.7.4.15 anonymous enum**

Enumerator

*kSDSPI\_DataErrorTokenError* Data error.  
*kSDSPI\_DataErrorTokenCardControllerError* Card controller error.  
*kSDSPI\_DataErrorTokenCardEccFailed* Card ecc error.  
*kSDSPI\_DataErrorTokenOutOfRange* Out of range.

**44.7.4.16 enum sdspi\_data\_token\_t**

Enumerator

*kSDSPI\_DataTokenBlockRead* Single block read, multiple block read.  
*kSDSPI\_DataTokenSingleBlockWrite* Single block write.  
*kSDSPI\_DataTokenMultipleBlockWrite* Multiple block write.  
*kSDSPI\_DataTokenStopTransfer* Stop transmission.

**44.7.4.17 enum sdspi\_data\_response\_token\_t**

Enumerator

*kSDSPI\_DataResponseTokenAccepted* Data accepted.  
*kSDSPI\_DataResponseTokenCrcError* Data rejected due to CRC error.  
*kSDSPI\_DataResponseTokenWriteError* Data rejected due to write error.

**44.7.4.18 enum sd\_command\_t**

Enumerator

*kSD\_SendRelativeAddress* Send Relative Address.  
*kSD\_Switch* Switch Function.  
*kSD\_SendInterfaceCondition* Send Interface Condition.  
*kSD\_VoltageSwitch* Voltage Switch.  
*kSD\_SpeedClassControl* Speed Class control.  
*kSD\_EraseWriteBlockStart* Write Block Start.  
*kSD\_EraseWriteBlockEnd* Write Block End.  
*kSD\_SendTuningBlock* Send Tuning Block.

**44.7.4.19 enum sdspi\_command\_t**

Enumerator

*kSDSPI\_CommandCrc* Command crc protection on/off.

#### 44.7.4.20 enum sd\_application\_command\_t

Enumerator

*kSD\_ApplicationSetBusWidth* Set Bus Width.  
*kSD\_ApplicationStatus* Send SD status.  
*kSD\_ApplicationSendNumberWriteBlocks* Send Number Of Written Blocks.  
*kSD\_ApplicationSetWriteBlockEraseCount* Set Write Block Erase Count.  
*kSD\_ApplicationSendOperationCondition* Send Operation Condition.  
*kSD\_ApplicationSetClearCardDetect* Set Connnect/Disconnect pull up on detect pin.  
*kSD\_ApplicationSendScr* Send Scr.

#### 44.7.4.21 anonymous enum

Enumerator

*kSDMMC\_CommandClassBasic* Card command class 0.  
*kSDMMC\_CommandClassBlockRead* Card command class 2.  
*kSDMMC\_CommandClassBlockWrite* Card command class 4.  
*kSDMMC\_CommandClassErase* Card command class 5.  
*kSDMMC\_CommandClassWriteProtect* Card command class 6.  
*kSDMMC\_CommandClassLockCard* Card command class 7.  
*kSDMMC\_CommandClassApplicationSpecific* Card command class 8.  
*kSDMMC\_CommandClassInputOutputMode* Card command class 9.  
*kSDMMC\_CommandClassSwitch* Card command class 10.

#### 44.7.4.22 anonymous enum

Enumerator

*kSD\_OcrPowerUpBusyFlag* Power up busy status.  
*kSD\_OcrHostCapacitySupportFlag* Card capacity status.  
*kSD\_OcrCardCapacitySupportFlag* Card capacity status.  
*kSD\_OcrSwitch18RequestFlag* Switch to 1.8V request.  
*kSD\_OcrSwitch18AcceptFlag* Switch to 1.8V accepted.  
*kSD\_OcrVdd27\_28Flag* VDD 2.7-2.8.  
*kSD\_OcrVdd28\_29Flag* VDD 2.8-2.9.  
*kSD\_OcrVdd29\_30Flag* VDD 2.9-3.0.  
*kSD\_OcrVdd30\_31Flag* VDD 2.9-3.0.  
*kSD\_OcrVdd31\_32Flag* VDD 3.0-3.1.  
*kSD\_OcrVdd32\_33Flag* VDD 3.1-3.2.  
*kSD\_OcrVdd33\_34Flag* VDD 3.2-3.3.  
*kSD\_OcrVdd34\_35Flag* VDD 3.3-3.4.  
*kSD\_OcrVdd35\_36Flag* VDD 3.4-3.5.

**44.7.4.23 anonymous enum**

Enumerator

*kSD\_SpecificationVersion1\_0* SD card version 1.0-1.01.*kSD\_SpecificationVersion1\_1* SD card version 1.10.*kSD\_SpecificationVersion2\_0* SD card version 2.00.*kSD\_SpecificationVersion3\_0* SD card version 3.0.**44.7.4.24 enum sd\_switch\_mode\_t**

Enumerator

*kSD\_SwitchCheck* SD switch mode 0: check function.*kSD\_SwitchSet* SD switch mode 1: set function.**44.7.4.25 anonymous enum**

Enumerator

*kSD\_CsdReadBlockPartialFlag* Partial blocks for read allowed [79:79].*kSD\_CsdWriteBlockMisalignFlag* Write block misalignment [78:78].*kSD\_CsdReadBlockMisalignFlag* Read block misalignment [77:77].*kSD\_CsdDsrImplementedFlag* DSR implemented [76:76].*kSD\_CsdEraseBlockEnabledFlag* Erase single block enabled [46:46].*kSD\_CsdWriteProtectGroupEnabledFlag* Write protect group enabled [31:31].*kSD\_CsdWriteBlockPartialFlag* Partial blocks for write allowed [21:21].*kSD\_CsdFileFormatGroupFlag* File format group [15:15].*kSD\_CsdCopyFlag* Copy flag [14:14].*kSD\_CsdPermanentWriteProtectFlag* Permanent write protection [13:13].*kSD\_CsdTemporaryWriteProtectFlag* Temporary write protection [12:12].**44.7.4.26 anonymous enum**

Enumerator

*kSD\_ScrDataStatusAfterErase* Data status after erases [55:55].*kSD\_ScrSdSpecification3* Specification version 3.00 or higher [47:47].**44.7.4.27 anonymous enum**

Enumerator

*kSD\_FunctionSDR12Default* SDR12 mode & default.*kSD\_FunctionSDR25HighSpeed* SDR25 & high speed.

*kSD\_FunctionSDR50* SDR50 mode.  
*kSD\_FunctionSDR104* SDR104 mode.  
*kSD\_FunctionDDR50* DDR50 mode.

#### 44.7.4.28 anonymous enum

Enumerator

*kSD\_GroupTimingMode* access mode group  
*kSD\_GroupCommandSystem* command system group  
*kSD\_GroupDriverStrength* driver strength group  
*kSD\_GroupCurrentLimit* current limit group

#### 44.7.4.29 enum sd\_timing\_mode\_t

Enumerator

*kSD\_TimingSDR12DefaultMode* Identification mode & SDR12.  
*kSD\_TimingSDR25HighSpeedMode* High speed mode & SDR25.  
*kSD\_TimingSDR50Mode* SDR50 mode.  
*kSD\_TimingSDR104Mode* SDR104 mode.  
*kSD\_TimingDDR50Mode* DDR50 mode.

#### 44.7.4.30 enum sd\_driver\_strength\_t

Enumerator

*kSD\_DriverStrengthTypeB* default driver strength  
*kSD\_DriverStrengthTypeA* driver strength TYPE A  
*kSD\_DriverStrengthTypeC* driver strength TYPE C  
*kSD\_DriverStrengthTypeD* driver strength TYPE D

#### 44.7.4.31 enum sd\_max\_current\_t

Enumerator

*kSD\_CurrentLimit200MA* default current limit  
*kSD\_CurrentLimit400MA* current limit to 400MA  
*kSD\_CurrentLimit600MA* current limit to 600MA  
*kSD\_CurrentLimit800MA* current limit to 800MA

#### 44.7.4.32 enum sdmmc\_command\_t

Enumerator

*kSDMMC\_GoIdleState* Go Idle State.  
*kSDMMC\_AllSendCid* All Send CID.  
*kSDMMC\_SetDsr* Set DSR.  
*kSDMMC\_SelectCard* Select Card.  
*kSDMMC\_SendCsd* Send CSD.  
*kSDMMC\_SendCid* Send CID.  
*kSDMMC\_StopTransmission* Stop Transmission.  
*kSDMMC\_SendStatus* Send Status.  
*kSDMMC\_GoInactiveState* Go Inactive State.  
*kSDMMC\_SetBlockLength* Set Block Length.  
*kSDMMC\_ReadSingleBlock* Read Single Block.  
*kSDMMC\_ReadMultipleBlock* Read Multiple Block.  
*kSDMMC\_SetBlockCount* Set Block Count.  
*kSDMMC\_WriteSingleBlock* Write Single Block.  
*kSDMMC\_WriteMultipleBlock* Write Multiple Block.  
*kSDMMC\_ProgramCsd* Program CSD.  
*kSDMMC\_SetWriteProtect* Set Write Protect.  
*kSDMMC\_ClearWriteProtect* Clear Write Protect.  
*kSDMMC\_SendWriteProtect* Send Write Protect.  
*kSDMMC\_Erase* Erase.  
*kSDMMC\_LockUnlock* Lock Unlock.  
*kSDMMC\_ApplicationCommand* Send Application Command.  
*kSDMMC\_GeneralCommand* General Purpose Command.  
*kSDMMC\_ReadOcr* Read OCR.

#### 44.7.4.33 anonymous enum

Enumerator

*kSDIO\_RegCCCRSdioVer* CCCR & SDIO version.  
*kSDIO\_RegSDVersion* SD version.  
*kSDIO\_RegIOEnable* io enable register  
*kSDIO\_RegIOReady* io ready register  
*kSDIO\_RegIOIntEnable* io interrupt enable register  
*kSDIO\_RegIOIntPending* io interrupt pending register  
*kSDIO\_RegIOAbort* io abort register  
*kSDIO\_RegBusInterface* bus interface register  
*kSDIO\_RegCardCapability* card capability register  
*kSDIO\_RegCommonCISPointer* common CIS pointer register  
*kSDIO\_RegBusSuspend* bus suspend register  
*kSDIO\_RegFunctionSelect* function select register  
*kSDIO\_RegExecutionFlag* execution flag register

*kSDIO\_RegReadyFlag* ready flag register  
*kSDIO\_RegFN0BlockSizeLow* FN0 block size register.  
*kSDIO\_RegFN0BlockSizeHigh* FN0 block size register.  
*kSDIO\_RegPowerControl* power control register  
*kSDIO\_RegBusSpeed* bus speed register  
*kSDIO\_RegUHSITimingSupport* UHS-I timing support register.  
*kSDIO\_RegDriverStrength* Driver strength register.  
*kSDIO\_RegInterruptExtension* Interrupt extension register.

#### 44.7.4.34 enum sdio\_command\_t

Enumerator

*kSDIO\_SendRelativeAddress* send relative address  
*kSDIO\_SendOperationCondition* send operation condition  
*kSDIO\_SendInterfaceCondition* send interface condition  
*kSDIO\_RWIODirect* read/write IO direct command  
*kSDIO\_RWIOExtended* read/write IO extended command

#### 44.7.4.35 enum sdio\_func\_num\_t

Enumerator

*kSDIO\_FunctionNum0* sdio function0  
*kSDIO\_FunctionNum1* sdio function1  
*kSDIO\_FunctionNum2* sdio function2  
*kSDIO\_FunctionNum3* sdio function3  
*kSDIO\_FunctionNum4* sdio function4  
*kSDIO\_FunctionNum5* sdio function5  
*kSDIO\_FunctionNum6* sdio function6  
*kSDIO\_FunctionNum7* sdio function7  
*kSDIO\_FunctionMemory* for combo card

#### 44.7.4.36 anonymous enum

Enumerator

*kSDIO\_StatusCmdCRCError* the CRC check of the previous cmd fail  
*kSDIO\_StatusIllegalCmd* cmd illegal for the card state  
*kSDIO\_StatusR6Error* special for R6 error status  
*kSDIO\_StatusError* A general or an unknown error occurred.  
*kSDIO\_StatusFunctionNumError* invalid function error  
*kSDIO\_StatusOutOfRange* cmd argument was out of the allowed range

#### 44.7.4.37 anonymous enum

Enumerator

|                                 |                       |
|---------------------------------|-----------------------|
| <i>kSDIO_OcrPowerUpBusyFlag</i> | Power up busy status. |
| <i>kSDIO_OcrIONumber</i>        | number of IO function |
| <i>kSDIO_OcrMemPresent</i>      | memory present flag   |
| <i>kSDIO_OcrVdd20_21Flag</i>    | VDD 2.0-2.1.          |
| <i>kSDIO_OcrVdd21_22Flag</i>    | VDD 2.1-2.2.          |
| <i>kSDIO_OcrVdd22_23Flag</i>    | VDD 2.2-2.3.          |
| <i>kSDIO_OcrVdd23_24Flag</i>    | VDD 2.3-2.4.          |
| <i>kSDIO_OcrVdd24_25Flag</i>    | VDD 2.4-2.5.          |
| <i>kSDIO_OcrVdd25_26Flag</i>    | VDD 2.5-2.6.          |
| <i>kSDIO_OcrVdd26_27Flag</i>    | VDD 2.6-2.7.          |
| <i>kSDIO_OcrVdd27_28Flag</i>    | VDD 2.7-2.8.          |
| <i>kSDIO_OcrVdd28_29Flag</i>    | VDD 2.8-2.9.          |
| <i>kSDIO_OcrVdd29_30Flag</i>    | VDD 2.9-3.0.          |
| <i>kSDIO_OcrVdd30_31Flag</i>    | VDD 2.9-3.0.          |
| <i>kSDIO_OcrVdd31_32Flag</i>    | VDD 3.0-3.1.          |
| <i>kSDIO_OcrVdd32_33Flag</i>    | VDD 3.1-3.2.          |
| <i>kSDIO_OcrVdd33_34Flag</i>    | VDD 3.2-3.3.          |
| <i>kSDIO_OcrVdd34_35Flag</i>    | VDD 3.3-3.4.          |
| <i>kSDIO_OcrVdd35_36Flag</i>    | VDD 3.4-3.5.          |

#### 44.7.4.38 anonymous enum

Enumerator

|                                                  |                                              |
|--------------------------------------------------|----------------------------------------------|
| <i>kSDIO_CCCRSupportDirectCmdDuringDataTrans</i> | support direct cmd during data transfer      |
| <i>kSDIO_CCCRSupportMultiBlock</i>               | support multi block mode                     |
| <i>kSDIO_CCCRSupportReadWait</i>                 | support read wait                            |
| <i>kSDIO_CCCRSupportSuspendResume</i>            | support suspend resume                       |
| <i>kSDIO_CCCRSupportIntDuring4BitDataTrans</i>   | support interrupt during 4-bit data transfer |
| <i>kSDIO_CCCRSupportLowSpeed1Bit</i>             | support low speed 1bit mode                  |
| <i>kSDIO_CCCRSupportLowSpeed4Bit</i>             | support low speed 4bit mode                  |
| <i>kSDIO_CCCRSupportMasterPowerControl</i>       | support master power control                 |
| <i>kSDIO_CCCRSupportHighSpeed</i>                | support high speed                           |
| <i>kSDIO_CCCRSupportContinuousSPIInt</i>         | support continuous SPI interrupt             |

#### 44.7.4.39 anonymous enum

Enumerator

|                                       |                                  |
|---------------------------------------|----------------------------------|
| <i>kSDIO_FBRSupportCSA</i>            | function support CSA             |
| <i>kSDIO_FBRSupportPowerSelection</i> | function support power selection |

**44.7.4.40 enum sdio\_bus\_width\_t**

Enumerator

*kSDIO\_DataBus1Bit* 1 bit bus mode  
*kSDIO\_DataBus4Bit* 4 bit bus mode  
*kSDIO\_DataBus8Bit* 8 bit bus mode

**44.7.4.41 enum mmc\_command\_t**

Enumerator

*kMMC\_SendOperationCondition* Send Operation Condition.  
*kMMC\_SetRelativeAddress* Set Relative Address.  
*kMMC\_SleepAwake* Sleep Awake.  
*kMMC\_Switch* Switch.  
*kMMC\_SendExtendedCsd* Send EXT\_CSD.  
*kMMC\_ReadDataUntilStop* Read Data Until Stop.  
*kMMC\_BusTestRead* Test Read.  
*kMMC\_SendingBusTest* test bus width cmd  
*kMMC\_WriteDataUntilStop* Write Data Until Stop.  
*kMMC\_SendTuningBlock* MMC sending tuning block.  
*kMMC\_ProgramCid* Program CID.  
*kMMC\_EraseGroupStart* Erase Group Start.  
*kMMC\_EraseGroupEnd* Erase Group End.  
*kMMC\_FastInputOutput* Fast IO.  
*kMMC\_GoInterruptState* Go interrupt State.

**44.7.4.42 enum mmc\_classified\_voltage\_t**

Enumerator

*kMMC\_ClassifiedVoltageHigh* High-voltage MMC card.  
*kMMC\_ClassifiedVoltageDual* Dual-voltage MMC card.

**44.7.4.43 enum mmc\_classified\_density\_t**

Enumerator

*kMMC\_ClassifiedDensityWithin2GB* Density byte is less than or equal 2GB.

**44.7.4.44 enum mmc\_access\_mode\_t**

Enumerator

*kMMC\_AccessModeByte* The card should be accessed as byte.*kMMC\_AccessModeSector* The card should be accessed as sector.**44.7.4.45 enum mmc\_voltage\_window\_t**

Enumerator

*kMMC\_VoltageWindowNone* voltage window is not define by user*kMMC\_VoltageWindow120* Voltage window is 1.20V.*kMMC\_VoltageWindow170to195* Voltage window is 1.70V to 1.95V.*kMMC\_VoltageWindows270to360* Voltage window is 2.70V to 3.60V.**44.7.4.46 enum mmc\_csd\_structure\_version\_t**

Enumerator

*kMMC\_CsdStrucureVersion10* CSD version No. 1.0*kMMC\_CsdStrucureVersion11* CSD version No. 1.1*kMMC\_CsdStrucureVersion12* CSD version No. 1.2*kMMC\_CsdStrucureVersionInExtcsd* Version coded in Extended CSD.**44.7.4.47 enum mmc\_specification\_version\_t**

Enumerator

*kMMC\_SpecificationVersion0* Allocated by MMCA.*kMMC\_SpecificationVersion1* Allocated by MMCA.*kMMC\_SpecificationVersion2* Allocated by MMCA.*kMMC\_SpecificationVersion3* Allocated by MMCA.*kMMC\_SpecificationVersion4* Version 4.1/4.2/4.3/4.41-4.5-4.51-5.0.**44.7.4.48 anonymous enum**

Enumerator

*kMMC\_ExtendedCsdRevision10* Revision 1.0.*kMMC\_ExtendedCsdRevision11* Revision 1.1.*kMMC\_ExtendedCsdRevision12* Revision 1.2.*kMMC\_ExtendedCsdRevision13* Revision 1.3 MMC4.3.*kMMC\_ExtendedCsdRevision14* Revision 1.4 obsolete.

- kMMC\_ExtendedCsdRevision15*** Revision 1.5 MMC4.41.
- kMMC\_ExtendedCsdRevision16*** Revision 1.6 MMC4.5.
- kMMC\_ExtendedCsdRevision17*** Revision 1.7 MMC5.0.

#### 44.7.4.49 enum mmc\_command\_set\_t

Enumerator

- kMMC\_CommandSetStandard*** Standard MMC.
- kMMC\_CommandSet1*** Command set 1.
- kMMC\_CommandSet2*** Command set 2.
- kMMC\_CommandSet3*** Command set 3.
- kMMC\_CommandSet4*** Command set 4.

#### 44.7.4.50 anonymous enum

Enumerator

- kMMC\_SupportAlternateBoot*** support alternative boot mode
- kMMC\_SupportDDRBoot*** support DDR boot mode
- kMMC\_SupportHighSpeedBoot*** support high speed boot mode

#### 44.7.4.51 enum mmc\_high\_speed\_timing\_t

Enumerator

- kMMC\_HighSpeedTimingNone*** MMC card using none high-speed timing.
- kMMC\_HighSpeedTiming*** MMC card using high-speed timing.
- kMMC\_HighSpeed200Timing*** MMC card high speed 200 timing.
- kMMC\_HighSpeed400Timing*** MMC card high speed 400 timing.
- kMMC\_EnhanceHighSpeed400Timing*** MMC card high speed 400 timing.

#### 44.7.4.52 enum mmc\_data\_bus\_width\_t

Enumerator

- kMMC\_DataBusWidth1bit*** MMC data bus width is 1 bit.
- kMMC\_DataBusWidth4bit*** MMC data bus width is 4 bits.
- kMMC\_DataBusWidth8bit*** MMC data bus width is 8 bits.
- kMMC\_DataBusWidth4bitDDR*** MMC data bus width is 4 bits ddr.
- kMMC\_DataBusWidth8bitDDR*** MMC data bus width is 8 bits ddr.
- kMMC\_DataBusWidth8bitDDRSTROBE*** MMC data bus width is 8 bits ddr strobe mode.

#### 44.7.4.53 enum mmc\_boot\_partition\_enable\_t

Enumerator

*kMMC\_BootPartitionEnableNot* Device not boot enabled (default)  
*kMMC\_BootPartitionEnablePartition1* Boot partition 1 enabled for boot.  
*kMMC\_BootPartitionEnablePartition2* Boot partition 2 enabled for boot.  
*kMMC\_BootPartitionEnableUserAera* User area enabled for boot.

#### 44.7.4.54 enum mmc\_boot\_timing\_mode\_t

Enumerator

*kMMC\_BootModeSDRWithDefaultTiming* boot mode single data rate with backward compatible timings  
*kMMC\_BootModeSDRWithHighSpeedTiming* boot mode single data rate with high speed timing  
*kMMC\_BootModeDDRTiming* boot mode dual date rate

#### 44.7.4.55 enum mmc\_boot\_partition\_wp\_t

Enumerator

*kMMC\_BootPartitionWPDisable* boot partition write protection disable  
*kMMC\_BootPartitionPwrWPToBothPartition* power on period write protection apply to both boot partitions  
*kMMC\_BootPartitionPermWPToBothPartition* permanent write protection apply to both boot partitions  
*kMMC\_BootPartitionPwrWPToPartition1* power on period write protection apply to partition1  
*kMMC\_BootPartitionPwrWPToPartition2* power on period write protection apply to partition2  
*kMMC\_BootPartitionPermWPToPartition1* permanent write protection apply to partition1  
*kMMC\_BootPartitionPermWPToPartition2* permanent write protection apply to partition2  
*kMMC\_BootPartitionPermWPToPartition1PwrWPToPartition2* permanent write protection apply to partition1, power on period write protection apply to partition2  
*kMMC\_BootPartitionPermWPToPartition2PwrWPToPartition1* permanent write protection apply to partition2, power on period write protection apply to partition1

#### 44.7.4.56 anonymous enum

Enumerator

*kMMC\_BootPartitionNotProtected* boot partition not protected  
*kMMC\_BootPartitionPwrProtected* boot partition is power on period write protected  
*kMMC\_BootPartitionPermProtected* boot partition is permanently protected

#### 44.7.4.57 enum mmc\_access\_partition\_t

Enumerator

*kMMC\_AccessPartitionUserAera* No access to boot partition (default), normal partition.  
*kMMC\_AccessPartitionBoot1* Read/Write boot partition 1.  
*kMMC\_AccessPartitionBoot2* Read/Write boot partition 2.  
*kMMC\_AccessRPMB* Replay protected mem block.  
*kMMC\_AccessGeneralPurposePartition1* access to general purpose partition 1  
*kMMC\_AccessGeneralPurposePartition2* access to general purpose partition 2  
*kMMC\_AccessGeneralPurposePartition3* access to general purpose partition 3  
*kMMC\_AccessGeneralPurposePartition4* access to general purpose partition 4

#### 44.7.4.58 anonymous enum

Enumerator

*kMMC\_CsdReadBlockPartialFlag* Partial blocks for read allowed.  
*kMMC\_CsdWriteBlockMisalignFlag* Write block misalignment.  
*kMMC\_CsdReadBlockMisalignFlag* Read block misalignment.  
*kMMC\_CsdDsrImplementedFlag* DSR implemented.  
*kMMC\_CsdWriteProtectGroupEnabledFlag* Write protect group enabled.  
*kMMC\_CsdWriteBlockPartialFlag* Partial blocks for write allowed.  
*kMMC\_ContentProtectApplicationFlag* Content protect application.  
*kMMC\_CsdFileFormatGroupFlag* File format group.  
*kMMC\_CsdCopyFlag* Copy flag.  
*kMMC\_CsdPermanentWriteProtectFlag* Permanent write protection.  
*kMMC\_CsdTemporaryWriteProtectFlag* Temporary write protection.

#### 44.7.4.59 enum mmc\_extended\_csd\_access\_mode\_t

Enumerator

*kMMC\_ExtendedCsdAccessModeCommandSet* Command set related setting.  
*kMMC\_ExtendedCsdAccessModeSetBits* Set bits in specific byte in Extended CSD.  
*kMMC\_ExtendedCsdAccessModeClearBits* Clear bits in specific byte in Extended CSD.  
*kMMC\_ExtendedCsdAccessModeWriteBits* Write a value to specific byte in Extended CSD.

#### 44.7.4.60 enum mmc\_extended\_csd\_index\_t

Enumerator

*kMMC\_ExtendedCsdIndexFlushCache* flush cache  
*kMMC\_ExtendedCsdIndexCacheControl* cache control

*kMMC\_ExtendedCsdIndexBootPartitionWP* Boot partition write protect.  
*kMMC\_ExtendedCsdIndexEraseGroupDefinition* Erase Group Def.  
*kMMC\_ExtendedCsdIndexBootBusConditions* Boot Bus conditions.  
*kMMC\_ExtendedCsdIndexBootConfigWP* Boot config write protect.  
*kMMC\_ExtendedCsdIndexPartitionConfig* Partition Config, before BOOT\_CONFIG.  
*kMMC\_ExtendedCsdIndexBusWidth* Bus Width.  
*kMMC\_ExtendedCsdIndexHighSpeedTiming* High-speed Timing.  
*kMMC\_ExtendedCsdIndexPowerClass* Power Class.  
*kMMC\_ExtendedCsdIndexCommandSet* Command Set.

#### 44.7.4.61 anonymous enum

Enumerator

*kMMC\_DriverStrength0* Driver type0 ,nominal impedance 50ohm.  
*kMMC\_DriverStrength1* Driver type1 ,nominal impedance 33ohm.  
*kMMC\_DriverStrength2* Driver type2 ,nominal impedance 66ohm.  
*kMMC\_DriverStrength3* Driver type3 ,nominal impedance 100ohm.  
*kMMC\_DriverStrength4* Driver type4 ,nominal impedance 40ohm.

#### 44.7.4.62 enum mmc\_extended\_csd\_flags\_t

Enumerator

*kMMC\_ExtCsdExtPartitionSupport* partitioning support[160]  
*kMMC\_ExtCsdEnhancePartitionSupport* partitioning support[160]  
*kMMC\_ExtCsdPartitioningSupport* partitioning support[160]  
*kMMC\_ExtCsdPrgCIDCSDInDDRModeSupport* CMD26 and CMD27 are support dual data rate [130].  
*kMMC\_ExtCsdBKOpsSupport* background operation feature support [502]  
*kMMC\_ExtCsdDataTagSupport* data tag support[499]  
*kMMC\_ExtCsdModeOperationCodeSupport* mode operation code support[493]

#### 44.7.4.63 enum mmc\_boot\_mode\_t

Enumerator

*kMMC\_BootModeNormal* Normal boot.  
*kMMC\_BootModeAlternative* Alternative boot.

### 44.7.5 Function Documentation

44.7.5.1 **status\_t SDMMC\_SelectCard ( *sdmmchost\_t \* host, uint32\_t relativeAddress, bool isSelected* )**

Parameters

|                        |                                       |
|------------------------|---------------------------------------|
| <i>host</i>            | host handler.                         |
| <i>relativeAddress</i> | Relative address.                     |
| <i>isSelected</i>      | True to put card into transfer state. |

Return values

|                                     |                       |
|-------------------------------------|-----------------------|
| <i>kStatus_SDMMC_TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>              | Operate successfully. |

#### 44.7.5.2 status\_t SDMMC\_SendApplicationCommand ( *sdmmchost\_t \* host, uint32\_t relativeAddress* )

Parameters

|                        |                        |
|------------------------|------------------------|
| <i>host</i>            | host handler.          |
| <i>relativeAddress</i> | Card relative address. |

Return values

|                                     |                       |
|-------------------------------------|-----------------------|
| <i>kStatus_SDMMC_TransferFailed</i> | Transfer failed.      |
| <i>kStatus_SDMMC_CardNotSupport</i> | Card doesn't support. |
| <i>kStatus_Success</i>              | Operate successfully. |

#### 44.7.5.3 status\_t SDMMC\_SetBlockCount ( *sdmmchost\_t \* host, uint32\_t blockCount* )

Parameters

|                   |               |
|-------------------|---------------|
| <i>host</i>       | host handler. |
| <i>blockCount</i> | Block count.  |

Return values

|                                     |                       |
|-------------------------------------|-----------------------|
| <i>kStatus_SDMMC_TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>              | Operate successfully. |

#### 44.7.5.4 status\_t SDMMC\_GoIdle ( *sdmmchost\_t \* host* )

Parameters

|             |               |
|-------------|---------------|
| <i>host</i> | host handler. |
|-------------|---------------|

Return values

|                                     |                       |
|-------------------------------------|-----------------------|
| <i>kStatus_SDMMC_TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>              | Operate successfully. |

#### 44.7.5.5 status\_t SDMMC\_SetBlockSize ( *sdmmchost\_t \* host, uint32\_t blockSize* )

Parameters

|                  |               |
|------------------|---------------|
| <i>host</i>      | host handler. |
| <i>blockSize</i> | Block size.   |

Return values

|                                     |                       |
|-------------------------------------|-----------------------|
| <i>kStatus_SDMMC_TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>              | Operate successfully. |

#### 44.7.5.6 status\_t SDMMC\_SetCardInactive ( *sdmmchost\_t \* host* )

Parameters

|             |               |
|-------------|---------------|
| <i>host</i> | host handler. |
|-------------|---------------|

Return values

|                                      |                       |
|--------------------------------------|-----------------------|
| <i>kStatus_SDMMC_-TransferFailed</i> | Transfer failed.      |
| <i>kStatus_Success</i>               | Operate successfully. |

# Chapter 45

## CODEC Driver

### 45.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

### Modules

- [CODEC Common Driver](#)
- [CODEC I2C Driver](#)
- [CS42888 Driver](#)
- [DA7212 Driver](#)
- [SGTL5000 Driver](#)
- [WM8904 Driver](#)
- [WM8960 Driver](#)

## 45.2 CODEC Common Driver

### 45.2.1 Overview

The codec common driver provides a codec control abstraction interface.

## Modules

- CODEC Adapter
- CS42888 Adapter
- DA7212 Adapter
- SGTL5000 Adapter
- WM8904 Adapter
- WM8960 Adapter

## Data Structures

- struct `codec_config_t`  
*Initialize structure of the codec. [More...](#)*
- struct `codec_capability_t`  
*codec capability [More...](#)*
- struct `codec_handle_t`  
*Codec handle definition. [More...](#)*

## Macros

- #define `CODEC_VOLUME_MAX_VALUE` (100U)  
*codec maximum volume range*

## Enumerations

- enum {
   
  `kStatus_CODEC_NotSupport` = MAKE\_STATUS(kStatusGroup\_CODEC, 0U),
   
  `kStatus_CODEC_DeviceNotRegistered` = MAKE\_STATUS(kStatusGroup\_CODEC, 1U),
   
  `kStatus_CODEC_I2CBusInitialFailed`,
   
  `kStatus_CODEC_I2CCommandTransferFailed` }
   
*CODEC status.*
- enum `codec_audio_protocol_t` {
   
  `kCODEC_BusI2S` = 0U,
   
  `kCODEC_BusLeftJustified` = 1U,
   
  `kCODEC_BusRightJustified` = 2U,
   
  `kCODEC_BusPCMA` = 3U,
   
  `kCODEC_BusPCMB` = 4U,
   
  `kCODEC_BusTDM` = 5U }

*AUDIO format definition.*

- enum {
   
kCODEC\_AudioSampleRate8KHz = 8000U,  
 kCODEC\_AudioSampleRate11025Hz = 11025U,  
 kCODEC\_AudioSampleRate12KHz = 12000U,  
 kCODEC\_AudioSampleRate16KHz = 16000U,  
 kCODEC\_AudioSampleRate22050Hz = 22050U,  
 kCODEC\_AudioSampleRate24KHz = 24000U,  
 kCODEC\_AudioSampleRate32KHz = 32000U,  
 kCODEC\_AudioSampleRate44100Hz = 44100U,  
 kCODEC\_AudioSampleRate48KHz = 48000U,  
 kCODEC\_AudioSampleRate96KHz = 96000U,  
 kCODEC\_AudioSampleRate192KHz = 192000U,  
 kCODEC\_AudioSampleRate384KHz = 384000U }

*audio sample rate definition*

- enum {
   
kCODEC\_AudioBitWidth16bit = 16U,  
 kCODEC\_AudioBitWidth20bit = 20U,  
 kCODEC\_AudioBitWidth24bit = 24U,  
 kCODEC\_AudioBitWidth32bit = 32U }

*audio bit width*

- enum **codec\_module\_t** {
   
kCODEC\_ModuleADC = 0U,  
 kCODEC\_ModuleDAC = 1U,  
 kCODEC\_ModulePGA = 2U,  
 kCODEC\_ModuleHeadphone = 3U,  
 kCODEC\_ModuleSpeaker = 4U,  
 kCODEC\_ModuleLinein = 5U,  
 kCODEC\_ModuleLineout = 6U,  
 kCODEC\_ModuleVref = 7U,  
 kCODEC\_ModuleMicbias = 8U,  
 kCODEC\_ModuleMic = 9U,  
 kCODEC\_ModuleI2SIn = 10U,  
 kCODEC\_ModuleI2SOut = 11U,  
 kCODEC\_ModuleMixer = 12U }

*audio codec module*

- enum **codec\_module\_ctrl\_cmd\_t** { kCODEC\_ModuleSwitchI2SInInterface = 0U }

*audio codec module control cmd*

- enum {
   
kCODEC\_ModuleI2SInInterfacePCM = 0U,  
 kCODEC\_ModuleI2SInInterfaceDSD = 1U }

*audio codec module digital interface*

- enum {

- ```

kCODEC_RecordSourceDifferentialLine = 1U,
kCODEC_RecordSourceLineInput = 2U,
kCODEC_RecordSourceDifferentialMic = 4U,
kCODEC_RecordSourceDigitalMic = 8U,
kCODEC_RecordSourceSingleEndMic = 16U }
    audio codec module record source value
```
- enum {
 

```

kCODEC_RecordChannelLeft1 = 1U,
kCODEC_RecordChannelLeft2 = 2U,
kCODEC_RecordChannelLeft3 = 4U,
kCODEC_RecordChannelRight1 = 1U,
kCODEC_RecordChannelRight2 = 2U,
kCODEC_RecordChannelRight3 = 4U,
kCODEC_RecordChannelDifferentialPositive1 = 1U,
kCODEC_RecordChannelDifferentialPositive2 = 2U,
kCODEC_RecordChannelDifferentialPositive3 = 4U,
kCODEC_RecordChannelDifferentialNegative1 = 8U,
kCODEC_RecordChannelDifferentialNegative2 = 16U,
kCODEC_RecordChannelDifferentialNegative3 = 32U }
```

*audio codec record channel*
- enum {
 

```

kCODEC_PlaySourcePGA = 1U,
kCODEC_PlaySourceInput = 2U,
kCODEC_PlaySourceDAC = 4U,
kCODEC_PlaySourceMixerIn = 1U,
kCODEC_PlaySourceMixerInLeft = 2U,
kCODEC_PlaySourceMixerInRight = 4U,
kCODEC_PlaySourceAux = 8U }
```

*audio codec module play source value*
- enum {
 

```

kCODEC_PlayChannelHeadphoneLeft = 1U,
kCODEC_PlayChannelHeadphoneRight = 2U,
kCODEC_PlayChannelSpeakerLeft = 4U,
kCODEC_PlayChannelSpeakerRight = 8U,
kCODEC_PlayChannelLineOutLeft = 16U,
kCODEC_PlayChannelLineOutRight = 32U,
kCODEC_PlayChannelLeft0 = 1U,
kCODEC_PlayChannelRight0 = 2U,
kCODEC_PlayChannelLeft1 = 4U,
kCODEC_PlayChannelRight1 = 8U,
kCODEC_PlayChannelLeft2 = 16U,
kCODEC_PlayChannelRight2 = 32U,
kCODEC_PlayChannelLeft3 = 64U,
kCODEC_PlayChannelRight3 = 128U }
```

*codec play channel*
- enum {

```
kCODEC_VolumeHeadphoneLeft = 1U,  
kCODEC_VolumeHeadphoneRight = 2U,  
kCODEC_VolumeSpeakerLeft = 4U,  
kCODEC_VolumeSpeakerRight = 8U,  
kCODEC_VolumeLineOutLeft = 16U,  
kCODEC_VolumeLineOutRight = 32U,  
kCODEC_VolumeLeft0 = 1UL << 0U,  
kCODEC_VolumeRight0 = 1UL << 1U,  
kCODEC_VolumeLeft1 = 1UL << 2U,  
kCODEC_VolumeRight1 = 1UL << 3U,  
kCODEC_VolumeLeft2 = 1UL << 4U,  
kCODEC_VolumeRight2 = 1UL << 5U,  
kCODEC_VolumeLeft3 = 1UL << 6U,  
kCODEC_VolumeRight3 = 1UL << 7U,  
kCODEC_VolumeDAC = 1UL << 8U }
```

*codec volume setting*

- enum {

```

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

```

*audio codec capability*

## Functions

- `status_t CODEC_Init (codec_handle_t *handle, codec_config_t *config)`  
*Codec initialization.*
- `status_t CODEC_Deinit (codec_handle_t *handle)`  
*Codec de-initilization.*
- `status_t CODEC_SetFormat (codec_handle_t *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t CODEC_ModuleControl (codec_handle_t *handle, codec_module_ctrl_cmd_t cmd, uint32_t data)`  
*codec module control.*
- `status_t CODEC_SetVolume (codec_handle_t *handle, uint32_t channel, uint32_t volume)`  
*set audio codec pl volume.*
- `status_t CODEC_SetMute (codec_handle_t *handle, uint32_t channel, bool mute)`  
*set audio codec module mute.*
- `status_t CODEC_SetPower (codec_handle_t *handle, codec_module_t module, bool powerOn)`  
*set audio codec power.*
- `status_t CODEC_SetRecord (codec_handle_t *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t CODEC_SetRecordChannel (codec_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t CODEC_SetPlay (codec_handle_t *handle, uint32_t playSource)`  
*codec set play source.*

## Driver version

- `#define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))`  
*CLOCK driver version 2.3.1.*

### 45.2.2 Data Structure Documentation

#### 45.2.2.1 struct codec\_config\_t

##### Data Fields

- `uint32_t codecDevType`  
*codec type*
- `void *codecDevConfig`  
*Codec device specific configuration.*

### 45.2.2.2 struct codec\_capability\_t

#### Data Fields

- `uint32_t codecModuleCapability`  
*codec module capability*
- `uint32_t codecPlayCapability`  
*codec play capability*
- `uint32_t codecRecordCapability`  
*codec record capability*
- `uint32_t codecVolumeCapability`  
*codec volume capability*

### 45.2.2.3 struct \_codec\_handle

codec handle declaration

- Application should allocate a buffer with CODEC\_HANDLE\_SIZE for handle definition, such as `uint8_t codecHandleBuffer[CODEC_HANDLE_SIZE]; codec_handle_t *codecHandle = codecHandleBuffer;`

#### Data Fields

- `codec_config_t * codecConfig`  
*codec configuration function pointer*
- `const codec_capability_t * codecCapability`  
*codec capability*
- `uint8_t codecDevHandle [HAL_CODEC_HANDLER_SIZE]`  
*codec device handle*

### 45.2.3 Macro Definition Documentation

#### 45.2.3.1 #define FSL\_CODEC\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

### 45.2.4 Enumeration Type Documentation

#### 45.2.4.1 anonymous enum

Enumerator

`kStatus_CODEC_NotSupport` CODEC not support status.

`kStatus_CODEC_DeviceNotRegistered` CODEC device register failed status.

`kStatus_CODEC_I2CBusInitialFailed` CODEC i2c bus initialization failed status.

`kStatus_CODEC_I2CCommandTransferFailed` CODEC i2c bus command transfer failed status.

#### 45.2.4.2 enum codec\_audio\_protocol\_t

Enumerator

- kCODEC\_BusI2S* I2S type.
- kCODEC\_BusLeftJustified* Left justified mode.
- kCODEC\_BusRightJustified* Right justified mode.
- kCODEC\_BusPCMA* DSP/PCM A mode.
- kCODEC\_BusPCMB* DSP/PCM B mode.
- kCODEC\_BusTDM* TDM mode.

#### 45.2.4.3 anonymous enum

Enumerator

- kCODEC\_AudioSampleRate8KHz* Sample rate 8000 Hz.
- kCODEC\_AudioSampleRate11025Hz* Sample rate 11025 Hz.
- kCODEC\_AudioSampleRate12KHz* Sample rate 12000 Hz.
- kCODEC\_AudioSampleRate16KHz* Sample rate 16000 Hz.
- kCODEC\_AudioSampleRate22050Hz* Sample rate 22050 Hz.
- kCODEC\_AudioSampleRate24KHz* Sample rate 24000 Hz.
- kCODEC\_AudioSampleRate32KHz* Sample rate 32000 Hz.
- kCODEC\_AudioSampleRate44100Hz* Sample rate 44100 Hz.
- kCODEC\_AudioSampleRate48KHz* Sample rate 48000 Hz.
- kCODEC\_AudioSampleRate96KHz* Sample rate 96000 Hz.
- kCODEC\_AudioSampleRate192KHz* Sample rate 192000 Hz.
- kCODEC\_AudioSampleRate384KHz* Sample rate 384000 Hz.

#### 45.2.4.4 anonymous enum

Enumerator

- kCODEC\_AudioBitWidth16bit* audio bit width 16
- kCODEC\_AudioBitWidth20bit* audio bit width 20
- kCODEC\_AudioBitWidth24bit* audio bit width 24
- kCODEC\_AudioBitWidth32bit* audio bit width 32

#### 45.2.4.5 enum codec\_module\_t

Enumerator

- kCODEC\_ModuleADC* codec module ADC
- kCODEC\_ModuleDAC* codec module DAC
- kCODEC\_ModulePGA* codec module PGA
- kCODEC\_ModuleHeadphone* codec module headphone

*kCODEC\_ModuleSpeaker* codec module speaker  
*kCODEC\_ModuleLinein* codec module linein  
*kCODEC\_ModuleLineout* codec module lineout  
*kCODEC\_ModuleVref* codec module VREF  
*kCODEC\_ModuleMicbias* codec module MIC BIAS  
*kCODEC\_ModuleMic* codec module MIC  
*kCODEC\_ModuleI2SIn* codec module I2S in  
*kCODEC\_ModuleI2SOut* codec module I2S out  
*kCODEC\_ModuleMixer* codec module mixer

#### 45.2.4.6 enum codec\_module\_ctrl\_cmd\_t

Enumerator

*kCODEC\_ModuleSwitchI2SInInterface* module digital interface siwtch.

#### 45.2.4.7 anonymous enum

Enumerator

*kCODEC\_ModuleI2SInInterfacePCM* Pcm interface.  
*kCODEC\_ModuleI2SInInterfaceDSD* DSD interface.

#### 45.2.4.8 anonymous enum

Enumerator

*kCODEC\_RecordSourceDifferentialLine* record source from differential line  
*kCODEC\_RecordSourceLineInput* record source from line input  
*kCODEC\_RecordSourceDifferentialMic* record source from differential mic  
*kCODEC\_RecordSourceDigitalMic* record source from digital microphone  
*kCODEC\_RecordSourceSingleEndMic* record source from single microphone

#### 45.2.4.9 anonymous enum

Enumerator

*kCODEC\_RecordChannelLeft1* left record channel 1  
*kCODEC\_RecordChannelLeft2* left record channel 2  
*kCODEC\_RecordChannelLeft3* left record channel 3  
*kCODEC\_RecordChannelRight1* right record channel 1  
*kCODEC\_RecordChannelRight2* right record channel 2  
*kCODEC\_RecordChannelRight3* right record channel 3  
*kCODEC\_RecordChannelDifferentialPositive1* differential positive record channel 1

*kCODEC\_RecordChannelDifferentialPositive2* differential positive record channel 2  
*kCODEC\_RecordChannelDifferentialPositive3* differential positive record channel 3  
*kCODEC\_RecordChannelDifferentialNegative1* differential negative record channel 1  
*kCODEC\_RecordChannelDifferentialNegative2* differential negative record channel 2  
*kCODEC\_RecordChannelDifferentialNegative3* differential negative record channel 3

#### 45.2.4.10 anonymous enum

Enumerator

*kCODEC\_PlaySourcePGA* play source PGA, bypass ADC  
*kCODEC\_PlaySourceInput* play source Input3  
*kCODEC\_PlaySourceDAC* play source DAC  
*kCODEC\_PlaySourceMixerIn* play source mixer in  
*kCODEC\_PlaySourceMixerInLeft* play source mixer in left  
*kCODEC\_PlaySourceMixerInRight* play source mixer in right  
*kCODEC\_PlaySourceAux* play source mixer in Aux

#### 45.2.4.11 anonymous enum

Enumerator

*kCODEC\_PlayChannelHeadphoneLeft* play channel headphone left  
*kCODEC\_PlayChannelHeadphoneRight* play channel headphone right  
*kCODEC\_PlayChannelSpeakerLeft* play channel speaker left  
*kCODEC\_PlayChannelSpeakerRight* play channel speaker right  
*kCODEC\_PlayChannelLineOutLeft* play channel lineout left  
*kCODEC\_PlayChannelLineOutRight* play channel lineout right  
*kCODEC\_PlayChannelLeft0* play channel left0  
*kCODEC\_PlayChannelRight0* play channel right0  
*kCODEC\_PlayChannelLeft1* play channel left1  
*kCODEC\_PlayChannelRight1* play channel right1  
*kCODEC\_PlayChannelLeft2* play channel left2  
*kCODEC\_PlayChannelRight2* play channel right2  
*kCODEC\_PlayChannelLeft3* play channel left3  
*kCODEC\_PlayChannelRight3* play channel right3

#### 45.2.4.12 anonymous enum

Enumerator

*kCODEC\_VolumeHeadphoneLeft* headphone left volume  
*kCODEC\_VolumeHeadphoneRight* headphone right volume  
*kCODEC\_VolumeSpeakerLeft* speaker left volume  
*kCODEC\_VolumeSpeakerRight* speaker right volume

*kCODEC\_VolumeLineOutLeft* lineout left volume  
*kCODEC\_VolumeLineOutRight* lineout right volume  
*kCODEC\_VolumeLeft0* left0 volume  
*kCODEC\_VolumeRight0* right0 volume  
*kCODEC\_VolumeLeft1* left1 volume  
*kCODEC\_VolumeRight1* right1 volume  
*kCODEC\_VolumeLeft2* left2 volume  
*kCODEC\_VolumeRight2* right2 volume  
*kCODEC\_VolumeLeft3* left3 volume  
*kCODEC\_VolumeRight3* right3 volume  
*kCODEC\_VolumeDAC* dac volume

#### 45.2.4.13 anonymous enum

Enumerator

*kCODEC\_SupportModuleADC* codec capability of module ADC  
*kCODEC\_SupportModuleDAC* codec capability of module DAC  
*kCODEC\_SupportModulePGA* codec capability of module PGA  
*kCODEC\_SupportModuleHeadphone* codec capability of module headphone  
*kCODEC\_SupportModuleSpeaker* codec capability of module speaker  
*kCODEC\_SupportModuleLinein* codec capability of module linein  
*kCODEC\_SupportModuleLineout* codec capability of module lineout  
*kCODEC\_SupportModuleVref* codec capability of module vref  
*kCODEC\_SupportModuleMicbias* codec capability of module mic bias  
*kCODEC\_SupportModuleMic* codec capability of module mic bias  
*kCODEC\_SupportModuleI2SIn* codec capability of module I2S in  
*kCODEC\_SupportModuleI2SOut* codec capability of module I2S out  
*kCODEC\_SupportModuleMixer* codec capability of module mixer  
*kCODEC\_SupportModuleI2SInSwitchInterface* codec capability of module I2S in switch interface  
  
*kCODEC\_SupportPlayChannelLeft0* codec capability of play channel left 0  
*kCODEC\_SupportPlayChannelRight0* codec capability of play channel right 0  
*kCODEC\_SupportPlayChannelLeft1* codec capability of play channel left 1  
*kCODEC\_SupportPlayChannelRight1* codec capability of play channel right 1  
*kCODEC\_SupportPlayChannelLeft2* codec capability of play channel left 2  
*kCODEC\_SupportPlayChannelRight2* codec capability of play channel right 2  
*kCODEC\_SupportPlayChannelLeft3* codec capability of play channel left 3  
*kCODEC\_SupportPlayChannelRight3* codec capability of play channel right 3  
*kCODEC\_SupportPlaySourcePGA* codec capability of set playback source PGA  
*kCODEC\_SupportPlaySourceInput* codec capability of set playback source INPUT  
*kCODEC\_SupportPlaySourceDAC* codec capability of set playback source DAC  
*kCODEC\_SupportPlaySourceMixerIn* codec capability of set play source Mixer in  
*kCODEC\_SupportPlaySourceMixerInLeft* codec capability of set play source Mixer in left  
*kCODEC\_SupportPlaySourceMixerInRight* codec capability of set play source Mixer in right

***kCODEC\_SupportPlaySourceAux*** codec capability of set play source aux

***kCODEC\_SupportRecordSourceDifferentialLine*** codec capability of record source differential line

***kCODEC\_SupportRecordSourceLineInput*** codec capability of record source line input

***kCODEC\_SupportRecordSourceDifferentialMic*** codec capability of record source differential mic

***kCODEC\_SupportRecordSourceDigitalMic*** codec capability of record digital mic

***kCODEC\_SupportRecordSourceSingleEndMic*** codec capability of single end mic

***kCODEC\_SupportRecordChannelLeft1*** left record channel 1

***kCODEC\_SupportRecordChannelLeft2*** left record channel 2

***kCODEC\_SupportRecordChannelLeft3*** left record channel 3

***kCODEC\_SupportRecordChannelRight1*** right record channel 1

***kCODEC\_SupportRecordChannelRight2*** right record channel 2

***kCODEC\_SupportRecordChannelRight3*** right record channel 3

## 45.2.5 Function Documentation

### 45.2.5.1 status\_t CODEC\_Init ( ***codec\_handle\_t \* handle***, ***codec\_config\_t \* config*** )

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configurations.

Returns

kStatus\_Success is success, else de-initial failed.

### 45.2.5.2 status\_t CODEC\_Deinit ( ***codec\_handle\_t \* handle*** )

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

### 45.2.5.3 status\_t CODEC\_SetFormat ( ***codec\_handle\_t \* handle***, ***uint32\_t mclk***, ***uint32\_t sampleRate***, ***uint32\_t bitWidth*** )

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.2.5.4 status\_t CODEC\_ModuleControl ( *codec\_handle\_t \* handle*, *codec\_module\_ctrl\_cmd\_t cmd*, *uint32\_t data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature.

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

#### 45.2.5.5 status\_t CODEC\_SetVolume ( *codec\_handle\_t \* handle*, *uint32\_t channel*, *uint32\_t volume* )

Parameters

<i>handle</i>	codec handle.
<i>channel</i>	audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.2.5.6 status\_t CODEC\_SetMute ( *codec\_handle\_t \* handle*, *uint32\_t channel*, *bool mute* )

Parameters

<i>handle</i>	codec handle.
<i>channel</i>	audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel.
<i>mute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.2.5.7 status\_t CODEC\_SetPower ( *codec\_handle\_t \* handle*, *codec\_module\_t module*, *bool powerOn* )

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.2.5.8 status\_t CODEC\_SetRecord ( *codec\_handle\_t \* handle*, *uint32\_t recordSource* )

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.2.5.9 status\_t CODEC\_SetRecordChannel ( *codec\_handle\_t \* handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel* )

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

#### 45.2.5.10 status\_t CODEC\_SetPlay ( *codec\_handle\_t \* handle, uint32\_t playSource* )

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

## 45.3 CODEC I2C Driver

### 45.3.1 Overview

The codec common driver provides a codec control abstraction interface.

## Data Structures

- struct `codec_i2c_config_t`  
*CODEC I2C configurations structure. [More...](#)*

## Macros

- `#define CODEC_I2C_MASTER_HANDLER_SIZE HAL_I2C_MASTER_HANDLE_SIZE`  
*codec i2c handler*

## Enumerations

- enum `codec_reg_addr_t` {
   
`kCODEC_RegAddr8Bit = 1U,`
  
`kCODEC_RegAddr16Bit = 2U }`
  
*CODEC device register address type.*
- enum `codec_reg_width_t` {
   
`kCODEC_RegWidth8Bit = 1U,`
  
`kCODEC_RegWidth16Bit = 2U,`
  
`kCODEC_RegWidth32Bit = 4U }`
  
*CODEC device register width.*

## Functions

- `status_t CODEC_I2C_Init (void *handle, uint32_t i2cInstance, uint32_t i2cBaudrate, uint32_t i2cSourceClockHz)`
  
*Codec i2c bus initialization.*
- `status_t CODEC_I2C_Deinit (void *handle)`
  
*Codec i2c de-initilization.*
- `status_t CODEC_I2C_Send (void *handle, uint8_t deviceAddress, uint32_t subAddress, uint8_t subaddressSize, uint8_t *txBuff, uint8_t txBuffSize)`
  
*codec i2c send function.*
- `status_t CODEC_I2C_Receive (void *handle, uint8_t deviceAddress, uint32_t subAddress, uint8_t subaddressSize, uint8_t *rxBuff, uint8_t rxBuffSize)`
  
*codec i2c receive function.*

## 45.3.2 Data Structure Documentation

### 45.3.2.1 struct codec\_i2c\_config\_t

#### Data Fields

- uint32\_t `codecI2CInstance`  
*i2c bus instance*
- uint32\_t `codecI2CSourceClock`  
*i2c bus source clock frequency*

## 45.3.3 Enumeration Type Documentation

### 45.3.3.1 enum codec\_reg\_addr\_t

Enumerator

***kCODEC\_RegAddr8Bit*** 8-bit register address.  
***kCODEC\_RegAddr16Bit*** 16-bit register address.

### 45.3.3.2 enum codec\_reg\_width\_t

Enumerator

***kCODEC\_RegWidth8Bit*** 8-bit register width.  
***kCODEC\_RegWidth16Bit*** 16-bit register width.  
***kCODEC\_RegWidth32Bit*** 32-bit register width.

## 45.3.4 Function Documentation

### 45.3.4.1 status\_t CODEC\_I2C\_Init ( void \* *handle*, uint32\_t *i2cInstance*, uint32\_t *i2cBaudrate*, uint32\_t *i2cSourceClockHz* )

Parameters

<i>handle</i>	i2c master handle.
<i>i2cInstance</i>	instance number of the i2c bus, such as 0 is corresponding to I2C0.

<i>i2cBaudrate</i>	i2c baudrate.
<i>i2cSource-ClockHz</i>	i2c source clock frequency.

Returns

kStatus\_HAL\_I2cSuccess is success, else initial failed.

#### 45.3.4.2 status\_t CODEC\_I2C\_Deinit ( void \* *handle* )

Parameters

<i>handle</i>	i2c master handle.
---------------	--------------------

Returns

kStatus\_HAL\_I2cSuccess is success, else deinitial failed.

#### 45.3.4.3 status\_t CODEC\_I2C\_Send ( void \* *handle*, uint8\_t *deviceAddress*, uint32\_t *subAddress*, uint8\_t *subaddressSize*, uint8\_t \* *txBuff*, uint8\_t *txBuffSize* )

Parameters

<i>handle</i>	i2c master handle.
<i>deviceAddress</i>	codec device address.
<i>subAddress</i>	register address.
<i>subaddressSize</i>	register address width.
<i>txBuff</i>	tx buffer pointer.
<i>txBuffSize</i>	tx buffer size.

Returns

kStatus\_HAL\_I2cSuccess is success, else send failed.

#### 45.3.4.4 status\_t CODEC\_I2C\_Receive ( void \* *handle*, uint8\_t *deviceAddress*, uint32\_t *subAddress*, uint8\_t *subaddressSize*, uint8\_t \* *rxBuff*, uint8\_t *rxBuffSize* )

## Parameters

<i>handle</i>	i2c master handle.
<i>deviceAddress</i>	codec device address.
<i>subAddress</i>	register address.
<i>subaddressSize</i>	register address width.
<i>rxBuff</i>	rx buffer pointer.
<i>rxBuffSize</i>	rx buffer size.

## Returns

kStatus\_HAL\_I2cSuccess is success, else receive failed.

## 45.4 CS42888 Driver

### 45.4.1 Overview

The cs42888 driver provides a codec control interface.

### Data Structures

- struct `cs42888_audio_format_t`  
*cs42888 audio format* [More...](#)
- struct `cs42888_config_t`  
*Initialize structure of CS42888.* [More...](#)
- struct `cs42888_handle_t`  
*cs42888 handler* [More...](#)

### Macros

- #define `CS42888_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*CS42888 handle size.*
- #define `CS42888_ID` 0x01U  
*Define the register address of CS42888.*
- #define `CS42888_AOUT_MAX_VOLUME_VALUE` 0xFFU  
*CS42888 volume setting range.*
- #define `CS42888_CACHEREGNUM` 28U  
*Cache register number.*
- #define `CS42888_I2C_ADDR` 0x48U  
*CS42888 I2C address.*
- #define `CS42888_I2C_BITRATE` (100000U)  
*CS42888 I2C baudrate.*

### Typedefs

- typedef void(\* `cs42888_reset` )(bool state)  
*cs42888 reset function pointer*

### Enumerations

- enum `cs42888_func_mode` {
   
`kCS42888_ModeMasterSSM` = 0x0,  
`kCS42888_ModeMasterDSM` = 0x1,  
`kCS42888_ModeMasterQSM` = 0x2,  
`kCS42888_ModeSlave` = 0x3
 }
- CS42888 support modes.*

- enum `cs42888_module_t` {
   
kCS42888\_ModuleDACPair1 = 0x2,  
 kCS42888\_ModuleDACPair2 = 0x4,  
 kCS42888\_ModuleDACPair3 = 0x8,  
 kCS42888\_ModuleDACPair4 = 0x10,  
 kCS42888\_ModuleADCPair1 = 0x20,  
 kCS42888\_ModuleADCPair2 = 0x40 }

*Modules in CS42888 board.*

- enum `cs42888_bus_t` {
   
kCS42888\_BusLeftJustified = 0x0,  
 kCS42888\_BusI2S = 0x1,  
 kCS42888\_BusRightJustified = 0x2,  
 kCS42888\_BusOL1 = 0x4,  
 kCS42888\_BusOL2 = 0x5,  
 kCS42888\_BusTDM = 0x6 }

*CS42888 supported audio bus type.*

- enum {
   
kCS42888\_AOUT1 = 1U,  
 kCS42888\_AOUT2 = 2U,  
 kCS42888\_AOUT3 = 3U,  
 kCS42888\_AOUT4 = 4U,  
 kCS42888\_AOUT5 = 5U,  
 kCS42888\_AOUT6 = 6U,  
 kCS42888\_AOUT7 = 7U,  
 kCS42888\_AOUT8 = 8U }

*CS42888 play channel.*

## Functions

- `status_t CS42888_Init (cs42888_handle_t *handle, cs42888_config_t *config)`  
*CS42888 initialize function.*
- `status_t CS42888_Deinit (cs42888_handle_t *handle)`  
*Deinit the CS42888 codec.*
- `status_t CS42888_SetProtocol (cs42888_handle_t *handle, cs42888_bus_t protocol, uint32_t bitWidth)`  
*Set the audio transfer protocol.*
- `void CS42888_SetFuncMode (cs42888_handle_t *handle, cs42888_func_mode mode)`  
*Set CS42888 to differernt working mode.*
- `status_t CS42888_SelectFunctionalMode (cs42888_handle_t *handle, cs42888_func_mode adcMode, cs42888_func_mode dacMode)`  
*Set CS42888 to differernt functional mode.*
- `status_t CS42888_SetAOUTVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)`  
*Set the volume of different modules in CS42888.*
- `status_t CS42888_SetAINVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)`  
*Set the volume of different modules in CS42888.*
- `uint8_t CS42888_GetAOUTVolume (cs42888_handle_t *handle, uint8_t channel)`

- `uint8_t CS42888_GetAINVolume (cs42888_handle_t *handle, uint8_t channel)`  
*Get the volume of different AIN channel in CS42888.*
- `status_t CS42888_SetMute (cs42888_handle_t *handle, uint8_t channelMask)`  
*Mute modules in CS42888.*
- `status_t CS42888_SetChannelMute (cs42888_handle_t *handle, uint8_t channel, bool isMute)`  
*Mute channel modules in CS42888.*
- `status_t CS42888_SetModule (cs42888_handle_t *handle, cs42888_module_t module, bool isEnabled)`  
*Enable/disable expected devices.*
- `status_t CS42888_ConfigDataFormat (cs42888_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)`  
*Configure the data format of audio data.*
- `status_t CS42888_WriteReg (cs42888_handle_t *handle, uint8_t reg, uint8_t val)`  
*Write register to CS42888 using I2C.*
- `status_t CS42888_ReadReg (cs42888_handle_t *handle, uint8_t reg, uint8_t *val)`  
*Read register from CS42888 using I2C.*
- `status_t CS42888_ModifyReg (cs42888_handle_t *handle, uint8_t reg, uint8_t mask, uint8_t val)`  
*Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`  
*cs42888 driver version 2.1.3.*

### 45.4.2 Data Structure Documentation

#### 45.4.2.1 struct cs42888\_audio\_format\_t

##### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*

#### 45.4.2.2 struct cs42888\_config\_t

##### Data Fields

- `cs42888_bus_t bus`  
*Audio transfer protocol.*
- `cs42888_audio_format_t format`  
*cs42888 audio format*

- `cs42888_func_mode ADCMode`  
*CS42888 ADC function mode.*
- `cs42888_func_mode DACMode`  
*CS42888 DAC function mode.*
- `bool master`  
*true is master, false is slave*
- `codec_i2c_config_t i2cConfig`  
*i2c bus configuration*
- `uint8_t slaveAddress`  
*slave address*
- `cs42888_reset reset`  
*reset function pointer*

## Field Documentation

- (1) `cs42888_func_mode cs42888_config_t::ADCMode`
- (2) `cs42888_func_mode cs42888_config_t::DACMode`

### 45.4.2.3 struct cs42888\_handle\_t

#### Data Fields

- `cs42888_config_t * config`  
*cs42888 config pointer*
- `uint8_t i2cHandle [CS42888_I2C_HANDLER_SIZE]`  
*i2c handle pointer*

### 45.4.3 Macro Definition Documentation

#### 45.4.3.1 #define FSL\_CS42888\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 3))

#### 45.4.3.2 #define CS42888\_ID 0x01U

#### 45.4.3.3 #define CS42888\_I2C\_ADDR 0x48U

### 45.4.4 Enumeration Type Documentation

#### 45.4.4.1 enum cs42888\_func\_mode

Enumerator

- kCS42888\_ModeMasterSSM* master single speed mode
- kCS42888\_ModeMasterDSM* master dual speed mode
- kCS42888\_ModeMasterQSM* master quad speed mode
- kCS42888\_ModeSlave* master single speed mode

#### 45.4.4.2 enum cs42888\_module\_t

Enumerator

- kCS42888\_ModuleDACPair1* DAC pair1 (AOUT1 and AOUT2) module in CS42888.
- kCS42888\_ModuleDACPair2* DAC pair2 (AOUT3 and AOUT4) module in CS42888.
- kCS42888\_ModuleDACPair3* DAC pair3 (AOUT5 and AOUT6) module in CS42888.
- kCS42888\_ModuleDACPair4* DAC pair4 (AOUT7 and AOUT8) module in CS42888.
- kCS42888\_ModuleADCPair1* ADC pair1 (AIN1 and AIN2) module in CS42888.
- kCS42888\_ModuleADCPair2* ADC pair2 (AIN3 and AIN4) module in CS42888.

#### 45.4.4.3 enum cs42888\_bus\_t

Enumerator

- kCS42888\_BusLeftJustified* Left justified format, up to 24 bits.
- kCS42888\_BusI2S* I2S format, up to 24 bits.
- kCS42888\_BusRightJustified* Right justified, can support 16bits and 24 bits.
- kCS42888\_BusOL1* One-Line #1 mode.
- kCS42888\_BusOL2* One-Line #2 mode.
- kCS42888\_BusTDM* TDM mode.

#### 45.4.4.4 anonymous enum

Enumerator

- kCS42888\_AOUT1* aout1
- kCS42888\_AOUT2* aout2
- kCS42888\_AOUT3* aout3
- kCS42888\_AOUT4* aout4
- kCS42888\_AOUT5* aout5
- kCS42888\_AOUT6* aout6
- kCS42888\_AOUT7* aout7
- kCS42888\_AOUT8* aout8

### 45.4.5 Function Documentation

#### 45.4.5.1 status\_t CS42888\_Init ( *cs42888\_handle\_t \* handle*, *cs42888\_config\_t \* config* )

The second parameter is NULL to CS42888 in this version. If users want to change the settings, they have to use *cs42888\_write\_reg()* or *cs42888\_modify\_reg()* to set the register value of CS42888. Note: If the *codec\_config* is NULL, it would initialize CS42888 using default settings. The default setting: *codec\_config->bus* = *kCS42888\_BusI2S* *codec\_config->ADCmode* = *kCS42888\_ModeSlave* *codec\_config->DACmode* = *kCS42888\_ModeSlave*

Parameters

<i>handle</i>	CS42888 handle structure.
<i>config</i>	CS42888 configuration structure.

#### 45.4.5.2 status\_t CS42888\_Deinit ( cs42888\_handle\_t \* *handle* )

This function close all modules in CS42888 to save power.

Parameters

<i>handle</i>	CS42888 handle structure pointer.
---------------	-----------------------------------

#### 45.4.5.3 status\_t CS42888\_SetProtocol ( cs42888\_handle\_t \* *handle*, cs42888\_bus\_t *protocol*, uint32\_t *bitWidth* )

CS42888 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>protocol</i>	Audio data transfer protocol.
<i>bitWidth</i>	bit width

#### 45.4.5.4 void CS42888\_SetFuncMode ( cs42888\_handle\_t \* *handle*, cs42888\_func\_mode *mode* )

**Deprecated** api, Do not use it anymore. It has been superceded by [CS42888\\_SelectFunctionalMode](#).

Parameters

<i>handle</i>	CS42888 handle structure.
<i>mode</i>	differenht working mode of CS42888.

#### 45.4.5.5 status\_t CS42888\_SelectFunctionalMode ( cs42888\_handle\_t \* *handle*, cs42888\_func\_mode *adcMode*, cs42888\_func\_mode *dacMode* )

Parameters

<i>handle</i>	CS42888 handle structure.
<i>adcMode</i>	differenht working mode of CS42888.
<i>dacMode</i>	differenht working mode of CS42888.

#### 45.4.5.6 status\_t CS42888\_SetAOUTVolume ( *cs42888\_handle\_t \* handle, uint8\_t channel, uint8\_t volume* )

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	AOUT channel, it shall be 1~8.
<i>volume</i>	Volume value need to be set.

#### 45.4.5.7 status\_t CS42888\_SetAINVolume ( *cs42888\_handle\_t \* handle, uint8\_t channel, uint8\_t volume* )

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	AIN channel, it shall be 1~4.
<i>volume</i>	Volume value need to be set.

#### 45.4.5.8 uint8\_t CS42888\_GetAOUTVolume ( *cs42888\_handle\_t \* handle, uint8\_t channel* )

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	AOUT channel, it shall be 1~8.

#### 45.4.5.9 `uint8_t CS42888_GetAINVolume ( cs42888_handle_t * handle, uint8_t channel )`

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	AIN channel, it shall be 1~4.

#### 45.4.5.10 `status_t CS42888_SetMute ( cs42888_handle_t * handle, uint8_t channelMask )`

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channelMask</i>	Channel mask for mute. Mute channel 0, it shall be 0x1, while mute channel 0 and 1, it shall be 0x3. Mute all channel, it shall be 0xFF. Each bit represent one channel, 1 means mute, 0 means unmute.

#### 45.4.5.11 `status_t CS42888_SetChannelMute ( cs42888_handle_t * handle, uint8_t channel, bool isMute )`

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	reference _cs42888_play_channel.
<i>isMute</i>	true is mute, falase is unmute.

#### 45.4.5.12 `status_t CS42888_SetModule ( cs42888_handle_t * handle, cs42888_module_t module, bool isEnabled )`

Parameters

<i>handle</i>	CS42888 handle structure.
<i>module</i>	Module expected to enable.
<i>isEnabled</i>	Enable or disable moudles.

#### 45.4.5.13 status\_t CS42888\_ConfigDataFormat ( cs42888\_handle\_t \* *handle*, uint32\_t *mclk*, uint32\_t *sample\_rate*, uint32\_t *bits* )

This function would configure the registers about the sample rate, bit depths.

Parameters

<i>handle</i>	CS42888 handle structure pointer.
<i>mclk</i>	Master clock frequency of I2S.
<i>sample_rate</i>	Sample rate of audio file running in CS42888. CS42888 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
<i>bits</i>	Bit depth of audio file (CS42888 only supports 16bit, 20bit, 24bit and 32 bit in HW).

#### 45.4.5.14 status\_t CS42888\_WriteReg ( cs42888\_handle\_t \* *handle*, uint8\_t *reg*, uint8\_t *val* )

Parameters

<i>handle</i>	CS42888 handle structure.
<i>reg</i>	The register address in CS42888.
<i>val</i>	Value needs to write into the register.

#### 45.4.5.15 status\_t CS42888\_ReadReg ( cs42888\_handle\_t \* *handle*, uint8\_t *reg*, uint8\_t \* *val* )

Parameters

<i>handle</i>	CS42888 handle structure.
<i>reg</i>	The register address in CS42888.
<i>val</i>	Value written to.

**45.4.5.16 status\_t CS42888\_ModifyReg ( *cs42888\_handle\_t \* handle, uint8\_t reg, uint8\_t mask, uint8\_t val* )**

Parameters

<i>handle</i>	CS42888 handle structure.
<i>reg</i>	The register address in CS42888.
<i>mask</i>	The mask code for the bits want to write. The bit you want to write should be 0.
<i>val</i>	Value needs to write into the register.

## 45.4.6 CS42888 Adapter

### 45.4.6.1 Overview

The cs42888 adapter provides a codec unify control interface.

#### Macros

- `#define HAL_CODEC_CS42888_HANDLER_SIZE (CS42888_I2C_HANDLER_SIZE + 4)`  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_CS42888_Init (void *handle, void *config)`  
*Codec initialization.*
- `status_t HAL_CODEC_CS42888_Deinit (void *handle)`  
*Codec de-initialization.*
- `status_t HAL_CODEC_CS42888_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t HAL_CODEC_CS42888_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `status_t HAL_CODEC_CS42888_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `status_t HAL_CODEC_CS42888_SetPower (void *handle, uint32_t module, bool powerOn)`  
*set audio codec module power.*
- `status_t HAL_CODEC_CS42888_SetRecord (void *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t HAL_CODEC_CS42888_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t HAL_CODEC_CS42888_SetPlay (void *handle, uint32_t playSource)`  
*codec set play source.*
- `status_t HAL_CODEC_CS42888_ModuleControl (void *handle, uint32_t cmd, uint32_t data)`  
*codec module control.*
- `static status_t HAL_CODEC_Init (void *handle, void *config)`  
*Codec initialization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initialization.*
- `static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)`

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 45.4.6.2 Function Documentation

##### 45.4.6.2.1 `status_t HAL_CODEC_CS42888_Init( void * handle, void * config )`

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus\_Success is success, else initial failed.

##### 45.4.6.2.2 `status_t HAL_CODEC_CS42888_Deinit( void * handle )`

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

##### 45.4.6.2.3 `status_t HAL_CODEC_CS42888_SetFormat( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.4 status\_t HAL\_CODEC\_CS42888\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.5 status\_t HAL\_CODEC\_CS42888\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.6 status\_t HAL\_CODEC\_CS42888\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.7 status\_t HAL\_CODEC\_CS42888\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.8 status\_t HAL\_CODEC\_CS42888\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.9 status\_t HAL\_CODEC\_CS42888\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.10 status\_t HAL\_CODEC\_CS42888\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus\_Success is success, else initial failed.

#### 45.4.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

#### 45.4.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.4.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

**45.4.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

**45.4.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

## 45.5 DA7212 Driver

### 45.5.1 Overview

The da7212 driver provides a codec control interface.

## Data Structures

- struct [da7212\\_pll\\_config\\_t](#)  
*da7212 pll configuration* [More...](#)
- struct [da7212\\_audio\\_format\\_t](#)  
*da7212 audio format* [More...](#)
- struct [da7212\\_config\\_t](#)  
*DA7212 configure structure.* [More...](#)
- struct [da7212\\_handle\\_t](#)  
*da7212 codec handler* [More...](#)

## Macros

- #define [DA7212\\_I2C\\_HANDLER\\_SIZE](#) CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*da7212 handle size*
- #define [DA7212\\_ADDRESS](#) (0x1A)  
*DA7212 I2C address.*
- #define [DA7212\\_HEADPHONE\\_MAX\\_VOLUME\\_VALUE](#) 0x3FU  
*da7212 volume setting range*

## Enumerations

- enum [da7212\\_Input\\_t](#) {  
 kDA7212\_Input\_AUX = 0x0,  
 kDA7212\_Input\_MIC1\_Dig,  
 kDA7212\_Input\_MIC1\_An,  
 kDA7212\_Input\_MIC2 }  
*DA7212 input source select.*
- enum [\\_da7212\\_play\\_channel](#) {  
 kDA7212\_HeadphoneLeft = 1U,  
 kDA7212\_HeadphoneRight = 2U,  
 kDA7212\_Speaker = 4U }  
*da7212 play channel*
- enum [da7212\\_Output\\_t](#) {  
 kDA7212\_Output\_HP = 0x0,  
 kDA7212\_Output\_SP }  
*DA7212 output device select.*

- enum \_da7212\_module {
 kDA7212\_ModuleADC,
 kDA7212\_ModuleDAC,
 kDA7212\_ModuleHeadphone,
 kDA7212\_ModuleSpeaker }
   
*DA7212 module.*
- enum da7212\_dac\_source\_t {
 kDA7212\_DACSourceADC = 0x0U,
 kDA7212\_DACSourceInputStream = 0x3U }
   
*DA7212 functionality.*
- enum da7212\_volume\_t {
 kDA7212\_DACGainMute = 0x7,
 kDA7212\_DACGainM72DB = 0x17,
 kDA7212\_DACGainM60DB = 0x1F,
 kDA7212\_DACGainM54DB = 0x27,
 kDA7212\_DACGainM48DB = 0x2F,
 kDA7212\_DACGainM42DB = 0x37,
 kDA7212\_DACGainM36DB = 0x3F,
 kDA7212\_DACGainM30DB = 0x47,
 kDA7212\_DACGainM24DB = 0x4F,
 kDA7212\_DACGainM18DB = 0x57,
 kDA7212\_DACGainM12DB = 0x5F,
 kDA7212\_DACGainM6DB = 0x67,
 kDA7212\_DACGain0DB = 0x6F,
 kDA7212\_DACGain6DB = 0x77,
 kDA7212\_DACGain12DB = 0x7F }
   
*DA7212 volume.*
- enum da7212\_protocol\_t {
 kDA7212\_BusI2S = 0x0,
 kDA7212\_BusLeftJustified,
 kDA7212\_BusRightJustified,
 kDA7212\_BusDSPMode }
   
*The audio data transfer protocol choice.*
- enum da7212\_sys\_clk\_source\_t {
 kDA7212\_SysClkSourceMCLK = 0U,
 kDA7212\_SysClkSourcePLL = 1U << 14 }
   
*da7212 system clock source*
- enum da7212\_pll\_clk\_source\_t { kDA7212\_PLLClkSourceMCLK = 0U }
   
*DA7212 pll clock source.*
- enum da7212\_pll\_out\_clk\_t {
 kDA7212\_PLLOutputClk11289600 = 11289600U,
 kDA7212\_PLLOutputClk12288000 = 12288000U }
   
*DA7212 output clock frequency.*
- enum da7212\_master\_bits\_t { }

```

kDA7212_MasterBits32PerFrame = 0U,
kDA7212_MasterBits64PerFrame = 1U,
kDA7212_MasterBits128PerFrame = 2U,
kDA7212_MasterBits256PerFrame = 3U }
    master mode bits per frame

```

## Functions

- `status_t DA7212_Init (da7212_handle_t *handle, da7212_config_t *codecConfig)`  
*DA7212 initialize function.*
- `status_t DA7212_ConfigAudioFormat (da7212_handle_t *handle, uint32_t masterClock_Hz, uint32_t sampleRate_Hz, uint32_t dataBits)`  
*Set DA7212 audio format.*
- `status_t DA7212_SetPLLConfig (da7212_handle_t *handle, da7212_pll_config_t *config)`  
*DA7212 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.*
- `void DA7212_ChangeHPVolume (da7212_handle_t *handle, da7212_volume_t volume)`  
*Set DA7212 playback volume.*
- `void DA7212_Mute (da7212_handle_t *handle, bool isMuted)`  
*Mute or unmute DA7212.*
- `void DA7212_ChangeInput (da7212_handle_t *handle, da7212_Input_t DA7212_Input)`  
*Set the input data source of DA7212.*
- `void DA7212_ChangeOutput (da7212_handle_t *handle, da7212_Output_t DA7212_Output)`  
*Set the output device of DA7212.*
- `status_t DA7212_SetChannelVolume (da7212_handle_t *handle, uint32_t channel, uint32_t volume)`  
*Set module volume.*
- `status_t DA7212_SetChannelMute (da7212_handle_t *handle, uint32_t channel, bool isMute)`  
*Set module mute.*
- `status_t DA7212_SetProtocol (da7212_handle_t *handle, da7212_protocol_t protocol)`  
*Set protocol for DA7212.*
- `status_t DA7212_SetMasterModeBits (da7212_handle_t *handle, uint32_t bitWidth)`  
*Set master mode bits per frame for DA7212.*
- `status_t DA7212_WriteRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t u8RegisterData)`  
*Write a register for DA7212.*
- `status_t DA7212_ReadRegister (da7212_handle_t *handle, uint8_t u8Register, uint8_t *pu8RegisterData)`  
*Get a register value of DA7212.*
- `status_t DA7212_Deinit (da7212_handle_t *handle)`  
*Deinit DA7212.*

## Driver version

- `#define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))`  
*CLOCK driver version 2.2.3.*

## 45.5.2 Data Structure Documentation

### 45.5.2.1 struct da7212\_pll\_config\_t

#### Data Fields

- `da7212_pll_clk_source_t source`  
*pll reference clock source*
- `uint32_t refClock_HZ`  
*pll reference clock frequency*
- `da7212_pll_out_clk_t outputClock_HZ`  
*pll output clock frequency*

### 45.5.2.2 struct da7212\_audio\_format\_t

#### Data Fields

- `uint32_t mclk_HZ`  
*master clock frequency*
- `uint32_t sampleRate`  
*sample rate*
- `uint32_t bitWidth`  
*bit width*
- `bool isBclkInvert`  
*bit clock invertet*

### 45.5.2.3 struct da7212\_config\_t

#### Data Fields

- `bool isMaster`  
*If DA7212 is master, true means master, false means slave.*
- `da7212_protocol_t protocol`  
*Audio bus format, can be I2S, LJ, RJ or DSP mode.*
- `da7212_dac_source_t dacSource`  
*DA7212 data source.*
- `da7212_audio_format_t format`  
*audio format*
- `uint8_t slaveAddress`  
*device address*
- `codec_i2c_config_t i2cConfig`  
*i2c configuration*
- `da7212_sys_clk_source_t sysClkSource`  
*system clock source*
- `da7212_pll_config_t * pll`  
*pll configuration*

#### Field Documentation

- (1) `bool da7212_config_t::isMaster`
- (2) `da7212_protocol_t da7212_config_t::protocol`
- (3) `da7212_dac_source_t da7212_config_t::dacSource`

#### 45.5.2.4 struct da7212\_handle\_t

##### Data Fields

- `da7212_config_t * config`  
*da7212 config pointer*
- `uint8_t i2cHandle [DA7212_I2C_HANDLER_SIZE]`  
*i2c handle*

#### 45.5.3 Macro Definition Documentation

##### 45.5.3.1 #define FSL\_DA7212\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 3))

#### 45.5.4 Enumeration Type Documentation

##### 45.5.4.1 enum da7212\_Input\_t

Enumerator

- kDA7212\_Input\_AUX* Input from AUX.
- kDA7212\_Input\_MIC1\_Dig* Input from MIC1 Digital.
- kDA7212\_Input\_MIC1\_An* Input from Mic1 Analog.
- kDA7212\_Input\_MIC2* Input from MIC2.

##### 45.5.4.2 enum \_da7212\_play\_channel

Enumerator

- kDA7212\_HeadphoneLeft* headphone left
- kDA7212\_HeadphoneRight* headphone right
- kDA7212\_Speaker* speaker channel

##### 45.5.4.3 enum da7212\_Output\_t

Enumerator

- kDA7212\_Output\_HP* Output to headphone.
- kDA7212\_Output\_SP* Output to speaker.

#### 45.5.4.4 enum \_da7212\_module

Enumerator

*kDA7212\_ModuleADC* module ADC  
*kDA7212\_ModuleDAC* module DAC  
*kDA7212\_ModuleHeadphone* module headphone  
*kDA7212\_ModuleSpeaker* module speaker

#### 45.5.4.5 enum da7212\_dac\_source\_t

Enumerator

*kDA7212\_DACSourceADC* DAC source from ADC.  
*kDA7212\_DACSourceInputStream* DAC source from.

#### 45.5.4.6 enum da7212\_volume\_t

Enumerator

*kDA7212\_DACGainMute* Mute DAC.  
*kDA7212\_DACGainM72DB* DAC volume -72db.  
*kDA7212\_DACGainM60DB* DAC volume -60db.  
*kDA7212\_DACGainM54DB* DAC volume -54db.  
*kDA7212\_DACGainM48DB* DAC volume -48db.  
*kDA7212\_DACGainM42DB* DAC volume -42db.  
*kDA7212\_DACGainM36DB* DAC volume -36db.  
*kDA7212\_DACGainM30DB* DAC volume -30db.  
*kDA7212\_DACGainM24DB* DAC volume -24db.  
*kDA7212\_DACGainM18DB* DAC volume -18db.  
*kDA7212\_DACGainM12DB* DAC volume -12db.  
*kDA7212\_DACGainM6DB* DAC volume -6db.  
*kDA7212\_DACGain0DB* DAC volume +0db.  
*kDA7212\_DACGain6DB* DAC volume +6db.  
*kDA7212\_DACGain12DB* DAC volume +12db.

#### 45.5.4.7 enum da7212\_protocol\_t

Enumerator

*kDA7212\_BusI2S* I2S Type.  
*kDA7212\_BusLeftJustified* Left justified.  
*kDA7212\_BusRightJustified* Right Justified.  
*kDA7212\_BusDSPMode* DSP mode.

#### 45.5.4.8 enum da7212\_sys\_clk\_source\_t

Enumerator

*kDA7212\_SysClkSourceMCLK* da7212 system clock soure from MCLK

*kDA7212\_SysClkSourcePLL* da7212 system clock soure from pLL

#### 45.5.4.9 enum da7212\_pll\_clk\_source\_t

Enumerator

*kDA7212\_PLLClkSourceMCLK* DA7212 PLL clock source from MCLK.

#### 45.5.4.10 enum da7212\_pll\_out\_clk\_t

Enumerator

*kDA7212\_PLLOutputClk11289600* output 112896000U

*kDA7212\_PLLOutputClk12288000* output 12288000U

#### 45.5.4.11 enum da7212\_master\_bits\_t

Enumerator

*kDA7212\_MasterBits32PerFrame* master mode bits32 per frame

*kDA7212\_MasterBits64PerFrame* master mode bits64 per frame

*kDA7212\_MasterBits128PerFrame* master mode bits128 per frame

*kDA7212\_MasterBits256PerFrame* master mode bits256 per frame

### 45.5.5 Function Documentation

#### 45.5.5.1 status\_t DA7212\_Init ( da7212\_handle\_t \* *handle*, da7212\_config\_t \* *codecConfig* )

Parameters

<i>handle</i>	DA7212 handle pointer.
---------------	------------------------

<i>codecConfig</i>	Codec configure structure. This parameter can be NULL, if NULL, set as default settings. The default setting:  <pre>* sgtl_init_t codec_config * codec_config.route = kDA7212_RoutePlayback * codec_config.bus = <b>kDA7212_BusI2S</b> * codec_config.isMaster = <b>false</b> *</pre>
--------------------	---

#### 45.5.5.2 status\_t DA7212\_ConfigAudioFormat ( *da7212\_handle\_t \* handle, uint32\_t masterClock\_Hz, uint32\_t sampleRate\_Hz, uint32\_t dataBits* )

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>masterClock_Hz</i>	Master clock frequency in Hz. If DA7212 is slave, use the frequency of master, if DA7212 as master, it should be 1228000 while sample rate frequency is 8k/12K/16-K/24K/32K/48K/96K, 11289600 whie sample rate is 11.025K/22.05K/44.1K
<i>sampleRate_Hz</i>	Sample rate frequency in Hz.
<i>dataBits</i>	How many bits in a word of a audio frame, DA7212 only supports 16/20/24/32 bits.

#### 45.5.5.3 status\_t DA7212\_SetPLLConfig ( *da7212\_handle\_t \* handle, da7212\_pll\_config\_t \* config* )

Parameters

<i>handle</i>	DA7212 handler pointer.
<i>config</i>	PLL configuration pointer.

#### 45.5.5.4 void DA7212\_ChangeHPVolume ( *da7212\_handle\_t \* handle, da7212\_volume\_t volume* )

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>volume</i>	The volume of playback.

**45.5.5.5 void DA7212\_Mute ( da7212\_handle\_t \* *handle*, bool *isMuted* )**

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>isMuted</i>	True means mute, false means unmute.

**45.5.5.6 void DA7212\_ChangeInput ( da7212\_handle\_t \* *handle*, da7212\_Input\_t *DA7212\_Input* )**

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>DA7212_Input</i>	Input data source.

**45.5.5.7 void DA7212\_ChangeOutput ( da7212\_handle\_t \* *handle*, da7212\_Output\_t *DA7212\_Output* )**

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>DA7212_Output</i>	Output device of DA7212.

**45.5.5.8 status\_t DA7212\_SetChannelVolume ( da7212\_handle\_t \* *handle*, uint32\_t *channel*, uint32\_t *volume* )**

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>channel</i>	shoule be a value of _da7212_channel.
<i>volume</i>	volume range 0 - 0x3F mapped to range -57dB - 6dB.

#### 45.5.5.9 status\_t DA7212\_SetChannelMute ( *da7212\_handle\_t \* handle, uint32\_t channel, bool isMute* )

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>channel</i>	shoule be a value of _da7212_channel.
<i>isMute</i>	true is mute, false is unmute.

#### 45.5.5.10 status\_t DA7212\_SetProtocol ( *da7212\_handle\_t \* handle, da7212\_protocol\_t protocol* )

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>protocol</i>	da7212_protocol_t.

#### 45.5.5.11 status\_t DA7212\_SetMasterModeBits ( *da7212\_handle\_t \* handle, uint32\_t bitWidth* )

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>bitWidth</i>	audio data bitwidth.

#### 45.5.5.12 status\_t DA7212\_WriteRegister ( *da7212\_handle\_t \* handle, uint8\_t u8Register, uint8\_t u8RegisterData* )

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>u8Register</i>	DA7212 register address to be written.
<i>u8RegisterData</i>	Data to be written into register

#### 45.5.5.13 status\_t DA7212\_ReadRegister ( da7212\_handle\_t \* *handle*, uint8\_t *u8Register*, uint8\_t \* *pu8RegisterData* )

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>u8Register</i>	DA7212 register address to be read.
<i>pu8RegisterData</i>	Pointer where the read out value to be stored.

#### 45.5.5.14 status\_t DA7212\_Deinit ( da7212\_handle\_t \* *handle* )

Parameters

<i>handle</i>	DA7212 handle pointer.
---------------	------------------------

## 45.5.6 DA7212 Adapter

### 45.5.6.1 Overview

The da7212 adapter provides a codec unify control interface.

#### Macros

- `#define HAL_CODEC_DA7212_HANDLER_SIZE (DA7212_I2C_HANDLER_SIZE + 4)`  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_DA7212_Init (void *handle, void *config)`  
*Codec initialization.*
- `status_t HAL_CODEC_DA7212_Deinit (void *handle)`  
*Codec de-initilization.*
- `status_t HAL_CODEC_DA7212_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t HAL_CODEC_DA7212_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `status_t HAL_CODEC_DA7212_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `status_t HAL_CODEC_DA7212_SetPower (void *handle, uint32_t module, bool powerOn)`  
*set audio codec module power.*
- `status_t HAL_CODEC_DA7212_SetRecord (void *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t HAL_CODEC_DA7212_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t HAL_CODEC_DA7212_SetPlay (void *handle, uint32_t playSource)`  
*codec set play source.*
- `status_t HAL_CODEC_DA7212_ModuleControl (void *handle, uint32_t cmd, uint32_t data)`  
*codec module control.*
- `static status_t HAL_CODEC_Init (void *handle, void *config)`  
*Codec initilization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initilization.*
- `static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)`

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 45.5.6.2 Function Documentation

##### 45.5.6.2.1 `status_t HAL_CODEC_DA7212_Init( void * handle, void * config )`

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus\_Success is success, else initial failed.

##### 45.5.6.2.2 `status_t HAL_CODEC_DA7212_Deinit( void * handle )`

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

##### 45.5.6.2.3 `status_t HAL_CODEC_DA7212_SetFormat( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.4 status\_t HAL\_CODEC\_DA7212\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.5 status\_t HAL\_CODEC\_DA7212\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.6 status\_t HAL\_CODEC\_DA7212\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.7 status\_t HAL\_CODEC\_DA7212\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.8 status\_t HAL\_CODEC\_DA7212\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.9 status\_t HAL\_CODEC\_DA7212\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.10 status\_t HAL\_CODEC\_DA7212\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus\_Success is success, else initial failed.

#### 45.5.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

#### 45.5.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.5.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

**45.5.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

**45.5.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

## 45.6 SGTL5000 Driver

### 45.6.1 Overview

The sgtl5000 driver provides a codec control interface.

### Data Structures

- struct `sgtl_audio_format_t`  
*Audio format configuration. [More...](#)*
- struct `sgtl_config_t`  
*Initialize structure of sgtl5000. [More...](#)*
- struct `sgtl_handle_t`  
*SGTL codec handler. [More...](#)*

### Macros

- #define `CHIP_ID` 0x0000U  
*Define the register address of sgtl5000.*
- #define `SGTL5000_HEADPHONE_MAX_VOLUME_VALUE` 0x7FU  
*SGTL5000 volume setting range.*
- #define `SGTL5000_I2C_ADDR` 0x0A  
*SGTL5000 I2C address.*
- #define `SGTL_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*sgtl handle size*
- #define `SGTL_I2C_BITRATE` 100000U  
*sgtl i2c baudrate*

### Enumerations

- enum `sgtl_module_t` {
   
`kSGTL_ModuleADC` = 0x0,  
`kSGTL_ModuleDAC`,  
`kSGTL_ModuleDAP`,  
`kSGTL_ModuleHP`,  
`kSGTL_ModuleI2SIN`,  
`kSGTL_ModuleI2SOUT`,  
`kSGTL_ModuleLineIn`,  
`kSGTL_ModuleLineOut`,  
`kSGTL_ModuleMicin` }
- Modules in Sglt5000 board.*
- enum `sgtl_route_t` {

```

kSGTL_RouteBypass = 0x0,
kSGTL_RoutePlayback,
kSGTL_RoutePlaybackandRecord,
kSGTL_RoutePlaybackwithDAP,
kSGTL_RoutePlaybackwithDAPandRecord,
kSGTL_RouteRecord }

Sgtl5000 data route.
• enum sgtl_protocol_t {
    kSGTL_BusI2S = 0x0,
    kSGTL_BusLeftJustified,
    kSGTL_BusRightJustified,
    kSGTL_BusPCMA,
    kSGTL_BusPCMB }

The audio data transfer protocol choice.
• enum {
    kSGTL_HeadphoneLeft = 0,
    kSGTL_HeadphoneRight = 1,
    kSGTL_LineoutLeft = 2,
    kSGTL_LineoutRight = 3 }

sgtl play channel
• enum {
    kSGTL_RecordSourceLineIn = 0U,
    kSGTL_RecordSourceMic = 1U }

sgtl record source _sgtl_record_source
• enum {
    kSGTL_PlaySourceLineIn = 0U,
    kSGTL_PlaySourceDAC = 1U }

sgtl play source _stgl_play_source
• enum sgtl_sclk_edge_t {
    kSGTL_SclkValidEdgeRising = 0U,
    kSGTL_SclkValidEdgeFailling = 1U }

SGTL SCLK valid edge.

```

## Functions

- `status_t SGTL_Init (sgtl_handle_t *handle, sgtl_config_t *config)`  
*sgtl5000 initialize function.*
- `status_t SGTL_SetDataRoute (sgtl_handle_t *handle, sgtl_route_t route)`  
*Set audio data route in sgtl5000.*
- `status_t SGTL_SetProtocol (sgtl_handle_t *handle, sgtl_protocol_t protocol)`  
*Set the audio transfer protocol.*
- `void SGTL_SetMasterSlave (sgtl_handle_t *handle, bool master)`  
*Set sgtl5000 as master or slave.*
- `status_t SGTL_SetVolume (sgtl_handle_t *handle, sgtl_module_t module, uint32_t volume)`  
*Set the volume of different modules in sgtl5000.*
- `uint32_t SGTL_GetVolume (sgtl_handle_t *handle, sgtl_module_t module)`  
*Get the volume of different modules in sgtl5000.*

- `status_t SGTL_SetMute (sgtl_handle_t *handle, sgtl_module_t module, bool mute)`  
*Mute/unmute modules in sgtl5000.*
- `status_t SGTL_EnableModule (sgtl_handle_t *handle, sgtl_module_t module)`  
*Enable expected devices.*
- `status_t SGTL_DisableModule (sgtl_handle_t *handle, sgtl_module_t module)`  
*Disable expected devices.*
- `status_t SGTL_Deinit (sgtl_handle_t *handle)`  
*Deinit the sgtl5000 codec.*
- `status_t SGTL_ConfigDataFormat (sgtl_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)`  
*Configure the data format of audio data.*
- `status_t SGTL_SetPlay (sgtl_handle_t *handle, uint32_t playSource)`  
*select SGTL codec play source.*
- `status_t SGTL_SetRecord (sgtl_handle_t *handle, uint32_t recordSource)`  
*select SGTL codec record source.*
- `status_t SGTL_WriteReg (sgtl_handle_t *handle, uint16_t reg, uint16_t val)`  
*Write register to sgtl using I2C.*
- `status_t SGTL_ReadReg (sgtl_handle_t *handle, uint16_t reg, uint16_t *val)`  
*Read register from sgtl using I2C.*
- `status_t SGTL_ModifyReg (sgtl_handle_t *handle, uint16_t reg, uint16_t clr_mask, uint16_t val)`  
*Modify some bits in the register using I2C.*

## Driver version

- `#define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`  
*CLOCK driver version 2.1.1.*

### 45.6.2 Data Structure Documentation

#### 45.6.2.1 struct sgtl\_audio\_format\_t

##### Data Fields

- `uint32_t mclk_HZ`  
*master clock*
- `uint32_t sampleRate`  
*Sample rate.*
- `uint32_t bitWidth`  
*Bit width.*
- `sgtl_sclk_edge_t sclkEdge`  
*sclk valid edge*

#### 45.6.2.2 struct sgtl\_config\_t

##### Data Fields

- `sgtl_route_t route`

- *Audio data route.*
- `sgtl_protocol_t bus`  
*Audio transfer protocol.*
- `bool master_slave`  
*Master or slave.*
- `sgtl_audio_format_t format`  
*audio format*
- `uint8_t slaveAddress`  
*code device slave address*
- `codec_i2c_config_t i2cConfig`  
*i2c bus configuration*

## Field Documentation

- (1) `sgtl_route_t sgtl_config_t::route`
- (2) `bool sgtl_config_t::master_slave`

True means master, false means slave.

## 45.6.2.3 struct sgtl\_handle\_t

### Data Fields

- `sgtl_config_t * config`  
*sgtl config pointer*
- `uint8_t i2cHandle [SGTL_I2C_HANDLER_SIZE]`  
*i2c handle*

## 45.6.3 Macro Definition Documentation

### 45.6.3.1 #define FSL\_SGTL5000\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

### 45.6.3.2 #define CHIP\_ID 0x0000U

### 45.6.3.3 #define SGTL5000\_I2C\_ADDR 0x0A

## 45.6.4 Enumeration Type Documentation

### 45.6.4.1 enum sgtl\_module\_t

Enumerator

- `kSGTL_ModuleADC` ADC module in SGTL5000.
- `kSGTL_ModuleDAC` DAC module in SGTL5000.
- `kSGTL_ModuleDAP` DAP module in SGTL5000.
- `kSGTL_ModuleHP` Headphone module in SGTL5000.

*kSGTL\_ModuleI2SIN* I2S-IN module in SGTL5000.  
*kSGTL\_ModuleI2SOUT* I2S-OUT module in SGTL5000.  
*kSGTL\_ModuleLineIn* Line-in moudle in SGTL5000.  
*kSGTL\_ModuleLineOut* Line-out module in SGTL5000.  
*kSGTL\_ModuleMicin* Micphone module in SGTL5000.

#### 45.6.4.2 enum sgtl\_route\_t

Note

Only provide some typical data route, not all route listed. Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

*kSGTL\_RouteBypass* LINEIN->Headphone.  
*kSGTL\_RoutePlayback* I2SIN->DAC->Headphone.  
*kSGTL\_RoutePlaybackandRecord* I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.  
*kSGTL\_RoutePlaybackwithDAP* I2SIN->DAP->DAC->Headphone.  
*kSGTL\_RoutePlaybackwithDAPandRecord* I2SIN->DAP->DAC->HP, LINEIN->ADC->I2SOUT.  
*kSGTL\_RouteRecord* LINEIN->ADC->I2SOUT.

#### 45.6.4.3 enum sgtl\_protocol\_t

Sgtl5000 only supports I2S format and PCM format.

Enumerator

*kSGTL\_BusI2S* I2S Type.  
*kSGTL\_BusLeftJustified* Left justified.  
*kSGTL\_BusRightJustified* Right Justified.  
*kSGTL\_BusPCMA* PCMA.  
*kSGTL\_BusPCMB* PCMB.

#### 45.6.4.4 anonymous enum

Enumerator

*kSGTL\_HeadphoneLeft* headphone left channel  
*kSGTL\_HeadphoneRight* headphone right channel  
*kSGTL\_LineoutLeft* lineout left channel  
*kSGTL\_LineoutRight* lineout right channel

#### 45.6.4.5 anonymous enum

Enumerator

*kSGTL\_RecordSourceLineIn* record source line in  
*kSGTL\_RecordSourceMic* record source single end

#### 45.6.4.6 anonymous enum

Enumerator

*kSGTL\_PlaySourceLineIn* play source line in  
*kSGTL\_PlaySourceDAC* play source line in

#### 45.6.4.7 enum sgtl\_sclk\_edge\_t

Enumerator

*kSGTL\_SclkValidEdgeRising* SCLK valid edge.  
*kSGTL\_SclkValidEdgeFailling* SCLK failling edge.

### 45.6.5 Function Documentation

#### 45.6.5.1 status\_t SGTL\_Init( sgtl\_handle\_t \* handle, sgtl\_config\_t \* config )

This function calls SGTL\_I2CInit(), and in this function, some configurations are fixed. The second parameter can be NULL. If users want to change the SGTL5000 settings, a configure structure should be prepared.

Note

If the codec\_config is NULL, it would initialize sgtl5000 using default settings. The default setting:

```
* sgtl_init_t codec_config
* codec_config.route = kSGTL_RoutePlaybackandRecord
* codec_config.bus = kSGTL_BusI2S
* codec_config.master = slave
*
```

Parameters

---

<i>handle</i>	Sgtl5000 handle structure.
<i>config</i>	sgtl5000 configuration structure. If this pointer equals to NULL, it means using the default configuration.

Returns

Initialization status

#### 45.6.5.2 status\_t SGTL\_SetDataRoute ( sgtl\_handle\_t \* *handle*, sgtl\_route\_t *route* )

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules.

Note

If a new route is set, the previous route would not work.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>route</i>	Audio data route in sgtl5000.

#### 45.6.5.3 status\_t SGTL\_SetProtocol ( sgtl\_handle\_t \* *handle*, sgtl\_protocol\_t *protocol* )

Sgtl5000 only supports I2S, I2S left, I2S right, PCM A, PCM B format.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>protocol</i>	Audio data transfer protocol.

#### 45.6.5.4 void SGTL\_SetMasterSlave ( sgtl\_handle\_t \* *handle*, bool *master* )

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>master</i>	1 represent master, 0 represent slave.

#### 45.6.5.5 status\_t SGTL\_SetVolume ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module*, *uint32\_t volume* )

This function would set the volume of sgtl5000 modules. This interface set module volume. The function assume that left channel and right channel has the same volume.

kSGTL\_ModuleADC volume range: 0 - 0xF, 0dB - 22.5dB  
kSGTL\_ModuleDAC volume range: 0x3C - 0xF0, 0dB - -90dB  
kSGTL\_ModuleHP volume range: 0 - 0x7F, 12dB - -51.5dB  
kSGTL\_ModuleLineOut volume range: 0 - 0x1F, 0.5dB steps

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.
<i>volume</i>	Volume value need to be set. The value is the exact value in register.

#### 45.6.5.6 uint32\_t SGTL\_GetVolume ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module* )

This function gets the volume of sgtl5000 modules. This interface get DAC module volume. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.

Returns

Module value, the value is exact value in register.

#### 45.6.5.7 status\_t SGTL\_SetMute ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module*, *bool mute* )

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.
<i>mute</i>	True means mute, and false means unmute.

#### 45.6.5.8 status\_t SGTL\_EnableModule ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module* )

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Module expected to enable.

#### 45.6.5.9 status\_t SGTL\_DisableModule ( *sgtl\_handle\_t \* handle*, *sgtl\_module\_t module* )

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Module expected to enable.

#### 45.6.5.10 status\_t SGTL\_Deinit ( *sgtl\_handle\_t \* handle* )

Shut down Sgtl5000 modules.

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
---------------	------------------------------------

#### 45.6.5.11 status\_t SGTL\_ConfigDataFormat ( *sgtl\_handle\_t \* handle*, *uint32\_t mclk*, *uint32\_t sample\_rate*, *uint32\_t bits* )

This function would configure the registers about the sample rate, bit depths.

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
<i>mclk</i>	Master clock frequency of I2S.
<i>sample_rate</i>	Sample rate of audio file running in sgtl5000. Sgtl5000 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
<i>bits</i>	Bit depth of audio file (Sgtl5000 only supports 16bit, 20bit, 24bit and 32 bit in HW).

#### 45.6.5.12 status\_t SGTL\_SetPlay ( *sgtl\_handle\_t \* handle, uint32\_t playSource* )

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
<i>playSource</i>	play source value, reference _sgtl_play_source.

Returns

kStatus\_Success, else failed.

#### 45.6.5.13 status\_t SGTL\_SetRecord ( *sgtl\_handle\_t \* handle, uint32\_t recordSource* )

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
<i>recordSource</i>	record source value, reference _sgtl_record_source.

Returns

kStatus\_Success, else failed.

#### 45.6.5.14 status\_t SGTL\_WriteReg ( *sgtl\_handle\_t \* handle, uint16\_t reg, uint16\_t val* )

Parameters

<i>handle</i>	Sgtl5000 handle structure.
---------------	----------------------------

<i>reg</i>	The register address in sgtl.
<i>val</i>	Value needs to write into the register.

#### 45.6.5.15 status\_t SGTL\_ReadReg ( *sgtl\_handle\_t \* handle, uint16\_t reg, uint16\_t \* val* )

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>reg</i>	The register address in sgtl.
<i>val</i>	Value written to.

#### 45.6.5.16 status\_t SGTL\_ModifyReg ( *sgtl\_handle\_t \* handle, uint16\_t reg, uint16\_t clr\_mask, uint16\_t val* )

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>reg</i>	The register address in sgtl.
<i>clr_mask</i>	The mask code for the bits want to write. The bit you want to write should be 0.
<i>val</i>	Value needs to write into the register.

## 45.6.6 SGTL5000 Adapter

### 45.6.6.1 Overview

The sgtl5000 adapter provides a codec unify control interface.

#### Macros

- `#define HAL_CODEC_SGTL_HANDLER_SIZE (SGTL_I2C_HANDLER_SIZE + 4)`  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_SGTL5000_Init (void *handle, void *config)`  
*Codec initialization.*
- `status_t HAL_CODEC_SGTL5000_Deinit (void *handle)`  
*Codec de-initilization.*
- `status_t HAL_CODEC_SGTL5000_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `status_t HAL_CODEC_SGTL5000_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `status_t HAL_CODEC_SGTL5000_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `status_t HAL_CODEC_SGTL5000_SetPower (void *handle, uint32_t module, bool powerOn)`  
*set audio codec module power.*
- `status_t HAL_CODEC_SGTL5000_SetRecord (void *handle, uint32_t recordSource)`  
*codec set record source.*
- `status_t HAL_CODEC_SGTL5000_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`  
*codec set record channel.*
- `status_t HAL_CODEC_SGTL5000_SetPlay (void *handle, uint32_t playSource)`  
*codec set play source.*
- `status_t HAL_CODEC_SGTL5000_ModuleControl (void *handle, uint32_t cmd, uint32_t data)`  
*codec module control.*
- `static status_t HAL_CODEC_Init (void *handle, void *config)`  
*Codec initilization.*
- `static status_t HAL_CODEC_Deinit (void *handle)`  
*Codec de-initilization.*
- `static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)`  
*set audio data format.*
- `static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)`  
*set audio codec module volume.*
- `static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)`  
*set audio codec module mute.*
- `static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)`

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 45.6.6.2 Function Documentation

##### 45.6.6.2.1 `status_t HAL_CODEC_SGTL5000_Init( void * handle, void * config )`

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus\_Success is success, else initial failed.

##### 45.6.6.2.2 `status_t HAL_CODEC_SGTL5000_Deinit( void * handle )`

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

##### 45.6.6.2.3 `status_t HAL_CODEC_SGTL5000_SetFormat( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.4 status\_t HAL\_CODEC\_SGTL5000\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.5 status\_t HAL\_CODEC\_SGTL5000\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.6 status\_t HAL\_CODEC\_SGTL5000\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.7 status\_t HAL\_CODEC\_SGTL5000\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.8 status\_t HAL\_CODEC\_SGTL5000\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.9 status\_t HAL\_CODEC\_SGTL5000\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.10 status\_t HAL\_CODEC\_SGTL5000\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus\_Success is success, else initial failed.

#### 45.6.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

#### 45.6.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.6.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

**45.6.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

**45.6.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

## 45.7 WM8960 Driver

### 45.7.1 Overview

The wm8960 driver provides a codec control interface.

## Data Structures

- struct `wm8960_audio_format_t`  
*wm8960 audio format More...*
- struct `wm8960_master_sysclk_config_t`  
*wm8960 master system clock configuration More...*
- struct `wm8960_config_t`  
*Initialize structure of WM8960. More...*
- struct `wm8960_handle_t`  
*wm8960 codec handler More...*

## Macros

- #define `WM8960_I2C_HANDLER_SIZE` CODEC\_I2C\_MASTER\_HANDLER\_SIZE  
*wm8960 handle size*
- #define `WM8960_LINVOL` 0x0U  
*Define the register address of WM8960.*
- #define `WM8960_CACHEREGNUM` 56U  
*Cache register number.*
- #define `WM8960_CLOCK2_BCLK_DIV_MASK` 0xFU  
*WM8960 CLOCK2 bits.*
- #define `WM8960_IFACE1_FORMAT_MASK` 0x03U  
*WM8960\_IFACE1 FORMAT bits.*
- #define `WM8960_IFACE1_WL_MASK` 0x0CU  
*WM8960\_IFACE1 WL bits.*
- #define `WM8960_IFACE1_LRP_MASK` 0x10U  
*WM8960\_IFACE1 LRP bit.*
- #define `WM8960_IFACE1_DLRSWAP_MASK` 0x20U  
*WM8960\_IFACE1 DLRSWAP bit.*
- #define `WM8960_IFACE1_MS_MASK` 0x40U  
*WM8960\_IFACE1 MS bit.*
- #define `WM8960_IFACE1_BCLKINV_MASK` 0x80U  
*WM8960\_IFACE1 BCLKINV bit.*
- #define `WM8960_IFACE1_ALRSWAP_MASK` 0x100U  
*WM8960\_IFACE1 ALRSWAP bit.*
- #define `WM8960_POWER1_VREF_MASK` 0x40U  
*WM8960\_POWER1.*
- #define `WM8960_POWER2_DACL_MASK` 0x100U  
*WM8960\_POWER2.*
- #define `WM8960_I2C_ADDR` 0x1A  
*WM8960 I2C address.*
- #define `WM8960_I2C_BAUDRATE` (100000U)

- WM8960 I<sub>2</sub>C baudrate.  
• #define **WM8960\_ADC\_MAX\_VOLUME\_vVALUE** 0xFFU  
WM8960 maximum volume value.

## Enumerations

- enum **wm8960\_module\_t** {
   
kWM8960\_ModuleADC = 0,  
kWM8960\_ModuleDAC = 1,  
kWM8960\_ModuleVREF = 2,  
kWM8960\_ModuleHP = 3,  
kWM8960\_ModuleMICB = 4,  
kWM8960\_ModuleMIC = 5,  
kWM8960\_ModuleLineIn = 6,  
kWM8960\_ModuleLineOut = 7,  
kWM8960\_ModuleSpeaker = 8,  
kWM8960\_ModuleOMIX = 9 }

*Modules in WM8960 board.*

- enum {
   
kWM8960\_HeadphoneLeft = 1,  
kWM8960\_HeadphoneRight = 2,  
kWM8960\_SpeakerLeft = 4,  
kWM8960\_SpeakerRight = 8 }

*wm8960 play channel*

- enum **wm8960\_play\_source\_t** {
   
kWM8960\_PlaySourcePGA = 1,  
kWM8960\_PlaySourceInput = 2,  
kWM8960\_PlaySourceDAC = 4 }

*wm8960 play source*

- enum **wm8960\_route\_t** {
   
kWM8960\_RouteBypass = 0,  
kWM8960\_RoutePlayback = 1,  
kWM8960\_RoutePlaybackandRecord = 2,  
kWM8960\_RouteRecord = 5 }

*WM8960 data route.*

- enum **wm8960\_protocol\_t** {
   
kWM8960\_BusI2S = 2,  
kWM8960\_BusLeftJustified = 1,  
kWM8960\_BusRightJustified = 0,  
kWM8960\_BusPCMA = 3,  
kWM8960\_BusPCMB = 3 | (1 << 4) }

*The audio data transfer protocol choice.*

- enum **wm8960\_input\_t** {

```

kWM8960_InputClosed = 0,
kWM8960_InputSingleEndedMic = 1,
kWM8960_InputDifferentialMicInput2 = 2,
kWM8960_InputDifferentialMicInput3 = 3,
kWM8960_InputLineINPUT2 = 4,
kWM8960_InputLineINPUT3 = 5 }

    wm8960 input source
• enum {
    kWM8960_AudioSampleRate8KHz = 8000U,
    kWM8960_AudioSampleRate11025Hz = 11025U,
    kWM8960_AudioSampleRate12KHz = 12000U,
    kWM8960_AudioSampleRate16KHz = 16000U,
    kWM8960_AudioSampleRate22050Hz = 22050U,
    kWM8960_AudioSampleRate24KHz = 24000U,
    kWM8960_AudioSampleRate32KHz = 32000U,
    kWM8960_AudioSampleRate44100Hz = 44100U,
    kWM8960_AudioSampleRate48KHz = 48000U,
    kWM8960_AudioSampleRate96KHz = 96000U,
    kWM8960_AudioSampleRate192KHz = 192000U,
    kWM8960_AudioSampleRate384KHz = 384000U }

        audio sample rate definition
• enum {
    kWM8960_AudioBitWidth16bit = 16U,
    kWM8960_AudioBitWidth20bit = 20U,
    kWM8960_AudioBitWidth24bit = 24U,
    kWM8960_AudioBitWidth32bit = 32U }

        audio bit width
• enum wm8960_sysclk_source_t {
    kWM8960_SysClkSourceMclk = 0U,
    kWM8960_SysClkSourceInternalPLL = 1U }

    wm8960 sysclk source

```

## Functions

- **status\_t WM8960\_Init (wm8960\_handle\_t \*handle, const wm8960\_config\_t \*config)**  
*WM8960 initialize function.*
- **status\_t WM8960\_Deinit (wm8960\_handle\_t \*handle)**  
*Deinit the WM8960 codec.*
- **status\_t WM8960\_SetDataRoute (wm8960\_handle\_t \*handle, wm8960\_route\_t route)**  
*Set audio data route in WM8960.*
- **status\_t WM8960\_SetLeftInput (wm8960\_handle\_t \*handle, wm8960\_input\_t input)**  
*Set left audio input source in WM8960.*
- **status\_t WM8960\_SetRightInput (wm8960\_handle\_t \*handle, wm8960\_input\_t input)**  
*Set right audio input source in WM8960.*
- **status\_t WM8960\_SetProtocol (wm8960\_handle\_t \*handle, **wm8960\_protocol\_t** protocol)**  
*Set the audio transfer protocol.*

- void [WM8960\\_SetMasterSlave](#) (wm8960\_handle\_t \*handle, bool master)  
*Set WM8960 as master or slave.*
- status\_t [WM8960\\_SetVolume](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, uint32\_t volume)  
*Set the volume of different modules in WM8960.*
- uint32\_t [WM8960\\_GetVolume](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module)  
*Get the volume of different modules in WM8960.*
- status\_t [WM8960\\_SetMute](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, bool isEnabled)  
*Mute modules in WM8960.*
- status\_t [WM8960\\_SetModule](#) (wm8960\_handle\_t \*handle, wm8960\_module\_t module, bool isEnabled)  
*Enable/disable expected devices.*
- status\_t [WM8960\\_SetPlay](#) (wm8960\_handle\_t \*handle, uint32\_t playSource)  
*SET the WM8960 play source.*
- status\_t [WM8960\\_ConfigDataFormat](#) (wm8960\_handle\_t \*handle, uint32\_t sysclk, uint32\_t sample\_rate, uint32\_t bits)  
*Configure the data format of audio data.*
- status\_t [WM8960\\_SetJackDetect](#) (wm8960\_handle\_t \*handle, bool isEnabled)  
*Enable/disable jack detect feature.*
- status\_t [WM8960\\_WriteReg](#) (wm8960\_handle\_t \*handle, uint8\_t reg, uint16\_t val)  
*Write register to WM8960 using I2C.*
- status\_t [WM8960\\_ReadReg](#) (uint8\_t reg, uint16\_t \*val)  
*Read register from WM8960 using I2C.*
- status\_t [WM8960\\_ModifyReg](#) (wm8960\_handle\_t \*handle, uint8\_t reg, uint16\_t mask, uint16\_t val)  
*Modify some bits in the register using I2C.*

## Driver version

- #define [FSL\\_WM8960\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 1))  
*CLOCK driver version 2.2.1.*

### 45.7.2 Data Structure Documentation

#### 45.7.2.1 struct [wm8960\\_audio\\_format\\_t](#)

##### Data Fields

- uint32\_t [mclk\\_HZ](#)  
*master clock frequency*
- uint32\_t [sampleRate](#)  
*sample rate*
- uint32\_t [bitWidth](#)  
*bit width*

#### 45.7.2.2 struct `wm8960_master_sysclk_config_t`

##### Data Fields

- `wm8960_sysclk_source_t sysclkSource`  
*sysclk source*
- `uint32_t sysclkFreq`  
*PLL output frequency value.*

#### 45.7.2.3 struct `wm8960_config_t`

##### Data Fields

- `wm8960_route_t route`  
*Audio data route.*
- `wm8960_protocol_t bus`  
*Audio transfer protocol.*
- `wm8960_audio_format_t format`  
*Audio format.*
- `bool master_slave`  
*Master or slave.*
- `wm8960_master_sysclk_config_t masterClock`  
*master clock configurations*
- `bool enableSpeaker`  
*True means enable class D speaker as output, false means no.*
- `wm8960_input_t leftInputSource`  
*Left input source for WM8960.*
- `wm8960_input_t rightInputSource`  
*Right input source for WM8960.*
- `wm8960_play_source_t playSource`  
*play source*
- `uint8_t slaveAddress`  
*wm8960 device address*
- `codec_i2c_config_t i2cConfig`  
*i2c configuration*

##### Field Documentation

(1) `wm8960_route_t wm8960_config_t::route`

(2) `bool wm8960_config_t::master_slave`

#### 45.7.2.4 struct `wm8960_handle_t`

##### Data Fields

- `const wm8960_config_t * config`  
*wm8904 config pointer*
- `uint8_t i2cHandle [WM8960_I2C_HANDLER_SIZE]`  
*i2c handle*

### 45.7.3 Macro Definition Documentation

**45.7.3.1 #define WM8960\_LINVOL 0x0U**

**45.7.3.2 #define WM8960\_I2C\_ADDR 0x1A**

### 45.7.4 Enumeration Type Documentation

**45.7.4.1 enum wm8960\_module\_t**

Enumerator

*kWM8960\_ModuleADC* ADC module in WM8960.  
*kWM8960\_ModuleDAC* DAC module in WM8960.  
*kWM8960\_ModuleVREF* VREF module.  
*kWM8960\_ModuleHP* Headphone.  
*kWM8960\_ModuleMICB* Mic bias.  
*kWM8960\_ModuleMIC* Input Mic.  
*kWM8960\_ModuleLineIn* Analog in PGA.  
*kWM8960\_ModuleLineOut* Line out module.  
*kWM8960\_ModuleSpeaker* Speaker module.  
*kWM8960\_ModuleOMIX* Output mixer.

**45.7.4.2 anonymous enum**

Enumerator

*kWM8960\_HeadphoneLeft* wm8960 headphone left channel  
*kWM8960\_HeadphoneRight* wm8960 headphone right channel  
*kWM8960\_SpeakerLeft* wm8960 speaker left channel  
*kWM8960\_SpeakerRight* wm8960 speaker right channel

**45.7.4.3 enum wm8960\_play\_source\_t**

Enumerator

*kWM8960\_PlaySourcePGA* wm8960 play source PGA  
*kWM8960\_PlaySourceInput* wm8960 play source Input  
*kWM8960\_PlaySourceDAC* wm8960 play source DAC

**45.7.4.4 enum wm8960\_route\_t**

Only provide some typical data route, not all route listed. Note: Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

*kWM8960\_RouteBypass* LINEIN->Headphone.  
*kWM8960\_RoutePlayback* I2SIN->DAC->Headphone.  
*kWM8960\_RoutePlaybackandRecord* I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.  
*kWM8960\_RouteRecord* LINEIN->ADC->I2SOUT.

#### 45.7.4.5 enum **wm8960\_protocol\_t**

WM8960 only supports I2S format and PCM format.

Enumerator

*kWM8960\_BusI2S* I2S type.  
*kWM8960\_BusLeftJustified* Left justified mode.  
*kWM8960\_BusRightJustified* Right justified mode.  
*kWM8960\_BusPCMA* PCM A mode.  
*kWM8960\_BusPCMB* PCM B mode.

#### 45.7.4.6 enum **wm8960\_input\_t**

Enumerator

*kWM8960\_InputClosed* Input device is closed.  
*kWM8960\_InputSingleEndedMic* Input as single ended mic, only use L/RINPUT1.  
*kWM8960\_InputDifferentialMicInput2* Input as differential mic, use L/RINPUT1 and L/RINPUT2.  
*kWM8960\_InputDifferentialMicInput3* Input as differential mic, use L/RINPUT1 and L/RINPUT3.  
*kWM8960\_InputLineINPUT2* Input as line input, only use L/RINPUT2.  
*kWM8960\_InputLineINPUT3* Input as line input, only use L/RINPUT3.

#### 45.7.4.7 anonymous enum

Enumerator

*kWM8960\_AudioSampleRate8KHz* Sample rate 8000 Hz.  
*kWM8960\_AudioSampleRate11025Hz* Sample rate 11025 Hz.  
*kWM8960\_AudioSampleRate12KHz* Sample rate 12000 Hz.  
*kWM8960\_AudioSampleRate16KHz* Sample rate 16000 Hz.  
*kWM8960\_AudioSampleRate22050Hz* Sample rate 22050 Hz.  
*kWM8960\_AudioSampleRate24KHz* Sample rate 24000 Hz.  
*kWM8960\_AudioSampleRate32KHz* Sample rate 32000 Hz.  
*kWM8960\_AudioSampleRate44100Hz* Sample rate 44100 Hz.  
*kWM8960\_AudioSampleRate48KHz* Sample rate 48000 Hz.

*kWM8960\_AudioSampleRate96KHz* Sample rate 96000 Hz.  
*kWM8960\_AudioSampleRate192KHz* Sample rate 192000 Hz.  
*kWM8960\_AudioSampleRate384KHz* Sample rate 384000 Hz.

#### 45.7.4.8 anonymous enum

Enumerator

*kWM8960\_AudioBitWidth16bit* audio bit width 16  
*kWM8960\_AudioBitWidth20bit* audio bit width 20  
*kWM8960\_AudioBitWidth24bit* audio bit width 24  
*kWM8960\_AudioBitWidth32bit* audio bit width 32

#### 45.7.4.9 enum **wm8960\_sysclk\_source\_t**

Enumerator

*kWM8960\_SysClkSourceMclk* sysclk source from external MCLK  
*kWM8960\_SysClkSourceInternalPLL* sysclk source from internal PLL

### 45.7.5 Function Documentation

#### 45.7.5.1 status\_t **WM8960\_Init** ( **wm8960\_handle\_t \* handle**, **const wm8960\_config\_t \* config** )

The second parameter is NULL to WM8960 in this version. If users want to change the settings, they have to use `wm8960_write_reg()` or `wm8960_modify_reg()` to set the register value of WM8960. Note: If the `codec_config` is NULL, it would initialize WM8960 using default settings. The default setting: `codec_config->route = kWM8960_RoutePlaybackandRecord` `codec_config->bus = kWM8960_BusI2S` `codec_config->master = slave`

Parameters

<i>handle</i>	WM8960 handle structure.
<i>config</i>	WM8960 configuration structure.

#### 45.7.5.2 status\_t **WM8960\_Deinit** ( **wm8960\_handle\_t \* handle** )

This function close all modules in WM8960 to save power.

Parameters

<i>handle</i>	WM8960 handle structure pointer.
---------------	----------------------------------

#### 45.7.5.3 status\_t WM8960\_SetDataRoute ( *wm8960\_handle\_t \* handle, wm8960\_route\_t route* )

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules. Note: If a new route is set, the previous route would not work.

Parameters

<i>handle</i>	WM8960 handle structure.
<i>route</i>	Audio data route in WM8960.

#### 45.7.5.4 status\_t WM8960\_SetLeftInput ( *wm8960\_handle\_t \* handle, wm8960\_input\_t input* )

Parameters

<i>handle</i>	WM8960 handle structure.
<i>input</i>	Audio input source.

#### 45.7.5.5 status\_t WM8960\_SetRightInput ( *wm8960\_handle\_t \* handle, wm8960\_input\_t input* )

Parameters

<i>handle</i>	WM8960 handle structure.
<i>input</i>	Audio input source.

#### 45.7.5.6 status\_t WM8960\_SetProtocol ( *wm8960\_handle\_t \* handle, wm8960\_protocol\_t protocol* )

WM8960 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

<i>handle</i>	WM8960 handle structure.
<i>protocol</i>	Audio data transfer protocol.

#### 45.7.5.7 void WM8960\_SetMasterSlave ( **wm8960\_handle\_t \* handle, bool master** )

Parameters

<i>handle</i>	WM8960 handle structure.
<i>master</i>	1 represent master, 0 represent slave.

#### 45.7.5.8 status\_t WM8960\_SetVolume ( **wm8960\_handle\_t \* handle, wm8960\_module\_t module, uint32\_t volume** )

This function would set the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Module:kWM8960\_ModuleADC, volume range value: 0 is mute, 1-255 is -97db to 30db  
 Module:kWM8960\_ModuleDAC, volume range value: 0 is mute, 1-255 is -127db to 0db  
 Module:kWM8960\_ModuleHP, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db  
 Module:kWM8960\_ModuleLineIn, volume range value: 0 - 0x3F is -17.25db to 30db  
 Module:kWM8960\_ModuleSpeaker, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Module to set volume, it can be ADC, DAC, Headphone and so on.
<i>volume</i>	Volume value need to be set.

#### 45.7.5.9 uint32\_t WM8960\_GetVolume ( **wm8960\_handle\_t \* handle, wm8960\_module\_t module** )

This function gets the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Module to set volume, it can be ADC, DAC, Headphone and so on.

Returns

Volume value of the module.

#### 45.7.5.10 status\_t WM8960\_SetMute ( *wm8960\_handle\_t \* handle, wm8960\_module\_t module, bool isEnabled* )

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Modules need to be mute.
<i>isEnabled</i>	Mute or unmute, 1 represent mute.

#### 45.7.5.11 status\_t WM8960\_SetModule ( *wm8960\_handle\_t \* handle, wm8960\_module\_t module, bool isEnabled* )

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Module expected to enable.
<i>isEnabled</i>	Enable or disable moudles.

#### 45.7.5.12 status\_t WM8960\_SetPlay ( *wm8960\_handle\_t \* handle, uint32\_t playSource* )

Parameters

<i>handle</i>	WM8960 handle structure.
<i>playSource</i>	play source , can be a value combine of kWM8960_ModuleHeadphoneSourcePG-A, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft.

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

**45.7.5.13 status\_t WM8960\_ConfigDataFormat ( *wm8960\_handle\_t \* handle, uint32\_t sysclk, uint32\_t sample\_rate, uint32\_t bits* )**

This function would configure the registers about the sample rate, bit depths.

Parameters

<i>handle</i>	WM8960 handle structure pointer.
<i>sysclk</i>	system clock of the codec which can be generated by MCLK or PLL output.
<i>sample_rate</i>	Sample rate of audio file running in WM8960. WM8960 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
<i>bits</i>	Bit depth of audio file (WM8960 only supports 16bit, 20bit, 24bit and 32 bit in HW).

**45.7.5.14 status\_t WM8960\_SetJackDetect ( *wm8960\_handle\_t \* handle, bool isEnabled* )**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>isEnabled</i>	Enable or disable moudles.

**45.7.5.15 status\_t WM8960\_WriteReg ( *wm8960\_handle\_t \* handle, uint8\_t reg, uint16\_t val* )**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>reg</i>	The register address in WM8960.
<i>val</i>	Value needs to write into the register.

**45.7.5.16 status\_t WM8960\_ReadReg ( *uint8\_t reg, uint16\_t \* val* )**

Parameters

<i>reg</i>	The register address in WM8960.
<i>val</i>	Value written to.

**45.7.5.17 status\_t WM8960\_ModifyReg ( *wm8960\_handle\_t \* handle, uint8\_t reg, uint16\_t mask, uint16\_t val* )**

## Parameters

<i>handle</i>	WM8960 handle structure.
<i>reg</i>	The register address in WM8960.
<i>mask</i>	The mask code for the bits want to write. The bit you want to write should be 0.
<i>val</i>	Value needs to write into the register.

## 45.7.6 WM8960 Adapter

### 45.7.6.1 Overview

The wm8960 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_WM8960_HANDLER_SIZE` (`WM8960_I2C_HANDLER_SIZE + 4`)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_WM8960_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_WM8960_Deinit` (void \*handle)  
*Codec de-initilization.*
- `status_t HAL_CODEC_WM8960_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_WM8960_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_WM8960_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_WM8960_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_WM8960_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_WM8960_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_WM8960_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_WM8960_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initilization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initilization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 45.7.6.2 Function Documentation

##### 45.7.6.2.1 `status_t HAL_CODEC_WM8960_Init( void * handle, void * config )`

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus\_Success is success, else initial failed.

##### 45.7.6.2.2 `status_t HAL_CODEC_WM8960_Deinit( void * handle )`

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

##### 45.7.6.2.3 `status_t HAL_CODEC_WM8960_SetFormat( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.4 status\_t HAL\_CODEC\_WM8960\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.5 status\_t HAL\_CODEC\_WM8960\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.6 status\_t HAL\_CODEC\_WM8960\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.7 status\_t HAL\_CODEC\_WM8960\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.8 status\_t HAL\_CODEC\_WM8960\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.9 status\_t HAL\_CODEC\_WM8960\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.10 status\_t HAL\_CODEC\_WM8960\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus\_Success is success, else initial failed.

#### 45.7.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus\_Success is success, else de-initial failed.

#### 45.7.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus\_Success is success, else configure failed.

#### 45.7.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus\_Success is success, else configure failed.

**45.7.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus\_Success is success, else configure failed.

**45.7.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus\_Success is success, else configure failed.

## 45.8 WM8904 Driver

### 45.8.1 Overview

The wm8904 driver provides a codec control interface.

## Data Structures

- struct [wm8904\\_fll\\_config\\_t](#)  
*wm8904 fll configuration* [More...](#)
- struct [wm8904\\_audio\\_format\\_t](#)  
*Audio format configuration.* [More...](#)
- struct [wm8904\\_config\\_t](#)  
*Configuration structure of WM8904.* [More...](#)
- struct [wm8904\\_handle\\_t](#)  
*wm8904 codec handler* [More...](#)

## Macros

- #define [WM8904\\_I2C\\_HANDLER\\_SIZE](#) (CODEC\_I2C\_MASTER\_HANDLER\_SIZE)  
*wm8904 handle size*
- #define [WM8904\\_DEBUG\\_REGISTER](#) 0  
*wm8904 debug macro*
- #define [WM8904\\_RESET](#) (0x00)  
*WM8904 register map.*
- #define [WM8904\\_I2C\\_ADDRESS](#) (0x1A)  
*WM8904 I2C address.*
- #define [WM8904\\_I2C\\_BITRATE](#) (400000U)  
*WM8904 I2C bit rate.*
- #define [WM8904\\_MAP\\_HEADPHONE\\_LINEOUT\\_MAX\\_VOLUME](#) 0x3FU  
*WM8904 maximum volume.*

## Enumerations

- enum {
   
*kStatus\_WM8904\_Success* = 0x0,  
*kStatus\_WM8904\_Fail* = 0x1
 }
   
*WM8904 status return codes.*
- enum {
   
*kWM8904\_LRCPolarityNormal* = 0U,  
*kWM8904\_LRCPolarityInverted* = 1U << 4U
 }
   
*WM8904 lrc polarity.*
- enum [wm8904\\_module\\_t](#) {

- ```
kWM8904_ModuleADC = 0,
kWM8904_ModuleDAC = 1,
kWM8904_ModulePGA = 2,
kWM8904_ModuleHeadphone = 3,
kWM8904_ModuleLineout = 4 }
```

*wm8904 module value*
- enum

```
wm8904 play channel
```
- enum `wm8904_timeslot_t` {

```
kWM8904_TimeSlot0 = 0U,
kWM8904_TimeSlot1 = 1U }
```

*WM8904 time slot.*
- enum `wm8904_protocol_t` {

```
kWM8904_ProtocolI2S = 0x2,
kWM8904_ProtocolLeftJustified = 0x1,
kWM8904_ProtocolRightJustified = 0x0,
kWM8904_ProtocolPCMA = 0x3,
kWM8904_ProtocolPCMB = 0x3 | (1 << 4) }
```

*The audio data transfer protocol.*
- enum `wm8904_fs_ratio_t` {

```
kWM8904_FsRatio64X = 0x0,
kWM8904_FsRatio128X = 0x1,
kWM8904_FsRatio192X = 0x2,
kWM8904_FsRatio256X = 0x3,
kWM8904_FsRatio384X = 0x4,
kWM8904_FsRatio512X = 0x5,
kWM8904_FsRatio768X = 0x6,
kWM8904_FsRatio1024X = 0x7,
kWM8904_FsRatio1408X = 0x8,
kWM8904_FsRatio1536X = 0x9 }
```

*The SYSCLK / fs ratio.*
- enum `wm8904_sample_rate_t` {

```
kWM8904_SampleRate8kHz = 0x0,
kWM8904_SampleRate12kHz = 0x1,
kWM8904_SampleRate16kHz = 0x2,
kWM8904_SampleRate24kHz = 0x3,
kWM8904_SampleRate32kHz = 0x4,
kWM8904_SampleRate48kHz = 0x5,
kWM8904_SampleRate11025Hz = 0x6,
kWM8904_SampleRate22050Hz = 0x7,
kWM8904_SampleRate44100Hz = 0x8 }
```

*Sample rate.*
- enum `wm8904_bit_width_t` {

```
kWM8904_BitWidth16 = 0x0,
kWM8904_BitWidth20 = 0x1,
kWM8904_BitWidth24 = 0x2,
```

```

kWM8904_BitWidth32 = 0x3 }

Bit width.
• enum {
    kWM8904_RecordSourceDifferentialLine = 1U,
    kWM8904_RecordSourceLineInput = 2U,
    kWM8904_RecordSourceDifferentialMic = 4U,
    kWM8904_RecordSourceDigitalMic = 8U }
    wm8904 record source
• enum {
    kWM8904_RecordChannelLeft1 = 1U,
    kWM8904_RecordChannelLeft2 = 2U,
    kWM8904_RecordChannelLeft3 = 4U,
    kWM8904_RecordChannelRight1 = 1U,
    kWM8904_RecordChannelRight2 = 2U,
    kWM8904_RecordChannelRight3 = 4U,
    kWM8904_RecordChannelDifferentialPositive1 = 1U,
    kWM8904_RecordChannelDifferentialPositive2 = 2U,
    kWM8904_RecordChannelDifferentialPositive3 = 4U,
    kWM8904_RecordChannelDifferentialNegative1 = 8U,
    kWM8904_RecordChannelDifferentialNegative2 = 16U,
    kWM8904_RecordChannelDifferentialNegative3 = 32U }
    wm8904 record channel
• enum {
    kWM8904_PlaySourcePGA = 1U,
    kWM8904_PlaySourceDAC = 4U }
    wm8904 play source
• enum wm8904\_sys\_clk\_source\_t {
    kWM8904_SysClkSourceMCLK = 0U,
    kWM8904_SysClkSourceFLL = 1U << 14 }
    wm8904 system clock source
• enum wm8904\_fll\_clk\_source\_t { kWM8904_FLLClkSourceMCLK = 0U }
    wm8904 fll clock source

```

## Functions

- [status\\_t WM8904\\_WriteRegister \(wm8904\\_handle\\_t \\*handle, uint8\\_t reg, uint16\\_t value\)](#)  
*WM8904 write register.*
- [status\\_t WM8904\\_ReadRegister \(wm8904\\_handle\\_t \\*handle, uint8\\_t reg, uint16\\_t \\*value\)](#)  
*WM8904 write register.*
- [status\\_t WM8904\\_ModifyRegister \(wm8904\\_handle\\_t \\*handle, uint8\\_t reg, uint16\\_t mask, uint16\\_t value\)](#)  
*WM8904 modify register.*
- [status\\_t WM8904\\_Init \(wm8904\\_handle\\_t \\*handle, \[wm8904\\\_config\\\_t\]\(#\) \\*wm8904Config\)](#)  
*Initializes WM8904.*
- [status\\_t WM8904\\_Deinit \(wm8904\\_handle\\_t \\*handle\)](#)  
*Deinitializes the WM8904 codec.*
- [void WM8904\\_GetDefaultConfig \(\[wm8904\\\_config\\\_t\]\(#\) \\*config\)](#)

*Fills the configuration structure with default values.*

- **status\_t WM8904\_SetMasterSlave** (`wm8904_handle_t *handle, bool master`)
 

*Sets WM8904 as master or slave.*
- **status\_t WM8904\_SetMasterClock** (`wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth`)
 

*Sets WM8904 master clock configuration.*
- **status\_t WM8904\_SetFLLConfig** (`wm8904_handle_t *handle, wm8904_fll_config_t *config`)
 

*WM8904 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.*
- **status\_t WM8904\_SetProtocol** (`wm8904_handle_t *handle, wm8904_protocol_t protocol`)
 

*Sets the audio data transfer protocol.*
- **status\_t WM8904\_SetAudioFormat** (`wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth`)
 

*Sets the audio data format.*
- **status\_t WM8904\_CheckAudioFormat** (`wm8904_handle_t *handle, wm8904_audio_format_t *format, uint32_t mclkFreq`)
 

*check and update the audio data format.*
- **status\_t WM8904\_SetVolume** (`wm8904_handle_t *handle, uint16_t volumeLeft, uint16_t volumeRight`)
 

*Sets the module output volume.*
- **status\_t WM8904\_SetMute** (`wm8904_handle_t *handle, bool muteLeft, bool muteRight`)
 

*Sets the headphone output mute.*
- **status\_t WM8904\_SelectLRCPolarity** (`wm8904_handle_t *handle, uint32_t polarity`)
 

*Select LRC polarity.*
- **status\_t WM8904\_EnableDACTDMMode** (`wm8904_handle_t *handle, wm8904_timeslot_t timeSlot`)
 

*Enable WM8904 DAC time slot.*
- **status\_t WM8904\_EnableADCTDMMode** (`wm8904_handle_t *handle, wm8904_timeslot_t timeSlot`)
 

*Enable WM8904 ADC time slot.*
- **status\_t WM8904\_SetModulePower** (`wm8904_handle_t *handle, wm8904_module_t module, bool isEnabled`)
 

*SET the module output power.*
- **status\_t WM8904\_SetDACVolume** (`wm8904_handle_t *handle, uint8_t volume`)
 

*SET the DAC module volume.*
- **status\_t WM8904\_SetChannelVolume** (`wm8904_handle_t *handle, uint32_t channel, uint32_t volume`)
 

*Sets the channel output volume.*
- **status\_t WM8904\_SetRecord** (`wm8904_handle_t *handle, uint32_t recordSource`)
 

*SET the WM8904 record source.*
- **status\_t WM8904\_SetRecordChannel** (`wm8904_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel`)
 

*SET the WM8904 record source.*
- **status\_t WM8904\_SetPlay** (`wm8904_handle_t *handle, uint32_t playSource`)
 

*SET the WM8904 play source.*
- **status\_t WM8904\_SetChannelMute** (`wm8904_handle_t *handle, uint32_t channel, bool isMute`)
 

*Sets the channel mute.*

## Driver version

- #define FSL\_WM8904\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 1))  
*WM8904 driver version 2.5.1.*

### 45.8.2 Data Structure Documentation

#### 45.8.2.1 struct wm8904\_fll\_config\_t

##### Data Fields

- `wm8904_fll_clk_source_t source`  
*fll reference clock source*
- `uint32_t refClock_HZ`  
*fll reference clock frequency*
- `uint32_t outputClock_HZ`  
*fll output clock frequency*

#### 45.8.2.2 struct wm8904\_audio\_format\_t

##### Data Fields

- `wm8904_fs_ratio_t fsRatio`  
*SYSCLK / fs ratio.*
- `wm8904_sample_rate_t sampleRate`  
*Sample rate.*
- `wm8904_bit_width_t bitWidth`  
*Bit width.*

#### 45.8.2.3 struct wm8904\_config\_t

##### Data Fields

- `bool master`  
*Master or slave.*
- `wm8904_sys_clk_source_t sysClkSource`  
*system clock source*
- `wm8904_fll_config_t * fll`  
*fll configuration*
- `wm8904_protocol_t protocol`  
*Audio transfer protocol.*
- `wm8904_audio_format_t format`  
*Audio format.*
- `uint32_t mclk_HZ`  
*MCLK frequency value.*
- `uint16_t recordSource`  
*record source*

- `uint16_t recordChannelLeft`  
*record channel*
- `uint16_t recordChannelRight`  
*record channel*
- `uint16_t playSource`  
*play source*
- `uint8_t slaveAddress`  
*code device slave address*
- `codec_i2c_config_t i2cConfig`  
*i2c bus configuration*

#### 45.8.2.4 struct `wm8904_handle_t`

##### Data Fields

- `wm8904_config_t * config`  
*wm8904 config pointer*
- `uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]`  
*i2c handle*

#### 45.8.3 Macro Definition Documentation

45.8.3.1 `#define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))`

45.8.3.2 `#define WM8904_I2C_ADDRESS (0x1A)`

45.8.3.3 `#define WM8904_I2C_BITRATE (400000U)`

#### 45.8.4 Enumeration Type Documentation

##### 45.8.4.1 anonymous enum

Enumerator

`kStatus_WM8904_Success` Success.  
`kStatus_WM8904_Fail` Failure.

##### 45.8.4.2 anonymous enum

Enumerator

`kWM8904_LRCPolarityNormal` LRC polarity normal.  
`kWM8904_LRCPolarityInverted` LRC polarity inverted.

#### 45.8.4.3 enum wm8904\_module\_t

Enumerator

- kWM8904\_ModuleADC* module ADC
- kWM8904\_ModuleDAC* module DAC
- kWM8904\_ModulePGA* module PGA
- kWM8904\_ModuleHeadphone* module headphone
- kWM8904\_ModuleLineout* module line out

#### 45.8.4.4 anonymous enum

#### 45.8.4.5 enum wm8904\_timeslot\_t

Enumerator

- kWM8904\_TimeSlot0* time slot0
- kWM8904\_TimeSlot1* time slot1

#### 45.8.4.6 enum wm8904\_protocol\_t

Enumerator

- kWM8904\_ProtocolI2S* I2S type.
- kWM8904\_ProtocolLeftJustified* Left justified mode.
- kWM8904\_ProtocolRightJustified* Right justified mode.
- kWM8904\_ProtocolPCMA* PCM A mode.
- kWM8904\_ProtocolPCMB* PCM B mode.

#### 45.8.4.7 enum wm8904\_fs\_ratio\_t

Enumerator

- kWM8904\_FsRatio64X* SYSCLK is  $64 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio128X* SYSCLK is  $128 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio192X* SYSCLK is  $192 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio256X* SYSCLK is  $256 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio384X* SYSCLK is  $384 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio512X* SYSCLK is  $512 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio768X* SYSCLK is  $768 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio1024X* SYSCLK is  $1024 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio1408X* SYSCLK is  $1408 * \text{sample rate} * \text{frame width}$ .
- kWM8904\_FsRatio1536X* SYSCLK is  $1536 * \text{sample rate} * \text{frame width}$ .

#### 45.8.4.8 enum `wm8904_sample_rate_t`

Enumerator

- kWM8904\_SampleRate8kHz* 8 kHz
- kWM8904\_SampleRate12kHz* 12kHz
- kWM8904\_SampleRate16kHz* 16kHz
- kWM8904\_SampleRate24kHz* 24kHz
- kWM8904\_SampleRate32kHz* 32kHz
- kWM8904\_SampleRate48kHz* 48kHz
- kWM8904\_SampleRate11025Hz* 11.025kHz
- kWM8904\_SampleRate22050Hz* 22.05kHz
- kWM8904\_SampleRate44100Hz* 44.1kHz

#### 45.8.4.9 enum `wm8904_bit_width_t`

Enumerator

- kWM8904\_BitWidth16* 16 bits
- kWM8904\_BitWidth20* 20 bits
- kWM8904\_BitWidth24* 24 bits
- kWM8904\_BitWidth32* 32 bits

#### 45.8.4.10 anonymous enum

Enumerator

- kWM8904\_RecordSourceDifferentialLine* record source from differential line
- kWM8904\_RecordSourceLineInput* record source from line input
- kWM8904\_RecordSourceDifferentialMic* record source from differential mic
- kWM8904\_RecordSourceDigitalMic* record source from digital microphone

#### 45.8.4.11 anonymous enum

Enumerator

- kWM8904\_RecordChannelLeft1* left record channel 1
- kWM8904\_RecordChannelLeft2* left record channel 2
- kWM8904\_RecordChannelLeft3* left record channel 3
- kWM8904\_RecordChannelRight1* right record channel 1
- kWM8904\_RecordChannelRight2* right record channel 2
- kWM8904\_RecordChannelRight3* right record channel 3
- kWM8904\_RecordChannelDifferentialPositive1* differential positive record channel 1
- kWM8904\_RecordChannelDifferentialPositive2* differential positive record channel 2
- kWM8904\_RecordChannelDifferentialPositive3* differential positive record channel 3

- kWM8904\_RecordChannelDifferentialNegative1* differential negative record channel 1
- kWM8904\_RecordChannelDifferentialNegative2* differential negative record channel 2
- kWM8904\_RecordChannelDifferentialNegative3* differential negative record channel 3

#### 45.8.4.12 anonymous enum

Enumerator

- kWM8904\_PlaySourcePGA* play source PGA, bypass ADC
- kWM8904\_PlaySourceDAC* play source Input3

#### 45.8.4.13 enum `wm8904_sys_clk_source_t`

Enumerator

- kWM8904\_SysClkSourceMCLK* wm8904 system clock soure from MCLK
- kWM8904\_SysClkSourceFLL* wm8904 system clock soure from FLL

#### 45.8.4.14 enum `wm8904_fll_clk_source_t`

Enumerator

- kWM8904\_FLLClkSourceMCLK* wm8904 FLL clock source from MCLK

### 45.8.5 Function Documentation

#### 45.8.5.1 status\_t `WM8904_WriteRegister` ( `wm8904_handle_t * handle, uint8_t reg, uint16_t value` )

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>reg</i>    | register address.        |
| <i>value</i>  | value to write.          |

Returns

`kStatus_Success`, else failed.

#### 45.8.5.2 status\_t `WM8904_ReadRegister` ( `wm8904_handle_t * handle, uint8_t reg, uint16_t * value` )

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>reg</i>    | register address.        |
| <i>value</i>  | value to read.           |

Returns

kStatus\_Success, else failed.

#### 45.8.5.3 status\_t WM8904\_ModifyRegister ( *wm8904\_handle\_t \* handle, uint8\_t reg, uint16\_t mask, uint16\_t value* )

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>reg</i>    | register address.        |
| <i>mask</i>   | register bits mask.      |
| <i>value</i>  | value to write.          |

Returns

kStatus\_Success, else failed.

#### 45.8.5.4 status\_t WM8904\_Init ( *wm8904\_handle\_t \* handle, wm8904\_config\_t \* wm8904Config* )

Parameters

|                     |                                 |
|---------------------|---------------------------------|
| <i>handle</i>       | WM8904 handle structure.        |
| <i>wm8904Config</i> | WM8904 configuration structure. |

#### 45.8.5.5 status\_t WM8904\_Deinit ( *wm8904\_handle\_t \* handle* )

This function resets WM8904.

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
|---------------|--------------------------|

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.6 void WM8904\_GetDefaultConfig ( *wm8904\_config\_t \* config* )

The default values are:

```
master = false; protocol = kWM8904_ProtocolI2S; format.fsRatio = kWM8904_FsRatio64X; format.-sampleRate = kWM8904_SampleRate48kHz; format.bitWidth = kWM8904_BitWidth16;
```

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>config</i> | default configurations of wm8904. |
|---------------|-----------------------------------|

#### 45.8.5.7 status\_t WM8904\_SetMasterSlave ( *wm8904\_handle\_t \* handle, bool master* )

**Deprecated** DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY [WM8904\\_SetMasterClock](#)

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>handle</i> | WM8904 handle structure.          |
| <i>master</i> | true for master, false for slave. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.8 status\_t WM8904\_SetMasterClock ( *wm8904\_handle\_t \* handle, uint32\_t sysclk, uint32\_t sampleRate, uint32\_t bitWidth* )

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>handle</i>     | WM8904 handle structure.       |
| <i>sysclk</i>     | system clock source frequency. |
| <i>sampleRate</i> | sample rate                    |
| <i>bitWidth</i>   | bit width                      |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.9 status\_t WM8904\_SetFLLConfig ( *wm8904\_handle\_t \* handle*, *wm8904\_fll\_config\_t \* config* )

Parameters

|               |                            |
|---------------|----------------------------|
| <i>handle</i> | wm8904 handler pointer.    |
| <i>config</i> | FLL configuration pointer. |

#### 45.8.5.10 status\_t WM8904\_SetProtocol ( *wm8904\_handle\_t \* handle*, *wm8904\_protocol\_t protocol* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>protocol</i> | Audio transfer protocol. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.11 status\_t WM8904\_SetAudioFormat ( *wm8904\_handle\_t \* handle*, *uint32\_t sysclk*, *uint32\_t sampleRate*, *uint32\_t bitWidth* )

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>handle</i>     | WM8904 handle structure.       |
| <i>sysclk</i>     | system clock source frequency. |
| <i>sampleRate</i> | Sample rate frequency in Hz.   |
| <i>bitWidth</i>   | Audio data bit width.          |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.12 status\_t WM8904\_CheckAudioFormat ( *wm8904\_handle\_t \* handle*, *wm8904\_audio\_format\_t \* format*, *uint32\_t mclkFreq* )

This api is used check the fsRatio setting based on the mclk and sample rate, if fsRatio setting is not correct, it will correct it according to mclk and sample rate.

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>format</i>   | audio data format        |
| <i>mclkFreq</i> | mclk frequency           |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.13 status\_t WM8904\_SetVolume ( *wm8904\_handle\_t \* handle*, *uint16\_t volumeLeft*, *uint16\_t volumeRight* )

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57DB, 63 for 6DB.

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
|---------------|--------------------------|

|                    |                       |
|--------------------|-----------------------|
| <i>volumeLeft</i>  | left channel volume.  |
| <i>volumeRight</i> | right channel volume. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.14 status\_t WM8904\_SetMute ( *wm8904\_handle\_t \* handle, bool muteLeft, bool muteRight* )

Parameters

|                  |                                              |
|------------------|----------------------------------------------|
| <i>handle</i>    | WM8904 handle structure.                     |
| <i>muteLeft</i>  | true to mute left channel, false to unmute.  |
| <i>muteRight</i> | true to mute right channel, false to unmute. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.15 status\_t WM8904\_SelectLRCPolarity ( *wm8904\_handle\_t \* handle, uint32\_t polarity* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>polarity</i> | LRC clock polarity.      |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.16 status\_t WM8904\_EnableDACTDMMode ( *wm8904\_handle\_t \* handle, wm8904\_timeslot\_t timeSlot* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>timeSlot</i> | timeslot number.         |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.17 status\_t WM8904\_EnableADCTDMMMode ( *wm8904\_handle\_t \* handle,* *wm8904\_timeslot\_t timeSlot* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>handle</i>   | WM8904 handle structure. |
| <i>timeSlot</i> | timeslot number.         |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.18 status\_t WM8904\_SetModulePower ( *wm8904\_handle\_t \* handle,* *wm8904\_module\_t module, bool isEnabled* )

Parameters

|                                |                                   |
|--------------------------------|-----------------------------------|
| <i>handle</i>                  | WM8904 handle structure.          |
| <i>module</i>                  | wm8904 module.                    |
| <i>isEnabled</i> , <i>true</i> | is power on, false is power down. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 45.8.5.19 status\_t WM8904\_SetDACVolume ( *wm8904\_handle\_t \* handle, uint8\_t volume* *)*

Parameters

|               |                          |
|---------------|--------------------------|
| <i>handle</i> | WM8904 handle structure. |
| <i>volume</i> | volume to be configured. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 45.8.5.20 status\_t WM8904\_SetChannelVolume ( *wm8904\_handle\_t \* handle, uint32\_t channel, uint32\_t volume* )

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57dB, 63 for 6DB.

Parameters

|                |                          |
|----------------|--------------------------|
| <i>handle</i>  | codec handle structure.  |
| <i>channel</i> | codec channel.           |
| <i>volume</i>  | volume value from 0 -63. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.21 status\_t WM8904\_SetRecord ( *wm8904\_handle\_t \* handle, uint32\_t recordSource* )

Parameters

|                     |                                                                                                                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>       | WM8904 handle structure.                                                                                                                                                                             |
| <i>recordSource</i> | record source , can be a value of kCODEC_ModuleRecordSourceDifferential-Line, kCODEC_ModuleRecordSourceDifferentialMic, kCODEC_ModuleRecord-SourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise.

#### 45.8.5.22 status\_t WM8904\_SetRecordChannel ( *wm8904\_handle\_t \* handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel* )

Parameters

|                            |                                                                                                                                                                                                            |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | WM8904 handle structure.                                                                                                                                                                                   |
| <i>leftRecord-Channel</i>  | channel number of left record channel when using differential source, channel number of single end left channel when using single end source, channel number of digital mic when using digital mic source. |
| <i>rightRecord-Channel</i> | channel number of right record channel when using differential source, channel number of single end right channel when using single end source.                                                            |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 45.8.5.23 status\_t WM8904\_SetPlay ( **wm8904\_handle\_t \* handle, uint32\_t playSource** )

Parameters

|                   |                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>     | WM8904 handle structure.                                                                                                                                        |
| <i>playSource</i> | play source , can be a value of kCODEC_ModuleHeadphoneSourcePGA, kCODEC_ModuleHeadphoneSourceDAC, kCODEC_ModuleLineoutSourcePGA, kCODEC_ModuleLineoutSourceDAC. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 45.8.5.24 status\_t WM8904\_SetChannelMute ( **wm8904\_handle\_t \* handle, uint32\_t channel, bool isMute** )

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>handle</i>  | codec handle structure.     |
| <i>channel</i> | codec module name.          |
| <i>isMute</i>  | true is mute, false unmute. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

## 45.8.6 WM8904 Adapter

### 45.8.6.1 Overview

The wm8904 adapter provides a codec unify control interface.

#### Macros

- #define `HAL_CODEC_WM8904_HANDLER_SIZE` (`WM8904_I2C_HANDLER_SIZE + 4`)  
*codec handler size*

#### Functions

- `status_t HAL_CODEC_WM8904_Init` (void \*handle, void \*config)  
*Codec initialization.*
- `status_t HAL_CODEC_WM8904_Deinit` (void \*handle)  
*Codec de-initilization.*
- `status_t HAL_CODEC_WM8904_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- `status_t HAL_CODEC_WM8904_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- `status_t HAL_CODEC_WM8904_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- `status_t HAL_CODEC_WM8904_SetPower` (void \*handle, uint32\_t module, bool powerOn)  
*set audio codec module power.*
- `status_t HAL_CODEC_WM8904_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- `status_t HAL_CODEC_WM8904_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- `status_t HAL_CODEC_WM8904_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- `status_t HAL_CODEC_WM8904_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*
- static `status_t HAL_CODEC_Init` (void \*handle, void \*config)  
*Codec initilization.*
- static `status_t HAL_CODEC_Deinit` (void \*handle)  
*Codec de-initilization.*
- static `status_t HAL_CODEC_SetFormat` (void \*handle, uint32\_t mclk, uint32\_t sampleRate, uint32\_t bitWidth)  
*set audio data format.*
- static `status_t HAL_CODEC_SetVolume` (void \*handle, uint32\_t playChannel, uint32\_t volume)  
*set audio codec module volume.*
- static `status_t HAL_CODEC_SetMute` (void \*handle, uint32\_t playChannel, bool isMute)  
*set audio codec module mute.*
- static `status_t HAL_CODEC_SetPower` (void \*handle, uint32\_t module, bool powerOn)

- static `status_t HAL_CODEC_SetRecord` (void \*handle, uint32\_t recordSource)  
*codec set record source.*
- static `status_t HAL_CODEC_SetRecordChannel` (void \*handle, uint32\_t leftRecordChannel, uint32\_t rightRecordChannel)  
*codec set record channel.*
- static `status_t HAL_CODEC_SetPlay` (void \*handle, uint32\_t playSource)  
*codec set play source.*
- static `status_t HAL_CODEC_ModuleControl` (void \*handle, uint32\_t cmd, uint32\_t data)  
*codec module control.*

#### 45.8.6.2 Function Documentation

##### 45.8.6.2.1 `status_t HAL_CODEC_WM8904_Init ( void * handle, void * config )`

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

##### 45.8.6.2.2 `status_t HAL_CODEC_WM8904_Deinit ( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

##### 45.8.6.2.3 `status_t HAL_CODEC_WM8904_SetFormat ( void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth )`

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.4 status\_t HAL\_CODEC\_WM8904\_SetVolume ( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.5 status\_t HAL\_CODEC\_WM8904\_SetMute ( void \* *handle*, uint32\_t *playChannel*, bool *isMute* )

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.6 status\_t HAL\_CODEC\_WM8904\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* )

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.7 status\_t HAL\_CODEC\_WM8904\_SetRecord ( void \* *handle*, uint32\_t *recordSource* )

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.8 status\_t HAL\_CODEC\_WM8904\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* )

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.9 status\_t HAL\_CODEC\_WM8904\_SetPlay ( void \* *handle*, uint32\_t *playSource* )

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.10 status\_t HAL\_CODEC\_WM8904\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* )

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.11 static status\_t HAL\_CODEC\_Init ( void \* *handle*, void \* *config* ) [inline], [static]

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | codec handle.        |
| <i>config</i> | codec configuration. |

Returns

kStatus\_Success is success, else initial failed.

#### 45.8.6.2.12 static status\_t HAL\_CODEC\_Deinit ( void \* *handle* ) [inline], [static]

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

#### 45.8.6.2.13 static status\_t HAL\_CODEC\_SetFormat( void \* *handle*, uint32\_t *mclk*, uint32\_t *sampleRate*, uint32\_t *bitWidth* ) [inline], [static]

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>handle</i>     | codec handle.                 |
| <i>mclk</i>       | master clock frequency in HZ. |
| <i>sampleRate</i> | sample rate in HZ.            |
| <i>bitWidth</i>   | bit width.                    |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.14 static status\_t HAL\_CODEC\_SetVolume( void \* *handle*, uint32\_t *playChannel*, uint32\_t *volume* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>volume</i>      | volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.        |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.15 static status\_t HAL\_CODEC\_SetMute( void \* *handle*, uint32\_t *playChannel*, bool *isMute* ) [inline], [static]

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>      | codec handle.                                                                     |
| <i>playChannel</i> | audio codec play channel, can be a value or combine value of _codec_play_channel. |
| <i>isMute</i>      | true is mute, false is unmute.                                                    |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.16 static status\_t HAL\_CODEC\_SetPower ( void \* *handle*, uint32\_t *module*, bool *powerOn* ) [inline], [static]

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>handle</i>  | codec handle.                          |
| <i>module</i>  | audio codec module.                    |
| <i>powerOn</i> | true is power on, false is power down. |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.17 static status\_t HAL\_CODEC\_SetRecord ( void \* *handle*, uint32\_t *recordSource* ) [inline], [static]

Parameters

|                     |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------|
| <i>handle</i>       | codec handle.                                                                       |
| <i>recordSource</i> | audio codec record source, can be a value or combine value of _codec_record_source. |

Returns

kStatus\_Success is success, else configure failed.

#### 45.8.6.2.18 static status\_t HAL\_CODEC\_SetRecordChannel ( void \* *handle*, uint32\_t *leftRecordChannel*, uint32\_t *rightRecordChannel* ) [inline], [static]

Parameters

|                            |                                                                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i>              | codec handle.                                                                                                                    |
| <i>leftRecord-Channel</i>  | audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel. |
| <i>rightRecord-Channel</i> | audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.          |

Returns

kStatus\_Success is success, else configure failed.

**45.8.6.2.19 static status\_t HAL\_CODEC\_SetPlay ( void \* *handle*, uint32\_t *playSource* ) [inline], [static]**

Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>handle</i>     | codec handle.                                                                   |
| <i>playSource</i> | audio codec play source, can be a value or combine value of _codec_play_source. |

Returns

kStatus\_Success is success, else configure failed.

**45.8.6.2.20 static status\_t HAL\_CODEC\_ModuleControl ( void \* *handle*, uint32\_t *cmd*, uint32\_t *data* ) [inline], [static]**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

|               |                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>handle</i> | codec handle.                                                                                                                                                                                                                                                         |
| <i>cmd</i>    | module control cmd, reference _codec_module_ctrl_cmd.                                                                                                                                                                                                                 |
| <i>data</i>   | value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations. |

Returns

kStatus\_Success is success, else configure failed.

# Chapter 46

## Serial Manager

### 46.1 Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are UART, USB CDC and SWO.

### Modules

- [Serial Port SWO](#)
- [Serial Port USB](#)
- [Serial Port Uart](#)

### Data Structures

- struct [serial\\_manager\\_config\\_t](#)  
*serial manager config structure. [More...](#)*
- struct [serial\\_manager\\_callback\\_message\\_t](#)  
*Callback message structure. [More...](#)*

### Macros

- #define [SERIAL\\_MANAGER\\_NON\\_BLOCKING\\_MODE](#) (0U)  
*Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)*
- #define [SERIAL\\_MANAGER\\_RING\\_BUFFER\\_FLOWCONTROL](#) (0U)  
*Enable or ring buffer flow control (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_UART](#) (0U)  
*Enable or disable uart port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_UART\\_DMA](#) (0U)  
*Enable or disable uart dma port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_USBCDC](#) (0U)  
*Enable or disable USB CDC port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SWO](#) (0U)  
*Enable or disable SWO port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_VIRTUAL](#) (0U)  
*Enable or disable USB CDC virtual port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_RPMSG](#) (0U)  
*Enable or disable rpmsg port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SPI\\_MASTER](#) (0U)  
*Enable or disable SPI Master port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SPI\\_SLAVE](#) (0U)  
*Enable or disable SPI Slave port (1 - enable, 0 - disable)*
- #define [SERIAL\\_MANAGER\\_TASK\\_HANDLE\\_TX](#) (0U)  
*Enable or disable SerialManager\_Task() handle TX to prevent recursive calling.*

- #define **SERIAL\_MANAGER\_WRITE\_TIME\_DELAY\_DEFAULT\_VALUE** (1U)  
*Set the default delay time in ms used by SerialManager\_WriteTimeDelay().*
- #define **SERIAL\_MANAGER\_READ\_TIME\_DELAY\_DEFAULT\_VALUE** (1U)  
*Set the default delay time in ms used by SerialManager\_ReadTimeDelay().*
- #define **SERIAL\_MANAGER\_TASK\_HANDLE\_RX\_AVAILABLE\_NOTIFY** (0U)  
*Enable or disable SerialManager\_Task() handle RX data available notify.*
- #define **SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE** (4U)  
*Set serial manager write handle size.*
- #define **SERIAL\_MANAGER\_USE\_COMMON\_TASK** (0U)  
*SERIAL\_PORT\_UART\_HANDLE\_SIZE/SERIAL\_PORT\_USB\_CDC\_HANDLE\_SIZE + serial manager dedicated size.*
- #define **SERIAL\_MANAGER\_HANDLE\_SIZE** (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 12U)  
*Definition of serial manager handle size.*
- #define **SERIAL\_MANAGER\_HANDLE\_DEFINE**(name) uint32\_t name[((**SERIAL\_MANAGER\_HANDLE\_SIZE** + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]  
*Defines the serial manager handle.*
- #define **SERIAL\_MANAGER\_WRITE\_HANDLE\_DEFINE**(name) uint32\_t name[((**SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE** + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]  
*Defines the serial manager write handle.*
- #define **SERIAL\_MANAGER\_READ\_HANDLE\_DEFINE**(name) uint32\_t name[((**SERIAL\_MANAGER\_READ\_HANDLE\_SIZE** + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]  
*Defines the serial manager read handle.*
- #define **SERIAL\_MANAGER\_TASK\_PRIORITY** (2U)  
*Macro to set serial manager task priority.*
- #define **SERIAL\_MANAGER\_TASK\_STACK\_SIZE** (1000U)  
*Macro to set serial manager task stack size.*

## Typedefs

- typedef void \* **serial\_handle\_t**  
*The handle of the serial manager module.*
- typedef void \* **serial\_write\_handle\_t**  
*The write handle of the serial manager module.*
- typedef void \* **serial\_read\_handle\_t**  
*The read handle of the serial manager module.*
- typedef void(\* **serial\_manager\_callback\_t** )(void \*callbackParam, **serial\_manager\_callback\_message\_t** \*message, **serial\_manager\_status\_t** status)  
*serial manager callback function*
- typedef void(\* **serial\_manager\_lowpower\_critical\_callback\_t** )(void)  
*serial manager Lowpower Critical callback function*

## Enumerations

- enum `serial_port_type_t` {
   
  `kSerialPort_None` = 0U,
   
  `kSerialPort_Uart` = 1U,
   
  `kSerialPort_UsbCdc`,
   
  `kSerialPort_Swo`,
   
  `kSerialPort_Virtual`,
   
  `kSerialPort_Rpmsg`,
   
  `kSerialPort_UartDma`,
   
  `kSerialPort_SpiMaster`,
   
  `kSerialPort_SpiSlave` }
   
    *serial port type*
- enum `serial_manager_type_t` {
   
  `kSerialManager_NonBlocking` = 0x0U,
   
  `kSerialManager_Blocking` = 0x8F41U }
   
    *serial manager type*
- enum `serial_manager_status_t` {
   
  `kStatus_SerialManager_Success` = `kStatus_Success`,
   
  `kStatus_SerialManager_Error` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1)`,
   
  `kStatus_SerialManager_Busy` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2)`,
   
  `kStatus_SerialManager_Notify` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3)`,
   
  `kStatus_SerialManager_Canceled`,
   
  `kStatus_SerialManager_HandleConflict` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5)`,
   
  `kStatus_SerialManager_RingBufferOverflow`,
   
  `kStatus_SerialManager_NotConnected` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7)` }
   
    *serial manager error code*

## Functions

- `serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const serial_manager_config_t *config)`

*Initializes a serial manager module with the serial manager handle and the user configuration structure.*
- `serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)`

*De-initializes the serial manager module instance.*
- `serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_write_handle_t writeHandle)`

*Opens a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)`

*Closes a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_read_handle_t readHandle)`

*Opens a reading handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t readHandle)`

*Closes a reading for the serial manager module.*

- `serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t writeHandle, uint8_t *buffer, uint32_t length)`  
*Transmits data with the blocking mode.*
- `serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length)`  
*Reads data with the blocking mode.*
- `serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)`  
*Prepares to enter low power consumption.*
- `serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t serialHandle)`  
*Restores from low power consumption.*
- `void SerialManager_SetLowpowerCriticalCb (const serial_manager_lowpower_critical_CBs_t *pfCallback)`  
*This function performs initialization of the callbacks structure used to disable lowpower when serial manager is active.*

## 46.2 Data Structure Documentation

### 46.2.1 struct serial\_manager\_config\_t

#### Data Fields

- `uint8_t * ringBuffer`  
*Ring buffer address, it is used to buffer data received by the hardware.*
- `uint32_t ringBufferSize`  
*The size of the ring buffer.*
- `serial_port_type_t type`  
*Serial port type.*
- `serial_manager_type_t blockType`  
*Serial manager port type.*
- `void * portConfig`  
*Serial port configuration.*

#### Field Documentation

##### (1) `uint8_t* serial_manager_config_t::ringBuffer`

Besides, the memory space cannot be free during the lifetime of the serial manager module.

### 46.2.2 struct serial\_manager\_callback\_message\_t

#### Data Fields

- `uint8_t * buffer`  
*Transferred buffer.*
- `uint32_t length`  
*Transferred data length.*

## 46.3 Macro Definition Documentation

**46.3.1 #define SERIAL\_MANAGER\_WRITE\_TIME\_DELAY\_DEFAULT\_VALUE (1U)**

**46.3.2 #define SERIAL\_MANAGER\_READ\_TIME\_DELAY\_DEFAULT\_VALUE (1U)**

**46.3.3 #define SERIAL\_MANAGER\_USE\_COMMON\_TASK (0U)**

Macro to determine whether use common task.

**46.3.4 #define SERIAL\_MANAGER\_HANDLE\_SIZE (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 12U)**

**46.3.5 #define SERIAL\_MANAGER\_HANDLE\_DEFINE( *name* ) uint32\_t  
*name*[((SERIAL\_MANAGER\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) /  
 sizeof(uint32\_t))]**

This macro is used to define a 4 byte aligned serial manager handle. Then use "(serial\_handle\_t)*name*" to get the serial manager handle.

The macro should be global and could be optional. You could also define serial manager handle by yourself.

This is an example,

```
* SERIAL_MANAGER_HANDLE_DEFINE(serialManagerHandle);
*
```

Parameters

|             |                                               |
|-------------|-----------------------------------------------|
| <i>name</i> | The name string of the serial manager handle. |
|-------------|-----------------------------------------------|

**46.3.6 #define SERIAL\_MANAGER\_WRITE\_HANDLE\_DEFINE( *name* ) uint32\_t  
*name*[((SERIAL\_MANAGER\_WRITE\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) / sizeof(uint32\_t))]**

This macro is used to define a 4 byte aligned serial manager write handle. Then use "(serial\_write\_handle\_t)*name*" to get the serial manager write handle.

The macro should be global and could be optional. You could also define serial manager write handle by yourself.

This is an example,

```
* SERIAL_MANAGER_WRITE_HANDLE_DEFINE(serialManagerwriteHandle);
*
```

## Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>name</i> | The name string of the serial manager write handle. |
|-------------|-----------------------------------------------------|

**46.3.7 #define SERIAL\_MANAGER\_READ\_HANDLE\_DEFINE( *name* ) uint32\_t  
*name[((SERIAL\_MANAGER\_READ\_HANDLE\_SIZE + sizeof(uint32\_t) - 1U) /  
sizeof(uint32\_t))]***

This macro is used to define a 4 byte aligned serial manager read handle. Then use "(serial\_read\_handle\_t)*name*" to get the serial manager read handle.

The macro should be global and could be optional. You could also define serial manager read handle by yourself.

This is an example,

```
* SERIAL_MANAGER_READ_HANDLE_DEFINE(serialManagerReadHandle);
*
```

## Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>name</i> | The name string of the serial manager read handle. |
|-------------|----------------------------------------------------|

**46.3.8 #define SERIAL\_MANAGER\_TASK\_PRIORITY (2U)**

**46.3.9 #define SERIAL\_MANAGER\_TASK\_STACK\_SIZE (1000U)**

## 46.4 Enumeration Type Documentation

### 46.4.1 enum serial\_port\_type\_t

## Enumerator

- kSerialPort\_None* Serial port is none.
- kSerialPort\_Uart* Serial port UART.
- kSerialPort\_UsbCdc* Serial port USB CDC.
- kSerialPort\_Swo* Serial port SWO.
- kSerialPort\_Virtual* Serial port Virtual.
- kSerialPort\_Rpmsg* Serial port RPMSG.
- kSerialPort\_UartDma* Serial port UART DMA.

*kSerialPort\_SpiMaster* Serial port SPIMASTER.

*kSerialPort\_SpiSlave* Serial port SPISLAVE.

#### 46.4.2 enum serial\_manager\_type\_t

Enumerator

*kSerialManager\_NonBlocking* None blocking handle.

*kSerialManager\_Blocking* Blocking handle.

#### 46.4.3 enum serial\_manager\_status\_t

Enumerator

*kStatus\_SerialManager\_Success* Success.

*kStatus\_SerialManager\_Error* Failed.

*kStatus\_SerialManager\_Busy* Busy.

*kStatus\_SerialManager\_Notify* Ring buffer is not empty.

*kStatus\_SerialManager\_Canceled* the non-blocking request is canceled

*kStatus\_SerialManager\_HandleConflict* The handle is opened.

*kStatus\_SerialManager\_RingBufferOverflow* The ring buffer is overflowed.

*kStatus\_SerialManager\_NotConnected* The host is not connected.

### 46.5 Function Documentation

#### 46.5.1 serial\_manager\_status\_t SerialManager\_Init ( serial\_handle\_t *serialHandle*, const serial\_manager\_config\_t \* *config* )

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter *serialHandle* is a pointer to point to a memory space of size [SERIAL\\_MANAGER\\_HANDLE\\_SIZE](#) allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to [serial\\_port\\_type\\_t](#) for serial port setting. These three types can be set by using [serial\\_manager\\_config\\_t](#).

Example below shows how to use this API to configure the Serial Manager. For UART,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_uart_config_t uartConfig;
* config.type = kSerialPort_Uart;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* uartConfig.instance = 0;
```

```

*   uartConfig.clockRate = 24000000;
*   uartConfig.baudRate = 115200;
*   uartConfig.parityMode = kSerialManager_UartParityDisabled;
*   uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
*   uartConfig.enableRx = 1;
*   uartConfig.enableTx = 1;
*   uartConfig.enableRxRTS = 0;
*   uartConfig.enableTxCTS = 0;
*   config.portConfig = &uartConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

For USB CDC,

```

*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_usb_cdc_config_t usbCdcConfig;
*   config.type = kSerialPort_UsbCdc;
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   usbCdcConfig.controllerIndex =
*       kSerialManager_UsbControllerKhci0;
*   config.portConfig = &usbCdcConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | Pointer to point to a memory space of size <a href="#">SERIAL_MANAGER_HANDLE_SIZE</a> allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_HANDLE_DEFINE(serialHandle)</a> ; or <code>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |
| <i>config</i>       | Pointer to user-defined configuration structure.                                                                                                                                                                                                                                                                                                                                                                                                       |

Return values

|                                      |                                                   |
|--------------------------------------|---------------------------------------------------|
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                                |
| <i>kStatus_SerialManager_Success</i> | The Serial Manager module initialization succeed. |

#### 46.5.2 **serial\_manager\_status\_t SerialManager\_Deinit ( serial\_handle\_t serialHandle )**

This function de-initializes the serial manager module instance. If the opened writing or reading handle is not closed, the function will return [kStatus\\_SerialManager\\_Busy](#).

## Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

## Return values

|                                       |                                                 |
|---------------------------------------|-------------------------------------------------|
| <i>kStatus_SerialManager_-Success</i> | The serial manager de-initialization succeed.   |
| <i>kStatus_SerialManager_-Busy</i>    | Opened reading or writing handle is not closed. |

#### 46.5.3 **serial\_manager\_status\_t SerialManager\_OpenWriteHandle ( serial\_handle\_t *serialHandle*, serial\_write\_handle\_t *writeHandle* )**

This function Opens a writing handle for the serial manager module. If the serial manager needs to be used in different tasks, the task should open a dedicated write handle for itself by calling [SerialManager\\_OpenWriteHandle](#). Since there can only one buffer for transmission for the writing handle at the same time, multiple writing handles need to be opened when the multiple transmission is needed for a task.

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.                                                                                                                                                                                                                                                       |
| <i>writeHandle</i>  | The serial manager module writing handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle)</a> ; or <code>uint32_t writeHandle[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |

## Return values

|                                              |                                |
|----------------------------------------------|--------------------------------|
| <i>kStatus_SerialManager_-Error</i>          | An error occurred.             |
| <i>kStatus_SerialManager_-HandleConflict</i> | The writing handle was opened. |

|                                       |                               |
|---------------------------------------|-------------------------------|
| <i>kStatus_SerialManager_-Success</i> | The writing handle is opened. |
|---------------------------------------|-------------------------------|

Example below shows how to use this API to write data. For task 1,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle1);
* static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
, (serial_write_handle_t)s_serialWriteHandle1);
* SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle1,
, Task1_SerialManagerTxCallback,
s_serialWriteHandle1);
* SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle1,
s_nonBlockingWelcome1,
sizeof(s_nonBlockingWelcome1) - 1U);
*
```

For task 2,

```
* static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle2);
* static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
* SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
, (serial_write_handle_t)s_serialWriteHandle2);
* SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle2,
, Task2_SerialManagerTxCallback,
s_serialWriteHandle2);
* SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle2,
s_nonBlockingWelcome2,
sizeof(s_nonBlockingWelcome2) - 1U);
*
```

#### 46.5.4 serial\_manager\_status\_t SerialManager\_CloseWriteHandle ( serial\_write\_handle\_t *writeHandle* )

This function Closes a writing handle for the serial manager module.

Parameters

|                    |                                                   |
|--------------------|---------------------------------------------------|
| <i>writeHandle</i> | The serial manager module writing handle pointer. |
|--------------------|---------------------------------------------------|

Return values

|                                       |                               |
|---------------------------------------|-------------------------------|
| <i>kStatus_SerialManager_-Success</i> | The writing handle is closed. |
|---------------------------------------|-------------------------------|

#### 46.5.5 serial\_manager\_status\_t SerialManager\_OpenReadHandle ( serial\_handle\_t *serialHandle*, serial\_read\_handle\_t *readHandle* )

This function Opens a reading handle for the serial manager module. The reading handle can not be opened multiple at the same time. The error code kStatus\_SerialManager\_Busy would be returned when

the previous reading handle is not closed. And there can only be one buffer for receiving for the reading handle at the same time.

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.                                                                                                                                                                                                                                                   |
| <i>readHandle</i>   | The serial manager module reading handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle)</a> ; or <code>uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |

## Return values

|                                      |                                        |
|--------------------------------------|----------------------------------------|
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                     |
| <i>kStatus_SerialManager_Success</i> | The reading handle is opened.          |
| <i>kStatus_SerialManager_Busy</i>    | Previous reading handle is not closed. |

Example below shows how to use this API to read data.

```
* static SERIAL_MANAGER_READ_HANDLE_DEFINE(s_serialReadHandle);
* SerialManager_OpenReadHandle((serial_handle_t)serialHandle,
*     (serial_read_handle_t)s_serialReadHandle);
* static uint8_t s_nonBlockingBuffer[64];
* SerialManager_InstallRxCallback((serial_read_handle_t)s_serialReadHandle,
*     APP_SerialManagerRxCallback,
*     s_serialReadHandle);
* SerialManager_ReadNonBlocking((serial_read_handle_t)s_serialReadHandle,
*     s_nonBlockingBuffer,
*     sizeof(s_nonBlockingBuffer));
*
```

#### 46.5.6 **serial\_manager\_status\_t SerialManager\_CloseReadHandle ( serial\_read\_handle\_t *readHandle* )**

This function Closes a reading for the serial manager module.

## Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>readHandle</i> | The serial manager module reading handle pointer. |
|-------------------|---------------------------------------------------|

Return values

|                                       |                               |
|---------------------------------------|-------------------------------|
| <i>kStatus_SerialManager_-Success</i> | The reading handle is closed. |
|---------------------------------------|-------------------------------|

#### 46.5.7 `serial_manager_status_t SerialManager_WriteBlocking ( serial_write_handle_t writeHandle, uint8_t * buffer, uint32_t length )`

This is a blocking function, which polls the sending queue, waits for the sending queue to be empty. This function sends data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for transmission for the writing handle at the same time.

Note

The function `SerialManager_WriteBlocking` and the function `SerialManager_WriteNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelWriting` cannot be used to abort the transmission of this function.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
| <i>buffer</i>      | Start address of the data to write.       |
| <i>length</i>      | Length of the data to write.              |

Return values

|                                       |                                                                  |
|---------------------------------------|------------------------------------------------------------------|
| <i>kStatus_SerialManager_-Success</i> | Successfully sent all data.                                      |
| <i>kStatus_SerialManager_-Busy</i>    | Previous transmission still not finished; data not all sent yet. |
| <i>kStatus_SerialManager_-Error</i>   | An error occurred.                                               |

#### 46.5.8 `serial_manager_status_t SerialManager_ReadBlocking ( serial_read_handle_t readHandle, uint8_t * buffer, uint32_t length )`

This is a blocking function, which polls the receiving buffer, waits for the receiving buffer to be full. This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

## Note

The function `SerialManager_ReadBlocking` and the function `SerialManager_ReadNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelReading` cannot be used to abort the transmission of this function.

## Parameters

|                         |                                                       |
|-------------------------|-------------------------------------------------------|
| <code>readHandle</code> | The serial manager module handle pointer.             |
| <code>buffer</code>     | Start address of the data to store the received data. |
| <code>length</code>     | The length of the data to be received.                |

## Return values

|                                             |                                                                      |
|---------------------------------------------|----------------------------------------------------------------------|
| <code>kStatus_SerialManager_-Success</code> | Successfully received all data.                                      |
| <code>kStatus_SerialManager_-Busy</code>    | Previous transmission still not finished; data not all received yet. |
| <code>kStatus_SerialManager_-Error</code>   | An error occurred.                                                   |

#### 46.5.9 `serial_manager_status_t SerialManager_EnterLowpower ( serial_handle_t serialHandle )`

This function is used to prepare to enter low power consumption.

## Parameters

|                           |                                           |
|---------------------------|-------------------------------------------|
| <code>serialHandle</code> | The serial manager module handle pointer. |
|---------------------------|-------------------------------------------|

## Return values

|                                             |                       |
|---------------------------------------------|-----------------------|
| <code>kStatus_SerialManager_-Success</code> | Successful operation. |
|---------------------------------------------|-----------------------|

#### 46.5.10 `serial_manager_status_t SerialManager_ExitLowpower ( serial_handle_t serialHandle )`

This function is used to restore from low power consumption.

Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

Return values

|                                       |                       |
|---------------------------------------|-----------------------|
| <i>kStatus_SerialManager_-Success</i> | Successful operation. |
|---------------------------------------|-----------------------|

### 46.5.11 void SerialManager\_SetLowpowerCriticalCb ( const serial\_manager\_lowpower\_critical\_CBs\_t \* *pfCallback* )

Parameters

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <i>pfCallback</i> | Pointer to the function structure used to allow/disable lowpower. |
|-------------------|-------------------------------------------------------------------|

## 46.6 Serial Port Uart

### 46.6.1 Overview

#### Macros

- #define **SERIAL\_PORT\_UART\_DMA\_RECEIVE\_DATA\_LENGTH** (64U)  
*serial port uart handle size*
- #define **SERIAL\_USE\_CONFIGURE\_STRUCTURE** (0U)  
*Enable or disable the configure structure pointer.*

#### Enumerations

- enum **serial\_port\_uart\_parity\_mode\_t** {
   
    **kSerialManager\_UartParityDisabled** = 0x0U,
   
    **kSerialManager\_UartParityEven** = 0x2U,
   
    **kSerialManager\_UartParityOdd** = 0x3U }
   
*serial port uart parity mode*
- enum **serial\_port\_uart\_stop\_bit\_count\_t** {
   
    **kSerialManager\_UartOneStopBit** = 0U,
   
    **kSerialManager\_UartTwoStopBit** = 1U }
   
*serial port uart stop bit count*

### 46.6.2 Enumeration Type Documentation

#### 46.6.2.1 enum serial\_port\_uart\_parity\_mode\_t

Enumerator

**kSerialManager\_UartParityDisabled** Parity disabled.  
**kSerialManager\_UartParityEven** Parity even enabled.  
**kSerialManager\_UartParityOdd** Parity odd enabled.

#### 46.6.2.2 enum serial\_port\_uart\_stop\_bit\_count\_t

Enumerator

**kSerialManager\_UartOneStopBit** One stop bit.  
**kSerialManager\_UartTwoStopBit** Two stop bits.

## 46.7 Serial Port USB

### 46.7.1 Overview

#### Modules

- [USB Device Configuration](#)

#### Data Structures

- struct [serial\\_port\\_usb\\_cdc\\_config\\_t](#)  
*serial port usb config struct* [More...](#)

#### Macros

- #define [SERIAL\\_PORT\\_USB\\_CDC\\_HANDLE\\_SIZE](#) (72U)  
*serial port usb handle size*
- #define [USB\\_DEVICE\\_INTERRUPT\\_PRIORITY](#) (3U)  
*USB interrupt priority.*

#### Enumerations

- enum [serial\\_port\\_usb\\_cdc\\_controller\\_index\\_t](#) {
   
   kSerialManager\_UsbControllerKhci0 = 0U,  
   kSerialManager\_UsbControllerKhci1 = 1U,  
   kSerialManager\_UsbControllerEhci0 = 2U,  
   kSerialManager\_UsbControllerEhci1 = 3U,  
   kSerialManager\_UsbControllerLpcIp3511Fs0 = 4U,  
   kSerialManager\_UsbControllerLpcIp3511Fs1 = 5U,  
   kSerialManager\_UsbControllerLpcIp3511Hs0 = 6U,  
   kSerialManager\_UsbControllerLpcIp3511Hs1 = 7U,  
   kSerialManager\_UsbControllerOhci0 = 8U,  
   kSerialManager\_UsbControllerOhci1 = 9U,  
   kSerialManager\_UsbControllerIp3516Hs0 = 10U,  
   kSerialManager\_UsbControllerIp3516Hs1 = 11U }
   
*USB controller ID.*

### 46.7.2 Data Structure Documentation

#### 46.7.2.1 struct serial\_port\_usb\_cdc\_config\_t

##### Data Fields

- `serial_port_usb_cdc_controller_index_t controllerIndex`  
*controller index*

#### 46.7.3 Enumeration Type Documentation

##### 46.7.3.1 enum serial\_port\_usb\_cdc\_controller\_index\_t

Enumerator

`kSerialManager_UsbControllerKhci0` KHCI 0U.

`kSerialManager_UsbControllerKhci1` KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerEhci0` EHCI 0U.

`kSerialManager_UsbControllerEhci1` EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerLpcIp3511Fs0` LPC USB IP3511 FS controller 0.

`kSerialManager_UsbControllerLpcIp3511Fs1` LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerLpcIp3511Hs0` LPC USB IP3511 HS controller 0.

`kSerialManager_UsbControllerLpcIp3511Hs1` LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerOhci0` OHCI 0U.

`kSerialManager_UsbControllerOhci1` OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

`kSerialManager_UsbControllerIp3516Hs0` IP3516HS 0U.

`kSerialManager_UsbControllerIp3516Hs1` IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

#### 46.7.4 USB Device Configuration

## 46.8 Serial Port SWO

### 46.8.1 Overview

#### Data Structures

- struct `serial_port_swo_config_t`  
*serial port swo config struct* [More...](#)

#### Macros

- #define `SERIAL_PORT_SWO_HANDLE_SIZE` (12U)  
*serial port swo handle size*

#### Enumerations

- enum `serial_port_swo_protocol_t` {
   
`kSerialManager_SwoProtocolManchester` = 1U,  
`kSerialManager_SwoProtocolNrz` = 2U }
   
*serial port swo protocol*

### 46.8.2 Data Structure Documentation

#### 46.8.2.1 struct `serial_port_swo_config_t`

##### Data Fields

- `uint32_t clockRate`  
*clock rate*
- `uint32_t baudRate`  
*baud rate*
- `uint32_t port`  
*Port used to transfer data.*
- `serial_port_swo_protocol_t protocol`  
*SWO protocol.*

### 46.8.3 Enumeration Type Documentation

#### 46.8.3.1 enum `serial_port_swo_protocol_t`

Enumerator

`kSerialManager_SwoProtocolManchester` SWO Manchester protocol.  
`kSerialManager_SwoProtocolNrz` SWO UART/NRZ protocol.

# Chapter 47

## I2s\_dma\_driver

### 47.1 Overview

#### Data Structures

- struct [i2s\\_dma\\_handle\\_t](#)  
*i2s dma handle* [More...](#)

#### Typedefs

- [typedef void\(\\* i2s\\_dma\\_transfer\\_callback\\_t \)\(I2S\\_Type \\*base, i2s\\_dma\\_handle\\_t \\*handle, status\\_t completionStatus, void \\*userData\)](#)  
*Callback function invoked from DMA API on completion.*

#### Driver version

- [#define FSL\\_I2S\\_DMA\\_DRIVER\\_VERSION \(MAKE\\_VERSION\(2, 3, 2\)\)](#)  
*I2S DMA driver version 2.3.2.*

#### DMA API

- [void I2S\\_TxTransferCreateHandleDMA \(I2S\\_Type \\*base, i2s\\_dma\\_handle\\_t \\*handle, dma\\_handle\\_t \\*dmaHandle, i2s\\_dma\\_transfer\\_callback\\_t callback, void \\*userData\)](#)  
*Initializes handle for transfer of audio data.*
- [status\\_t I2S\\_TxTransferSendDMA \(I2S\\_Type \\*base, i2s\\_dma\\_handle\\_t \\*handle, i2s\\_transfer\\_t transfer\)](#)  
*Begins or queue sending of the given data.*
- [void I2S\\_TransferAbortDMA \(I2S\\_Type \\*base, i2s\\_dma\\_handle\\_t \\*handle\)](#)  
*Aborts transfer of data.*
- [void I2S\\_RxTransferCreateHandleDMA \(I2S\\_Type \\*base, i2s\\_dma\\_handle\\_t \\*handle, dma\\_handle\\_t \\*dmaHandle, i2s\\_dma\\_transfer\\_callback\\_t callback, void \\*userData\)](#)  
*Initializes handle for reception of audio data.*
- [status\\_t I2S\\_RxTransferReceiveDMA \(I2S\\_Type \\*base, i2s\\_dma\\_handle\\_t \\*handle, i2s\\_transfer\\_t transfer\)](#)  
*Begins or queue reception of data into given buffer.*
- [void I2S\\_DMACallback \(dma\\_handle\\_t \\*handle, void \\*userData, bool transferDone, uint32\\_t tcds\)](#)  
*Invoked from DMA interrupt handler.*
- [void I2S\\_TransferInstallLoopDMADescriptorMemory \(i2s\\_dma\\_handle\\_t \\*handle, void \\*dmaDescriptorAddr, size\\_t dmaDescriptorNum\)](#)  
*Install DMA descriptor memory for loop transfer only.*
- [status\\_t I2S\\_TransferSendLoopDMA \(I2S\\_Type \\*base, i2s\\_dma\\_handle\\_t \\*handle, i2s\\_transfer\\_t \\*xfer, uint32\\_t loopTransferCount\)](#)  
*Send link transfer data using DMA.*

- **status\_t I2S\_TransferReceiveLoopDMA** (I2S\_Type \*base, i2s\_dma\_handle\_t \*handle, i2s\_transfer\_t \*xfer, uint32\_t loopTransferCount)  
*Receive link transfer data using DMA.*

## 47.2 Data Structure Documentation

### 47.2.1 struct \_i2s\_dma\_handle

Members not to be accessed / modified outside of the driver.

#### Data Fields

- **uint32\_t state**  
*Internal state of I2S DMA transfer.*
- **uint8\_t bytesPerFrame**  
*bytes per frame*
- **i2s\_dma\_transfer\_callback\_t completionCallback**  
*Callback function pointer.*
- **void \*userData**  
*Application data passed to callback.*
- **dma\_handle\_t \*dmaHandle**  
*DMA handle.*
- **volatile i2s\_transfer\_t i2sQueue [I2S\_NUM\_BUFFERS]**  
*Transfer queue storing transfer buffers.*
- **volatile uint8\_t queueUser**  
*Queue index where user's next transfer will be stored.*
- **volatile uint8\_t queueDriver**  
*Queue index of buffer actually used by the driver.*
- **dma\_descriptor\_t \*i2sLoopDMADescriptor**  
*descriptor pool pointer*
- **size\_t i2sLoopDMADescriptorNum**  
*number of descriptor in descriptors pool*

## 47.3 Macro Definition Documentation

### 47.3.1 #define FSL\_I2S\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 2))

## 47.4 Typedef Documentation

### 47.4.1 **typedef void(\* i2s\_dma\_transfer\_callback\_t)(I2S\_Type \*base, i2s\_dma\_handle\_t \*handle, status\_t completionStatus, void \*userData)**

Parameters

|                          |                                          |
|--------------------------|------------------------------------------|
| <i>base</i>              | I2S base pointer.                        |
| <i>handle</i>            | pointer to I2S transaction.              |
| <i>completion-Status</i> | status of the transaction.               |
| <i>userData</i>          | optional pointer to user arguments data. |

## 47.5 Function Documentation

**47.5.1 void I2S\_TxTransferCreateHandleDMA ( *I2S\_Type* \* *base*, *i2s\_dma\_handle\_t* \* *handle*, *dma\_handle\_t* \* *dmaHandle*, *i2s\_dma\_transfer\_callback\_t* *callback*, *void* \* *userData* )**

Parameters

|                  |                                                            |
|------------------|------------------------------------------------------------|
| <i>base</i>      | I2S base pointer.                                          |
| <i>handle</i>    | pointer to handle structure.                               |
| <i>dmaHandle</i> | pointer to dma handle structure.                           |
| <i>callback</i>  | function to be called back when transfer is done or fails. |
| <i>userData</i>  | pointer to data passed to callback.                        |

**47.5.2 status\_t I2S\_TxTransferSendDMA ( *I2S\_Type* \* *base*, *i2s\_dma\_handle\_t* \* *handle*, *i2s\_transfer\_t* *transfer* )**

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | I2S base pointer.            |
| <i>handle</i>   | pointer to handle structure. |
| <i>transfer</i> | data buffer.                 |

Return values

|                        |
|------------------------|
| <i>kStatus_Success</i> |
|------------------------|

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <i>kStatus_I2S_Busy</i> | if all queue slots are occupied with unsent buffers. |
|-------------------------|------------------------------------------------------|

#### 47.5.3 void I2S\_TransferAbortDMA ( *I2S\_Type* \* *base*, *i2s\_dma\_handle\_t* \* *handle* )

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | I2S base pointer.            |
| <i>handle</i> | pointer to handle structure. |

#### 47.5.4 void I2S\_RxTransferCreateHandleDMA ( *I2S\_Type* \* *base*, *i2s\_dma\_handle\_t* \* *handle*, *dma\_handle\_t* \* *dmaHandle*, *i2s\_dma\_transfer\_callback\_t* *callback*, *void* \* *userData* )

Parameters

|                  |                                                            |
|------------------|------------------------------------------------------------|
| <i>base</i>      | I2S base pointer.                                          |
| <i>handle</i>    | pointer to handle structure.                               |
| <i>dmaHandle</i> | pointer to dma handle structure.                           |
| <i>callback</i>  | function to be called back when transfer is done or fails. |
| <i>userData</i>  | pointer to data passed to callback.                        |

#### 47.5.5 *status\_t* I2S\_RxTransferReceiveDMA ( *I2S\_Type* \* *base*, *i2s\_dma\_handle\_t* \* *handle*, *i2s\_transfer\_t* *transfer* )

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | I2S base pointer.            |
| <i>handle</i>   | pointer to handle structure. |
| <i>transfer</i> | data buffer.                 |

Return values

|                         |                                                                  |
|-------------------------|------------------------------------------------------------------|
| <i>kStatus_Success</i>  |                                                                  |
| <i>kStatus_I2S_Busy</i> | if all queue slots are occupied with buffers which are not full. |

#### 47.5.6 void I2S\_DMACallback ( *dma\_handle\_t \* handle, void \* userData, bool transferDone, uint32\_t tcds* )

Parameters

|                     |                                  |
|---------------------|----------------------------------|
| <i>handle</i>       | pointer to DMA handle structure. |
| <i>userData</i>     | argument for user callback.      |
| <i>transferDone</i> | if transfer was done.            |
| <i>tcds</i>         |                                  |

#### 47.5.7 void I2S\_TransferInstallLoopDMADescriptorMemory ( *i2s\_dma\_handle\_t \* handle, void \* dmaDescriptorAddr, size\_t dmaDescriptorNum* )

This function used to register DMA descriptor memory for the i2s loop dma transfer.

It must be called before I2S\_TransferSendLoopDMA/I2S\_TransferReceiveLoopDMA and after I2S\_RxTransferCreateHandleDMA/I2S\_TxTransferCreateHandleDMA.

User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

Parameters

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>handle</i>              | Pointer to i2s DMA transfer handle. |
| <i>dma-Descriptor-Addr</i> | DMA descriptor start address.       |
| <i>dma-DescriptorNum</i>   | DMA descriptor number.              |

#### 47.5.8 status\_t I2S\_TransferSendLoopDMA ( *I2S\_Type \* base*, *i2s\_dma\_handle\_t \* handle*, *i2s\_transfer\_t \* xfer*, *uint32\_t loopTransferCount* )

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S\_InstallDMADescriptor-Memory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S\_TransferAbortDMA to stop the loop transfer.

Parameters

|                           |                                                                  |
|---------------------------|------------------------------------------------------------------|
| <i>base</i>               | I2S peripheral base address.                                     |
| <i>handle</i>             | Pointer to usart_dma_handle_t structure.                         |
| <i>xfer</i>               | I2S DMA transfer structure. See <a href="#">i2s_transfer_t</a> . |
| <i>loopTransfer-Count</i> | loop count                                                       |

Return values

|                        |
|------------------------|
| <i>kStatus_Success</i> |
|------------------------|

#### 47.5.9 status\_t I2S\_TransferReceiveLoopDMA ( *I2S\_Type \* base*, *i2s\_dma\_handle\_t \* handle*, *i2s\_transfer\_t \* xfer*, *uint32\_t loopTransferCount* )

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S\_InstallDMADescriptor-Memory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S\_TransferAbortDMA to stop the loop transfer.

## Parameters

|                          |                                                                  |
|--------------------------|------------------------------------------------------------------|
| <i>base</i>              | I2S peripheral base address.                                     |
| <i>handle</i>            | Pointer to usart_dma_handle_t structure.                         |
| <i>xfer</i>              | I2S DMA transfer structure. See <a href="#">i2s_transfer_t</a> . |
| <i>loopTransferCount</i> | loop count                                                       |

## Return values

|                        |  |
|------------------------|--|
| <i>kStatus_Success</i> |  |
|------------------------|--|

# Chapter 48

## Hashcrypt\_driver

### 48.1 Overview

#### Macros

- `#define HASHCRYPT_MODE_SHA1 0x1`  
*Algorithm definitions correspond with the values for Mode field in Control register !*

#### Enumerations

- enum `hashcrypt_algo_t` {  
  `kHASHCRYPT_Sha1` = HASHCRYPT\_MODE\_SHA1,  
  `kHASHCRYPT_Sha256` = HASHCRYPT\_MODE\_SHA256,  
  `kHASHCRYPT_Aes` = HASHCRYPT\_MODE\_AES }  
*Algorithm used for Hashcrypt operation.*

#### Functions

- void `HASHCRYPT_Init` (HASHCRYPT\_Type \*base)  
*Enables clock and disables reset for HASHCRYPT peripheral.*
- void `HASHCRYPT_Deinit` (HASHCRYPT\_Type \*base)  
*Disables clock for HASHCRYPT peripheral.*

#### Driver version

- `#define FSL_HASHCRYPT_DRIVER_VERSION (MAKE_VERSION(2, 2, 6))`  
*HASHCRYPT driver version.*

### 48.2 Macro Definition Documentation

#### 48.2.1 `#define FSL_HASHCRYPT_DRIVER_VERSION (MAKE_VERSION(2, 2, 6))`

Version 2.2.6.

Current version: 2.2.6

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Support loading AES key from unaligned address
- Version 2.0.2
  - Support loading AES key from unaligned address for different compiler and core variants

- Version 2.0.3
  - Remove SHA512 and AES ICB algorithm definitions
- Version 2.0.4
  - Add SHA context switch support
- Version 2.1.0
  - Update the register name and macro to align with new header.
- Version 2.1.1
  - Fix MISRA C-2012.
- Version 2.1.2
  - Support loading AES input data from unaligned address.
- Version 2.1.3
  - Fix MISRA C-2012.
- Version 2.1.4
  - Fix context switch cannot work when switching from AES.
- Version 2.1.5
  - Add data synchronization barrier inside hashcrypt\_sha\_ldm\_stm\_16\_words() to prevent possible optimization issue.
- Version 2.2.0
  - Add AES-OFB and AES-CFB mixed IP/SW modes.
- Version 2.2.1
  - Add data synchronization barrier inside hashcrypt\_sha\_ldm\_stm\_16\_words() prevent compiler from reordering memory write when -O2 or higher is used.
- Version 2.2.2
  - Add data synchronization barrier inside hashcrypt\_sha\_ldm\_stm\_16\_words() to fix optimization issue
- Version 2.2.3
  - Added check for size in hashcrypt\_aes\_one\_block to prevent overflowing COUNT field in MEMCTRL register, if its bigger than COUNT field do a multiple runs.
- Version 2.2.4
  - In all HASHCRYPT\_AES\_xx functions have been added setting CTRL\_MODE bitfield to 0 after processing data, which decreases power consumption.
- Version 2.2.5
  - Add data synchronization barrier and instruction synchronization barrier inside hashcrypt\_sha\_process\_message\_data() to fix optimization issue
- Version 2.2.6
  - Add data synchronization barrier inside [HASHCRYPT\\_SHA\\_Update\(\)](#) and hashcrypt\_get\_data() function to fix optimization issue on MDK and ARMGCC release targets

## 48.3 Enumeration Type Documentation

### 48.3.1 enum hashcrypt\_algo\_t

Enumerator

*kHASHCRYPT\_Sha1* SHA\_1.

*kHASHCRYPT\_Sha256* SHA\_256.  
*kHASHCRYPT\_Aes* AES.

## 48.4 Function Documentation

### 48.4.1 void HASHCRYPT\_Init ( HASHCRYPT\_Type \* *base* )

Enable clock and disable reset for HASHCRYPT.

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | HASHCRYPT base address |
|-------------|------------------------|

### 48.4.2 void HASHCRYPT\_Deinit ( HASHCRYPT\_Type \* *base* )

Disable clock and enable reset.

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | HASHCRYPT base address |
|-------------|------------------------|

# Chapter 49

## Skboot\_authenticate

### 49.1 Overview

#### Enumerations

- enum `skboot_status_t` {  
    `kStatus_SKBOOT_Success` = 0x5ac3c35au,  
    `kStatus_SKBOOT_Fail` = 0xc35ac35au,  
    `kStatus_SKBOOT_InvalidArgument` = 0xc35a5ac3u,  
    `kStatus_SKBOOT_KeyStoreMarkerInvalid` = 0xc3c35a5au,  
    `kStatus_SKBOOT_HashcryptFinishedWithStatusSuccess`,  
    `kStatus_SKBOOT_HashcryptFinishedWithStatusFail`,  
    `kStatus_SKBOOT_Success` = 0x5ac3c35au,  
    `kStatus_SKBOOT_Fail` = 0xc35ac35au,  
    `kStatus_SKBOOT_InvalidArgument` = 0xc35a5ac3u,  
    `kStatus_SKBOOT_KeyStoreMarkerInvalid` = 0xc3c35a5au }  
        *SKBOOT return status.*
- enum `secure_bool_t` {  
    `kSECURE_TRUE` = 0xc33cc33cU,  
    `kSECURE_FALSE` = 0x5aa55aa5U,  
    `kSECURE_CALLPROTECT_SECURITY_FLAGS` = 0xc33c5aa5U,  
    `kSECURE_CALLPROTECT_IS_APP_READY` = 0x5aa5c33cU,  
    `kSECURE_TRACKER_VERIFIED` = 0x55aacc33U,  
    `kSECURE_TRUE` = 0xc33cc33cU,  
    `kSECURE_FALSE` = 0x5aa55aa5U }  
        *Secure bool flag.*

#### Functions

- `skboot_status_t skboot_authenticate (const uint8_t *imageStartAddr, secure_bool_t *isSignedVerified)`  
        *Authenticate entry function with ARENA allocator init.*
- void `HASH_IRQHandler (void)`  
        *Interface for image authentication API.*

### 49.2 Enumeration Type Documentation

#### 49.2.1 enum `skboot_status_t`

Enumerator

`kStatus_SKBOOT_Success` SKBOOT return success status.

***kStatus\_SKBOOT\_Fail*** SKBOOT return fail status.  
***kStatus\_SKBOOT\_InvalidArgument*** SKBOOT return invalid argument status.  
***kStatus\_SKBOOT\_KeyStoreMarkerInvalid*** SKBOOT return Keystore invalid Marker status.  
***kStatus\_SKBOOT\_HashcryptFinishedWithStatusSuccess*** SKBOOT return Hashcrypt finished with the success status.  
***kStatus\_SKBOOT\_HashcryptFinishedWithStatusFail*** SKBOOT return Hashcrypt finished with the fail status.  
***kStatus\_SKBOOT\_Success*** PRINCE Success.  
***kStatus\_SKBOOT\_Fail*** PRINCE Fail.  
***kStatus\_SKBOOT\_InvalidArgument*** PRINCE Invalid argument.  
***kStatus\_SKBOOT\_KeyStoreMarkerInvalid*** PRINCE Invalid marker.

## 49.2.2 enum secure\_bool\_t

Enumerator

***kSECURE\_TRUE*** Secure true flag.  
***kSECURE\_FALSE*** Secure false flag.  
***kSECURE\_CALLPROTECT\_SECURITY\_FLAGS*** Secure call protect the security flag.  
***kSECURE\_CALLPROTECT\_IS\_APP\_READY*** Secure call protect the app is ready flag.  
***kSECURE\_TRACKER\_VERIFIED*** Secure tracker verified flag.  
***kSECURE\_TRUE*** PRINCE true.  
***kSECURE\_FALSE*** PRINCE false.

## 49.3 Function Documentation

### 49.3.1 skboot\_status\_t skboot\_authenticate ( *const uint8\_t \*imageStartAddr*, *secure\_bool\_t \*isSignVerified* )

This is called by ROM boot or by ROM API g\_skbootAuthenticateInterface

## 49.3.2 CODEC Adapter

### 49.3.2.1 Overview

#### Enumerations

- enum {  
  kCODEC\_WM8904,  
  kCODEC\_WM8960,  
  kCODEC\_WM8524,  
  kCODEC\_SGTL5000,  
  kCODEC\_DA7212,  
  kCODEC\_CS42888,  
  kCODEC\_CS42448,  
  kCODEC\_AK4497,  
  kCODEC\_AK4458,  
  kCODEC\_TFA9XXX,  
  kCODEC\_TFA9896,  
  kCODEC\_WM8962 }  
    *codec type*

### 49.3.2.2 Enumeration Type Documentation

#### 49.3.2.2.1 anonymous enum

Enumerator

*kCODEC\_WM8904* wm8904  
*kCODEC\_WM8960* wm8960  
*kCODEC\_WM8524* wm8524  
*kCODEC\_SGTL5000* sgtl5000  
*kCODEC\_DA7212* da7212  
*kCODEC\_CS42888* CS42888.  
*kCODEC\_CS42448* CS42448.  
*kCODEC\_AK4497* AK4497.  
*kCODEC\_AK4458* ak4458  
*kCODEC\_TFA9XXX* tfa9xxx  
*kCODEC\_TFA9896* tfa9896  
*kCODEC\_WM8962* wm8962

**How to Reach Us:**

**Home Page:**

[nxp.com](http://nxp.com)

**Web Support:**

[nxp.com/support](http://nxp.com/support)

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 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](http://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, Freescale, the Freescale logo, Kinetis, Processor Expert, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, Mbed, Mbed Enabled, and Vision are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. 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.

© 2021 NXP B.V.

