

Document Number: MCUXSDKAPIRM  
Rev 2.16.100  
Sept 2024

# 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</b>	<b>Overview</b>	<b>7</b>
<b>4.2</b>	<b>Data Structure Documentation</b>	<b>12</b>
4.2.1	struct clock_frg_clk_config_t	12
4.2.2	struct clock_avpll_config_t	12
<b>4.3</b>	<b>Macro Definition Documentation</b>	<b>13</b>
4.3.1	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	13
4.3.2	FSL_CLOCK_DRIVER_VERSION	13
4.3.3	GPIO_CLOCKS	13
4.3.4	CACHE64_CLOCKS	13
4.3.5	FLEXSPI_CLOCKS	13
4.3.6	FLEXCOMM_CLOCKS	14
4.3.7	USART_CLOCKS	14
4.3.8	I2C_CLOCKS	14
4.3.9	SPI_CLOCKS	14
4.3.10	ACOMP_CLOCKS	14
4.3.11	ADC_CLOCKS	15
4.3.12	DAC_CLOCKS	15
4.3.13	LCDIC_CLOCKS	15
4.3.14	DMA_CLOCKS	15
4.3.15	DMIC_CLOCKS	15
4.3.16	ENET_CLOCKS	16
4.3.17	ENET_EXTRA_CLOCKS	16
4.3.18	CTIMER_CLOCKS	16
4.3.19	UTICK_CLOCKS	16
4.3.20	MRT_CLOCKS	16
4.3.21	SCT_CLOCKS	17
4.3.22	RTC_CLOCKS	17
4.3.23	WWDT_CLOCKS	17

Section No.	Title	Page No.
4.3.24	<a href="#">TRNG_CLOCKS</a>	17
4.3.25	<a href="#">USIM_CLOCKS</a>	17
4.3.26	<a href="#">CLK_GATE_REG_OFFSET_SHIFT</a>	18
<b>4.4</b>	<b><a href="#">Enumeration Type Documentation</a></b>	<b>18</b>
4.4.1	<a href="#">clock_name_t</a>	18
4.4.2	<a href="#">clock_tcpcu_flexspi_div_t</a>	18
4.4.3	<a href="#">clock_tddr_flexspi_div_t</a>	18
4.4.4	<a href="#">clock_t3_mci irc_config_t</a>	18
4.4.5	<a href="#">clock_avpll_ch_freq_t</a>	18
<b>4.5</b>	<b><a href="#">Function Documentation</a></b>	<b>19</b>
4.5.1	<a href="#">CLOCK_GetT3PllMciIrcClkFreq</a>	19
4.5.2	<a href="#">CLOCK_GetT3PllMci213mClkFreq</a>	19
4.5.3	<a href="#">CLOCK_GetT3PllMci256mClkFreq</a>	19
4.5.4	<a href="#">CLOCK_GetT3PllMciFlexspiClkFreq</a>	19
4.5.5	<a href="#">CLOCK_GetTcpuMciClkFreq</a>	19
4.5.6	<a href="#">CLOCK_GetTcpuMciFlexspiClkFreq</a>	20
4.5.7	<a href="#">CLOCK_GetTddrMciFlexspiClkFreq</a>	20
4.5.8	<a href="#">CLOCK_GetTddrMciEnetClkFreq</a>	20
4.5.9	<a href="#">CLOCK_EnableClock</a>	20
4.5.10	<a href="#">CLOCK_DisableClock</a>	20
4.5.11	<a href="#">CLOCK_AttachClk</a>	20
4.5.12	<a href="#">CLOCK_SetClkDiv</a>	21
4.5.13	<a href="#">CLOCK_GetFreq</a>	21
4.5.14	<a href="#">CLOCK_GetFRGClock</a>	21
4.5.15	<a href="#">CLOCK_SetFRGClock</a>	21
4.5.16	<a href="#">CLOCK_GetFFroFreq</a>	21
4.5.17	<a href="#">CLOCK_GetSFroFreq</a>	22
4.5.18	<a href="#">CLOCK_GetAvPllCh1Freq</a>	22
4.5.19	<a href="#">CLOCK_GetAvPllCh2Freq</a>	22
4.5.20	<a href="#">CLOCK_GetMainClkFreq</a>	22
4.5.21	<a href="#">CLOCK_GetCoreSysClkFreq</a>	22
4.5.22	<a href="#">CLOCK_GetSystickClkFreq</a>	22
4.5.23	<a href="#">CLOCK_GetSysOscFreq</a>	23
4.5.24	<a href="#">CLOCK_GetMclkInClkFreq</a>	23
4.5.25	<a href="#">CLOCK_GetLpOscFreq</a>	23
4.5.26	<a href="#">CLOCK_GetClk32KFreq</a>	23
4.5.27	<a href="#">CLOCK_EnableXtal32K</a>	23
4.5.28	<a href="#">CLOCK_EnableRtc32K</a>	23
4.5.29	<a href="#">CLOCK_SetClkinFreq</a>	24
4.5.30	<a href="#">CLOCK_SetMcclinFreq</a>	24
4.5.31	<a href="#">CLOCK_GetDmicClkFreq</a>	24
4.5.32	<a href="#">CLOCK_GetLcdClkFreq</a>	24
4.5.33	<a href="#">CLOCK_GetWdtClkFreq</a>	24

Section No.	Title	Page No.
4.5.34	<code>CLOCK_GetMclkClkFreq</code>	25
4.5.35	<code>CLOCK_GetSctClkFreq</code>	25
4.5.36	<code>CLOCK_GetFlexCommClkFreq</code>	25
4.5.37	<code>CLOCK_GetCTimerClkFreq</code>	25
4.5.38	<code>CLOCK_GetUtickClkFreq</code>	25
4.5.39	<code>CLOCK_GetFlexspiClkFreq</code>	26
4.5.40	<code>CLOCK_GetUsimClkFreq</code>	26
4.5.41	<code>CLOCK_GetGauClkFreq</code>	26
4.5.42	<code>CLOCK_GetOSTimerClkFreq</code>	26
4.5.43	<code>CLOCK_InitTcpuRefClk</code>	26
4.5.44	<code>CLOCK_InitTddrRefClk</code>	26
4.5.45	<code>CLOCK_InitT3RefClk</code>	27
4.5.46	<code>CLOCK_DeinitT3RefClk</code>	27
4.5.47	<code>CLOCK_InitAvPll</code>	27
4.5.48	<code>CLOCK_DeinitAvPll</code>	27
4.5.49	<code>CLOCK_ConfigAvPllCh</code>	27
4.5.50	<code>CLOCK_EnableAvPllCh</code>	28
4.5.51	<code>CLOCK_DisableAvPllCh</code>	29
4.5.52	<code>CLOCK_EnableUsbhsPhyClock</code>	29
4.5.53	<code>CLOCK_DisableUsbhsPhyClock</code>	29
<b>4.6</b>	<b>Variable Documentation</b>	<b>29</b>
4.6.1	<code>g_clkinFreq</code>	29
4.6.2	<code>g_mclkInFreq</code>	29

## Chapter 5 I2S\_BRIDGE: I2S bridging and signal sharing configuration

## Chapter 6 IO\_MUX Driver

<b>6.1</b>	<b>Overview</b>	<b>31</b>
<b>6.2</b>	<b>Macro Definition Documentation</b>	<b>35</b>
6.2.1	<code>FSL_IO_MUX_DRIVER_VERSION</code>	35
<b>6.3</b>	<b>Enumeration Type Documentation</b>	<b>36</b>
6.3.1	<code>io_mux_pin_config_t</code>	36
<b>6.4</b>	<b>Function Documentation</b>	<b>36</b>
6.4.1	<code>IO_MUX_SetPinMux</code>	36
6.4.2	<code>IO_MUX_SetPinConfig</code>	37
6.4.3	<code>IO_MUX_SetPinOutLevelInSleep</code>	37
6.4.4	<code>IO_MUX_SetRfPinOutLevelInSleep</code>	38

Section No.	Title	Page No.
<b>Chapter 7 IPED Driver</b>		
<b>8.1</b>	<b>Overview</b>	<b>40</b>
<b>8.2</b>	<b>Macro Definition Documentation</b>	<b>41</b>
8.2.1	FSL_OCOTP_DRIVER_VERSION	41
8.2.2	FSL_OCOTP_UID_LENGTH	41
<b>8.3</b>	<b>Enumeration Type Documentation</b>	<b>41</b>
8.3.1	anonymous enum	41
<b>8.4</b>	<b>Function Documentation</b>	<b>41</b>
8.4.1	OCOTP_OtpInit	41
8.4.2	OCOTP_OtpDeinit	41
8.4.3	OCOTP_OtpFuseRead	41
8.4.4	OCOTP_ReadSocOtp	42
8.4.5	OCOTP_ReadUniqueId	42
8.4.6	OCOTP_ReadSVC	42
8.4.7	OCOTP_ReadPackage	43
<b>Chapter 9 Power Driver</b>		
<b>9.1</b>	<b>Overview</b>	<b>44</b>
<b>9.2</b>	<b>Data Structure Documentation</b>	<b>47</b>
9.2.1	struct power_init_config_t	47
9.2.2	struct power_sleep_config_t	47
9.2.3	struct power_gdet_data_t	48
<b>9.3</b>	<b>Macro Definition Documentation</b>	<b>48</b>
9.3.1	FSL_POWER_DRIVER_VERSION	48
<b>9.4</b>	<b>Typedef Documentation</b>	<b>48</b>
9.4.1	capt_pulse_timer_callback_t	48
9.4.2	power_switch_callback_t	48
<b>9.5</b>	<b>Enumeration Type Documentation</b>	<b>49</b>
9.5.1	power_wakeup_edge_t	49
9.5.2	power_wakeup_pin_t	49
9.5.3	power_reset_cause_t	49
9.5.4	power_reset_source_t	49
9.5.5	_clk_pm3_buck_bits	50
<b>9.6</b>	<b>Function Documentation</b>	<b>50</b>

Section No.	Title	Page No.
9.6.1	<a href="#">POWER_EnableResetSource</a>	50
9.6.2	<a href="#">POWER_DisableResetSource</a>	50
9.6.3	<a href="#">POWER_GetResetCause</a>	50
9.6.4	<a href="#">POWER_ClearResetCause</a>	50
9.6.5	<a href="#">POWER_ConfigWakeupPin</a>	51
9.6.6	<a href="#">POWER_GetWakeupStatus</a>	51
9.6.7	<a href="#">POWER_ClearWakeupStatus</a>	51
9.6.8	<a href="#">POWER_EnableWakeup</a>	51
9.6.9	<a href="#">POWER_DisableWakeup</a>	51
9.6.10	<a href="#">AT_QUICKACCESS_SECTION_CODE</a>	52
9.6.11	<a href="#">POWER_GetWakenMode</a>	52
9.6.12	<a href="#">POWER_GetCurrentSleepConfig</a>	52
9.6.13	<a href="#">POWER_InitPowerConfig</a>	52
9.6.14	<a href="#">POWER_ConfigCauInSleep</a>	52
9.6.15	<a href="#">POWER_SetPowerSwitchCallback</a>	53
9.6.16	<a href="#">AT_QUICKACCESS_SECTION_CODE</a>	53
9.6.17	<a href="#">PMU_EnableWlanWakeups</a>	53
9.6.18	<a href="#">PMU_DisableWlanWakeups</a>	53
9.6.19	<a href="#">PMU_EnableBleWakeups</a>	54
9.6.20	<a href="#">PMU_DisableBleWakeups</a>	54
9.6.21	<a href="#">POWER_EnableCaptSlowPulseTimer</a>	54
9.6.22	<a href="#">POWER_EnableCaptFastPulseTimer</a>	54
9.6.23	<a href="#">POWER_InitVoltage</a>	55
9.6.24	<a href="#">Power_InitLoadGdetCfg</a>	55
9.6.25	<a href="#">AT_QUICKACCESS_SECTION_CODE</a>	55
9.6.26	<a href="#">POWER_TrimSvc</a>	55

## Chapter 10 Reset Driver

<b>10.1</b>	<b>Overview</b>	<b>57</b>
<b>10.2</b>	<b>Macro Definition Documentation</b>	<b>59</b>
10.2.1	<a href="#">FSL_RESET_DRIVER_VERSION</a>	59
10.2.2	<a href="#">CRC_RSTS</a>	59
<b>10.3</b>	<b>Enumeration Type Documentation</b>	<b>59</b>
10.3.1	<a href="#">RSTCTL_RSTn_t</a>	59
<b>10.4</b>	<b>Function Documentation</b>	<b>61</b>
10.4.1	<a href="#">RESET_SetPeripheralReset</a>	61
10.4.2	<a href="#">RESET_ClearPeripheralReset</a>	62
10.4.3	<a href="#">RESET_PeripheralReset</a>	62
10.4.4	<a href="#">RESET_ReleasePeripheralReset</a>	62

Section No.	Title	Page No.
<a href="#">Chapter 11</a>	<a href="#">ACOMP: Analog Comparator</a>	
<a href="#">Chapter 12</a>	<a href="#">ADC: Analog Digital Converter</a>	
<a href="#">12.1</a>	<a href="#">Overview</a>	<b>64</b>
<a href="#">12.2</a>	<a href="#">Data Structure Documentation</a>	<b>72</b>
12.2.1	<code>struct adc_config_t</code>	72
<a href="#">12.3</a>	<a href="#">Macro Definition Documentation</a>	<b>74</b>
12.3.1	<code>FSL_ADC_DRIVER_VERSION</code>	74
<a href="#">12.4</a>	<a href="#">Enumeration Type Documentation</a>	<b>74</b>
12.4.1	<code>_adc_interrupt_enable</code>	74
12.4.2	<code>_adc_status_flags</code>	74
12.4.3	<code>adc_clock_divider_t</code>	75
12.4.4	<code>adc_analog_portion_power_mode_t</code>	76
12.4.5	<code>adc_resolution_t</code>	76
12.4.6	<code>adc_warm_up_time_t</code>	77
12.4.7	<code>adc_vref_source_t</code>	77
12.4.8	<code>adc_input_mode_t</code>	78
12.4.9	<code>adc_conversion_mode_t</code>	78
12.4.10	<code>adc_scan_length_t</code>	78
12.4.11	<code>adc_average_length_t</code>	79
12.4.12	<code>adc_input_gain_t</code>	79
12.4.13	<code>adc_result_width_t</code>	79
12.4.14	<code>adc_fifo_threshold_t</code>	79
12.4.15	<code>adc_calibration_ref_t</code>	79
12.4.16	<code>adc_scan_channel_t</code>	80
12.4.17	<code>adc_channel_source_t</code>	80
12.4.18	<code>adc_temperature_sensor_mode_t</code>	81
12.4.19	<code>adc_audio_pga_voltage_gain_t</code>	81
12.4.20	<code>adc_audio_voice_level_t</code>	81
<a href="#">12.5</a>	<a href="#">Function Documentation</a>	<b>81</b>
12.5.1	<code>ADC_Init</code>	81
12.5.2	<code>ADC_GetDefaultConfig</code>	81
12.5.3	<code>ADC_Deinit</code>	82
12.5.4	<code>ADC_DoSoftwareReset</code>	82
12.5.5	<code>ADC_SelectAnalogPortionPowerMode</code>	82
12.5.6	<code>ADC_DoAutoCalibration</code>	82
12.5.7	<code>ADC_GetAutoCalibrationData</code>	83
12.5.8	<code>ADC_ResetAutoCalibrationData</code>	83
12.5.9	<code>ADC_DoUserCalibration</code>	83
12.5.10	<code>ADC_EnableTemperatureSensor</code>	84

Section No.	Title	Page No.
12.5.11	ADC_SetTemperatureSensorMode .....	84
12.5.12	ADC_EnableAudio .....	84
12.5.13	ADC_SetAudioPGAVoltageGain .....	85
12.5.14	ADC_ConfigAudioVoiceLevel .....	85
12.5.15	ADC_SetScanChannel .....	85
12.5.16	ADC_DoSoftwareTrigger .....	86
12.5.17	ADC_StopConversion .....	86
12.5.18	ADC_GetConversionResult .....	86
12.5.19	ADC_GetFifoDataCount .....	86
12.5.20	ADC_EnableInterrupts .....	87
12.5.21	ADC_DisableInterrupts .....	87
12.5.22	ADC_GetStatusFlags .....	87
12.5.23	ADC_ClearStatusFlags .....	87

## Chapter 13 CACHE: CACHE Memory Controller

<b>13.1</b>	<b>Overview .....</b>	<b>89</b>
<b>13.2</b>	<b>Function groups .....</b>	<b>89</b>
13.2.1	CACHE Operation .....	89
<b>13.3</b>	<b>Data Structure Documentation .....</b>	<b>91</b>
13.3.1	struct cache64_config_t .....	91
<b>13.4</b>	<b>Macro Definition Documentation .....</b>	<b>91</b>
13.4.1	FSL_CACHE_DRIVER_VERSION .....	91
13.4.2	CACHE64_LINESIZE_BYTE .....	91
13.4.3	CACHE64_REGION_NUM .....	91
13.4.4	CACHE64_REGION_ALIGNMENT .....	91
<b>13.5</b>	<b>Enumeration Type Documentation .....</b>	<b>91</b>
13.5.1	cache64_policy_t .....	91
<b>13.6</b>	<b>Function Documentation .....</b>	<b>91</b>
13.6.1	CACHE64GetInstance .....	91
13.6.2	CACHE64GetInstanceByAddr .....	92
13.6.3	CACHE64Init .....	92
13.6.4	CACHE64GetDefaultConfig .....	92
13.6.5	CACHE64EnableCache .....	93
13.6.6	CACHE64DisableCache .....	93
13.6.7	CACHE64InvalidateCache .....	93
13.6.8	CACHE64InvalidateCacheByRange .....	93
13.6.9	CACHE64CleanCache .....	94
13.6.10	CACHE64CleanCacheByRange .....	95
13.6.11	CACHE64CleanInvalidateCache .....	95

Section No.	Title	Page No.
13.6.12	CACHE64_CleanInvalidateCacheByRange .....	95
13.6.13	CACHE64_EnableWriteBuffer .....	96
13.6.14	ICACHE_InvalidateByRange .....	97
13.6.15	DCACHE_InvalidateByRange .....	97
13.6.16	DCACHE_CleanByRange .....	97
13.6.17	DCACHE_CleanInvalidateByRange .....	98

## Chapter 14 Common Driver

<b>14.1 Overview .....</b>	<b>99</b>
<b>14.2 Macro Definition Documentation .....</b>	<b>102</b>
14.2.1 FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ .....	102
14.2.2 MAKE_STATUS .....	102
14.2.3 MAKE_VERSION .....	102
14.2.4 FSL_COMMON_DRIVER_VERSION .....	103
14.2.5 DEBUG_CONSOLE_DEVICE_TYPE_NONE .....	103
14.2.6 DEBUG_CONSOLE_DEVICE_TYPE_UART .....	103
14.2.7 DEBUG_CONSOLE_DEVICE_TYPE_LPUART .....	103
14.2.8 DEBUG_CONSOLE_DEVICE_TYPE_LPSCI .....	103
14.2.9 DEBUG_CONSOLE_DEVICE_TYPE_USBCDC .....	103
14.2.10 DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM .....	103
14.2.11 DEBUG_CONSOLE_DEVICE_TYPE_IUART .....	103
14.2.12 DEBUG_CONSOLE_DEVICE_TYPE_VUSART .....	103
14.2.13 DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART .....	103
14.2.14 DEBUG_CONSOLE_DEVICE_TYPE_SWO .....	103
14.2.15 DEBUG_CONSOLE_DEVICE_TYPE_QSCI .....	103
14.2.16 MIN .....	103
14.2.17 MAX .....	103
14.2.18 ARRAY_SIZE .....	103
14.2.19 UINT16_MAX .....	103
14.2.20 UINT32_MAX .....	103
14.2.21 SUPPRESS_FALL_THROUGH_WARNING .....	103
<b>14.3 Typedef Documentation .....</b>	<b>104</b>
14.3.1 status_t .....	104
<b>14.4 Enumeration Type Documentation .....</b>	<b>104</b>
14.4.1 _status_groups .....	104
14.4.2 anonymous enum .....	107
<b>14.5 Function Documentation .....</b>	<b>107</b>
14.5.1 SDK_Malloc .....	107
14.5.2 SDK_Free .....	107
14.5.3 SDK_DelayAtLeastUs .....	107

Section No.	Title	Page No.
<b>Chapter 15 CRC: Cyclic Redundancy Check Driver</b>		
<b>15.1 Overview</b>	<b>109</b>	
<b>15.2 CRC Driver Initialization and Configuration</b>	<b>109</b>	
<b>15.3 CRC Write Data</b>	<b>109</b>	
<b>15.4 CRC Get Checksum</b>	<b>109</b>	
<b>15.5 Comments about API usage in RTOS</b>	<b>110</b>	
<b>15.6 Data Structure Documentation</b>	<b>111</b>	
15.6.1 struct crc_config_t	111	
<b>15.7 Macro Definition Documentation</b>	<b>112</b>	
15.7.1 FSL_CRC_DRIVER_VERSION	112	
15.7.2 CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT	112	
<b>15.8 Enumeration Type Documentation</b>	<b>112</b>	
15.8.1 crc_polynomial_t	112	
<b>15.9 Function Documentation</b>	<b>113</b>	
15.9.1 CRC_Init	113	
15.9.2 CRC_Deinit	114	
15.9.3 CRC_Reset	114	
15.9.4 CRC_WriteSeed	114	
15.9.5 CRC_GetDefaultConfig	114	
15.9.6 CRC_GetConfig	115	
15.9.7 CRC_WriteData	115	
15.9.8 CRC_Get32bitResult	115	
15.9.9 CRC_Get16bitResult	116	
<b>Chapter 16 CTIMER: Standard counter/timers</b>		
<b>16.1 Overview</b>	<b>118</b>	
<b>16.2 Function groups</b>	<b>118</b>	
16.2.1 Initialization and deinitialization	118	
16.2.2 PWM Operations	118	
16.2.3 Match Operation	118	
16.2.4 Input capture operations	118	
<b>16.3 Typical use case</b>	<b>119</b>	
16.3.1 Match example	119	
16.3.2 PWM output example	119	

Section No.	Title	Page No.
<b>16.4 Data Structure Documentation</b>		<b>122</b>
16.4.1    struct <a href="#">ctimer_match_config_t</a>		122
16.4.2    struct <a href="#">ctimer_config_t</a>		123
<b>16.5 Enumeration Type Documentation</b>		<b>123</b>
16.5.1 <a href="#">ctimer_capture_channel_t</a>		123
16.5.2 <a href="#">ctimer_capture_edge_t</a>		123
16.5.3 <a href="#">ctimer_match_t</a>		124
16.5.4 <a href="#">ctimer_external_match_t</a>		124
16.5.5 <a href="#">ctimer_match_output_control_t</a>		124
16.5.6 <a href="#">ctimer_interrupt_enable_t</a>		124
16.5.7 <a href="#">ctimer_status_flags_t</a>		125
16.5.8 <a href="#">ctimer_callback_type_t</a>		125
<b>16.6 Function Documentation</b>		<b>125</b>
16.6.1 <a href="#">CTIMER_Init</a>		125
16.6.2 <a href="#">CTIMER_Deinit</a>		125
16.6.3 <a href="#">CTIMER_GetDefaultConfig</a>		126
16.6.4 <a href="#">CTIMER_SetupPwmPeriod</a>		126
16.6.5 <a href="#">CTIMER_SetupPwm</a>		127
16.6.6 <a href="#">CTIMER_UpdatePwmPulsePeriod</a>		127
16.6.7 <a href="#">CTIMER_UpdatePwmDutycycle</a>		128
16.6.8 <a href="#">CTIMER_SetupMatch</a>		128
16.6.9 <a href="#">CTIMER_GetOutputMatchStatus</a>		129
16.6.10 <a href="#">CTIMER_SetupCapture</a>		130
16.6.11 <a href="#">CTIMER_GetTimerCountValue</a>		130
16.6.12 <a href="#">CTIMER_RegisterCallBack</a>		130
16.6.13 <a href="#">CTIMER_EnableInterrupts</a>		131
16.6.14 <a href="#">CTIMER_DisableInterrupts</a>		131
16.6.15 <a href="#">CTIMER_GetEnabledInterrupts</a>		131
16.6.16 <a href="#">CTIMER_GetStatusFlags</a>		132
16.6.17 <a href="#">CTIMER_ClearStatusFlags</a>		133
16.6.18 <a href="#">CTIMER_StartTimer</a>		133
16.6.19 <a href="#">CTIMER_StopTimer</a>		133
16.6.20 <a href="#">CTIMER_Reset</a>		133
16.6.21 <a href="#">CTIMER_SetPrescale</a>		134
16.6.22 <a href="#">CTIMER_GetCaptureValue</a>		134
16.6.23 <a href="#">CTIMER_EnableResetMatchChannel</a>		134
16.6.24 <a href="#">CTIMER_EnableStopMatchChannel</a>		135
16.6.25 <a href="#">CTIMER_EnableMatchChannelReload</a>		136
16.6.26 <a href="#">CTIMER_EnableRisingEdgeCapture</a>		136
16.6.27 <a href="#">CTIMER_EnableFallingEdgeCapture</a>		136
16.6.28 <a href="#">CTIMER_SetShadowValue</a>		137

Section No.	Title	Page No.
<b>Chapter 17 DAC: Digital Analog Converter</b>		
<b>17.1 Overview</b>		<b>138</b>
<b>17.2 Data Structure Documentation</b>		<b>143</b>
17.2.1 <code>struct dac_config_t</code>		143
17.2.2 <code>struct dac_channel_config_t</code>		143
17.2.3 <code>struct dac_triangle_config_t</code>		144
<b>17.3 Macro Definition Documentation</b>		<b>144</b>
17.3.1 <code>FSL_DAC_DRIVER_VERSION</code>		144
<b>17.4 Enumeration Type Documentation</b>		<b>145</b>
17.4.1 <code>_dac_interrupt_enable</code>		145
17.4.2 <code>_dac_status_flags</code>		145
17.4.3 <code>dac_conversion_rate_t</code>		145
17.4.4 <code>dac_reference_voltage_source_t</code>		146
17.4.5 <code>dac_output_voltage_range_t</code>		146
17.4.6 <code>dac_channel_output_t</code>		146
17.4.7 <code>dac_channel_trigger_type_t</code>		146
17.4.8 <code>dac_channel_timing_mode_t</code>		146
17.4.9 <code>dac_channel_wave_type_t</code>		147
17.4.10 <code>dac_triangle_mamp_t</code>		147
17.4.11 <code>dac_triangle_step_size_t</code>		147
17.4.12 <code>dac_triangle_waveform_type_t</code>		148
<b>17.5 Function Documentation</b>		<b>148</b>
17.5.1 <code>DAC_Init</code>		148
17.5.2 <code>DAC_GetDefaultConfig</code>		148
17.5.3 <code>DAC_Deinit</code>		148
17.5.4 <code>DAC_SetChannelConfig</code>		148
17.5.5 <code>DAC_ResetChannel</code>		149
17.5.6 <code>DAC_EnableChannelConversion</code>		149
17.5.7 <code>DAC_SetChannelOutMode</code>		149
17.5.8 <code>DAC_EnableChannelTriggerMode</code>		150
17.5.9 <code>DAC_SetChannelTrigSource</code>		150
17.5.10 <code>DAC_SetChannelTrigType</code>		151
17.5.11 <code>DAC_SetChannelTimingMode</code>		151
17.5.12 <code>DAC_EnableChannelDMA</code>		152
17.5.13 <code>DAC_SetChannelWaveType</code>		152
17.5.14 <code>DAC_SetChannelData</code>		152
17.5.15 <code>DAC_SetTriangleConfig</code>		153
17.5.16 <code>DAC_EnableInterrupts</code>		153
17.5.17 <code>DAC_DisableInterrupts</code>		153
17.5.18 <code>DAC_GetStatusFlags</code>		154

Section No.	Title	Page No.
17.5.19	DAC_ClearStatusFlags .....	154
<b>Chapter 18 DMA: Direct Memory Access Controller Driver</b>		
<b>18.1</b>	<b>Overview .....</b>	<b>155</b>
<b>18.2</b>	<b>Typical use case .....</b>	<b>155</b>
18.2.1	DMA Operation .....	155
<b>18.3</b>	<b>Data Structure Documentation .....</b>	<b>160</b>
18.3.1	struct dma_descriptor_t .....	160
18.3.2	struct dma_xfercfg_t .....	160
18.3.3	struct dma_channel_trigger_t .....	161
18.3.4	struct dma_channel_config_t .....	161
18.3.5	struct dma_transfer_config_t .....	162
18.3.6	struct dma_handle_t .....	162
<b>18.4</b>	<b>Macro Definition Documentation .....</b>	<b>163</b>
18.4.1	FSL_DMA_DRIVER_VERSION .....	163
18.4.2	FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE .....	163
18.4.3	DMA_ALLOCATE_HEAD_DESCRIPTOR .....	163
18.4.4	DMA_ALLOCATE_HEAD_DESCRIPTOR_AT_NONCACHEABLE .....	163
18.4.5	DMA_ALLOCATE_LINK_DESCRIPTOR .....	163
18.4.6	DMA_ALLOCATE_LINK_DESCRIPTOR_AT_NONCACHEABLE .....	164
18.4.7	DMA_DESCRIPTOR_END_ADDRESS .....	164
<b>18.5</b>	<b>Typedef Documentation .....</b>	<b>164</b>
18.5.1	dma_callback .....	164
<b>18.6</b>	<b>Enumeration Type Documentation .....</b>	<b>164</b>
18.6.1	anonymous enum .....	164
18.6.2	anonymous enum .....	165
18.6.3	anonymous enum .....	165
18.6.4	dma_priority_t .....	165
18.6.5	dma_irq_t .....	165
18.6.6	dma_trigger_type_t .....	165
18.6.7	anonymous enum .....	166
18.6.8	dma_trigger_burst_t .....	166
18.6.9	dma_burst_wrap_t .....	167
18.6.10	dma_transfer_type_t .....	167
<b>18.7</b>	<b>Function Documentation .....</b>	<b>167</b>
18.7.1	DMA_Init .....	167
18.7.2	DMA_Deinit .....	167
18.7.3	DMA_InstallDescriptorMemory .....	167
18.7.4	DMA_ChannelIsActive .....	168

Section No.	Title	Page No.
18.7.5	DMA_ChannelIsBusy .....	168
18.7.6	DMA_EnableChannelInterrupts .....	168
18.7.7	DMA_DisableChannelInterrupts .....	169
18.7.8	DMA_EnableChannel .....	169
18.7.9	DMA_DisableChannel .....	169
18.7.10	DMA_EnableChannelPeriphRq .....	169
18.7.11	DMA_DisableChannelPeriphRq .....	170
18.7.12	DMA_ConfigureChannelTrigger .....	170
18.7.13	DMA_SetChannelConfig .....	170
18.7.14	DMA_SetChannelXferConfig .....	171
18.7.15	DMA_GetRemainingBytes .....	171
18.7.16	DMA_SetChannelPriority .....	172
18.7.17	DMA_GetChannelPriority .....	172
18.7.18	DMA_SetChannelConfigValid .....	172
18.7.19	DMA_DoChannelSoftwareTrigger .....	173
18.7.20	DMA_LoadChannelTransferConfig .....	173
18.7.21	DMA_CreateDescriptor .....	173
18.7.22	DMA_SetupDescriptor .....	174
18.7.23	DMA_SetupChannelDescriptor .....	174
18.7.24	DMA_LoadChannelDescriptor .....	175
18.7.25	DMA_AbortTransfer .....	175
18.7.26	DMA_CreateHandle .....	175
18.7.27	DMA_SetCallback .....	176
18.7.28	DMA_PrepTransfer .....	177
18.7.29	DMA_PrepChannelTransfer .....	177
18.7.30	DMA_SubmitTransfer .....	178
18.7.31	DMA_SubmitChannelTransferParameter .....	178
18.7.32	DMA_SubmitChannelDescriptor .....	179
18.7.33	DMA_SubmitChannelTransfer .....	180
18.7.34	DMA_StartTransfer .....	181
18.7.35	DMA_IRQHandle .....	181

## Chapter 19 DMIC: Digital Microphone

19.1	Overview .....	183
19.2	Function groups .....	183
19.2.1	Initialization and deinitialization .....	183
19.2.2	Configuration .....	183
19.2.3	DMIC Data and status .....	183
19.2.4	DMIC Interrupt Functions .....	183
19.2.5	DMIC HWVAD Functions .....	184
19.2.6	DMIC HWVAD Interrupt Functions .....	184
19.3	Typical use case .....	184

Section No.	Title	Page No.
19.3.1	DMIC DMA Configuration .....	184
19.3.2	DMIC use case .....	184
<b>19.4</b>	<b>DMIC Driver .....</b>	<b>185</b>
19.4.1	Overview .....	185
19.4.2	Data Structure Documentation .....	188
19.4.3	Macro Definition Documentation .....	189
19.4.4	Typedef Documentation .....	189
19.4.5	Enumeration Type Documentation .....	189
19.4.6	Function Documentation .....	191
 <b>Chapter 20 ENET: Ethernet MAC Driver</b>		
<b>20.1</b>	<b>Overview .....</b>	<b>202</b>
<b>20.2</b>	<b>Operations of Ethernet MAC Driver .....</b>	<b>202</b>
20.2.1	MII interface Operation .....	202
20.2.2	MAC address filter .....	202
20.2.3	Other Basic control Operations .....	202
20.2.4	Transactional Operation .....	202
20.2.5	PTP IEEE 1588 Feature Operation .....	203
<b>20.3</b>	<b>Typical use case .....</b>	<b>203</b>
20.3.1	ENET Initialization, receive, and transmit operations .....	203
<b>20.4</b>	<b>Data Structure Documentation .....</b>	<b>211</b>
20.4.1	struct enet_rx_bd_struct_t .....	211
20.4.2	struct enet_tx_bd_struct_t .....	211
20.4.3	struct enet_data_error_stats_t .....	212
20.4.4	struct enet_rx_frame_error_t .....	212
20.4.5	struct enet_transfer_stats_t .....	213
20.4.6	struct enet_frame_info_t .....	214
20.4.7	struct enet_tx_dirty_ring_t .....	214
20.4.8	struct enet_buffer_config_t .....	215
20.4.9	struct enet_intcoalesce_config_t .....	216
20.4.10	struct enet_config_t .....	217
20.4.11	struct enet_tx_bd_ring_t .....	219
20.4.12	struct enet_rx_bd_ring_t .....	220
20.4.13	struct _enet_handle .....	220
<b>20.5</b>	<b>Macro Definition Documentation .....</b>	<b>222</b>
20.5.1	FSL_ENET_DRIVER_VERSION .....	222
20.5.2	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK .....	222
20.5.3	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK .....	222
20.5.4	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK .....	222

Section No.	Title	Page No.
20.5.5	<a href="#">ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask</a>	222
20.5.6	<a href="#">ENET_BUFFDESCRIPTOR_RX_LAST_MASK</a>	222
20.5.7	<a href="#">ENET_BUFFDESCRIPTOR_RX_MISS_MASK</a>	222
20.5.8	<a href="#">ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK</a>	222
20.5.9	<a href="#">ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK</a>	222
20.5.10	<a href="#">ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK</a>	222
20.5.11	<a href="#">ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK</a>	222
20.5.12	<a href="#">ENET_BUFFDESCRIPTOR_RX_CRC_MASK</a>	222
20.5.13	<a href="#">ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK</a>	222
20.5.14	<a href="#">ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK</a>	222
20.5.15	<a href="#">ENET_BUFFDESCRIPTOR_TX_READY_MASK</a>	222
20.5.16	<a href="#">ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK</a>	222
20.5.17	<a href="#">ENET_BUFFDESCRIPTOR_TX_WRAP_MASK</a>	222
20.5.18	<a href="#">ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK</a>	222
20.5.19	<a href="#">ENET_BUFFDESCRIPTOR_TX_LAST_MASK</a>	222
20.5.20	<a href="#">ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK</a>	222
20.5.21	<a href="#">ENET_BUFFDESCRIPTOR_RX_ERR_MASK</a>	222
20.5.22	<a href="#">ENET_FRAME_MAX_FRAMELEN</a>	223
20.5.23	<a href="#">ENET_FRAME_VLAN_TAGLEN</a>	223
20.5.24	<a href="#">ENET_FRAME_CRC_LEN</a>	223
20.5.25	<a href="#">ENET_FIFO_MIN_RX_FULL</a>	223
20.5.26	<a href="#">ENET_RX_MIN_BUFFERSIZE</a>	223
20.5.27	<a href="#">ENET_PHY_MAXADDRESS</a>	223
20.5.28	<a href="#">ENET_TX_INTERRUPT</a>	223
20.5.29	<a href="#">ENET_RX_INTERRUPT</a>	223
20.5.30	<a href="#">ENET_TS_INTERRUPT</a>	223
20.5.31	<a href="#">ENET_ERR_INTERRUPT</a>	223
<b>20.6</b>	<b>Typedef Documentation</b>	<b>224</b>
20.6.1	<a href="#">enet_rx_alloc_callback_t</a>	224
20.6.2	<a href="#">enet_rx_free_callback_t</a>	224
20.6.3	<a href="#">enet_callback_t</a>	224
20.6.4	<a href="#">enet_isr_t</a>	224
<b>20.7</b>	<b>Enumeration Type Documentation</b>	<b>224</b>
20.7.1	<a href="#">anonymous enum</a>	224
20.7.2	<a href="#">enet_mii_mode_t</a>	224
20.7.3	<a href="#">enet_mii_speed_t</a>	224
20.7.4	<a href="#">enet_mii_duplex_t</a>	225
20.7.5	<a href="#">enet_mii_write_t</a>	225
20.7.6	<a href="#">enet_mii_read_t</a>	225
20.7.7	<a href="#">enet_mii_extend_opcode</a>	225
20.7.8	<a href="#">enet_special_control_flag_t</a>	225
20.7.9	<a href="#">enet_interrupt_enable_t</a>	226
20.7.10	<a href="#">enet_event_t</a>	226

Section No.	Title	Page No.
20.7.11	<a href="#">enet_tx_accelerator_t</a>	227
20.7.12	<a href="#">enet_rx_accelerator_t</a>	227
<b>20.8</b>	<b>Function Documentation</b>	<b>227</b>
20.8.1	<a href="#">ENETGetInstance</a>	227
20.8.2	<a href="#">ENETGetDefaultConfig</a>	227
20.8.3	<a href="#">ENETUp</a>	228
20.8.4	<a href="#">ENETInit</a>	229
20.8.5	<a href="#">ENETDown</a>	230
20.8.6	<a href="#">ENETDeinit</a>	230
20.8.7	<a href="#">ENETReset</a>	230
20.8.8	<a href="#">ENETResetHardware</a>	230
20.8.9	<a href="#">ENETSetMII</a>	231
20.8.10	<a href="#">ENETSetSMI</a>	232
20.8.11	<a href="#">ENETGetSMI</a>	232
20.8.12	<a href="#">ENETReadSMIData</a>	232
20.8.13	<a href="#">ENETStartSMIWrite</a>	233
20.8.14	<a href="#">ENETStartSMIRead</a>	233
20.8.15	<a href="#">ENETMDIOWrite</a>	234
20.8.16	<a href="#">ENETMDIORead</a>	234
20.8.17	<a href="#">ENETStartExtC45SMIWriteReg</a>	234
20.8.18	<a href="#">ENETStartExtC45SMIWriteData</a>	236
20.8.19	<a href="#">ENETStartExtC45SMIReadData</a>	236
20.8.20	<a href="#">ENETMDIOC45Write</a>	236
20.8.21	<a href="#">ENETMDIOC45Read</a>	237
20.8.22	<a href="#">ENETSetMacAddr</a>	237
20.8.23	<a href="#">ENETGetMacAddr</a>	238
20.8.24	<a href="#">ENETAddMulticastGroup</a>	239
20.8.25	<a href="#">ENETLeaveMulticastGroup</a>	239
20.8.26	<a href="#">ENETActiveRead</a>	239
20.8.27	<a href="#">ENETEnableSleepMode</a>	240
20.8.28	<a href="#">ENETGetAccelFunction</a>	241
20.8.29	<a href="#">ENETEnableInterrupts</a>	241
20.8.30	<a href="#">ENETDisableInterrupts</a>	241
20.8.31	<a href="#">ENETGetInterruptStatus</a>	242
20.8.32	<a href="#">ENETClearInterruptStatus</a>	242
20.8.33	<a href="#">ENETSetRxISRHandler</a>	242
20.8.34	<a href="#">ENETSetTxISRHandler</a>	243
20.8.35	<a href="#">ENETSetErrISRHandler</a>	243
20.8.36	<a href="#">ENETGetRxErrBeforeReadFrame</a>	243
20.8.37	<a href="#">ENETEnableStatistics</a>	244
20.8.38	<a href="#">ENETGetStatistics</a>	244
20.8.39	<a href="#">ENETResetStatistics</a>	244
20.8.40	<a href="#">ENETGetRxFrameSize</a>	245
20.8.41	<a href="#">ENETReadFrame</a>	245

Section No.	Title	Page No.
20.8.42	<a href="#">ENET_SendFrame</a>	246
20.8.43	<a href="#">ENET_SetTxReclaim</a>	247
20.8.44	<a href="#">ENET_ReclaimTxDescriptor</a>	248
20.8.45	<a href="#">ENET_GetRxFrame</a>	249
20.8.46	<a href="#">ENET_StartTxFrame</a>	250
20.8.47	<a href="#">ENET_TransmitIRQHandler</a>	250
20.8.48	<a href="#">ENET_ReceiveIRQHandler</a>	250
20.8.49	<a href="#">ENET_ErrorIRQHandler</a>	251
20.8.50	<a href="#">ENET_Ptp1588IRQHandler</a>	251
20.8.51	<a href="#">ENET_CommonFrame0IRQHandler</a>	251
<b>20.9</b>	<b><a href="#">Variable Documentation</a></b>	<b>251</b>
20.9.1	<a href="#">s_enetClock</a>	251

## Chapter 21 FLEXCOMM: FLEXCOMM Driver

<b>21.1</b>	<b><a href="#">Overview</a></b>	<b>252</b>
<b>21.2</b>	<b><a href="#">FLEXCOMM Driver</a></b>	<b>253</b>
21.2.1	<a href="#">Overview</a>	253
21.2.2	<a href="#">Macro Definition Documentation</a>	254
21.2.3	<a href="#">Typedef Documentation</a>	254
21.2.4	<a href="#">Enumeration Type Documentation</a>	254
21.2.5	<a href="#">Function Documentation</a>	254
21.2.6	<a href="#">Variable Documentation</a>	254

## Chapter 22 FLEXSPI: Flexible Serial Peripheral Interface Driver

<b>22.1</b>	<b><a href="#">Overview</a></b>	<b>255</b>
<b>22.2</b>	<b><a href="#">Data Structure Documentation</a></b>	<b>260</b>
22.2.1	<a href="#">struct flexspi_config_t</a>	260
22.2.2	<a href="#">struct flexspi_device_config_t</a>	264
22.2.3	<a href="#">struct flexspi_transfer_t</a>	265
22.2.4	<a href="#">struct _flexspi_handle</a>	266
<b>22.3</b>	<b><a href="#">Macro Definition Documentation</a></b>	<b>267</b>
22.3.1	<a href="#">FSL_FLEXSPI_DRIVER_VERSION</a>	267
22.3.2	<a href="#">FLEXSPI_LUT_SEQ</a>	267
<b>22.4</b>	<b><a href="#">Typedef Documentation</a></b>	<b>267</b>
22.4.1	<a href="#">flexspi_transfer_callback_t</a>	267
<b>22.5</b>	<b><a href="#">Enumeration Type Documentation</a></b>	<b>267</b>
22.5.1	<a href="#">anonymous enum</a>	267
22.5.2	<a href="#">anonymous enum</a>	267

Section No.	Title	Page No.
22.5.3	<a href="#">flexspi_pad_t</a>	268
22.5.4	<a href="#">flexspi_flags_t</a>	268
22.5.5	<a href="#">flexspi_read_sample_clock_t</a>	269
22.5.6	<a href="#">flexspi_cs_interval_cycle_unit_t</a>	269
22.5.7	<a href="#">flexspi_ahb_write_wait_unit_t</a>	270
22.5.8	<a href="#">flexspi_ip_error_code_t</a>	270
22.5.9	<a href="#">flexspi_ahb_error_code_t</a>	270
22.5.10	<a href="#">flexspi_port_t</a>	271
22.5.11	<a href="#">flexspi_arb_command_source_t</a>	271
22.5.12	<a href="#">flexspi_command_type_t</a>	271
<b>22.6</b>	<b>Function Documentation</b>	<b>271</b>
22.6.1	<a href="#">FLEXSPIGetInstance</a>	271
22.6.2	<a href="#">FLEXSPICheckAndClearError</a>	271
22.6.3	<a href="#">FLEXSPIInit</a>	271
22.6.4	<a href="#">FLEXSPIGetDefaultConfig</a>	272
22.6.5	<a href="#">FLEXSPIDeinit</a>	272
22.6.6	<a href="#">FLEXSPIUpdateDlValue</a>	272
22.6.7	<a href="#">FLEXSIPISetFlashConfig</a>	272
22.6.8	<a href="#">FLEXSPISoftwareReset</a>	273
22.6.9	<a href="#">FLEXSPIEnable</a>	273
22.6.10	<a href="#">FLEXSPIEnableInterrupts</a>	273
22.6.11	<a href="#">FLEXSPIDisableInterrupts</a>	273
22.6.12	<a href="#">FLEXSPIEnableTxDMA</a>	274
22.6.13	<a href="#">FLEXSPIEnableRxDMA</a>	274
22.6.14	<a href="#">FLEXSPIGetTxFifoAddress</a>	274
22.6.15	<a href="#">FLEXSPIGetRxFifoAddress</a>	274
22.6.16	<a href="#">FLEXSPIResetFifos</a>	275
22.6.17	<a href="#">FLEXSPIGetFifoCounts</a>	275
22.6.18	<a href="#">FLEXSPIGetInterruptStatusFlags</a>	275
22.6.19	<a href="#">FLEXSPIClearInterruptStatusFlags</a>	276
22.6.20	<a href="#">FLEXSPIGetDataLearningPhase</a>	276
22.6.21	<a href="#">FLEXSPIGetArbitratorCommandSource</a>	276
22.6.22	<a href="#">FLEXSPIGetIPCommandErrorCode</a>	277
22.6.23	<a href="#">FLEXSPIGetAHBCommandErrorCode</a>	277
22.6.24	<a href="#">FLEXSPIGetBusIdleStatus</a>	277
22.6.25	<a href="#">FLEXSPIUpdateRxSampleClock</a>	278
22.6.26	<a href="#">FLEXSPIUpdateLUT</a>	278
22.6.27	<a href="#">FLEXSPIWriteData</a>	278
22.6.28	<a href="#">FLEXSPIReadData</a>	279
22.6.29	<a href="#">FLEXSPIWriteBlocking</a>	279
22.6.30	<a href="#">FLEXSPIReadBlocking</a>	280
22.6.31	<a href="#">FLEXSPITransferBlocking</a>	280
22.6.32	<a href="#">FLEXSPITransferCreateHandle</a>	281
22.6.33	<a href="#">FLEXSPITransferNonBlocking</a>	281

Section No.	Title	Page No.
22.6.34	<a href="#">FLEXSPI_TransferGetCount</a>	282
22.6.35	<a href="#">FLEXSPI_TransferAbort</a>	282
22.6.36	<a href="#">FLEXSPI_TransferHandleIRQ</a>	283
 <b>Chapter 23 FMEAS: Frequency Measure Driver</b>		
<b>23.1</b>	<a href="#">Overview</a>	<b>284</b>
<b>23.2</b>	<a href="#">Frequency Measure Driver operation</a>	<b>284</b>
<b>23.3</b>	<a href="#">Typical use case</a>	<b>284</b>
<b>23.4</b>	<a href="#">Macro Definition Documentation</a>	<b>285</b>
23.4.1	<a href="#">FSL_FMEAS_DRIVER_VERSION</a>	285
<b>23.5</b>	<a href="#">Function Documentation</a>	<b>285</b>
23.5.1	<a href="#">FMEAS_StartMeasure</a>	285
23.5.2	<a href="#">FMEAS_IsMeasureComplete</a>	285
23.5.3	<a href="#">FMEAS_GetFrequency</a>	285
 <b>Chapter 24 GDMA: General DMA(GDMA) Driver</b>		
<b>24.1</b>	<a href="#">Overview</a>	<b>286</b>
<b>24.2</b>	<a href="#">Data Structure Documentation</a>	<b>289</b>
24.2.1	<a href="#">struct gdma_channel_xfer_config_t</a>	289
24.2.2	<a href="#">struct gdma_handle_t</a>	291
<b>24.3</b>	<a href="#">Macro Definition Documentation</a>	<b>292</b>
24.3.1	<a href="#">GDMA_DESC_LLI</a>	292
<b>24.4</b>	<a href="#">Typedef Documentation</a>	<b>292</b>
24.4.1	<a href="#">gdma_callback_t</a>	292
<b>24.5</b>	<a href="#">Enumeration Type Documentation</a>	<b>292</b>
24.5.1	<a href="#">gdma_transfer_width_t</a>	292
24.5.2	<a href="#">gdma_burst_size_t</a>	292
24.5.3	<a href="#">_gdma_ahb_prot</a>	293
24.5.4	<a href="#">gdma_priority_t</a>	293
24.5.5	<a href="#">_gdma_interrupt_enable</a>	294
24.5.6	<a href="#">_gdma_interrupt_flags</a>	294
<b>24.6</b>	<a href="#">Function Documentation</a>	<b>294</b>
24.6.1	<a href="#">__ALIGNED</a>	294
24.6.2	<a href="#">GDMA_Init</a>	294
24.6.3	<a href="#">GDMA_Deinit</a>	295

Section No.	Title	Page No.
24.6.4	<a href="#">GDMA_SetChannelSourceAddress</a>	295
24.6.5	<a href="#">GDMA_SetChannelDestAddress</a>	295
24.6.6	<a href="#">GDMA_StartChannel</a>	295
24.6.7	<a href="#">GDMA_StopChannel</a>	296
24.6.8	<a href="#">GDMA_IsChannelBusy</a>	296
24.6.9	<a href="#">GDMA_EnableChannelInterrupts</a>	296
24.6.10	<a href="#">GDMA_DisableChannelInterrupts</a>	297
24.6.11	<a href="#">GDMA_GetChannelInterruptFlags</a>	297
24.6.12	<a href="#">GDMA_ClearChannelInterruptFlags</a>	297
24.6.13	<a href="#">GDMA_GetChannelFinishedDescriptorNumber</a>	298
24.6.14	<a href="#">GDMA_ClearChannelFinishedDescriptorNumber</a>	298
24.6.15	<a href="#">GDMA_SetChannelPriority</a>	298
24.6.16	<a href="#">GDMA_SetChannelTransferConfig</a>	299
24.6.17	<a href="#">GDMA_CreateHandle</a>	299
24.6.18	<a href="#">GDMA_SetCallback</a>	300
24.6.19	<a href="#">GDMA_SubmitTransfer</a>	300
24.6.20	<a href="#">GDMA_StartTransfer</a>	301
24.6.21	<a href="#">GDMA_AbortTransfer</a>	301
24.6.22	<a href="#">GDMA_IRQHandle</a>	301

## Chapter 25 GPIO: General Purpose I/O

<b>25.1 Overview</b>	<b>302</b>
<b>25.2 Function groups</b>	<b>302</b>
25.2.1 Initialization and deinitialization	302
25.2.2 Pin manipulation	302
25.2.3 Port manipulation	302
25.2.4 Port masking	302
<b>25.3 Typical use case</b>	<b>302</b>
<b>25.4 Data Structure Documentation</b>	<b>304</b>
25.4.1 <code>struct gpio_pin_config_t</code>	304
25.4.2 <code>struct gpio_interrupt_config_t</code>	304
<b>25.5 Macro Definition Documentation</b>	<b>304</b>
25.5.1 <code>FSL_GPIO_DRIVER_VERSION</code>	304
<b>25.6 Enumeration Type Documentation</b>	<b>304</b>
25.6.1 <code>gpio_pin_direction_t</code>	304
25.6.2 <code>gpio_pin_enable_mode_t</code>	304
25.6.3 <code>gpio_pin_enable_polarity_t</code>	305
25.6.4 <code>gpio_interrupt_index_t</code>	305
<b>25.7 Function Documentation</b>	<b>305</b>

Section No.	Title	Page No.
25.7.1	<code>GPIO_PortInit</code>	305
25.7.2	<code>GPIO_PinInit</code>	305
25.7.3	<code>GPIO_PinWrite</code>	307
25.7.4	<code>GPIO_PinRead</code>	307
25.7.5	<code>GPIO_PortSet</code>	308
25.7.6	<code>GPIO_PortClear</code>	309
25.7.7	<code>GPIO_PortToggle</code>	309

## Chapter 26 I2C: Inter-Integrated Circuit Driver

<b>26.1</b>	<b>Overview</b>	<b>310</b>
<b>26.2</b>	<b>Typical use case</b>	<b>310</b>
26.2.1	Master Operation in functional method	310
26.2.2	Master Operation in interrupt transactional method	311
26.2.3	Master Operation in DMA transactional method	312
26.2.4	Slave Operation in functional method	312
26.2.5	Slave Operation in interrupt transactional method	313
<b>26.3</b>	<b>I2C Driver</b>	<b>315</b>
26.3.1	Overview	315
26.3.2	Macro Definition Documentation	317
26.3.3	Enumeration Type Documentation	317
<b>26.4</b>	<b>I2C Master Driver</b>	<b>320</b>
26.4.1	Overview	320
26.4.2	Data Structure Documentation	322
26.4.3	Typedef Documentation	325
26.4.4	Enumeration Type Documentation	326
26.4.5	Function Documentation	326
<b>26.5</b>	<b>I2C Slave Driver</b>	<b>338</b>
26.5.1	Overview	338
26.5.2	Data Structure Documentation	340
26.5.3	Typedef Documentation	343
26.5.4	Enumeration Type Documentation	344
26.5.5	Function Documentation	345

## Chapter 27 I2S: I2S Driver

<b>27.1</b>	<b>Overview</b>	<b>353</b>
<b>27.2</b>	<b>I2S Driver Initialization and Configuration</b>	<b>353</b>
<b>27.3</b>	<b>I2S Transmit Data</b>	<b>353</b>

Section No.	Title	Page No.
<b>27.4 I2S Interrupt related functions</b>	354	
<b>27.5 I2S Other functions</b>	354	
<b>27.6 I2S Data formats</b>	354	
27.6.1 DMA mode	354	
27.6.2 Interrupt mode	357	
<b>27.7 I2S Driver Examples</b>	358	
27.7.1 Interrupt mode examples	358	
27.7.2 DMA mode examples	359	
<b>27.8 I2S Driver</b>	362	
27.8.1 Overview	362	
27.8.2 Data Structure Documentation	365	
27.8.3 Macro Definition Documentation	367	
27.8.4 Typedef Documentation	367	
27.8.5 Enumeration Type Documentation	368	
27.8.6 Function Documentation	369	
<b>Chapter 28 IMU: Inter CPU Messaging Unit</b>		
<b>28.1 Overview</b>	379	
<b>28.2 Data Structure Documentation</b>	380	
28.2.1 struct IMU_Type	380	
<b>28.3 Macro Definition Documentation</b>	380	
28.3.1 FSL_IMU_DRIVER_VERSION	380	
28.3.2 IMU_RX_FIFO_EMPTY	380	
28.3.3 IMU_ERR_TX_FIFO_LOCKED	381	
28.3.4 IMU_MSG_FIFO_MAX_COUNT	381	
28.3.5 IMU_MAX_MSG_FIFO_WATER_MARK	381	
<b>28.4 Enumeration Type Documentation</b>	381	
28.4.1 _imu_status_flags	381	
28.4.2 _imu_interrupts	381	
<b>28.5 Function Documentation</b>	381	
28.5.1 IMU_Init	381	
28.5.2 IMU_Deinit	381	
28.5.3 IMU_WriteMsg	382	
28.5.4 IMU_ReadMsg	383	
28.5.5 IMU_SendMsgsBlocking	383	
28.5.6 IMU_TrySendMsgs	384	
28.5.7 IMU_TryReceiveMsgs	384	

Section No.	Title	Page No.
28.5.8	IMU_ReceiveMsgsBlocking .....	385
28.5.9	IMU_SendMsgPtrBlocking .....	385
28.5.10	IMU_LockSendFifo .....	386
28.5.11	IMU_FlushSendFifo .....	386
28.5.12	IMU_SetSendFifoWaterMark .....	387
28.5.13	IMU_GetReceivedMsgCount .....	388
28.5.14	IMU_GetSendFifoEmptySpace .....	388
28.5.15	IMU_GetStatusFlags .....	388
28.5.16	IMU_ClearPendingInterrupts .....	388

## Chapter 29 INPUTMUX: Input Multiplexing Driver

<b>29.1</b>	<b>Overview .....</b>	<b>390</b>
<b>29.2</b>	<b>Input Multiplexing Driver operation .....</b>	<b>390</b>
<b>29.3</b>	<b>Typical use case .....</b>	<b>390</b>
<b>29.4</b>	<b>Enumeration Type Documentation .....</b>	<b>391</b>
29.4.1	inputmux_connection_t .....	391
29.4.2	inputmux_signal_t .....	392
<b>29.5</b>	<b>Function Documentation .....</b>	<b>392</b>
29.5.1	INPUTMUX_Init .....	392
29.5.2	INPUTMUX_AttachSignal .....	392
29.5.3	INPUTMUX_EnableSignal .....	393
29.5.4	INPUTMUX_Deinit .....	393

## Chapter 30 LCDIC: LCD Interface Controller

<b>30.1</b>	<b>Overview .....</b>	<b>395</b>
<b>30.2</b>	<b>LCDIC Driver .....</b>	<b>396</b>
30.2.1	Overview .....	396
30.2.2	Data Structure Documentation .....	401
30.2.3	Macro Definition Documentation .....	408
30.2.4	Typedef Documentation .....	408
30.2.5	Enumeration Type Documentation .....	408
30.2.6	Function Documentation .....	412
<b>30.3</b>	<b>LCDIC DMA Driver .....</b>	<b>428</b>
30.3.1	Overview .....	428
30.3.2	Data Structure Documentation .....	428
30.3.3	Typedef Documentation .....	430
30.3.4	Function Documentation .....	430

Section No.	Title	Page No.
<b>Chapter 31 MRT: Multi-Rate Timer</b>		
<b>31.1 Overview</b>		<b>433</b>
<b>31.2 Function groups</b>		<b>433</b>
31.2.1 Initialization and deinitialization		433
31.2.2 Timer period Operations		433
31.2.3 Start and Stop timer operations		433
31.2.4 Get and release channel		434
31.2.5 Status		434
31.2.6 Interrupt		434
<b>31.3 Typical use case</b>		<b>434</b>
31.3.1 MRT tick example		434
<b>31.4 Data Structure Documentation</b>		<b>436</b>
31.4.1 struct mrt_config_t		436
<b>31.5 Enumeration Type Documentation</b>		<b>436</b>
31.5.1 mrt_chnl_t		436
31.5.2 mrt_timer_mode_t		436
31.5.3 mrt_interrupt_enable_t		437
31.5.4 mrt_status_flags_t		437
<b>31.6 Function Documentation</b>		<b>437</b>
31.6.1 MRT_Init		437
31.6.2 MRT_Deinit		437
31.6.3 MRT_GetDefaultConfig		437
31.6.4 MRT_SetupChannelMode		438
31.6.5 MRT_EnableInterrupts		438
31.6.6 MRT_DisableInterrupts		438
31.6.7 MRT_GetEnabledInterrupts		438
31.6.8 MRT_GetStatusFlags		439
31.6.9 MRT_ClearStatusFlags		439
31.6.10 MRT_UpdateTimerPeriod		439
31.6.11 MRT_GetCurrentTimerCount		440
31.6.12 MRT_StartTimer		440
31.6.13 MRT_StopTimer		441
31.6.14 MRT_GetIdleChannel		441
31.6.15 MRT_ReleaseChannel		441

## Chapter 32 OSTIMER: OS Event Timer Driver

<b>32.1 Overview</b>		<b>442</b>
<b>32.2 Function groups</b>		<b>442</b>

Section No.	Title	Page No.
32.2.1	Initialization and deinitialization .....	442
32.2.2	OSTIMER status .....	442
32.2.3	OSTIMER set match value .....	442
32.2.4	OSTIMER get timer count .....	442
<b>32.3</b>	<b>Typical use case .....</b>	<b>443</b>
<b>32.4</b>	<b>Macro Definition Documentation .....</b>	<b>444</b>
32.4.1	FSL_OSTIMER_DRIVER_VERSION .....	444
<b>32.5</b>	<b>Typedef Documentation .....</b>	<b>444</b>
32.5.1	ostimer_callback_t .....	444
<b>32.6</b>	<b>Enumeration Type Documentation .....</b>	<b>444</b>
32.6.1	_ostimer_flags .....	444
<b>32.7</b>	<b>Function Documentation .....</b>	<b>444</b>
32.7.1	OSTIMER_Init .....	444
32.7.2	OSTIMER_Deinit .....	444
32.7.3	OSTIMER_GrayToDecimal .....	444
32.7.4	OSTIMER_DecimalToGray .....	445
32.7.5	OSTIMER_GetStatusFlags .....	445
32.7.6	OSTIMER_ClearStatusFlags .....	445
32.7.7	OSTIMER_SetMatchRawValue .....	446
32.7.8	OSTIMER_SetMatchValue .....	446
32.7.9	OSTIMER_SetMatchRegister .....	447
32.7.10	OSTIMER_EnableMatchInterrupt .....	447
32.7.11	OSTIMER_DisableMatchInterrupt .....	447
32.7.12	OSTIMER_GetCurrentTimerRawValue .....	448
32.7.13	OSTIMER_GetCurrentTimerValue .....	448
32.7.14	OSTIMER_GetCaptureRawValue .....	448
32.7.15	OSTIMER_GetCaptureValue .....	449
32.7.16	OSTIMER_HandleIRQ .....	449

## Chapter 33 PINT: Pin Interrupt and Pattern Match Driver

<b>33.1</b>	<b>Overview .....</b>	<b>450</b>
<b>33.2</b>	<b>Pin Interrupt and Pattern match Driver operation .....</b>	<b>450</b>
33.2.1	Pin Interrupt use case .....	450
33.2.2	Pattern match use case .....	450
<b>33.3</b>	<b>Typedef Documentation .....</b>	<b>453</b>
33.3.1	pint_cb_t .....	453
<b>33.4</b>	<b>Enumeration Type Documentation .....</b>	<b>453</b>

Section No.	Title	Page No.
33.4.1	pint_pin_enable_t .....	453
33.4.2	pint_pin_int_t .....	453
33.4.3	pint_pmatch_input_src_t .....	454
33.4.4	pint_pmatch_bslice_t .....	454
33.4.5	pint_pmatch_bslice_cfg_t .....	454
<b>33.5</b>	<b>Function Documentation .....</b>	<b>455</b>
33.5.1	PINT_Init .....	455
33.5.2	PINT_PinInterruptConfig .....	456
33.5.3	PINT_PinInterruptGetConfig .....	456
33.5.4	PINT_PinInterruptClrStatus .....	457
33.5.5	PINT_PinInterruptGetStatus .....	457
33.5.6	PINT_PinInterruptClrStatusAll .....	457
33.5.7	PINT_PinInterruptGetStatusAll .....	458
33.5.8	PINT_PinInterruptClrFallFlag .....	458
33.5.9	PINT_PinInterruptGetFallFlag .....	458
33.5.10	PINT_PinInterruptClrFallFlagAll .....	459
33.5.11	PINT_PinInterruptGetFallFlagAll .....	459
33.5.12	PINT_PinInterruptClrRiseFlag .....	460
33.5.13	PINT_PinInterruptGetRiseFlag .....	461
33.5.14	PINT_PinInterruptClrRiseFlagAll .....	461
33.5.15	PINT_PinInterruptGetRiseFlagAll .....	461
33.5.16	PINT_PatternMatchConfig .....	462
33.5.17	PINT_PatternMatchGetConfig .....	462
33.5.18	PINT_PatternMatchGetStatus .....	463
33.5.19	PINT_PatternMatchGetStatusAll .....	463
33.5.20	PINT_PatternMatchResetDetectLogic .....	463
33.5.21	PINT_PatternMatchEnable .....	464
33.5.22	PINT_PatternMatchDisable .....	464
33.5.23	PINT_PatternMatchEnableRXEV .....	464
33.5.24	PINT_PatternMatchDisableRXEV .....	465
33.5.25	PINT_EnableCallback .....	465
33.5.26	PINT_DisableCallback .....	465
33.5.27	PINT_Deinit .....	466
33.5.28	PINT_EnableCallbackByIndex .....	466
33.5.29	PINT_DisableCallbackByIndex .....	466

## Chapter 34 POWERQUAD: PowerQuad hardware accelerator

<b>34.1</b>	<b>Overview .....</b>	<b>468</b>
<b>34.2</b>	<b>Function groups .....</b>	<b>469</b>
34.2.1	POWERQUAD functional Operation .....	469
<b>34.3</b>	<b>Data Structure Documentation .....</b>	<b>475</b>

Section No.	Title	Page No.
34.3.1	struct pq_prescale_t .....	475
34.3.2	struct pq_config_t .....	476
34.3.3	struct pq_biquad_param_t .....	477
34.3.4	struct pq_biquad_state_t .....	478
34.3.5	struct pq_biquad_cascade_df2_instance .....	478
34.3.6	union pq_float_t .....	478
<b>34.4</b>	<b>Macro Definition Documentation .....</b>	<b>479</b>
34.4.1	FSL_POWERQUAD_DRIVER_VERSION .....	479
34.4.2	PQ_Initiate_Vector_Func .....	479
34.4.3	PQ_End_Vector_Func .....	479
34.4.4	PQ_StartVector .....	479
34.4.5	PQ_StartVectorFixed16 .....	480
34.4.6	PQ_StartVectorQ15 .....	480
34.4.7	PQ_EndVector .....	482
34.4.8	PQ_Vector8F32 .....	482
34.4.9	PQ_Vector8Fixed32 .....	482
34.4.10	PQ_Vector8Fixed16 .....	483
34.4.11	PQ_Vector8Q15 .....	483
34.4.12	PQ_DF2_Vector8_FP .....	483
34.4.13	PQ_DF2_Vector8_FX .....	484
34.4.14	PQ_Vector8BiquadDf2F32 .....	484
34.4.15	PQ_Vector8BiquadDf2Fixed32 .....	485
34.4.16	PQ_Vector8BiquadDf2Fixed16 .....	485
34.4.17	PQ_DF2_Cascade_Vector8_FP .....	486
34.4.18	PQ_DF2_Cascade_Vector8_FX .....	486
34.4.19	PQ_Vector8BiquadDf2CascadeF32 .....	487
34.4.20	PQ_Vector8BiquadDf2CascadeFixed32 .....	488
34.4.21	PQ_Vector8BiquadDf2CascadeFixed16 .....	488
34.4.22	POWERQUAD_MAKE_MATRIX_LEN .....	489
34.4.23	PQ_Q31_2_FLOAT .....	489
34.4.24	PQ_Q15_2_FLOAT .....	489
<b>34.5</b>	<b>Enumeration Type Documentation .....</b>	<b>489</b>
34.5.1	pq_computationengine_t .....	489
34.5.2	pq_format_t .....	489
34.5.3	pq_cordic_iter_t .....	490
<b>34.6</b>	<b>Function Documentation .....</b>	<b>490</b>
34.6.1	PQ_GetDefaultConfig .....	490
34.6.2	PQ_SetConfig .....	490
34.6.3	PQ_SetCoprocessorScaler .....	490
34.6.4	PQ_Init .....	491
34.6.5	PQ_Deinit .....	491
34.6.6	PQ_SetFormat .....	491

Section No.	Title	Page No.
34.6.7	PQ_WaitDone .....	491
34.6.8	PQ_LnF32 .....	492
34.6.9	PQ_InvF32 .....	492
34.6.10	PQ_SqrtF32 .....	492
34.6.11	PQ_InvSqrtF32 .....	492
34.6.12	PQ_EtoxF32 .....	493
34.6.13	PQ_EtonxF32 .....	493
34.6.14	PQ_SinF32 .....	493
34.6.15	PQ_CosF32 .....	493
34.6.16	PQ_BiquadF32 .....	494
34.6.17	PQ_DivF32 .....	494
34.6.18	PQ_Biquad1F32 .....	494
34.6.19	PQ_LnFixed .....	494
34.6.20	PQ_InvFixed .....	495
34.6.21	PQ_SqrtFixed .....	495
34.6.22	PQ_InvSqrtFixed .....	495
34.6.23	PQ_EtoxFixed .....	495
34.6.24	PQ_EtonxFixed .....	496
34.6.25	PQ_SinQ31 .....	496
34.6.26	PQ_SinQ15 .....	496
34.6.27	PQ_CosQ31 .....	496
34.6.28	PQ_CosQ15 .....	497
34.6.29	PQ_BiquadFixed .....	497
34.6.30	PQ_VectorLnF32 .....	497
34.6.31	PQ_VectorInvF32 .....	497
34.6.32	PQ_VectorSqrtF32 .....	498
34.6.33	PQ_VectorInvSqrtF32 .....	498
34.6.34	PQ_VectorEtoxF32 .....	498
34.6.35	PQ_VectorEtonxF32 .....	498
34.6.36	PQ_VectorSinF32 .....	499
34.6.37	PQ_VectorCosF32 .....	499
34.6.38	PQ_VectorLnFixed32 .....	499
34.6.39	PQ_VectorInvFixed32 .....	499
34.6.40	PQ_VectorSqrtFixed32 .....	500
34.6.41	PQ_VectorInvSqrtFixed32 .....	500
34.6.42	PQ_VectorEtoxFixed32 .....	500
34.6.43	PQ_VectorEtonxFixed32 .....	501
34.6.44	PQ_VectorSinQ15 .....	502
34.6.45	PQ_VectorCosQ15 .....	502
34.6.46	PQ_VectorSinQ31 .....	502
34.6.47	PQ_VectorCosQ31 .....	502
34.6.48	PQ_VectorLnFixed16 .....	503
34.6.49	PQ_VectorInvFixed16 .....	503
34.6.50	PQ_VectorSqrtFixed16 .....	503
34.6.51	PQ_VectorInvSqrtFixed16 .....	503

Section No.	Title	Page No.
34.6.52	PQ_VectorEtoxFixed16 .....	504
34.6.53	PQ_VectorEtonxFixed16 .....	504
34.6.54	PQ_VectorBiquadDf2F32 .....	504
34.6.55	PQ_VectorBiquadDf2Fixed32 .....	504
34.6.56	PQ_VectorBiquadDf2Fixed16 .....	505
34.6.57	PQ_VectorBiquadCascadeDf2F32 .....	505
34.6.58	PQ_VectorBiquadCascadeDf2Fixed32 .....	505
34.6.59	PQ_VectorBiquadCascadeDf2Fixed16 .....	506
34.6.60	PQ_ArcTanFixed .....	507
34.6.61	PQ_ArctanhFixed .....	507
34.6.62	PQ_Arctan2Fixed .....	508
34.6.63	PQ_Biquad1Fixed .....	508
34.6.64	PQ_TransformCFFT .....	509
34.6.65	PQ_TransformRFFT .....	509
34.6.66	PQ_TransformIFFT .....	509
34.6.67	PQ_TransformCDCT .....	510
34.6.68	PQ_TransformRDCT .....	510
34.6.69	PQ_TransformIDCT .....	510
34.6.70	PQ_BiquadBackUpInternalState .....	511
34.6.71	PQ_BiquadRestoreInternalState .....	511
34.6.72	PQ_BiquadCascadeDf2Init .....	511
34.6.73	PQ_BiquadCascadeDf2F32 .....	512
34.6.74	PQ_BiquadCascadeDf2Fixed32 .....	512
34.6.75	PQ_BiquadCascadeDf2Fixed16 .....	512
34.6.76	PQ_FIR .....	513
34.6.77	PQ_FIRIncrement .....	513
34.6.78	PQ_MatrixAddition .....	513
34.6.79	PQ_MatrixSubtraction .....	514
34.6.80	PQ_MatrixMultiplication .....	514
34.6.81	PQ_MatrixProduct .....	515
34.6.82	PQ_VectorDotProduct .....	516
34.6.83	PQ_MatrixInversion .....	516
34.6.84	PQ_MatrixTranspose .....	517
34.6.85	PQ_MatrixScale .....	518

## Chapter 35 RTC: Real Time Clock

35.1	Overview .....	519
35.2	Function groups .....	519
35.2.1	Initialization and deinitialization .....	519
35.2.2	Set & Get Datetime .....	519
35.2.3	Set & Get Alarm .....	519
35.2.4	Start & Stop timer .....	519
35.2.5	Status .....	520

Section No.	Title	Page No.
35.2.6	Interrupt .....	520
35.2.7	High resolution timer .....	520
<b>35.3</b>	<b>Typical use case .....</b>	<b>520</b>
35.3.1	RTC tick example .....	520
<b>35.4</b>	<b>Data Structure Documentation .....</b>	<b>522</b>
35.4.1	struct rtc_datetime_t .....	522
<b>35.5</b>	<b>Enumeration Type Documentation .....</b>	<b>523</b>
35.5.1	rtc_interrupt_enable_t .....	523
35.5.2	rtc_status_flags_t .....	523
<b>35.6</b>	<b>Function Documentation .....</b>	<b>523</b>
35.6.1	RTC_Init .....	523
35.6.2	RTC_Deinit .....	523
35.6.3	RTC_SetDatetime .....	524
35.6.4	RTC_GetDatetime .....	524
35.6.5	RTC_SetAlarm .....	524
35.6.6	RTC_GetAlarm .....	525
35.6.7	RTC_EnableWakeupTimer .....	525
35.6.8	RTC_GetEnabledWakeupTimer .....	525
35.6.9	RTC_EnableSubsecCounter .....	526
35.6.10	RTC_GetSubsecValue .....	526
35.6.11	RTC_SetSecondsTimerMatch .....	526
35.6.12	RTC_GetSecondsTimerMatch .....	526
35.6.13	RTC_SetSecondsTimerCount .....	527
35.6.14	RTC_GetSecondsTimerCount .....	527
35.6.15	RTC_SetWakeupCount .....	527
35.6.16	RTC_GetWakeupCount .....	528
35.6.17	RTC_EnableWakeUpTimerInterruptFromDPD .....	529
35.6.18	RTC_EnableAlarmTimerInterruptFromDPD .....	529
35.6.19	RTC_EnableInterrupts .....	529
35.6.20	RTC_DisableInterrupts .....	530
35.6.21	RTC_GetEnabledInterrupts .....	530
35.6.22	RTC_GetStatusFlags .....	530
35.6.23	RTC_ClearStatusFlags .....	531
35.6.24	RTC_EnableTimer .....	531
35.6.25	RTC_StartTimer .....	531
35.6.26	RTC_StopTimer .....	533
35.6.27	RTC_Reset .....	533

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

<b>36.1</b>	<b>Overview .....</b>	<b>534</b>
-------------	-----------------------	------------

Section No.	Title	Page No.
<b>36.2 Function groups</b>		<b>534</b>
36.2.1 Initialization and deinitialization		534
36.2.2 PWM Operations		534
36.2.3 Status		534
36.2.4 Interrupt		534
<b>36.3 SCTimer State machine and operations</b>		<b>535</b>
36.3.1 SCTimer event operations		535
36.3.2 SCTimer state operations		535
36.3.3 SCTimer action operations		535
<b>36.4 16-bit counter mode</b>		<b>535</b>
<b>36.5 Typical use case</b>		<b>536</b>
36.5.1 PWM output		536
<b>36.6 Data Structure Documentation</b>		<b>541</b>
36.6.1 struct sctimer_pwm_signal_param_t		541
36.6.2 struct sctimer_config_t		541
<b>36.7 Typedef Documentation</b>		<b>542</b>
36.7.1 sctimer_event_callback_t		542
<b>36.8 Enumeration Type Documentation</b>		<b>543</b>
36.8.1 sctimer_pwm_mode_t		543
36.8.2 sctimer_counter_t		543
36.8.3 sctimer_input_t		543
36.8.4 sctimer_out_t		543
36.8.5 sctimer_pwm_level_select_t		544
36.8.6 sctimer_clock_mode_t		544
36.8.7 sctimer_clock_select_t		544
36.8.8 sctimer_conflict_resolution_t		545
36.8.9 sctimer_event_active_direction_t		545
36.8.10 sctimer_interrupt_enable_t		545
36.8.11 sctimer_status_flags_t		545
<b>36.9 Function Documentation</b>		<b>546</b>
36.9.1 SCTIMER_Init		546
36.9.2 SCTIMER_Deinit		546
36.9.3 SCTIMER_GetDefaultConfig		546
36.9.4 SCTIMER_SetupPwm		547
36.9.5 SCTIMER_UpdatePwmDutycycle		548
36.9.6 SCTIMER_EnableInterrupts		548
36.9.7 SCTIMER_DisableInterrupts		548
36.9.8 SCTIMER_GetEnabledInterrupts		549
36.9.9 SCTIMER_GetStatusFlags		550

Section No.	Title	Page No.
36.9.10	SCTIMER_ClearStatusFlags .....	550
36.9.11	SCTIMER_StartTimer .....	550
36.9.12	SCTIMER_StopTimer .....	551
36.9.13	SCTIMER_CreateAndScheduleEvent .....	551
36.9.14	SCTIMER_ScheduleEvent .....	552
36.9.15	SCTIMER_IncreaseState .....	552
36.9.16	SCTIMER_GetCurrentState .....	552
36.9.17	SCTIMER_SetCounterState .....	553
36.9.18	SCTIMER_GetCounterState .....	553
36.9.19	SCTIMER_SetupCaptureAction .....	553
36.9.20	SCTIMER_SetCallback .....	554
36.9.21	SCTIMER_SetupStateLdMethodAction .....	554
36.9.22	SCTIMER_SetupNextStateActionwithLdMethod .....	554
36.9.23	SCTIMER_SetupNextStateAction .....	555
36.9.24	SCTIMER_SetupEventActiveDirection .....	555
36.9.25	SCTIMER_SetupOutputSetAction .....	556
36.9.26	SCTIMER_SetupOutputClearAction .....	557
36.9.27	SCTIMER_SetupOutputToggleAction .....	557
36.9.28	SCTIMER_SetupCounterLimitAction .....	557
36.9.29	SCTIMER_SetupCounterStopAction .....	558
36.9.30	SCTIMER_SetupCounterStartAction .....	558
36.9.31	SCTIMER_SetupCounterHaltAction .....	558
36.9.32	SCTIMER_SetupDmaTriggerAction .....	559
36.9.33	SCTIMER_SetCOUNTValue .....	559
36.9.34	SCTIMER_GetCOUNTValue .....	559
36.9.35	SCTIMER_SetEventInState .....	560
36.9.36	SCTIMER_ClearEventInState .....	560
36.9.37	SCTIMER_GetEventInState .....	560
36.9.38	SCTIMER_GetCaptureValue .....	561
36.9.39	SCTIMER_EventHandleIRQ .....	561

## Chapter 37 SPI: Serial Peripheral Interface Driver

37.1	Overview .....	563
37.2	Typical use case .....	563
37.2.1	SPI master transfer using an interrupt method .....	563
37.2.2	SPI Send/receive using a DMA method .....	564
37.3	SPI Driver .....	566
37.3.1	Overview .....	566
37.3.2	Data Structure Documentation .....	569
37.3.3	Macro Definition Documentation .....	573
37.3.4	Typedef Documentation .....	573
37.3.5	Enumeration Type Documentation .....	573

Section No.	Title	Page No.
37.3.6	Variable Documentation .....	576

## **Chapter 38 USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver**

<b>38.1</b>	<b>Overview .....</b>	<b>577</b>
<b>38.2</b>	<b>Typical use case .....</b>	<b>578</b>
38.2.1	USART Send/receive using a polling method .....	578
38.2.2	USART Send/receive using an interrupt method .....	578
38.2.3	USART Receive using the ringbuffer feature .....	579
38.2.4	USART Send/Receive using the DMA method .....	580
<b>38.3</b>	<b>USART Driver .....</b>	<b>582</b>
38.3.1	Overview .....	582
38.3.2	Data Structure Documentation .....	587
38.3.3	Macro Definition Documentation .....	589
38.3.4	Typedef Documentation .....	590
38.3.5	Enumeration Type Documentation .....	590
38.3.6	Function Documentation .....	593

## **Chapter 39 UTICK: MictoTick Timer Driver**

<b>39.1</b>	<b>Overview .....</b>	<b>610</b>
<b>39.2</b>	<b>Typical use case .....</b>	<b>610</b>
<b>39.3</b>	<b>Macro Definition Documentation .....</b>	<b>611</b>
39.3.1	FSL_UTICK_DRIVER_VERSION .....	611
<b>39.4</b>	<b>Typedef Documentation .....</b>	<b>611</b>
39.4.1	utick_callback_t .....	611
<b>39.5</b>	<b>Enumeration Type Documentation .....</b>	<b>611</b>
39.5.1	utick_mode_t .....	611
<b>39.6</b>	<b>Function Documentation .....</b>	<b>611</b>
39.6.1	UTICK_Init .....	611
39.6.2	UTICK_Deinit .....	611
39.6.3	UTICK_GetStatusFlags .....	611
39.6.4	UTICK_ClearStatusFlags .....	612
39.6.5	UTICK_SetTick .....	612
39.6.6	UTICK_HandleIRQ .....	612

## **Chapter 40 WWDT: Windowed Watchdog Timer Driver**

<b>40.1</b>	<b>Overview .....</b>	<b>614</b>
-------------	-----------------------	------------

Section No.	Title	Page No.
<b>40.2 Function groups</b>		<b>614</b>
40.2.1 Initialization and deinitialization		614
40.2.2 Status		614
40.2.3 Interrupt		614
40.2.4 Watch dog Refresh		614
<b>40.3 Typical use case</b>		<b>614</b>
<b>40.4 Data Structure Documentation</b>		<b>616</b>
40.4.1 <code>struct wwdt_config_t</code>		616
<b>40.5 Macro Definition Documentation</b>		<b>616</b>
40.5.1 <code>FSL_WWDT_DRIVER_VERSION</code>		616
<b>40.6 Enumeration Type Documentation</b>		<b>616</b>
40.6.1 <code>_wwdt_status_flags_t</code>		616
<b>40.7 Function Documentation</b>		<b>617</b>
40.7.1 <code>WWDT_GetDefaultConfig</code>		617
40.7.2 <code>WWDT_Init</code>		617
40.7.3 <code>WWDT_Deinit</code>		617
40.7.4 <code>WWDT_Enable</code>		618
40.7.5 <code>WWDT_Disable</code>		618
40.7.6 <code>WWDT_GetStatusFlags</code>		618
40.7.7 <code>WWDT_ClearStatusFlags</code>		619
40.7.8 <code>WWDT_SetWarningValue</code>		619
40.7.9 <code>WWDT_SetTimeoutValue</code>		619
40.7.10 <code>WWDT_SetWindowValue</code>		621
40.7.11 <code>WWDT_Refresh</code>		621

## Chapter 41 Debug Console

<b>41.1 Overview</b>	<b>622</b>
<b>41.2 Function groups</b>	<b>622</b>
41.2.1 Initialization	622
41.2.2 Advanced Feature	623
41.2.3 <code>SDK_DEBUGCONSOLE</code> and <code>SDK_DEBUGCONSOLE_UART</code>	627
<b>41.3 Typical use case</b>	<b>628</b>
<b>41.4 Macro Definition Documentation</b>	<b>630</b>
41.4.1 <code>DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN</code>	630
41.4.2 <code>DEBUGCONSOLE_REDIRECT_TO_SDK</code>	630
41.4.3 <code>DEBUGCONSOLE_DISABLE</code>	630
41.4.4 <code>SDK_DEBUGCONSOLE</code>	630

Section No.	Title	Page No.
41.4.5	PRINTF .....	630
<b>41.5</b>	<b>Function Documentation .....</b>	<b>630</b>
41.5.1	DbgConsole_Init .....	630
41.5.2	DbgConsole_Deinit .....	631
41.5.3	DbgConsole_EnterLowpower .....	631
41.5.4	DbgConsole_ExitLowpower .....	632
41.5.5	DbgConsole_Printf .....	632
41.5.6	DbgConsole_Vprintf .....	632
41.5.7	DbgConsole_Putchar .....	632
41.5.8	DbgConsole_Scanf .....	633
41.5.9	DbgConsole_Getchar .....	633
41.5.10	DbgConsole_BlockingPrintf .....	634
41.5.11	DbgConsole_BlockingVprintf .....	634
41.5.12	DbgConsole_Flush .....	634
41.5.13	DbgConsole_TryGetchar .....	635
<b>41.6</b>	<b>debug console configuration .....</b>	<b>637</b>
41.6.1	Overview .....	637
41.6.2	Macro Definition Documentation .....	638

## Chapter 42 Debug Console Lite

<b>42.1</b>	<b>Overview .....</b>	<b>640</b>
<b>42.2</b>	<b>Function groups .....</b>	<b>640</b>
42.2.1	Initialization .....	640
42.2.2	Advanced Feature .....	641
42.2.3	SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART .....	645
<b>42.3</b>	<b>Typical use case .....</b>	<b>646</b>
<b>42.4</b>	<b>Macro Definition Documentation .....</b>	<b>647</b>
42.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN .....	647
42.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK .....	648
42.4.3	DEBUGCONSOLE_DISABLE .....	648
42.4.4	SDK_DEBUGCONSOLE .....	648
42.4.5	PRINTF_FLOAT_ENABLE .....	648
42.4.6	SCANF_FLOAT_ENABLE .....	648
42.4.7	PRINTF_ADVANCED_ENABLE .....	648
42.4.8	SCANF_ADVANCED_ENABLE .....	648
42.4.9	PRINTF .....	648
<b>42.5</b>	<b>Function Documentation .....</b>	<b>648</b>
42.5.1	DbgConsole_Init .....	648
42.5.2	DbgConsole_Deinit .....	649

Section No.	Title	Page No.
42.5.3	DbgConsole_EnterLowpower .....	649
42.5.4	DbgConsole_ExitLowpower .....	649
42.5.5	DbgConsole_Printf .....	650
42.5.6	DbgConsole_Vprintf .....	651
42.5.7	DbgConsole_Putchar .....	651
42.5.8	DbgConsole_Scanf .....	651
42.5.9	DbgConsole_Getchar .....	652

## Chapter 43 Shell

<b>43.1</b>	<b>Overview .....</b>	<b>653</b>
<b>43.2</b>	<b>Function groups .....</b>	<b>653</b>
43.2.1	Initialization .....	653
43.2.2	Advanced Feature .....	653
43.2.3	Shell Operation .....	653
<b>43.3</b>	<b>Data Structure Documentation .....</b>	<b>655</b>
43.3.1	struct shell_command_t .....	655
<b>43.4</b>	<b>Macro Definition Documentation .....</b>	<b>656</b>
43.4.1	SHELL_NON_BLOCKING_MODE .....	656
43.4.2	SHELL_AUTO_COMPLETE .....	656
43.4.3	SHELL_BUFFER_SIZE .....	656
43.4.4	SHELL_MAX_ARGS .....	656
43.4.5	SHELL_HISTORY_COUNT .....	656
43.4.6	SHELL_HANDLE_SIZE .....	656
43.4.7	SHELL_USE_COMMON_TASK .....	657
43.4.8	SHELL_TASK_PRIORITY .....	657
43.4.9	SHELL_TASK_STACK_SIZE .....	657
43.4.10	SHELL_PRINT_COPYRIGHT .....	657
43.4.11	SHELL_HANDLE_DEFINE .....	657
43.4.12	SHELL_COMMAND_DEFINE .....	657
43.4.13	SHELL_COMMAND .....	658
<b>43.5</b>	<b>Typedef Documentation .....</b>	<b>658</b>
43.5.1	cmd_function_t .....	658
<b>43.6</b>	<b>Enumeration Type Documentation .....</b>	<b>658</b>
43.6.1	shell_status_t .....	658
<b>43.7</b>	<b>Function Documentation .....</b>	<b>658</b>
43.7.1	SHELL_Init .....	658
43.7.2	SHELL_RegisterCommand .....	659
43.7.3	SHELL_UnregisterCommand .....	660
43.7.4	SHELL_Write .....	660

Section No.	Title	Page No.
43.7.5	SHELL_Printf .....	661
43.7.6	SHELL_WriteSynchronization .....	661
43.7.7	SHELL_PrintfSynchronization .....	661
43.7.8	SHELL_ChangePrompt .....	662
43.7.9	SHELL_PrintPrompt .....	662
43.7.10	SHELL_Task .....	662
43.7.11	SHELL_checkRunningInIsr .....	663

## Chapter 44 CODEC Driver

<b>44.1</b>	<b>Overview .....</b>	<b>664</b>
<b>44.2</b>	<b>CODEC Common Driver .....</b>	<b>665</b>
44.2.1	Overview .....	665
44.2.2	Data Structure Documentation .....	670
44.2.3	Macro Definition Documentation .....	671
44.2.4	Enumeration Type Documentation .....	671
44.2.5	Function Documentation .....	676
<b>44.3</b>	<b>CODEC I2C Driver .....</b>	<b>680</b>
44.3.1	Overview .....	680
44.3.2	Data Structure Documentation .....	681
44.3.3	Enumeration Type Documentation .....	681
44.3.4	Function Documentation .....	681

## Chapter 45 Serial Manager

<b>45.1</b>	<b>Overview .....</b>	<b>684</b>
<b>45.2</b>	<b>Data Structure Documentation .....</b>	<b>688</b>
45.2.1	struct serial_manager_config_t .....	688
45.2.2	struct serial_manager_callback_message_t .....	688
<b>45.3</b>	<b>Macro Definition Documentation .....</b>	<b>688</b>
45.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE .....	688
45.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE .....	688
45.3.3	SERIAL_MANAGER_USE_COMMON_TASK .....	688
45.3.4	SERIAL_MANAGER_HANDLE_SIZE .....	689
45.3.5	SERIAL_MANAGER_HANDLE_DEFINE .....	689
45.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE .....	689
45.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE .....	690
45.3.8	SERIAL_MANAGER_TASK_PRIORITY .....	690
45.3.9	SERIAL_MANAGER_TASK_STACK_SIZE .....	690
<b>45.4</b>	<b>Enumeration Type Documentation .....</b>	<b>690</b>

Section No.	Title	Page No.
45.4.1	serial_port_type_t .....	690
45.4.2	serial_manager_type_t .....	691
45.4.3	serial_manager_status_t .....	691
<b>45.5</b>	<b>Function Documentation .....</b>	<b>691</b>
45.5.1	SerialManager_Init .....	691
45.5.2	SerialManager_Deinit .....	693
45.5.3	SerialManager_OpenWriteHandle .....	694
45.5.4	SerialManager_CloseWriteHandle .....	695
45.5.5	SerialManager_OpenReadHandle .....	695
45.5.6	SerialManager_CloseReadHandle .....	696
45.5.7	SerialManager_WriteBlocking .....	696
45.5.8	SerialManager_ReadBlocking .....	697
45.5.9	SerialManager_WriteNonBlocking .....	698
45.5.10	SerialManager_ReadNonBlocking .....	699
45.5.11	SerialManager_TryRead .....	700
45.5.12	SerialManager_CancelWriting .....	700
45.5.13	SerialManager_CancelReading .....	701
45.5.14	SerialManager_InstallTxCallback .....	701
45.5.15	SerialManager_InstallRxCallback .....	702
45.5.16	SerialManager_needPollingIsr .....	702
45.5.17	SerialManager_EnterLowpower .....	703
45.5.18	SerialManager_ExitLowpower .....	704
45.5.19	SerialManager_SetLowpowerCriticalCb .....	704

## Chapter 46 Spi\_cmsis\_driver

<b>46.1</b>	<b>Function groups .....</b>	<b>705</b>
46.1.1	SPI CMSIS GetVersion Operation .....	705
46.1.2	SPI CMSIS GetCapabilities Operation .....	705
46.1.3	SPI CMSIS Initialize and Uninitialize Operation .....	705
46.1.4	SPI CMSIS Transfer Operation .....	705
46.1.5	SPI CMSIS Status Operation .....	705
46.1.6	SPI CMSIS Control Operation .....	706
<b>46.2</b>	<b>Typical use case .....</b>	<b>706</b>
46.2.1	Master Operation .....	706
46.2.2	Slave Operation .....	706

## Chapter 47 I2c\_cmsis\_driver

<b>47.1</b>	<b>I2C CMSIS Driver .....</b>	<b>707</b>
47.1.1	Master Operation in interrupt transactional method .....	707
47.1.2	Master Operation in DMA transactional method .....	707
47.1.3	Slave Operation in interrupt transactional method .....	708

Section No.	Title	Page No.
<b>Chapter 48 Usart_cmsis_driver</b>		
<b>48.1 USART Send Methods</b>	<b>709</b>	
48.1.1 USART Send/receive using an interrupt method	709	
48.1.2 USART Send/Receive using the DMA method	709	
<b>Chapter 49 CDOG</b>		
<b>49.1 Overview</b>	<b>711</b>	
<b>49.2 Macro Definition Documentation</b>	<b>712</b>	
49.2.1 FSL_CDOG_DRIVER_VERSION	712	
<b>49.3 Function Documentation</b>	<b>712</b>	
49.3.1 CDOG_Init	712	
49.3.2 CDOG_Deinit	712	
49.3.3 CDOG_GetDefaultConfig	713	
49.3.4 CDOG_Stop	713	
49.3.5 CDOG_Start	713	
49.3.6 CDOG_Check	713	
49.3.7 CDOG_Set	714	
49.3.8 CDOG_Add	714	
49.3.9 CDOG_Add1	714	
49.3.10 CDOG_Add16	714	
49.3.11 CDOG_Add256	715	
49.3.12 CDOG_Sub	715	
49.3.13 CDOG_Sub1	715	
49.3.14 CDOG_Sub16	715	
49.3.15 CDOG_Sub256	715	
49.3.16 CDOG_WritePersistent	716	
49.3.17 CDOG_ReadPersistent	716	
<b>Chapter 50 Dmic_dma_driver</b>		
<b>50.1 Overview</b>	<b>717</b>	
<b>50.2 Data Structure Documentation</b>	<b>718</b>	
50.2.1 struct dmic_transfer_t	718	
50.2.2 struct _dmic_dma_handle	718	
<b>50.3 Typedef Documentation</b>	<b>719</b>	
50.3.1 dmic_dma_transfer_callback_t	719	
<b>50.4 Function Documentation</b>	<b>719</b>	
50.4.1 DMIC_TransferCreateHandleDMA	719	
50.4.2 DMIC_TransferReceiveDMA	719	

Section No.	Title	Page No.
50.4.3	<a href="#">DMIC_TransferAbortReceiveDMA</a>	720
50.4.4	<a href="#">DMIC_TransferGetReceiveCountDMA</a>	720
50.4.5	<a href="#">DMIC_InstallDMADescriptorMemory</a>	721

## [Chapter 51 Flexspi\\_dma](#)

<b>51.1</b>	<a href="#">Overview</a>	<b>722</b>
<b>51.2</b>	<a href="#">Data Structure Documentation</a>	<b>723</b>
51.2.1	<a href="#">struct _flexspi_dma_handle</a>	723
<b>51.3</b>	<a href="#">Macro Definition Documentation</a>	<b>723</b>
51.3.1	<a href="#">FSL_FLEXSPI_DMA_DRIVER_VERSION</a>	723
<b>51.4</b>	<a href="#">Enumeration Type Documentation</a>	<b>724</b>
51.4.1	<a href="#">flexspi_dma_transfer_nsize_t</a>	724
<b>51.5</b>	<a href="#">Function Documentation</a>	<b>724</b>
51.5.1	<a href="#">FLEXSPI_TransferCreateHandleDMA</a>	724
51.5.2	<a href="#">FLEXSPI_TransferUpdateSizeDMA</a>	724
51.5.3	<a href="#">FLEXSPI_TransferDMA</a>	725
51.5.4	<a href="#">FLEXSPI_TransferAbortDMA</a>	726
51.5.5	<a href="#">FLEXSPI_TransferGetTransferCountDMA</a>	726

## [Chapter 52 I2c\\_dma\\_driver](#)

<b>52.1</b>	<a href="#">Overview</a>	<b>727</b>
<b>52.2</b>	<a href="#">Data Structure Documentation</a>	<b>728</b>
52.2.1	<a href="#">struct _i2c_master_dma_handle</a>	728
<b>52.3</b>	<a href="#">Macro Definition Documentation</a>	<b>728</b>
52.3.1	<a href="#">FSL_I2C_DMA_DRIVER_VERSION</a>	728
<b>52.4</b>	<a href="#">Typedef Documentation</a>	<b>729</b>
52.4.1	<a href="#">i2c_master_dma_transfer_callback_t</a>	729
52.4.2	<a href="#">flexcomm_i2c_dma_master_irq_handler_t</a>	729
<b>52.5</b>	<a href="#">Function Documentation</a>	<b>729</b>
52.5.1	<a href="#">I2C_MasterTransferCreateHandleDMA</a>	729
52.5.2	<a href="#">I2C_MasterTransferDMA</a>	729
52.5.3	<a href="#">I2C_MasterTransferGetCountDMA</a>	730
52.5.4	<a href="#">I2C_MasterTransferAbortDMA</a>	730

Section No.	Title	Page No.
<b>Chapter 53 I2s_dma_driver</b>		
<b>53.1 Overview</b>	<b>731</b>	
<b>53.2 Data Structure Documentation</b>	<b>732</b>	
53.2.1 struct _i2s_dma_handle	732	
<b>53.3 Macro Definition Documentation</b>	<b>732</b>	
53.3.1 FSL_I2S_DMA_DRIVER_VERSION	732	
<b>53.4 Typedef Documentation</b>	<b>732</b>	
53.4.1 i2s_dma_transfer_callback_t	732	
<b>53.5 Function Documentation</b>	<b>733</b>	
53.5.1 I2S_TxTransferCreateHandleDMA	733	
53.5.2 I2S_TxTransferSendDMA	733	
53.5.3 I2S_TransferAbortDMA	734	
53.5.4 I2S_RxTransferCreateHandleDMA	734	
53.5.5 I2S_RxTransferReceiveDMA	734	
53.5.6 I2S_DMACallback	735	
53.5.7 I2S_TransferInstallLoopDMADescriptorMemory	735	
53.5.8 I2S_TransferSendLoopDMA	736	
53.5.9 I2S_TransferReceiveLoopDMA	736	
<b>Chapter 54 Spi_dma_driver</b>		
<b>54.1 Overview</b>	<b>738</b>	
<b>54.2 Data Structure Documentation</b>	<b>739</b>	
54.2.1 struct _spi_dma_handle	739	
<b>54.3 Macro Definition Documentation</b>	<b>739</b>	
54.3.1 FSL_SPI_DMA_DRIVER_VERSION	739	
<b>54.4 Typedef Documentation</b>	<b>740</b>	
54.4.1 spi_dma_callback_t	740	
<b>54.5 Function Documentation</b>	<b>740</b>	
54.5.1 SPI_MasterTransferCreateHandleDMA	740	
54.5.2 SPI_MasterTransferDMA	740	
54.5.3 SPI_MasterHalfDuplexTransferDMA	741	
54.5.4 SPI_SlaveTransferCreateHandleDMA	741	
54.5.5 SPI_SlaveTransferDMA	742	
54.5.6 SPI_MasterTransferAbortDMA	742	
54.5.7 SPI_MasterTransferGetCountDMA	743	
54.5.8 SPI_SlaveTransferAbortDMA	743	

Section No.	Title	Page No.
54.5.9	SPI_SlaveTransferGetCountDMA .....	743
<b>Chapter 55 Usart_dma_driver</b>		
<b>55.1</b>	<b>Overview</b> .....	<b>745</b>
<b>55.2</b>	<b>Data Structure Documentation</b> .....	<b>746</b>
55.2.1	struct _usart_dma_handle .....	746
<b>55.3</b>	<b>Macro Definition Documentation</b> .....	<b>746</b>
55.3.1	FSL_USART_DMA_DRIVER_VERSION .....	746
<b>55.4</b>	<b>Typedef Documentation</b> .....	<b>747</b>
55.4.1	usart_dma_transfer_callback_t .....	747
<b>55.5</b>	<b>Function Documentation</b> .....	<b>747</b>
55.5.1	USART_TransferCreateHandleDMA .....	747
55.5.2	USART_TransferSendDMA .....	747
55.5.3	USART_TransferReceiveDMA .....	748
55.5.4	USART_TransferAbortSendDMA .....	748
55.5.5	USART_TransferAbortReceiveDMA .....	748
55.5.6	USART_TransferGetReceiveCountDMA .....	749
55.5.7	USART_TransferGetSendCountDMA .....	749
55.5.8	CODEC Adapter .....	751
55.5.9	Wm8904_adapter .....	752
<b>55.6</b>	<b>Wm8904</b> .....	<b>760</b>
55.6.1	Overview .....	760
55.6.2	Data Structure Documentation .....	764
55.6.3	Macro Definition Documentation .....	765
55.6.4	Enumeration Type Documentation .....	765
55.6.5	Function Documentation .....	768
<b>55.7</b>	<b>Serial_port_swo</b> .....	<b>777</b>
55.7.1	Overview .....	777
55.7.2	Data Structure Documentation .....	777
55.7.3	Enumeration Type Documentation .....	777
<b>55.8</b>	<b>Serial_port_uart</b> .....	<b>778</b>
55.8.1	Overview .....	778
55.8.2	Enumeration Type Documentation .....	778
<b>55.9</b>	<b>Serial_port_usb</b> .....	<b>779</b>
55.9.1	Overview .....	779
55.9.2	Data Structure Documentation .....	779
55.9.3	Enumeration Type Documentation .....	780

# 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<sup>TM</sup>. 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<sup>®</sup> 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

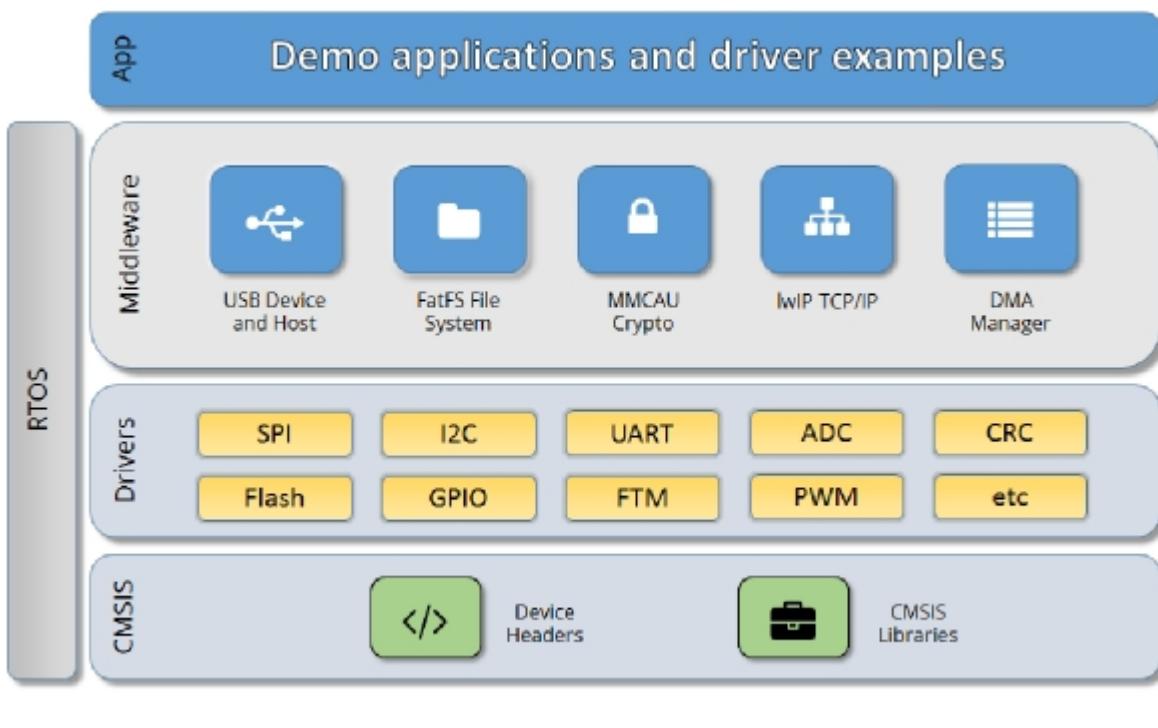
## Architectural Overview

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 (RC32M, SFLL, and so on) configuration
- Clock mux and divider configuration
- Getting clock frequency

### Files

- file `fsl_clock.h`

### Data Structures

- struct `clock_frg_clk_config_t`  
*PLL configuration for FRG.* [More...](#)
- struct `clock_avpll_config_t`  
*AVPLL configuration.* [More...](#)

### Macros

- `#define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0`  
*Configure whether driver controls clock.*
- `#define GPIO_CLOCKS`  
*Clock ip name array for GPIO.*
- `#define CACHE64_CLOCKS`  
*Clock ip name array for CACHE64.*
- `#define FLEXSPI_CLOCKS`  
*Clock ip name array for FLEXSPI.*
- `#define FLEXCOMM_CLOCKS`  
*Clock ip name array for FLEXCOMM.*
- `#define USART_CLOCKS`  
*Clock ip name array for LPUART.*
- `#define I2C_CLOCKS`  
*Clock ip name array for I2C.*
- `#define SPI_CLOCKS`  
*Clock ip name array for SPI.*
- `#define ACOMP_CLOCKS`  
*Clock ip name array for ACOMP.*
- `#define ADC_CLOCKS`  
*Clock ip name array for ADC.*
- `#define DAC_CLOCKS`

- `#define LCDIC_CLOCKS`  
*Clock ip name array for DAC.*
- `#define DMA_CLOCKS`  
*Clock ip name array for LCDIC.*
- `#define DMIC_CLOCKS`  
*Clock ip name array for DMA.*
- `#define ENET_CLOCKS`  
*Clock ip name array for DMIC.*
- `#define ENET_EXTRA_CLOCKS`  
*Clock ip name array for ENET.*
- `#define POWERQUAD_CLOCKS`  
*Extra clock ip name array for ENET.*
- `#define OSTIMER_CLOCKS`  
*Clock ip name array for Powerquad.*
- `#define CTIMER_CLOCKS`  
*Clock ip name array for OSTimer.*
- `#define UTICK_CLOCKS`  
*Clock ip name array for CT32B.*
- `#define MRT_CLOCKS`  
*Clock ip name array for UTICK.*
- `#define SCT_CLOCKS`  
*Clock ip name array for MRT.*
- `#define RTC_CLOCKS`  
*Clock ip name array for SCT.*
- `#define WWDT_CLOCKS`  
*Clock ip name array for RTC.*
- `#define TRNG_CLOCKS`  
*Clock ip name array for WWDT.*
- `#define USIM_CLOCKS`  
*Clock ip name array for TRNG.*
- `#define CLK_GATE_REG_OFFSET_SHIFT 8U`  
*Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.*

## Enumerations

- enum `clock_name_t` {
   
  `kCLOCK_CoreSysClk,`
  
  `kCLOCK_BusClk,`
  
  `kCLOCK_MclkClk }`
  
*Clock name used to get clock frequency.*
- enum `clock_ip_name_t`
  
*Peripheral clock name definition used for clock gate.*
- enum `clock_attach_id_t`
  
*Peripheral clock source selection definition.*
- enum `clock_div_name_t`
  
*Clock divider definition.*
- enum `clock_tcpc_flexspi_div_t` {
   
  `kCLOCK_TcpcFlexspiDiv12 = 0,`
  
  `kCLOCK_TcpcFlexspiDiv11,`
  
  `kCLOCK_TcpcFlexspiDiv10,`

- `kCLOCK_TcpuFlexspiDiv9 }`  
*TCPU PLL divider for tcpu\_mci\_flexspi\_clk.*
- enum `clock_tddr_flexspi_div_t` {
 `kCLOCK_TddrFlexspiDiv11` = 0,  
`kCLOCK_TddrFlexspiDiv10`,  
`kCLOCK_TddrFlexspiDiv9`,  
`kCLOCK_TddrFlexspiDiv8` }
- TDDR PLL divider for tddr\_mci\_flexspi\_clk.*
- enum `clock_t3_mci_irc_config_t` {
 `kCLOCK_T3MciIrc60m` = 0,  
`kCLOCK_T3MciIrc48m` }
- T3 PLL IRC configuration.*
- enum `clock_avpll_ch_freq_t` {
 `kCLOCK_AvPllChUnchanged` = 0,  
`kCLOCK_AvPllChFreq2p048m`,  
`kCLOCK_AvPllChFreq4p096m`,  
`kCLOCK_AvPllChFreq6p144m`,  
`kCLOCK_AvPllChFreq8p192m`,  
`kCLOCK_AvPllChFreq11p2896m`,  
`kCLOCK_AvPllChFreq12m`,  
`kCLOCK_AvPllChFreq12p288m`,  
`kCLOCK_AvPllChFreq24p576m`,  
`kCLOCK_AvPllChFreq64m`,  
`kCLOCK_AvPllChFreq98p304m` }
- AVPLL channel1 frequency configuration.*

## Functions

- `uint32_t CLOCK_GetT3PllMciIrcClkFreq (void)`  
*Return Frequency of t3pll\_mci\_48\_60m\_irc.*
- `uint32_t CLOCK_GetT3PllMci213mClkFreq (void)`  
*Return Frequency of t3pll\_mci\_213p3m.*
- `uint32_t CLOCK_GetT3PllMci256mClkFreq (void)`  
*Return Frequency of t3pll\_mci\_256m.*
- `uint32_t CLOCK_GetT3PllMciFlexspiClkFreq (void)`  
*Return Frequency of t3pll\_mci\_flexspi\_clk.*
- `uint32_t CLOCK_GetTcpuMciClkFreq (void)`  
*Return Frequency of tcpu\_mci\_clk.*
- `uint32_t CLOCK_GetTcpuMciFlexspiClkFreq (void)`  
*Return Frequency of tcpu\_mci\_flexspi\_clk.*
- `uint32_t CLOCK_GetTddrMciFlexspiClkFreq (void)`  
*Return Frequency of tddr\_mci\_flexspi\_clk.*
- `uint32_t CLOCK_GetTddrMciEnetClkFreq (void)`  
*Return Frequency of tddr\_mci\_enet\_clk.*
- void `CLOCK_EnableClock (clock_ip_name_t clk)`  
*Enable the clock for specific IP.*
- void `CLOCK_DisableClock (clock_ip_name_t clk)`  
*Disable the clock for specific IP.*
- void `CLOCK_AttachClk (clock_attach_id_t connection)`

- Configure the clock selection muxes.
- void **CLOCK\_SetClkDiv** (*clock\_div\_name\_t* name, *uint32\_t* divider)
  - Setup clock dividers.
- *uint32\_t* **CLOCK\_GetFreq** (*clock\_name\_t* *clockName*)
  - Return Frequency of selected clock.
- *uint32\_t* **CLOCK\_GetFRGClock** (*uint32\_t* id)
  - Return Input frequency for the Fractional baud rate generator.
- void **CLOCK\_SetFRGClock** (*const clock\_frg\_clk\_config\_t* \**config*)
  - Set output of the Fractional baud rate generator.
- *uint32\_t* **CLOCK\_GetFFroFreq** (*void*)
  - Return Frequency of FFRO.
- *uint32\_t* **CLOCK\_GetSFroFreq** (*void*)
  - Return Frequency of SFRO.
- *uint32\_t* **CLOCK\_GetAvPllCh1Freq** (*void*)
  - Return Frequency of AUDIO PLL (AVPLL CH1)
- *uint32\_t* **CLOCK\_GetAvPllCh2Freq** (*void*)
  - Return Frequency of AVPLL CH2.
- *uint32\_t* **CLOCK\_GetMainClkFreq** (*void*)
  - Return Frequency of main clk.
- *uint32\_t* **CLOCK\_GetCoreSysClkFreq** (*void*)
  - Return Frequency of core/bus clk.
- *uint32\_t* **CLOCK\_GetSystickClkFreq** (*void*)
  - Return Frequency of systick clk.
- static *uint32\_t* **CLOCK\_GetSysOscFreq** (*void*)
  - Return Frequency of sys osc Clock.
- static *uint32\_t* **CLOCK\_GetMclkInClkFreq** (*void*)
  - Return Frequency of MCLK Input Clock.
- static *uint32\_t* **CLOCK\_GetLpOscFreq** (*void*)
  - Return Frequency of LPOS.
- static *uint32\_t* **CLOCK\_GetClk32KFreq** (*void*)
  - Return Frequency of CLK\_32K.
- void **CLOCK\_EnableXtal32K** (*bool* enable)
  - Enables and disables 32KHz XTAL.
- void **CLOCK\_EnableRtc32K** (*bool* enable)
  - Enables and disables RTC 32KHz.
- static void **CLOCK\_SetClkinFreq** (*uint32\_t* freq)
  - Set the CLKIN (CLKIN pin) frequency based on GPIO4 input.
- static void **CLOCK\_SetMclkinFreq** (*uint32\_t* freq)
  - Set the MCLK in (mclk\_in) clock frequency based on board setting.
- *uint32\_t* **CLOCK\_GetDmicClkFreq** (*void*)
  - Return Frequency of DMIC clk.
- *uint32\_t* **CLOCK\_GetLcdClkFreq** (*void*)
  - Return Frequency of LCD clk.
- *uint32\_t* **CLOCK\_GetWdtClkFreq** (*void*)
  - Return Frequency of WDT clk.
- *uint32\_t* **CLOCK\_GetMclkClkFreq** (*void*)
  - Return Frequency of mclk.
- *uint32\_t* **CLOCK\_GetSctClkFreq** (*void*)
  - Return Frequency of sct.
- *uint32\_t* **CLOCK\_GetFlexCommClkFreq** (*uint32\_t* id)
  - Return Frequency of Flexcomm functional Clock.

- `uint32_t CLOCK_GetCTimerClkFreq (uint32_t id)`  
*Return Frequency of CTimer Clock.*
- `uint32_t CLOCK_GetUtickClkFreq (void)`  
*Return Frequency of Utick Clock.*
- `uint32_t CLOCK_GetFlexspiClkFreq (void)`  
*Return Frequency of Flexspi Clock.*
- `uint32_t CLOCK_GetUsimClkFreq (void)`  
*Return Frequency of USIM Clock.*
- `uint32_t CLOCK_GetGauClkFreq (void)`  
*Return Frequency of GAU Clock.*
- `uint32_t CLOCK_GetOSTimerClkFreq (void)`  
*Return Frequency of OSTimer Clock.*
- `uint32_t CLOCK_InitTcpuRefClk (uint32_t targetHz, clock_tcpu_flexspi_div_t div)`  
*Initialize TCPU FVCO to target frequency.*
- `void CLOCK_DeinitTcpuRefClk (void)`  
*Deinit the TCPU reference clock.*
- `void CLOCK_InitTddrRefClk (clock_tddr_flexspi_div_t div)`  
*Initialize the TDDR reference clock.*
- `void CLOCK_DeinitTddrRefClk (void)`  
*Deinit the TDDR reference clock.*
- `void CLOCK_InitT3RefClk (clock_t3_mci_irc_config_t cnfg)`  
*Initialize the T3 reference clock.*
- `void CLOCK_DeinitT3RefClk (void)`  
*Deinit the T3 reference clock.*
- `void CLOCK_InitAvPll (const clock_avpll_config_t *cnfg)`  
*Initialize the AVPLL.*
- `void CLOCK_DeinitAvPll (void)`  
*Deinit the AVPLL.*
- `void CLOCK_ConfigAvPllCh (clock_avpll_ch_freq_t ch1Freq, clock_avpll_ch_freq_t ch2Freq, bool enableCali)`  
*Update the AVPLL channel configuration.*
- `void CLOCK_EnableAvPllCh (bool enableCh1, bool enableCh2, bool enableCali)`  
*Enable the AVPLL channel.*
- `void CLOCK_DisableAvPllCh (bool disableCh1, bool disableCh2)`  
*Disable the AVPLL.*
- `void CLOCK_EnableUsbhsPhyClock (void)`  
*Enable USB HS PHY PLL clock.*
- `void CLOCK_DisableUsbhsPhyClock (void)`  
*Disable USB HS PHY PLL clock.*

## Variables

- `volatile uint32_t g_clkinFreq`  
*External CLK\_IN pin clock frequency (clkin) clock frequency.*
- `volatile uint32_t g_mclkInFreq`  
*External MCLK IN clock frequency.*

## Driver version

- `#define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`  
*CLOCK driver version 2.3.0.*

## 4.2 Data Structure Documentation

### 4.2.1 struct clock\_frg\_clk\_config\_t

#### Public Types

- enum {
   
    kCLOCK\_FrgMainClk = 0,  
 kCLOCK\_FrgPllDiv,  
 kCLOCK\_FrgSFro,  
 kCLOCK\_FrgFFro }

#### Data Fields

- uint8\_t num  
*FRG clock.*
- uint8\_t divider  
*Denominator of the fractional divider.*
- uint8\_t mult  
*Numerator of the fractional divider.*

#### Member Enumeration Documentation

##### (1) anonymous enum

Enumerator

- kCLOCK\_FrgMainClk** Main System clock.
- kCLOCK\_FrgPllDiv** Main pll clock divider.
- kCLOCK\_FrgSFro** 16MHz FRO
- kCLOCK\_FrgFFro** FRO48/60.

#### Field Documentation

##### (1) uint8\_t clock\_frg\_clk\_config\_t::divider

##### (2) uint8\_t clock\_frg\_clk\_config\_t::mult

### 4.2.2 struct clock\_avpll\_config\_t

#### Data Fields

- **clock\_avpll\_ch\_freq\_t ch1Freq**  
*AVPLL channel 1 frequency configuration.*
- **clock\_avpll\_ch\_freq\_t ch2Freq**  
*AVPLL channel 2 frequency configuration.*
- bool **enableCali**  
*Enable calibration.*

## 4.3 Macro Definition Documentation

### 4.3.1 #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.

### 4.3.2 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 0))

### 4.3.3 #define GPIO\_CLOCKS

**Value:**

```
{           \
    kCLOCK_HsGpio0, kCLOCK_HsGpio1 \
}
```

### 4.3.4 #define CACHE64\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Flexspi, kCLOCK_Flexspi \
}
```

### 4.3.5 #define FLEXSPI\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Flexspi \
}
```

#### 4.3.6 #define FLEXCOMM\_CLOCKS

**Value:**

```
{\n    kCLOCK_Flexcomm0, kCLOCK_Flexcomm1, kCLOCK_Flexcomm2, kCLOCK_Flexcomm3, kCLOCK_Flexcomm14 \n}
```

#### 4.3.7 #define USART\_CLOCKS

**Value:**

```
{\n    kCLOCK_Flexcomm0, kCLOCK_Flexcomm1, kCLOCK_Flexcomm2, kCLOCK_Flexcomm3 \n}
```

#### 4.3.8 #define I2C\_CLOCKS

**Value:**

```
{\n    kCLOCK_Flexcomm0, kCLOCK_Flexcomm1, kCLOCK_Flexcomm2, kCLOCK_Flexcomm3 \n}
```

#### 4.3.9 #define SPI\_CLOCKS

**Value:**

```
{\n    kCLOCK_Flexcomm0, kCLOCK_Flexcomm1, kCLOCK_Flexcomm2, kCLOCK_Flexcomm3, kCLOCK_Flexcomm14 \n}
```

#### 4.3.10 #define ACOMP\_CLOCKS

**Value:**

```
{\n    kCLOCK_Gau \n}
```

#### 4.3.11 #define ADC\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Gau, kCLOCK_Gau \
}
```

#### 4.3.12 #define DAC\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Gau \
}
```

#### 4.3.13 #define LCDIC\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Lcdic \
}
```

#### 4.3.14 #define DMA\_CLOCKS

**Value:**

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

#### 4.3.15 #define DMIC\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Dmic0 \
}
```

#### 4.3.16 #define ENET\_CLOCKS

**Value:**

```
{           \
    kCLOCK_EnetIpg \ \
}
```

#### 4.3.17 #define ENET\_EXTRA\_CLOCKS

**Value:**

```
{           \
    kCLOCK_EnetIpgS \ \
}
```

#### 4.3.18 #define CTIMER\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Ct32b0, kCLOCK_Ct32b1, kCLOCK_Ct32b2, kCLOCK_Ct32b3, kCLOCK_Ct32b4 \
}
```

#### 4.3.19 #define UTICK\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Utick \ \
}
```

#### 4.3.20 #define MRT\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Mrt, kCLOCK_FreeMrt \
}
```

#### 4.3.21 #define SCT\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Sct \
}
```

#### 4.3.22 #define RTC\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Rtc \
}
```

#### 4.3.23 #define WWDT\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Wwdt0 \
}
```

#### 4.3.24 #define TRNG\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Trng \
}
```

#### 4.3.25 #define USIM\_CLOCKS

**Value:**

```
{           \
    kCLOCK_Usim \
}
```

**4.3.26 #define CLK\_GATE\_REG\_OFFSET\_SHIFT 8U****4.4 Enumeration Type Documentation****4.4.1 enum clock\_name\_t**

Enumerator

*kCLOCK\_CoreSysClk* Core clock (aka HCLK)*kCLOCK\_BusClk* Bus clock (AHB/APB clock, aka HCLK)*kCLOCK\_MclkClk* MCLK, to MCLK pin.**4.4.2 enum clock\_tcpcu\_flexspi\_div\_t**

Enumerator

*kCLOCK\_TcpcuFlexspiDiv12* Divided by 12.*kCLOCK\_TcpcuFlexspiDiv11* Divided by 11.*kCLOCK\_TcpcuFlexspiDiv10* Divided by 10.*kCLOCK\_TcpcuFlexspiDiv9* Divided by 9.**4.4.3 enum clock\_tddr\_flexspi\_div\_t**

Enumerator

*kCLOCK\_TddrFlexspiDiv11* Divided by 11.*kCLOCK\_TddrFlexspiDiv10* Divided by 10.*kCLOCK\_TddrFlexspiDiv9* Divided by 9.*kCLOCK\_TddrFlexspiDiv8* Divided by 8.**4.4.4 enum clock\_t3\_mci\_irc\_config\_t**

Enumerator

*kCLOCK\_T3MciIrc60m* T3 MCI IRC 59.53MHz.*kCLOCK\_T3MciIrc48m* T3 MCI IRC 48.30MHz.**4.4.5 enum clock\_avpll\_ch\_freq\_t**

Enumerator

*kCLOCK\_AvPllChUnchanged* AVPLL channel frequency unchanged.

*kCLOCK\_AvPllChFreq2p048m* AVPLL channel frequency 2.048MHz.  
*kCLOCK\_AvPllChFreq4p096m* AVPLL channel frequency 4.096MHz.  
*kCLOCK\_AvPllChFreq6p144m* AVPLL channel frequency 6.144MHz.  
*kCLOCK\_AvPllChFreq8p192m* AVPLL channel frequency 8.192MHz.  
*kCLOCK\_AvPllChFreq11p2896m* AVPLL channel frequency 11.2896MHz.  
*kCLOCK\_AvPllChFreq12m* AVPLL channel frequency 12MHz.  
*kCLOCK\_AvPllChFreq12p288m* AVPLL channel frequency 12.288MHz.  
*kCLOCK\_AvPllChFreq24p576m* AVPLL channel frequency 24.576MHz.  
*kCLOCK\_AvPllChFreq64m* AVPLL channel frequency 64MHz.  
*kCLOCK\_AvPllChFreq98p304m* AVPLL channel frequency 98.304MHz.

## 4.5 Function Documentation

### 4.5.1 uint32\_t CLOCK\_GetT3PllMciIrcClkFreq ( void )

Returns

Frequency of t3pll\_mci\_48\_60m\_irc

### 4.5.2 uint32\_t CLOCK\_GetT3PllMci213mClkFreq ( void )

Returns

Frequency of t3pll\_mci\_213p3m

### 4.5.3 uint32\_t CLOCK\_GetT3PllMci256mClkFreq ( void )

Returns

Frequency of t3pll\_mci\_256m

### 4.5.4 uint32\_t CLOCK\_GetT3PllMciFlexspiClkFreq ( void )

Returns

Frequency of t3pll\_mci\_flexspi\_clk

### 4.5.5 uint32\_t CLOCK\_GetTcpuMciClkFreq ( void )

Returns

Frequency of tcpu\_mci\_clk

**4.5.6 uint32\_t CLOCK\_GetTcpuMciFlexspiClkFreq ( void )**

Returns

Frequency of tcpu\_mci\_flexspi\_clk

**4.5.7 uint32\_t CLOCK\_GetTddrMciFlexspiClkFreq ( void )**

Returns

Frequency of tddr\_mci\_flexspi\_clk

**4.5.8 uint32\_t CLOCK\_GetTddrMciEnetClkFreq ( void )**

Returns

Frequency of tddr\_mci\_enet\_clk

**4.5.9 void CLOCK\_EnableClock ( clock\_ip\_name\_t clk )**

Parameters

<i>clk</i>	Which clock to enable, see <a href="#">clock_ip_name_t</a> .
------------	--

**4.5.10 void CLOCK\_DisableClock ( clock\_ip\_name\_t clk )**

Parameters

<i>clk</i>	Which clock to disable, see <a href="#">clock_ip_name_t</a> .
------------	---

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

Parameters

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

#### 4.5.12 void CLOCK\_SetClkDiv ( *clock\_div\_name\_t name*, *uint32\_t divider* )

Parameters

<i>name</i>	: Clock divider name
<i>divider</i>	: Value to be divided.

#### 4.5.13 uint32\_t CLOCK\_GetFreq ( *clock\_name\_t clockName* )

Returns

Frequency of selected clock

#### 4.5.14 uint32\_t CLOCK\_GetFRGClock ( *uint32\_t id* )

Returns

Input Frequency for FRG

#### 4.5.15 void CLOCK\_SetFRGClock ( *const clock\_frg\_clk\_config\_t \* config* )

Parameters

<i>config</i>	: Configuration to set to FRGn clock.
---------------	---------------------------------------

#### 4.5.16 uint32\_t CLOCK\_GetFFroFreq ( *void* )

Returns

Frequency of FFRO

**4.5.17 uint32\_t CLOCK\_GetSFroFreq( void )**

Returns

Frequency of SFRO

**4.5.18 uint32\_t CLOCK\_GetAvPllCh1Freq( void )**

Returns

Frequency of AUDIO PLL

**4.5.19 uint32\_t CLOCK\_GetAvPllCh2Freq( void )**

Returns

Frequency of AVPLL CH2

**4.5.20 uint32\_t CLOCK\_GetMainClkFreq( void )**

Returns

Frequency of main clk

**4.5.21 uint32\_t CLOCK\_GetCoreSysClkFreq( void )**

Returns

Frequency of core/bus clk

**4.5.22 uint32\_t CLOCK\_GetSystickClkFreq( void )**

Returns

Frequency of systick clk

**4.5.23 static uint32\_t CLOCK\_GetSysOscFreq( void ) [inline], [static]**

Returns

Frequency of sys osc Clock. Or CLK\_IN pin frequency.

**4.5.24 static uint32\_t CLOCK\_GetMclkInClkFreq( void ) [inline], [static]**

Returns

Frequency of MCLK input Clock.

**4.5.25 static uint32\_t CLOCK\_GetLpOscFreq( void ) [inline], [static]**

Returns

Frequency of LPOSC

**4.5.26 static uint32\_t CLOCK\_GetClk32KFreq( void ) [inline], [static]**

Returns

Frequency of 32KHz osc

**4.5.27 void CLOCK\_EnableXtal32K( bool enable )**

Parameters

<i>enable</i>	: true to enable 32k XTAL clock, false to disable clock
---------------	---

**4.5.28 void CLOCK\_EnableRtc32K( bool enable )**

Parameters

<i>enable</i>	: true to enable 32k RTC clock, false to disable clock
---------------	--

#### 4.5.29 static void CLOCK\_SetClkinFreq( uint32\_t *freq* ) [inline], [static]

Parameters

<i>freq</i>	: The CLK_IN pin input clock frequency in Hz.
-------------	---

#### 4.5.30 static void CLOCK\_SetMclkinFreq( uint32\_t *freq* ) [inline], [static]

Parameters

<i>freq</i>	: The MCLK input clock frequency in Hz.
-------------	---

#### 4.5.31 uint32\_t CLOCK\_GetDmicClkFreq( void )

Returns

Frequency of DMIC clk

#### 4.5.32 uint32\_t CLOCK\_GetLcdClkFreq( void )

Returns

Frequency of LCD clk

#### 4.5.33 uint32\_t CLOCK\_GetWdtClkFreq( void )

Returns

Frequency of WDT clk

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

Returns

Frequency of mclk clk

**4.5.35 uint32\_t CLOCK\_GetSctClkFreq ( void )**

Returns

Frequency of sct clk

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

Parameters

<i>id</i>	: flexcomm index to get frequency.
-----------	------------------------------------

Returns

Frequency of Flexcomm functional Clock

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

Parameters

<i>id</i>	: ctimer index to get frequency.
-----------	----------------------------------

Returns

Frequency of CTimer Clock

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

Returns

Frequency of Utick Clock

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

Returns

Frequency of Flexspi.

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

Returns

Frequency of USIM.

**4.5.41 uint32\_t CLOCK\_GetGauClkFreq ( void )**

Returns

Frequency of GAU.

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

Returns

Frequency of OSTimer.

**4.5.43 uint32\_t CLOCK\_InitTcpuRefClk ( uint32\_t *targetHz*, clock\_tcpu\_flexspi\_div\_t *div* )**

For 40MHz XTAL, FVCO ranges from 3000MHz to 3840MHz.

For 38.4MHz XTAL, FVCO ranges from 2995.2MHz to 3840MHz

Parameters

<i>targetHz</i>	: Target FVCO frequency in Hz.
<i>div</i>	: Divider for tcpu_mci_flexspi_clk.

Returns

Actual FVCO frequency in Hz.

**4.5.44 void CLOCK\_InitTddrRefClk ( clock\_tddr\_flexspi\_div\_t *div* )**

Parameters

<i>div</i>	: Divider for tddr_mci_flexspi_clk.
------------	-------------------------------------

#### 4.5.45 void CLOCK\_InitT3RefClk ( **clock\_t3\_mci\_irc\_config\_t** *cfg* )

Parameters

<i>cfg</i>	: t3pll_mci_48_60m_irc clock configuration
------------	--

#### 4.5.46 void CLOCK\_DeinitT3RefClk ( void )

#### 4.5.47 void CLOCK\_InitAvPll ( **const clock\_avpll\_config\_t** \* *cfg* )

Both channel 1 and 2 are enabled.

Parameters

<i>cfg</i>	: AVPLL clock configuration
------------	-----------------------------

#### 4.5.48 void CLOCK\_DeinitAvPll ( void )

All channels are disabled.

#### 4.5.49 void CLOCK\_ConfigAvPllCh ( **clock\_avpll\_ch\_freq\_t** *ch1Freq*, **clock\_avpll\_ch\_freq\_t** *ch2Freq*, **bool** *enableCali* )

Enable/Disable state keeps unchanged.

Parameters

<i>ch1Freq</i>	: Channel 1 frequency to set.
<i>ch2Freq</i>	: Channel 2 frequency to set.

<i>enableCali</i>	: Enable AVPLL calibration.
-------------------	-----------------------------

#### 4.5.50 void CLOCK\_EnableAvPllCh ( bool *enableCh1*, bool *enableCh2*, bool *enableCali* )

Parameters

<i>enableCh1</i>	: Enable AVPLL channel1, channel unchanged on false.
<i>enableCh2</i>	: Enable AVPLL channel2, channel unchanged on false.
<i>enableCali</i>	: Enable AVPLL calibration.

#### 4.5.51 void CLOCK\_DisableAvPllCh ( bool *disableCh1*, bool *disableCh2* )

Parameters

<i>disableCh1</i>	: Disable AVPLL channel1, channel unchanged on false.
<i>disableCh2</i>	: Disable AVPLL channel2, channel unchanged on false.

#### 4.5.52 void CLOCK\_EnableUsbhsPhyClock ( void )

This function enables USB HS PHY PLL clock.

#### 4.5.53 void CLOCK\_DisableUsbhsPhyClock ( void )

This function disables USB HS PHY PLL clock.

### 4.6 Variable Documentation

#### 4.6.1 volatile uint32\_t g\_clkinFreq

The CLK\_IN pin (clkin) clock frequency in Hz, when the clock is setup, use the function CLOCK\_SetClkinFreq to set the value in to clock driver. For example, if CLK\_IN is 16MHz,

```
* CLOCK_SetClkinFreq(16000000);
*
```

## 4.6.2 volatile uint32\_t g\_mclkInFreq

The MCLK in (mclk\_in) PIN clock frequency in Hz, when the clock is setup, use the function CLOCK\_SetMclkInFreq to set the value in to clock driver. For example, if mclk\_In is 16MHz,

```
* CLOCK_SetMclkInFreq(16000000);  
*
```

## **Chapter 5**

### **I2S\_BRIDGE: I2S bridging and signal sharing configuration**

The MCUXpresso SDK provides a peripheral driver for the I2S Bridging/Signal Sharing module of MCUXpresso SDK devices. For furter details, see the corresponding chapter.

# Chapter 6

## IO\_MUX Driver

### 6.1 Overview

IO\_MUX driver supports peripheral IO multiplex configuration.

#### Files

- file [fsl\\_io\\_mux.h](#)

#### Driver version

- #define [FSL\\_IO\\_MUX\\_DRIVER\\_VERSION\(MAKE\\_VERSION\(2, 2, 0\)\)](#)  
*IO\_MUX driver version 2.2.0.*

#### Pin function ID

The pin function ID is a tuple of <GPIO\_0\_31\_Mask GPIO\_32\_63\_Mask GPIO\_FC\_SetMask GPIO\_FC\_ClearMask FSEL\_SetMask FSEL\_ClearMask CTimer\_SetMask CTimer\_ClearMask SCTimerSetMask SCTimerClearMask>

GPIO\_FC\_xxxMask: bit[0:10] maps to FCn[0:10]; bit[15:12] is the register offset from FC0; bit[16] indicates GPIO should be operated; bit[17] indicates SGPIO need to be operated. CTimer\_xxxMask: bit[0:14] maps to C\_TIMER\_IN[0:14]; bit[16:30] maps to C\_TIMER\_OUT[0:14].

- enum [io\\_mux\\_pin\\_config\\_t](#)  
*IO MUX pin configuration.*
- enum [io\\_mux\\_sleep\\_pin\\_level\\_t](#)  
*IO MUX sleep pin level.*
- #define [IO\\_MUX\\_GPIO\\_FC\\_MASK\(gpio, fcIdx, fcMsk\)](#) (((uint32\_t)(gpio) << 16) | (((uint32\_t)(fcIdx)&0xFUL) << 12) | ((uint32\_t)(fcMsk)&0xFFFFUL))
- #define [IO\\_MUX\\_SGPIO\\_FLAG\(mask\)](#) (((uint32\_t)(mask) >> 17) & 1UL)
- #define [IO\\_MUX\\_GPIO\\_FLAG\(mask\)](#) (((uint32\_t)(mask) >> 16) & 1UL)
- #define [IO\\_MUX\\_FC\\_OFFSET\(mask\)](#) (((uint32\_t)(mask) >> 12) & 0xFUL)
- #define [IO\\_MUX\\_FC\\_MASK\(mask\)](#) ((uint32\_t)(mask)&0x7FFUL)
- #define [IO\\_MUX\\_CTIMER\\_MASK\(inMsk, outMsk\)](#) (((uint32\_t)(outMsk) << 16) | ((uint32\_t)(inMsk)&0xFFFFUL))
- #define [IO\\_MUX\\_CTIMER\\_IN\\_MASK\(mask\)](#) ((uint32\_t)(mask)&0x7FFFUL)
- #define [IO\\_MUX\\_CTIMER\\_OUT\\_MASK\(mask\)](#) (((uint32\_t)(mask) >> 16) & 0x7FFFUL)
- #define [IO\\_MUX\\_SCTIMER\\_MASK\(inMsk, outMsk\)](#) (((((uint32\_t)(outMsk)&0x3FFUL) << 16) | ((uint32\_t)(inMsk)&0xFFUL)))
- #define [IO\\_MUX\\_FC0\\_USART\\_SCK](#)
- #define [IO\\_MUX\\_FC0\\_USART\\_DATA](#)
- #define [IO\\_MUX\\_FC0\\_USART\\_CMD](#)
- #define [IO\\_MUX\\_FC0\\_I2C\\_2\\_3](#)
- #define [IO\\_MUX\\_FC0\\_I2S](#)
- #define [IO\\_MUX\\_FC0\\_I2S\\_DATA](#)

- #define IO\_MUX\_FC0\_SPI\_SS0
- #define IO\_MUX\_FC1\_USART\_SCK
- #define IO\_MUX\_FC1\_USART\_DATA
- #define IO\_MUX\_FC1\_USART\_CMD
- #define IO\_MUX\_FC1\_I2C\_8\_9
- #define IO\_MUX\_FC1\_I2S
- #define IO\_MUX\_FC1\_I2S\_DATA
- #define IO\_MUX\_FC1\_SPI\_SS0
- #define IO\_MUX\_FC2\_USART\_SCK
- #define IO\_MUX\_FC2\_USART\_DATA
- #define IO\_MUX\_FC2\_USART\_CMD
- #define IO\_MUX\_FC2\_I2C\_13\_14
- #define IO\_MUX\_FC2\_I2C\_16\_17
- #define IO\_MUX\_FC2\_I2S
- #define IO\_MUX\_FC2\_I2S\_DATA
- #define IO\_MUX\_FC2\_SPI\_SS0
- #define IO\_MUX\_FC3\_USART\_SCK
- #define IO\_MUX\_FC3\_USART\_DATA
- #define IO\_MUX\_FC3\_USART\_CMD
- #define IO\_MUX\_FC3\_I2C\_24\_26
- #define IO\_MUX\_FC3\_I2C\_19\_20
- #define IO\_MUX\_FC3\_I2S
- #define IO\_MUX\_FC3\_I2S\_DATA
- #define IO\_MUX\_FC3\_SPI\_SS0
- #define IO\_MUX\_FC14\_USART\_SCK
- #define IO\_MUX\_FC14\_USART\_DATA
- #define IO\_MUX\_FC14\_USART\_CMD
- #define IO\_MUX\_FC14\_I2C\_56\_57
- #define IO\_MUX\_FC14\_I2S
- #define IO\_MUX\_FC14\_I2S\_DATA
- #define IO\_MUX\_FC14\_SPI\_SS0
- #define IO\_MUX\_QUAD\_SPI\_FLASH
- #define IO\_MUX\_QUAD\_SPI\_PSRAM
- #define IO\_MUX\_PDM
- #define IO\_MUX\_USB
- #define IO\_MUX\_SCT\_OUT\_0
- #define IO\_MUX\_SCT\_OUT\_1
- #define IO\_MUX\_SCT\_OUT\_8
- #define IO\_MUX\_SCT\_OUT\_4
- #define IO\_MUX\_SCT\_OUT\_5
- #define IO\_MUX\_SCT\_OUT\_6
- #define IO\_MUX\_SCT\_OUT\_7
- #define IO\_MUX\_SCT\_OUT\_9
- #define IO\_MUX\_SCT\_IN\_0
- #define IO\_MUX\_SCT\_IN\_1
- #define IO\_MUX\_SCT\_IN\_2
- #define IO\_MUX\_SCT\_IN\_3
- #define IO\_MUX\_SCT\_IN\_4
- #define IO\_MUX\_SCT\_IN\_5
- #define IO\_MUX\_SCT\_IN\_6
- #define IO\_MUX\_SCT\_IN\_7
- #define IO\_MUX\_CT0\_MAT0\_OUT
- #define IO\_MUX\_CT0\_MAT1\_OUT
- #define IO\_MUX\_CT0\_MAT2\_OUT
- #define IO\_MUX\_CT0\_MAT3\_OUT
- #define IO\_MUX\_CT1\_MAT0\_OUT

- #define IO\_MUX\_CT1\_MAT1\_OUT
- #define IO\_MUX\_CT1\_MAT2\_OUT
- #define IO\_MUX\_CT1\_MAT3\_OUT
- #define IO\_MUX\_CT2\_MAT0\_OUT
- #define IO\_MUX\_CT2\_MAT1\_OUT
- #define IO\_MUX\_CT2\_MAT2\_OUT
- #define IO\_MUX\_CT2\_MAT3\_OUT
- #define IO\_MUX\_CT3\_MAT0\_OUT
- #define IO\_MUX\_CT3\_MAT1\_OUT
- #define IO\_MUX\_CT3\_MAT2\_OUT
- #define IO\_MUX\_CT\_INP0
- #define IO\_MUX\_CT\_INP1
- #define IO\_MUX\_CT\_INP2
- #define IO\_MUX\_CT\_INP3
- #define IO\_MUX\_CT\_INP4
- #define IO\_MUX\_CT\_INP5
- #define IO\_MUX\_CT\_INP6
- #define IO\_MUX\_CT\_INP7
- #define IO\_MUX\_CT\_INP8
- #define IO\_MUX\_CT\_INP9
- #define IO\_MUX\_CT\_INP10
- #define IO\_MUX\_CT\_INP11
- #define IO\_MUX\_CT\_INP12
- #define IO\_MUX\_CT\_INP13
- #define IO\_MUX\_CT\_INP14
- #define IO\_MUX\_MCLK
- #define IO\_MUX\_UTICK
- #define IO\_MUX\_USIM
- #define IO\_MUX\_LCD\_8080
- #define IO\_MUX\_LCD\_SPI
- #define IO\_MUX\_FREQ\_GPIO\_CLK
- #define IO\_MUX\_GPIO\_INT\_BMATCH
- #define IO\_MUX\_GAU\_TRIGGER0
- #define IO\_MUX\_ACOMP0\_GPIO\_OUT
- #define IO\_MUX\_ACOMP0\_EDGE\_PULSE
- #define IO\_MUX\_ACOMP1\_GPIO\_OUT
- #define IO\_MUX\_ACOMP1\_EDGE\_PULSE
- #define IO\_MUX\_GAU\_TRIGGER1
- #define IO\_MUX\_SDIO
- #define IO\_MUX\_ENET\_CLK
- #define IO\_MUX\_ENET\_RX
- #define IO\_MUX\_ENET\_TX
- #define IO\_MUX\_ENET\_MDIO
- #define IO\_MUX\_ENET\_TIMER0
- #define IO\_MUX\_ENET\_TIMER1
- #define IO\_MUX\_ENET\_TIMER2
- #define IO\_MUX\_ENET\_TIMER3
- #define IO\_MUX\_CLKIN\_FRM\_PD
- #define IO\_MUX\_GPIO0
- #define IO\_MUX\_GPIO1
- #define IO\_MUX\_GPIO2
- #define IO\_MUX\_GPIO3
- #define IO\_MUX\_GPIO4
- #define IO\_MUX\_GPIO5
- #define IO\_MUX\_GPIO6
- #define IO\_MUX\_GPIO7

- #define IO\_MUX\_GPIO8
- #define IO\_MUX\_GPIO9
- #define IO\_MUX\_GPIO10
- #define IO\_MUX\_GPIO11
- #define IO\_MUX\_GPIO12
- #define IO\_MUX\_GPIO13
- #define IO\_MUX\_GPIO14
- #define IO\_MUX\_GPIO15
- #define IO\_MUX\_GPIO16
- #define IO\_MUX\_GPIO17
- #define IO\_MUX\_GPIO18
- #define IO\_MUX\_GPIO19
- #define IO\_MUX\_GPIO20
- #define IO\_MUX\_GPIO21
- #define IO\_MUX\_GPIO22
- #define IO\_MUX\_GPIO23
- #define IO\_MUX\_GPIO24
- #define IO\_MUX\_GPIO25
- #define IO\_MUX\_GPIO26
- #define IO\_MUX\_GPIO27
- #define IO\_MUX\_GPIO28
- #define IO\_MUX\_GPIO29
- #define IO\_MUX\_GPIO30
- #define IO\_MUX\_GPIO31
- #define IO\_MUX\_GPIO32
- #define IO\_MUX\_GPIO33
- #define IO\_MUX\_GPIO34
- #define IO\_MUX\_GPIO35
- #define IO\_MUX\_GPIO36
- #define IO\_MUX\_GPIO37
- #define IO\_MUX\_GPIO38
- #define IO\_MUX\_GPIO39
- #define IO\_MUX\_GPIO40
- #define IO\_MUX\_GPIO41
- #define IO\_MUX\_GPIO42
- #define IO\_MUX\_GPIO43
- #define IO\_MUX\_GPIO44
- #define IO\_MUX\_GPIO45
- #define IO\_MUX\_GPIO46
- #define IO\_MUX\_GPIO47
- #define IO\_MUX\_GPIO48
- #define IO\_MUX\_GPIO49
- #define IO\_MUX\_GPIO50
- #define IO\_MUX\_GPIO51
- #define IO\_MUX\_GPIO52
- #define IO\_MUX\_GPIO53
- #define IO\_MUX\_GPIO54
- #define IO\_MUX\_GPIO55
- #define IO\_MUX\_GPIO56
- #define IO\_MUX\_GPIO57
- #define IO\_MUX\_GPIO58
- #define IO\_MUX\_GPIO59
- #define IO\_MUX\_GPIO60
- #define IO\_MUX\_GPIO61
- #define IO\_MUX\_GPIO62
- #define IO\_MUX\_GPIO63

- #define IO\_MUX\_SGPIO0
- #define IO\_MUX\_SGPIO1
- #define IO\_MUX\_SGPIO2
- #define IO\_MUX\_SGPIO3
- #define IO\_MUX\_SGPIO4
- #define IO\_MUX\_SGPIO5
- #define IO\_MUX\_SGPIO6
- #define IO\_MUX\_SGPIO7
- #define IO\_MUX\_SGPIO8
- #define IO\_MUX\_SGPIO9
- #define IO\_MUX\_SGPIO10
- #define IO\_MUX\_SGPIO11
- #define IO\_MUX\_SGPIO12
- #define IO\_MUX\_SGPIO13
- #define IO\_MUX\_SGPIO14
- #define IO\_MUX\_SGPIO15
- #define IO\_MUX\_SGPIO16
- #define IO\_MUX\_SGPIO17
- #define IO\_MUX\_SGPIO18
- #define IO\_MUX\_SGPIO19
- #define IO\_MUX\_SGPIO20
- #define IO\_MUX\_SGPIO21
- #define IO\_MUX\_SGPIO22
- #define IO\_MUX\_SGPIO23
- #define IO\_MUX\_SGPIO24
- #define IO\_MUX\_SGPIO25
- #define IO\_MUX\_SGPIO26
- #define IO\_MUX\_SGPIO27
- #define IO\_MUX\_SGPIO28
- #define IO\_MUX\_SGPIO29
- #define IO\_MUX\_SGPIO30
- #define IO\_MUX\_SGPIO31
- #define IO\_MUX\_AON\_CAPTURE

## Configuration

- static void [IO\\_MUX\\_SetPinMux](#) (uint32\_t pinLowMask, uint32\_t pinHighMask, uint32\_t gpioFc-  
SetMask, uint32\_t gpioFcClrMask, uint32\_t fselSetMask, uint32\_t fselClrMask, uint32\_t ctimer-  
SetMask, uint32\_t ctimerClrMask, uint32\_t sctimerSetMask, uint32\_t sctimerClrMask)  
*Sets the IO\_MUX pin mux mode.*
- static void [IO\\_MUX\\_SetPinConfig](#) (uint32\_t pin, [io\\_mux\\_pin\\_config\\_t](#) config)  
*Sets the IO\_MUX pin mux pull up/down configuartion.*
- static void [IO\\_MUX\\_SetPinOutLevelInSleep](#) (uint32\_t pin, [io\\_mux\\_sleep\\_pin\\_level\\_t](#) level)  
*Sets IO output level in sleep mode.*
- static void [IO\\_MUX\\_SetRfPinOutLevelInSleep](#) (uint32\_t pin, [io\\_mux\\_sleep\\_pin\\_level\\_t](#) level)  
*Sets RF Switch Pin 0-3 output level in sleep mode.*

## 6.2 Macro Definition Documentation

### 6.2.1 #define FSL\_IO\_MUX\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))

## 6.3 Enumeration Type Documentation

### 6.3.1 enum io\_mux\_pin\_config\_t

Bit [1:0] for pull configuration Bit [3:2] for drive strength configuration

## 6.4 Function Documentation

### 6.4.1 static void IO\_MUX\_SetPinMux ( *uint32\_t pinLowMask, uint32\_t pinHighMask, uint32\_t gpioFcSetMask, uint32\_t gpioFcClrMask, uint32\_t fselSetMask, uint32\_t fselClrMask, uint32\_t ctimerSetMask, uint32\_t ctimerClrMask, uint32\_t sctimerSetMask, uint32\_t sctimerClrMask* ) [inline], [static]

Note

The parameters can be filled with the pin function ID macros.

This is an example to set the GPIO2/GPIO3 as the Flexcomm0 UART RX/TX:

```
* IO_MUX_SetPinMux (IO_MUX_FC0_USART_DATA);
*
```

This is an example to set the GPIO6/GPIO10 as Flexcomm1 I2C SDA/SCL:

```
* IO_MUX_SetPinMux (IO_MUX_FC1_I2C_6_10);
*
```

Parameters

<i>pinLowMask</i>	The GPIO0-31 pins mask.
<i>pinHighMask</i>	The GPIO32-63 pins mask.
<i>gpioFcSetMask</i>	The GPIO and Flexcomm registers mask to set, defined by IO_MUX_GPIO_FC_MASK()
<i>gpioFcClrMask</i>	The GPIO and Flexcomm registers mask to clear, defined by IO_MUX_GPIO_FC_MASK()
<i>fselSetMask</i>	The FSEL register mask to set

<i>fselClrMask</i>	The FSEL register mask to clear
<i>ctimerSetMask</i>	The C_TIMER_IN/C_TIMER_OUT register mask to set, defined by IO_MUX_C-TIMER_MASK()
<i>ctimerClrMask</i>	The C_TIMER_IN/C_TIMER_OUT register mask to clear, defined by IO_MUX_C-TIMER_MASK()
<i>sctimerSet-Mask</i>	The SC_TIMER register mask to set
<i>sctimerClr-Mask</i>	The SC_TIMER register mask to clear

#### 6.4.2 static void IO\_MUX\_SetPinConfig ( uint32\_t *pin*, io\_mux\_pin\_config\_t *config* ) [inline], [static]

This is an example to set the GPIO2 pin to pull down:

```
* IO_MUX_SetPinConfig(2U, IO_MUX_PinConfigPullDown);
*
```

Parameters

<i>pin</i>	The GPIO pin index to config.
<i>config</i>	The pull up/down setting for the pin.

#### 6.4.3 static void IO\_MUX\_SetPinOutLevelInSleep ( uint32\_t *pin*, io\_mux\_sleep\_pin\_level\_t *level* ) [inline], [static]

If level set to IO\_MUX\_SleepPinLevelUnchanged, the IO configuration is same as the active mode.

This is an example to set the GPIO2 pin to output high during sleep:

```
* IO_MUX_SetPinOutLevelInSleep(2U, IO_MUX_SleepPinLevelHigh);
*
```

Parameters

<i>pin</i>	The GPIO pin index to config.
<i>level</i>	Output level in sleep.

#### 6.4.4 static void IO\_MUX\_SetRfPinOutLevelInSleep ( uint32\_t *pin*, io\_mux\_sleep\_pin\_level\_t *level* ) [inline], [static]

If level set to IO\_MUX\_SleepPinLevelUnchanged, the IO configuration is same as the active mode.

This is an example to set the RF\_CNTL0 pin to output low during sleep:

```
* IO_MUX_SetRfPinOutLevelInSleep(0U, IO_MUX_SleepPinLevelLow);  
*
```

##### Parameters

<i>pin</i>	The RF Switch pin index to config.
<i>level</i>	Output level in sleep.

# **Chapter 7**

## **IPED Driver**

IPED driver abstracts register usage of IPED-relevant configuration registers of FLEXSPI peripheral.

# Chapter 8

## OCOTP Driver

### 8.1 Overview

OCOTP driver supports fuse read.

### Files

- file `fsl_ocotp.h`

### Macros

- #define `FSL_OCOTP_UID_LENGTH` 16U  
*OCOTP unique ID length.*

### Enumerations

- enum  
*OTP Status Group.*
- enum {  
  `kStatus_OOTP_InvalidAddress` = MAKE\_STATUS(kStatusGroup\_OtpGroup, 1),  
  `kStatus_OOTP_Timeout` = MAKE\_STATUS(kStatusGroup\_OtpGroup, 7) }  
*OTP Error Status definitions.*

### Functions

- `status_t OCOTP_OtpInit (void)`  
*Initialize OTP controller.*
- `status_t OCOTP_OtpDeinit (void)`  
*De-Initialize OTP controller.*
- `status_t OCOTP_OtpFuseRead (uint32_t addr, uint32_t *data)`  
*Read Fuse value from OTP Fuse Block.*
- `status_t OCOTP_ReadSocOtp (uint64_t *data, uint32_t tag)`  
*Read Fuse line with specific tag value from SoC OTP.*
- `status_t OCOTP_ReadUniqueId (uint8_t *uid, uint32_t *idLen)`  
*Read unique ID from OTP Fuse Block.*
- `status_t OCOTP_ReadSVC (uint64_t *svc)`  
*Read Static Voltage Compansation from SOC OTP.*
- `status_t OCOTP_ReadPackage (uint32_t *pack)`  
*Read package type from SOC OTP.*

### Driver version

- #define `FSL_OCOTP_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 1)`)  
*OCOTP driver version 2.2.1.*

## 8.2 Macro Definition Documentation

8.2.1 `#define FSL_OCOTP_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))`

8.2.2 `#define FSL_OCOTP_UID_LENGTH 16U`

## 8.3 Enumeration Type Documentation

8.3.1 anonymous enum

Enumerator

`kStatus OTP_InvalidAddress` Invalid OTP address.

`kStatus OTP_Timeout` OTP operation time out.

## 8.4 Function Documentation

8.4.1 `status_t OCOTP_OtpInit ( void )`

This function enables OTP Controller clock.

Returns

`kStatus_Success`

8.4.2 `status_t OCOTP_OtpDeinit ( void )`

This function disables OTP Controller Clock.

Returns

`kStatus_Success`

8.4.3 `status_t OCOTP_OtpFuseRead ( uint32_t addr, uint32_t * data )`

This function reads fuse data from OTP Fuse block to specified data buffer.

Parameters

---

<i>addr</i>	Fuse address
<i>data</i>	Buffer to hold the data read from OTP Fuse block

Returns

kStatus\_Success - Data read from OTP Fuse block successfully  
kStatus OTP\_Timeout - OTP read timeout  
kStatus\_InvalidArgument - data pointer is invalid

#### 8.4.4 status\_t OCOTP\_ReadSocOtp ( uint64\_t \* *data*, uint32\_t *tag* )

This function read Fuse line with specific tag value from SoC OTP to specified data buffer.

Parameters

<i>data</i>	Buffer to hold the data read from SoC OTP
<i>tag</i>	Tag value to match

Returns

kStatus\_Success - Data read from SoC OTP successfully  
kStatus\_Fail - Data read from SoC OTP failed, or cannot find the tag  
kStatus\_InvalidArgument - data pointer is invalid

#### 8.4.5 status\_t OCOTP\_ReadUniqueId ( uint8\_t \* *uid*, uint32\_t \* *idLen* )

This function read unique ID from OTP Fuse block to specified data buffer.

Parameters

<i>uid</i>	The buffer to store unique ID, buffer byte length is FSL_OCOTP_UID_LENGTH.
<i>idLen[in/out]</i>	The unique ID byte length. Set the length to read, return the length read out.

Returns

kStatus\_Success - Data read from OTP Fuse block successfully  
kStatus OTP\_Timeout - OTP read timeout  
kStatus\_InvalidArgument - data pointer is invalid

#### 8.4.6 status\_t OCOTP\_ReadSVC ( uint64\_t \* *svc* )

This function read SVC from OTP Fuse block to specified data buffer.

Parameters

<i>svc</i>	The buffer to store SVC.
------------	--------------------------

Returns

kStatus\_Success - Data read from SOC OTP successfully  
kStatus\_Fail - SOC OTP read failure

#### 8.4.7 status\_t OCOTP\_ReadPackage ( uint32\_t \* *pack* )

Parameters

<i>pack</i>	The buffer to store package type.
-------------	-----------------------------------

Returns

kStatus\_Success - Data read from SOC OTP successfully  
kStatus\_Fail - SOC OTP read failure

# Chapter 9

## Power Driver

### 9.1 Overview

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

### Files

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

### Data Structures

- struct [power\\_init\\_config\\_t](#)  
*Init configuration. [More...](#)*
- struct [power\\_sleep\\_config\\_t](#)  
*Sleep configuration. [More...](#)*
- struct [power\\_gdet\\_data\\_t](#)  
*Glitch detector configuration. [More...](#)*

### Typedefs

- [typedef void\(\\* capt\\_pulse\\_timer\\_callback\\_t \)\(void \\*param\)](#)  
*Capture timer callback function.*
- [typedef void\(\\* power\\_switch\\_callback\\_t \)\(uint32\\_t mode, void \\*param\)](#)  
*Power mode switch callback function.*
- [typedef bool\(\\* power\\_load\\_gdet\\_cfg \)\(power\\_gdet\\_data\\_t \\*data\)](#)  
*Glitch detector configuration load function.*

### Enumerations

- enum [power\\_wakeup\\_edge\\_t](#) {  
  [kPOWER\\_WakeupEdgeLow](#) = 0U,  
  [kPOWER\\_WakeupEdgeHigh](#) = 1U }  
*Pin edge for wakeup.*
- enum [power\\_wakeup\\_pin\\_t](#) {  
  [kPOWER\\_WakeupPin0](#) = 0U,  
  [kPOWER\\_WakeupPin1](#) = 1U }  
*Wakeup pin.*
- enum [power\\_reset\\_cause\\_t](#) {

```
kPOWER_ResetCauseSysResetReq = 1U << 0U,
kPOWER_ResetCauseLockup = 1U << 1U,
kPOWER_ResetCauseWdt = 1U << 2U,
kPOWER_ResetCauseApResetReq = 1U << 3U,
kPOWER_ResetCauseCodeWdt = 1U << 4U,
kPOWER_ResetCauseItrc = 1U << 5U,
kPOWER_ResetCauseResetB = 1U << 6U,
kPOWER_ResetCauseAll = 0x7FU }
```

*Reset cause.*

- enum power\_reset\_source\_t {
 kPOWER\_ResetSourceSysResetReq = 1U << 0U,
 kPOWER\_ResetSourceLockup = 1U << 1U,
 kPOWER\_ResetSourceWdt = 1U << 2U,
 kPOWER\_ResetSourceApResetReq = 1U << 3U,
 kPOWER\_ResetSourceCodeWdt = 1U << 4U,
 kPOWER\_ResetSourceItrc = 1U << 5U,
 kPOWER\_ResetSourceAll = 0x3FU }

*Reset source.*

- enum \_pm2\_mem\_pu\_bits
 *PM2 mem power up bits definition.*
  - enum \_pm2\_ana\_pu\_bits
 *PM2 ana power up bits definition.*
  - enum \_clk\_gate\_bits
 *clock gate bits definition*
  - enum \_clk\_pm3\_buck\_bits {
 kPOWER\_Pm3Buck18 = (1UL << 7),
 kPOWER\_Pm3Buck11 = (1UL << 6) }
- PM3 buck control bits definition.*
- enum capt\_slow\_pulse\_width\_t
 *Capture slow pulse width.*
  - enum capt\_slow\_pulse\_edge\_t
 *Capture slow pulse edge.*

## Functions

- \_\_STATIC\_INLINE void POWER\_EnableResetSource (uint32\_t source)
 *Enable system reset source.*
- \_\_STATIC\_INLINE void POWER\_DisableResetSource (uint32\_t source)
 *Disable system reset source.*
- \_\_STATIC\_INLINE uint32\_t POWER\_GetResetCause (void)
 *Get last reset cause.*
- \_\_STATIC\_INLINE void POWER\_ClearResetCause (uint32\_t cause)
 *Clear last reset cause.*
- \_\_STATIC\_INLINE void POWER\_ConfigWakeupPin (power\_wakeup\_pin\_t pin, power\_wakeup\_edge\_t edge)
 *Configure pin edge for wakeup.*
- bool POWER\_GetWakeupStatus (IRQn\_Type irq)
 *Check if IRQ is the wakeup source.*
- void POWER\_ClearWakeupStatus (IRQn\_Type irq)

- Clear wakeup status.
- void **POWER\_EnableWakeup** (IRQn\_Type irq)  
*Enable the Wakeup interrupt.*
- void **POWER\_DisableWakeup** (IRQn\_Type irq)  
*Disable the Wakeup interrupts.*
- **AT\_QUICKACCESS\_SECTION\_CODE** (void POWER\_SetSleepMode(uint32\_t mode))  
*Set power mode on idle.*
- \_\_STATIC\_INLINE uint32\_t **POWER\_GetWakenMode** (void)  
*Get power mode waken up from.*
- void **POWER\_GetCurrentSleepConfig** (power\_sleep\_config\_t \*config)  
*Get current sleep configuration.*
- void **POWER\_InitPowerConfig** (const power\_init\_config\_t \*config)  
*Initialize power configuration.*
- void **POWER\_ConfigCauInSleep** (bool pdCau)  
*Configure CAU\_SOC\_SLP\_REF\_GEN\_CLK on/off status in SoC sleep mode.*
- void **POWER\_SetPowerSwitchCallback** (power\_switch\_callback\_t pre, void \*preParam, power\_switch\_callback\_t post, void \*postParam)  
*Set power mode switch callback. The callbacks are called with interrupt disabled.*
- **AT\_QUICKACCESS\_SECTION\_CODE** (bool POWER\_EnterPowerMode(uint32\_t mode, const power\_sleep\_config\_t \*config))  
*Switch system into certain power mode.*
- void **POWER\_PowerOnWlan** (void)  
*Power on WLAN.*
- void **POWER\_PowerOffWlan** (void)  
*Power off WLAN.*
- \_\_STATIC\_INLINE void **PMU\_EnableWlanWakeups** (uint8\_t wlWakeups)  
*Enable MCI wakeup WLAN.*
- \_\_STATIC\_INLINE void **PMU\_DisableWlanWakeups** (uint8\_t wlWakeups)  
*Disable MCI wakeup WLAN.*
- void **POWER\_PowerOnBle** (void)  
*Power on BLE.*
- void **POWER\_PowerOffBle** (void)  
*Power off BLE.*
- \_\_STATIC\_INLINE void **PMU\_EnableBleWakeups** (uint8\_t bleWakeups)  
*Enable MCI wakeup BLE.*
- \_\_STATIC\_INLINE void **PMU\_DisableBleWakeups** (uint8\_t bleWakeups)  
*Disable MCI wakeup BLE.*
- void **POWER\_PowerOnGau** (void)  
*Power on GAU.*
- void **POWER\_PowerOffGau** (void)  
*Power off GAU.*
- void **POWER\_EnableCaptSlowPulseTimer** (capt\_slow\_pulse\_width\_t width, capt\_slow\_pulse\_edge\_t edge, uint32\_t timeout, capt\_pulse\_timer\_callback\_t cb, void \*param)  
*Enable capture slow pulse timer with 32768Hz clock source.*
- void **POWER\_EnableCaptFastPulseTimer** (uint32\_t timeout, capt\_pulse\_timer\_callback\_t cb, void \*param)  
*Enable capture fast pulse timer with 3.84/4MHz clock source.*
- void **POWER\_DisableCaptPulseTimer** (void)  
*Disable capture pulse timer.*
- void **POWER\_InitVoltage** (uint32\_t dro, uint32\_t pack)  
*Configure power rail voltage and LVD/HVD threshold.*

- void `Power_InitLoadGdetCfg` (`power_load_gdet_cfg` `loadFunc`, const `power_gdet_data_t` \*`data`, `uint32_t` `pack`)  
*Initialize glitch detector configuration.*
- `AT_QUICKACCESS_SECTION_CODE` (void `POWER_DisableGDetVSensors(void)`)  
*Disable GDET and VSensors.*
- `AT_QUICKACCESS_SECTION_CODE` (bool `POWER_EnableGDetVSensors(void)`)  
*Enable GDET and VSensors.*
- `uint32_t POWER_TrimSvc` (`uint32_t` `gdetTrim`, `uint32_t` `pack`)  
*Apply SVC GDC equation and get the SVC trim configuration.*

## Driver version

- #define `FSL_POWER_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 0)`)  
*POWER driver version 2.5.0.*

## 9.2 Data Structure Documentation

### 9.2.1 struct power\_init\_config\_t

#### Data Fields

- bool `iBuck`  
*true: VCORE and AVDD18 supplied from iBuck; false: supplied from external DCDC.*
- bool `gateCauRefClk`  
*true: CAU\_SOC\_SLP\_REF\_GEN\_CLK gated; false: CAU\_SOC\_SLP\_REF\_GEN\_CLK on.*

#### Field Documentation

- (1) `bool power_init_config_t::iBuck`
- (2) `bool power_init_config_t::gateCauRefClk`

### 9.2.2 struct power\_sleep\_config\_t

#### Data Fields

- `uint32_t pm2MemPuCfg`  
*Modules to keep powered on in PM2 mode.*
- `uint32_t pm2AnaPuCfg`  
*Ana to keep powered on in PM2 mode.*
- `uint32_t clkGate`  
*Source clock gate control.*
- `uint32_t memPdCfg`  
*PMU MEM\_CFG: Power Down memory configuration.*
- `uint32_t pm3BuckCfg`  
*PMIP BUCK control in PM3 mode.*

**Field Documentation**

(1) `uint32_t power_sleep_config_t::pm2MemPuCfg`

Logical OR of the enums in `_pm2_mem_pu_bits`.

(2) `uint32_t power_sleep_config_t::pm2AnaPuCfg`

Logical OR of the enums in `_pm2_ana_pu_bits`.

(3) `uint32_t power_sleep_config_t::clkGate`

Logical OR of the enums in `_clk_gate_bits`.

(4) `uint32_t power_sleep_config_t::memPdCfg`

Bit0-5 for PM3, bit8 for PM4. bit0: ram0-5 384KB bit1: ram6 64KB bit2: ram7 64KB bit3: ram8-9 128KB bit4: ram10-13 256KB bit5: ram14-18 320KB. bit8: aon mem higher 8KB

(5) `uint32_t power_sleep_config_t::pm3BuckCfg`

Logical OR of the enums in `_clk_pm3_buck_bits`.

**9.2.3 struct power\_gdet\_data\_t****9.3 Macro Definition Documentation**

**9.3.1 #define FSL\_POWER\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 0))**

**9.4 Typedef Documentation**

**9.4.1 typedef void(\* capt\_pulse\_timer\_callback\_t)(void \*param)**

Parameters

<i>param</i>	: User parameter for callback.
--------------	--------------------------------

**9.4.2 typedef void(\* power\_switch\_callback\_t)(uint32\_t mode, void \*param)**

Parameters

<i>mode</i>	: Power mode to switch.
<i>param</i>	: User parameter for callback.

## 9.5 Enumeration Type Documentation

### 9.5.1 enum power\_wakeup\_edge\_t

Enumerator

*kPOWER\_WakeupEdgeLow* Wakeup on pin low level.

*kPOWER\_WakeupEdgeHigh* Wakeup on pin high level.

### 9.5.2 enum power\_wakeup\_pin\_t

Enumerator

*kPOWER\_WakeupPin0* Wakeup0 pin.

*kPOWER\_WakeupPin1* Wakeup1 pin.

### 9.5.3 enum power\_reset\_cause\_t

Enumerator

*kPOWER\_ResetCauseSysResetReq* CM33 system soft reset request.

*kPOWER\_ResetCauseLockup* CM33 locked up.

*kPOWER\_ResetCauseWdt* Watchdog timer.

*kPOWER\_ResetCauseApResetReq* Debug mailbox reset.

*kPOWER\_ResetCauseCodeWdt* Code watchdog timer.

*kPOWER\_ResetCauseItrc* ITRC\_CHIP reset.

*kPOWER\_ResetCauseResetB* sw\_resetb\_scantest reset.

*kPOWER\_ResetCauseAll* All reset causes. Used in [POWER\\_ClearResetCause\(\)](#).

### 9.5.4 enum power\_reset\_source\_t

Enumerator

*kPOWER\_ResetSourceSysResetReq* CM33 system soft reset request.

*kPOWER\_ResetSourceLockup* CM33 locked up.

*kPOWER\_ResetSourceWdt* Watchdog timer.

*kPOWER\_ResetSourceApResetReq* Debug mailbox reset.

*kPOWER\_ResetSourceCodeWdt* Code watchdog timer.

*kPOWER\_ResetSourceItrc* ITRC\_CHIP reset.

*kPOWER\_ResetSourceAll* All reset sources.

### 9.5.5 enum \_clk\_pm3\_buck\_bits

Enumerator

*kPOWER\_Pm3Buck18* 1: Use normal buck18 level in PM3. 0: Use sleep buck18 level in PM3

*kPOWER\_Pm3Buck11* 1: Use normal buck11 level in PM3. 0: Use sleep buck11 level in PM3

## 9.6 Function Documentation

### 9.6.1 \_\_STATIC\_INLINE void POWER\_EnableResetSource ( uint32\_t source )

Parameters

<i>source</i>	: A bitmask of of <a href="#">power_reset_source_t</a>
---------------	--

### 9.6.2 \_\_STATIC\_INLINE void POWER\_DisableResetSource ( uint32\_t source )

Parameters

<i>source</i>	: A bitmask of of <a href="#">power_reset_source_t</a>
---------------	--

### 9.6.3 \_\_STATIC\_INLINE uint32\_t POWER\_GetResetCause ( void )

Returns

Or'ed cause of [power\\_reset\\_cause\\_t](#)

### 9.6.4 \_\_STATIC\_INLINE void POWER\_ClearResetCause ( uint32\_t cause )

Parameters

<i>cause</i>	: A bitmask of of <a href="#">power_reset_cause_t</a>
--------------	---

### 9.6.5 **`__STATIC_INLINE void POWER_ConfigWakeupPin ( power_wakeup_pin_t pin, power_wakeup_edge_t edge )`**

Parameters

<i>pin</i>	: Wakeup pin
<i>edge</i>	: Pin level for wakeup

### 9.6.6 **`bool POWER_GetWakeupStatus ( IRQn_Type irq )`**

Parameters

<i>irq</i>	: IRQ number
------------	--------------

Returns

true if IRQ is the wakeup source, false otherwise.

### 9.6.7 **`void POWER_ClearWakeupStatus ( IRQn_Type irq )`**

Parameters

<i>irq</i>	: IRQ number
------------	--------------

### 9.6.8 **`void POWER_EnableWakeup ( IRQn_Type irq )`**

Parameters

<i>irq</i>	: IRQ number
------------	--------------

### 9.6.9 **`void POWER_DisableWakeup ( IRQn_Type irq )`**

Parameters

<i>irq</i>	: IRQ number
------------	--------------

### 9.6.10 AT\_QUICKACCESS\_SECTION\_CODE ( void *POWER\_SetSleepMode*uint32\_t *mode* )

Parameters

<i>mode</i>	: 0 ~ 4 stands for PM0 ~ PM4.
-------------	-------------------------------

### 9.6.11 \_\_STATIC\_INLINE uint32\_t *POWER\_GetWakenMode* ( void )

Returns

Power mode.

### 9.6.12 void *POWER\_GetCurrentSleepConfig* ( power\_sleep\_config\_t \* *config* )

Parameters

<i>config</i>	: Pointer to config structure to save current config.
---------------	---

### 9.6.13 void *POWER\_InitPowerConfig* ( const power\_init\_config\_t \* *config* )

Parameters

<i>config</i>	: Pointer to init config structure.
---------------	-------------------------------------

### 9.6.14 void *POWER\_ConfigCauInSleep* ( bool *pdCau* )

Parameters

<i>pdCau</i>	: true for clock off; false for clock on.
--------------	---

### 9.6.15 void POWER\_SetPowerSwitchCallback ( *power\_switch\_callback\_t pre*, *void \* preParam*, *power\_switch\_callback\_t post*, *void \* postParam* )

Parameters

<i>pre</i>	: Function called before power mode switch
<i>preParam</i>	: User parameter for pre callback
<i>post</i>	: Function called after power mode switch
<i>postParam</i>	: User parameter for post callback

### 9.6.16 AT\_QUICKACCESS\_SECTION\_CODE ( *bool POWER\_EnterPowerMode**uint32\_t mode*, *const power\_sleep\_config\_t \*config* )

Parameters

<i>mode</i>	: 0 ~ 4 stands for PM0 ~ PM4.
<i>config</i>	: Sleep configuration on PM2-PM4.

Returns

True for success, else failure.

### 9.6.17 \_\_STATIC\_INLINE void PMU\_EnableWlanWakeups ( *uint8\_t wlWakeup* )

Parameters

<i>wlWakeup</i>	: 8 bits wakeup mask
-----------------	----------------------

### 9.6.18 \_\_STATIC\_INLINE void PMU\_DisableWlanWakeups ( *uint8\_t wlWakeup* )

Parameters

<i>wlWakeup</i>	: 8 bits wakeup mask
-----------------	----------------------

### 9.6.19 **\_\_STATIC\_INLINE void PMU\_EnableBleWakeup ( uint8\_t *bleWakeup* )**

Parameters

<i>bleWakeup</i>	: 8 bits wakeup mask
------------------	----------------------

### 9.6.20 **\_\_STATIC\_INLINE void PMU\_DisableBleWakeup ( uint8\_t *bleWakeup* )**

Parameters

<i>bleWakeup</i>	: 8 bits wakeup mask
------------------	----------------------

### 9.6.21 **void POWER\_EnableCaptSlowPulseTimer ( capt\_slow\_pulse\_width\_t *width*, capt\_slow\_pulse\_edge\_t *edge*, uint32\_t *timeout*, capt\_pulse\_timer\_callback\_t *cb*, void \* *param* )**

Parameters

<i>width</i>	: input capture filter width in cycles
<i>edge</i>	: trigger condition of counter
<i>timeout</i>	: timer expire counter which will trigger callback
<i>callback</i>	: callback function on timer expire
<i>param</i>	: callback parameter

### 9.6.22 **void POWER\_EnableCaptFastPulseTimer ( uint32\_t *timeout*, capt\_pulse\_timer\_callback\_t *cb*, void \* *param* )**

Parameters

<i>timeout</i>	: timer expire counter which will trigger callback
<i>callback</i>	: callback function on timer expire
<i>param</i>	: callback parameter

### 9.6.23 void POWER\_InitVoltage ( *uint32\_t dro, uint32\_t pack* )

Parameters

<i>dro</i>	: trim value from fuse.
<i>pack</i>	: Device package type: 0 - QFN, 1 - CSP, 2 - BGA

### 9.6.24 void Power\_InitLoadGdetCfg ( *power\_load\_gdet\_cfg loadFunc, const power\_gdet\_data\_t \* data, uint32\_t pack* )

Parameters

<i>loadFunc</i>	: function pointer to the GDET load configuration.
<i>data</i>	: GDET config data loaded from fuse.
<i>pack</i>	: Device package type: 0 - QFN, 1 - CSP, 2 - BGA

### 9.6.25 AT\_QUICKACCESS\_SECTION\_CODE ( *bool POWER\_EnableGDetV-Sensorsvoid* )

Returns

True for success, else failure.

### 9.6.26 *uint32\_t POWER\_TrimSvc ( *uint32\_t gdetTrim, uint32\_t pack* )*

## Parameters

<i>gdetTrim</i>	: GDET trim value from fuse.
<i>pack</i>	: Device package type: 0 - QFN, 1 - CSP, 2 - BGA

# Chapter 10

## Reset Driver

### 10.1 Overview

Reset driver supports peripheral reset and system reset.

#### Macros

- `#define RST_CTL0_PSCCTL0 0`  
*Reset control registers index.*
- `#define CRC_RSTS`

#### Typedefs

- `typedef RSTCTL_RSTn_t reset_ip_name_t`  
*IP reset handle.*

## Enumerations

- enum `RSTCTL_RSTn_t` {
   
`kPOWERQUAD_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 8U,
   
`kPKC_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 9U,
   
`kELS_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 10U,
   
`kPUF_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 11U,
   
`kFLEXSPI_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 16U,
   
`kHPU_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 20U,
   
`kUSB_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 22U,
   
`kSCT_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 24U,
   
`kaON_MEM_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 25U,
   
`kGDMA_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 28U,
   
`kDMA0_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 29U,
   
`kDMA1_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 30U,
   
`kSDIO_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL0 << 8`) | 31U,
   
`kELS_APB_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 0U,
   
`kELS_GDET_REF_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 1U,
   
`kSDIO_SLV_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 2U,
   
`kGAU_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 16U,
   
`kOTP_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 17U,
   
`kSECURE_GPIO_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 24U,
   
`kENET_IPG_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 25U,
   
`kENET_IPG_S_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 26U,
   
`kTRNG_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL1 << 8`) | 27U,
   
`kUTICK_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL2 << 8`) | 0U,
   
`kWWDT_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL2 << 8`) | 1U,
   
`kUSIM_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL2 << 8`) | 2U,
   
`kFREEMRT_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL2 << 8`) | 26U,
   
`kLCDIC_RST_SHIFT_RSTn` = (`RST_CTL0_PSCCTL2 << 8`) | 27U,
   
`kFC0_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL0 << 8`) | 8U,
   
`kFC1_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL0 << 8`) | 9U,
   
`kFC2_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL0 << 8`) | 10U,
   
`kFC3_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL0 << 8`) | 11U,
   
`kFC14_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL0 << 8`) | 22U,
   
`kDMIC_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL0 << 8`) | 24U,
   
`kOSEVENT_TIMER_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL0 << 8`) | 27U,
   
`kHGPIO0_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL1 << 8`) | 0U,
   
`kHGPIO1_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL1 << 8`) | 1U,
   
`kCRC_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL1 << 8`) | 16U,
   
`kFREQME_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL1 << 8`) | 31U,
   
`kCT32B0_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL2 << 8`) | 0U,
   
`kCT32B1_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL2 << 8`) | 1U,
   
`kCT32B2_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL2 << 8`) | 2U,
   
`kCT32B3_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL2 << 8`) | 3U,
   
`kCT32B4_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL2 << 8`) | 4U,
   
`kMRT_RST_SHIFT_RSTn` = (`RST_CTL1_PSCCTL2 << 8`) | 8U,

**kINPUTMUX\_RST\_SHIFT\_RSTn = (RST\_CTL1\_PSCCTL2 << 8) | 31U }**

*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.*
- static void **RESET\_ReleasePeripheralReset** (reset\_ip\_name\_t peripheral)  
*Release peripheral module.*

## Driver version

- #define **FSL\_RESET\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 1))  
*reset driver version 2.1.1.*

## 10.2 Macro Definition Documentation

### 10.2.1 #define FSL\_RESET\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

### 10.2.2 #define CRC\_RSTS

#### Value:

```
{
    \_kCRC_RST_SHIFT_RSTn \
} /* Reset bits for CRC peripheral */
```

Array initializers with peripheral reset bits

## 10.3 Enumeration Type Documentation

### 10.3.1 enum RSTCTL\_RSTn\_t

Defines the enumeration for peripheral reset control bits in RSTCLTx registers

Enumerator

**kPOWERQUAD\_RST\_SHIFT\_RSTn** POWERQUAD reset control  
**kPKC\_RST\_SHIFT\_RSTn** PKC reset control  
**kELS\_RST\_SHIFT\_RSTn** ELS reset control  
**kPUF\_RST\_SHIFT\_RSTn** Physical unclonable function reset control  
**kFLEXSPI\_RST\_SHIFT\_RSTn** FLEXSPI reset control  
**kHPU\_RST\_SHIFT\_RSTn** HPU reset control

***kUSB\_RST\_SHIFT\_RSTn*** USB reset control  
***kSCT\_RST\_SHIFT\_RSTn*** Standard ctimers reset control  
***KAON\_MEM\_RST\_SHIFT\_RSTn*** AON MEM reset control  
***kGDMA\_RST\_SHIFT\_RSTn*** GDMA reset control  
***kDMA0\_RST\_SHIFT\_RSTn*** DMA0 reset control  
***kDMA1\_RST\_SHIFT\_RSTn*** DMA1 reset control  
***kSDIO\_RST\_SHIFT\_RSTn*** SDIO reset control  
***kELS\_APB\_RST\_SHIFT\_RSTn*** ELS\_APB reset control  
***kELS\_GDET\_REF\_RST\_SHIFT\_RSTn*** ELS\_GDET\_REF\_RST reset control  
***kSDIO\_SLV\_SHIFT\_RSTn*** SDIO\_SLV reset control  
***kGAU\_RST\_SHIFT\_RSTn*** GAU reset control  
***kOTP\_RST\_SHIFT\_RSTn*** OTP reset control  
***kSECURE\_GPIO\_RST\_SHIFT\_RSTn*** Security GPIO reset control  
***kENET\_IPG\_RST\_SHIFT\_RSTn*** ENET\_IPG reset control  
***kENET\_IPG\_S\_RST\_SHIFT\_RSTn*** ENET\_IPG\_S reset control  
***kTRNG\_RST\_SHIFT\_RSTn*** TRNG reset control  
***kUTICK\_RST\_SHIFT\_RSTn*** Micro-tick timer reset control  
***kWWDT\_RST\_SHIFT\_RSTn*** Windowed Watchdog timer reset control  
***kUSIM\_RST\_SHIFT\_RSTn*** USIM reset control  
***kFREEMRT\_RST\_SHIFT\_RSTn*** FREEMRT reset control  
***kLCDIC\_RST\_SHIFT\_RSTn*** LCDIC 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  
***kFC14\_RST\_SHIFT\_RSTn*** Flexcomm Interface 14 reset control  
***kDMIC\_RST\_SHIFT\_RSTn*** Digital microphone interface reset control  
***kOSEVENT\_TIMER\_RST\_SHIFT\_RSTn*** Osevent Timer reset control  
***kHGPIO0\_RST\_SHIFT\_RSTn*** HGPIO 0 reset control  
***kHGPIO1\_RST\_SHIFT\_RSTn*** HGPIO 1 reset control  
***kCRC\_RST\_SHIFT\_RSTn*** CRC reset control  
***kFREQME\_RST\_SHIFT\_RSTn*** Frequency Measure reset control  
***kCT32B0\_RST\_SHIFT\_RSTn*** CT32B0 reset control  
***kCT32B1\_RST\_SHIFT\_RSTn*** CT32B1 reset control  
***kCT32B2\_RST\_SHIFT\_RSTn*** CT32B3 reset control  
***kCT32B3\_RST\_SHIFT\_RSTn*** CT32B4 reset control  
***kCT32B4\_RST\_SHIFT\_RSTn*** CT32B4 reset control  
***kmrt\_RST\_SHIFT\_RSTn*** Multi-rate timer (MRT) reset control  
***kPINT\_RST\_SHIFT\_RSTn*** GPIO\_INT reset control  
***kINPUTMUX\_RST\_SHIFT\_RSTn*** PMUX reset control

## 10.4 Function Documentation

### 10.4.1 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.
-------------------	--

#### 10.4.2 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.
-------------------	---

#### 10.4.3 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.
-------------------	--

#### 10.4.4 static void RESET\_ReleasePeripheralReset ( *reset\_ip\_name\_t peripheral* ) [inline], [static]

Release peripheral module.

Parameters

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

# Chapter 11

## ACOMP: Analog Comparator

The MCUXpresso SDK provides a driver for the Analog Converter (ACOMP) module of MCUXpresso SDK devices.

The ACOMP driver can be divided into 4 function groups.

### ACOMP Control Interfaces

The functions in this group can be used to control the ACOMP module. User can invoke ACOMP\_GetDefaultConfig() function to configure the default setting, then invoking ACOMP\_Init() to initialize the module. All ACOMP feature(except software reset comparators and clock divider) can be enabled by invoking ACOMP\_Init() function.

### ACOMP Result Interface

There is only one function in this function groups, it is ACOMP\_GetResult(). User can get the result of comparator by invoking this function.

### ACOMP Interrupt Control Interfaces

The functions in this group can be used to enable/disable interrupts.

### ACOMP Status Flag Interfaces

The ACOMP\_GetStatusFlags() function in this group can return all status flags. The ACOMP\_ClearStatusFlags() can be used to clear status flags, but please note that no all status flags can be cleared by software.

# Chapter 12

## ADC: Analog Digital Converter

### 12.1 Overview

The MCUXpresso SDK provides a driver for the Analog Digital converter (ADC) module of MCUXpresso SDK devices.

### Function Groups

The ADC driver can be divided into 7 function groups.

#### ADC Basic Control Interfaces

The functions in this group can be used to configure the whole ADC module. To initialize the ADC module, the [ADC\\_Init\(\)](#) is provided. Before invoking [ADC\\_Init\(\)](#) function, user can use [ADC\\_GetDefaultConfig\(\)](#) function to get default configurations.

#### ADC Calibration Control Interfaces

The ADC module supports both automatic calibration and user defined calibration. To do auto calibration, the function [ADC\\_DoAutoCalibration\(\)](#) is provided, to do user defined calibration, the function [ADC\\_DoUserCalibration\(\)](#) is provided.

#### ADC Temperature Sensor Control Interfaces

The functions in this group can be used to control temperature sensor, including enabling temperature sensor and setting temperature sensor mode.

#### ADC Audio Control Interfaces

This function group contains 3 functions to control audio. [ADC\\_EnableAudio\(\)](#) can be used to enable/disable audio PGA and decimation rate select. [ADC\\_SetAudioPGAVoltageGain\(\)](#) can be used to set audio PGA voltage gain. [ADC\\_ConfigAudioVoiceLevel\(\)](#) can be used to configure audio voice level.

## ADC Conversion Related Interfaces

Conversion related function are placed in this function group. If the trigger mode is set as software trigger, invoking `ADC_DoSoftwareTrigger()` to trigger the scan. `ADC_SetScanChannel()` function can be used to set each channel's mux source. After conversion completed, `ADC_GetConversionResult()` can be used to get 32-bit width packed ADC conversion result.

## ADC Interrupt Control Interfaces

The functions in this group can be used to enable/disable interrupts.

## ADC Status Control Interfaces

The `ACOMP_GetStatusFlags()` function in this group can return all status flags. The `ACOMP_ClearStatusFlags()` can be used to clear status flags, but please note that no all status flags can be cleared by software.

## Data Structures

- struct `adc_config_t`  
*The structure of adc options, including clock divider, power mode, and so on. [More...](#)*

## Macros

- #define `FSL_ADC_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`  
*ADC driver version.*

## Enumerations

- enum `_adc_interrupt_enable` {
   
`kADC_DataReadyInterruptEnable` = `ADC_ADC_REG_IMR_RDY_MASK_MASK`,
   
`kADC_GainSaturationInterruptEnable` = `ADC_ADC_REG_IMR_GAINSAT_MASK_MASK`,
   
`kADC_OffsetSaturationInterruptEnable` = `ADC_ADC_REG_IMR_OFFSETSAT_MASK_MASK`,
   
`kADC_NegativeSaturationInterruptEnable` = `ADC_ADC_REG_IMR_DATASAT_NEG_MASK_MASK`,
   
`kADC_PositiveSaturationInterruptEnable` = `ADC_ADC_REG_IMR_DATASAT_POS_MASK_MASK`,
   
`kADC_FifoOverrunInterruptEnable` = `ADC_ADC_REG_IMR_FIFO_OVERFLOW_MASK_MASK`,
   
`kADC_FifoUnderrunInterruptEnable` = `ADC_ADC_REG_IMR_FIFO_UNDERFLOW_MASK_MASK`
}

*The enumeration of interrupts, this enumeration can be used to enable/disable interrupts.*

- enum \_adc\_status\_flags {  
    kADC\_DataReadyInterruptFlag = 1UL << 0UL,  
    kADC\_GainSaturationInterruptFlag = 1UL << 1UL,  
    kADC\_OffsetSaturationInterruptFlag = 1UL << 2UL,  
    kADC\_NegativeSaturationInterruptFlag = 1UL << 3UL,  
    kADC\_PositiveSaturationInterruptFlag = 1UL << 4UL,  
    kADC\_FifoOverrunInterruptFlag = 1UL << 5UL,  
    kADC\_FifoUnderrunInterruptFlag = 1UL << 6UL,  
    kADC\_DataReadyRawFlag = 1UL << 7UL,  
    kADC\_GainSaturationRawFlag = 1UL << 8UL,  
    kADC\_OffsetSaturationRawFlag = 1UL << 9UL,  
    kADC\_NegativeSaturationRawFlag = 1UL << 10UL,  
    kADC\_PositiveSaturationRawFlag = 1UL << 11UL,  
    kADC\_FifoOverrunRawFlag = 1UL << 12UL,  
    kADC\_FifoUnderrunRawFlag = 1UL << 13UL,  
    kADC\_ActiveStatusFlag = 1UL << 14UL,  
    kADC\_FIFONotEmptyStatusFlag = 1UL << 15UL,  
    kADC\_FifoFullStatusFlag = 1UL << 16UL }

*The enumeration of adc status flags, including interrupt flags, raw flags, and so on.*

- enum adc\_clock\_divider\_t {

```

kADC_ClockDivider1 = 0U,
kADC_ClockDivider2 = 1U,
kADC_ClockDivider3 = 2U,
kADC_ClockDivider4 = 3U,
kADC_ClockDivider5 = 4U,
kADC_ClockDivider6 = 5U,
kADC_ClockDivider7 = 6U,
kADC_ClockDivider8 = 7U,
kADC_ClockDivider9 = 8U,
kADC_ClockDivider10 = 9U,
kADC_ClockDivider11 = 10U,
kADC_ClockDivider12 = 11U,
kADC_ClockDivider13 = 12U,
kADC_ClockDivider14 = 13U,
kADC_ClockDivider15 = 14U,
kADC_ClockDivider16 = 15U,
kADC_ClockDivider17 = 16U,
kADC_ClockDivider18 = 17U,
kADC_ClockDivider19 = 18U,
kADC_ClockDivider20 = 19U,
kADC_ClockDivider21 = 20U,
kADC_ClockDivider22 = 21U,
kADC_ClockDivider23 = 22U,
kADC_ClockDivider24 = 23U,
kADC_ClockDivider25 = 24U,
kADC_ClockDivider26 = 25U,
kADC_ClockDivider27 = 26U,
kADC_ClockDivider28 = 27U,
kADC_ClockDivider29 = 28U,
kADC_ClockDivider30 = 29U,
kADC_ClockDivider31 = 30U,
kADC_ClockDivider32 = 31U }

```

*ADC clock divider ratio type.*

- enum `adc_analog_portion_power_mode_t` {
   
  `kADC_PowerModeFullBiasingCurrent` = 0U,
   
  `kADC_PowerModeHalfBiasingCurrent` }
- ADC analog portion low-power mode selection.*
- enum `adc_resolution_t` {
   
  `kADC_Resolution12Bit` = 0U,
   
  `kADC_Resolution14Bit` = 1U,
   
  `kADC_Resolution16Bit` = 2U,
   
  `kADC_Resolution16BitAudio` = 3U }
- ADC resolution type.*
- enum `adc_warm_up_time_t` {

```

kADC_WarmUpTime1us = 0U,
kADC_WarmUpTime2us,
kADC_WarmUpTime3us,
kADC_WarmUpTime4us,
kADC_WarmUpTime5us,
kADC_WarmUpTime6us,
kADC_WarmUpTime7us,
kADC_WarmUpTime8us,
kADC_WarmUpTime9us,
kADC_WarmUpTime10us,
kADC_WarmUpTime11us,
kADC_WarmUpTime12us,
kADC_WarmUpTime13us,
kADC_WarmUpTime14us,
kADC_WarmUpTime15us,
kADC_WarmUpTime16us,
kADC_WarmUpTime17us,
kADC_WarmUpTime18us,
kADC_WarmUpTime19us,
kADC_WarmUpTime20us,
kADC_WarmUpTime21us,
kADC_WarmUpTime22us,
kADC_WarmUpTime23us,
kADC_WarmUpTime24us,
kADC_WarmUpTime25us,
kADC_WarmUpTime26us,
kADC_WarmUpTime27us,
kADC_WarmUpTime28us,
kADC_WarmUpTime29us,
kADC_WarmUpTime30us,
kADC_WarmUpTime31us,
kADC_WarmUpTime32us,
kADC_WarmUpStateBypass = 0x20U }

```

*The enumeration of adc warm up time, the ADC warm-up state can also bypassed.*

- enum `adc_vref_source_t` {
 

```

kADC_Vref1P8V = 0U,
kADC_Vref1P2V = 1U,
kADC_VrefExternal = 2U,
kADC_VrefInternal1P2V = 3U }

```

*ADC voltage reference source type.*
- enum `adc_input_mode_t` {
 

```

kADC_InputSingleEnded = 0U,
kADC_InputDifferential = 1U }

```

*ADC input mode type.*
- enum `adc_conversion_mode_t` {

- ```

kADC_ConversionOneShot = 0U,
kADC_ConversionContinuous = 1U }

    ADC conversion mode type.
• enum adc_scan_length_t {
    kADC_ScanLength_1 = 0U,
    kADC_ScanLength_2 = 1U,
    kADC_ScanLength_3 = 2U,
    kADC_ScanLength_4 = 3U,
    kADC_ScanLength_5 = 4U,
    kADC_ScanLength_6 = 5U,
    kADC_ScanLength_7 = 6U,
    kADC_ScanLength_8 = 7U,
    kADC_ScanLength_9 = 8U,
    kADC_ScanLength_10 = 9U,
    kADC_ScanLength_11 = 10U,
    kADC_ScanLength_12 = 11U,
    kADC_ScanLength_13 = 12U,
    kADC_ScanLength_14 = 13U,
    kADC_ScanLength_15 = 14U,
    kADC_ScanLength_16 = 15U }

    ADC scan length type.
• enum adc_average_length_t {
    kADC_AverageNone = 0U,
    kADC_Average2 = 1U,
    kADC_Average4 = 2U,
    kADC_Average8 = 3U,
    kADC_Average16 = 4U }

    ADC average length type.
• enum adc_input_gain_t {
    kADC_InputGain0P5 = 0U,
    kADC_InputGain1 = 1U,
    kADC_InputGain2 = 2U }

    ADC input buffer gain type.
• enum adc_result_width_t {
    kADC_ResultWidth16 = 0U,
    kADC_ResultWidth32 = 1U }

    ADC result width type.
• enum adc_fifo_threshold_t {
    kADC_FifoThresholdData1 = 0U,
    kADC_FifoThresholdData4,
    kADC_FifoThresholdData8,
    kADC_FifoThresholdData16 }

    The threshold of FIFO.
• enum adc_calibration_ref_t {
    kADC_CalibrationVrefInternal = 0,
    kADC_CalibrationVrefExternal = 1 }

```

*ADC calibration voltage reference type.*

- enum `adc_scan_channel_t` {
   
kADC\_ScanChannel0 = 0U,  
 kADC\_ScanChannel1 = 1U,  
 kADC\_ScanChannel2 = 2U,  
 kADC\_ScanChannel3 = 3U,  
 kADC\_ScanChannel4 = 4U,  
 kADC\_ScanChannel5 = 5U,  
 kADC\_ScanChannel6 = 6U,  
 kADC\_ScanChannel7 = 7U,  
 kADC\_ScanChannel8 = 8U,  
 kADC\_ScanChannel9 = 9U,  
 kADC\_ScanChannel10 = 10U,  
 kADC\_ScanChannel11 = 11U,  
 kADC\_ScanChannel12 = 12U,  
 kADC\_ScanChannel13 = 13U,  
 kADC\_ScanChannel14 = 14U,  
 kADC\_ScanChannel15 = 15U }

*ADC scan channel type.*

- enum `adc_channel_source_t` {
   
kADC\_CH0 = 0U,  
 kADC\_CH1 = 1U,  
 kADC\_CH2 = 2U,  
 kADC\_CH3 = 3U,  
 kADC\_CH4 = 4U,  
 kADC\_CH5 = 5U,  
 kADC\_CH6 = 6U,  
 kADC\_CH7 = 7U,  
 kADC\_VBATS = 8U,  
 kADC\_VREF = 9U,  
 kADC\_DACA = 10U,  
 kADC\_DACB = 11U,  
 kADC\_VSSA = 12U,  
 kADC\_CH0\_CH1 = 0U,  
 kADC\_CH2\_CH3 = 1U,  
 kADC\_CH4\_CH5 = 2U,  
 kADC\_CH6\_CH7 = 3U,  
 kADC\_DACA\_DACB = 4U }

*ADC channel source type.*

- enum `adc_temperature_sensor_mode_t` { `kADC_TSensorExternal` }

*Temperature sensor mode, including internal diode mode and external diode mode.*

- enum `adc_audio_pga_voltage_gain_t` {
   
kADC\_AudioGain4 = 0U,  
 kADC\_AudioGain8 = 1U,  
 kADC\_AudioGain16 = 2U,  
 kADC\_AudioGain32 = 3U }

- enum `adc_audio_voice_level_t` {
   
    `kADC_VoiceLevel0` = 0U,  
     `kADC_VoiceLevel1` = 1U,  
     `kADC_VoiceLevel2` = 2U,  
     `kADC_VoiceLevel3` = 3U }
   
    *ADC audio voice level selection.*

## ADC Basic Control Interfaces

- void `ADC_Init` (ADC\_Type \*base, const `adc_config_t` \*config)  
*Initialize ADC module, including clock divider, power mode, and so on.*
- void `ADC_GetDefaultConfig` (`adc_config_t` \*config)  
*Get default configuration.*
- void `ADC_Deinit` (ADC\_Type \*base)  
*De-initialize the ADC module.*
- static void `ADC_DoSoftwareReset` (ADC\_Type \*base)  
*Reset the whole ADC block.*
- static void `ADC_SelectAnalogPortionPowerMode` (ADC\_Type \*base, `adc_analog_portion_power_mode_t` powerMode)  
*Select ADC analog portion power mode.*

## ADC Calibration Control Interfaces

- `status_t ADC_DoAutoCalibration` (ADC\_Type \*base, `adc_calibration_ref_t` calVref)  
*Do automatic calibration measurement.*
- static void `ADC_GetAutoCalibrationData` (ADC\_Type \*base, `uint16_t` \*offsetCal, `uint16_t` \*gainCal)  
*Get the ADC automatic calibration data.*
- static void `ADC_ResetAutoCalibrationData` (ADC\_Type \*base)  
*Reset the automatic calibration data.*
- static void `ADC_DoUserCalibration` (ADC\_Type \*base, `uint16_t` offsetCal, `uint16_t` gainCal)  
*Do user defined calibration.*

## ADC Temperature Sensor Control Interfaces

- static void `ADC_EnableTemperatureSensor` (ADC\_Type \*base, bool enable)  
*Enable/disable temperature sensor.*
- static void `ADC_SetTemperatureSensorMode` (ADC\_Type \*base, `adc_temperature_sensor_mode_t` tSensorMode)  
*Set temperature sensor mode, available selections are internal diode mode and external diode mode.*

## ADC Audio Control Interfaces

- static void `ADC_EnableAudio` (ADC\_Type \*base, bool enable)  
*Enable/disable audio PGA and decimation rate select.*
- static void `ADC_SetAudioPGAVoltageGain` (ADC\_Type \*base, `adc_audio_pga_voltage_gain_t` voltageGain)  
*Set audio PGA voltage gain.*

- void [ADC\\_ConfigAudioVoiceLevel](#) (ADC\_Type \*base, bool enableDetect, [adc\\_audio\\_voice\\_level\\_t](#) voiceLevel)  
*Configure audio voice level.*

## ADC Conversion Related Interfaces

- void [ADC\\_SetScanChannel](#) (ADC\_Type \*base, [adc\\_scan\\_channel\\_t](#) scanChannel, [adc\\_channel\\_source\\_t](#) channelSource)  
*Set scan channel mux source.*
- static void [ADC\\_DoSoftwareTrigger](#) (ADC\_Type \*base)  
*If trigger mode is selected as software trigger, invoking this function to start conversion.*
- static void [ADC\\_StopConversion](#) (ADC\_Type \*base)  
*Invoke this function to stop conversion.*
- static uint32\_t [ADC\\_GetConversionResult](#) (ADC\_Type \*base)  
*Get the 32-bit width packed ADC conversion result.*
- static uint8\_t [ADC\\_GetFifoDataCount](#) (ADC\_Type \*base)  
*Get the ADC FIFO data count.*

## ADC Interrupt Control Interfaces

- static void [ADC\\_EnableInterrupts](#) (ADC\_Type \*base, uint32\_t interruptMask)  
*Enable interrupts, such as conversion data ready interrupt, gain correction saturation interrupt, FIFO under run interrupt, and so on.*
- static void [ADC\\_DisableInterrupts](#) (ADC\_Type \*base, uint32\_t interruptMask)  
*Disable interrupts, such as conversion data ready interrupt, gain correction saturation interrupt, FIFO under run interrupt, and so on.*

## ADC Status Control Interfaces

- uint32\_t [ADC\\_GetStatusFlags](#) (ADC\_Type \*base)  
*Get status flags, including interrupt flags, raw flags, and so on.*
- static void [ADC\\_ClearStatusFlags](#) (ADC\_Type \*base, uint32\_t statusFlagsMask)  
*Clear status flags.*

## 12.2 Data Structure Documentation

### 12.2.1 struct adc\_config\_t

#### Data Fields

- [adc\\_clock\\_divider\\_t](#) clockDivider: 5U  
*Analog 64M clock division ratio, please refer to [adc\\_clock\\_divider\\_t](#).*
- [adc\\_resolution\\_t](#) resolution: 2U  
*Configure ADC resolution, please refer to [adc\\_resolution\\_t](#).*
- [adc\\_warm\\_up\\_time\\_t](#) warmupTime: 6U  
*Configure warm-up time.*
- [adc\\_vref\\_source\\_t](#) vrefSource: 2U  
*Configure voltage reference source, please refer to [adc\\_vref\\_source\\_t](#).*
- [adc\\_input\\_mode\\_t](#) inputMode: 1U

- Configure input mode, such as `kADC_InputSingleEnded` or `kADC_InputDifferential`.
- `adc_conversion_mode_t conversionMode`: 1U  
Configure conversion mode, such as `kADC_ConversionOneShot` or `kADC_ConversionContinuous`.
- `adc_scan_length_t scanLength`: 4U  
Configure the length of scan, please refer to `adc_scan_length_t`.
- `adc_average_length_t averageLength`: 3U  
Configure hardware average number, please refer to `adc_average_length_t`.
- `adc_trigger_source_t triggerSource`: 3U  
Configure trigger source, the trigger source can be divided into hardware trigger and software trigger, please refer to `adc_trigger_source_t` for details.
- `adc_input_gain_t inputGain`: 2U  
Configure ADC input buffer gain, please refer to `adc_input_gain_t`.
- `bool enableInputGainBuffer`: 1U  
Enable/Disable input gain buffer.
- `bool enableInputBufferChop`: 1U  
Enable/Disable input buffer chopper.
- `bool enableChop`: 1U  
Enable/Disable the ADC chopper.
- `adc_result_width_t resultWidth`: 1U  
Select result FIFO data packed format, please refer to `adc_result_width_t`.
- `adc_fifo_threshold_t fifoThreshold`: 2U  
Configure FIFO threshold, please refer to `adc_fifo_threshold_t`.
- `bool enableDMA`: 1U  
Enable/Disable DMA request.
- `bool enableADC`: 1U  
Enable/Disable ADC module.

## Field Documentation

- (1) `adc_clock_divider_t adc_config_t::clockDivider`
- (2) `adc_resolution_t adc_config_t::resolution`
- (3) `adc_warm_up_time_t adc_config_t::warmupTime`
- (4) `adc_vref_source_t adc_config_t::vrefSource`
- (5) `adc_input_mode_t adc_config_t::inputMode`
- (6) `adc_conversion_mode_t adc_config_t::conversionMode`
- (7) `adc_scan_length_t adc_config_t::scanLength`
- (8) `adc_trigger_source_t adc_config_t::triggerSource`
- (9) `adc_input_gain_t adc_config_t::inputGain`
- (10) `bool adc_config_t::enableInputGainBuffer`
  - **true** Enable input gain buffer.
  - **false** Disable input gain buffer.

(11) **bool adc\_config\_t::enableInputBufferChop**

- **true** Enable input buffer chopper;
- **false** Disable input buffer chopper.

(12) **bool adc\_config\_t::enableChop**

- **true** Enable the chopper;
- **false** Disable the chopper.

(13) **adc\_result\_width\_t adc\_config\_t::resultWidth**(14) **adc\_fifo\_threshold\_t adc\_config\_t::fifoThreshold**(15) **bool adc\_config\_t::enableDMA**

- **true** Enable DMA request.
- **false** Disable DMA request.

(16) **bool adc\_config\_t::enableADC**

- **true** Enable ADC module.
- **false** Disable ADC module.

**12.3 Macro Definition Documentation****12.3.1 #define FSL\_ADC\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))**

Version 2.2.0.

**12.4 Enumeration Type Documentation****12.4.1 enum \_adc\_interrupt\_enable**

Enumerator

*kADC\_DataReadyInterruptEnable* Conversion data ready interrupt.

*kADC\_GainSaturationInterruptEnable* Gain correction saturation interrupt.

*kADC\_OffsetSaturationInterruptEnable* Offset correction saturation interrupt enable.

*kADC\_NegativeSaturationInterruptEnable* ADC data negative side saturation interrupt enable.

*kADC\_PositiveSaturationInterruptEnable* ADC data positive side saturation interrupt enable.

*kADC\_FifoOverrunInterruptEnable* FIFO overrun interrupt enable.

*kADC\_FifoUnderrunInterruptEnable* FIFO underrun interrupt enable.

**12.4.2 enum \_adc\_status\_flags**

## Note

The raw flags will be captured regardless the interrupt mask. Both interrupt flags and raw flags can be cleared.

## Enumerator

- kADC\_DataReadyInterruptFlag*** Conversion Data Ready interrupt flag.
- kADC\_GainSaturationInterruptFlag*** Gain correction saturation interrupt flag.
- kADC\_OffsetSaturationInterruptFlag*** Offset correction saturation interrupt flag.
- kADC\_NegativeSaturationInterruptFlag*** ADC data negative side saturation interrupt flag.
- kADC\_PositiveSaturationInterruptFlag*** ADC data positive side saturation interrupt flag.
- kADC\_FifoOverrunInterruptFlag*** FIFO overrun interrupt flag.
- kADC\_FifoUnderrunInterruptFlag*** FIFO underrun interrupt flag.
- kADC\_DataReadyRawFlag*** Conversion data ready raw flag, this flag will be captured regardless the interrupt mask.
- kADC\_GainSaturationRawFlag*** Gain correction saturation raw flag, this flag will be captured regardless the interrupt mask.
- kADC\_OffsetSaturationRawFlag*** Offset correction saturation raw flag, this flag will be captured regardless the interrupt mask.
- kADC\_NegativeSaturationRawFlag*** ADC data negative side saturation raw flag, this flag will be captured regardless the interrupt mask.
- kADC\_PositiveSaturationRawFlag*** ADC data positive side saturation raw flag, this flag will be captured regardless the interrupt mask.
- kADC\_FifoOverrunRawFlag*** FIFO overrun raw flag, this flag will be captured regardless the interrupt mask.
- kADC\_FifoUnderrunRawFlag*** FIFO underrun interrupt mask, this flag will be captured regardless the interrupt mask.
- kADC\_ActiveStatusFlag*** ADC conversion active status flag.
- kADC\_FIFONotEmptyStatusFlag*** FIFO not empty status flag.
- kADC\_FifoFullStatusFlag*** FIFO full status flag.

**12.4.3 enum adc\_clock\_divider\_t**

## Enumerator

- kADC\_ClockDivider1*** Clock divider ratio is 1.
- kADC\_ClockDivider2*** Clock divider ratio is 2.
- kADC\_ClockDivider3*** Clock divider ratio is 3.
- kADC\_ClockDivider4*** Clock divider ratio is 4.
- kADC\_ClockDivider5*** Clock divider ratio is 5.
- kADC\_ClockDivider6*** Clock divider ratio is 6.
- kADC\_ClockDivider7*** Clock divider ratio is 7.
- kADC\_ClockDivider8*** Clock divider ratio is 8.
- kADC\_ClockDivider9*** Clock divider ratio is 9.

- kADC\_ClockDivider10* Clock divider ratio is 10.
- kADC\_ClockDivider11* Clock divider ratio is 11.
- kADC\_ClockDivider12* Clock divider ratio is 12.
- kADC\_ClockDivider13* Clock divider ratio is 13.
- kADC\_ClockDivider14* Clock divider ratio is 14.
- kADC\_ClockDivider15* Clock divider ratio is 15.
- kADC\_ClockDivider16* Clock divider ratio is 16.
- kADC\_ClockDivider17* Clock divider ratio is 17.
- kADC\_ClockDivider18* Clock divider ratio is 18.
- kADC\_ClockDivider19* Clock divider ratio is 19.
- kADC\_ClockDivider20* Clock divider ratio is 20.
- kADC\_ClockDivider21* Clock divider ratio is 21.
- kADC\_ClockDivider22* Clock divider ratio is 22.
- kADC\_ClockDivider23* Clock divider ratio is 23.
- kADC\_ClockDivider24* Clock divider ratio is 24.
- kADC\_ClockDivider25* Clock divider ratio is 25.
- kADC\_ClockDivider26* Clock divider ratio is 26.
- kADC\_ClockDivider27* Clock divider ratio is 27.
- kADC\_ClockDivider28* Clock divider ratio is 28.
- kADC\_ClockDivider29* Clock divider ratio is 29.
- kADC\_ClockDivider30* Clock divider ratio is 30.
- kADC\_ClockDivider31* Clock divider ratio is 31.
- kADC\_ClockDivider32* Clock divider ratio is 32.

#### 12.4.4 enum adc\_analog\_portion\_power\_mode\_t

Enumerator

- kADC\_PowerModeFullBiasingCurrent* Full biasing current.
- kADC\_PowerModeHalfBiasingCurrent* Half biasing current.

#### 12.4.5 enum adc\_resolution\_t

Enumerator

- kADC\_Resolution12Bit* 12-bit resolution
- kADC\_Resolution14Bit* 14-bit resolution
- kADC\_Resolution16Bit* 16-bit resolution
- kADC\_Resolution16BitAudio* 16-bit resolution for audio application

## 12.4.6 enum adc\_warm\_up\_time\_t

Enumerator

|                               |                             |
|-------------------------------|-----------------------------|
| <i>kADC_WarmUpTime1us</i>     | ADC warm-up time is 1 us.   |
| <i>kADC_WarmUpTime2us</i>     | ADC warm-up time is 2 us.   |
| <i>kADC_WarmUpTime3us</i>     | ADC warm-up time is 3 us.   |
| <i>kADC_WarmUpTime4us</i>     | ADC warm-up time is 4 us.   |
| <i>kADC_WarmUpTime5us</i>     | ADC warm-up time is 5 us.   |
| <i>kADC_WarmUpTime6us</i>     | ADC warm-up time is 6 us.   |
| <i>kADC_WarmUpTime7us</i>     | ADC warm-up time is 7 us.   |
| <i>kADC_WarmUpTime8us</i>     | ADC warm-up time is 8 us.   |
| <i>kADC_WarmUpTime9us</i>     | ADC warm-up time is 9 us.   |
| <i>kADC_WarmUpTime10us</i>    | ADC warm-up time is 10 us.  |
| <i>kADC_WarmUpTime11us</i>    | ADC warm-up time is 11 us.  |
| <i>kADC_WarmUpTime12us</i>    | ADC warm-up time is 12 us.  |
| <i>kADC_WarmUpTime13us</i>    | ADC warm-up time is 13 us.  |
| <i>kADC_WarmUpTime14us</i>    | ADC warm-up time is 14 us.  |
| <i>kADC_WarmUpTime15us</i>    | ADC warm-up time is 15 us.  |
| <i>kADC_WarmUpTime16us</i>    | ADC warm-up time is 16 us.  |
| <i>kADC_WarmUpTime17us</i>    | ADC warm-up time is 17 us.  |
| <i>kADC_WarmUpTime18us</i>    | ADC warm-up time is 18 us.  |
| <i>kADC_WarmUpTime19us</i>    | ADC warm-up time is 19 us.  |
| <i>kADC_WarmUpTime20us</i>    | ADC warm-up time is 20 us.  |
| <i>kADC_WarmUpTime21us</i>    | ADC warm-up time is 21 us.  |
| <i>kADC_WarmUpTime22us</i>    | ADC warm-up time is 22 us.  |
| <i>kADC_WarmUpTime23us</i>    | ADC warm-up time is 23 us.  |
| <i>kADC_WarmUpTime24us</i>    | ADC warm-up time is 24 us.  |
| <i>kADC_WarmUpTime25us</i>    | ADC warm-up time is 25 us.  |
| <i>kADC_WarmUpTime26us</i>    | ADC warm-up time is 26 us.  |
| <i>kADC_WarmUpTime27us</i>    | ADC warm-up time is 27 us.  |
| <i>kADC_WarmUpTime28us</i>    | ADC warm-up time is 28 us.  |
| <i>kADC_WarmUpTime29us</i>    | ADC warm-up time is 29 us.  |
| <i>kADC_WarmUpTime30us</i>    | ADC warm-up time is 30 us.  |
| <i>kADC_WarmUpTime31us</i>    | ADC warm-up time is 31 us.  |
| <i>kADC_WarmUpTime32us</i>    | ADC warm-up time is 32 us.  |
| <i>kADC_WarmUpStateBypass</i> | ADC warm-up state bypassed. |

## 12.4.7 enum adc\_vref\_source\_t

Enumerator

|                      |                          |
|----------------------|--------------------------|
| <i>kADC_Vref1P8V</i> | Internal 1.8V reference. |
| <i>kADC_Vref1P2V</i> | Internal 1.2V reference. |

*kADC\_VrefExternal* External single-ended reference though ADC\_CH3.

*kADC\_VrefInternal1P2V* Internal 1.2V reference with cap filter though ADC\_CH3.

#### 12.4.8 enum adc\_input\_mode\_t

Enumerator

*kADC\_InputSingleEnded* Single-ended mode.

*kADC\_InputDifferential* Differential mode.

#### 12.4.9 enum adc\_conversion\_mode\_t

Enumerator

*kADC\_ConversionOneShot* One shot mode.

*kADC\_ConversionContinuous* Continuous mode.

#### 12.4.10 enum adc\_scan\_length\_t

Enumerator

*kADC\_ScanLength\_1* Scan length is 1.

*kADC\_ScanLength\_2* Scan length is 2.

*kADC\_ScanLength\_3* Scan length is 3.

*kADC\_ScanLength\_4* Scan length is 4.

*kADC\_ScanLength\_5* Scan length is 5.

*kADC\_ScanLength\_6* Scan length is 6.

*kADC\_ScanLength\_7* Scan length is 7.

*kADC\_ScanLength\_8* Scan length is 8.

*kADC\_ScanLength\_9* Scan length is 9.

*kADC\_ScanLength\_10* Scan length is 10.

*kADC\_ScanLength\_11* Scan length is 11.

*kADC\_ScanLength\_12* Scan length is 12.

*kADC\_ScanLength\_13* Scan length is 13.

*kADC\_ScanLength\_14* Scan length is 14.

*kADC\_ScanLength\_15* Scan length is 15.

*kADC\_ScanLength\_16* Scan length is 16.

#### 12.4.11 enum adc\_average\_length\_t

Enumerator

*kADC\_AverageNone* Average length: no average.

*kADC\_Average2* Average length: 2.

*kADC\_Average4* Average length: 4.

*kADC\_Average8* Average length: 8.

*kADC\_Average16* Average length: 16.

#### 12.4.12 enum adc\_input\_gain\_t

Enumerator

*kADC\_InputGain0P5* Input buffer gain is 0.5.

*kADC\_InputGain1* Input buffer gain is 1.

*kADC\_InputGain2* Input buffer gain is 2.

#### 12.4.13 enum adc\_result\_width\_t

Enumerator

*kADC\_ResultWidth16* 16-bit final result buffer width

*kADC\_ResultWidth32* 32-bit final result buffer width

#### 12.4.14 enum adc\_fifo\_threshold\_t

Enumerator

*kADC\_FifoThresholdData1* FIFO Threshold is 1 data.

*kADC\_FifoThresholdData4* FIFO Threshold is 4 data.

*kADC\_FifoThresholdData8* FIFO Threshold is 8 data.

*kADC\_FifoThresholdData16* FIFO Threshold is 16 data.

#### 12.4.15 enum adc\_calibration\_ref\_t

Enumerator

*kADC\_CalibrationVrefInternal* Internal vref as input for calibration.

*kADC\_CalibrationVrefExternal* External vref as input for calibration.

**12.4.16 enum adc\_scan\_channel\_t**

Enumerator

- kADC\_ScanChannel0*** Scan channel 0.
- kADC\_ScanChannel1*** Scan channel 1.
- kADC\_ScanChannel2*** Scan channel 2.
- kADC\_ScanChannel3*** Scan channel 3.
- kADC\_ScanChannel4*** Scan channel 4.
- kADC\_ScanChannel5*** Scan channel 5.
- kADC\_ScanChannel6*** Scan channel 6.
- kADC\_ScanChannel7*** Scan channel 7.
- kADC\_ScanChannel8*** Scan channel 8.
- kADC\_ScanChannel9*** Scan channel 9.
- kADC\_ScanChannel10*** Scan channel 10.
- kADC\_ScanChannel11*** Scan channel 11.
- kADC\_ScanChannel12*** Scan channel 12.
- kADC\_ScanChannel13*** Scan channel 13.
- kADC\_ScanChannel14*** Scan channel 14.
- kADC\_ScanChannel15*** Scan channel 15.

**12.4.17 enum adc\_channel\_source\_t**

Enumerator

- kADC\_CH0*** Single-ended mode, channel[0] and vssa.
- kADC\_CH1*** Single-ended mode, channel[1] and vssa.
- kADC\_CH2*** Single-ended mode, channel[2] and vssa.
- kADC\_CH3*** Single-ended mode, channel[3] and vssa.
- kADC\_CH4*** Single-ended mode, channel[4] and vssa.
- kADC\_CH5*** Single-ended mode, channel[5] and vssa.
- kADC\_CH6*** Single-ended mode, channel[6] and vssa.
- kADC\_CH7*** Single-ended mode, channel[7] and vssa.
- kADC\_VBATS*** Single-ended mode, vbat\_s and vssa.
- kADC\_VREF*** Single-ended mode, vref\_12 and vssa.
- kADC\_DACA*** Single-ended mode, daca and vssa.
- kADC\_DACB*** Single-ended mode, dacb and vssa.
- kADC\_VSSA*** Single-ended mode, vssa and vssa.
- kADC\_CH0\_CH1*** Differential mode, channel[0] and channel[1].
- kADC\_CH2\_CH3*** Differential mode, channel[2] and channel[3].
- kADC\_CH4\_CH5*** Differential mode, channel[4] and channel[5].
- kADC\_CH6\_CH7*** Differential mode, channel[6] and channel[7].
- kADC\_DACA\_DACB*** Differential mode, daca and dacb.

### 12.4.18 enum adc\_temperature\_sensor\_mode\_t

Enumerator

*kADC\_TSensorExternal* External diode mode.

### 12.4.19 enum adc\_audio\_pga\_voltage\_gain\_t

Enumerator

*kADC\_AudioGain4* Audio pga gain is 4.

*kADC\_AudioGain8* Audio pga gain is 8.

*kADC\_AudioGain16* Audio pga gain is 16.

*kADC\_AudioGain32* Audio pga gain is 32.

### 12.4.20 enum adc\_audio\_voice\_level\_t

Enumerator

*kADC\_VoiceLevel0* Input voice level >+255LSB or <-256LSB.

*kADC\_VoiceLevel1* Input voice level >+511LSB or <-512LSB.

*kADC\_VoiceLevel2* Input voice level >+1023LSB or <-1024LSB.

*kADC\_VoiceLevel3* Input voice level >+2047LSB or <-2048LSB.

## 12.5 Function Documentation

### 12.5.1 void ADC\_Init ( ADC\_Type \* *base*, const adc\_config\_t \* *config* )

Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                |
| <i>config</i> | The pointer to the structure <a href="#">adc_config_t</a> . |

### 12.5.2 void ADC\_GetDefaultConfig ( adc\_config\_t \* *config* )

```
* config->clockDivider = kADC_ClockDivider1;
* config->powerMode = kADC_PowerModeFullBiasingCurrent;
* config->resolution = kADC_Resolution12Bit;
* config->warmupTime = kADC_WarmUpTime16us;
* config->vrefSource = kADC_Vref1P2V;
* config->inputMode = kADC_InputSingleEnded;
* config->conversionMode = kADC_ConversionContinuous;
* config->scanLength = kADC_ScanLength_1;
* config->averageLength = kADC_AverageNone;
```

```

* config->triggerSource = kADC_TriggerSourceSoftware;
* config->inputGain = kADC_InputGain1;
* config->enableInputGainBuffer = false;
* config->resultWidth = kADC_ResultWidth16;
* config->fifoThreshold = kADC_FifoThresholdData1;
* config->enableDMA = false;
* config->enableADC = false;
*

```

Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>config</i> | The Pointer to the structure <a href="#">adc_config_t</a> . |
|---------------|-------------------------------------------------------------|

### 12.5.3 void ADC\_Deinit ( ADC\_Type \* *base* )

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

### 12.5.4 static void ADC\_DoSoftwareReset ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

### 12.5.5 static void ADC\_SelectAnalogPortionPowerMode ( ADC\_Type \* *base*, adc\_analog\_portion\_power\_mode\_t *powerMode* ) [inline], [static]

Parameters

|                  |                                                                                             |
|------------------|---------------------------------------------------------------------------------------------|
| <i>base</i>      | ADC peripheral base address.                                                                |
| <i>powerMode</i> | The power mode to be set, please refer to <a href="#">adc_analog_portion_power_mode_t</a> . |

### 12.5.6 status\_t ADC\_DoAutoCalibration ( ADC\_Type \* *base*, adc\_calibration\_ref\_t *calVref* )

## Note

After auto calibrate successful, user can invoke [ADC\\_GetAutoCalibrationData\(\)](#) to get self offset calibration value and self gain calibration value.

## Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | ADC peripheral base address.                                                                                         |
| <i>calVref</i> | The input reference channel for gain calibration, please refer to <a href="#">adc_calibration_ref_t</a> for details. |

## Return values

|                                        |                              |
|----------------------------------------|------------------------------|
| <a href="#"><i>kStatus_Success</i></a> | Auto calibrate successfully. |
| <a href="#"><i>kStatus_Fail</i></a>    | Auto calibrate failure.      |

**12.5.7 static void ADC\_GetAutoCalibrationData ( ADC\_Type \* *base*, uint16\_t \* *offsetCal*, uint16\_t \* *gainCal* ) [inline], [static]**

## Parameters

|                  |                                                                      |
|------------------|----------------------------------------------------------------------|
| <i>base</i>      | ADC peripheral base address.                                         |
| <i>offsetCal</i> | Self offset calibration data pointer, evaluate NULL if not required. |
| <i>gainCal</i>   | Self gain calibration data pointer, evaluate NULL if not required.   |

**12.5.8 static void ADC\_ResetAutoCalibrationData ( ADC\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

**12.5.9 static void ADC\_DoUserCalibration ( ADC\_Type \* *base*, uint16\_t *offsetCal*, uint16\_t *gainCal* ) [inline], [static]**

Parameters

|                  |                                       |
|------------------|---------------------------------------|
| <i>base</i>      | ADC peripheral base address.          |
| <i>offsetCal</i> | User defined offset calibration data. |
| <i>gainCal</i>   | User defined gain calibration date.   |

### 12.5.10 static void ADC\_EnableTemperatureSensor ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

Note

This function is useful only when the channel source is temperature sensor.

Parameters

|               |                                                                                                                                                                                           |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                                                                                                                                              |
| <i>enable</i> | Used to enable/disable temperature sensor. <ul style="list-style-type: none"> <li>• <b>true</b> Enable temperature sensor.</li> <li>• <b>false</b> Disable temperature sensor.</li> </ul> |

### 12.5.11 static void ADC\_SetTemperatureSensorMode ( ADC\_Type \* *base*, adc\_temperature\_sensor\_mode\_t *tSensorMode* ) [inline], [static]

Parameters

|                    |                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>        | ADC peripheral base address.                                                                           |
| <i>tSensorMode</i> | The temperature sensor mode to be set, please refer to <a href="#">adc_temperature_sensor_mode_t</a> . |

### 12.5.12 static void ADC\_EnableAudio ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                                                                                                                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                                                                                                                                                                                                    |
| <i>enable</i> | Used to enable/disable audio PGA and decimation rate select. <ul style="list-style-type: none"> <li>• <b>true</b> Enable audio PGA and decimation rate select.</li> <li>• <b>false</b> Disable audio PGA and decimation rate select.</li> </ul> |

**12.5.13 static void ADC\_SetAudioPGAVoltageGain ( ADC\_Type \* *base*, adc\_audio\_pga\_voltage\_gain\_t *voltageGain* ) [inline], [static]**

Parameters

|                    |                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i>        | ADC peripheral base address.                                                                        |
| <i>voltageGain</i> | The selected audio PGA voltage gain, please refer to <a href="#">adc_audio_pga_voltage_gain_t</a> . |

**12.5.14 void ADC\_ConfigAudioVoiceLevel ( ADC\_Type \* *base*, bool *enableDetect*, adc\_audio\_voice\_level\_t *voiceLevel* )**

Parameters

|                     |                                                                                                                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | ADC peripheral base address.                                                                                                                                                                       |
| <i>enableDetect</i> | Used to enable/disable voice level detection. <ul style="list-style-type: none"> <li>• <b>true</b> Enable voice level detection.</li> <li>• <b>false</b> Disable voice level detection.</li> </ul> |
| <i>voiceLevel</i>   | Selected voice level, please refer to <a href="#">adc_audio_voice_level_t</a> .                                                                                                                    |

**12.5.15 void ADC\_SetScanChannel ( ADC\_Type \* *base*, adc\_scan\_channel\_t *scanChannel*, adc\_channel\_source\_t *channelSource* )**

Parameters

|                      |                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | ADC peripheral base address.                                                                                        |
| <i>scanChannel</i>   | The selected channel, please refer to <a href="#">adc_scan_channel_t</a> for details.                               |
| <i>channelSource</i> | The mux source to be set to the selected channel, please refer to <a href="#">adc_channel_source_t</a> for details. |

### 12.5.16 static void ADC\_DoSoftwareTrigger ( ADC\_Type \* *base* ) [inline], [static]

Note

This API will also clear the FIFO.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

### 12.5.17 static void ADC\_StopConversion ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

### 12.5.18 static uint32\_t ADC\_GetConversionResult ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

Returns

32-bit width packed ADC conversion result.

### 12.5.19 static uint8\_t ADC\_GetFifoDataCount ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

Returns

ADC FIFO data count.

#### 12.5.20 static void ADC\_EnableInterrupts ( ADC\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Parameters

|                      |                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>          | ADC peripheral base address.                                                                       |
| <i>interruptMask</i> | The interrupts to be enabled, should be the OR'ed value of <a href="#">_adc_interrupt_enable</a> . |

#### 12.5.21 static void ADC\_DisableInterrupts ( ADC\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Parameters

|                      |                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i>          | ADC peripheral base address.                                                                        |
| <i>interruptMask</i> | The interrupts to be disabled, should be the OR'ed value of <a href="#">_adc_interrupt_enable</a> . |

#### 12.5.22 uint32\_t ADC\_GetStatusFlags ( ADC\_Type \* *base* )

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

Returns

The OR'ed value of ADC status flags, please refer to [\\_adc\\_status\\_flags](#) for details.

#### 12.5.23 static void ADC\_ClearStatusFlags ( ADC\_Type \* *base*, uint32\_t *statusFlagsMask* ) [inline], [static]

## Note

Only interrupt flags and raw flags can be cleared.

## Parameters

|                         |                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------|
| <i>base</i>             | ADC peripheral base address.                                                                                  |
| <i>statusFlags-Mask</i> | The OR'ed value of status flags to be cleared, please refer to <a href="#">_adc_status_flags</a> for details. |

# Chapter 13

## CACHE: CACHE Memory Controller

### 13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the CACHE Controller of MCUXpresso SDK devices.

The CACHE driver is created to help the user more easily operate the cache memory. The APIs for basic operations are including the following three levels: 1L. The local cache driver API. This level provides the caches controller drivers.

2L. The unified cache driver API. This level provides many APIs for unified cache driver APIs for combined L1 and L2 cache maintain operations. This is provided for SDK drivers (DMA, ENET, US-DHC, etc) which should do the cache maintenance in their transactional APIs. Because in this arch, there is no L2 cache so the unified cache driver API directly calls local driver APIs.

### 13.2 Function groups

#### 13.2.1 CACHE Operation

There are Enable/Disable APIs for cache control and cache maintenance operations as Invalidate/Clean/-CleanInvalidate by all and by address range.

### Data Structures

- struct `cache64_config_t`  
*CACHE64 configuration structure. More...*

### Macros

- #define `CACHE64_LINESIZE_BYTE` (FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTE)  
*cache line size.*
- #define `CACHE64_REGION_NUM` (3U)  
*cache region number.*
- #define `CACHE64_REGION_ALIGNMENT` (0x400U)  
*cache region alignment.*

### Enumerations

- enum `cache64_policy_t` {  
  kCACHE64\_PolicyNonCacheable = 0,  
  kCACHE64\_PolicyWriteThrough = 1,  
  kCACHE64\_PolicyWriteBack = 2 }  
*Level 2 cache controller way size.*

## Driver version

- #define **FSL\_CACHE\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 0, 8))  
*cache driver version.*

## cache control for cache64

- uint32\_t **CACHE64GetInstance** (CACHE64\_POLSEL\_Type \*base)  
*Returns an instance number given peripheral base address.*
- uint32\_t **CACHE64GetInstanceByAddr** (uint32\_t address)  
*brief Returns an instance number given physical memory address.*
- status\_t **CACHE64\_Init** (CACHE64\_POLSEL\_Type \*base, const **cache64\_config\_t** \*config)  
*Initializes an CACHE64 instance with the user configuration structure.*
- void **CACHE64\_GetDefaultConfig** (**cache64\_config\_t** \*config)  
*Gets the default configuration structure.*
- void **CACHE64\_EnableCache** (CACHE64\_CTRL\_Type \*base)  
*Enables the cache.*
- void **CACHE64\_DisableCache** (CACHE64\_CTRL\_Type \*base)  
*Disables the cache.*
- void **CACHE64\_InvalidateCache** (CACHE64\_CTRL\_Type \*base)  
*Invalidates the cache.*
- void **CACHE64\_InvalidateCacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Invalidates cache by range.*
- void **CACHE64\_CleanCache** (CACHE64\_CTRL\_Type \*base)  
*Cleans the cache.*
- void **CACHE64\_CleanCacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans cache by range.*
- void **CACHE64\_CleanInvalidateCache** (CACHE64\_CTRL\_Type \*base)  
*Cleans and invalidates the cache.*
- void **CACHE64\_CleanInvalidateCacheByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans and invalidate cache by range.*
- void **CACHE64\_EnableWriteBuffer** (CACHE64\_CTRL\_Type \*base, bool enable)  
*Enables/disables the write buffer.*

## Unified Cache Control for all caches

- static void **ICACHE\_InvalidateByRange** (uint32\_t address, uint32\_t size\_byte)  
*Invalidates instruction cache by range.*
- static void **DCACHE\_InvalidateByRange** (uint32\_t address, uint32\_t size\_byte)  
*Invalidates data cache by range.*
- static void **DCACHE\_CleanByRange** (uint32\_t address, uint32\_t size\_byte)  
*Clean data cache by range.*
- static void **DCACHE\_CleanInvalidateByRange** (uint32\_t address, uint32\_t size\_byte)  
*Cleans and Invalidates data cache by range.*

## 13.3 Data Structure Documentation

### 13.3.1 struct cache64\_config\_t

#### Data Fields

- `uint32_t boundaryAddr [CACHE64_REGION_NUM-1]`  
*< The cache controller can divide whole memory into 3 regions.*

#### Field Documentation

##### (1) `uint32_t cache64_config_t::boundaryAddr[CACHE64_REGION_NUM-1]`

Boundary address is the FlexSPI internal address (start from 0) instead of system address (start from Flex-SPI AMBA base) to split adjacent regions and must be 1KB aligned. The boundary address itself locates in upper region. Cacheable policy for each region.

## 13.4 Macro Definition Documentation

### 13.4.1 `#define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 8))`

### 13.4.2 `#define CACHE64_LINESIZE_BYTE (FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE)`

### 13.4.3 `#define CACHE64_REGION_NUM (3U)`

### 13.4.4 `#define CACHE64_REGION_ALIGNMENT (0x400U)`

## 13.5 Enumeration Type Documentation

### 13.5.1 enum cache64\_policy\_t

Enumerator

*kCACHE64\_PolicyNonCacheable* Non-cacheable.

*kCACHE64\_PolicyWriteThrough* Write through.

*kCACHE64\_PolicyWriteBack* Write back.

## 13.6 Function Documentation

### 13.6.1 `uint32_t CACHE64_GetInstance ( CACHE64_POLSEL_Type * base )`

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | The peripheral base address. |
|-------------|------------------------------|

Returns

CACHE64\_POLSEL instance number starting from 0.

### 13.6.2 `uint32_t CACHE64_GetInstanceByAddr ( uint32_t address )`

param *address* The physical memory address.

Returns

CACHE64\_CTRL instance number starting from 0.

### 13.6.3 `status_t CACHE64_Init ( CACHE64_POLSEL_Type * base, const cache64_config_t * config )`

This function configures the CACHE64 module with user-defined settings. Call the [CACHE64\\_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration.

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | CACHE64_POLSEL peripheral base address.            |
| <i>config</i> | Pointer to a user-defined configuration structure. |

Return values

|                        |                            |
|------------------------|----------------------------|
| <i>kStatus_Success</i> | CACHE64 initialize succeed |
|------------------------|----------------------------|

### 13.6.4 `void CACHE64_GetDefaultConfig ( cache64_config_t * config )`

This function initializes the CACHE64 configuration structure to a default value. The default values are first region covers whole cacheable area, and policy set to write back.

Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>config</i> | Pointer to a configuration structure. |
|---------------|---------------------------------------|

### 13.6.5 void CACHE64\_EnableCache ( CACHE64\_CTRL\_Type \* *base* )

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | CACHE64_CTRL peripheral base address. |
|-------------|---------------------------------------|

### 13.6.6 void CACHE64\_DisableCache ( CACHE64\_CTRL\_Type \* *base* )

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | CACHE64_CTRL peripheral base address. |
|-------------|---------------------------------------|

### 13.6.7 void CACHE64\_InvalidateCache ( CACHE64\_CTRL\_Type \* *base* )

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | CACHE64_CTRL peripheral base address. |
|-------------|---------------------------------------|

### 13.6.8 void CACHE64\_InvalidateCacheByRange ( uint32\_t *address*, uint32\_t *size\_byte* )

Parameters

|                  |                                                                |
|------------------|----------------------------------------------------------------|
| <i>address</i>   | The physical address of cache.                                 |
| <i>size_byte</i> | size of the memory to be invalidated, should be larger than 0. |

Note

Address and size should be aligned to "CACHE64\_LINESIZE\_BYTE". The startAddr here will be forced to align to CACHE64\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

13.6.9 void CACHE64\_CleanCache ( CACHE64\_CTRL\_Type \* *base* )

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | CACHE64_CTRL peripheral base address. |
|-------------|---------------------------------------|

### 13.6.10 void CACHE64\_CleanCacheByRange ( *uint32\_t address, uint32\_t size\_byte* )

Parameters

|                  |                                                            |
|------------------|------------------------------------------------------------|
| <i>address</i>   | The physical address of cache.                             |
| <i>size_byte</i> | size of the memory to be cleaned, should be larger than 0. |

Note

Address and size should be aligned to "CACHE64\_LINESIZE\_BYTE". The startAddr here will be forced to align to CACHE64\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

### 13.6.11 void CACHE64\_CleanInvalidateCache ( CACHE64\_CTRL\_Type \* *base* )

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | CACHE64_CTRL peripheral base address. |
|-------------|---------------------------------------|

### 13.6.12 void CACHE64\_CleanInvalidateCacheByRange ( *uint32\_t address, uint32\_t size\_byte* )

Parameters

|                  |                                                                            |
|------------------|----------------------------------------------------------------------------|
| <i>address</i>   | The physical address of cache.                                             |
| <i>size_byte</i> | size of the memory to be Cleaned and Invalidated, should be larger than 0. |

Note

Address and size should be aligned to "CACHE64\_LINESIZE\_BYTE". The startAddr here will be forced to align to CACHE64\_LINESIZE\_BYTE if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

13.6.13 void CACHE64\_EnableWriteBuffer ( CACHE64\_CTRL\_Type \* *base*, bool *enable* )

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | CACHE64_CTRL peripheral base address.                                                         |
| <i>enable</i> | The enable or disable flag. true - enable the write buffer. false - disable the write buffer. |

### 13.6.14 static void ICACHE\_InvalidateByRange ( uint32\_t *address*, uint32\_t *size\_byte* ) [inline], [static]

Parameters

|                  |                                                                |
|------------------|----------------------------------------------------------------|
| <i>address</i>   | The physical address.                                          |
| <i>size_byte</i> | size of the memory to be invalidated, should be larger than 0. |

Note

Address and size should be aligned to CACHE64\_LINESIZE\_BYTEx due to the cache operation unit FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTEx. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

### 13.6.15 static void DCACHE\_InvalidateByRange ( uint32\_t *address*, uint32\_t *size\_byte* ) [inline], [static]

Parameters

|                  |                                                                |
|------------------|----------------------------------------------------------------|
| <i>address</i>   | The physical address.                                          |
| <i>size_byte</i> | size of the memory to be invalidated, should be larger than 0. |

Note

Address and size should be aligned to CACHE64\_LINESIZE\_BYTEx due to the cache operation unit FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTEx. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

### 13.6.16 static void DCACHE\_CleanByRange ( uint32\_t *address*, uint32\_t *size\_byte* ) [inline], [static]

## Parameters

|                  |                                                            |
|------------------|------------------------------------------------------------|
| <i>address</i>   | The physical address.                                      |
| <i>size_byte</i> | size of the memory to be cleaned, should be larger than 0. |

## Note

Address and size should be aligned to CACHE64\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

### 13.6.17 static void DCACHE\_CleanInvalidateByRange ( **uint32\_t address, uint32\_t size\_byte** ) [inline], [static]

## Parameters

|                  |                                                                            |
|------------------|----------------------------------------------------------------------------|
| <i>address</i>   | The physical address.                                                      |
| <i>size_byte</i> | size of the memory to be Cleaned and Invalidated, should be larger than 0. |

## Note

Address and size should be aligned to CACHE64\_LINESIZE\_BYTE due to the cache operation unit FSL\_FEATURE\_CACHE64\_CTRL\_LINESIZE\_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size\_byte, application should make sure the alignment or make sure the right operation order if the size\_byte is not aligned.

# Chapter 14

## Common Driver

### 14.1 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 ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))`  
*Computes the number of elements in an array.*
- `#define UINT64_H(X) (((uint32_t)((uint64_t)(X)) >> 32U) & 0xFFFFFFFFFULL)`  
*Macro to get upper 32 bits of a 64-bit value.*
- `#define UINT64_L(X) (((uint32_t)((uint64_t)(X)) & 0xFFFFFFFFFULL))`  
*Macro to get lower 32 bits of a 64-bit value.*
- `#define SUPPRESS_FALL_THROUGH_WARNING()`  
*For switch case code block, if case section ends without "break;" statement, there will be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc.*

#### 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_EPC_USAR` = 57,  
    `kStatusGroup_DMIC` = 58,  
    `kStatusGroup_SDIF` = 59,  
}

```

kStatusGroup_GLIKEY = 168 }

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, 2))  
*common driver version.*

## Debug console type definition.

- #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

- `#define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U`  
*Debug console based on QSCI.*

## Min/max macros

- `#define MIN(a, b) (((a) < (b)) ? (a) : (b))`  
*Computes the minimum of a and b.*
- `#define MAX(a, b) (((a) > (b)) ? (a) : (b))`  
*Computes the maximum of a and b.*

## UINT16\_MAX(UINT32\_MAX value

- `#define UINT16_MAX ((uint16_t)-1)`  
*Max value of uint16\_t type.*
- `#define UINT32_MAX ((uint32_t)-1)`  
*Max value of uint32\_t type.*

## 14.2 Macro Definition Documentation

### 14.2.1 #define FSL\_DRIVER\_TRANSFER\_DOUBLE\_WEAK\_IRQ 1

### 14.2.2 #define MAKE\_STATUS( group, code ) (((group)\*100L) + (code)))

### 14.2.3 #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 |

14.2.4 #define FSL\_COMMON\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 2))

14.2.5 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE 0U

14.2.6 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_UART 1U

14.2.7 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPUART 2U

14.2.8 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPSCI 3U

14.2.9 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_USBCDC 4U

14.2.10 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_FLEXCOMM 5U

14.2.11 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_IUART 6U

14.2.12 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_VUSART 7U

14.2.13 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_MINI\_USART 8U

14.2.14 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_SWO 9U

14.2.15 #define DEBUG\_CONSOLE\_DEVICE\_TYPE\_QSCI 10U

14.2.16 #define MIN( a, b ) (((a) < (b)) ? (a) : (b))

14.2.17 #define MAX( a, b ) (((a) > (b)) ? (a) : (b))

14.2.18 #define ARRAY\_SIZE( x ) (sizeof(x) / sizeof((x)[0]))

14.2.19 #define UINT16\_MAX ((uint16\_t)-1)

14.2.20 #define UINT32\_MAX ((uint32\_t)-1)

14.2.21 #define SUPPRESS\_FALL\_THROUGH\_WARNING( )

To suppress this warning, "SUPPRESS\_FALL\_THROUGH\_WARNING();" need to be added at the end of each case section which misses "break;" statement.

## 14.3 Typedef Documentation

### 14.3.1 `typedef int32_t status_t`

## 14.4 Enumeration Type Documentation

### 14.4.1 `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_MCULCD` 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\_XSPI* Group number for XSPI status codes.  
*kStatusGroup\_PNGDEC* Group number for PNGDEC status codes.  
*kStatusGroup\_JPEGDEC* Group number for JPEGDEC 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\_HAL\_ADC\_SENSOR* Group number for HAL ADC SENSOR 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\_ELEMU* Group number for ELEMU 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\_CSS\_PKC* Group number for CSS 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.  
*kStatusGroup\_ELE* Group number for ELE status codes.  
*kStatusGroup\_GLIKEY* Group number for GLIKEY status codes.

#### 14.4.2 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.

### 14.5 Function Documentation

#### 14.5.1 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. |
|------------|-------------------|

#### 14.5.2 void SDK\_Free ( void \* *ptr* )

Parameters

|            |                           |
|------------|---------------------------|
| <i>ptr</i> | The memory to be release. |
|------------|---------------------------|

#### 14.5.3 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 15

## CRC: Cyclic Redundancy Check Driver

### 15.1 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.

### 15.2 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.

### 15.3 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.

### 15.4 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.

## 15.5 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.*

## 15.6 Data Structure Documentation

### 15.6.1 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::reverseIn`
- (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`

**15.7 Macro Definition Documentation****15.7.1 #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

**15.7.2 #define CRC\_DRIVER\_USE\_CRC16\_CCITT\_FALSE\_AS\_DEFAULT 1**

Uses CRC-16/CCITT-FALSE as default.

**15.8 Enumeration Type Documentation****15.8.1 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*  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

## 15.9 Function Documentation

### 15.9.1 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. |

### 15.9.2 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. |
|-------------|-------------------------|

### 15.9.3 void CRC\_Reset ( **CRC\_Type** \* *base* )

Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | CRC peripheral address. |
|-------------|-------------------------|

### 15.9.4 void CRC\_WriteSeed ( **CRC\_Type** \* *base*, **uint32\_t** *seed* )

Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | CRC peripheral address. |
| <i>seed</i> | CRC Seed value.         |

### 15.9.5 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 |
|---------------|--------------------------------------|

### 15.9.6 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 |

### 15.9.7 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. |

### 15.9.8 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.

**15.9.9 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 16

## CTIMER: Standard counter/timers

### 16.1 Overview

The MCUXpresso SDK provides a driver for the cTimer module of MCUXpresso SDK devices.

### 16.2 Function groups

The cTimer driver supports the generation of PWM signals, input capture, and setting up the timer match conditions.

#### 16.2.1 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.

#### 16.2.2 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.

#### 16.2.3 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.

#### 16.2.4 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.

## 16.3 Typical use case

### 16.3.1 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

### 16.3.2 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\_EnableMatchChannelReload** (CTIMER\_Type \*base, **ctimer\_match\_t** match, bool enable)  
*Enable reload channel falling edge.*
- 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.*
- static void **CTIMER\_SetShadowValue** (CTIMER\_Type \*base, **ctimer\_match\_t** match, uint32\_t matchvalue)  
*Set the specified match shadow channel.*

## 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)

- static void **CTIMER\_UpdatePwmPulsePeriod** (CTIMER\_Type \*base, **ctimer\_match\_t** matchChannel, uint32\_t pulsePeriod)
 

*Configures the PWM signal parameters.*
- 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.*

*Updates the pulse period 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.*

## 16.4 Data Structure Documentation

### 16.4.1 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*

### 16.4.2 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.*

## 16.5 Enumeration Type Documentation

### 16.5.1 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.

### 16.5.2 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.

### 16.5.3 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.

### 16.5.4 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.

### 16.5.5 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.

### 16.5.6 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.

### 16.5.7 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.

### 16.5.8 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

## 16.6 Function Documentation

### 16.6.1 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. |

### 16.6.2 void CTIMER\_Deinit ( **CTIMER\_Type** \* *base* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

### 16.6.3 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. |
|---------------|----------------------------------------------|

### 16.6.4 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. |

**16.6.5 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-<br/>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-<br/>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. |

**16.6.6 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               |

**16.6.7 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 |

**16.6.8 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 |

### 16.6.9 `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 <a href="#">ctimer_external_match_t</a> |

Returns

The mask of external match channel status flags. Users need to use the `_ctimer_external_match` type to decode the return variables.

**16.6.10 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 |

**16.6.11 static uint32\_t CTIMER\_GetTimerCountValue ( CTIMER\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | Ctimer peripheral base address. |
|-------------|---------------------------------|

Returns

return the timer count value.

**16.6.12 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 |

#### 16.6.13 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> |

#### 16.6.14 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> |

#### 16.6.15 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](#)

16.6.16 **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](#)

#### 16.6.17 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> |

#### 16.6.18 static void CTIMER\_StartTimer ( **CTIMER\_Type** \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

#### 16.6.19 static void CTIMER\_StopTimer ( **CTIMER\_Type** \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

#### 16.6.20 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 |
|-------------|--------------------------------|

### 16.6.21 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                 |

### 16.6.22 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.

### 16.6.23 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. |

#### 16.6.24 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. |

#### 16.6.25 static void CTIMER\_EnableMatchChannelReload ( **CTIMER\_Type** \* *base*, **ctimer\_match\_t** *match*, **bool** *enable* ) [inline], [static]

Enable the specified match channel reload match shadow value.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | Ctimer peripheral base address. |
| <i>match</i>  | match channel used.             |
| <i>enable</i> | Enable .                        |

#### 16.6.26 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.     |

**16.6.27 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.    |

**16.6.28 static void CTIMER\_SetShadowValue ( CTIMER\_Type \* *base*,  
                  ctimer\_match\_t *match*, uint32\_t *matchvalue* ) [inline], [static]**

Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>base</i>       | Ctimer peripheral base address.                       |
| <i>match</i>      | match channel used.                                   |
| <i>matchvalue</i> | Reload the value of the corresponding match register. |

# Chapter 17

## DAC: Digital Analog Converter

### 17.1 Overview

The MCUXpresso SDK provides a driver for the Digital Analog Converter (DAC) module of MCUXpresso SDK devices.

Based on the features of the DAC module, the DAC driver is divided into several function groups.

#### Module Initialization Interfaces

The functions in this group can be used to initialize or de-initialize the DAC module. To initialize the DAC module, the function [DAC\\_GetDefaultConfig\(\)](#) can be invoked to get the module's default configuration.

#### Channels Control Interfaces

This function group contains channel control APIs. To configure the channels' all options directly at one time, the function [DAC\\_SetChannelConfig\(\)](#) is provided, this function takes the pointer to structure that is the type of `dac_channel_config_t` as the parameter. All channel-related options are covered in this structure type. To set some specific options of channels, some low-level APIs also provided in this function group. All APIs in this function group take channelMask as the parameter, it means the mask of channel ID. If both channel A and channel B are aimed to set the same options, then users just need to invoke related APIs once with the channelMask set as the OR'ed value of channel A and channel B.

#### Triangle Waveform Configuration Interface

There is only one API in this function group, it is [DAC\\_SetTriangleConfig\(\)](#) function. This function is used to configure the triangle waveform when channel A's wave type is selected as the triangle type.

#### Interrupts Control Interfaces.

The APIs in this function group can be used to enable/disable interrupts.

#### Status Flags Control Interfaces

The APIs in this function group can be used to get/clear status flags.

## Data Structures

- struct `dac_config_t`  
*The structure of dac module basic configuration, including conversion rate, output range, and reference voltage source.* [More...](#)
- struct `dac_channel_config_t`  
*The structure of dac channel configuration, such as trigger type, wave type, timing mode, and so on.* [More...](#)
- struct `dac_triangle_config_t`  
*The structure of triangle waveform, including maximum value, minimum value, step size, and so on.* [More...](#)

## Macros

- `#define FSL_DAC_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`  
*DAC driver version.*
- `#define IS_DAC_CHANNEL_A_WAVE(CH_WAVE) ((CH_WAVE) <= kDAC_WaveNoiseDifferential)`  
*DAC channel A wave mode check.*
- `#define IS_DAC_CHANNEL_B_WAVE(CH_WAVE) (((CH_WAVE) == kDAC_WaveNormal) || ((CH_WAVE) == kDAC_WaveNoiseDifferential))`  
*DAC channel B wave mode check.*

## Enumerations

- enum `_dac_interrupt_enable` {
   
`kDAC_ChannelAReadyInterruptEnable` = DAC\_IMR\_A\_RDY\_INT\_MSK\_MASK,  
`kDAC_ChannelBReadyInterruptEnable` = DAC\_IMR\_B\_RDY\_INT\_MSK\_MASK,  
`kDAC_ChannelATimeoutInterruptEnable` = DAC\_IMR\_A\_TO\_INT\_MSK\_MASK,  
`kDAC_ChannelBTimeoutInterruptEnable` = DAC\_IMR\_B\_TO\_INT\_MSK\_MASK,  
`kDAC_TriangleOverflowInterruptEnable` = DAC\_IMR\_TRIA\_OVFL\_INT\_MSK\_MASK }
- The enumeration of interrupts that DAC support.*
- enum `_dac_status_flags` {
   
`kDAC_ChannelADataReadyInterruptFlag` = DAC\_ISR\_A\_RDY\_INT\_MASK,  
`kDAC_ChannelBDataReadyInterruptFlag` = DAC\_ISR\_B\_RDY\_INT\_MASK,  
`kDAC_ChannelATimeoutInterruptFlag` = DAC\_ISR\_A\_TO\_INT\_MASK,  
`kDAC_ChannelBTimeoutInterruptFlag` = DAC\_ISR\_B\_TO\_INT\_MASK,  
`kDAC_TriangleOverflowInterruptFlag` = DAC\_ISR\_TRIA\_OVFL\_INT\_MASK,  
`kDAC_RawChannelADataReadyFlag` = DAC\_IRSR\_A\_RDY\_INT\_RAW\_MASK << 5UL,  
`kDAC_RawChannelBDataReadyFlag` = DAC\_IRSR\_B\_RDY\_INT\_RAW\_MASK << 5UL,  
`kDAC_RawChannelATimeoutFlag` = DAC\_IRSR\_A\_TO\_INT\_RAW\_MASK << 5UL,  
`kDAC_RawChannelBTimeoutFlag` = DAC\_IRSR\_B\_TO\_INT\_RAW\_MASK << 5UL,  
`kDAC_RawTriangleOverflowFlag` = DAC\_IRSR\_TRIA\_OVFL\_INT\_RAW\_MASK << 5UL,  
`kDAC_ChannelAConversionCompleteFlag` = DAC\_STATUS\_A\_DV\_MASK << 10UL,  
`kDAC_ChannelBConversionCompleteFlag` = DAC\_STATUS\_B\_DV\_MASK << 10UL }
- The enumeration of DAC status flags, including interrupt status flags, raw status flags, and conversion status flags.*
- enum `dac_channel_id_t`

*The enumeration of dac channels, including channel A and channel B.*

- enum `dac_conversion_rate_t` {
   
kDAC\_ConversionRate62P5KHZ = 0U,  
 kDAC\_ConversionRate125KHZ,  
 kDAC\_ConversionRate250KHZ,  
 kDAC\_ConversionRate500KHZ }

*The enumeration of dac converion rate, including 62.5 KHz, 125 KHz, 250 KHz, and 500 KHz.*

- enum `dac_reference_voltage_source_t` {
   
kDAC\_ReferenceInternalVoltageSource = 0U,  
 kDAC\_ReferenceExternalVoltageSource }

*The enumeration of dac reference voltage source.*

- enum `dac_output_voltage_range_t` {
   
kDAC\_RangeSmall = 0U,  
 kDAC\_RangeMiddle = 1U,  
 kDAC\_RangeLarge = 3U }

*The enumeration of dac output voltage range.*

- enum `dac_channel_output_t` {
   
kDAC\_ChannelOutputInternal = 0U,  
 kDAC\_ChannelOutputPAD }

*The enumeration of dac channel's output mode.*

- enum `dac_channel_trigger_type_t` {
   
kDAC\_RisingEdgeTrigger = 1U,  
 kDAC\_FallingEdgeTrigger = 2U,  
 kDAC\_BothEdgeTrigger = 3U }

*The enumeration of dac channel's trigger type, including rising edge trigger, falling edge trigger, and both edge triggers.*

- enum `dac_channel_timing_mode_t` {
   
kDAC\_NonTimingCorrelated = 0U,  
 kDAC\_TimingCorrelated }

*The enumeration of dac channel timing mode.*

- enum `dac_channel_wave_type_t` {
   
kDAC\_WaveNormal = 0U,  
 kDAC\_WaveTriangle = 1U,  
 kDAC\_WaveSine = 2U,  
 kDAC\_WaveNoiseDifferential = 3U }

*The enumerator of channel output wave type, please note that not all wave types are effective to A and B channel.*

- enum `dac_triangle_mamp_t` {

- ```

kDAC_TriangleAmplitude63 = 0U,
kDAC_TriangleAmplitude127,
kDAC_TriangleAmplitude191,
kDAC_TriangleAmplitude255,
kDAC_TriangleAmplitude319,
kDAC_TriangleAmplitude383,
kDAC_TriangleAmplitude447,
kDAC_TriangleAmplitude511,
kDAC_TriangleAmplitude575,
kDAC_TriangleAmplitude639,
kDAC_TriangleAmplitude703,
kDAC_TriangleAmplitude767,
kDAC_TriangleAmplitude831,
kDAC_TriangleAmplitude895,
kDAC_TriangleAmplitude959,
kDAC_TriangleAmplitude1023 }

    DAC triangle maximum amplitude type.
• enum dac_triangle_step_size_t {
  kDAC_TriangleStepSize1 = 0U,
  kDAC_TriangleStepSize3,
  kDAC_TriangleStepSize15,
  kDAC_TriangleStepSize511 }

    DAC triangle step size type.
• enum dac_triangle_waveform_type_t {
  kDAC_TriangleFull = 0U,
  kDAC_TriangleHalf }

    DAC triangle waveform type.

```

## Module Initialization Interfaces

- void `DAC_Init` (`DAC_Type` \*base, const `dac_config_t` \*config)  
*Initializes DAC module, including set reference voltage source, set conversion range, and set output voltage range.*
- void `DAC_GetDefaultConfig` (`dac_config_t` \*config)  
*Gets the default configurations of DAC module.*
- void `DAC_Deinit` (`DAC_Type` \*base)  
*De-initializes the DAC module, including reset clock divider, reset each channel, and so on.*

## Channels Control Interfaces

- void `DAC_SetChannelConfig` (`DAC_Type` \*base, `uint32_t` channelMask, const `dac_channel_config_t` \*channelConfig)  
*Configures the DAC channels, including enable channel conversion, set wave type, set timing mode, and so on.*
- static void `DAC_ResetChannel` (`DAC_Type` \*base, `uint32_t` channelMask)  
*Does software reset for the selected DAC channels.*
- static void `DAC_EnableChannelConversion` (`DAC_Type` \*base, `uint32_t` channelMask, `bool` enable)

- static void **DAC\_SetChannelOutMode** (DAC\_Type \*base, uint32\_t channelMask, **dac\_channel\_output\_t** outMode)
 

*Enables/Disables selected channel conversion.*

*Sets channels out mode, including kDAC\_ChannelOutputInternal and kDAC\_ChannelOutputPad.*
- static void **DAC\_EnableChannelTriggerMode** (DAC\_Type \*base, uint32\_t channelMask, bool enable)
 

*Enables/Disables channels trigger mode.*
- static void **DAC\_SetChannelTrigSource** (DAC\_Type \*base, uint32\_t channelMask, **dac\_channel\_trigger\_source\_t** trigSource)
 

*Sets channels trigger source.*
- static void **DAC\_SetChannelTrigType** (DAC\_Type \*base, uint32\_t channelMask, **dac\_channel\_trigger\_type\_t** trigType)
 

*Sets channels trigger type, such as rising edge trigger, falling edge trigger, or both edge trigger.*
- static void **DAC\_SetChannelTimingMode** (DAC\_Type \*base, uint32\_t channelMask, **dac\_channel\_timing\_mode\_t** timingMode)
 

*Sets channels timing mode, including not-timing related or timing related.*
- static void **DAC\_EnableChannelDMA** (DAC\_Type \*base, uint32\_t channelMask, bool enable)
 

*Enables/Disables channels DMA.*
- static void **DAC\_SetChannelWaveType** (DAC\_Type \*base, uint32\_t channelMask, **dac\_channel\_wave\_type\_t** waveType)
 

*Sets channels wave type, such as sine, noise, or triangle.*
- static void **DAC\_SetChannelData** (DAC\_Type \*base, uint32\_t channelMask, uint16\_t data)
 

*Sets DAC channels data.*

## Triangle Waveform Configuration Interface

- void **DAC\_SetTriangleConfig** (DAC\_Type \*base, const **dac\_triangle\_config\_t** \*triangleConfig)
 

*Configures the options of triangle waveform.*

## Interrupts Control Interfaces.

- static void **DAC\_EnableInterrupts** (DAC\_Type \*base, uint32\_t interruptMask)
 

*Enables interrupts, such as channel A data ready interrupt, channel A timeout interrupt, and so on.*
- static void **DAC\_DisableInterrupts** (DAC\_Type \*base, uint32\_t interruptMask)
 

*Disables interrupts, such as channel B data ready interrupt, channel B timeout interrupt, and so on.*

## Status Flags Control Interfaces

- static uint32\_t **DAC\_GetStatusFlags** (DAC\_Type \*base)
 

*Gets the status flags, including interrupt status flags, raw status flags, and conversion status flags.*
- static void **DAC\_ClearStatusFlags** (DAC\_Type \*base, uint32\_t statusFlagsMask)
 

*Clears the interrupts status flags, such as channel A data ready interrupt flag, channel B data ready interrupt flag, and so on.*

## 17.2 Data Structure Documentation

### 17.2.1 struct dac\_config\_t

#### Data Fields

- `dac_conversion_rate_t conversionRate`  
*Configure DAC conversion rate, please refer to [dac\\_conversion\\_rate\\_t](#).*
- `dac_reference_voltage_source_t refSource`  
*Configure DAC vref source, please refer to [dac\\_reference\\_voltage\\_source\\_t](#).*
- `dac_output_voltage_range_t rangeSelect`  
*Configure DAC channel output range, please refer to [dac\\_output\\_voltage\\_range\\_t](#).*

#### Field Documentation

- (1) `dac_conversion_rate_t dac_config_t::conversionRate`
- (2) `dac_reference_voltage_source_t dac_config_t::refSource`
- (3) `dac_output_voltage_range_t dac_config_t::rangeSelect`

### 17.2.2 struct dac\_channel\_config\_t

#### Data Fields

- `bool enableConversion`  
*Enable/Disable selected channel's conversion.*
- `dac_channel_output_t outMode`  
*Configure channel output mode, please refer to [dac\\_channel\\_output\\_t](#).*
- `bool enableDMA`  
*Enable/Disable channel DAM data transfer.*
- `bool enableTrigger`  
*Enable/Disable external event trigger.*
- `dac_channel_trigger_type_t triggerType`  
*Configure the channel trigger type, please refer to [dac\\_channel\\_trigger\\_type\\_t](#).*
- `dac_channel_trigger_source_t triggerSource`  
*Configure DAC channel trigger source, please refer to [dac\\_channel\\_trigger\\_source\\_t](#).*
- `dac_channel_timing_mode_t timingMode`  
*Configure channel timing mode, please refer to [dac\\_channel\\_timing\\_mode\\_t](#).*
- `dac_channel_wave_type_t waveType`  
*Configure wave type for the selected channel, please refer to [dac\\_channel\\_wave\\_type\\_t](#).*

#### Field Documentation

- (1) `bool dac_channel_config_t::enableConversion`

- **true** Enable selected channel's conversion.
- **false** Disable selected channel's conversion.

(2) `bool dac_channel_config_t::enableDMA`

- **true** DMA data transfer enabled.
- **false** DMA data transfer disabled.

(3) `bool dac_channel_config_t::enableTrigger`

(4) `dac_channel_trigger_type_t dac_channel_config_t::triggerType`

(5) `dac_channel_trigger_source_t dac_channel_config_t::triggerSource`

(6) `dac_channel_timing_mode_t dac_channel_config_t::timingMode`

(7) `dac_channel_wave_type_t dac_channel_config_t::waveType`

### 17.2.3 struct `dac_triangle_config_t`

#### Data Fields

- `dac_triangle_mamp_t triangleMamp`  
*Configure triangle maximum value.*
- `dac_triangle_step_size_t triangleStepSize`  
*Configure triangle step size.*
- `dac_triangle_waveform_type_t triangleWaveform`  
*Configure triangle waveform type.*
- `uint32_t triangleBase`  
*Configure triangle minimum value.*

#### Field Documentation

(1) `dac_triangle_mamp_t dac_triangle_config_t::triangleMamp`

(2) `dac_triangle_step_size_t dac_triangle_config_t::triangleStepSize`

(3) `dac_triangle_waveform_type_t dac_triangle_config_t::triangleWaveform`

(4) `uint32_t dac_triangle_config_t::triangleBase`

### 17.3 Macro Definition Documentation

#### 17.3.1 #define `FSL_DAC_DRIVER_VERSION(MAKE_VERSION(2, 1, 0))`

Version 2.1.0.

## 17.4 Enumeration Type Documentation

### 17.4.1 enum \_dac\_interrupt\_enable

Enumerator

- kDAC\_ChannelAReadyInterruptEnable*** Enable channel A data ready interrupt.
- kDAC\_ChannelBReadyInterruptEnable*** Enable channel B data ready interrupt.
- kDAC\_ChannelATimeoutInterruptEnable*** Enable channel A time out interrupt.
- kDAC\_ChannelBTimeoutInterruptEnable*** Enable channel B time out interrupt.
- kDAC\_TriangleOverflowInterruptEnable*** Enable triangle overflow interrupt.

### 17.4.2 enum \_dac\_status\_flags

Note

The interrupt status flags can only be asserted upon both enabling and happening of related interrupts. Comparatively, the raw status flags will be asserted as long as related events happen regardless of whether related interrupts are enabled or not.

Only interrupt status flags can be cleared manually.

Enumerator

- kDAC\_ChannelADataReadyInterruptFlag*** Channel A data ready.
- kDAC\_ChannelBDataReadyInterruptFlag*** Channel B data ready.
- kDAC\_ChannelATimeoutInterruptFlag*** Channel A time out.
- kDAC\_ChannelBTimeoutInterruptFlag*** Channel B time out.
- kDAC\_TriangleOverflowInterruptFlag*** Triangle overflow.
- kDAC\_RawChannelADataReadyFlag*** Channel A data ready raw.
- kDAC\_RawChannelBDataReadyFlag*** Channel B data ready raw.
- kDAC\_RawChannelATimeoutFlag*** Channel A timeout raw.
- kDAC\_RawChannelBTimeoutFlag*** Channel B timeout raw.
- kDAC\_RawTriangleOverflowFlag*** Triangle overflow raw.
- kDAC\_ChannelAConversionCompleteFlag*** Channel A conversion complete.
- kDAC\_ChannelBConversionCompleteFlag*** Channel B conversion complete.

### 17.4.3 enum dac\_conversion\_rate\_t

Enumerator

- kDAC\_ConversionRate62P5KHZ*** DAC Conversion Rate selects as 62.5 KHz.
- kDAC\_ConversionRate125KHZ*** DAC Conversion Rate selects as 125 KHz.
- kDAC\_ConversionRate250KHZ*** DAC Conversion Rate selects as 250 KHz.
- kDAC\_ConversionRate500KHZ*** DAC Conversion Rate selects as 500 KHz.

#### 17.4.4 enum dac\_reference\_voltage\_source\_t

Enumerator

*kDAC\_ReferenceInternalVoltageSource* Select internal voltage reference.

*kDAC\_ReferenceExternalVoltageSource* Select external voltage reference.

#### 17.4.5 enum dac\_output\_voltage\_range\_t

Enumerator

*kDAC\_RangeSmall* DAC output small range.

*kDAC\_RangeMiddle* DAC output middle range.

*kDAC\_RangeLarge* DAC output large range.

#### 17.4.6 enum dac\_channel\_output\_t

Enumerator

*kDAC\_ChannelOutputInternal* Enable internal output but disable output to pad.

*kDAC\_ChannelOutputPAD* Enable output to pad but disable internal output.

#### 17.4.7 enum dac\_channel\_trigger\_type\_t

Enumerator

*kDAC\_RisingEdgeTrigger* Rising edge trigger.

*kDAC\_FallingEdgeTrigger* Falling edge trigger.

*kDAC\_BothEdgeTrigger* Rising and Falling edge trigger.

#### 17.4.8 enum dac\_channel\_timing\_mode\_t

Enumerator

*kDAC\_NonTimingCorrelated* DAC non-timing-correlated mode.

*kDAC\_TimingCorrelated* DAC timing-correlated mode.

### 17.4.9 enum dac\_channel\_wave\_type\_t

Enumerator

*kDAC\_WaveNormal* No predefined waveform, effective to A or B channel.

*kDAC\_WaveTriangle* Triangle wave, effective only to A channel.

*kDAC\_WaveSine* Sine wave, effective only to A channel.

*kDAC\_WaveNoiseDifferential* Noise wave, effective only to A channel; Differential mode, one's complemental code from A data, effective only to B channel.

### 17.4.10 enum dac\_triangle\_mamp\_t

Enumerator

*kDAC\_TriangleAmplitude63* DAC triangle amplitude 63 lsb.

*kDAC\_TriangleAmplitude127* DAC triangle amplitude 127 lsb.

*kDAC\_TriangleAmplitude191* DAC triangle amplitude 191 lsb.

*kDAC\_TriangleAmplitude255* DAC triangle amplitude 255 lsb.

*kDAC\_TriangleAmplitude319* DAC triangle amplitude 319 lsb.

*kDAC\_TriangleAmplitude383* DAC triangle amplitude 383 lsb.

*kDAC\_TriangleAmplitude447* DAC triangle amplitude 447 lsb.

*kDAC\_TriangleAmplitude511* DAC triangle amplitude 511 lsb.

*kDAC\_TriangleAmplitude575* DAC triangle amplitude 575 lsb.

*kDAC\_TriangleAmplitude639* DAC triangle amplitude 639 lsb.

*kDAC\_TriangleAmplitude703* DAC triangle amplitude 703 lsb.

*kDAC\_TriangleAmplitude767* DAC triangle amplitude 767 lsb.

*kDAC\_TriangleAmplitude831* DAC triangle amplitude 831 lsb.

*kDAC\_TriangleAmplitude895* DAC triangle amplitude 895 lsb.

*kDAC\_TriangleAmplitude959* DAC triangle amplitude 959 lsb.

*kDAC\_TriangleAmplitude1023* DAC triangle amplitude 1023 lsb.

### 17.4.11 enum dac\_triangle\_step\_size\_t

Enumerator

*kDAC\_TriangleStepSize1* DAC triangle step size 1 lsb.

*kDAC\_TriangleStepSize3* DAC triangle step size 3 lsb.

*kDAC\_TriangleStepSize15* DAC triangle step size 15 lsb.

*kDAC\_TriangleStepSize511* DAC triangle step size 511 lsb.

### 17.4.12 enum dac\_triangle\_waveform\_type\_t

Enumerator

*kDAC\_TriangleFull* DAC full triangle waveform.

*kDAC\_TriangleHalf* DAC half triangle waveform.

## 17.5 Function Documentation

### 17.5.1 void DAC\_Init ( DAC\_Type \* *base*, const dac\_config\_t \* *config* )

Parameters

<i>base</i>	DAC peripheral base address.
<i>config</i>	Pointer to the structure which in type of <a href="#">dac_config_t</a> .

### 17.5.2 void DAC\_GetDefaultConfig ( dac\_config\_t \* *config* )

```
*     config->conversionRate = kDAC\_ConversionRate62P5KHZ;
*     config->refSource = kDAC\_ReferenceInternalVoltageSource;
*     config->rangeSelect = kDAC\_RangeLarge;
*
```

Parameters

<i>config</i>	Pointer to the structure which in the type of <a href="#">dac_config_t</a> .
---------------	--

### 17.5.3 void DAC\_Deinit ( DAC\_Type \* *base* )

Parameters

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

### 17.5.4 void DAC\_SetChannelConfig ( DAC\_Type \* *base*, uint32\_t *channelMask*, const dac\_channel\_config\_t \* *channelConfig* )

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel, can be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>channelConfig</i>	The pointer of structure which in the type of <a href="#">dac_channel_config_t</a> .

#### 17.5.5 static void DAC\_ResetChannel ( **DAC\_Type** \* *base*, **uint32\_t** *channelMask* ) [inline], [static]

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel to be reset, should be the OR'ed value of <a href="#">dac_channel_id_t</a> .

#### 17.5.6 static void DAC\_EnableChannelConversion ( **DAC\_Type** \* *base*, **uint32\_t** *channelMask*, **bool** *enable* ) [inline], [static]

Note

To enable/disable the conversions of both channels, invoking this API with the parameter **channelMask** set as **kDAC\_ChannelA|kDAC\_ChannelB** .

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel to be reset, can be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>enable</i>	Enable/Disable channel conversion. <ul style="list-style-type: none"> <li>• <b>true</b> Enable selected channels' conversion.</li> <li>• <b>false</b> Disable selected channels' conversion.</li> </ul>

#### 17.5.7 static void DAC\_SetChannelOutMode ( **DAC\_Type** \* *base*, **uint32\_t** *channelMask*, **dac\_channel\_output\_t** *outMode* ) [inline], [static]

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel, can be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>outMode</i>	The out mode of selected channels, please refer to <a href="#">dac_channel_output_t</a> for details.

### 17.5.8 static void DAC\_EnableChannelTriggerMode ( DAC\_Type \* *base*, uint32\_t *channelMask*, bool *enable* ) [inline], [static]

Note

To enable/disable the trigger mode of both two channels, invoking this API with the parameter **channelMask** set as **kDAC\_ChannelA|kDAC\_ChannelB** .

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel, can be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>enable</i>	Enable/Disable channel trigger mode. <ul style="list-style-type: none"> <li>• <b>true</b> Channels' conversion triggered by external event enabled.</li> <li>• <b>false</b> Channels' conversion triggered by external event disabled.</li> </ul>

### 17.5.9 static void DAC\_SetChannelTrigSource ( DAC\_Type \* *base*, uint32\_t *channelMask*, dac\_channel\_trigger\_source\_t *trigSource* ) [inline], [static]

Note

To set the same trigger source to both two channels, invoking this API with the parameter **channelMask** set as **kDAC\_ChannelA|kDAC\_ChannelB** .

Parameters

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

<i>channelMask</i>	The mask of channel, can be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>trigSource</i>	The selected trigger source, please refer to <a href="#">dac_channel_trigger_source_t</a> for details.

**17.5.10 static void DAC\_SetChannelTrigType ( DAC\_Type \* *base*, uint32\_t *channelMask*, dac\_channel\_trigger\_type\_t *trigType* ) [inline], [static]**

Note

To set the same trigger type to both two channels, invoking this API with the parameter **channelMask** set as **kDAC\_ChannelA|kDAC\_ChannelB** .

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel, can be the OR'ed value of <a href="#">dac_channel_id_t</a> ;
<i>trigType</i>	The selected trigger type, please refer to <a href="#">dac_channel_trigger_type_t</a> ;

**17.5.11 static void DAC\_SetChannelTimingMode ( DAC\_Type \* *base*, uint32\_t *channelMask*, dac\_channel\_timing\_mode\_t *timingMode* ) [inline], [static]**

Note

To the same timing mode to both two channels, invoking this API with the parameter **channelMask** set as **kDAC\_ChannelA|kDAC\_ChannelB** .

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel, can be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>timingMode</i>	The selected timing mode, please refer to <a href="#">dac_channel_timing_mode_t</a> for details.

**17.5.12 static void DAC\_EnableChannelDMA ( DAC\_Type \* *base*, uint32\_t *channelMask*, bool *enable* ) [inline], [static]**

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel, can be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>enable</i>	Enable/Disable channel DMA data transfer. <ul style="list-style-type: none"><li>• <b>true</b> DMA data transfer enabled.</li><li>• <b>false</b> DMA data transfer disabled.</li></ul>

**17.5.13 static void DAC\_SetChannelWaveType ( DAC\_Type \* *base*, uint32\_t *channelMask*, dac\_channel\_wave\_type\_t *waveType* ) [inline], [static]**

Note

To set the same wave type to both channel, invoking this API with the parameter **channelMask** set as **kDAC\_ChannelA|kDAC\_ChannelB** .

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel, should be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>waveType</i>	The wave type to set, please refer to <a href="#">dac_channel_wave_type_t</a> .

**17.5.14 static void DAC\_SetChannelData ( DAC\_Type \* *base*, uint32\_t *channelMask*, uint16\_t *data* ) [inline], [static]**

Note

To set the same data to both channel, invoking this API with the parameter **channelMask** set as **kDAC\_ChannelA|kDAC\_ChannelB** .

Parameters

<i>base</i>	DAC peripheral base address.
<i>channelMask</i>	The mask of channel, can be the OR'ed value of <a href="#">dac_channel_id_t</a> .
<i>data</i>	

### 17.5.15 void DAC\_SetTriangleConfig ( DAC\_Type \* *base*, const dac\_triangle\_config\_t \* *triangleConfig* )

Note

This API should be invoked to set the options of triangle waveform when channel A's output wave type is selected as [kDAC\\_WaveTriangle](#).

Parameters

<i>base</i>	DAC peripheral base address.
<i>triangleConfig</i>	The pointer of structure which in the type of <a href="#">dac_triangle_config_t</a> .

### 17.5.16 static void DAC\_EnableInterrupts ( DAC\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Parameters

<i>base</i>	DAC peripheral base address.
<i>interruptMask</i>	The or'ed value of the interrupts to be enabled, please refer to <a href="#">_dac_interrupt_enable</a> .

### 17.5.17 static void DAC\_DisableInterrupts ( DAC\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

Parameters

<i>base</i>	DAC peripheral base address.
<i>interruptMask</i>	The or'ed value of the interrupts to be disabled, please refer to <a href="#">_dac_interrupt_enable</a> .

### 17.5.18 static uint32\_t DAC\_GetStatusFlags ( DAC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The mask of status flags, please refer to [\\_dac\\_status\\_flags](#).

### 17.5.19 static void DAC\_ClearStatusFlags ( **DAC\_Type** \* *base*, **uint32\_t** *statusFlagsMask* ) [inline], [static]

Parameters

<i>base</i>	DAC peripheral base address.
<i>statusFlags- Mask</i>	The mask of the status flags to be cleared, please refer to <a href="#">_dac_status_flags</a> .

# Chapter 18

## DMA: Direct Memory Access Controller Driver

### 18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Direct Memory Access (DMA) of MCUXpresso SDK devices.

### 18.2 Typical use case

#### 18.2.1 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, 3))  
*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)
 

*Enables the interrupt source for the DMA transfer.*
- static void **DMA\_DisableChannel** (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\\_PrepareTransfer](#) (`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\\_PrepareChannelTransfer](#) (`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.*

## 18.3 Data Structure Documentation

### 18.3.1 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.*

### 18.3.2 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

### 18.3.3 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**

### 18.3.4 struct dma\_channel\_config\_t

## Data Fields

- void \* **srcStartAddr**  
*Source data address.*
- void \* **dstStartAddr**

- `void *nextDesc`  
*Destination data address.*
- `uint32_t xferCfg`  
*Chain custom descriptor.*
- `dma_channel_trigger_t *trigger`  
*channel transfer configurations*
- `dma_channel_trigger_t *trigger`  
*DMA trigger type.*
- `bool isPeriph`  
*select the request type*

### 18.3.5 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.*

### 18.3.6 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

## 18.4 Macro Definition Documentation

### 18.4.1 #define FSL\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 3))

Version 2.5.3.

### 18.4.2 #define FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SIZE (16U)

DMA head link descriptor table align size

### 18.4.3 #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. |

### 18.4.4 #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. |

### 18.4.5 #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. |

**18.4.6 #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. |

**18.4.7 #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          |

## 18.5 Typedef Documentation

**18.5.1 typedef void(\* dma\_callback)(struct \_dma\_handle \*handle, void \*userData,  
bool transferDone, uint32\_t intmode)**

## 18.6 Enumeration Type Documentation

### 18.6.1 anonymous enum

Enumerator

***kStatus\_DMA\_Busy*** Channel is busy and can't handle the transfer request.

### 18.6.2 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 |

### 18.6.3 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 |

### 18.6.4 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.  |

### 18.6.5 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. |

### 18.6.6 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.

### 18.6.7 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

### 18.6.8 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.

### 18.6.9 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.

### 18.6.10 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)

## 18.7 Function Documentation

### 18.7.1 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. |
|-------------|------------------------------|

### 18.7.2 void DMA\_Deinit ( DMA\_Type \* *base* )

This function gates the DMA clock.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | DMA peripheral base address. |
|-------------|------------------------------|

### 18.7.3 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 |

#### 18.7.4 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.

#### 18.7.5 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.

#### 18.7.6 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.          |

#### 18.7.7 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.          |

#### 18.7.8 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.          |

#### 18.7.9 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.          |

#### 18.7.10 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.          |

### 18.7.11 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.

### 18.7.12 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.       |

### 18.7.13 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. |

**18.7.14 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

**18.7.15 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.

### 18.7.16 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.      |

### 18.7.17 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.

### 18.7.18 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.          |

**18.7.19 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.          |

**18.7.20 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.     |

**18.7.21 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. |

### 18.7.22 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.       |

### 18.7.23 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.     |

### 18.7.24 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. |

### 18.7.25 void DMA\_AbortTransfer ( dma\_handle\_t \* *handle* )

This function aborts DMA transfer specified by handle.

Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

### 18.7.26 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.                                                         |

### 18.7.27 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. |

### 18.7.28 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).

18.7.29 **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.                                                                |

### 18.7.30 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.   |

### 18.7.31 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.                                                                |

#### 18.7.32 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.  |

### 18.7.33 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_PrepChannelTransfer(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_PrepChannelTransfer(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.   |

### 18.7.34 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. |
|---------------|---------------------|

### 18.7.35 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 19

## DMIC: Digital Microphone

### 19.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Digital Microphone (DMIC) module.

The DMIC driver is created to help the user more easily operate the DMIC module. This driver can be used to perform basic and advanced DMIC operations. The driver can be used to transfer data from DMIC to memory using DMA as well as in interrupt mode. The DMIC and DMA transfer in pingpong mode is preferred as DMIC is a streaming device.

### 19.2 Function groups

#### 19.2.1 Initialization and deinitialization

This function group implements DMIC initialization and deinitialization API. [DMIC\\_Init\(\)](#) function Enables the clock to the DMIC register interface. [DMIC\\_Dinit\(\)](#) function Disables the clock to the DMIC register interface.

#### 19.2.2 Configuration

This function group implements DMIC configuration API. [DMIC\\_ConfigIO\(\)](#) function configures the use of PDM (Pulse Density modulation) pins. [DMIC\\_SetOperationMode\(\)](#) function configures the mode of operation either in DMA or in interrupt. [DMIC\\_ConfigChannel\(\)](#) function configures the various properties of a DMIC channel. [DMIC\\_Use2fs\(\)](#) function configures the clock scaling used for PCM data output. [DMIC\\_EnableChannel\(\)](#) function enables a particular DMIC channel. [DMIC\\_FifoChannel\(\)](#) function configures FIFO settings for a DMIC channel.

#### 19.2.3 DMIC Data and status

This function group implements the API to get data and status of DMIC FIFO. [DMIC\\_FifoGetStatus\(\)](#) function gives the status of a DMIC FIFO. [DMIC\\_ClearStatus\(\)](#) function clears the status of a DMIC FIFO. [DMIC\\_FifoGetData\(\)](#) function gets data from a DMIC FIFO.

#### 19.2.4 DMIC Interrupt Functions

[DMIC\\_EnableIntCallback\(\)](#) enables the interrupt for the selected DMIC peripheral. [DMIC\\_DisableIntCallback\(\)](#) disables the interrupt for the selected DMIC peripheral.

### 19.2.5 DMIC HWVAD Functions

This function group implements the API for HWVAD. [DMIC\\_SetGainNoiseEstHwvad\(\)](#) Sets the gain value for the noise estimator. [DMIC\\_SetGainSignalEstHwvad\(\)](#) Sets the gain value for the signal estimator. [DMIC\\_SetFilterCtrlHwvad\(\)](#) Sets the HWVAD filter cutoff frequency parameter. [DMIC\\_SetInputGainHwvad\(\)](#) Sets the input gain of HWVAD. [DMIC\\_CtrlClrIntrHwvad\(\)](#) Clears HWVAD internal interrupt flag. [DMIC\\_FilterResetHwvad\(\)](#) Resets HWVAD filters. [DMIC\\_GetNoiseEnvlpEst\(\)](#) Gets the value from output of the filter z7.

### 19.2.6 DMIC HWVAD Interrupt Functions

[DMIC\\_HwvadEnableIntCallback\(\)](#) enables the HWVAD interrupt for the selected DMIC peripheral. [DMIC\\_HwvadDisableIntCallback\(\)](#) disables the HWVAD interrupt for the selected DMIC peripheral.

## 19.3 Typical use case

### 19.3.1 DMIC DMA Configuration

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

### 19.3.2 DMIC use case

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

## Modules

- [DMIC Driver](#)

## 19.4 DMIC Driver

### 19.4.1 Overview

#### Files

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

#### Data Structures

- struct [dmic\\_channel\\_config\\_t](#)  
*DMIC Channel configuration structure.* [More...](#)

#### Typedefs

- typedef void(\* [dmic\\_callback\\_t](#) )(void)  
*DMIC Callback function.*
- typedef void(\* [dmic\\_hwvad\\_callback\\_t](#) )(void)  
*HWVAD Callback function.*

#### Enumerations

- enum {
   
*kStatus\_DMIC\_Busy* = MAKE\_STATUS(kStatusGroup\_DMIC, 0),
   
*kStatus\_DMIC\_Idle* = MAKE\_STATUS(kStatusGroup\_DMIC, 1),
   
*kStatus\_DMIC\_OverRunError* = MAKE\_STATUS(kStatusGroup\_DMIC, 2),
   
*kStatus\_DMIC\_UnderRunError* = MAKE\_STATUS(kStatusGroup\_DMIC, 3) }
   
*\_dmc\_status DMIC transfer status.*
- enum [operation\\_mode\\_t](#) {
   
*kDMIC\_OperationModeInterrupt* = 1U,
   
*kDMIC\_OperationModeDma* = 2U }
   
*DMIC different operation modes.*
- enum [stereo\\_side\\_t](#) {
   
*kDMIC\_Left* = 0U,
   
*kDMIC\_Right* = 1U }
   
*DMIC left/right values.*
- enum [pdm\\_div\\_t](#) {

```
kDMIC_PdmDiv1 = 0U,
kDMIC_PdmDiv2 = 1U,
kDMIC_PdmDiv3 = 2U,
kDMIC_PdmDiv4 = 3U,
kDMIC_PdmDiv6 = 4U,
kDMIC_PdmDiv8 = 5U,
kDMIC_PdmDiv12 = 6U,
kDMIC_PdmDiv16 = 7U,
kDMIC_PdmDiv24 = 8U,
kDMIC_PdmDiv32 = 9U,
kDMIC_PdmDiv48 = 10U,
kDMIC_PdmDiv64 = 11U,
kDMIC_PdmDiv96 = 12U,
kDMIC_PdmDiv128 = 13U }
```

*DMIC Clock pre-divider values.*

- enum `compensation_t` {
 

```
kDMIC_CompValueZero = 0U,
kDMIC_CompValueNegativePoint16 = 1U,
kDMIC_CompValueNegativePoint15 = 2U,
kDMIC_CompValueNegativePoint13 = 3U }
```

*Pre-emphasis Filter coefficient value for 2FS and 4FS modes.*

- enum `dc_removal_t` {
 

```
kDMIC_DcNoRemove = 0U,
kDMIC_DcCut155 = 1U,
kDMIC_DcCut78 = 2U,
kDMIC_DcCut39 = 3U }
```

*DMIC DC filter control values.*

- enum `dmic_channel_t` {
 

```
kDMIC_Channel0 = 0U,
kDMIC_Channel1 = 1U,
kDMIC_Channel2 = 2U,
kDMIC_Channel3 = 3U }
```

*DMIC Channel number.*

- enum {
 

```
kDMIC_EnableChannel0 = 1 << 0U,
kDMIC_EnableChannel1 = 1 << 1U,
kDMIC_EnableChannel2 = 1 << 2U,
kDMIC_EnableChannel3 = 1 << 3U }
```

*\_dmic\_channel\_mask DMIC Channel mask.*

- enum `dmic_phy_sample_rate_t` {
 

```
kDMIC_PhysFullSpeed = 0U,
kDMIC_PhysHalfSpeed = 1U }
```

*DMIC and decimator sample rates.*

## DMIC version

- #define **FSL\_DMIC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 3, 2))  
*DMIC driver version 2.3.2.*

## Initialization and deinitialization

- uint32\_t **DMICGetInstance** (DMIC\_Type \*base)  
*Get the DMIC instance from peripheral base address.*
- void **DMICInit** (DMIC\_Type \*base)  
*Turns DMIC Clock on.*
- void **DMICDeInit** (DMIC\_Type \*base)  
*Turns DMIC Clock off.*
- void **DMICSetOperationMode** (DMIC\_Type \*base, operation\_mode\_t mode)  
*Set DMIC operating mode.*
- void **DMICUse2fs** (DMIC\_Type \*base, bool use2fs)  
*Configure Clock scaling.*

## Channel configuration

- void **DMICCfgChannelDc** (DMIC\_Type \*base, dmic\_channel\_t channel, dc\_removal\_t dc\_cut\_level, uint32\_t post\_dc\_gain\_reduce, bool saturate16bit)  
*Configure DMIC channel.*
- static void **DMICEnableChannelSignExtend** (DMIC\_Type \*base, dmic\_channel\_t channel, bool enable)  
*Enable channel sign extend which allows processing of 24bit audio data on 32bit machines.*
- void **DMICConfigChannel** (DMIC\_Type \*base, dmic\_channel\_t channel, stereo\_side\_t side, dmic\_channel\_config\_t \*channel\_config)  
*Configure DMIC channel.*
- void **DMICEnableChannnel** (DMIC\_Type \*base, uint32\_t channelmask)  
*Enable a particualr channel.*
- void **DMICFifoChannel** (DMIC\_Type \*base, uint32\_t channel, uint32\_t trig\_level, uint32\_t enable, uint32\_t resetn)  
*Configure fifo settings for DMIC channel.*
- static void **DMICEnableChannelInterrupt** (DMIC\_Type \*base, dmic\_channel\_t channel, bool enable)  
*Enable a particualr channel interrupt request.*
- static void **DMICEnableChannelDma** (DMIC\_Type \*base, dmic\_channel\_t channel, bool enable)  
*Enable a particualr channel dma request.*
- static void **DMICEnableChannelFifo** (DMIC\_Type \*base, dmic\_channel\_t channel, bool enable)  
*Enable a particualr channel fifo.*
- static void **DMICDoFifoReset** (DMIC\_Type \*base, dmic\_channel\_t channel)  
*Channel fifo reset.*
- static uint32\_t **DMICFifoGetStatus** (DMIC\_Type \*base, uint32\_t channel)  
*Get FIFO status.*
- static void **DMICFifoClearStatus** (DMIC\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Clear FIFO status.*
- static uint32\_t **DMICFifoGetData** (DMIC\_Type \*base, uint32\_t channel)

- static uint32\_t **DMIC\_FifoGetAddress** (DMIC\_Type \*base, uint32\_t channel)
 

*Get FIFO address.*
- void **DMIC\_ResetChannelDecimator** (DMIC\_Type \*base, uint32\_t channelMask, bool reset)
 

*DMIC channel Decimator reset.*

## Register callback.

- void **DMIC\_EnableIntCallback** (DMIC\_Type \*base, dmic\_callback\_t cb)
 

*Enable callback.*
- void **DMIC\_DisableIntCallback** (DMIC\_Type \*base, dmic\_callback\_t cb)
 

*Disable callback.*

## HWVAD

- static void **DMIC\_SetGainNoiseEstHwvad** (DMIC\_Type \*base, uint32\_t value)
 

*Sets the gain value for the noise estimator.*
- static void **DMIC\_SetGainSignalEstHwvad** (DMIC\_Type \*base, uint32\_t value)
 

*Sets the gain value for the signal estimator.*
- static void **DMIC\_SetFilterCtrlHwvad** (DMIC\_Type \*base, uint32\_t value)
 

*Sets the hwvad filter cutoff frequency parameter.*
- static void **DMIC\_SetInputGainHwvad** (DMIC\_Type \*base, uint32\_t value)
 

*Sets the input gain of hwvad.*
- static void **DMIC\_CtrlClrIntrHwvad** (DMIC\_Type \*base, bool st10)
 

*Clears hwvad internal interrupt flag.*
- static void **DMIC\_FilterResetHwvad** (DMIC\_Type \*base, bool rslt)
 

*Resets hwvad filters.*
- static uint16\_t **DMIC\_GetNoiseEnvlpEst** (DMIC\_Type \*base)
 

*Gets the value from output of the filter z7.*
- void **DMIC\_HwvadEnableIntCallback** (DMIC\_Type \*base, dmic\_hwvad\_callback\_t vadcb)
 

*Enable hwvad callback.*
- void **DMIC\_HwvadDisableIntCallback** (DMIC\_Type \*base, dmic\_hwvad\_callback\_t vadcb)
 

*Disable callback.*

## 19.4.2 Data Structure Documentation

### 19.4.2.1 struct dmic\_channel\_config\_t

#### Data Fields

- **pdm\_div\_t divhfclk**

*DMIC Clock pre-divider values.*
- **uint32\_t osr**

*oversampling rate(CIC decimation rate) for PCM*
- **int32\_t gainshft**

*4FS PCM data gain control*
- **compensation\_t preac2coef**

- *Pre-emphasis Filter coefficient value for 2FS.*
- `compensation_t preac4coef`  
*Pre-emphasis Filter coefficient value for 4FS.*
- `dc_removal_t dc_cut_level`  
*DMIC DC filter control values.*
- `uint32_t post_dc_gain_reduce`  
*Fine gain adjustment in the form of a number of bits to downshift.*
- `dmic_phy_sample_rate_t sample_rate`  
*DMIC and decimator sample rates.*
- `bool saturate16bit`  
*Selects 16-bit saturation.*
- `bool enableSignExtend`  
*sign extend feature which allows processing of 24bit audio data on 32bit machine*

## Field Documentation

(1) `dc_removal_t dmic_channel_config_t::dc_cut_level`

(2) `bool dmic_channel_config_t::saturate16bit`

0 means results roll over if out range and do not saturate. 1 means if the result overflows, it saturates at 0xFFFF for positive overflow and 0x8000 for negative overflow.

## 19.4.3 Macro Definition Documentation

19.4.3.1 `#define FSL_DMIC_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))`

## 19.4.4 Typedef Documentation

19.4.4.1 `typedef void(* dmic_callback_t)(void)`

19.4.4.2 `typedef void(* dmic_hwvad_callback_t)(void)`

## 19.4.5 Enumeration Type Documentation

19.4.5.1 `anonymous enum`

Enumerator

`kStatus_DMIC_Busy` DMIC is busy.

`kStatus_DMIC_Idle` DMIC is idle.

`kStatus_DMIC_OverRunError` DMIC over run Error.

`kStatus_DMIC_UnderRunError` DMIC under run Error.

#### 19.4.5.2 enum operation\_mode\_t

Enumerator

*kDMIC\_OperationModeInterrupt* Interrupt mode.

*kDMIC\_OperationModeDma* DMA mode.

#### 19.4.5.3 enum stereo\_side\_t

Enumerator

*kDMIC\_Left* Left Stereo channel.

*kDMIC\_Right* Right Stereo channel.

#### 19.4.5.4 enum pdm\_div\_t

Enumerator

*kDMIC\_PdmDiv1* DMIC pre-divider set in divide by 1.

*kDMIC\_PdmDiv2* DMIC pre-divider set in divide by 2.

*kDMIC\_PdmDiv3* DMIC pre-divider set in divide by 3.

*kDMIC\_PdmDiv4* DMIC pre-divider set in divide by 4.

*kDMIC\_PdmDiv6* DMIC pre-divider set in divide by 6.

*kDMIC\_PdmDiv8* DMIC pre-divider set in divide by 8.

*kDMIC\_PdmDiv12* DMIC pre-divider set in divide by 12.

*kDMIC\_PdmDiv16* DMIC pre-divider set in divide by 16.

*kDMIC\_PdmDiv24* DMIC pre-divider set in divide by 24.

*kDMIC\_PdmDiv32* DMIC pre-divider set in divide by 32.

*kDMIC\_PdmDiv48* DMIC pre-divider set in divide by 48.

*kDMIC\_PdmDiv64* DMIC pre-divider set in divide by 64.

*kDMIC\_PdmDiv96* DMIC pre-divider set in divide by 96.

*kDMIC\_PdmDiv128* DMIC pre-divider set in divide by 128.

#### 19.4.5.5 enum compensation\_t

Enumerator

*kDMIC\_CompValueZero* Compensation 0.

*kDMIC\_CompValueNegativePoint16* Compensation -0.16.

*kDMIC\_CompValueNegativePoint15* Compensation -0.15.

*kDMIC\_CompValueNegativePoint13* Compensation -0.13.

#### 19.4.5.6 enum dc\_removal\_t

Enumerator

- kDMIC\_DcNoRemove* Flat response no filter.
- kDMIC\_DcCut155* Cut off Frequency is 155 Hz.
- kDMIC\_DcCut78* Cut off Frequency is 78 Hz.
- kDMIC\_DcCut39* Cut off Frequency is 39 Hz.

#### 19.4.5.7 enum dmic\_channel\_t

Enumerator

- kDMIC\_Channel0* DMIC channel 0.
- kDMIC\_Channel1* DMIC channel 1.
- kDMIC\_Channel2* DMIC channel 2.
- kDMIC\_Channel3* DMIC channel 3.

#### 19.4.5.8 anonymous enum

Enumerator

- kDMIC\_EnableChannel0* DMIC channel 0 mask.
- kDMIC\_EnableChannel1* DMIC channel 1 mask.
- kDMIC\_EnableChannel2* DMIC channel 2 mask.
- kDMIC\_EnableChannel3* DMIC channel 3 mask.

#### 19.4.5.9 enum dmic\_phy\_sample\_rate\_t

Enumerator

- kDMIC\_PhysFullSpeed* Decimator gets one sample per each chosen clock edge of PDM interface.
- kDMIC\_PhysHalfSpeed* PDM clock to Microphone is halved, decimator receives each sample twice.

### 19.4.6 Function Documentation

#### 19.4.6.1 uint32\_t DMIC\_GetInstance ( **DMIC\_Type** \* *base* )

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | DMIC peripheral base address. |
|-------------|-------------------------------|

Returns

DMIC instance.

#### 19.4.6.2 void DMIC\_Init ( DMIC\_Type \* *base* )

Parameters

|             |             |
|-------------|-------------|
| <i>base</i> | : DMIC base |
|-------------|-------------|

Returns

Nothing

#### 19.4.6.3 void DMIC\_DelInit ( DMIC\_Type \* *base* )

Parameters

|             |             |
|-------------|-------------|
| <i>base</i> | : DMIC base |
|-------------|-------------|

Returns

Nothing

#### 19.4.6.4 void DMIC\_SetOperationMode ( DMIC\_Type \* *base*, operation\_mode\_t *mode* )

**Deprecated** Do not use this function. It has been superceded by [DMIC\\_EnableChannelInterrupt](#), [DMIC\\_EnableChannelDma](#).

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | : The base address of DMIC interface |
| <i>mode</i> | : DMIC mode                          |

Returns

Nothing

#### 19.4.6.5 void DMIC\_Use2fs ( DMIC\_Type \* *base*, bool *use2fs* )

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | : The base address of DMIC interface |
| <i>use2fs</i> | : clock scaling                      |

Returns

Nothing

#### 19.4.6.6 void DMIC\_CfgChannelDc ( DMIC\_Type \* *base*, dmic\_channel\_t *channel*, dc\_removal\_t *dc\_cut\_level*, uint32\_t *post\_dc\_gain\_reduce*, bool *saturate16bit* )

Parameters

|                            |                                                                      |
|----------------------------|----------------------------------------------------------------------|
| <i>base</i>                | : The base address of DMIC interface                                 |
| <i>channel</i>             | : DMIC channel                                                       |
| <i>dc_cut_level</i>        | : dc_removal_t, Cut off Frequency                                    |
| <i>post_dc_gain_reduce</i> | : Fine gain adjustment in the form of a number of bits to downshift. |
| <i>saturate16bit</i>       | : If selects 16-bit saturation.                                      |

#### 19.4.6.7 static void DMIC\_EnableChannelSignExtend ( DMIC\_Type \* *base*, dmic\_channel\_t *channel*, bool *enable* ) [inline], [static]

Parameters

|                |                                                            |
|----------------|------------------------------------------------------------|
| <i>base</i>    | : The base address of DMIC interface                       |
| <i>channel</i> | : DMIC channel                                             |
| <i>enable</i>  | : true is enable sign extend, false is disable sign extend |

#### 19.4.6.8 void DMIC\_ConfigChannel ( **DMIC\_Type** \* *base*, **dmic\_channel\_t** *channel*, **stereo\_side\_t** *side*, **dmic\_channel\_config\_t** \* *channel\_config* )

Parameters

|                       |                                          |
|-----------------------|------------------------------------------|
| <i>base</i>           | : The base address of DMIC interface     |
| <i>channel</i>        | : DMIC channel                           |
| <i>side</i>           | : stereo_side_t, choice of left or right |
| <i>channel_config</i> | : Channel configuration                  |

Returns

Nothing

#### 19.4.6.9 void DMIC\_EnableChannnel ( **DMIC\_Type** \* *base*, **uint32\_t** *channelmask* )

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | : The base address of DMIC interface |
| <i>channelmask</i> | reference _dmic_channel_mask         |

Returns

Nothing

#### 19.4.6.10 void DMIC\_FifoChannel ( **DMIC\_Type** \* *base*, **uint32\_t** *channel*, **uint32\_t** *trig\_level*, **uint32\_t** *enable*, **uint32\_t** *resetn* )

Parameters

|                   |                                      |
|-------------------|--------------------------------------|
| <i>base</i>       | : The base address of DMIC interface |
| <i>channel</i>    | : DMIC channel                       |
| <i>trig_level</i> | : FIFO trigger level                 |
| <i>enable</i>     | : FIFO level                         |
| <i>resetn</i>     | : FIFO reset                         |

Returns

Nothing

#### 19.4.6.11 static void DMIC\_EnableChannelInterrupt( DMIC\_Type \* *base*, dmic\_channel\_t *channel*, bool *enable* ) [inline], [static]

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : Channel selection                  |
| <i>enable</i>  | : true is enable, false is disable   |

#### 19.4.6.12 static void DMIC\_EnableChannelDma( DMIC\_Type \* *base*, dmic\_channel\_t *channel*, bool *enable* ) [inline], [static]

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : Channel selection                  |
| <i>enable</i>  | : true is enable, false is disable   |

#### 19.4.6.13 static void DMIC\_EnableChannelFifo( DMIC\_Type \* *base*, dmic\_channel\_t *channel*, bool *enable* ) [inline], [static]

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : Channel selection                  |
| <i>enable</i>  | : true is enable, false is disable   |

**19.4.6.14 static void DMIC\_DoFifoReset ( DMIC\_Type \* *base*, dmic\_channel\_t *channel* )  
[inline], [static]**

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : Channel selection                  |

**19.4.6.15 static uint32\_t DMIC\_FifoGetStatus ( DMIC\_Type \* *base*, uint32\_t *channel* )  
[inline], [static]**

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : DMIC channel                       |

Returns

FIFO status

**19.4.6.16 static void DMIC\_FifoClearStatus ( DMIC\_Type \* *base*, uint32\_t *channel*,  
uint32\_t *mask* ) [inline], [static]**

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : DMIC channel                       |

|             |                      |
|-------------|----------------------|
| <i>mask</i> | : Bits to be cleared |
|-------------|----------------------|

Returns

FIFO status

**19.4.6.17 static uint32\_t DMIC\_FifoGetData ( DMIC\_Type \* *base*, uint32\_t *channel* )  
[inline], [static]**

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : DMIC channel                       |

Returns

FIFO data

**19.4.6.18 static uint32\_t DMIC\_FifoGetAddress ( DMIC\_Type \* *base*, uint32\_t *channel* )  
[inline], [static]**

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : DMIC channel                       |

Returns

FIFO data

**19.4.6.19 void DMIC\_ResetChannelDecimator ( DMIC\_Type \* *base*, uint32\_t  
*channelMask*, bool *reset* )**

Parameters

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <i>base</i>        | : The base address of DMIC interface                   |
| <i>channelMask</i> | : DMIC channel mask, reference _dmic_channel_mask      |
| <i>reset</i>       | : true is reset decimator, false is release decimator. |

#### 19.4.6.20 void DMIC\_EnableIntCallback ( DMIC\_Type \* *base*, dmic\_callback\_t *cb* )

This function enables the interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | Base address of the DMIC peripheral.         |
| <i>cb</i>   | callback Pointer to store callback function. |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 19.4.6.21 void DMIC\_DisableIntCallback ( DMIC\_Type \* *base*, dmic\_callback\_t *cb* )

This function disables the interrupt for the selected DMIC peripheral.

Parameters

|             |                                               |
|-------------|-----------------------------------------------|
| <i>base</i> | Base address of the DMIC peripheral.          |
| <i>cb</i>   | callback Pointer to store callback function.. |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 19.4.6.22 static void DMIC\_SetGainNoiseEstHwvad ( DMIC\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

|              |                                     |
|--------------|-------------------------------------|
| <i>base</i>  | DMIC base pointer                   |
| <i>value</i> | gain value for the noise estimator. |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 19.4.6.23 static void DMIC\_SetGainSignalEstHwvad ( DMIC\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | DMIC base pointer                    |
| <i>value</i> | gain value for the signal estimator. |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 19.4.6.24 static void DMIC\_SetFilterCtrlHwvad ( DMIC\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

|              |                          |
|--------------|--------------------------|
| <i>base</i>  | DMIC base pointer        |
| <i>value</i> | cut off frequency value. |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 19.4.6.25 static void DMIC\_SetInputGainHwvad ( DMIC\_Type \* *base*, uint32\_t *value* ) [inline], [static]

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | DMIC base pointer           |
| <i>value</i> | input gain value for hwvad. |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

**19.4.6.26 static void DMIC\_CtrlClrlntrHwvad ( DMIC\_Type \* *base*, bool *st10* )  
[inline], [static]**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | DMIC base pointer |
| <i>st10</i> | bit value.        |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

**19.4.6.27 static void DMIC\_FilterResetHwvad ( DMIC\_Type \* *base*, bool *rstt* )  
[inline], [static]**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | DMIC base pointer |
| <i>rstt</i> | Reset bit value.  |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

**19.4.6.28 static uint16\_t DMIC\_GetNoiseEnvIpEst ( DMIC\_Type \* *base* ) [inline],  
[static]**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | DMIC base pointer |
|-------------|-------------------|

Return values

|               |               |
|---------------|---------------|
| <i>output</i> | of filter z7. |
|---------------|---------------|

#### 19.4.6.29 void DMIC\_HwvadEnableIntCallback ( DMIC\_Type \* *base*, dmic\_hwvad\_callback\_t *vadcb* )

This function enables the hwvad interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

|              |                                              |
|--------------|----------------------------------------------|
| <i>base</i>  | Base address of the DMIC peripheral.         |
| <i>vadcb</i> | callback Pointer to store callback function. |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

#### 19.4.6.30 void DMIC\_HwvadDisableIntCallback ( DMIC\_Type \* *base*, dmic\_hwvad\_callback\_t *vadcb* )

This function disables the hwvad interrupt for the selected DMIC peripheral.

Parameters

|              |                                               |
|--------------|-----------------------------------------------|
| <i>base</i>  | Base address of the DMIC peripheral.          |
| <i>vadcb</i> | callback Pointer to store callback function.. |

Return values

|              |
|--------------|
| <i>None.</i> |
|--------------|

# Chapter 20

## ENET: Ethernet MAC Driver

### 20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 10/100 Mbps Ethernet MAC (ENET) module of MCUXpresso SDK devices.

#### ENET: Ethernet MAC Driver {EthernetMACDriver}

### 20.2 Operations of Ethernet MAC Driver

#### 20.2.1 MII interface Operation

The MII interface is the interface connected with MAC and PHY. the Serial management interface - MII management interface should be set before any access to the external PHY chip register. Call [ENET\\_SetSMI\(\)](#) to initialize the MII management interface. Use [ENET\\_StartSMIRead\(\)](#), [ENET\\_StartSMIWrite\(\)](#), and [ENET\\_ReadSMIData\(\)](#) to read/write to PHY registers. This function group sets up the MII and serial management SMI interface, gets data from the SMI interface, and starts the SMI read and write command. Use [ENET\\_SetMII\(\)](#) to configure the MII before successfully getting data from the external PHY.

#### 20.2.2 MAC address filter

This group sets/gets the ENET mac address and the multicast group address filter. [ENET\\_AddMulticastGroup\(\)](#) should be called to add the ENET MAC to the multicast group. The IEEE 1588 feature requires receiving the PTP message.

#### 20.2.3 Other Basic control Operations

This group has the receive active API [ENET\\_ActiveRead\(\)](#) for single and multiple rings. The [ENET\\_AVBConfigure\(\)](#) is provided to configure the AVB features to support the AVB frames transmission. Note that due to the AVB frames transmission scheme being a credit-based TX scheme, it is only supported with the Enhanced buffer descriptors. Because of this, the AVB configuration should only be done with the Enhanced buffer descriptor. When the AVB feature is required, make sure the the "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" is defined before using this feature.

#### 20.2.4 Transactional Operation

For ENET receive, the [ENET\\_GetRxFrameSize\(\)](#) function needs to be called to get the received data size. Then, call the [ENET\\_ReadFrame\(\)](#) function to get the received data. If the received error occurs, call the

`ENET_GetRxErrBeforeReadFrame()` function after `ENET_GetRxFrameSize()` and before `ENET_ReadFrame()` functions to get the detailed error information.

For ENET transmit, call the `ENET_SendFrame()` function to send the data out. The transmit data error information is only accessible for the IEEE 1588 enhanced buffer descriptor mode. When the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined, the `ENET_GetTxErrAfterSendFrame()` can be used to get the detail transmit error information. The transmit error information can only be updated by uDMA after the data is transmitted. The `ENET_GetTxErrAfterSendFrame()` function is recommended to be called on the transmit interrupt handler.

If send/read frame with zero-copy mechanism is needed, there're special APIs like `ENET_GetRxFrame()` and `ENET_StartTxFrame()`. The send frame zero-copy APIs can't be used mixed with `ENET_SendFrame()` for the same ENET peripheral, same as read frame zero-copy APIs.

## 20.2.5 PTP IEEE 1588 Feature Operation

This function group configures the PTP IEEE 1588 feature, starts/stops/gets/sets/adjusts the PTP IEEE 1588 timer, gets the receive/transmit frame timestamp, and PTP IEEE 1588 timer channel feature setting.

The `ENET_Ptp1588Configure()` function needs to be called when the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined and the IEEE 1588 feature is required.

## 20.3 Typical use case

### 20.3.1 ENET Initialization, receive, and transmit operations

For the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` undefined use case, use the legacy type buffer descriptor transmit/receive the frame as follows. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet`. For the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` defined use case, add the PTP IEEE 1588 configuration to enable the PTP IEEE 1588 feature. The initialization occurs as follows. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet`

## Data Structures

- struct `enet_rx_bd_struct_t`  
*Defines the receive buffer descriptor structure for the little endian system. [More...](#)*
- struct `enet_tx_bd_struct_t`  
*Defines the enhanced transmit buffer descriptor structure for the little endian system. [More...](#)*
- struct `enet_data_error_stats_t`  
*Defines the ENET data error statistics structure. [More...](#)*
- struct `enet_rx_frame_error_t`  
*Defines the Rx frame error structure. [More...](#)*
- struct `enet_transfer_stats_t`  
*Defines the ENET transfer statistics structure. [More...](#)*
- struct `enet_frame_info_t`  
*Defines the frame info structure. [More...](#)*

- struct `enet_tx_dirty_ring_t`  
Defines the ENET transmit dirty addresses ring/queue structure. [More...](#)
- struct `enet_buffer_config_t`  
Defines the receive buffer descriptor configuration structure. [More...](#)
- struct `enet_intcoalesce_config_t`  
Defines the interrupt coalescing configure structure. [More...](#)
- struct `enet_config_t`  
Defines the basic configuration structure for the ENET device. [More...](#)
- struct `enet_tx_bd_ring_t`  
Defines the ENET transmit buffer descriptor ring/queue structure. [More...](#)
- struct `enet_rx_bd_ring_t`  
Defines the ENET receive buffer descriptor ring/queue structure. [More...](#)
- struct `enet_handle_t`  
Defines the ENET handler structure. [More...](#)

## Macros

- `#define ENET_BUFFDESCRIPTOR_RX_ERR_MASK`  
Defines the receive error status flag mask.

## Typedefs

- `typedef void *(*enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)`  
Defines the ENET Rx memory buffer alloc function pointer.
- `typedef void(*enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)`  
Defines the ENET Rx memory buffer free function pointer.
- `typedef void(*enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)`  
ENET callback function.
- `typedef void(*enet_isr_t)(ENET_Type *base, enet_handle_t *handle)`  
Define interrupt IRQ handler.

## Enumerations

- enum {
   
`kStatus_ENET_InitMemoryFail,`
  
`kStatus_ENET_RxFrameError = MAKE_STATUS(kStatusGroup_ENET, 1U),`
  
`kStatus_ENET_RxFrameFail = MAKE_STATUS(kStatusGroup_ENET, 2U),`
  
`kStatus_ENET_RxFrameEmpty = MAKE_STATUS(kStatusGroup_ENET, 3U),`
  
`kStatus_ENET_RxFrameDrop = MAKE_STATUS(kStatusGroup_ENET, 4U),`
  
`kStatus_ENET_TxFrameOverLen = MAKE_STATUS(kStatusGroup_ENET, 5U),`
  
`kStatus_ENET_TxFrameBusy = MAKE_STATUS(kStatusGroup_ENET, 6U),`
  
`kStatus_ENET_TxFrameFail = MAKE_STATUS(kStatusGroup_ENET, 7U) }`
  
Defines the status return codes for transaction.
- enum `enet_mii_mode_t` {
   
`kENET_MiiMode = 0U,`
  
`kENET_RmiiMode = 1U }`
  
Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

- enum `enet_mii_speed_t` {  
  kENET\_MiiSpeed10M = 0U,  
  kENET\_MiiSpeed100M = 1U }  
  *Defines the 10/100/1000 Mbps speed for the MII data interface.*
- enum `enet_mii_duplex_t` {  
  kENET\_MiiHalfDuplex = 0U,  
  kENET\_MiiFullDuplex }  
  *Defines the half or full duplex for the MII data interface.*
- enum `enet_mii_write_t` {  
  kENET\_MiiWriteNoCompliant = 0U,  
  kENET\_MiiWriteValidFrame }  
  *Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.*
- enum `enet_mii_read_t` {  
  kENET\_MiiReadValidFrame = 2U,  
  kENET\_MiiReadNoCompliant = 3U }  
  *Defines the read operation for the MII management frame.*
- enum `enet_mii_extend_opcode` {  
  kENET\_MiiAddrWrite\_C45 = 0U,  
  kENET\_MiiWriteFrame\_C45 = 1U,  
  kENET\_MiiReadFrame\_C45 = 3U }  
  *Define the MII opcode for extended MDIO\_CLAUSES\_45 Frame.*
- enum `enet_special_control_flag_t` {  
  kENET\_ControlFlowControlEnable = 0x0001U,  
  kENET\_ControlRxPayloadCheckEnable = 0x0002U,  
  kENET\_ControlRxPadRemoveEnable = 0x0004U,  
  kENET\_ControlRxBroadCastRejectEnable = 0x0008U,  
  kENET\_ControlMacAddrInsert = 0x0010U,  
  kENET\_ControlStoreAndFwdDisable = 0x0020U,  
  kENET\_ControlSMIPreambleDisable = 0x0040U,  
  kENET\_ControlPromiscuousEnable = 0x0080U,  
  kENET\_ControlMIILoopEnable = 0x0100U,  
  kENET\_ControlVLANTagEnable = 0x0200U }  
  *Defines a special configuration for ENET MAC controller.*
- enum `enet_interrupt_enable_t` {

```

kENET_BabrInterrupt = ENET_EIR_BABR_MASK,
kENET_BabtInterrupt = ENET_EIR_BABT_MASK,
kENET_GraceStopInterrupt = ENET_EIR_GRA_MASK,
kENET_TxFrameInterrupt = ENET_EIR_TXF_MASK,
kENET_TxBufferInterrupt = ENET_EIR_TXB_MASK,
kENET_RxFrameInterrupt = ENET_EIR_RXF_MASK,
kENET_RxBufferInterrupt = ENET_EIR_RXB_MASK,
kENET_MiiInterrupt = ENET_EIR_MII_MASK,
kENET_EBusERInterrupt = ENET_EIR_EBERR_MASK,
kENET_LateCollisionInterrupt = ENET_EIR_LC_MASK,
kENET_RetryLimitInterrupt = ENET_EIR_RL_MASK,
kENET_UnderrunInterrupt = ENET_EIR_UN_MASK,
kENET_PayloadRxInterrupt = ENET_EIR_PLR_MASK,
kENET_WakeupInterrupt = ENET_EIR_WAKEUP_MASK,
kENET_TsAvailInterrupt = ENET_EIR_TS_AVAIL_MASK,
kENET_TsTimerInterrupt = ENET_EIR_TS_TIMER_MASK }

```

*List of interrupts supported by the peripheral.*

- enum `enet_event_t` {
   
kENET\_RxEvent,  
kENET\_TxEvent,  
kENET\_ErrEvent,  
kENET\_WakeUpEvent,  
kENET\_TimeStampEvent,  
kENET\_TimeStampAvailEvent }

*Defines the common interrupt event for callback use.*

- enum `enet_tx_accelerator_t` {
   
kENET\_TxAccelIsShift16Enabled = ENET\_TACC\_SHIFT16\_MASK,  
kENET\_TxAccelIpCheckEnabled = ENET\_TACC\_IPCHK\_MASK,  
kENET\_TxAccelProtoCheckEnabled = ENET\_TACC\_PROCHK\_MASK }

*Defines the transmit accelerator configuration.*

- enum `enet_rx_accelerator_t` {
   
kENET\_RxAccelPadRemoveEnabled = ENET\_RACC\_PADREM\_MASK,  
kENET\_RxAccelIpCheckEnabled = ENET\_RACC\_IPDIS\_MASK,  
kENET\_RxAccelProtoCheckEnabled = ENET\_RACC\_PRODIS\_MASK,  
kENET\_RxAccelMacCheckEnabled = ENET\_RACC\_LINEDIS\_MASK,  
kENET\_RxAccelIsShift16Enabled = ENET\_RACC\_SHIFT16\_MASK }

*Defines the receive accelerator configuration.*

## Functions

- `uint32_t ENET_GetInstance (ENET_Type *base)`  
*Get the ENET instance from peripheral base address.*

## Variables

- const `clock_ip_name_t s_enetClock []`  
*Pointers to enet clocks for each instance.*

## Driver version

- #define `FSL_ENET_DRIVER_VERSION` (`MAKE_VERSION(2, 9, 1)`)  
*Defines the driver version.*

## Control and status region bit masks of the receive buffer descriptor.

Defines the queue number.

- #define `ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK` 0x8000U  
*Empty bit mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK` 0x4000U  
*Software owner one mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_WRAP_MASK` 0x2000U  
*Next buffer descriptor is the start address.*
- #define `ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_MASK` 0x1000U  
*Software owner two mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_LAST_MASK` 0x0800U  
*Last BD of the frame mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_MISS_MASK` 0x0100U  
*Received because of the promiscuous mode.*
- #define `ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK` 0x0080U  
*Broadcast packet mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK` 0x0040U  
*Multicast packet mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK` 0x0020U  
*Length violation mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK` 0x0010U  
*Non-octet aligned frame mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_CRC_MASK` 0x0004U  
*CRC error mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK` 0x0002U  
*FIFO overrun mask.*
- #define `ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK` 0x0001U  
*Frame is truncated mask.*

## Control and status bit masks of the transmit buffer descriptor.

- #define `ENET_BUFFDESCRIPTOR_TX_READY_MASK` 0x8000U  
*Ready bit mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_SOFTOWNER1_MASK` 0x4000U  
*Software owner one mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_WRAP_MASK` 0x2000U  
*Wrap buffer descriptor mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_SOFTOWNER2_MASK` 0x1000U  
*Software owner two mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_LAST_MASK` 0x0800U  
*Last BD of the frame mask.*
- #define `ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK` 0x0400U  
*Transmit CRC mask.*

## Defines some Ethernet parameters.

- #define ENET\_FRAME\_MAX\_FRAMELEN 1518U  
*Default maximum Ethernet frame size without VLAN tag.*
- #define ENET\_FRAME\_VLAN\_TAGLEN 4U  
*Ethernet single VLAN tag size.*
- #define ENET\_FRAME\_CRC\_LEN 4U  
*CRC size in a frame.*
- #define ENET\_FRAME\_TX\_LEN\_LIMITATION(x) (((x)->RCR & ENET\_RCR\_MAX\_FL\_-  
MASK) >> ENET\_RCR\_MAX\_FL\_SHIFT) - ENET\_FRAME\_CRC\_LEN)
- #define ENET\_FIFO\_MIN\_RX\_FULL 5U  
*ENET minimum receive FIFO full.*
- #define ENET\_RX\_MIN\_BUFFERSIZE 256U  
*ENET minimum buffer size.*
- #define ENET\_PHY\_MAXADDRESS (ENET\_MMFR\_PA\_MASK >> ENET\_MMFR\_PA\_SHI-  
FT)  
*Maximum PHY address.*
- #define ENET\_TX\_INTERRUPT ((uint32\_t)kENET\_TxFrameInterrupt | (uint32\_t)kENET\_Tx-  
BufferInterrupt)  
*Enet Tx interrupt flag.*
- #define ENET\_RX\_INTERRUPT ((uint32\_t)kENET\_RxFrameInterrupt | (uint32\_t)kENET\_Rx-  
BufferInterrupt)  
*Enet Rx interrupt flag.*
- #define ENET\_TS\_INTERRUPT ((uint32\_t)kENET\_TsTimerInterrupt | (uint32\_t)kENET\_Ts-  
AvailInterrupt)  
*Enet timestamp interrupt flag.*
- #define ENET\_ERR\_INTERRUPT  
*Enet error interrupt flag.*

## Initialization and De-initialization

- void ENET\_GetDefaultConfig (enet\_config\_t \*config)  
*Gets the ENET default configuration structure.*
- status\_t ENET\_Up (ENET\_Type \*base, enet\_handle\_t \*handle, const enet\_config\_t \*config, const  
enet\_buffer\_config\_t \*bufferConfig, uint8\_t \*macAddr, uint32\_t srcClock\_Hz)  
*Initializes the ENET module.*
- status\_t ENET\_Init (ENET\_Type \*base, enet\_handle\_t \*handle, const enet\_config\_t \*config, const  
enet\_buffer\_config\_t \*bufferConfig, uint8\_t \*macAddr, uint32\_t srcClock\_Hz)  
*Initializes the ENET module.*
- void ENET\_Down (ENET\_Type \*base)  
*Stops the ENET module.*
- void ENET\_Deinit (ENET\_Type \*base)  
*Deinitializes the ENET module.*
- static void ENET\_Reset (ENET\_Type \*base)  
*Resets the ENET module.*
- void ENET\_ResetHardware (void)  
*Resets the ENET hardware.*

## MII interface operation

- void ENET\_SetMII (ENET\_Type \*base, enet\_mii\_speed\_t speed, enet\_mii\_duplex\_t duplex)

- void **ENET\_SetSMI** (ENET\_Type \*base, uint32\_t srcClock\_Hz, bool isPreambleDisabled)
 

*Sets the ENET MII speed and duplex.*
- static bool **ENET\_GetSMI** (ENET\_Type \*base)
 

*Gets the ENET SMI(serial management interface)- MII management interface.*
- static uint32\_t **ENET\_ReadSMIData** (ENET\_Type \*base)
 

*Reads data from the PHY register through an SMI interface.*
- static void **ENET\_StartSMIWrite** (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, **enet\_mii-write\_t** operation, uint16\_t data)
 

*Sends the MDIO IEEE802.3 Clause 22 format write command.*
- static void **ENET\_StartSMIRead** (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, **enet\_mii-read\_t** operation)
 

*Sends the MDIO IEEE802.3 Clause 22 format read command.*
- status\_t **ENET\_MDIOWrite** (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t data)
 

*MDIO write with IEEE802.3 Clause 22 format.*
- status\_t **ENET\_MDIORead** (ENET\_Type \*base, uint8\_t phyAddr, uint8\_t regAddr, uint16\_t \*pData)
 

*MDIO read with IEEE802.3 Clause 22 format.*
- static void **ENET\_StartExtC45SMIWriteReg** (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr)
 

*Sends the MDIO IEEE802.3 Clause 45 format write register command.*
- static void **ENET\_StartExtC45SMIWriteData** (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t data)
 

*Sends the MDIO IEEE802.3 Clause 45 format write data command.*
- static void **ENET\_StartExtC45SMIReadData** (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr)
 

*Sends the MDIO IEEE802.3 Clause 45 format read data command.*
- status\_t **ENET\_MDIOC45Write** (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr, uint16\_t data)
 

*MDIO write with IEEE802.3 Clause 45 format.*
- status\_t **ENET\_MDIOC45Read** (ENET\_Type \*base, uint8\_t portAddr, uint8\_t devAddr, uint16\_t regAddr, uint16\_t \*pData)
 

*MDIO read with IEEE802.3 Clause 45 format.*

## MAC Address Filter

- void **ENET\_SetMacAddr** (ENET\_Type \*base, uint8\_t \*macAddr)
 

*Sets the ENET module Mac address.*
- void **ENET\_GetMacAddr** (ENET\_Type \*base, uint8\_t \*macAddr)
 

*Gets the ENET module Mac address.*
- void **ENET\_AddMulticastGroup** (ENET\_Type \*base, uint8\_t \*address)
 

*Adds the ENET device to a multicast group.*
- void **ENET\_LeaveMulticastGroup** (ENET\_Type \*base, uint8\_t \*address)
 

*Moves the ENET device from a multicast group.*

## Other basic operation

- static void **ENET\_ActiveRead** (ENET\_Type \*base)
 

*Activates frame reception for multiple rings.*
- static void **ENET\_EnableSleepMode** (ENET\_Type \*base, bool enable)

*Enables/disables the MAC to enter sleep mode.*

- static void [ENET\\_GetAccelFunction](#) (ENET\_Type \*base, uint32\_t \*txAccelOption, uint32\_t \*rxAccelOption)  
*Gets ENET transmit and receive accelerator functions from MAC controller.*

## Interrupts.

- static void [ENET\\_EnableInterrupts](#) (ENET\_Type \*base, uint32\_t mask)  
*Enables the ENET interrupt.*
- static void [ENET\\_DisableInterrupts](#) (ENET\_Type \*base, uint32\_t mask)  
*Disables the ENET interrupt.*
- static uint32\_t [ENET\\_GetInterruptStatus](#) (ENET\_Type \*base)  
*Gets the ENET interrupt status flag.*
- static void [ENET\\_ClearInterruptStatus](#) (ENET\_Type \*base, uint32\_t mask)  
*Clears the ENET interrupt events status flag.*
- void [ENET\\_SetRxISRHandler](#) (ENET\_Type \*base, enet\_isr\_t ISRHandler)  
*Set the second level Rx IRQ handler.*
- void [ENET\\_SetTxISRHandler](#) (ENET\_Type \*base, enet\_isr\_t ISRHandler)  
*Set the second level Tx IRQ handler.*
- void [ENET\\_SetErrISRHandler](#) (ENET\_Type \*base, enet\_isr\_t ISRHandler)  
*Set the second level Err IRQ handler.*

## Transactional operation

- void [ENET\\_GetRxErrBeforeReadFrame](#) (enet\_handle\_t \*handle, enet\_data\_error\_stats\_t \*eErrorStatic, uint8\_t ringId)  
*Gets the error statistics of a received frame for ENET specified ring.*
- void [ENET\\_EnableStatistics](#) (ENET\_Type \*base, bool enable)  
*Enables/disables collection of transfer statistics.*
- void [ENET\\_GetStatistics](#) (ENET\_Type \*base, enet\_transfer\_stats\_t \*statistics)  
*Gets transfer statistics.*
- void [ENET\\_ResetStatistics](#) (ENET\_Type \*base)  
*Resets transfer statistics.*
- status\_t [ENET\\_GetRxFrameSize](#) (enet\_handle\_t \*handle, uint32\_t \*length, uint8\_t ringId)  
*Gets the size of the read frame for specified ring.*
- status\_t [ENET\\_ReadFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t \*data, uint32\_t length, uint8\_t ringId, uint32\_t \*ts)  
*Reads a frame from the ENET device.*
- status\_t [ENET\\_SendFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, const uint8\_t \*data, uint32\_t length, uint8\_t ringId, bool tsFlag, void \*context)  
*Transmits an ENET frame for specified ring.*
- status\_t [ENET\\_SetTxReclaim](#) (enet\_handle\_t \*handle, bool isEnabled, uint8\_t ringId)  
*Enable or disable tx descriptors reclaim mechanism.*
- void [ENET\\_ReclaimTxDescriptor](#) (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t ringId)  
*Reclaim tx descriptors.*
- status\_t [ENET\\_GetRxFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, enet\_rx\_frame\_struct\_t \*rxFrame, uint8\_t ringId)  
*Receives one frame in specified BD ring with zero copy.*
- status\_t [ENET\\_StartTxFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, enet\_tx\_frame\_struct\_t \*txFrame, uint8\_t ringId)

- Sends one frame in specified BD ring with zero copy.  
void **ENET\_TransmitIRQHandler** (ENET\_Type \*base, enet\_handle\_t \*handle)  
*The transmit IRQ handler.*
- void **ENET\_ReceiveIRQHandler** (ENET\_Type \*base, enet\_handle\_t \*handle)  
*The receive IRQ handler.*
- void **ENET\_ErrorIRQHandler** (ENET\_Type \*base, enet\_handle\_t \*handle)  
*Some special IRQ handler including the error, mii, wakeup irq handler.*
- void **ENET\_Ptp1588IRQHandler** (ENET\_Type \*base)  
*the common IRQ handler for the 1588 irq handler.*
- void **ENET\_CommonFrame0IRQHandler** (ENET\_Type \*base)  
*the common IRQ handler for the tx/rx/error etc irq handler.*

## 20.4 Data Structure Documentation

### 20.4.1 struct enet\_rx\_bd\_struct\_t

#### Data Fields

- uint16\_t **length**  
*Buffer descriptor data length.*
- uint16\_t **control**  
*Buffer descriptor control and status.*
- uint32\_t **buffer**  
*Data buffer pointer.*

#### Field Documentation

- (1) uint16\_t **enet\_rx\_bd\_struct\_t::length**
- (2) uint16\_t **enet\_rx\_bd\_struct\_t::control**
- (3) uint32\_t **enet\_rx\_bd\_struct\_t::buffer**

### 20.4.2 struct enet\_tx\_bd\_struct\_t

#### Data Fields

- uint16\_t **length**  
*Buffer descriptor data length.*
- uint16\_t **control**  
*Buffer descriptor control and status.*
- uint32\_t **buffer**  
*Data buffer pointer.*

**Field Documentation**

- (1) `uint16_t enet_tx_bd_struct_t::length`
- (2) `uint16_t enet_tx_bd_struct_t::control`
- (3) `uint32_t enet_tx_bd_struct_t::buffer`

**20.4.3 struct enet\_data\_error\_stats\_t****Data Fields**

- `uint32_t statsRxLenGreaterErr`  
*Receive length greater than RCR[MAX\_FL].*
- `uint32_t statsRxAlignErr`  
*Receive non-octet alignment.*
- `uint32_t statsRxFcsErr`  
*Receive CRC error.*
- `uint32_t statsRxOverRunErr`  
*Receive over run.*
- `uint32_t statsRxTruncateErr`  
*Receive truncate.*

**Field Documentation**

- (1) `uint32_t enet_data_error_stats_t::statsRxLenGreaterErr`
- (2) `uint32_t enet_data_error_stats_t::statsRxFcsErr`
- (3) `uint32_t enet_data_error_stats_t::statsRxOverRunErr`
- (4) `uint32_t enet_data_error_stats_t::statsRxTruncateErr`

**20.4.4 struct enet\_rx\_frame\_error\_t****Data Fields**

- `bool statsRxTruncateErr: 1`  
*Receive truncate.*
- `bool statsRxOverRunErr: 1`  
*Receive over run.*
- `bool statsRxFcsErr: 1`  
*Receive CRC error.*
- `bool statsRxAlignErr: 1`  
*Receive non-octet alignment.*
- `bool statsRxLenGreaterErr: 1`  
*Receive length greater than RCR[MAX\_FL].*

**Field Documentation**

- (1) **bool enet\_rx\_frame\_error\_t::statsRxTruncateErr**
- (2) **bool enet\_rx\_frame\_error\_t::statsRxOverRunErr**
- (3) **bool enet\_rx\_frame\_error\_t::statsRxFcsErr**
- (4) **bool enet\_rx\_frame\_error\_t::statsRxAlignErr**
- (5) **bool enet\_rx\_frame\_error\_t::statsRxLenGreaterErr**

**20.4.5 struct enet\_transfer\_stats\_t****Data Fields**

- **uint32\_t statsRxFrameCount**  
*Rx frame number.*
- **uint32\_t statsRxFrameOk**  
*Good Rx frame number.*
- **uint32\_t statsRxCrcErr**  
*Rx frame number with CRC error.*
- **uint32\_t statsRxAlignErr**  
*Rx frame number with alignment error.*
- **uint32\_t statsRxDropInvalidSFD**  
*Dropped frame number due to invalid SFD.*
- **uint32\_t statsRxFifoOverflowErr**  
*Rx FIFO overflow count.*
- **uint32\_t statsTxFrameCount**  
*Tx frame number.*
- **uint32\_t statsTxFrameOk**  
*Good Tx frame number.*
- **uint32\_t statsTxCrcAlignErr**  
*The transmit frame is error.*
- **uint32\_t statsTxFifoUnderRunErr**  
*Tx FIFO underrun count.*

**Field Documentation**

- (1) `uint32_t enet_transfer_stats_t::statsRxFrameCount`
- (2) `uint32_t enet_transfer_stats_t::statsRxFrameOk`
- (3) `uint32_t enet_transfer_stats_t::statsRxCrcErr`
- (4) `uint32_t enet_transfer_stats_t::statsRxAlignErr`
- (5) `uint32_t enet_transfer_stats_t::statsRxDropInvalidSFD`
- (6) `uint32_t enet_transfer_stats_t::statsRxFifoOverflowErr`
- (7) `uint32_t enet_transfer_stats_t::statsTxFrameCount`
- (8) `uint32_t enet_transfer_stats_t::statsTxFrameOk`
- (9) `uint32_t enet_transfer_stats_t::statsTxCrcAlignErr`
- (10) `uint32_t enet_transfer_stats_t::statsTxFifoUnderRunErr`

**20.4.6 struct enet\_frame\_info\_t****Data Fields**

- `void * context`  
*User specified data.*

**20.4.7 struct enet\_tx\_dirty\_ring\_t****Data Fields**

- `enet_frame_info_t * txDirtyBase`  
*Dirty buffer descriptor base address pointer.*
- `uint16_t txGenIdx`  
*tx generate index.*
- `uint16_t txConsumIdx`  
*tx consume index.*
- `uint16_t txRingLen`  
*tx ring length.*
- `bool isFull`  
*tx ring is full flag.*

**Field Documentation**

- (1) `enet_frame_info_t* enet_tx_dirty_ring_t::txDirtyBase`
- (2) `uint16_t enet_tx_dirty_ring_t::txGenIdx`
- (3) `uint16_t enet_tx_dirty_ring_t::txConsumIdx`
- (4) `uint16_t enet_tx_dirty_ring_t::txRingLen`
- (5) `bool enet_tx_dirty_ring_t::isFull`

**20.4.8 struct enet\_buffer\_config\_t**

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

1. The aligned receive and transmit buffer size must be evenly divisible by ENET\_BUFF\_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET\_BUFF\_ALIGNMENT" and the cache line size.
2. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET\_BUFF\_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
3. The aligned transmit and receive data buffer start address must be evenly divisible by ENET\_BUFF\_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber \* rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber \* txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET\_BUFF\_ALIGNMENT" and the cache line size.

**Data Fields**

- `uint16_t rxBdNumber`  
*Receive buffer descriptor number.*
- `uint16_t txBdNumber`  
*Transmit buffer descriptor number.*
- `uint16_t rxBuffSizeAlign`  
*Aligned receive data buffer size.*
- `uint16_t txBuffSizeAlign`  
*Aligned transmit data buffer size.*
- `volatile enet_rx_bd_struct_t * rxBdStartAddrAlign`  
*Aligned receive buffer descriptor start address: should be non-cacheable.*
- `volatile enet_tx_bd_struct_t * txBdStartAddrAlign`  
*Aligned transmit buffer descriptor start address: should be non-cacheable.*
- `uint8_t * rxBufferAlign`  
*Receive data buffer start address.*
- `uint8_t * txBufferAlign`  
*Transmit data buffer start address.*
- `bool rxMaintainEnable`

- *Receive buffer cache maintain.*
- **bool txMaintainEnable**  
*Transmit buffer cache maintain.*
- **enet\_frame\_info\_t \* txFrameInfo**  
*Transmit frame information start address.*

## Field Documentation

- (1) **uint16\_t enet\_buffer\_config\_t::rxBdNumber**
- (2) **uint16\_t enet\_buffer\_config\_t::txBdNumber**
- (3) **uint16\_t enet\_buffer\_config\_t::rxBuffSizeAlign**
- (4) **uint16\_t enet\_buffer\_config\_t::txBuffSizeAlign**
- (5) **volatile enet\_rx\_bd\_struct\_t\* enet\_buffer\_config\_t::rxBdStartAddrAlign**
- (6) **volatile enet\_tx\_bd\_struct\_t\* enet\_buffer\_config\_t::txBdStartAddrAlign**
- (7) **uint8\_t\* enet\_buffer\_config\_t::rxBufferAlign**
- (8) **uint8\_t\* enet\_buffer\_config\_t::txBufferAlign**
- (9) **bool enet\_buffer\_config\_t::rxMaintainEnable**
- (10) **bool enet\_buffer\_config\_t::txMaintainEnable**
- (11) **enet\_frame\_info\_t\* enet\_buffer\_config\_t::txFrameInfo**

### 20.4.9 struct enet\_intcoalesce\_config\_t

## Data Fields

- **uint8\_t txCoalesceFrameCount [FSL\_FEATURE\_ENET\_QUEUE]**  
*Transmit interrupt coalescing frame count threshold.*
- **uint16\_t txCoalesceTimeCount [FSL\_FEATURE\_ENET\_QUEUE]**  
*Transmit interrupt coalescing timer count threshold.*
- **uint8\_t rxCoalesceFrameCount [FSL\_FEATURE\_ENET\_QUEUE]**  
*Receive interrupt coalescing frame count threshold.*
- **uint16\_t rxCoalesceTimeCount [FSL\_FEATURE\_ENET\_QUEUE]**  
*Receive interrupt coalescing timer count threshold.*

**Field Documentation**

- (1) `uint8_t enet_intcoalesce_config_t::txCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- (2) `uint16_t enet_intcoalesce_config_t::txCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`
- (3) `uint8_t enet_intcoalesce_config_t::rxCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- (4) `uint16_t enet_intcoalesce_config_t::rxCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`

**20.4.10 struct enet\_config\_t**

Note:

1. macSpecialConfig is used for a special control configuration, A logical OR of "enet\_special\_control\_flag\_t". For a special configuration for MAC, set this parameter to 0.
2. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO .... 3 - 192 bytes written to TX FIFO .... The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO .... txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
3. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET\_FIFO\_MIN\_RX\_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size if smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.
4. When "kENET\_ControlFlowControlEnable" is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
5. When "kENET\_ControlStoreAndFwdDisabled" is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
6. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet\_tx\_accelerator\_t" and "enet\_rx\_accelerator\_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET\_ControlStoreAndFwdDisabled should not be set.
7. The intCoalesceCfg can be used in the rx or tx enabled cases to decrease the CPU loading.

**Data Fields**

- `uint32_t macSpecialConfig`  
*Mac special configuration.*
- `uint32_t interrupt`  
*Mac interrupt source.*
- `uint16_t rxMaxFrameLen`  
*Receive maximum frame length.*
- `enet_mii_mode_t miiMode`  
*MII mode.*
- `enet_mii_speed_t miiSpeed`

- **`enet_mii_duplex_t`** *MII Speed.*
- **`enet_rx_accelerConfig`** *MII duplex.*
- **`uint8_t txAccelerConfig`** *Receive accelerator; A logical OR of "enet\_rx\_accelerator\_t".*
- **`uint8_t txAccelerConfig`** *Transmit accelerator; A logical OR of "enet\_rx\_accelerator\_t".*
- **`uint16_t pauseDuration`** *For flow control enabled case: Pause duration.*
- **`uint8_t rxFifoEmptyThreshold`** *For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.*
- **`uint8_t rxFifoStatEmptyThreshold`** *For flow control enabled case: number of frames in the receive FIFO, independent of size, that can be accept.*
- **`uint8_t rxFifoFullThreshold`** *For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.*
- **`uint8_t txFifoWatermark`** *For store and forward disable case, the data required in TX FIFO before a frame transmit start.*
- **`enet_intcoalesce_config_t * intCoalesceCfg`** *If the interrupt coalesce is not required in the ring n(0,1,2), please set to NULL.*
- **`uint8_t ringNum`** *Number of used rings.*
- **`enet_rx_alloc_callback_t rxBuffAlloc`** *Callback function to alloc memory, must be provided for zero-copy Rx.*
- **`enet_rx_free_callback_t rxBuffFree`** *Callback function to free memory, must be provided for zero-copy Rx.*
- **`enet_callback_t callback`** *General callback function.*
- **`void * userData`** *Callback function parameter.*

## Field Documentation

### (1) `uint32_t enet_config_t::macSpecialConfig`

A logical OR of "enet\_special\_control\_flag\_t".

### (2) `uint32_t enet_config_t::interrupt`

A logical OR of "enet\_interrupt\_enable\_t".

- (3) `uint16_t enet_config_t::rxMaxFrameLen`
- (4) `enet_mii_mode_t enet_config_t::miiMode`
- (5) `enet_mii_speed_t enet_config_t::miiSpeed`
- (6) `enet_mii_duplex_t enet_config_t::miiDuplex`
- (7) `uint8_t enet_config_t::rxAccelerConfig`
- (8) `uint8_t enet_config_t::txAccelerConfig`
- (9) `uint16_t enet_config_t::pauseDuration`
- (10) `uint8_t enet_config_t::rxFifoEmptyThreshold`
- (11) `uint8_t enet_config_t::rxFifoStatEmptyThreshold`

If the limit is reached, reception continues and a pause frame is triggered.

- (12) `uint8_t enet_config_t::rxFifoFullThreshold`
- (13) `uint8_t enet_config_t::txFifoWatermark`
- (14) `enet_intcoalesce_config_t* enet_config_t::intCoalesceCfg`
- (15) `uint8_t enet_config_t::ringNum`

default with 1 – single ring.

- (16) `enet_rx_alloc_callback_t enet_config_t::rxBuffAlloc`
- (17) `enet_rx_free_callback_t enet_config_t::rxBuffFree`
- (18) `enet_callback_t enet_config_t::callback`
- (19) `void* enet_config_t::userData`

#### 20.4.11 struct enet\_tx\_bd\_ring\_t

### Data Fields

- volatile `enet_tx_bd_struct_t * txBdBase`  
*Buffer descriptor base address pointer.*
- `uint16_t txGenIdx`  
*The current available transmit buffer descriptor pointer.*
- `uint16_t txConsumIdx`  
*Transmit consume index.*
- volatile `uint16_t txDescUsed`  
*Transmit descriptor used number.*

- `uint16_t txRingLen`  
*Transmit ring length.*

**Field Documentation**

- (1) `volatile enet_tx_bd_struct_t* enet_tx_bd_ring_t::txBdBase`
- (2) `uint16_t enet_tx_bd_ring_t::txGenIdx`
- (3) `uint16_t enet_tx_bd_ring_t::txConsumIdx`
- (4) `volatile uint16_t enet_tx_bd_ring_t::txDescUsed`
- (5) `uint16_t enet_tx_bd_ring_t::txRingLen`

**20.4.12 struct enet\_rx\_bd\_ring\_t****Data Fields**

- `volatile enet_rx_bd_struct_t * rxBdBase`  
*Buffer descriptor base address pointer.*
- `uint16_t rxGenIdx`  
*The current available receive buffer descriptor pointer.*
- `uint16_t rxRingLen`  
*Receive ring length.*

**Field Documentation**

- (1) `volatile enet_rx_bd_struct_t* enet_rx_bd_ring_t::rxBdBase`
- (2) `uint16_t enet_rx_bd_ring_t::rxGenIdx`
- (3) `uint16_t enet_rx_bd_ring_t::rxRingLen`

**20.4.13 struct \_enet\_handle****Data Fields**

- `enet_rx_bd_ring_t rxBdRing` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer descriptor.*
- `enet_tx_bd_ring_t txBdRing` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer descriptor.*
- `uint16_t rxBuffSizeAlign` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer size alignment.*
- `uint16_t txBuffSizeAlign` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer size alignment.*
- `bool rxMaintainEnable` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer cache maintain.*
- `bool txMaintainEnable` [FSL\_FEATURE\_ENET\_QUEUE]

- *Transmit buffer cache maintain.*
- **uint8\_t ringNum**  
*Number of used rings.*
- **enet\_callback\_t callback**  
*Callback function.*
- **void \* userData**  
*Callback function parameter.*
- **enet\_tx\_dirty\_ring\_t txDirtyRing** [FSL\_FEATURE\_ENET\_QUEUE]  
*Ring to store tx frame information.*
- **bool txReclaimEnable** [FSL\_FEATURE\_ENET\_QUEUE]  
*Tx reclaim enable flag.*
- **enet\_rx\_alloc\_callback\_t rxBuffAlloc**  
*Callback function to alloc memory for zero copy Rx.*
- **enet\_rx\_free\_callback\_t rxBuffFree**  
*Callback function to free memory for zero copy Rx.*
- **uint8\_t multicastCount** [64]  
*Multicast collisions counter.*

## Field Documentation

- (1) **enet\_rx\_bd\_ring\_t enet\_handle\_t::rxBdRing** [FSL\_FEATURE\_ENET\_QUEUE]
- (2) **enet\_tx\_bd\_ring\_t enet\_handle\_t::txBdRing** [FSL\_FEATURE\_ENET\_QUEUE]
- (3) **uint16\_t enet\_handle\_t::rxBuffSizeAlign** [FSL\_FEATURE\_ENET\_QUEUE]
- (4) **uint16\_t enet\_handle\_t::txBuffSizeAlign** [FSL\_FEATURE\_ENET\_QUEUE]
- (5) **bool enet\_handle\_t::rxMaintainEnable** [FSL\_FEATURE\_ENET\_QUEUE]
- (6) **bool enet\_handle\_t::txMaintainEnable** [FSL\_FEATURE\_ENET\_QUEUE]
- (7) **uint8\_t enet\_handle\_t::ringNum**
- (8) **enet\_callback\_t enet\_handle\_t::callback**
- (9) **void\* enet\_handle\_t::userData**
- (10) **enet\_tx\_dirty\_ring\_t enet\_handle\_t::txDirtyRing** [FSL\_FEATURE\_ENET\_QUEUE]
- (11) **bool enet\_handle\_t::txReclaimEnable** [FSL\_FEATURE\_ENET\_QUEUE]
- (12) **enet\_rx\_alloc\_callback\_t enet\_handle\_t::rxBuffAlloc**
- (13) **enet\_rx\_free\_callback\_t enet\_handle\_t::rxBuffFree**

## 20.5 Macro Definition Documentation

- 20.5.1 `#define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 9, 1))`
- 20.5.2 `#define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U`
- 20.5.3 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U`
- 20.5.4 `#define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U`
- 20.5.5 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U`
- 20.5.6 `#define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U`
- 20.5.7 `#define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U`
- 20.5.8 `#define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U`
- 20.5.9 `#define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U`
- 20.5.10 `#define ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK 0x0020U`
- 20.5.11 `#define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U`
- 20.5.12 `#define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U`
- 20.5.13 `#define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U`
- 20.5.14 `#define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U`
- 20.5.15 `#define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U`
- 20.5.16 `#define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK 0x4000U`
- 20.5.17 `#define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U`
- 20.5.18 `#define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK 0x1000U`
- 20.5.19 `#define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U`
- 20.5.20 ~~`#define ENET_BUFFDESCRIPTOR_TX_TRANSMITCRC_MASK 0x0400U`~~
- 20.5.21 `#define ENET_BUFFDESCRIPTOR_RX_ERR_MASK`

```
(ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK |
 ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK | \
 ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK |
 ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK |
 ENET_BUFFDESCRIPTOR_RX_CRC_MASK)
```

**20.5.22 #define ENET\_FRAME\_MAX\_FRAMELEN 1518U**

**20.5.23 #define ENET\_FRAME\_VLAN\_TAGLEN 4U**

**20.5.24 #define ENET\_FRAME\_CRC\_LEN 4U**

**20.5.25 #define ENET\_FIFO\_MIN\_RX\_FULL 5U**

**20.5.26 #define ENET\_RX\_MIN\_BUFFERSIZE 256U**

**20.5.27 #define ENET\_PHY\_MAXADDRESS (ENET\_MMFR\_PA\_MASK >>  
ENET\_MMFR\_PA\_SHIFT)**

**20.5.28 #define ENET\_TX\_INTERRUPT ((uint32\_t)kENET\_TxFrameInterrupt |  
(uint32\_t)kENET\_TxBufferInterrupt)**

**20.5.29 #define ENET\_RX\_INTERRUPT ((uint32\_t)kENET\_RxFrameInterrupt |  
(uint32\_t)kENET\_RxBufferInterrupt)**

**20.5.30 #define ENET\_TS\_INTERRUPT ((uint32\_t)kENET\_TsTimerInterrupt |  
(uint32\_t)kENET\_TsAvailInterrupt)**

**20.5.31 #define ENET\_ERR\_INTERRUPT**

**Value:**

```
((uint32_t)kENET_BabrInterrupt | (uint32_t)
 kENET_BabtInterrupt | (uint32_t)kENET_EBusERInterrupt | \
 (uint32_t)kENET_LateCollisionInterrupt | (uint32_t)
 kENET_RetryLimitInterrupt | \
 (uint32_t)kENET_UnderrunInterrupt | (uint32_t)
 kENET_PayloadRxInterrupt)
```

## 20.6 Typedef Documentation

- 20.6.1 `typedef void*(* enet_rx_alloc_callback_t)(ENET_Type *base, void *userData, uint8_t ringId)`
- 20.6.2 `typedef void(* enet_rx_free_callback_t)(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)`
- 20.6.3 `typedef void(* enet_callback_t)(ENET_Type *base, enet_handle_t *handle, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)`
- 20.6.4 `typedef void(* enet_isr_t)(ENET_Type *base, enet_handle_t *handle)`

## 20.7 Enumeration Type Documentation

### 20.7.1 anonymous enum

Enumerator

- kStatus\_ENET\_InitMemoryFail* Init fails since buffer memory is not enough.
- kStatus\_ENET\_RxFrameError* A frame received but data error happen.
- kStatus\_ENET\_RxFrameFail* Failed to receive a frame.
- kStatus\_ENET\_RxFrameEmpty* No frame arrive.
- kStatus\_ENET\_RxFrameDrop* Rx frame is dropped since no buffer memory.
- kStatus\_ENET\_TxFrameOverLen* Tx frame over length.
- kStatus\_ENET\_TxFrameBusy* Tx buffer descriptors are under process.
- kStatus\_ENET\_TxFrameFail* Transmit frame fail.

### 20.7.2 enum enet\_mii\_mode\_t

Enumerator

- kENET\_MiiMode* MII mode for data interface.
- kENET\_RmiiMode* RMII mode for data interface.

### 20.7.3 enum enet\_mii\_speed\_t

Notice: "kENET\_MiiSpeed1000M" only supported when mii mode is "kENET\_RgmiiMode".

Enumerator

- kENET\_MiiSpeed10M* Speed 10 Mbps.
- kENET\_MiiSpeed100M* Speed 100 Mbps.

## 20.7.4 enum enet\_mii\_duplex\_t

Enumerator

*kENET\_MiiHalfDuplex* Half duplex mode.

*kENET\_MiiFullDuplex* Full duplex mode.

## 20.7.5 enum enet\_mii\_write\_t

Enumerator

*kENET\_MiiWriteNoCompliant* Write frame operation, but not MII-compliant.

*kENET\_MiiWriteValidFrame* Write frame operation for a valid MII management frame.

## 20.7.6 enum enet\_mii\_read\_t

Enumerator

*kENET\_MiiReadValidFrame* Read frame operation for a valid MII management frame.

*kENET\_MiiReadNoCompliant* Read frame operation, but not MII-compliant.

## 20.7.7 enum enet\_mii\_extend\_opcode

Enumerator

*kENET\_MiiAddrWrite\_C45* Address Write operation.

*kENET\_MiiWriteFrame\_C45* Write frame operation for a valid MII management frame.

*kENET\_MiiReadFrame\_C45* Read frame operation for a valid MII management frame.

## 20.7.8 enum enet\_special\_control\_flag\_t

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the [enet\\_config\\_t](#). The kENET\_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the [enet\\_config\\_t](#).

Enumerator

*kENET\_ControlFlowControlEnable* Enable ENET flow control: pause frame.

*kENET\_ControlRxPayloadCheckEnable* Enable ENET receive payload length check.

*kENET\_ControlRxPadRemoveEnable* Padding is removed from received frames.

*kENET\_ControlRxBroadCastRejectEnable* Enable broadcast frame reject.

*kENET\_ControlMacAddrInsert* Enable MAC address insert.

*kENET\_ControlStoreAndFwdDisable* Enable FIFO store and forward.

*kENET\_ControlSMIPreambleDisable* Enable SMI preamble.

*kENET\_ControlPromiscuousEnable* Enable promiscuous mode.

*kENET\_ControlMIILoopEnable* Enable ENET MII loop back.

*kENET\_ControlVLANTagEnable* Enable normal VLAN (single vlan tag).

## 20.7.9 enum enet\_interrupt\_enable\_t

This enumeration uses one-bit encoding to allow a logical OR of multiple members. Members usually map to interrupt enable bits in one or more peripheral registers.

Enumerator

*kENET\_BabRInterrupt* Babbling receive error interrupt source.

*kENET\_BabTInterrupt* Babbling transmit error interrupt source.

*kENET\_GraceStopInterrupt* Graceful stop complete interrupt source.

*kENET\_TxFrameInterrupt* TX FRAME interrupt source.

*kENET\_TxBufferInterrupt* TX BUFFER interrupt source.

*kENET\_RxFrameInterrupt* RX FRAME interrupt source.

*kENET\_RxBufferInterrupt* RX BUFFER interrupt source.

*kENET\_MiiInterrupt* MII interrupt source.

*kENET\_EBusERInterrupt* Ethernet bus error interrupt source.

*kENET\_LateCollisionInterrupt* Late collision interrupt source.

*kENET\_RetryLimitInterrupt* Collision Retry Limit interrupt source.

*kENET\_UnderrunInterrupt* Transmit FIFO underrun interrupt source.

*kENET\_PayloadRxInterrupt* Payload Receive error interrupt source.

*kENET\_WakeupInterrupt* WAKEUP interrupt source.

*kENET\_TsAvailInterrupt* TS AVAIL interrupt source for PTP.

*kENET\_TsTimerInterrupt* TS WRAP interrupt source for PTP.

## 20.7.10 enum enet\_event\_t

Enumerator

*kENET\_RxEvent* Receive event.

*kENET\_TxEvent* Transmit event.

*kENET\_ErrEvent* Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .

*kENET\_WakeUpEvent* Wake up from sleep mode event.

*kENET\_TimeStampEvent* Time stamp event.

*kENET\_TimeStampAvailEvent* Time stamp available event.

### 20.7.11 enum enet\_tx\_accelerator\_t

Enumerator

- kENET\_TxAccelIsShift16Enabled*** Transmit FIFO shift-16.
- kENET\_TxAccelIpCheckEnabled*** Insert IP header checksum.
- kENET\_TxAccelProtoCheckEnabled*** Insert protocol checksum.

### 20.7.12 enum enet\_rx\_accelerator\_t

Enumerator

- kENET\_RxAccelPadRemoveEnabled*** Padding removal for short IP frames.
- kENET\_RxAccelIpCheckEnabled*** Discard with wrong IP header checksum.
- kENET\_RxAccelProtoCheckEnabled*** Discard with wrong protocol checksum.
- kENET\_RxAccelMacCheckEnabled*** Discard with Mac layer errors.
- kENET\_RxAccelIsShift16Enabled*** Receive FIFO shift-16.

## 20.8 Function Documentation

### 20.8.1 uint32\_t ENET\_GetInstance ( ENET\_Type \* *base* )

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

Returns

ENET instance.

### 20.8.2 void ENET\_GetDefaultConfig ( enet\_config\_t \* *config* )

The purpose of this API is to get the default ENET MAC controller configure structure for [ENET\\_Init\(\)](#). User may use the initialized structure unchanged in [ENET\\_Init\(\)](#), or modify some fields of the structure before calling [ENET\\_Init\(\)](#). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>config</i> | The ENET mac controller configuration structure pointer. |
|---------------|----------------------------------------------------------|

**20.8.3 status\_t ENET\_Up ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, const enet\_config\_t \* *config*, const enet\_buffer\_config\_t \* *bufferConfig*, uint8\_t \* *macAddr*, uint32\_t *srcClock\_Hz* )**

This function initializes the module with the ENET configuration.

## Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" and calling ENET\_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET\\_Up\(\)](#).

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | ENET peripheral base address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <i>handle</i>       | ENET handler pointer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>config</i>       | ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.                                                                                                                                                                                                                                                                                      |
| <i>bufferConfig</i> | ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer. |

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>macAddr</i>     | ENET mac address of Ethernet device. This MAC address should be provided. |
| <i>srcClock_Hz</i> | The internal module clock source for MII clock.                           |

Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to initialize the ethernet driver.    |
| <i>kStatus_ENET_Init-MemoryFail</i> | Init fails since buffer memory is not enough. |

#### 20.8.4 status\_t ENET\_Init( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, const enet\_config\_t \* *config*, const enet\_buffer\_config\_t \* *bufferConfig*, uint8\_t \* *macAddr*, uint32\_t *srcClock\_Hz* )

This function ungates the module clock and initializes it with the ENET configuration.

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" and calling ENET\_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET\\_Init\(\)](#).

Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | ENET peripheral base address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <i>handle</i>       | ENET handler pointer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>config</i>       | ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.                                                                                                                                                                                                                                                                                      |
| <i>bufferConfig</i> | ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer. |

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>macAddr</i>     | ENET mac address of Ethernet device. This MAC address should be provided. |
| <i>srcClock_Hz</i> | The internal module clock source for MII clock.                           |

Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to initialize the ethernet driver.    |
| <i>kStatus_ENET_Init-MemoryFail</i> | Init fails since buffer memory is not enough. |

## 20.8.5 void ENET\_Down ( ENET\_Type \* *base* )

This function disables the ENET module.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

## 20.8.6 void ENET\_Deinit ( ENET\_Type \* *base* )

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

## 20.8.7 static void ENET\_Reset ( ENET\_Type \* *base* ) [inline], [static]

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

## 20.8.8 void ENET\_ResetHardware ( void )

This function resets ENET related resources in the hardware.

**20.8.9 void ENET\_SetMII ( ENET\_Type \* *base*, enet\_mii\_speed\_t *speed*, enet\_mii\_duplex\_t *duplex* )**

This API is provided to dynamically change the speed and duplex for MAC.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>speed</i>  | The speed of the RMII mode.   |
| <i>duplex</i> | The duplex of the RMII mode.  |

#### 20.8.10 void ENET\_SetSMI( ENET\_Type \* *base*, uint32\_t *srcClock\_Hz*, bool *isPreambleDisabled* )

Parameters

|                           |                                                                                                                                                   |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | ENET peripheral base address.                                                                                                                     |
| <i>srcClock_Hz</i>        | This is the ENET module clock frequency. See clock distribution.                                                                                  |
| <i>isPreambleDisabled</i> | The preamble disable flag. <ul style="list-style-type: none"> <li>• true Enables the preamble.</li> <li>• false Disables the preamble.</li> </ul> |

#### 20.8.11 static bool ENET\_GetSMI( ENET\_Type \* *base* ) [inline], [static]

This API is used to get the SMI configuration to check whether the MII management interface has been set.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

Returns

The SMI setup status true or false.

#### 20.8.12 static uint32\_t ENET\_ReadSMIData( ENET\_Type \* *base* ) [inline], [static]

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

Returns

The data read from PHY

#### 20.8.13 static void ENET\_StartSMIWrite ( ENET\_Type \* *base*, uint8\_t *phyAddr*, uint8\_t *regAddr*, enet\_mii\_write\_t *operation*, uint16\_t *data* ) [inline], [static]

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIOWrite\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

|                  |                                              |
|------------------|----------------------------------------------|
| <i>base</i>      | ENET peripheral base address.                |
| <i>phyAddr</i>   | The PHY address. Range from 0 ~ 31.          |
| <i>regAddr</i>   | The PHY register address. Range from 0 ~ 31. |
| <i>operation</i> | The write operation.                         |
| <i>data</i>      | The data written to PHY.                     |

#### 20.8.14 static void ENET\_StartSMIRead ( ENET\_Type \* *base*, uint8\_t *phyAddr*, uint8\_t *regAddr*, enet\_mii\_read\_t *operation* ) [inline], [static]

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIORRead\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | ENET peripheral base address.       |
| <i>phyAddr</i> | The PHY address. Range from 0 ~ 31. |

|                  |                                              |
|------------------|----------------------------------------------|
| <i>regAddr</i>   | The PHY register address. Range from 0 ~ 31. |
| <i>operation</i> | The read operation.                          |

### 20.8.15 status\_t ENET\_MDIOWrite ( ENET\_Type \* *base*, uint8\_t *phyAddr*, uint8\_t *regAddr*, uint16\_t *data* )

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | ENET peripheral base address.        |
| <i>phyAddr</i> | The PHY address. Range from 0 ~ 31.  |
| <i>regAddr</i> | The PHY register. Range from 0 ~ 31. |
| <i>data</i>    | The data written to PHY.             |

Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

### 20.8.16 status\_t ENET\_MDIORRead ( ENET\_Type \* *base*, uint8\_t *phyAddr*, uint8\_t *regAddr*, uint16\_t \* *pData* )

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | ENET peripheral base address.        |
| <i>phyAddr</i> | The PHY address. Range from 0 ~ 31.  |
| <i>regAddr</i> | The PHY register. Range from 0 ~ 31. |
| <i>pData</i>   | The data read from PHY.              |

Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

### 20.8.17 static void ENET\_StartExtC45SMIWriteReg ( ENET\_Type \* *base*, uint8\_t *portAddr*, uint8\_t *devAddr*, uint16\_t *regAddr* ) [inline], [static]

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIODC45Write\(\)](#)/[ENET\\_MDIODC45Read\(\)](#) can be called. For

customized requirements, implement with combining separated APIs.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |
| <i>regAddr</i>  | The PHY register address.           |

#### 20.8.18 static void ENET\_StartExtC45SMIWriteData ( ENET\_Type \* *base*, uint8\_t *portAddr*, uint8\_t *devAddr*, uint16\_t *data* ) [inline], [static]

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIOC45Write\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |
| <i>data</i>     | The data written to PHY.            |

#### 20.8.19 static void ENET\_StartExtC45SMIReadData ( ENET\_Type \* *base*, uint8\_t *portAddr*, uint8\_t *devAddr* ) [inline], [static]

After calling this function, need to check whether the transmission is over then do next MDIO operation. For ease of use, encapsulated [ENET\\_MDIOC45Read\(\)](#) can be called. For customized requirements, implement with combining separated APIs.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |

#### 20.8.20 status\_t ENET\_MDIOC45Write ( ENET\_Type \* *base*, uint8\_t *portAddr*, uint8\_t *devAddr*, uint16\_t *regAddr*, uint16\_t *data* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |
| <i>regAddr</i>  | The PHY register address.           |
| <i>data</i>     | The data written to PHY.            |

Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

#### 20.8.21 status\_t ENET\_MDIOD45Read ( ENET\_Type \* *base*, uint8\_t *portAddr*, uint8\_t *devAddr*, uint16\_t *regAddr*, uint16\_t \* *pData* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | ENET peripheral base address.       |
| <i>portAddr</i> | The MDIO port address(PHY address). |
| <i>devAddr</i>  | The device address.                 |
| <i>regAddr</i>  | The PHY register address.           |
| <i>pData</i>    | The data read from PHY.             |

Returns

kStatus\_Success MDIO access succeeds.  
kStatus\_Timeout MDIO access timeout.

#### 20.8.22 void ENET\_SetMacAddr ( ENET\_Type \* *base*, uint8\_t \* *macAddr* )

Parameters

|                |                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                     |
| <i>macAddr</i> | The six-byte Mac address pointer. The pointer is allocated by application and input into the API. |

**20.8.23 void ENET\_GetMacAddr ( ENET\_Type \* *base*, uint8\_t \* *macAddr* )**

Parameters

|                |                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                     |
| <i>macAddr</i> | The six-byte Mac address pointer. The pointer is allocated by application and input into the API. |

**20.8.24 void ENET\_AddMulticastGroup ( ENET\_Type \* *base*, uint8\_t \* *address* )**

Parameters

|                |                                                                        |
|----------------|------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                          |
| <i>address</i> | The six-byte multicast group address which is provided by application. |

**20.8.25 void ENET\_LeaveMulticastGroup ( ENET\_Type \* *base*, uint8\_t \* *address* )**

Parameters

|                |                                                                        |
|----------------|------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                          |
| <i>address</i> | The six-byte multicast group address which is provided by application. |

**20.8.26 static void ENET\_ActiveRead ( ENET\_Type \* *base* ) [inline],  
[static]**

This function is to active the enet read process.

Note

This must be called after the MAC configuration and state are ready. It must be called after the [ENET\\_Init\(\)](#). This should be called when the frame reception is required.

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

**20.8.27 static void ENET\_EnableSleepMode ( ENET\_Type \* *base*, bool *enable* )  
[inline], [static]**

This function is used to set the MAC enter sleep mode. When entering sleep mode, the magic frame wakeup interrupt should be enabled to wake up MAC from the sleep mode and reset it to normal mode.

## Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                     |
| <i>enable</i> | True enable sleep mode, false disable sleep mode. |

**20.8.28 static void ENET\_GetAccelFunction ( ENET\_Type \* *base*, uint32\_t \*  
*txAccelOption*, uint32\_t \**rxAccelOption* ) [inline], [static]**

## Parameters

|                      |                                                                                                                                                |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | ENET peripheral base address.                                                                                                                  |
| <i>txAccelOption</i> | The transmit accelerator option. The "enet_tx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option. |
| <i>rxAccelOption</i> | The receive accelerator option. The "enet_rx_accelerator_t" is recommended to be used to as the mask to get the exact the accelerator option.  |

**20.8.29 static void ENET\_EnableInterrupts ( ENET\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See [enet\\_interrupt\\_enable\\_t](#). For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
*     ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt |  
*                           kENET_RxFrameInterrupt);  
*
```

## Parameters

|             |                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                |
| <i>mask</i> | ENET interrupts to enable. This is a logical OR of the enumeration <a href="#">enet_interrupt_enable_t</a> . |

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

This function disables the ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [enet\\_interrupt\\_enable\\_t](#). For example, to disable the TX frame interrupt and RX frame interrupt, do the following.

```
*     ENET_DisableInterrupts(ENET, kENET_TxFrameInterrupt |
    kENET_RxFrameInterrupt);
*
```

## Parameters

|             |                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                 |
| <i>mask</i> | ENET interrupts to disable. This is a logical OR of the enumeration <a href="#">enet_interrupt_enable_t</a> . |

### 20.8.31 static uint32\_t ENET\_GetInterruptStatus ( ENET\_Type \* *base* ) [inline], [static]

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

## Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration [enet\\_interrupt\\_enable\\_t](#).

### 20.8.32 static void ENET\_ClearInterruptStatus ( ENET\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function clears enabled ENET interrupts according to the provided mask. The mask is a logical OR of enumeration members. See the [enet\\_interrupt\\_enable\\_t](#). For example, to clear the TX frame interrupt and RX frame interrupt, do the following.

```
*     ENET_ClearInterruptStatus(ENET,
    kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);
*
```

## Parameters

|             |                                                                                                                                     |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                                       |
| <i>mask</i> | ENET interrupt source to be cleared. This is the logical OR of members of the enumeration <a href="#">enet_interrupt_enable_t</a> . |

**20.8.33 void ENET\_SetRxISRHandler ( ENET\_Type \* *base*, enet\_isr\_t *ISRHandler* )**

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | ENET peripheral base address. |
| <i>ISRHandler</i> | The handler to install.       |

**20.8.34 void ENET\_SetTxISRHandler ( ENET\_Type \* *base*, enet\_isr\_t *ISRHandler* )**

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | ENET peripheral base address. |
| <i>ISRHandler</i> | The handler to install.       |

**20.8.35 void ENET\_SetErrISRHandler ( ENET\_Type \* *base*, enet\_isr\_t *ISRHandler* )**

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | ENET peripheral base address. |
| <i>ISRHandler</i> | The handler to install.       |

**20.8.36 void ENET\_GetRxErrBeforeReadFrame ( enet\_handle\_t \* *handle*, enet\_data\_error\_stats\_t \* *eErrorStatic*, uint8\_t *ringId* )**

This API must be called after the ENET\_GetRxFrameSize and before the [ENET\\_ReadFrame\(\)](#). If the ENET\_GetRxFrameSize returns kStatus\_ENET\_RxFrameError, the ENET\_GetRxErrBeforeReadFrame can be used to get the exact error statistics. This is an example.

```

*     status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*     if (status == kStatus_ENET_RxFrameError)
*     {
*         Comments: Get the error information of the received frame.
*         ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
*         Comments: update the receive buffer.
*         ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
*     }
*

```

## Parameters

|                     |                                                                                             |
|---------------------|---------------------------------------------------------------------------------------------|
| <i>handle</i>       | The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init. |
| <i>eErrorStatic</i> | The error statistics structure pointer.                                                     |
| <i>ringId</i>       | The ring index, range from 0 ~ (FSL_FEATURE_ENET_INSTANCE_QUEUEEn(x) - 1).                  |

**20.8.37 void ENET\_EnableStatistics ( ENET\_Type \* *base*, bool *enable* )**

Note that this function does not reset any of the already collected data, use the function ENET\_ResetStatistics to clear the transfer statistics if needed.

## Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                           |
| <i>enable</i> | True enable statistics collection, false disable statistics collection. |

**20.8.38 void ENET\_GetStatistics ( ENET\_Type \* *base*, enet\_transfer\_stats\_t \* *statistics* )**

Copies the actual value of hardware counters into the provided structure. Calling this function does not reset the counters in hardware.

## Parameters

|                   |                                   |
|-------------------|-----------------------------------|
| <i>base</i>       | ENET peripheral base address.     |
| <i>statistics</i> | The statistics structure pointer. |

**20.8.39 void ENET\_ResetStatistics ( ENET\_Type \* *base* )**

Sets the value of hardware transfer counters to zero.

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

**20.8.40 status\_t ENET\_GetRxFrameSize ( enet\_handle\_t \* *handle*, uint32\_t \* *length*, uint8\_t *ringId* )**

This function gets a received frame size from the ENET buffer descriptors.

## Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET\_GetRxFrameSize, [ENET\\_ReadFrame\(\)](#) should be called to receive frame and update the BD if the result is not "kStatus\_ENET\_RxFrameEmpty".

## Parameters

|               |                                                                                     |
|---------------|-------------------------------------------------------------------------------------|
| <i>handle</i> | The ENET handler structure. This is the same handler pointer used in the ENET_Init. |
| <i>length</i> | The length of the valid frame received.                                             |
| <i>ringId</i> | The ring index or ring number.                                                      |

## Return values

|                                  |                                                                                                                                      |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_ENET_RxFrameEmpty</i> | No frame received. Should not call ENET_ReadFrame to read frame.                                                                     |
| <i>kStatus_ENET_RxFrameError</i> | Data error happens. ENET_ReadFrame should be called with NULL data and NULL length to update the receive buffers.                    |
| <i>kStatus_Success</i>           | Receive a frame Successfully then the ENET_ReadFrame should be called with the right data buffer and the captured data length input. |

**20.8.41 status\_t ENET\_ReadFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint8\_t \* *data*, uint32\_t *length*, uint8\_t *ringId*, uint32\_t \* *ts* )**

This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through *ts* pointer if the *ts* is not NULL.

## Note

It doesn't store the timestamp in the receive timestamp queue. The ENET\_GetRxFrameSize should be used to get the size of the prepared data buffer. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption. This is an example:

```

*      uint32_t length;
*      enet_handle_t g_handle;
*      Comments: Get the received frame size firstly.
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (length != 0)
*      {
*          Comments: Allocate memory here with the size of "length"
*          uint8_t *data = memory allocate interface;
*          if (!data)
*          {
*              ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
*              Comments: Add the console warning log.
*          }
*          else
*          {
*              status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
*              Comments: Call stack input API to deliver the data to stack
*          }
*      }
*      else if (status == kStatus_ENET_RxFrameError)
*      {
*          Comments: Update the received buffer when a error frame is received.
*          ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
*      }
*

```

## Parameters

|               |                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                                                      |
| <i>handle</i> | The ENET handler structure. This is the same handler pointer used in the ENET_Init.                |
| <i>data</i>   | The data buffer provided by user to store the frame which memory size should be at least "length". |
| <i>length</i> | The size of the data buffer which is still the length of the received frame.                       |
| <i>ringId</i> | The ring index or ring number.                                                                     |
| <i>ts</i>     | The timestamp address to store received timestamp.                                                 |

## Returns

The execute status, successful or failure.

#### 20.8.42 status\_t ENET\_SendFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, const uint8\_t \* *data*, uint32\_t *length*, uint8\_t *ringId*, bool *tsFlag*, void \* *context* )

## Note

The CRC is automatically appended to the data. Input the data to send without the CRC. This API uses memcpy to copy data from DMA buffer to application buffer, 4 bytes aligned data buffer in 32 bits platforms provided by user may let compiler use optimization instruction to reduce time consumption.

## Parameters

|                |                                                                                   |
|----------------|-----------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                     |
| <i>handle</i>  | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>data</i>    | The data buffer provided by user to send.                                         |
| <i>length</i>  | The length of the data to send.                                                   |
| <i>ringId</i>  | The ring index or ring number.                                                    |
| <i>tsFlag</i>  | Timestamp enable flag.                                                            |
| <i>context</i> | Used by user to handle some events after transmit over.                           |

## Return values

|                                 |                                                                                                                                                                                                                                                   |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>          | Send frame succeed.                                                                                                                                                                                                                               |
| <i>kStatus_ENET_TxFrameBusy</i> | Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with <i>kStatus_ENET_TxFrameBusy</i> . |

### 20.8.43 status\_t ENET\_SetTxReclaim ( *enet\_handle\_t \* handle, bool isEnabled, uint8\_t ringId* )

## Note

This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

## Parameters

|                  |                                                                                   |
|------------------|-----------------------------------------------------------------------------------|
| <i>handle</i>    | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>isEnabled</i> | Enable or disable flag.                                                           |
| <i>ringId</i>    | The ring index or ring number.                                                    |

Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Succeed to enable/disable Tx reclaim. |
| <i>kStatus_Fail</i>    | Fail to enable/disable Tx reclaim.    |

#### 20.8.44 void ENET\_ReclaimTxDescriptor ( **ENET\_Type** \* *base*, **enet\_handle\_t** \* *handle*, **uint8\_t** *ringId* )

This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.

Parameters

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                                     |
| <i>handle</i> | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>ringId</i> | The ring index or ring number.                                                    |

#### 20.8.45 **status\_t** ENET\_GetRxFrame ( **ENET\_Type** \* *base*, **enet\_handle\_t** \* *handle*, **enet\_rx\_frame\_struct\_t** \* *rxFrame*, **uint8\_t** *ringId* )

This function uses the user-defined allocation and free callbacks. Every time application gets one frame through this function, driver stores the buffer address(es) in enet\_buffer\_struct\_t and allocate new buffer(s) for the BD(s). If there's no memory buffer in the pool, this function drops current one frame to keep the Rx frame in BD ring is as fresh as possible.

Note

Application must provide a memory pool including at least BD number + n buffers in order for this function to work properly, because each BD must always take one buffer while driver is running, then other extra n buffer(s) can be taken by application. Here n is the ceil(max\_frame\_length(set by RCR) / bd\_rx\_size(set by MRBR)). Application must also provide an array structure in rxFrame->rxBuffArray with n index to receive one complete frame in any case.

Parameters

|                |                                                                                   |
|----------------|-----------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                     |
| <i>handle</i>  | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>rxFrame</i> | The received frame information structure provided by user.                        |
| <i>ringId</i>  | The ring index or ring number.                                                    |

Return values

|                                   |                                                                 |
|-----------------------------------|-----------------------------------------------------------------|
| <i>kStatus_Success</i>            | Succeed to get one frame and allocate new memory for Rx buffer. |
| <i>kStatus_ENET_RxFrame-Empty</i> | There's no Rx frame in the BD.                                  |
| <i>kStatus_ENET_RxFrame-Error</i> | There's issue in this receiving.                                |
| <i>kStatus_ENET_RxFrame-Drop</i>  | There's no new buffer memory for BD, drop this frame.           |

#### 20.8.46 status\_t ENET\_StartTxFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_tx\_frame\_struct\_t \* *txFrame*, uint8\_t *ringId* )

This function supports scattered buffer transmit, user needs to provide the buffer array.

Note

Tx reclaim should be enabled to ensure the Tx buffer ownership can be given back to application after Tx is over.

Parameters

|                |                                                                                   |
|----------------|-----------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                     |
| <i>handle</i>  | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>txFrame</i> | The Tx frame structure.                                                           |
| <i>ringId</i>  | The ring index or ring number.                                                    |

Return values

|                                     |                                                                         |
|-------------------------------------|-------------------------------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed to send one frame.                                              |
| <i>kStatus_ENET_TxFrame-Busy</i>    | The BD is not ready for Tx or the reclaim operation still not finishes. |
| <i>kStatus_ENET_TxFrame-OverLen</i> | The Tx frame length is over max ethernet frame length.                  |

#### 20.8.47 void ENET\_TransmitIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |

#### 20.8.48 void ENET\_ReceiveIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |

#### 20.8.49 void ENET\_ErrorIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |

#### 20.8.50 void ENET\_Ptp1588IRQHandler ( ENET\_Type \* *base* )

This is used for the 1588 timer interrupt.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

#### 20.8.51 void ENET\_CommonFrame0IRQHandler ( ENET\_Type \* *base* )

This is used for the combined tx/rx/error interrupt for single/mutli-ring (frame 0).

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

## 20.9 Variable Documentation

### 20.9.1 `const clock_ip_name_t s_enetClock[]`

# Chapter 21

## FLEXCOMM: FLEXCOMM Driver

### 21.1 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](#)

## 21.2 FLEXCOMM Driver

### 21.2.1 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.*

## 21.2.2 Macro Definition Documentation

21.2.2.1 `#define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

## 21.2.3 Typedef Documentation

21.2.3.1 `typedef void(* flexcomm_irq_handler_t)(void *base, void *handle)`

## 21.2.4 Enumeration Type Documentation

21.2.4.1 `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.

## 21.2.5 Function Documentation

21.2.5.1 `uint32_t FLEXCOMM_GetInstance ( void * base )`

21.2.5.2 `status_t FLEXCOMM_Init ( void * base, FLEXCOMM_PERIPH_T periph )`

21.2.5.3 `void FLEXCOMM_SetIRQHandler ( void * base, flexcomm_irq_handler_t handler, void * flexcommHandle )`

It is used by drivers register IRQ handler according to FLEXCOMM mode

## 21.2.6 Variable Documentation

21.2.6.1 `IRQn_Type const kFlexcommIrqs[]`

## Chapter 22

# FLEXSPI: Flexible Serial Peripheral Interface Driver

### 22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flexible Serial Peripheral Interface (FLEXSPI) module of MCUXpresso SDK/i.MX devices.

FLEXSPI driver includes functional APIs and interrupt/EDMA non-blocking transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FLEXSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FLEXSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FLEXSPI 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 `flexspi_handle_t`/`flexspi_edma_handle_t` as the second parameter. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandle` API. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandleEDMA` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXSPI_TransferNonBlocking()` and `FLEXSPI_TransferEDMA()` set up data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXSPI_Idle` status.

### Data Structures

- struct `flexspi_config_t`  
*FLEXSPI configuration structure. [More...](#)*
- struct `flexspi_device_config_t`  
*External device configuration items. [More...](#)*
- struct `flexspi_transfer_t`  
*Transfer structure for FLEXSPI. [More...](#)*
- struct `flexspi_handle_t`  
*Transfer handle structure for FLEXSPI. [More...](#)*

### Macros

- #define `FLEXSPI_LUT_SEQ`(cmd0, pad0, op0, cmd1, pad1, op1)  
*Formula to form FLEXSPI instructions in LUT table.*

## Typedefs

- `typedef void(* flexspi_transfer_callback_t )(FLEXSPI_Type *base, flexspi_handle_t *handle, status_t status, void *userData)`  
*FLEXSPI transfer callback function.*

## Enumerations

- `enum {`  
 `kStatus_FLEXSPI_Busy = MAKE_STATUS(kStatusGroup_FLEXSPI, 0),`  
 `kStatus_FLEXSPI_SequenceExecutionTimeout = MAKE_STATUS(kStatusGroup_FLEXSPI, 1),`  
 `kStatus_FLEXSPI_IpCommandSequenceError = MAKE_STATUS(kStatusGroup_FLEXSPI, 2),`  
 `kStatus_FLEXSPI_IpCommandGrantTimeout = MAKE_STATUS(kStatusGroup_FLEXSPI, 3) }`  
*Status structure of FLEXSPI.*
- `enum {`  
 `kFLEXSPI_Command_STOP = 0x00U,`  
 `kFLEXSPI_Command_SDR = 0x01U,`  
 `kFLEXSPI_Command_RADDR_SDR = 0x02U,`  
 `kFLEXSPI_Command_CADDR_SDR = 0x03U,`  
 `kFLEXSPI_Command_MODE1_SDR = 0x04U,`  
 `kFLEXSPI_Command_MODE2_SDR = 0x05U,`  
 `kFLEXSPI_Command_MODE4_SDR = 0x06U,`  
 `kFLEXSPI_Command_MODE8_SDR = 0x07U,`  
 `kFLEXSPI_Command_WRITE_SDR = 0x08U,`  
 `kFLEXSPI_Command_READ_SDR = 0x09U,`  
 `kFLEXSPI_Command_LEARN_SDR = 0x0AU,`  
 `kFLEXSPI_Command_DATSZ_SDR = 0x0BU,`  
 `kFLEXSPI_Command_DUMMY_SDR = 0x0CU,`  
 `kFLEXSPI_Command_DUMMY_RWDS_SDR = 0x0DU,`  
 `kFLEXSPI_Command_DDR = 0x21U,`  
 `kFLEXSPI_Command_RADDR_DDR = 0x22U,`  
 `kFLEXSPI_Command_CADDR_DDR = 0x23U,`  
 `kFLEXSPI_Command_MODE1_DDR = 0x24U,`  
 `kFLEXSPI_Command_MODE2_DDR = 0x25U,`  
 `kFLEXSPI_Command_MODE4_DDR = 0x26U,`  
 `kFLEXSPI_Command_MODE8_DDR = 0x27U,`  
 `kFLEXSPI_Command_WRITE_DDR = 0x28U,`  
 `kFLEXSPI_Command_READ_DDR = 0x29U,`  
 `kFLEXSPI_Command_LEARN_DDR = 0x2AU,`  
 `kFLEXSPI_Command_DATSZ_DDR = 0x2BU,`  
 `kFLEXSPI_Command_DUMMY_DDR = 0x2CU,`  
 `kFLEXSPI_Command_DUMMY_RWDS_DDR = 0x2DU,`  
 `kFLEXSPI_Command_JUMP_ON_CS = 0x1FU }`  
*CMD definition of FLEXSPI, use to form LUT instruction, \_flexspi\_command.*
- `enum flexspi_pad_t {`

```
kFLEXSPI_1PAD = 0x00U,
kFLEXSPI_2PAD = 0x01U,
kFLEXSPI_4PAD = 0x02U,
kFLEXSPI_8PAD = 0x03U }
```

*pad definition of FLEXSPI, use to form LUT instruction.*

- enum `flexspi_flags_t` {
   
kFLEXSPI\_SequenceExecutionTimeoutFlag = FLEXSPI\_INTEN\_SEQTIMEOUTEN\_MASK,
 kFLEXSPI\_AhbBusTimeoutFlag = FLEXSPI\_INTEN\_AHBBUSTIMEOUTEN\_MASK,
 kFLEXSPI\_SckStoppedBecauseTxEmptyFlag,
 kFLEXSPI\_SckStoppedBecauseRxFullFlag,
 kFLEXSPI\_DataLearningFailedFlag = FLEXSPI\_INTEN\_DATALEARNFAILEN\_MASK,
 kFLEXSPI\_IpTxFifoWatermarkEmptyFlag = FLEXSPI\_INTEN\_IPTXWEEN\_MASK,
 kFLEXSPI\_IpRxFifoWatermarkAvailableFlag = FLEXSPI\_INTEN\_IPRXWAEN\_MASK,
 kFLEXSPI\_AhbCommandSequenceErrorFlag,
 kFLEXSPI\_IpCommandSequenceErrorFlag = FLEXSPI\_INTEN\_IPCMDERREN\_MASK,
 kFLEXSPI\_AhbCommandGrantTimeoutFlag,
 kFLEXSPI\_IpCommandGrantTimeoutFlag,
 kFLEXSPI\_IpCommandExecutionDoneFlag,
 kFLEXSPI\_AllInterruptFlags = 0xFFFFU }

*FLEXSPI interrupt status flags.*

- enum `flexspi_read_sample_clock_t` {
   
kFLEXSPI\_ReadSampleClkLoopbackInternally = 0x0U,
 kFLEXSPI\_ReadSampleClkLoopbackFromDqsPad = 0x1U,
 kFLEXSPI\_ReadSampleClkLoopbackFromSckPad = 0x2U,
 kFLEXSPI\_ReadSampleClkExternalInputFromDqsPad = 0x3U }

*FLEXSPI sample clock source selection for Flash Reading.*

- enum `flexspi_cs_interval_cycle_unit_t` {
   
kFLEXSPI\_CsIntervalUnit1SckCycle = 0x0U,
 kFLEXSPI\_CsIntervalUnit256SckCycle = 0x1U }

*FLEXSPI interval unit for flash device select.*

- enum `flexspi_ahb_write_wait_unit_t` {
   
kFLEXSPI\_AhbWriteWaitUnit2AhbCycle = 0x0U,
 kFLEXSPI\_AhbWriteWaitUnit8AhbCycle = 0x1U,
 kFLEXSPI\_AhbWriteWaitUnit32AhbCycle = 0x2U,
 kFLEXSPI\_AhbWriteWaitUnit128AhbCycle = 0x3U,
 kFLEXSPI\_AhbWriteWaitUnit512AhbCycle = 0x4U,
 kFLEXSPI\_AhbWriteWaitUnit2048AhbCycle = 0x5U,
 kFLEXSPI\_AhbWriteWaitUnit8192AhbCycle = 0x6U,
 kFLEXSPI\_AhbWriteWaitUnit32768AhbCycle = 0x7U }

*FLEXSPI AHB wait interval unit for writing.*

- enum `flexspi_ip_error_code_t` {

```

kFLEXSPI_IpCmdErrorNoError = 0x0U,
kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd = 0x2U,
kFLEXSPI_IpCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_IpCmdErrorInvalidAddress = 0x6U,
kFLEXSPI_IpCmdErrorSequenceExecutionTimeout = 0xEU,
kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss = 0xFU }

```

*Error Code when IP command Error detected.*

- enum `flexspi_ahb_error_code_t` {
 

```

kFLEXSPI_AhbCmdErrorNoError = 0x0U,
kFLEXSPI_AhbCmdErrorJumpOnCsInWriteCmd = 0x2U,
kFLEXSPI_AhbCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_AhbCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_AhbCmdSequenceExecutionTimeout = 0x6U }
```

*Error Code when AHB command Error detected.*

- enum `flexspi_port_t` {
 

```

kFLEXSPI_PortA1 = 0x0U,
kFLEXSPI_PortA2,
kFLEXSPI_PortB1,
kFLEXSPI_PortB2 }
```
- enum `flexspi_arb_command_source_t`

*FLEXSPI operation port select.*
- enum `flexspi_command_type_t`

*Trigger source of current command sequence granted by arbitrator.*

```

kFLEXSPI_Command,
kFLEXSPI_Config }
```

*Command type.*

## Driver version

- #define `FSL_FLEXSPI_DRIVER_VERSION` (`MAKE_VERSION(2, 6, 0)`)  
*FLEXSPI driver version.*

## Initialization and deinitialization

- `uint32_t FLEXSPIGetInstance (FLEXSPI_Type *base)`  
*Get the instance number for FLEXSPI.*
- `status_t FLEXSPI_CheckAndClearError (FLEXSPI_Type *base, uint32_t status)`  
*Check and clear IP command execution errors.*
- `void FLEXSPI_Init (FLEXSPI_Type *base, const flexspi_config_t *config)`  
*Initializes the FLEXSPI module and internal state.*
- `void FLEXSPI_GetDefaultConfig (flexspi_config_t *config)`  
*Gets default settings for FLEXSPI.*
- `void FLEXSPI_Deinit (FLEXSPI_Type *base)`  
*Deinitializes the FLEXSPI module.*

- void **FLEXSPI\_UpdateDllValue** (FLEXSPI\_Type \*base, **flexspi\_device\_config\_t** \*config, **flexspi\_port\_t** port)  
*Update FLEXSPI DLL value depending on currently flexspi root clock.*
- void **FLEXSPI\_SetFlashConfig** (FLEXSPI\_Type \*base, **flexspi\_device\_config\_t** \*config, **flexspi\_port\_t** port)  
*Configures the connected device parameter.*
- static void **FLEXSPI\_SoftwareReset** (FLEXSPI\_Type \*base)  
*Software reset for the FLEXSPI logic.*
- static void **FLEXSPI\_Enable** (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables the FLEXSPI module.*

## Interrupts

- static void **FLEXSPI\_EnableInterrupts** (FLEXSPI\_Type \*base, uint32\_t mask)  
*Enables the FLEXSPI interrupts.*
- static void **FLEXSPI\_DisableInterrupts** (FLEXSPI\_Type \*base, uint32\_t mask)  
*Disable the FLEXSPI interrupts.*

## DMA control

- static void **FLEXSPI\_EnableTxDMA** (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables FLEXSPI IP Tx FIFO DMA requests.*
- static void **FLEXSPI\_EnableRxDMA** (FLEXSPI\_Type \*base, bool enable)  
*Enables or disables FLEXSPI IP Rx FIFO DMA requests.*
- static uint32\_t **FLEXSPI\_GetTxFifoAddress** (FLEXSPI\_Type \*base)  
*Gets FLEXSPI IP tx fifo address for DMA transfer.*
- static uint32\_t **FLEXSPI\_GetRxFifoAddress** (FLEXSPI\_Type \*base)  
*Gets FLEXSPI IP rx fifo address for DMA transfer.*

## FIFO control

- static void **FLEXSPI\_ResetFifos** (FLEXSPI\_Type \*base, bool txFifo, bool rxFifo)  
*Clears the FLEXSPI IP FIFO logic.*
- static void **FLEXSPI\_GetFifoCounts** (FLEXSPI\_Type \*base, size\_t \*txCount, size\_t \*rxCount)  
*Gets the valid data entries in the FLEXSPI FIFOs.*

## Status

- static uint32\_t **FLEXSPI\_GetInterruptStatusFlags** (FLEXSPI\_Type \*base)  
*Get the FLEXSPI interrupt status flags.*
- static void **FLEXSPI\_ClearInterruptStatusFlags** (FLEXSPI\_Type \*base, uint32\_t mask)  
*Get the FLEXSPI interrupt status flags.*
- static void **FLEXSPI\_GetDataLearningPhase** (FLEXSPI\_Type \*base, uint8\_t \*portAPhase, uint8\_t \*portBPhase)  
*Gets the sampling clock phase selection after Data Learning.*
- static **flexspi\_arb\_command\_source\_t** **FLEXSPI\_GetArbitratorCommandSource** (FLEXSPI\_Type \*base)  
*Gets the trigger source of current command sequence granted by arbitrator.*
- static **flexspi\_ip\_error\_code\_t** **FLEXSPI\_GetIPCommandErrorCode** (FLEXSPI\_Type \*base, uint8\_t \*index)

- Gets the error code when IP command error detected.
- static `flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode` (`FLEXSPI_Type *base, uint8_t *index`)  
Gets the error code when AHB command error detected.
- static bool `FLEXSPI_GetBusIdleStatus` (`FLEXSPI_Type *base`)  
Returns whether the bus is idle.

## Bus Operations

- void `FLEXSPI_UpdateRxSampleClock` (`FLEXSPI_Type *base, flexspi_read_sample_clock_t clockSource`)  
*Update read sample clock source.*
- void `FLEXSPI_UpdateLUT` (`FLEXSPI_Type *base, uint32_t index, const uint32_t *cmd, uint32_t count`)  
*Updates the LUT table.*
- static void `FLEXSPI_WriteData` (`FLEXSPI_Type *base, uint32_t data, uint8_t fifoIndex`)  
*Writes data into FIFO.*
- static uint32\_t `FLEXSPI_ReadData` (`FLEXSPI_Type *base, uint8_t fifoIndex`)  
*Receives data from data FIFO.*
- status\_t `FLEXSPI_WriteBlocking` (`FLEXSPI_Type *base, uint8_t *buffer, size_t size`)  
*Sends a buffer of data bytes using blocking method.*
- status\_t `FLEXSPI_ReadBlocking` (`FLEXSPI_Type *base, uint8_t *buffer, size_t size`)  
*Receives a buffer of data bytes using a blocking method.*
- status\_t `FLEXSPI_TransferBlocking` (`FLEXSPI_Type *base, flexspi_transfer_t *xfer`)  
*Execute command to transfer a buffer data bytes using a blocking method.*

## Transactional

- void `FLEXSPI_TransferCreateHandle` (`FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_callback_t callback, void *userData`)  
*Initializes the FLEXSPI handle which is used in transactional functions.*
- status\_t `FLEXSPI_TransferNonBlocking` (`FLEXSPI_Type *base, flexspi_handle_t *handle, flexspi_transfer_t *xfer`)  
*Performs a interrupt non-blocking transfer on the FLEXSPI bus.*
- status\_t `FLEXSPI_TransferGetCount` (`FLEXSPI_Type *base, flexspi_handle_t *handle, size_t *count`)  
*Gets the master transfer status during a interrupt non-blocking transfer.*
- void `FLEXSPI_TransferAbort` (`FLEXSPI_Type *base, flexspi_handle_t *handle`)  
*Aborts an interrupt non-blocking transfer early.*
- void `FLEXSPI_TransferHandleIRQ` (`FLEXSPI_Type *base, flexspi_handle_t *handle`)  
*Master interrupt handler.*

## 22.2 Data Structure Documentation

### 22.2.1 struct `flexspi_config_t`

#### Data Fields

- `flexspi_read_sample_clock_t rxSampleClock`

- *Sample Clock source selection for Flash Reading.*
- **bool enableSckFreeRunning**  
*Enable/disable SCK output free-running.*
- **bool enableCombination**  
*Enable/disable combining PORT A and B Data Pins  
(SIOA[3:0] and SIOB[3:0]) to support Flash Octal mode.*
- **bool enableDoze**  
*Enable/disable doze mode support.*
- **bool enableHalfSpeedAccess**  
*Enable/disable divide by 2 of the clock for half speed commands.*
- **flexspi\_read\_sample\_clock\_t rxSampleClockPortB**  
*Sample Clock source\_b selection for Flash Reading.*
- **bool rxSampleClockDiff**  
*Sample Clock source or source\_b selection for Flash Reading.*
- **bool enableSckBDiffOpt**  
*Enable/disable SCKB pad use as SCKA differential clock output, when enable, Port B flash access is not available.*
- **bool enableSameConfigForAll**  
*Enable/disable same configuration for all connected devices when enabled, same configuration in FLASHA1CRx is applied to all.*
- **uint16\_t seqTimeoutCycle**  
*Timeout wait cycle for command sequence execution, timeout after ahbGrantTimeoutCycle\*1024 serial root clock cycles.*
- **uint8\_t ipGrantTimeoutCycle**  
*Timeout wait cycle for IP command grant, timeout after ipGrantTimeoutCycle\*1024 AHB clock cycles.*
- **uint8\_t txWatermark**  
*FLEXSPI IP transmit watermark value.*
- **uint8\_t rxWatermark**  
*FLEXSPI receive watermark value.*
- **bool enableAHBWriteIpTxFifo**  
*Enable AHB bus write access to IP TX FIFO.*
- **bool enableAHBWriteIpRxFifo**  
*Enable AHB bus write access to IP RX FIFO.*
- **uint8\_t ahbGrantTimeoutCycle**  
*Timeout wait cycle for AHB command grant, timeout after ahbGrantTimeoutCycle\*1024 AHB clock cycles.*
- **uint16\_t ahbBusTimeoutCycle**  
*Timeout wait cycle for AHB read/write access, timeout after ahbBusTimeoutCycle\*1024 AHB clock cycles.*
- **uint8\_t resumeWaitCycle**  
*Wait cycle for idle state before suspended command sequence resume, timeout after ahbBusTimeoutCycle AHB clock cycles.*
- **flexspi\_ahbBuffer\_config\_t buffer [FSL\_FEATURE\_FLEXSPI\_AHB\_BUFFER\_COUNT]**  
*AHB buffer size.*
- **bool enableClearAHBBufferOpt**  
*Enable/disable automatically clean AHB RX Buffer and TX Buffer when FLEXSPI returns STOP mode ACK.*
- **bool enableReadAddressOpt**  
*Enable/disable remove AHB read burst start address alignment limitation.*
- **bool enableAHBPrefetch**

Enable/disable AHB read prefetch feature, when enabled, FLEXSPI  
*will fetch more data than current AHB burst.*

- bool **enableAHBBufferable**

Enable/disable AHB bufferable write access support, when enabled,  
*FLEXSPI return before waiting for command execution finished.*

- bool **enableAHBCachable**

*Enable AHB bus cachable read access support.*

## Field Documentation

- (1) `flexspi_read_sample_clock_t flexspi_config_t::rxSampleClock`
- (2) `bool flexspi_config_t::enableSckFreeRunning`
- (3) `bool flexspi_config_t::enableCombination`
- (4) `bool flexspi_config_t::enableDoze`
- (5) `bool flexspi_config_t::enableHalfSpeedAccess`
- (6) `flexspi_read_sample_clock_t flexspi_config_t::rxSampleClockPortB`
- (7) `bool flexspi_config_t::rxSampleClockDiff`
- (8) `bool flexspi_config_t::enableSckBDiffOpt`
- (9) `bool flexspi_config_t::enableSameConfigForAll`
- (10) `uint16_t flexspi_config_t::seqTimeoutCycle`
- (11) `uint8_t flexspi_config_t::ipGrantTimeoutCycle`
- (12) `uint8_t flexspi_config_t::txWatermark`
- (13) `uint8_t flexspi_config_t::rxWatermark`
- (14) `bool flexspi_config_t::enableAHBWriteIpTxFifo`
- (15) `bool flexspi_config_t::enableAHBWriteIpRxFifo`
- (16) `uint8_t flexspi_config_t::ahbGrantTimeoutCycle`
- (17) `uint16_t flexspi_config_t::ahbBusTimeoutCycle`
- (18) `uint8_t flexspi_config_t::resumeWaitCycle`
- (19) `flexspi_ahbBuffer_config_t flexspi_config_t::buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT]`
- (20) `bool flexspi_config_t::enableClearAHBBufferOpt`
- (21) `bool flexspi_config_t::enableReadAddressOpt`

when enable, there is no AHB read burst start address alignment limitation.

- (22) **bool flexspi\_config\_t::enableAHBPrefetch**
- (23) **bool flexspi\_config\_t::enableAHBBufferable**
- (24) **bool flexspi\_config\_t::enableAHBCachable**

## 22.2.2 struct flexspi\_device\_config\_t

### Data Fields

- **uint32\_t flexspiRootClk**  
*FLEXSPI serial root clock.*
- **bool isSck2Enabled**  
*FLEXSPI use SCK2.*
- **uint32\_t flashSize**  
*Flash size in KByte.*
- **bool addressShift**  
*Address shift.*
- **flexspi\_cs\_interval\_cycle\_unit\_t CSIntervalUnit**  
*CS interval unit, 1 or 256 cycle.*
- **uint16\_t CSInterval**  
*CS line assert interval, multiply CS interval unit to get the CS line assert interval cycles.*
- **uint8\_t CSHoldTime**  
*CS line hold time.*
- **uint8\_t CSSetupTime**  
*CS line setup time.*
- **uint8\_t dataValidTime**  
*Data valid time for external device.*
- **uint8\_t columnspace**  
*Column space size.*
- **bool enableWordAddress**  
*If enable word address.*
- **uint8\_t AWRSeqIndex**  
*Sequence ID for AHB write command.*
- **uint8\_t AWRSeqNumber**  
*Sequence number for AHB write command.*
- **uint8\_t ARDSeqIndex**  
*Sequence ID for AHB read command.*
- **uint8\_t ARDSeqNumber**  
*Sequence number for AHB read command.*
- **flexspi\_ahb\_write\_wait\_unit\_t AHBWriteWaitUnit**  
*AHB write wait unit.*
- **uint16\_t AHBWriteWaitInterval**  
*AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.*
- **bool enableWriteMask**  
*Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.*

**Field Documentation**

- (1) **uint32\_t flexspi\_device\_config\_t::flexspiRootClk**
- (2) **bool flexspi\_device\_config\_t::isSck2Enabled**
- (3) **uint32\_t flexspi\_device\_config\_t::flashSize**
- (4) **bool flexspi\_device\_config\_t::addressShift**
- (5) **flexspi\_cs\_interval\_cycle\_unit\_t flexspi\_device\_config\_t::CSIntervalUnit**
- (6) **uint16\_t flexspi\_device\_config\_t::CSInterval**
- (7) **uint8\_t flexspi\_device\_config\_t::CSHoldTime**
- (8) **uint8\_t flexspi\_device\_config\_t::CSSetupTime**
- (9) **uint8\_t flexspi\_device\_config\_t::dataValidTime**
- (10) **uint8\_t flexspi\_device\_config\_t::columnspace**
- (11) **bool flexspi\_device\_config\_t::enableWordAddress**
- (12) **uint8\_t flexspi\_device\_config\_t::AWRSeqIndex**
- (13) **uint8\_t flexspi\_device\_config\_t::AWRSeqNumber**
- (14) **uint8\_t flexspi\_device\_config\_t::ARDSeqIndex**
- (15) **uint8\_t flexspi\_device\_config\_t::ARDSeqNumber**
- (16) **flexspi\_ahb\_write\_wait\_unit\_t flexspi\_device\_config\_t::AHBWriteWaitUnit**
- (17) **uint16\_t flexspi\_device\_config\_t::AHBWriteWaitInterval**
- (18) **bool flexspi\_device\_config\_t::enableWriteMask**

**22.2.3 struct flexspi\_transfer\_t****Data Fields**

- **uint32\_t deviceAddress**  
*Operation device address.*
- **flexspi\_port\_t port**  
*Operation port.*
- **flexspi\_command\_type\_t cmdType**  
*Execution command type.*
- **uint8\_t seqIndex**  
*Sequence ID for command.*

- `uint8_t SeqNumber`  
*Sequence number for command.*
- `uint32_t * data`  
*Data buffer.*
- `size_t dataSize`  
*Data size in bytes.*

**Field Documentation**

- (1) `uint32_t flexspi_transfer_t::deviceAddress`
- (2) `flexspi_port_t flexspi_transfer_t::port`
- (3) `flexspi_command_type_t flexspi_transfer_t::cmdType`
- (4) `uint8_t flexspi_transfer_t::seqIndex`
- (5) `uint8_t flexspi_transfer_t::SeqNumber`
- (6) `uint32_t* flexspi_transfer_t::data`
- (7) `size_t flexspi_transfer_t::dataSize`

**22.2.4 struct \_flexspi\_handle****Data Fields**

- `uint32_t state`  
*Internal state for FLEXSPI transfer.*
- `uint8_t * data`  
*Data buffer.*
- `size_t dataSize`  
*Remaining Data size in bytes.*
- `size_t transferTotalSize`  
*Total Data size in bytes.*
- `flexspi_transfer_callback_t completionCallback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`  
*FLEXSPI callback function parameter.*

**Field Documentation**

- (1) `uint8_t* flexspi_handle_t::data`
- (2) `size_t flexspi_handle_t::dataSize`
- (3) `size_t flexspi_handle_t::transferTotalSize`
- (4) `void* flexspi_handle_t::userData`

## 22.3 Macro Definition Documentation

**22.3.1 #define FSL\_FLEXSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))**

**22.3.2 #define FLEXSPI\_LUT\_SEQ( cmd0, pad0, op0, cmd1, pad1, op1 )**

**Value:**

```
(FLEXSPI_LUT_OPERAND0 (op0) | FLEXSPI_LUT_NUM_PADS0 (pad0) | FLEXSPI_LUT_OPCODE0 (cmd0) | FLEXSPI_LUT_OPERAND1
  (op1) | \
  FLEXSPI_LUT_NUM_PADS1 (pad1) | FLEXSPI_LUT_OPCODE1 (cmd1))
```

## 22.4 Typedef Documentation

**22.4.1 typedef void(\* flexspi\_transfer\_callback\_t)(FLEXSPI\_Type \*base,  
flexspi\_handle\_t \*handle, status\_t status, void \*userData)**

## 22.5 Enumeration Type Documentation

### 22.5.1 anonymous enum

Enumerator

**kStatus\_FLEXSPI\_Busy** FLEXSPI is busy.

**kStatus\_FLEXSPI\_SequenceExecutionTimeout** Sequence execution timeout error occurred during FLEXSPI transfer.

**kStatus\_FLEXSPI\_IpCommandSequenceError** IP command Sequence execution timeout error occurred during FLEXSPI transfer.

**kStatus\_FLEXSPI\_IpCommandGrantTimeout** IP command grant timeout error occurred during FLEXSPI transfer.

### 22.5.2 anonymous enum

Enumerator

**kFLEXSPI\_Command\_STOP** Stop execution, deassert CS.

**kFLEXSPI\_Command\_SDR** Transmit Command code to Flash, using SDR mode.

**kFLEXSPI\_Command\_RADDR\_SDR** Transmit Row Address to Flash, using SDR mode.

**kFLEXSPI\_Command\_CADDR\_SDR** Transmit Column Address to Flash, using SDR mode.

**kFLEXSPI\_Command\_MODE1\_SDR** Transmit 1-bit Mode bits to Flash, using SDR mode.

**kFLEXSPI\_Command\_MODE2\_SDR** Transmit 2-bit Mode bits to Flash, using SDR mode.

**kFLEXSPI\_Command\_MODE4\_SDR** Transmit 4-bit Mode bits to Flash, using SDR mode.

**kFLEXSPI\_Command\_MODE8\_SDR** Transmit 8-bit Mode bits to Flash, using SDR mode.

**kFLEXSPI\_Command\_WRITE\_SDR** Transmit Programming Data to Flash, using SDR mode.

**kFLEXSPI\_Command\_READ\_SDR** Receive Read Data from Flash, using SDR mode.

***kFLEXSPI\_Command\_LEARN\_SDR*** Receive Read Data or Preamble bit from Flash, SDR mode.

***kFLEXSPI\_Command\_DATSZ\_SDR*** Transmit Read/Program Data size (byte) to Flash, SDR mode.

***kFLEXSPI\_Command\_DUMMY\_SDR*** Leave data lines undriven by FlexSPI controller.

***kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

***kFLEXSPI\_Command\_DDR*** Transmit Command code to Flash, using DDR mode.

***kFLEXSPI\_Command\_RADDR\_DDR*** Transmit Row Address to Flash, using DDR mode.

***kFLEXSPI\_Command\_CADDR\_DDR*** Transmit Column Address to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE1\_DDR*** Transmit 1-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE2\_DDR*** Transmit 2-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE4\_DDR*** Transmit 4-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE8\_DDR*** Transmit 8-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_WRITE\_DDR*** Transmit Programming Data to Flash, using DDR mode.

***kFLEXSPI\_Command\_READ\_DDR*** Receive Read Data from Flash, using DDR mode.

***kFLEXSPI\_Command\_LEARN\_DDR*** Receive Read Data or Preamble bit from Flash, DDR mode.

***kFLEXSPI\_Command\_DATSZ\_DDR*** Transmit Read/Program Data size (byte) to Flash, DDR mode.

***kFLEXSPI\_Command\_DUMMY\_DDR*** Leave data lines undriven by FlexSPI controller.

***kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

***kFLEXSPI\_Command\_JUMP\_ON\_CS*** Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence.

### 22.5.3 enum flexspi\_pad\_t

Enumerator

***kFLEXSPI\_1PAD*** Transmit command/address and transmit/receive data only through DATA0/DATA1.

***kFLEXSPI\_2PAD*** Transmit command/address and transmit/receive data only through DATA[1:0].

***kFLEXSPI\_4PAD*** Transmit command/address and transmit/receive data only through DATA[3:0].

***kFLEXSPI\_8PAD*** Transmit command/address and transmit/receive data only through DATA[7:0].

### 22.5.4 enum flexspi\_flags\_t

Enumerator

***kFLEXSPI\_SequenceExecutionTimeoutFlag*** Sequence execution timeout.

***kFLEXSPI\_AhbBusTimeoutFlag*** AHB Bus timeout.

***kFLEXSPI\_SckStoppedBecauseTxEmptyFlag*** SCK is stopped during command sequence because Async TX FIFO empty.

***kFLEXSPI\_SckStoppedBecauseRxFullFlag*** SCK is stopped during command sequence because Async RX FIFO full.

***kFLEXSPI\_DataLearningFailedFlag*** Data learning failed.

***kFLEXSPI\_IpTxFifoWatermarkEmptyFlag*** IP TX FIFO WaterMark empty.

***kFLEXSPI\_IpRxFifoWatermarkAvailableFlag*** IP RX FIFO WaterMark available.

***kFLEXSPI\_AhbCommandSequenceErrorFlag*** AHB triggered Command Sequences Error.

***kFLEXSPI\_IpCommandSequenceErrorFlag*** IP triggered Command Sequences Error.

***kFLEXSPI\_AhbCommandGrantTimeoutFlag*** AHB triggered Command Sequences Grant Timeout.

***kFLEXSPI\_IpCommandGrantTimeoutFlag*** IP triggered Command Sequences Grant Timeout.

***kFLEXSPI\_IpCommandExecutionDoneFlag*** IP triggered Command Sequences Execution finished.

***kFLEXSPI\_AllInterruptFlags*** All flags.

## 22.5.5 enum flexspi\_read\_sample\_clock\_t

Enumerator

***kFLEXSPI\_ReadSampleClkLoopbackInternally*** Dummy Read strobe generated by FlexSPI Controller and loopback internally.

***kFLEXSPI\_ReadSampleClkLoopbackFromDqsPad*** Dummy Read strobe generated by FlexSPI Controller and loopback from DQS pad.

***kFLEXSPI\_ReadSampleClkLoopbackFromSckPad*** SCK output clock and loopback from SCK pad.

***kFLEXSPI\_ReadSampleClkExternalInputFromDqsPad*** Flash provided Read strobe and input from DQS pad.

## 22.5.6 enum flexspi\_cs\_interval\_cycle\_unit\_t

Enumerator

***kFLEXSPI\_CsIntervalUnit1SckCycle*** Chip selection interval: CSINTERVAL \* 1 serial clock cycle.

***kFLEXSPI\_CsIntervalUnit256SckCycle*** Chip selection interval: CSINTERVAL \* 256 serial clock cycle.

## 22.5.7 enum flexspi\_ahb\_write\_wait\_unit\_t

Enumerator

*kFLEXSPI\_AhbWriteWaitUnit2AhbCycle* AWRWAIT unit is 2 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit8AhbCycle* AWRWAIT unit is 8 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit32AhbCycle* AWRWAIT unit is 32 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit128AhbCycle* AWRWAIT unit is 128 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit512AhbCycle* AWRWAIT unit is 512 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit2048AhbCycle* AWRWAIT unit is 2048 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit8192AhbCycle* AWRWAIT unit is 8192 ahb clock cycle.  
*kFLEXSPI\_AhbWriteWaitUnit32768AhbCycle* AWRWAIT unit is 32768 ahb clock cycle.

## 22.5.8 enum flexspi\_ip\_error\_code\_t

Enumerator

*kFLEXSPI\_IpCmdErrorNoError* No error.  
*kFLEXSPI\_IpCmdErrorJumpOnCsInIpCmd* IP command with JMP\_ON\_CS instruction used.  
*kFLEXSPI\_IpCmdErrorUnknownOpCode* Unknown instruction opcode in the sequence.  
*kFLEXSPI\_IpCmdErrorSdrDummyInDdrSequence* Instruction DUMMY\_SDR/DUMMY\_RW-DS\_SDR used in DDR sequence.  
*kFLEXSPI\_IpCmdErrorDdrDummyInSdrSequence* Instruction DUMMY\_DDR/DUMMY\_RW-DS\_DDR used in SDR sequence.  
*kFLEXSPI\_IpCmdErrorInvalidAddress* Flash access start address exceed the whole flash address range (A1/A2/B1/B2).  
*kFLEXSPI\_IpCmdErrorSequenceExecutionTimeout* Sequence execution timeout.  
*kFLEXSPI\_IpCmdErrorFlashBoundaryAcrosss* Flash boundary crossed.

## 22.5.9 enum flexspi\_ahb\_error\_code\_t

Enumerator

*kFLEXSPI\_AhbCmdErrorNoError* No error.  
*kFLEXSPI\_AhbCmdErrorJumpOnCsInWriteCmd* AHB Write command with JMP\_ON\_CS instruction used in the sequence.  
*kFLEXSPI\_AhbCmdErrorUnknownOpCode* Unknown instruction opcode in the sequence.  
*kFLEXSPI\_AhbCmdErrorSdrDummyInDdrSequence* Instruction DUMMY\_SDR/DUMMY\_R-WDS\_SDR used in DDR sequence.  
*kFLEXSPI\_AhbCmdErrorDdrDummyInSdrSequence* Instruction DUMMY\_DDR/DUMMY\_R-WDS\_DDR used in SDR sequence.  
*kFLEXSPI\_AhbCmdSequenceExecutionTimeout* Sequence execution timeout.

### 22.5.10 enum flexspi\_port\_t

Enumerator

- kFLEXSPI\_PortA1*** Access flash on A1 port.
- kFLEXSPI\_PortA2*** Access flash on A2 port.
- kFLEXSPI\_PortB1*** Access flash on B1 port.
- kFLEXSPI\_PortB2*** Access flash on B2 port.

### 22.5.11 enum flexspi\_arb\_command\_source\_t

### 22.5.12 enum flexspi\_command\_type\_t

Enumerator

- kFLEXSPI\_Command*** FlexSPI operation: Only command, both TX and Rx buffer are ignored.
- kFLEXSPI\_Config*** FlexSPI operation: Configure device mode, the TX fifo size is fixed in LUT.

## 22.6 Function Documentation

### 22.6.1 uint32\_t FLEXSPI\_GetInstance ( ***FLEXSPI\_Type*** \* *base* )

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | FLEXSPI base pointer. |
|-------------|-----------------------|

### 22.6.2 status\_t FLEXSPI\_CheckAndClearError ( ***FLEXSPI\_Type*** \* *base*, ***uint32\_t*** *status* )

Parameters

|               |                       |
|---------------|-----------------------|
| <i>base</i>   | FLEXSPI base pointer. |
| <i>status</i> | interrupt status.     |

### 22.6.3 void FLEXSPI\_Init ( ***FLEXSPI\_Type*** \* *base*, ***const flexspi\_config\_t*** \* *config* )

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | FLEXSPI configure structure.     |

#### 22.6.4 void FLEXSPI\_GetDefaultConfig ( **flexspi\_config\_t \* config** )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>config</i> | FLEXSPI configuration structure. |
|---------------|----------------------------------|

#### 22.6.5 void FLEXSPI\_Deinit ( **FLEXSPI\_Type \* base** )

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
|-------------|----------------------------------|

#### 22.6.6 void FLEXSPI\_UpdateDIIValue ( **FLEXSPI\_Type \* base,** **flexspi\_device\_config\_t \* config, flexspi\_port\_t port** )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | Flash configuration parameters.  |
| <i>port</i>   | FLEXSPI Operation port.          |

#### 22.6.7 void FLEXSPI\_SetFlashConfig ( **FLEXSPI\_Type \* base,** **flexspi\_device\_config\_t \* config, flexspi\_port\_t port** )

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>config</i> | Flash configuration parameters.  |
| <i>port</i>   | FLEXSPI Operation port.          |

## 22.6.8 static void FLEXSPI\_SoftwareReset ( FLEXSPI\_Type \* *base* ) [inline], [static]

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
|-------------|----------------------------------|

## 22.6.9 static void FLEXSPI\_Enable ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                |
| <i>enable</i> | True means enable FLEXSPI, false means disable. |

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

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

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

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

**22.6.12 static void FLEXSPI\_EnableTxDMA ( FLEXSPI\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                               |
| <i>enable</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |

**22.6.13 static void FLEXSPI\_EnableRxDMA ( FLEXSPI\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                              |
| <i>enable</i> | Enable flag for receive DMA request. Pass true for enable, false for disable. |

**22.6.14 static uint32\_t FLEXSPI\_GetTxFifoAddress ( FLEXSPI\_Type \* *base* )  
[inline], [static]**

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
|-------------|----------------------------------|

Return values

|            |                  |
|------------|------------------|
| <i>The</i> | tx fifo address. |
|------------|------------------|

**22.6.15 static uint32\_t FLEXSPI\_GetRxFifoAddress ( FLEXSPI\_Type \* *base* )  
[inline], [static]**

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
|-------------|----------------------------------|

Return values

|            |                  |
|------------|------------------|
| <i>The</i> | rx fifo address. |
|------------|------------------|

### 22.6.16 static void FLEXSPI\_ResetFifos ( **FLEXSPI\_Type** \* *base*, bool *txFifo*, bool *rxFifo* ) [inline], [static]

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address. |
| <i>txFifo</i> | Pass true to reset TX FIFO.      |
| <i>rxFifo</i> | Pass true to reset RX FIFO.      |

### 22.6.17 static void FLEXSPI\_GetFifoCounts ( **FLEXSPI\_Type** \* *base*, size\_t \* *txCount*, size\_t \* *rxCount* ) [inline], [static]

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | FLEXSPI peripheral base address.                                                                                             |
| out | <i>txCount</i> | Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required. |
| out | <i>rxCount</i> | Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.  |

### 22.6.18 static uint32\_t FLEXSPI\_GetInterruptStatusFlags ( **FLEXSPI\_Type** \* *base* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
|-------------|----------------------------------|

Return values

|                  |                                                                                                |
|------------------|------------------------------------------------------------------------------------------------|
| <i>interrupt</i> | status flag, use status flag to AND <code>flexspi_flags_t</code> could get the related status. |
|------------------|------------------------------------------------------------------------------------------------|

### 22.6.19 static void FLEXSPI\_ClearInterruptStatusFlags ( `FLEXSPI_Type` \* *base*, `uint32_t` *mask* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

### 22.6.20 static void FLEXSPI\_GetDataLearningPhase ( `FLEXSPI_Type` \* *base*, `uint8_t` \* *portAPhase*, `uint8_t` \* *portBPhase* ) [inline], [static]

Parameters

|                   |                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>       | FLEXSPI peripheral base address.                                                              |
| <i>portAPhase</i> | Pointer to a <code>uint8_t</code> type variable to receive the selected clock phase on PORTA. |
| <i>portBPhase</i> | Pointer to a <code>uint8_t</code> type variable to receive the selected clock phase on PORTB. |

### 22.6.21 static `flexspi_arb_command_source_t` FLEXSPI\_GetArbitrator-CommandSource ( `FLEXSPI_Type` \* *base* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
|-------------|----------------------------------|

Return values

|                |                                     |
|----------------|-------------------------------------|
| <i>trigger</i> | source of current command sequence. |
|----------------|-------------------------------------|

**22.6.22 static flexspi\_ip\_error\_code\_t FLEXSPI\_GetIPCommandErrorCode ( FLEXSPI\_Type \* *base*, uint8\_t \* *index* ) [inline], [static]**

Parameters

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                      |
| <i>index</i> | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

|              |                                      |
|--------------|--------------------------------------|
| <i>error</i> | code when IP command error detected. |
|--------------|--------------------------------------|

**22.6.23 static flexspi\_ahb\_error\_code\_t FLEXSPI\_GetAHBCommandErrorCode ( FLEXSPI\_Type \* *base*, uint8\_t \* *index* ) [inline], [static]**

Parameters

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                      |
| <i>index</i> | Pointer to a uint8_t type variable to receive the sequence index when error detected. |

Return values

|              |                                       |
|--------------|---------------------------------------|
| <i>error</i> | code when AHB command error detected. |
|--------------|---------------------------------------|

**22.6.24 static bool FLEXSPI\_GetBusIdleStatus ( FLEXSPI\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
|-------------|----------------------------------|

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is idle. |
| <i>false</i> | Bus is busy. |

### 22.6.25 void FLEXSPI\_UpdateRxSampleClock ( **FLEXSPI\_Type** \* *base*, **flexspi\_read\_sample\_clock\_t** *clockSource* )

Parameters

|                    |                                                                 |
|--------------------|-----------------------------------------------------------------|
| <i>base</i>        | FLEXSPI peripheral base address.                                |
| <i>clockSource</i> | clockSource of type <a href="#">flexspi_read_sample_clock_t</a> |

### 22.6.26 void FLEXSPI\_UpdateLUT ( **FLEXSPI\_Type** \* *base*, **uint32\_t** *index*, **const uint32\_t** \* *cmd*, **uint32\_t** *count* )

Parameters

|              |                                                                                                                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                                                                                                                                                       |
| <i>index</i> | From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory. |
| <i>cmd</i>   | Command sequence array.                                                                                                                                                                                                |
| <i>count</i> | Number of sequences.                                                                                                                                                                                                   |

### 22.6.27 static void FLEXSPI\_WriteData ( **FLEXSPI\_Type** \* *base*, **uint32\_t** *data*, **uint8\_t** *fifoIndex* ) [**inline**], [**static**]

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | FLEXSPI peripheral base address |
|-------------|---------------------------------|

|                  |                         |
|------------------|-------------------------|
| <i>data</i>      | The data bytes to send  |
| <i>fifoIndex</i> | Destination fifo index. |

**22.6.28 static uint32\_t FLEXSPI\_ReadData ( FLEXSPI\_Type \* *base*, uint8\_t *fifoIndex* ) [inline], [static]**

Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address |
| <i>fifoIndex</i> | Source fifo index.              |

Returns

The data in the FIFO.

**22.6.29 status\_t FLEXSPI\_WriteBlocking ( FLEXSPI\_Type \* *base*, uint8\_t \* *buffer*, size\_t *size* )**

Note

This function blocks via polling until all bytes have been sent.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address  |
| <i>buffer</i> | The data bytes to send           |
| <i>size</i>   | The number of data bytes to send |

Return values

|                                                 |                             |
|-------------------------------------------------|-----------------------------|
| <i>kStatus_Success</i>                          | write success without error |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout  |

|                                                |                                    |
|------------------------------------------------|------------------------------------|
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i> | IP command sequence error detected |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>  | IP command grant timeout detected  |

### 22.6.30 status\_t FLEXSPI\_ReadBlocking ( **FLEXSPI\_Type** \* *base*, **uint8\_t** \* *buffer*, **size\_t** *size* )

Note

This function blocks via polling until all bytes have been sent.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address     |
| <i>buffer</i> | The data bytes to send              |
| <i>size</i>   | The number of data bytes to receive |

Return values

|                                                 |                                     |
|-------------------------------------------------|-------------------------------------|
| <i>kStatus_Success</i>                          | read success without error          |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout          |
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i>  | IP command sequencen error detected |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>   | IP command grant timeout detected   |

### 22.6.31 status\_t FLEXSPI\_TransferBlocking ( **FLEXSPI\_Type** \* *base*, **flexspi\_transfer\_t** \* *xfer* )

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | FLEXSPI peripheral base address    |
| <i>xfer</i> | pointer to the transfer structure. |

Return values

|                                                 |                                        |
|-------------------------------------------------|----------------------------------------|
| <i>kStatus_Success</i>                          | command transfer success without error |
| <i>kStatus_FLEXSPI_SequenceExecutionTimeout</i> | sequence execution timeout             |
| <i>kStatus_FLEXSPI_IpCommandSequenceError</i>   | IP command sequence error detected     |
| <i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>    | IP command grant timeout detected      |

**22.6.32 void FLEXSPI\_TransferCreateHandle ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle*, flexspi\_transfer\_callback\_t *callback*, void \* *userData* )**

Parameters

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>base</i>     | FLEXSPI peripheral base address.                                   |
| <i>handle</i>   | pointer to flexspi_handle_t structure to store the transfer state. |
| <i>callback</i> | pointer to user callback function.                                 |
| <i>userData</i> | user parameter passed to the callback function.                    |

**22.6.33 status\_t FLEXSPI\_TransferNonBlocking ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle*, flexspi\_transfer\_t \* *xfer* )**

Note

Calling the API returns immediately after transfer initiates. The user needs to call FLEXSPI\_GetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not *kStatus\_FLEXSPI\_Busy*, the transfer is finished. For FLEXSPI\_Read, the dataSize should be multiple of rx watermark level, or FLEXSPI could not read data properly.

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                       |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | pointer to <a href="#">flexspi_transfer_t</a> structure.               |

Return values

|                             |                                           |
|-----------------------------|-------------------------------------------|
| <i>kStatus_Success</i>      | Successfully start the data transmission. |
| <i>kStatus_FLEXSPI_Busy</i> | Previous transmission still not finished. |

### 22.6.34 status\_t FLEXSPI\_TransferGetCount ( **FLEXSPI\_Type** \* *base*, **flexspi\_handle\_t** \* *handle*, **size\_t** \* *count* )

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                       |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.    |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

### 22.6.35 void FLEXSPI\_TransferAbort ( **FLEXSPI\_Type** \* *base*, **flexspi\_handle\_t** \* *handle* )

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                      |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state |

**22.6.36 void FLEXSPI\_TransferHandleIRQ ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle* )**

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.       |
| <i>handle</i> | pointer to flexspi_handle_t structure. |

# Chapter 23

## FMEAS: Frequency Measure Driver

### 23.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Frequency Measure function of MCUXpresso SDK devices' SYSCON module.

It measures frequency of any on-chip or off-chip clock signal. The more precise and higher accuracy clock is selected as a reference clock. The resulting frequency is internally computed from the ratio of value of selected target and reference clock counters.

### 23.2 Frequency Measure Driver operation

`INPUTMUX_AttachSignal()` function has to be used to select reference and target clock signal sources.

`FMEAS_StartMeasure()` function starts the measurement cycle.

`FMEAS_IsMeasureComplete()` can be polled to check if the measurement cycle has finished.

`FMEAS_GetFrequency()` returns the frequency of the target clock. Frequency of the reference clock has to be provided as a parameter.

### 23.3 Typical use case

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

### Files

- file `fsl_fmeas.h`

### Driver version

- `#define FSL_FMEAS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`  
*Defines LPC Frequency Measure driver version 2.1.1.*

### FMEAS Functional Operation

- static void `FMEAS_StartMeasure` (FMEAS\_SYSCON\_Type \*base)  
*Starts a frequency measurement cycle.*
- static bool `FMEAS_IsMeasureComplete` (FMEAS\_SYSCON\_Type \*base)  
*Indicates when a frequency measurement cycle is complete.*
- uint32\_t `FMEAS_GetFrequency` (FMEAS\_SYSCON\_Type \*base, uint32\_t refClockRate)  
*Returns the computed value for a frequency measurement cycle.*

## 23.4 Macro Definition Documentation

### 23.4.1 #define FSL\_FMEAS\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

## 23.5 Function Documentation

### 23.5.1 static void FMEAS\_StartMeasure ( FMEAS\_SYSCon\_Type \* *base* ) [inline], [static]

Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>base</i> | : SYSCon peripheral base address. |
|-------------|-----------------------------------|

Set the reference clock count cycle to  $2^{20}$  times

### 23.5.2 static bool FMEAS\_IsMeasureComplete ( FMEAS\_SYSCon\_Type \* *base* ) [inline], [static]

Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>base</i> | : SYSCon peripheral base address. |
|-------------|-----------------------------------|

Returns

true if a measurement cycle is active, otherwise false.

### 23.5.3 uint32\_t FMEAS\_GetFrequency ( FMEAS\_SYSCon\_Type \* *base*, uint32\_t *refClockRate* )

Parameters

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| <i>base</i>         | : SYSCon peripheral base address.                                   |
| <i>refClockRate</i> | : Reference clock rate used during the frequency measurement cycle. |

Returns

Frequency in Hz.

# Chapter 24

## GDMA: General DMA(GDMA) Driver

### 24.1 Overview

The MCUXpresso SDK provides a driver for the GDMA.

GDMA driver provides two types of APIs:

The first type is functional APIs, or the basic channel operation APIs. Such as [GDMA\\_SetChannelSourceAddress](#), [GDMA\\_StartChannel](#). Generally, functional APIs use the `GDMA_Type *base` as parameter. They are simple basic function blocks and does not handle the interrupts. Application should implement the ISR if only functional APIs used.

The second type is transactional APIs. Such as [GDMA\\_SubmitTransfer](#). Generally these APIs uses the parameter `gdma_handle_t`. Transactional APIs handles the interrupts, user could get the interrupt status by registering callback.

The functional APIs and transactional APIs are very similar to each other, please refer the driver examples.

### Data Structures

- struct [gdma\\_channel\\_xfer\\_config\\_t](#)  
*GDMA channel transfer configuration. [More...](#)*
- struct [gdma\\_handle\\_t](#)  
*GDMA transfer handle structure. [More...](#)*

### Macros

- #define [GDMA\\_DESC\\_LLI](#)(linkListAddr, stopAfterDescFinished, enableDescInterrupt)  
*Macro for GDMA link list descriptor LLI.*

### Typedefs

- typedef void(\* [gdma\\_callback\\_t](#) )(struct [\\_gdma\\_handle](#) \*handle, void \*userData, uint32\_t interrupts)  
*Define Callback function for GDMA.*

### Enumerations

- enum [gdma\\_transfer\\_width\\_t](#) {  
  [kGDMA\\_TransferWidth1Byte](#) = 1U,  
  [kGDMA\\_TransferWidth2Byte](#) = 2U,  
  [kGDMA\\_TransferWidth4Byte](#) = 3U }  
*GDMA transfer width.*

- enum `gdma_burst_size_t` {
   
  `kGDMA_BurstSize1` = 0U,
   
  `kGDMA_BurstSize4` = 1U,
   
  `kGDMA_BurstSize8` = 2U,
   
  `kGDMA_BurstSize16` = 3U,
   
  `kGDMA_BurstSizeWrap4` = 5U,
   
  `kGDMA_BurstSizeWrap8` = 6U,
   
  `kGDMA_BurstSizeWrap16` = 7U }

*GDMA burst size.*

- enum `_gdma_ahb_prot` {
   
  `kGDMA_ProtUserMode` = (0U << 0U),
   
  `kGDMA_ProtPrivilegedMode` = (1U << 0U),
   
  `kGDMA_ProtUnbufferable` = (0U << 1U),
   
  `kGDMA_ProtBufferable` = (1U << 1U),
   
  `kGDMA_ProtUncacheable` = (0U << 2U),
   
  `kGDMA_ProtCacheable` = (1U << 2U) }

*GDMA AHB HPROT flags.*

- enum `gdma_priority_t` {
   
  `kGDMA_ChannelPriority0` = 0,
   
  `kGDMA_ChannelPriority1`,
   
  `kGDMA_ChannelPriority2`,
   
  `kGDMA_ChannelPriority3`,
   
  `kGDMA_ChannelPriority4`,
   
  `kGDMA_ChannelPriority5`,
   
  `kGDMA_ChannelPriority6`,
   
  `kGDMA_ChannelPriority7`,
   
  `kGDMA_ChannelPriority8`,
   
  `kGDMA_ChannelPriority9`,
   
  `kGDMA_ChannelPriority10`,
   
  `kGDMA_ChannelPriority11`,
   
  `kGDMA_ChannelPriority12`,
   
  `kGDMA_ChannelPriority13`,
   
  `kGDMA_ChannelPriority14`,
   
  `kGDMA_ChannelPriority15` }

*GDMA channel priority.*

- enum `_gdma_interrupt_enable` {
   
  `kGDMA_DescriptorTransferDoneInterruptEnable` = GDMA\_CHNL\_INT\_MASK\_DESC\_TFRINT\_T\_MASK,
   
  `kGDMA_AddressErrorInterruptEnable` = GDMA\_CHNL\_INT\_MASK\_ADDRERRINT\_MASK,
   
  `kGDMA_BusErrorInterruptEnable` = GDMA\_CHNL\_INT\_MASK\_BUSERRINT\_MASK,
   
  `kGDMA_TransferDoneInterruptEnable` = GDMA\_CHNL\_INT\_MASK\_TFRINT\_MASK,
   
  `kGDMA_BlockTransferDoneInterruptEnable` = GDMA\_CHNL\_INT\_MASK\_BLOCKINT\_MASK,
   
  `kGDMA_AllInterruptEnable` }

*GDMA interrupts to enable.*

- enum `_gdma_interrupt_flags` {

```

kGDMA_DescriptorTransferDoneFlag = GDMA_CHNL_INT_DESC_STATUS_TFRINT_MASK,
kGDMA_ChannelInterruptFlag = GDMA_CHNL_INT_STATUS_CHLINT_MASK,
kGDMA_AddressErrorFlag = GDMA_CHNL_INT_STATUS_ADDRERRINT_MASK,
kGDMA_BusErrorFlag = GDMA_CHNL_INT_STATUS_BUSERRINT_MASK,
kGDMA_TransferDoneFlag = GDMA_CHNL_INT_STATUS_TFRINT_MASK,
kGDMA_BlockTransferDoneFlag = GDMA_CHNL_INT_STATUS_BLOCKINT_MASK,
kGDMA_AllInterruptFlag }

```

*GDMA interrupt status flags.*

## Functions

- struct **\_\_ALIGNED(16) \_gdma\_descriptor**  
*GDMA channel link list descriptor structure.*

## Driver version

- #define **FSL\_GDMA\_DRIVER\_VERSION** (**MAKE\_VERSION(2, 0, 3)**)

## GDMA initialization and De-initialization

- void **GDMA\_Init** (GDMA\_Type \*base)  
*Initializes GDMA peripheral.*
- void **GDMA\_Deinit** (GDMA\_Type \*base)  
*Deinitializes GDMA peripheral.*

## GDMA Channel Operation

- static void **GDMA\_SetChannelSourceAddress** (GDMA\_Type \*base, uint8\_t channel, uint32\_t addr)  
*Set GDMA channel source address.*
- static void **GDMA\_SetChannelDestAddress** (GDMA\_Type \*base, uint8\_t channel, uint32\_t addr)  
*Set GDMA channel destination address.*
- static void **GDMA\_StartChannel** (GDMA\_Type \*base, uint8\_t channel)  
*Start GDMA channel to work.*
- static void **GDMA\_StopChannel** (GDMA\_Type \*base, uint8\_t channel)  
*Stop GDMA channel.*
- static bool **GDMA\_IsChannelBusy** (GDMA\_Type \*base, uint8\_t channel)  
*Return whether GDMA channel is processing transfer.*
- static void **GDMA\_EnableChannelInterrupts** (GDMA\_Type \*base, uint8\_t channel, uint32\_t interrupts)  
*Enables the interrupt for the GDMA transfer.*
- static void **GDMA\_DisableChannelInterrupts** (GDMA\_Type \*base, uint8\_t channel, uint32\_t interrupts)  
*Disables the interrupt for the GDMA transfer.*
- static uint32\_t **GDMA\_GetChannelInterruptFlags** (GDMA\_Type \*base, uint8\_t channel)  
*Get the GDMA channel interrupt flags.*
- static void **GDMA\_ClearChannelInterruptFlags** (GDMA\_Type \*base, uint8\_t channel, uint32\_t flags)  
*Clear the GDMA channel interrupt flags.*

- static uint32\_t **GDMA\_GetChannelFinishedDescriptorNumber** (GDMA\_Type \*base, uint8\_t channel)  
*Get the number of finished descriptor.*
- static void **GDMA\_ClearChannelFinishedDescriptorNumber** (GDMA\_Type \*base, uint8\_t channel)  
*Clear the number of finished descriptor.*
- static void **GDMA\_SetChannelPriority** (GDMA\_Type \*base, uint8\_t channel, **gdma\_priority\_t** priority)  
*Set priority of channel.*
- status\_t **GDMA\_SetChannelTransferConfig** (GDMA\_Type \*base, uint8\_t channel, const **gdma\_channel\_xfer\_config\_t** \*config)  
*Set channel transfer configuration.*

## GDMA Transactional Operation

- void **GDMA\_CreateHandle** (**gdma\_handle\_t** \*handle, GDMA\_Type \*base, uint8\_t channel)  
*Creates the GDMA handle.*
- void **GDMA\_SetCallback** (**gdma\_handle\_t** \*handle, **gdma\_callback\_t** callback, void \*userData)  
*Installs a callback function for the GDMA transfer.*
- status\_t **GDMA\_SubmitTransfer** (**gdma\_handle\_t** \*handle, **gdma\_channel\_xfer\_config\_t** \*config)  
*Submits the GDMA channel transfer request.*
- void **GDMA\_StartTransfer** (**gdma\_handle\_t** \*handle)  
*GDMA start transfer.*
- void **GDMA\_AbortTransfer** (**gdma\_handle\_t** \*handle)  
*Abort running transfer by handle.*
- void **GDMA\_IRQHandler** (GDMA\_Type \*base)  
*GDMA IRQ handler.*

## 24.2 Data Structure Documentation

### 24.2.1 struct gdma\_channel\_xfer\_config\_t

Note

The transfer configuration must follow the requirements:

- SRCBSIZE \* SRCWIDTH == DESTBSIZE \* DESTWIDTH
- If wrap not used, the address should align with WIDTH
- If wrap used, the address should align with WIDTH \* BURST\_SIZE.

## Data Fields

- uint32\_t **srcAddr**  
*Source data address.*
- uint32\_t **destAddr**  
*Destination data address.*
- uint8\_t **ahbProt**  
*GDMA AHB HPROT flags, it could be OR'ed value of `_gdma_ahb_prot`.*
- **gdma\_burst\_size\_t** **srcBurstSize**  
*Source address burst size.*
- **gdma\_burst\_size\_t** **destBurstSize**

- *Destination address burst size.*  
• `gdma_transfer_width_t srcWidth`  
    *Source transfer width.*
- `gdma_transfer_width_t destWidth`  
    *Destination transfer width.*
- `bool srcAddrInc`  
    *Increase source address on each successive access.*
- `bool destAddrInc`  
    *Increase destination address on each successive access.*
- `uint16_t transferLen`  
    *Transfer length in bytes, max value is 8 \* 1024 - 1, should align with transfer size.*
- `bool enableLinkList`  
    *Enable link list or not.*
- `bool enableDescInterrupt`  
    *Generate interrupt when descriptor transfer finished, only used when `enableLinkList` is true.*
- `bool stopAfterDescFinished`  
    *Stop channel when descriptor transfer finished, only used when `enableLinkList` is true.*
- `uint32_t linkListAddr`  
    *Link list address, only used when `enableLinkList` is true.*

**Field Documentation**

- (1) `uint8_t gdma_channel_xfer_config_t::ahbProt`
- (2) `gdma_burst_size_t gdma_channel_xfer_config_t::srcBurstSize`
- (3) `gdma_burst_size_t gdma_channel_xfer_config_t::destBurstSize`
- (4) `gdma_transfer_width_t gdma_channel_xfer_config_t::srcWidth`
- (5) `gdma_transfer_width_t gdma_channel_xfer_config_t::destWidth`
- (6) `bool gdma_channel_xfer_config_t::srcAddrInc`
- (7) `bool gdma_channel_xfer_config_t::destAddrInc`
- (8) `uint16_t gdma_channel_xfer_config_t::transferLen`
- (9) `bool gdma_channel_xfer_config_t::enableLinkList`
- (10) `bool gdma_channel_xfer_config_t::enableDescInterrupt`
- (11) `bool gdma_channel_xfer_config_t::stopAfterDescFinished`
- (12) `uint32_t gdma_channel_xfer_config_t::linkListAddr`

**24.2.2 struct gdma\_handle\_t****Data Fields**

- `GDMA_Type * gdma`  
*GDMA peripheral base address.*
- `uint8_t channel`  
*GDMA channel number.*
- `gdma_callback_t callback`  
*Callback function.*
- `void * userData`  
*Callback function parameter.*

**Field Documentation**

- (1) `gdma_callback_t gdma_handle_t::callback`

Invoked interrupt happens.

## 24.3 Macro Definition Documentation

### 24.3.1 #define GDMA\_DESC\_LLI( *linkListAddr*, *stopAfterDescFinished*, *enableDescInterrupt* )

**Value:**

```
((uint32_t)(linkListAddr)&GDMA_LLI_LLI_MASK) | ((enableDescInterrupt) ? GDMA_LLI_DESC_INT_EN_MASK : 0UL) |
\ ((stopAfterDescFinished) ? GDMA_LLI_STOP_MASK : 0UL))
```

This macro constructs gdma\_descriptor\_t::lli.

Parameters

|                              |                                                         |
|------------------------------|---------------------------------------------------------|
| <i>linkListAddr</i>          | Address of next link list descriptor item.              |
| <i>stopAfterDescFinished</i> | Stop or not after this descriptor transfer done.        |
| <i>enableDescInterrupt</i>   | Generate interrupt after this descriptor transfer done. |

## 24.4 Typedef Documentation

### 24.4.1 `typedef void(* gdma_callback_t)(struct _gdma_handle *handle, void *userData, uint32_t interrupts)`

handle: Pointer to the GDMA driver handle. userData: The userData registered using [GDMA\\_SetCallback](#). interrupts: The interrupts flags of the specific channel.

## 24.5 Enumeration Type Documentation

### 24.5.1 enum gdma\_transfer\_width\_t

Enumerator

- kGDMA\_TransferWidth1Byte* 1 byte.
- kGDMA\_TransferWidth2Byte* 2 bytes.
- kGDMA\_TransferWidth4Byte* 4 bytes.

### 24.5.2 enum gdma\_burst\_size\_t

Enumerator

- kGDMA\_BurstSize1* Burst 1.

*kGDMA\_BurstSize4* Burst 4.  
*kGDMA\_BurstSize8* Burst 8.  
*kGDMA\_BurstSize16* Burst 16.  
*kGDMA\_BurstSizeWrap4* Wrap 4.  
*kGDMA\_BurstSizeWrap8* Wrap 8.  
*kGDMA\_BurstSizeWrap16* Wrap 16.

#### 24.5.3 enum \_gdma\_ahb\_prot

Enumerator

*kGDMA\_ProtUserMode* The access is in user mode.  
*kGDMA\_ProtPrivilegedMode* The access is in privileged mode.  
*kGDMA\_ProtUnbufferable* The access is not bufferable.  
*kGDMA\_ProtBufferable* The access is bufferable.  
*kGDMA\_ProtUncacheable* The access is not cacheable.  
*kGDMA\_ProtCacheable* The access is cacheable.

#### 24.5.4 enum gdma\_priority\_t

Enumerator

*kGDMA\_ChannelPriority0* Lowest channel priority - priority 0.  
*kGDMA\_ChannelPriority1* Channel priority 1.  
*kGDMA\_ChannelPriority2* Channel priority 2.  
*kGDMA\_ChannelPriority3* Channel priority 3.  
*kGDMA\_ChannelPriority4* Channel priority 4.  
*kGDMA\_ChannelPriority5* Channel priority 5.  
*kGDMA\_ChannelPriority6* Channel priority 6.  
*kGDMA\_ChannelPriority7* Channel priority 7.  
*kGDMA\_ChannelPriority8* Channel priority 8.  
*kGDMA\_ChannelPriority9* Channel priority 9.  
*kGDMA\_ChannelPriority10* Channel priority 10.  
*kGDMA\_ChannelPriority11* Channel priority 11.  
*kGDMA\_ChannelPriority12* Channel priority 12.  
*kGDMA\_ChannelPriority13* Channel priority 13.  
*kGDMA\_ChannelPriority14* Channel priority 14.  
*kGDMA\_ChannelPriority15* Highest channel priority - priority 15.

### 24.5.5 enum \_gdma\_interrupt\_enable

Enumerator

***kGDMA\_DescriptorTransferDoneInterruptEnable*** Descriptor transfer done interrupt. This happens when the descriptor is configured to generate interrupt when transfer done.

***kGDMA\_AddressErrorInterruptEnable*** Channel source or destination address is not aligned to corresponding transfer width.

***kGDMA\_BusErrorInterruptEnable*** AHB bus interrupt.

***kGDMA\_TransferDoneInterruptEnable*** DMA transfer done interrupt.

***kGDMA\_BlockTransferDoneInterruptEnable*** DMA block single/burst transfer done interrupt.

***kGDMA\_AllInterruptEnable*** All interrupt enable.

### 24.5.6 enum \_gdma\_interrupt\_flags

Enumerator

***kGDMA\_DescriptorTransferDoneFlag*** Descriptor transfer done interrupt. This happens when the descriptor is configured to generate interrupt when transfer done.

***kGDMA\_ChannelInterruptFlag*** OR of the content of the respective unmasksed interrupt of channel.

***kGDMA\_AddressErrorFlag*** Channel source or destination address is not aligned to corresponding transfer width.

***kGDMA\_BusErrorFlag*** AHB bus interrupt.

***kGDMA\_TransferDoneFlag*** DMA transfer done interrupt.

***kGDMA\_BlockTransferDoneFlag*** DMA block single/burst transfer done interrupt.

***kGDMA\_AllInterruptFlag*** All interrupt flags.

## 24.6 Function Documentation

### 24.6.1 struct \_\_ALIGNED( 16 )

< Source address.

< Destination address.

< Link list item.

< Transfer control.

### 24.6.2 void GDMA\_Init( GDMA\_Type \* base )

It ungates the GDMA access clock, after this function, the GDMA module is ready to be used.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | GDMA peripheral base address. |
|-------------|-------------------------------|

#### 24.6.3 void GDMA\_Deinit( **GDMA\_Type** \* *base* )

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | GDMA peripheral base address. |
|-------------|-------------------------------|

#### 24.6.4 static void GDMA\_SetChannelSourceAddress( **GDMA\_Type** \* *base*, **uint8\_t** *channel*, **uint32\_t** *addr* ) [inline], [static]

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | GDMA peripheral base address. |
| <i>channel</i> | GDMA channel number.          |
| <i>addr</i>    | Source address.               |

#### 24.6.5 static void GDMA\_SetChannelDestAddress( **GDMA\_Type** \* *base*, **uint8\_t** *channel*, **uint32\_t** *addr* ) [inline], [static]

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | GDMA peripheral base address. |
| <i>channel</i> | GDMA channel number.          |
| <i>addr</i>    | Destination address.          |

#### 24.6.6 static void GDMA\_StartChannel( **GDMA\_Type** \* *base*, **uint8\_t** *channel* ) [inline], [static]

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | GDMA peripheral base address. |
| <i>channel</i> | GDMA channel number.          |

#### 24.6.7 static void GDMA\_StopChannel ( **GDMA\_Type** \* *base*, **uint8\_t** *channel* ) [**inline**], [**static**]

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | GDMA peripheral base address. |
| <i>channel</i> | GDMA channel number.          |

#### 24.6.8 static bool GDMA\_IsChannelBusy ( **GDMA\_Type** \* *base*, **uint8\_t** *channel* ) [**inline**], [**static**]

When [GDMA\\_StopChannel](#) is called, if the channel is on service, it does not stop immediately, application could call this API to check whether the channel is stopped.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | GDMA peripheral base address. |
| <i>channel</i> | GDMA channel number.          |

Returns

True if the channel is busy, false if not.

#### 24.6.9 static void GDMA\_EnableChannelInterrupts ( **GDMA\_Type** \* *base*, **uint8\_t** *channel*, **uint32\_t** *interrupts* ) [**inline**], [**static**]

Parameters

---

|                   |                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------|
| <i>base</i>       | GDMA peripheral base address.                                                           |
| <i>channel</i>    | GDMA channel number.                                                                    |
| <i>interrupts</i> | The interrupts to enable, it is OR'ed value of <a href="#">_gdma_interrupt_enable</a> . |

#### 24.6.10 static void GDMA\_DisableChannelInterrupts ( **GDMA\_Type** \* *base*, **uint8\_t** *channel*, **uint32\_t** *interrupts* ) [inline], [static]

Parameters

|                   |                                                                                          |
|-------------------|------------------------------------------------------------------------------------------|
| <i>base</i>       | GDMA peripheral base address.                                                            |
| <i>channel</i>    | GDMA channel number.                                                                     |
| <i>interrupts</i> | The interrupts to disable, it is OR'ed value of <a href="#">_gdma_interrupt_enable</a> . |

#### 24.6.11 static **uint32\_t** GDMA\_GetChannelInterruptFlags ( **GDMA\_Type** \* *base*, **uint8\_t** *channel* ) [inline], [static]

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | GDMA peripheral base address. |
| <i>channel</i> | GDMA channel number.          |

Returns

The interrupt flags, it is OR'ed value of [\\_gdma\\_interrupt\\_flags](#).

#### 24.6.12 static void GDMA\_ClearChannelInterruptFlags ( **GDMA\_Type** \* *base*, **uint8\_t** *channel*, **uint32\_t** *flags* ) [inline], [static]

The [kGDMA\\_ChannelInterruptFlag](#) is OR'ed status of all other unmasked interrupt flags, it could not be clear directly, it should be cleared by clear all other flags.

Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | GDMA peripheral base address.                                                              |
| <i>channel</i> | GDMA channel number.                                                                       |
| <i>flags</i>   | The interrupt flags to clear, it is OR'ed value of <a href="#">_gdma_interrupt_flags</a> . |

#### 24.6.13 static uint32\_t GDMA\_GetChannelFinishedDescriptorNumber ( GDMA\_Type \* *base*, uint8\_t *channel* ) [inline], [static]

The counter increases when an item of descriptor is done in linklist mode.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | GDMA peripheral base address. |
| <i>channel</i> | GDMA channel number.          |

Returns

Number of finished descriptor.

#### 24.6.14 static void GDMA\_ClearChannelFinishedDescriptorNumber ( GDMA\_Type \* *base*, uint8\_t *channel* ) [inline], [static]

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | GDMA peripheral base address. |
| <i>channel</i> | GDMA channel number.          |

#### 24.6.15 static void GDMA\_SetChannelPriority ( GDMA\_Type \* *base*, uint8\_t *channel*, gdma\_priority\_t *priority* ) [inline], [static]

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | GDMA peripheral base address. |
|-------------|-------------------------------|

|                 |                         |
|-----------------|-------------------------|
| <i>channel</i>  | GDMA channel number.    |
| <i>priority</i> | Channel priority value. |

#### 24.6.16 **status\_t GDMA\_SetChannelTransferConfig ( GDMA\_Type \* *base*, uint8\_t *channel*, const gdma\_channel\_xfer\_config\_t \* *config* )**

This function configures the channel transfer, after configured, [GDMA\\_StartChannel](#) could be called to start the transfer.

This function must be called when previous transfer finished. Application can use [GDMA\\_IsChannelBusy](#) to check whether the channel has finished the previous work.

##### Note

The transfer configuration must follow the requirements:

- SRCBSIZE \* SRCWIDTH == DESTBSIZE \* DESTWIDTH
- If wrap not used, the address should align with WIDTH
- If wrap used, the address should align with WIDTH \* BURST\_SIZE.

##### Parameters

|                |                                                             |
|----------------|-------------------------------------------------------------|
| <i>base</i>    | GDMA base address.                                          |
| <i>channel</i> | GDMA channel number. Pointer to the transfer configuration. |

##### Return values

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_Fail</i>            | GDMA is busy with previous transfer. |
| <i>kStatus_Success</i>         | Configuration set successfully.      |
| <i>kStatus_InvalidArgument</i> | Configuration wrong.                 |

#### 24.6.17 **void GDMA\_CreateHandle ( gdma\_handle\_t \* *handle*, GDMA\_Type \* *base*, uint8\_t *channel* )**

This function is called if using transaction API for GDMA. This function initializes the internal state of GDMA handle.

Parameters

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| <i>handle</i>  | GDMA handle pointer. It stores callback function and parameters. |
| <i>base</i>    | GDMA peripheral base address.                                    |
| <i>channel</i> | GDMA channel number.                                             |

#### **24.6.18 void GDMA\_SetCallback ( *gdma\_handle\_t \* handle*, *gdma\_callback\_t callback*, *void \* userData* )**

This callback is called in GDMA IRQ handler to inform user the interrupt status.

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>handle</i>   | GDMA handle pointer.             |
| <i>callback</i> | GDMA callback function pointer.  |
| <i>userData</i> | Parameter for callback function. |

#### **24.6.19 status\_t GDMA\_SubmitTransfer ( *gdma\_handle\_t \* handle*, *gdma\_channel\_xfer\_config\_t \* config* )**

After this function, user could call [GDMA\\_StartTransfer](#) to start GDMA transfer.

This function must be called when previous transfer finished. Application can use [GDMA\\_IsChannelBusy](#) to check whether the channel has finished the previous work.

Note

The transfer configuration must follow the requirements:

- SRCBSIZE \* SRCWIDTH == DESTBSIZE \* DESTWIDTH
- If wrap not used, the address should align with WIDTH
- If wrap used, the address should align with WIDTH \* BURST\_SIZE.

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | GDMA handle pointer. |
|---------------|----------------------|

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>config</i> | Pointer to GDMA transfer configuration structure. |
|---------------|---------------------------------------------------|

Return values

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_Fail</i>            | GDMA is busy with previous transfer. |
| <i>kStatus_Success</i>         | Configuration set successfully.      |
| <i>kStatus_InvalidArgument</i> | Configuration wrong.                 |

#### 24.6.20 void GDMA\_StartTransfer ( *gdma\_handle\_t \* handle* )

User can call this function after [GDMA\\_SubmitTransfer](#).

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | GDMA handle pointer. |
|---------------|----------------------|

#### 24.6.21 void GDMA\_AbortTransfer ( *gdma\_handle\_t \* handle* )

When this function is called, if the channel is on service, it only stops when service finished.

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | GDMA handle pointer. |
|---------------|----------------------|

#### 24.6.22 void GDMA\_IRQHandler ( *GDMA\_Type \* base* )

This function checks all GDMA channel interrupts and inform application the interrupt flags through user registered callback.

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | GDMA peripheral. |
|-------------|------------------|

# Chapter 25

## GPIO: General Purpose I/O

### 25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Purpose I/O (GPIO) module of MCUXpresso SDK devices.

### 25.2 Function groups

#### 25.2.1 Initialization and deinitialization

The function [GPIO\\_PinInit\(\)](#) initializes the GPIO with specified configuration.

#### 25.2.2 Pin manipulation

The function [GPIO\\_PinWrite\(\)](#) set output state of selected GPIO pin. The function [GPIO\\_PinRead\(\)](#) read input value of selected GPIO pin.

#### 25.2.3 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.

#### 25.2.4 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.

### 25.3 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...](#)*
- struct `gpio_interrupt_config_t`  
*Configures the interrupt generation condition. [More...](#)*

## Enumerations

- enum `gpio_pin_direction_t` {
   
`kGPIO_DigitalInput` = 0U,  
`kGPIO_DigitalOutput` = 1U }
   
*LPC GPIO direction definition.*
- enum `gpio_pin_enable_mode_t` {
   
`kGPIO_PinIntEnableLevel` = `GPIO_PIN_INT_LEVEL`,  
`kGPIO_PinIntEnableEdge` = `GPIO_PIN_INT_EDGE` }
   
*GPIO Pin Interrupt enable mode.*
- enum `gpio_pin_enable_polarity_t` {
   
`kGPIO_PinIntEnableHighOrRise`,  
`kGPIO_PinIntEnableLowOrFall` }
   
*GPIO Pin Interrupt enable polarity.*
- enum `gpio_interrupt_index_t` {
   
`kGPIO_InterruptA` = 0U,  
`kGPIO_InterruptB` = 1U }
   
*LPC GPIO interrupt index 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.*

## 25.4 Data Structure Documentation

### 25.4.1 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.*

### 25.4.2 struct gpio\_interrupt\_config\_t

## 25.5 Macro Definition Documentation

### 25.5.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 7))

## 25.6 Enumeration Type Documentation

### 25.6.1 enum gpio\_pin\_direction\_t

Enumerator

*kGPIO\_DigitalInput* Set current pin as digital input.

*kGPIO\_DigitalOutput* Set current pin as digital output.

### 25.6.2 enum gpio\_pin\_enable\_mode\_t

Enumerator

*kGPIO\_PinIntEnableLevel* Generate Pin Interrupt on level mode.

*kGPIO\_PinIntEnableEdge* Generate Pin Interrupt on edge mode.

### 25.6.3 enum gpio\_pin\_enable\_polarity\_t

Enumerator

*kGPIO\_PinIntEnableHighOrRise* Generate Pin Interrupt on high level or rising edge.

*kGPIO\_PinIntEnableLowOrFall* Generate Pin Interrupt on low level or falling edge.

### 25.6.4 enum gpio\_interrupt\_index\_t

Enumerator

*kGPIO\_InterruptA* Set current pin as interrupt A.

*kGPIO\_InterruptB* Set current pin as interrupt B.

## 25.7 Function Documentation

### 25.7.1 void GPIO\_PortInit ( *GPIO\_Type* \* *base*, *uint32\_t* *port* )

This function ungates the GPIO clock.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | GPIO peripheral base pointer. |
| <i>port</i> | GPIO port number.             |

### 25.7.2 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               |

### 25.7.3 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> |

### 25.7.4 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> |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|

25.7.5 **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                        |

**25.7.6 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                        |

**25.7.7 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 26

## I2C: Inter-Integrated Circuit Driver

### 26.1 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.

### 26.2 Typical use case

#### 26.2.1 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);

```

## 26.2.2 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;

```

### 26.2.3 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;
```

### 26.2.4 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;

```

## 26.2.5 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 Driver
- I2C Master Driver
- I2C Slave Driver

## 26.3 I2C Driver

### 26.3.1 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, 3))

*I2C driver version.*

### 26.3.2 Macro Definition Documentation

**26.3.2.1 #define FSL\_I2C\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 3))**

**26.3.2.2 #define I2C\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

**26.3.2.3 #define I2C\_MASTER\_TRANSMIT\_IGNORE\_LAST\_NACK 1U /\* Define to one means master ignores the last byte's nack and considers the transfer successful. \*/**

### 26.3.3 Enumeration Type Documentation

#### 26.3.3.1 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.

#### 26.3.3.2 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

### 26.3.3.3 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.

## 26.4 I2C Master Driver

### 26.4.1 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.*

### 26.4.2 Data Structure Documentation

#### 26.4.2.1 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`

### 26.4.2.2 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**

### 26.4.2.3 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`

## 26.4.3 Typedef Documentation

### 26.4.3.1 `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.                     |

**26.4.4 Enumeration Type Documentation****26.4.4.1 enum i2c\_direction\_t**

Enumerator

*kI2C\_Write* Master transmit.*kI2C\_Read* Master receive.**26.4.4.2 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.**26.4.4.3 enum \_i2c\_transfer\_states****26.4.5 Function Documentation****26.4.5.1 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

|            |                     |                                                                                                          |
|------------|---------------------|----------------------------------------------------------------------------------------------------------|
| <i>out</i> | <i>masterConfig</i> | User provided configuration structure for default values. Refer to <a href="#">i2c_master_config_t</a> . |
|------------|---------------------|----------------------------------------------------------------------------------------------------------|

#### 26.4.5.2 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.            |

#### 26.4.5.3 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. |
|-------------|----------------------------------|

#### 26.4.5.4 **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.

#### 26.4.5.5 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. |
|-------------|----------------------------------|

#### 26.4.5.6 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. |

#### 26.4.5.7 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](#).

#### 26.4.5.8 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](#).

#### 26.4.5.9 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](#).

#### 26.4.5.10 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. |

#### **26.4.5.11 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. |

#### **26.4.5.12 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.

#### **26.4.5.13 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. |

#### 26.4.5.14 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. |

#### 26.4.5.15 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. |

#### 26.4.5.16 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.                |

#### 26.4.5.17 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.  |

#### 26.4.5.18 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. |

#### 26.4.5.19 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.                            |

**26.4.5.20 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.                            |

**26.4.5.21 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. |

#### 26.4.5.22 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.      |

#### 26.4.5.23 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. |

#### 26.4.5.24 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> |  |

#### 26.4.5.25 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.       |

#### 26.4.5.26 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. |

## 26.5 I2C Slave Driver

### 26.5.1 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.

## Slave IRQ handler

- void [I2C\\_SlaveTransferHandleIRQ](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)  
*Reusable routine to handle slave interrupts.*

### 26.5.2 Data Structure Documentation

#### 26.5.2.1 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](#)

#### 26.5.2.2 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`

### 26.5.2.3 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](#).

#### 26.5.2.4 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`

## 26.5.3 Typedef Documentation

### 26.5.3.1 `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.                           |

**26.5.3.2 `typedef void(* flexcomm_i2c_master_irq_handler_t)(I2C_Type *base, i2c_master_handle_t *handle)`**

**26.5.3.3 `typedef void(* flexcomm_i2c_slave_irq_handler_t)(I2C_Type *base, i2c_slave_handle_t *handle)`**

#### 26.5.4 Enumeration Type Documentation

##### 26.5.4.1 `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.

##### 26.5.4.2 `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.

##### 26.5.4.3 `enum i2c_slave_bus_speed_t`

##### 26.5.4.4 `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.

## 26.5.5 Function Documentation

### 26.5.5.1 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

|     |                    |                                                                                                                    |
|-----|--------------------|--------------------------------------------------------------------------------------------------------------------|
| out | <i>slaveConfig</i> | User provided configuration structure that is set to default values. Refer to <a href="#">i2c_slave_config_t</a> . |
|-----|--------------------|--------------------------------------------------------------------------------------------------------------------|

### 26.5.5.2 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. |

#### **26.5.5.3 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.                                                  |

#### **26.5.5.4 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. |
|-------------|----------------------------------|

#### **26.5.5.5 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. |

#### 26.5.5.6 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.

#### 26.5.5.7 *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).

#### 26.5.5.8 *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).

#### 26.5.5.9 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.      |

#### 26.5.5.10 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 [kI2C\\_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 i2c_slave_handle_t structure which stores the transfer state.                                                                                                                                                                                                         |
| <i>eventMask</i> | Bit mask formed by OR'ing together <i>i2c_slave_transfer_event_t</i> 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 <i>kI2C_SlaveAllEvents</i> to enable all events. |

## Return values

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

**26.5.5.11 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 <i>i2c_slave_transfer_t</i> 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 <i>i2c_slave_transfer_event_t</i> 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 <i>kI2C_SlaveAllEvents</i> to enable all events. |

Return values

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

#### 26.5.5.12 **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

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

#### 26.5.5.13 **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.

#### 26.5.5.14 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> |  |

#### 26.5.5.15 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> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

#### 26.5.5.16 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 <i>i2c_slave_handle_t</i> structure which stores the transfer state. |

# Chapter 27

## I2S: I2S Driver

### 27.1 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.

### 27.2 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.

### 27.3 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.

## 27.4 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.

## 27.5 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.

## 27.6 I2S Data formats

### 27.6.1 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):

## 27.6.2 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.

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.

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):

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 | 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 |

## 27.7 I2S Driver Examples

### 27.7.1 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);
    }
}

```

## 27.7.2 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

## 27.8 I2S Driver

### 27.8.1 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.*
- `void I2S_EnableSecondaryChannel (I2S_Type *base, uint32_t channel, bool oneChannel, uint32_t position)`  
*Enables I2S secondary channel.*
- `static void I2S_DisableSecondaryChannel (I2S_Type *base, uint32_t channel)`  
*Disables I2S secondary channel.*
- `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.*

## 27.8.2 Data Structure Documentation

### 27.8.2.1 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 pdmData**  
*Data source is the D-Mic subsystem.*
- **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)*

### 27.8.2.2 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`

### 27.8.2.3 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 FIFO48H, false use FIFO48H.*
- `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.*

### 27.8.3 Macro Definition Documentation

27.8.3.1 `#define FSL_I2S_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))`

27.8.3.2 `#define I2S_NUM_BUFFERS (4U)`

### 27.8.4 Typedef Documentation

27.8.4.1 `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. |

## 27.8.5 Enumeration Type Documentation

### 27.8.5.1 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.

### 27.8.5.2 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.

### 27.8.5.3 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.

#### 27.8.5.4 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.

#### 27.8.5.5 anonymous enum

Enumerator

*kI2S\_SecondaryChannel1* secondary channel 1

*kI2S\_SecondaryChannel2* secondary channel 2

*kI2S\_SecondaryChannel3* secondary channel 3

### 27.8.6 Function Documentation

#### 27.8.6.1 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. |

#### 27.8.6.2 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. |

**27.8.6.3 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. |
|---------------|-----------------------------------------|

**27.8.6.4 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. |
|---------------|-----------------------------------------|

**27.8.6.5 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. |
|-------------|-------------------|

**27.8.6.6 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.      |

**27.8.6.7 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.                        |

#### 27.8.6.8 **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. |

#### 27.8.6.9 **void I2S\_TxTransferAbort ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle* )**

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | I2S base pointer.            |
| <i>handle</i> | pointer to handle structure. |

#### 27.8.6.10 **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.                        |

#### 27.8.6.11 `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. |

#### 27.8.6.12 `void I2S_RxTransferAbort ( I2S_Type * base, i2s_handle_t * handle )`

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | I2S base pointer.            |
| <i>handle</i> | pointer to handle structure. |

#### 27.8.6.13 `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. |

#### 27.8.6.14 **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. |

#### 27.8.6.15 **static void I2S\_Enable ( I2S\_Type \* *base* ) [inline], [static]**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | I2S base pointer. |
|-------------|-------------------|

#### 27.8.6.16 **void I2S\_EnableSecondaryChannel ( I2S\_Type \* *base*, uint32\_t *channel*, bool *oneChannel*, uint32\_t *position* )**

Parameters

|                   |                                                                                   |
|-------------------|-----------------------------------------------------------------------------------|
| <i>base</i>       | I2S base pointer.                                                                 |
| <i>channel</i>    | secondary channel channel number, reference <code>_i2s_secondary_channel</code> . |
| <i>oneChannel</i> | true is treated as single channel, functionality left channel for this pair.      |
| <i>position</i>   | define the location within the frame of the data, should not bigger than 0x1FFU.  |

27.8.6.17 **static void I2S\_DisableSecondaryChannel ( I2S\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

Parameters

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>base</i>    | I2S base pointer.                                                   |
| <i>channel</i> | secondary channel channel number, reference _i2s_secondary_channel. |

#### 27.8.6.18 static void I2S\_Disable( I2S\_Type \* *base* ) [inline], [static]

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | I2S base pointer. |
|-------------|-------------------|

#### 27.8.6.19 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. |

#### 27.8.6.20 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. |

#### 27.8.6.21 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.

#### **27.8.6.22 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.

#### **27.8.6.23 void I2S\_TxHandleIRQ ( I2S\_Type \* *base*, i2s\_handle\_t \* *handle* )**

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | I2S base pointer.            |
| <i>handle</i> | pointer to handle structure. |

#### **27.8.6.24 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 28

## IMU: Inter CPU Messaging Unit

### 28.1 Overview

The MCUXpresso SDK provides a driver for the IMU module of MCUXpresso SDK devices.

### Data Structures

- struct **IMU\_Type**  
*IMU register structure.* [More...](#)

### Macros

- #define **IMU\_RX\_FIFO\_EMPTY**(link)  
*Get Rx FIFO empty status.*
- #define **IMU\_ERR\_TX\_FIFO\_LOCKED** (-1L)  
*IMU driver returned error value.*
- #define **IMU\_MSG\_FIFO\_MAX\_COUNT** 16U  
*Maximum message numbers in FIFO.*
- #define **IMU\_MAX\_MSG\_FIFO\_WATER\_MARK** (**IMU\_MSG\_FIFO\_MAX\_COUNT** - 1U)  
*Maximum message FIFO warter mark.*

### Enumerations

- enum **\_imu\_status\_flags**  
*IMU status flags.*
- enum **\_imu\_interrupts**  
*IMU interrupt.*

### Driver version

- #define **FSL\_IMU\_DRIVER\_VERSION** (**MAKE\_VERSION(2, 1, 1)**)  
*IMU driver version.*

### IMU initialization.

- **status\_t IMU\_Init** (imu\_link\_t link)  
*Initializes the IMU module.*
- **void IMU\_Deinit** (imu\_link\_t link)  
*De-initializes the IMU module.*

### IMU Message

- static void **IMU\_WriteMsg** (imu\_link\_t link, uint32\_t msg)  
*Write one message to TX FIFO.*

- static uint32\_t **IMU\_ReadMsg** (imu\_link\_t link)  
*Read one message from RX FIFO.*
- int32\_t **IMU\_SendMsgsBlocking** (imu\_link\_t link, const uint32\_t \*msgs, int32\_t msgCount, bool lockSendFifo)  
*Blocking to send messages.*
- int32\_t **IMU\_TrySendMsgs** (imu\_link\_t link, const uint32\_t \*msgs, int32\_t msgCount, bool lockSendFifo)  
*Try to send messages.*
- int32\_t **IMU\_TryReceiveMsgs** (imu\_link\_t link, uint32\_t \*msgs, int32\_t msgCount, bool lockSendFifo)  
*Try to receive messages.*
- int32\_t **IMU\_ReceiveMsgsBlocking** (imu\_link\_t link, uint32\_t \*msgs, int32\_t msgCount, bool lockSendFifo)  
*Blocking to receive messages.*
- int32\_t **IMU\_SendMsgPtrBlocking** (imu\_link\_t link, uint32\_t msgPtr, bool lockSendFifo)  
*Blocking to send messages pointer.*
- static void **IMU\_LockSendFifo** (imu\_link\_t link, bool lock)  
*Lock or unlock the TX FIFO.*
- void **IMU\_FlushSendFifo** (imu\_link\_t link)  
*Flush the send FIFO.*
- static void **IMU\_SetSendFifoWaterMark** (imu\_link\_t link, uint8\_t waterMark)  
*Set send FIFO warter mark.*
- static uint32\_t **IMU\_GetReceivedMsgCount** (imu\_link\_t link)  
*Get the message count in receive FIFO.*
- static uint32\_t **IMU\_GetSendFifoEmptySpace** (imu\_link\_t link)  
*Get the empty slot in send FIFO.*

## Status and Interrupt.

- uint32\_t **IMU\_GetStatusFlags** (imu\_link\_t link)  
*Gets the IMU status flags.*
- static void **IMU\_ClearPendingInterrupts** (imu\_link\_t link, uint32\_t mask)  
*Clear the IMU IRQ.*

## 28.2 Data Structure Documentation

### 28.2.1 struct IMU\_Type

## 28.3 Macro Definition Documentation

### 28.3.1 #define FSL\_IMU\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

### 28.3.2 #define IMU\_RX\_FIFO\_EMPTY( link )

#### Value:

```
(0UL != \
(((IMU_Type *) (uintptr_t) IMU_PEER_CPU_BASE(link)) ->MSG_FIFO_STATUS &
IMU_MSG_FIFO_STATUS_MSG_FIFO_EMPTY_MASK))
```

28.3.3 `#define IMU_ERR_TX_FIFO_LOCKED (-1L)`

28.3.4 `#define IMU_MSG_FIFO_MAX_COUNT 16U`

28.3.5 `#define IMU_MAX_MSG_FIFO_WATER_MARK (IMU_MSG_FIFO_MAX_COUNT - 1U)`

## 28.4 Enumeration Type Documentation

28.4.1 `enum _imu_status_flags`

28.4.2 `enum _imu_interrupts`

## 28.5 Function Documentation

28.5.1 `status_t IMU_Init ( imu_link_t link )`

This function sets IMU to initialized state, including:

- Flush the send FIFO.
- Unlock the send FIFO.
- Set the water mark to (IMU\_MAX\_MSG\_FIFO\_WATER\_MARK)
- Flush the read FIFO.

Parameters

|                   |           |
|-------------------|-----------|
| <code>link</code> | IMU link. |
|-------------------|-----------|

Return values

|                                      |                           |
|--------------------------------------|---------------------------|
| <code>kStatus_InvalidArgument</code> | The link is invalid.      |
| <code>kStatus_Success</code>         | Initialized successfully. |

28.5.2 `void IMU_Deinit ( imu_link_t link )`

Parameters

|                   |           |
|-------------------|-----------|
| <code>link</code> | IMU link. |
|-------------------|-----------|

### 28.5.3 static void IMU\_WriteMsg ( *imu\_link\_t link*, *uint32\_t msg* ) [inline], [static]

This function writes message to the TX FIFO, user need to make sure there is empty space in the TX FIFO, and TX FIFO not locked before calling this function.

Parameters

|             |                      |
|-------------|----------------------|
| <i>link</i> | IMU link.            |
| <i>msg</i>  | The message to send. |

#### 28.5.4 static uint32\_t IMU\_ReadMsg ( imu\_link\_t *link* ) [inline], [static]

User need to make sure there is available message in the RX FIFO.

Parameters

|             |           |
|-------------|-----------|
| <i>link</i> | IMU link. |
|-------------|-----------|

Returns

The message.

#### 28.5.5 int32\_t IMU\_SendMsgsBlocking ( imu\_link\_t *link*, const uint32\_t \* *msgs*, int32\_t *msgCount*, bool *lockSendFifo* )

This function blocks until all messages have been filled to TX FIFO.

- If the TX FIFO is locked, this function returns IMU\_ERR\_TX\_FIFO\_LOCKED.
- If TX FIFO not locked, this function waits the available empty slot in TX FIFO, and fills the message to TX FIFO.
- To lock TX FIFO after filling all messages, set *lockSendFifo* to true.

Parameters

|                     |                                                                             |
|---------------------|-----------------------------------------------------------------------------|
| <i>link</i>         | IMU link.                                                                   |
| <i>msgs</i>         | The messages to send.                                                       |
| <i>msgCount</i>     | Message count, one message is a 32-bit word.                                |
| <i>lockSendFifo</i> | If set to true, the TX FIFO is locked after all messages filled to TX FIFO. |

Returns

If TX FIFO is locked, this function returns IMU\_ERR\_TX\_FIFO\_LOCKED, otherwise, this function returns the actual message count sent out, it equals *msgCount* because this function is blocking function, it returns until all messages have been filled into TX FIFO.

### 28.5.6 int32\_t IMU\_TrySendMsgs ( *imu\_link\_t link*, *const uint32\_t \* msgs*, *int32\_t msgCount*, *bool lockSendFifo* )

This function is similar with [IMU\\_SendMsgsBlocking](#), the difference is, this function tries to send as many as possible, if there is not enough empty slot in TX FIFO, this function fills messages to available empty slots and returns how many messages have been filled.

- If the TX FIFO is locked, this function returns IMU\_ERR\_TX\_FIFO\_LOCKED.
- If TX FIFO not locked, this function fills messages to TX FIFO empty slot, and returns how many messages have been filled.
- If *lockSendFifo* is set to true, TX FIFO is locked after all messages have been filled to TX FIFO. In other word, TX FIFO is locked if the function return value equals *msgCount*, when *lockSendFifo* set to true.

Parameters

|                     |                                                                             |
|---------------------|-----------------------------------------------------------------------------|
| <i>link</i>         | IMU link.                                                                   |
| <i>msgs</i>         | The messages to send.                                                       |
| <i>msgCount</i>     | Message count, one message is a 32-bit word.                                |
| <i>lockSendFifo</i> | If set to true, the TX FIFO is locked after all messages filled to TX FIFO. |

Returns

If TX FIFO is locked, this function returns IMU\_ERR\_TX\_FIFO\_LOCKED, otherwise, this function returns the actual message count sent out.

### 28.5.7 int32\_t IMU\_TryReceiveMsgs ( *imu\_link\_t link*, *uint32\_t \* msgs*, *int32\_t desiredMsgCount*, *bool \* needAckLock* )

This function tries to read messages from RX FIFO. It reads the messages already exists in RX FIFO and returns the actual read count.

- If the RX FIFO has enough messages, this function reads the messages and returns.
- If the RX FIFO does not have enough messages, this function reads the messages in RX FIFO and returns the actual read count.
- During message reading, if RX FIFO is empty and locked, in this case peer CPU will not send message until current CPU send lock ack message. Then this function returns the message count actually received, and sets *needAckLock* to true to inform upper layer.

## Parameters

|                        |                                                                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>link</i>            | IMU link.                                                                                                                                    |
| <i>msgs</i>            | The buffer to read messages.                                                                                                                 |
| <i>desiredMsgCount</i> | Desired read count, one message is a 32-bit word.                                                                                            |
| <i>needAckLock</i>     | Upper layer should always check this value. When this is set to true by this function, upper layer should send lock ack message to peer CPU. |

## Returns

Count of messages actually received.

### 28.5.8 **int32\_t IMU\_ReceiveMsgsBlocking ( imu\_link\_t *link*, uint32\_t \* *msgs*, int32\_t *desiredMsgCount*, bool \* *needAckLock* )**

This function blocks until all desired messages have been received or the RX FIFO is locked.

- If the RX FIFO has enough messages, this function reads the messages and returns.
- If the RX FIFO does not have enough messages, this function waits for the new messages.
- During message reading, if RX FIFO is empty and locked, in this case peer CPU will not send message until current CPU send lock ack message. Then this function returns the message count actually received, and sets *needAckLock* to true to inform upper layer.

## Parameters

|                        |                                                                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>link</i>            | IMU link.                                                                                                                                    |
| <i>msgs</i>            | The buffer to read messages.                                                                                                                 |
| <i>desiredMsgCount</i> | Desired read count, one message is a 32-bit word.                                                                                            |
| <i>needAckLock</i>     | Upper layer should always check this value. When this is set to true by this function, upper layer should send lock ack message to peer CPU. |

## Returns

Count of messages actually received.

### 28.5.9 **int32\_t IMU\_SendMsgPtrBlocking ( imu\_link\_t *link*, uint32\_t *msgPtr*, bool *lockSendFifo* )**

Compared with [IMU\\_SendMsgsBlocking](#), this function fills message pointer to TX FIFO, but not the message content.

This function blocks until the message pointer is filled to TX FIFO.

- If the TX FIFO is locked, this function returns IMU\_ERR\_TX\_FIFO\_LOCKED.
- If TX FIFO not locked, this function waits the available empty slot in TX FIFO, and fills the message pointer to TX FIFO.
- To lock TX FIFO after filling the message pointer, set `lockSendFifo` to true.

Parameters

|                    |                                                                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>link</i>        | IMU link.                                                                                                                                    |
| <i>msgPtr</i>      | The buffer pointer to message to send.                                                                                                       |
| <i>needAckLock</i> | Upper layer should always check this value. When this is set to true by this function, upper layer should send lock ack message to peer CPU. |

Return values

|                               |                                       |
|-------------------------------|---------------------------------------|
| <i>0</i>                      | The message pointer set successfully. |
| <i>IMU_ERR_TX_FIFO_LOCKED</i> | The TX FIFO is locked, send failed.   |

### 28.5.10 static void IMU\_LockSendFifo ( `imu_link_t link`, `bool lock` ) [inline], [static]

Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>link</i> | IMU link.                                       |
| <i>lock</i> | Use true to lock the FIFO, use false to unlock. |

### 28.5.11 void IMU\_FlushSendFifo ( `imu_link_t link` )

Flush all messages in send FIFO.

Parameters

|             |           |
|-------------|-----------|
| <i>link</i> | IMU link. |
|-------------|-----------|

**28.5.12 static void IMU\_SetSendFifoWaterMark ( *imu\_link\_t link*, *uint8\_t waterMark* ) [inline], [static]**

The water mark must be less than IMU\_MAX\_MSG\_FIFO\_WATER\_MARK, i.e.  $0 < \text{waterMark} \leq \text{IMU\_MAX\_MSG\_FIFO\_WATER\_MARK}$ .

Parameters

|                  |                        |
|------------------|------------------------|
| <i>link</i>      | IMU link.              |
| <i>waterMark</i> | Send FIFO warter mark. |

### 28.5.13 static uint32\_t IMU\_GetReceivedMsgCount ( imu\_link\_t *link* ) [inline], [static]

Parameters

|             |           |
|-------------|-----------|
| <i>link</i> | IMU link. |
|-------------|-----------|

Returns

The message count in receive FIFO.

### 28.5.14 static uint32\_t IMU\_GetSendFifoEmptySpace ( imu\_link\_t *link* ) [inline], [static]

Parameters

|             |           |
|-------------|-----------|
| <i>link</i> | IMU link. |
|-------------|-----------|

Returns

The empty slot count in send FIFO.

### 28.5.15 uint32\_t IMU\_GetStatusFlags ( imu\_link\_t *link* )

Parameters

|             |           |
|-------------|-----------|
| <i>link</i> | IMU link. |
|-------------|-----------|

Returns

Bit mask of the IMU status flags, see [\\_imu\\_status\\_flags](#).

### 28.5.16 static void IMU\_ClearPendingInterrupts ( imu\_link\_t *link*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                            |
|-------------|----------------------------------------------------------------------------|
| <i>link</i> | IMU link.                                                                  |
| <i>mask</i> | Bit mask of the interrupts to clear, see <a href="#">_imu_interrupts</a> . |

# Chapter 29

## INPUTMUX: Input Multiplexing Driver

### 29.1 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.

### 29.2 Input Multiplexing Driver operation

INPUTMUX\_AttachSignal function configures the specified input

### 29.3 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](#)

## Macros

- `#define SCT0_PMUX_ID 0x00U`  
*PeriphInmux IDs.*

## Enumerations

- enum [inputmux\\_connection\\_t](#) {  
    [kINPUTMUX\\_Gpio3Inp0ToSct0](#) = 0U + (SCT0\_PMUX\_ID << PMUX\_SHIFT),  
    [kINPUTMUX\\_DebugHaltedToSct0](#) = 23U + (SCT0\_PMUX\_ID << PMUX\_SHIFT),  
    [kINPUTMUX\\_GpioPort1Pin31ToPintsel](#) = 63U + (PINTSEL\_PMUX\_ID << PMUX\_SHIFT),  
    [kINPUTMUX\\_AvpllCh2ClkoutToFreqmeas](#) = 27U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIFT),  
    [kINPUTMUX\\_FlexcommDmaCmpltDone1ToTimer0CaptureChannels](#) = 25U + (CT32BIT0\_CAP\_PMUX\_ID << PMUX\_SHIFT),  
    [kINPUTMUX\\_BtuHostTrigger2ToTimer1CaptureChannels](#) = 21U + (CT32BIT1\_CAP\_PMUX\_ID << PMUX\_SHIFT),  
    [kINPUTMUX\\_FlexcommDmaCmpltDone1ToTimer2CaptureChannels](#) = 25U + (CT32BIT2\_CAP\_PMUX\_ID << PMUX\_SHIFT),  
    [kINPUTMUX\\_BtuHostTrigger2ToTimer3CaptureChannels](#) = 21U + (CT32BIT3\_CAP\_PMUX\_ID << PMUX\_SHIFT),  
};

```

<< PMUX_SHIFT) ,
kINPUTMUX_LcdTxRegToDmaSingleToDma0 = 33U + (DMA0_ITRIG_PMUX_ID << PMU-
X_SHIFT) ,
kINPUTMUX_LcdTxRegToDmaSingleToDma1 = 33U + (DMA1_ITRIG_PMUX_ID << PMU-
X_SHIFT) ,
kINPUTMUX_Dma0OtrigChannel32ToTriginChannels = 32U + (DMA0_OTRIG_PMUX_ID <<
PMUX_SHIFT) }

INPUTMUX connections type.
• enum inputmux\_signal\_t {
  kINPUTMUX_Dmac0InputTriggerPint0Ena = 0U + (DMA0_ITRIG_EN0_ID << ENA_SHIFT) ,
  kINPUTMUX_Dmac0InputTriggerLcdTxRegToDmaSingleEna = 1U + (DMA0_ITRIG_EN1_ID
  << ENA_SHIFT) ,
  kINPUTMUX_Dmac1InputTriggerLcdTxRegToDmaSingleEna = 1U + (DMA1_ITRIG_EN1_ID
  << ENA_SHIFT) ,
  kINPUTMUX_Flexcomm14TxToDmac0Ch27RequestEna = 27U + (DMA0_REQ_ENA0_ID <<
ENA_SHIFT) }

INPUTMUX signal enable/disable type.

```

## 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\\_EnableSignal](#) (INPUTMUX\_Type \*base, [inputmux\\_signal\\_t](#) signal, bool enable)  
*Enable/disable a signal.*
- void [INPUTMUX\\_Deinit](#) (INPUTMUX\_Type \*base)  
*Deinitialize INPUTMUX peripheral.*

## Driver version

- #define [FSL\\_INPUTMUX\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 7))  
*Group interrupt driver version for SDK.*

## 29.4 Enumeration Type Documentation

### 29.4.1 enum [inputmux\\_connection\\_t](#)

Enumerator

*kINPUTMUX\_Gpio3Inp0ToSct0* SCT INMUX.  
*kINPUTMUX\_DebugHaltedToSct0* Pin Interrupt.  
*kINPUTMUX\_GpioPort1Pin31ToPintsel* Frequency measure.  
*kINPUTMUX\_AvpllCh2ClkoutToFreqmeas* CTmier0 capture input mux.  
*kINPUTMUX\_FlexcommDmaCmpltDone1ToTimer0CaptureChannels* CTmier1 capture input mux.  
*kINPUTMUX\_BtuHostTrigger2ToTimer1CaptureChannels* CTmier2 capture input mux.

*kINPUTMUX\_FlexcommDmaCmpltDone1ToTimer2CaptureChannels* CTmier3 capture input mux.

*kINPUTMUX\_BtuHostTrigger2ToTimer3CaptureChannels* DMA0 ITRIG.

*kINPUTMUX\_LcdTxRegToDmaSingleToDma0* DMA1 ITRIG.

*kINPUTMUX\_LcdTxRegToDmaSingleToDma1* DMA0 OTRIG.

*kINPUTMUX\_Dma0OtrigChannel32ToTrigInChannels* DMA1 OTRIG.

## 29.4.2 enum inputmux\_signal\_t

Enumerator

*kINPUTMUX\_Dmac0InputTriggerPint0Ena* DMA0 input trigger source enable.

*kINPUTMUX\_Dmac0InputTriggerLcdTxRegToDmaSingleEna* DMA1 input trigger source enable.

*kINPUTMUX\_Dmac1InputTriggerLcdTxRegToDmaSingleEna* DMA0 REQ signal.

*kINPUTMUX\_Flexcomm14TxToDmac0Ch27RequestEna* DMA1 REQ signal.

## 29.5 Function Documentation

### 29.5.1 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> |
|--------------|

### 29.5.2 void INPUTMUX\_AttachSignal( INPUTMUX\_Type \* *base*, uint32\_t *index*, inputmux\_connection\_t *connection* )

This function attaches multiplexed signals from INPUTMUX to target signals. For example, to attach GPIO PORT0 Pin 5 to PINT peripheral, do the following:

```
*     INPUTMUX_AttachSignal(INPUTMUX, 2, kINPUTMUX_GpioPort0Pin5ToPintsel);
*
```

In this example, INTMUX has 8 registers for PINT, PINT\_SEL0~PINT\_SEL7. With parameter *index* specified as 2, this function configures register PINT\_SEL2.

Parameters

|                   |                                                                                              |
|-------------------|----------------------------------------------------------------------------------------------|
| <i>base</i>       | Base address of the INPUTMUX peripheral.                                                     |
| <i>index</i>      | The serial number of destination register in the group of INPUTMUX registers with same name. |
| <i>connection</i> | Applies signal from source signals collection to target signal.                              |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 29.5.3 void INPUTMUX\_EnableSignal( INPUTMUX\_Type \* *base*, inputmux\_signal\_t *signal*, bool *enable* )

This function gates the INPUTPMUX clock.

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | Base address of the INPUTMUX peripheral.  |
| <i>signal</i> | Enable signal register id and bit offset. |
| <i>enable</i> | Selects enable or disable.                |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 29.5.4 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 30

## LCDIC: LCD Interface Controller

### 30.1 Overview

LCDIC is a MCU interface controller, the main features:

- Support QVGA (320\*240) or below
- Support RGB565 or below
- Support Tearing Effect Signal to eliminate tearing effect
- Support DMA transmission
- LCD refreshing rate 50/60Hz
- Frame updating rate 15 fps or below
- Support Intel 8080 interface with 8-bit data bus
- Support 3/4-wire SPI interface with 8-bit data bus

LCDIC module is initialized and configued by the function LCDIC\_Init. After calling this function, the LCDIC is ready to work.

LCDIC can send reset sequence to the reset pin to reset the LCD panel, call LCDIC\_SendResetSequence to send the reset sequence. To monitor the reset done event, application could poll the status or register callback by LCDIC\_SetResetSequenceDoneCallback.

For data send and receive, LCDIC driver provides two types of function, the first type is blocking functions, the function returns only when the transfer finished or error happens. Another type is non-blocking functions, the transfer start functions return before transfer done, application should get the transfer done event by register callback function.

### Modules

- [LCDIC DMA Driver](#)
- [LCDIC Driver](#)

## 30.2 LCDIC Driver

### 30.2.1 Overview

#### Data Structures

- struct `lcdic_config_t`  
*LCDIC configuration.* [More...](#)
- union `lcdic_trx_cmd_t`  
*LCDIC TRX command.* [More...](#)
- struct `lcdic_repeat_tx_xfer_t`  
*LCDIC repeat data TX transfer structure.* [More...](#)
- struct `lcdic_tx_xfer_t`  
*LCDIC data array TX transfer structure.* [More...](#)
- struct `lcdic_rx_xfer_t`  
*LCDIC data array RX transfer structure.* [More...](#)
- struct `lcdic_xfer_t`  
*LCDIC transfer structure.* [More...](#)
- struct `lcdic_handle_t`  
*LCDIC handle structure.* [More...](#)

#### Macros

- #define `LCDIC_RESET_STATE_DELAY` 130u  
*Delay used in LCDIC\_ResetState.*

#### Typedefs

- typedef void(\* `lcdic_transfer_callback_t`)*(LCDIC\_Type \*base, lcdic\_handle\_t \*handle, status\_t status, void \*userData)*  
*LCDIC transfer callback function.*
- typedef void(\* `lcdic_transfer_irq_handler_t`)*(LCDIC\_Type \*base, void \*handle)*  
*Typedef for transactional APIs IRQ handler.*
- typedef void(\* `lcdic_reset_done_callback_t`)*(LCDIC\_Type \*base)*  
*Typedef for reset sequence sent done callback.*

#### Enumerations

- enum `lcdic_mode_t` {  
`kLCDIC_3WireSPI` = 0U,  
`kLCDIC_4WireSPI` = LCDIC\_CTRL\_SPI\_MD\_MASK,  
`kLCDIC_I8080` = LCDIC\_CTRL\_LCDIC\_MD\_MASK }  
*LCDIC mode.*
- enum `lcdic_endian_t` {  
`kLCDIC_BigEndian` = 0U,  
`kLCDIC_LittleEndian` }

- *LCDIC byte order data endian.*
- enum `lcdic_rx_threshold_t` {
   
    `kLCDIC_RxThreshold0Word` = 0U,
   
    `kLCDIC_RxThreshold1Word` }
   
*LCDIC RX FIFO threshold.*
- enum `lcdic_tx_threshold_t` {
   
    `kLCDIC_TxThreshold0Word` = 0U,
   
    `kLCDIC_TxThreshold1Word`,
   
    `kLCDIC_TxThreshold2Word`,
   
    `kLCDIC_TxThreshold3Word`,
   
    `kLCDIC_TxThreshold4Word`,
   
    `kLCDIC_TxThreshold5Word`,
   
    `kLCDIC_TxThreshold6Word`,
   
    `kLCDIC_TxThreshold7Word` }
   
*LCDIC TX FIFO threshold.*
- enum `lcdic_reset_polarity_t` {
   
    `kLCDIC_ResetActiveLow` = 0U,
   
    `kLCDIC_ResetActiveHigh` }
- *LCDIC reset signal polarity.*
- enum {
   
    `kLCDIC_I8080_CsActiveLow` = 0U,
   
    `kLCDIC_I8080_CsActiveHigh` = `LCDIC_I8080_CTRL0_CS_POL_MASK`,
   
    `kLCDIC_I8080_DcCmdLow` = 0U,
   
    `kLCDIC_I8080_DcCmdHigh` = `LCDIC_I8080_CTRL0_DC_POL_MASK`,
   
    `kLCDIC_I8080_RdActiveLow` = 0U,
   
    `kLCDIC_I8080_RdActiveHigh` = `LCDIC_I8080_CTRL0_RD_POL_MASK`,
   
    `kLCDIC_I8080_WrActiveLow` = 0U,
   
    `kLCDIC_I8080_WrActiveHigh` = `LCDIC_I8080_CTRL0_WR_POL_MASK`,
   
    `kLCDIC_I8080_CsEnableIdleOff` = `LCDIC_I8080_CTRL0_EN_IDLE_OFF_MASK`,
   
    `kLCDIC_I8080_CsEnableDcSwitchOff` = `LCDIC_I8080_CTRL0_EN_DC_OFF_MASK` }
- *LCDIC I8080 control flags.*
- enum {
   
    `kLCDIC_SPI_MsbFirst` = 0U,
   
    `kLCDIC_SPI_LsbFirst` = `LCDIC_SPI_CTRL_SDAT_ENDIAN_MASK`,
   
    `kLCDIC_SPI_ClkActiveHigh` = 0U,
   
    `kLCDIC_SPI_ClkActiveLow` = `LCDIC_SPI_CTRL_CPOL_MASK`,
   
    `kLCDIC_SPI_ClkPhaseFirstEdge` = 0U,
   
    `kLCDIC_SPI_ClkPhaseSecondEdge` = `LCDIC_SPI_CTRL_CPHA_MASK`,
   
    `kLCDIC_SPI_DcCmdLow` = 0U,
   
    `kLCDIC_SPI_DcCmdHigh` = `LCDIC_SPI_CTRL_DC_POL_MASK` }
- *LCDIC SPI mode control flags.*
- enum `_lcdic_interrupt` { , `kLCDIC_AllInterrupt` }
- *LCDIC interrupts.*
- enum { , `kLCDIC_AllFlag` }
- *LCDIC status flags.*
- enum {

- ```
kLCDIC_TeNoSync = 0,
kLCDIC_TeRisingEdgeSync,
kLCDIC_TeFallingEdgeSync }
```

*LCDIC TE sync mode.*
- enum {

```
kLCDIC_ShortTimeout = 0,
kLCDIC_LongTimeout }
```

*LCDIC TRX command timeout mode.*
- enum {

```
kLCDIC_DataFormatByte = 0,
kLCDIC_DataFormatHalfWord,
kLCDIC_DataFormatWord }
```

*LCDIC data format.*
- enum {

```
kLCDIC_Command = 0,
kLCDIC_Data }
```

*LCDIC data or command.*
- enum {

```
kLCDIC_RX = 0,
kLCDIC_TX }
```

*LCDIC TX or RX.*
- enum `lcdic_xfer_mode_t` {

```
kLCDIC_XferCmdOnly = 0,
kLCDIC_XferSendRepeatData,
kLCDIC_XferSenddataArray,
kLCDIC_XferReceivedataArray }
```

*LCDIC transfer mode.*

## Driver version

- #define **FSL\_LCDIC\_DRIVER\_VERSION** (`MAKE_VERSION(2, 1, 0)`)

## Initialization and deinitialization

- `status_t LCDIC_Init (LCDIC_Type *base, const lcdic_config_t *config)`

*Initialize the LCDIC.*
- void `LCDIC_Deinit (LCDIC_Type *base)`

*De-initialize the LCDIC.*
- void `LCDIC_GetDefaultConfig (lcdic_config_t *config)`

*Get the default configuration for to initialize the LCDIC.*
- void `LCDIC_ResetState (LCDIC_Type *base)`

*Reset the LCDIC.*

## Interrupts

- static void **LCDIC\_EnableInterrupts** (LCDIC\_Type \*base, uint32\_t interrupts)  
*Enables LCDIC interrupts.*
- static void **LCDIC\_DisableInterrupts** (LCDIC\_Type \*base, uint32\_t interrupts)  
*Disable LCDIC interrupts.*
- static uint32\_t **LCDIC\_GetInterruptStatus** (LCDIC\_Type \*base)  
*Get LCDIC interrupt pending status.*
- static uint32\_t **LCDIC\_GetInterruptRawStatus** (LCDIC\_Type \*base)  
*Get LCDIC raw interrupt status.*
- static void **LCDIC\_ClearInterruptStatus** (LCDIC\_Type \*base, uint32\_t interrupts)  
*Clear LCDIC interrupt status.*
- static uint32\_t **LCDIC\_GetStatusFlags** (LCDIC\_Type \*base)  
*Get LCDIC status flags.*
- static uint32\_t **LCDIC\_GetProcessingTrxCmd** (LCDIC\_Type \*base)  
*Get current on-going LCDIC TRX-CMD.*

## FIFO

- static void **LCDIC\_SetTxThreshold** (LCDIC\_Type \*base, lcdic\_tx\_threshold\_t threshold)  
*Set TX FIFO threshold.*
- static void **LCDIC\_SetRxThreshold** (LCDIC\_Type \*base, lcdic\_rx\_threshold\_t threshold)  
*Set RX FIFO threshold.*
- status\_t **LCDIC\_WriteTxFifoBlocking** (LCDIC\_Type \*base, const uint32\_t \*data, uint32\_t dataLen\_Word)  
*Write the TX FIFO using blocking way.*
- status\_t **LCDIC\_ReadRxFifoBlocking** (LCDIC\_Type \*base, uint32\_t \*data, uint32\_t dataLen\_Word)  
*Read the RX FIFO using blocking way.*

## Misc Operations

- static void **LCDIC\_SendResetSequence** (LCDIC\_Type \*base)  
*Send reset sequence to the reset pin.*
- void **LCDIC\_SetResetSequenceDoneCallback** (lcdic\_reset\_done\_callback\_t callback)  
*Set the callback called when reset sequence sent done.*
- static void **LCDIC\_EnableDMA** (LCDIC\_Type \*base, bool enable)  
*Enable or disable to trigger DMA.*

## Blocking transfer

- status\_t **LCDIC\_SendCommandBlocking** (LCDIC\_Type \*base, uint8\_t cmd)  
*Send command using blocking way.*
- status\_t **LCDIC\_SendRepeatDataBlocking** (LCDIC\_Type \*base, const lcdic\_repeat\_tx\_xfer\_t \*xfer)  
*Send repeat data using blocking way.*

- `status_t LCDIC_SendDataArrayBlocking (LCDIC_Type *base, const lcdic_tx_xfer_t *xfer)`  
*Send data array using blocking way.*
- `status_t LCDIC_ReadDataArrayBlocking (LCDIC_Type *base, const lcdic_rx_xfer_t *xfer)`  
*Read data array using blocking way.*
- `status_t LCDC_TransferBlocking (LCDIC_Type *base, const lcdic_xfer_t *xfer)`  
*LCDIC data transfer using blocking way.*

## Transactional APIs

- `status_t LCDIC_TransferCreateHandle (LCDIC_Type *base, lcdic_handle_t *handle, lcdic_transfer_callback_t callback, void *userData)`  
*Initializes the LCDIC driver handle, which is used in transactional functions.*
- `status_t LCDIC_TransferNonBlocking (LCDIC_Type *base, lcdic_handle_t *handle, lcdic_xfer_t *xfer)`  
*Transfer data using IRQ.*
- `status_t LCDIC_SendCommandNonBlocking (LCDIC_Type *base, lcdic_handle_t *handle, uint8_t cmd)`  
*Send command using interrupt-driven way.*
- `status_t LCDIC_SendRepeatDataNonBlocking (LCDIC_Type *base, lcdic_handle_t *handle, const lcdic_repeat_tx_xfer_t *xfer)`  
*Send repeat data using interrupt-driven way.*
- `status_t LCDIC_SendDataArrayNonBlocking (LCDIC_Type *base, lcdic_handle_t *handle, const lcdic_tx_xfer_t *xfer)`  
*Send data array using interrupt-driven way.*
- `status_t LCDIC_ReadDataArrayNonBlocking (LCDIC_Type *base, lcdic_handle_t *handle, const lcdic_rx_xfer_t *xfer)`  
*Read data array using interrupt-driven way.*
- `void LCDIC_TransferHandleIRQ (LCDIC_Type *base, void *handle)`  
*LCDIC IRQ handler function.*
- `void LCDIC_TransferInstallIRQHandler (uint32_t instance, void *handle, lcdic_transfer_irq_handler_t handler)`  
*Install the IRQ handler.*

## Helper functions

- `uint32_t LCDIC_GetInstance (LCDIC_Type *base)`  
*Get the instance from the base address.*
- `IRQn_Type LCDIC_GetIRQn (uint32_t instance)`  
*Get IRQn for specific instance.*
- `uint32_t LCDIC_FillByteToWord (const uint8_t *bytes, uint8_t len)`  
*Get data from byte array, and fill to 4-byte word.*
- `void LCDIC_ExtractByteFromWord (uint32_t word, uint8_t *bytes, uint8_t len)`  
*Get data from 4-byte, and fill to byte array.*
- `status_t LCDIC_PreparesendCommand (LCDIC_Type *base, uint8_t cmd)`  
*Prepare the command sending.*
- `status_t LCDIC_PreparesendRepeatData (LCDIC_Type *base, const lcdic_repeat_tx_xfer_t *xfer)`  
*Prepare the repeat data sending.*

- **status\_t LCDIC\_PreparesendDataArray** (LCDIC\_Type \*base, const **lcdic\_tx\_xfer\_t** \*xfer, uint32\_t \*xferSizeWordAligned, uint8\_t \*xferSizeWordUnaligned, uint32\_t \*wordUnalignedData)  
*Prepare sending data array.*
- **status\_t LCDIC\_PreparesreadDataArray** (LCDIC\_Type \*base, const **lcdic\_rx\_xfer\_t** \*xfer, uint32\_t \*xferSizeWordAligned, uint8\_t \*xferSizeWordUnaligned)  
*Prepare reading data array.*

## 30.2.2 Data Structure Documentation

### 30.2.2.1 struct lcdic\_config\_t

#### Data Fields

- **lcdic\_mode\_t mode**  
*LCDIC work mode.*
- **lcdic\_endian\_t endian**  
*Data endian.*
- **lcdic\_rx\_threshold\_t rxThreshold**  
*RX FIFO threshold.*
- **lcdic\_tx\_threshold\_t txThreshold**  
*TX FIFO threshold.*
- **uint8\_t timerRatio0**  
*Valid range: 0~15.*
- **uint8\_t timerRatio1**  
*Valid range: 0~15.*
- **uint8\_t resetPulseWidth\_Timer0**  
*Reset pulse width, in the unit of timer0 period.*
- **uint8\_t resetSequence**  
*Reset sequence, it is a 8-bit value sent to reset pin from LSB.*
- **uint8\_t resetSequencePulseNum**  
*Reset sequence pulse number, valid range is 1 ~ 8.*
- **lcdic\_reset\_polarity\_t resetPolarity**  
*Reset signal polarity.*
- **uint8\_t i8080CtrlFlags**  
*I8080 control flags, it is OR'ed value of [\\_lcdic\\_i8080\\_ctrl\\_flags](#).*
- **uint8\_t csWaitTime**  
*Minimum CS inactive pulse width.*
- **uint8\_t csSetupTime**  
*Minimum CS setup time before WR/RD.*
- **uint8\_t csHoldTime**  
*Minimum CS hold time after WR/RD.*
- **uint8\_t dcSetupTime**  
*Minimum DC setup time before WR/RD/CS.*
- **uint8\_t dcHoldTime**  
*Minimum DC hold time after WR/RD/CS.*
- **uint8\_t writeDataSetupTime**  
*Minimum write data setup time after WR active.*
- **uint8\_t writeDataHoldTime**  
*Minimum write data setup time before WR active.*

- `uint8_t writeEnableActiveWidth`  
*Minimum write enable active pulse width.*
- `uint8_t writeEnableInactiveWidth`  
*Minimum write enable inactive pulse width.*
- `uint8_t readEnableActiveWidth`  
*Minimum read enable active pulse width.*
- `uint8_t readEnableInactiveWidth`  
*Minimum read enable inactive pulse width.*
- `uint8_t spiCtrlFlags`  
*SPI control flags, it is OR'ed value of `_lcdic_spi_ctrl_flags`.*
- `uint8_t teTimeoutTime_Timer1`  
*Tearing effect timeout time.*
- `uint8_t teSyncWaitTime_Timer1`  
*Tearing effect signal synchronization wait time.*
- `uint8_t cmdShortTimeout_Timer0`  
*Command short timeout.*
- `uint8_t cmdLongTimeout_Timer1`  
*Command long timeout.*

## Field Documentation

- (1) `lcdic_mode_t lcdic_config_t::mode`
- (2) `lcdic_endian_t lcdic_config_t::endian`
- (3) `lcdic_rx_threshold_t lcdic_config_t::rxThreshold`
- (4) `lcdic_tx_threshold_t lcdic_config_t::txThreshold`
- (5) `uint8_t lcdic_config_t::timerRatio0`

`freq(timer0) = freq(lcdic_clk) / (2 ^ timerRatio0).`

- (6) `uint8_t lcdic_config_t::timerRatio1`

`freq(timer1) = freq(timer0) / (2 ^ timerRatio1).`

- (7) `uint8_t lcdic_config_t::resetPulseWidth_Timer0`

Valid range 1 ~ 64.

- (8) **uint8\_t lcdic\_config\_t::resetSequence**
- (9) **uint8\_t lcdic\_config\_t::resetSequencePulseNum**
- (10) **lcdic\_reset\_polarity\_t lcdic\_config\_t::resetPolarity**
- (11) **uint8\_t lcdic\_config\_t::i8080CtrlFlags**
- (12) **uint8\_t lcdic\_config\_t::csWaitTime**

$T(\text{csW}) = T(\text{lcdic\_clk}) * \text{csWaitTime}$ , valid range 0-7.

- (13) **uint8\_t lcdic\_config\_t::csSetupTime**

$T(\text{css}) = T(\text{lcdic\_clk}) * \text{csSetupTime}$ , valid range 0-255.

- (14) **uint8\_t lcdic\_config\_t::csHoldTime**

$T(\text{csh}) = T(\text{lcdic\_clk}) * \text{csHoldTime}$ , valid range 0-7.

- (15) **uint8\_t lcdic\_config\_t::dcSetupTime**

$T(\text{dcs}) = T(\text{lcdic\_clk}) * \text{dsSetupTime}$ , valid range 0-7.

- (16) **uint8\_t lcdic\_config\_t::dcHoldTime**

$T(\text{dch}) = T(\text{lcdic\_clk}) * \text{dsHoldTime}$ , valid range 0-7.

- (17) **uint8\_t lcdic\_config\_t::writeDataSetupTime**

$T(\text{wdh}) = T(\text{lcdic\_clk}) * \text{writeDataSetupTime}$ , valid range 0-7.

- (18) **uint8\_t lcdic\_config\_t::writeDataHoldTime**

$T(\text{wds}) = T(\text{lcdic\_clk}) * \text{writeDataHoldTime}$ , valid range 0-7.

- (19) **uint8\_t lcdic\_config\_t::writeEnableActiveWidth**

$T(\text{waw}) = T(\text{lcdic\_clk}) * \text{writeEnableActiveWidth}$ , valid range 0-63.

- (20) **uint8\_t lcdic\_config\_t::writeEnableInactiveWidth**

$T(\text{wiw}) = T(\text{lcdic\_clk}) * \text{writeEnableInactiveWidth}$ , valid range 0-63.

- (21) **uint8\_t lcdic\_config\_t::readEnableActiveWidth**

$T(\text{raw}) = T(\text{lcdic\_clk}) * \text{readEnableActiveWidth}$ , valid range 0-255.

- (22) **uint8\_t lcdic\_config\_t::readEnableInactiveWidth**

$T(\text{riw}) = T(\text{lcdic\_clk}) * \text{readEnableInactiveWidth}$ , valid range 0-255.

- (23) `uint8_t lcdic_config_t::spiCtrlFlags`  
 (24) `uint8_t lcdic_config_t::teTimeoutTime_Timer1`  
 $T(\text{te\_to}) = T(\text{timer1}) * \text{teTimeoutTime\_Timer1}.$

- (25) `uint8_t lcdic_config_t::teSyncWaitTime_Timer1`  
 $T(\text{tew}) = T(\text{timer1}) * \text{teSyncWaitTime\_Timer1}.$
- (26) `uint8_t lcdic_config_t::cmdShortTimeout_Timer0`  
 $T(\text{cmd\_short\_to}) = T(\text{timer0}) * \text{cmdShortTimeout\_Timer0}.$
- (27) `uint8_t lcdic_config_t::cmdLongTimeout_Timer1`  
 $T(\text{cmd\_long\_to}) = T(\text{timer1}) * \text{cmdLongTimeout\_Timer1}.$

### 30.2.2.2 union lcdic\_trx\_cmd\_t

#### Field Documentation

- (1) `uint32_t lcdic_trx_cmd_t::dataLen`
- (2) `uint32_t lcdic_trx_cmd_t::dummyCount`
- (3) `uint32_t lcdic_trx_cmd_t::useAutoRepeat`
- (4) `uint32_t lcdic_trx_cmd_t::teSyncMode`
- (5) `uint32_t lcdic_trx_cmd_t::trxTimeoutMode`
- (6) `uint32_t lcdic_trx_cmd_t::dataFormat`
- (7) `uint32_t lcdic_trx_cmd_t::enableCmdDoneInt`
- (8) `uint32_t lcdic_trx_cmd_t::cmdOrData`
- (9) `uint32_t lcdic_trx_cmd_t::trx`

### 30.2.2.3 struct lcdic\_repeat\_tx\_xfer\_t

#### Data Fields

- `uint8_t cmd`  
*Command.*
- `uint8_t teSyncMode`  
*TE sync mode, see [\\_lcdic\\_te\\_sync\\_mode](#).*
- `uint8_t trxTimeoutMode`  
*TRX command timeout mode, see [\\_lcdic\\_trx\\_timeout\\_mode](#).*

- `uint8_t dataFormat`  
*Data format, see [\\_lcdic\\_data\\_format](#).*
- `uint32_t dataLen`  
*Data length.*
- `uint32_t txRepeatData`  
*The repeat data.*

## Field Documentation

- (1) `uint8_t lcdic_repeat_tx_xfer_t::cmd`
- (2) `uint8_t lcdic_repeat_tx_xfer_t::teSyncMode`
- (3) `uint8_t lcdic_repeat_tx_xfer_t::trxTimeoutMode`
- (4) `uint8_t lcdic_repeat_tx_xfer_t::dataFormat`
- (5) `uint32_t lcdic_repeat_tx_xfer_t::dataLen`
- (6) `uint32_t lcdic_repeat_tx_xfer_t::txRepeatData`

### 30.2.2.4 struct `lcdic_tx_xfer_t`

#### Data Fields

- `uint8_t cmd`  
*Command.*
- `uint8_t teSyncMode`  
*TE sync mode, see [\\_lcdic\\_te\\_sync\\_mode](#).*
- `uint8_t trxTimeoutMode`  
*TRX command timeout mode, see [\\_lcdic\\_trx\\_timeout\\_mode](#).*
- `uint8_t dataFormat`  
*Data format, see [\\_lcdic\\_data\\_format](#).*
- `uint32_t dataLen`  
*Data length.*
- `const uint8_t * txData`  
*The data to send.*

**Field Documentation**

- (1) `uint8_t lcdic_tx_xfer_t::cmd`
- (2) `uint8_t lcdic_tx_xfer_t::teSyncMode`
- (3) `uint8_t lcdic_tx_xfer_t::trxTimeoutMode`
- (4) `uint8_t lcdic_tx_xfer_t::dataFormat`
- (5) `uint32_t lcdic_tx_xfer_t::dataLen`
- (6) `const uint8_t* lcdic_tx_xfer_t::txData`

**30.2.2.5 struct lcdic\_rx\_xfer\_t****Data Fields**

- `uint8_t cmd`  
*Command.*
- `uint8_t dummyCount`  
*Dummy cycle between TX and RX, only used for SPI mode.*
- `uint8_t trxTimeoutMode`  
*TRX command timeout mode, see [\\_lcdic\\_trx\\_timeout\\_mode](#).*
- `uint8_t dataFormat`  
*Data format, see [\\_lcdic\\_data\\_format](#).*
- `uint32_t dataLen`  
*Data length.*
- `uint8_t * rxData`  
*Pointer to the data receive array.*

**Field Documentation**

- (1) `uint8_t lcdic_rx_xfer_t::cmd`
- (2) `uint8_t lcdic_rx_xfer_t::dummyCount`
- (3) `uint8_t lcdic_rx_xfer_t::trxTimeoutMode`
- (4) `uint8_t lcdic_rx_xfer_t::dataFormat`
- (5) `uint32_t lcdic_rx_xfer_t::dataLen`
- (6) `uint8_t* lcdic_rx_xfer_t::rxData`

**30.2.2.6 struct lcdic\_xfer\_t****Data Fields**

- `lcdic_xfer_mode_t mode`  
*Transfer mode.*

- `uint8_t cmdToSendOnly`  
*Command to send in mode `kLCDIC_XferCmdOnly`.*
- `lcdic_repeat_tx_xfer_t repeatTxXfer`  
*For mode `kLCDIC_XferSendRepeatData`.*
- `lcdic_tx_xfer_t txXfer`  
*For mode `kLCDIC_XferSenddataArray`.*
- `lcdic_rx_xfer_t rxXfer`  
*For mode `kLCDIC_XferReceivedataArray`.*

## Field Documentation

- (1) `lcdic_xfer_mode_t lcdic_xfer_t::mode`
- (2) `uint8_t lcdic_xfer_t::cmdToSendOnly`
- (3) `lcdic_repeat_tx_xfer_t lcdic_xfer_t::repeatTxXfer`
- (4) `lcdic_tx_xfer_t lcdic_xfer_t::txXfer`
- (5) `lcdic_rx_xfer_t lcdic_xfer_t::rxXfer`

### 30.2.2.7 struct \_lcdic\_handle

#### Data Fields

- `volatile bool xferInProgress`  
*Transfer in progress.*
- `lcdic_xfer_mode_t xferMode`  
*On-going transfer mode.*
- `lcdic_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*LCDIC callback function parameter.*
- `uint32_t xferSizeWordAligned`  
*4-byte aligned part of the transfer size.*
- `uint8_t xferSizeWordUnaligned`  
*4-byte unaligned part of the transfer size.*
- `uint32_t tmpData`  
*Temp data for driver internal use.*
- `const uint8_t * txData`  
*Data array to send.*
- `uint8_t * rxData`  
*RX data array.*

## Field Documentation

- (1) volatile bool `lcdic_handle_t::xferInProgress`
- (2) `lcdic_xfer_mode_t lcdic_handle_t::xferMode`
- (3) `lcdic_transfer_callback_t lcdic_handle_t::callback`
- (4) `void* lcdic_handle_t::userData`
- (5) `const uint8_t* lcdic_handle_t::txData`
- (6) `uint8_t* lcdic_handle_t::rxData`
- (7) `uint32_t lcdic_handle_t::xferSizeWordAligned`
- (8) `uint8_t lcdic_handle_t::xferSizeWordUnaligned`
- (9) `uint32_t lcdic_handle_t::tmpData`

### 30.2.3 Macro Definition Documentation

#### 30.2.3.1 `#define LCDIC_RESET_STATE_DELAY 130u`

This should be larger than 5 \* core clock / LCDIC function clock.

### 30.2.4 Typedef Documentation

#### 30.2.4.1 `typedef void(* lcdic_transfer_callback_t)(LCDIC_Type *base, lcdic_handle_t *handle, status_t status, void *userData)`

The status is `kStatus_Success` when transfer finished successfully, it is `kStatus_Timeout` when timeout happened.

#### 30.2.4.2 `typedef void(* lcdic_transfer_irq_handler_t)(LCDIC_Type *base, void *handle)`

#### 30.2.4.3 `typedef void(* lcdic_reset_done_callback_t)(LCDIC_Type *base)`

### 30.2.5 Enumeration Type Documentation

#### 30.2.5.1 `enum lcdic_mode_t`

Enumerator

`kLCDIC_3WireSPI` 3-wire SPI mode.

`kLCDIC_4WireSPI` 4-wire SPI mode.

*kLCDIC\_I8080* I8080 mode.

### 30.2.5.2 enum lcdic\_endian\_t

Enumerator

*kLCDIC\_BigEndian* Big endian.

*kLCDIC\_LittleEndian* Little endian.

### 30.2.5.3 enum lcdic\_rx\_threshold\_t

RX threshold interrupt happens if the occupied word number in RX FIFO is bigger than the threshold value.

Enumerator

*kLCDIC\_RxThreshold0Word* 0 word.

*kLCDIC\_RxThreshold1Word* 1 word.

### 30.2.5.4 enum lcdic\_tx\_threshold\_t

TX threshold interrupt happens if the empty word number in TX FIFO is bigger than the threshold value.

Enumerator

*kLCDIC\_TxThreshold0Word* 0 word.

*kLCDIC\_TxThreshold1Word* 1 word.

*kLCDIC\_TxThreshold2Word* 2 word.

*kLCDIC\_TxThreshold3Word* 3 word.

*kLCDIC\_TxThreshold4Word* 4 word.

*kLCDIC\_TxThreshold5Word* 5 word.

*kLCDIC\_TxThreshold6Word* 6 word.

*kLCDIC\_TxThreshold7Word* 7 word.

### 30.2.5.5 enum lcdic\_reset\_polarity\_t

Enumerator

*kLCDIC\_ResetActiveLow* Active low.

*kLCDIC\_ResetActiveHigh* Active high.

### 30.2.5.6 anonymous enum

Enumerator

*kLCDIC\_I8080\_CsActiveLow* CS active low.  
*kLCDIC\_I8080\_CsActiveHigh* CS active high.  
*kLCDIC\_I8080\_DcCmdLow* DC 0 means command, while 1 means data.  
*kLCDIC\_I8080\_DcCmdHigh* DC 1 means command, while 0 means data.  
*kLCDIC\_I8080\_RdActiveLow* RD active low.  
*kLCDIC\_I8080\_RdActiveHigh* RD active high.  
*kLCDIC\_I8080\_WrActiveLow* WR active low.  
*kLCDIC\_I8080\_WrActiveHigh* WR active high.  
*kLCDIC\_I8080\_CsEnableIdleOff* CS off while no transmission.  
*kLCDIC\_I8080\_CsEnableDcSwitchOff* CS off while DC switches.

### 30.2.5.7 anonymous enum

Enumerator

*kLCDIC\_SPI\_MsbFirst* MSB(bit 7) sent and received first.  
*kLCDIC\_SPI\_LsbFirst* LSB(bit 0) sent and received first.  
*kLCDIC\_SPI\_ClkActiveHigh* CPOL=0. Clock active-high (idle low)  
*kLCDIC\_SPI\_ClkActiveLow* CPOL=1. Clock active-low (idle high)  
*kLCDIC\_SPI\_ClkPhaseFirstEdge* CPHA=0. Data sample at first clock edge.  
*kLCDIC\_SPI\_ClkPhaseSecondEdge* CPHA=1. Data sample at second clock edge.  
*kLCDIC\_SPI\_DcCmdLow* DC 0 means command, while 1 means data.  
*kLCDIC\_SPI\_DcCmdHigh* DC 1 means command, while 0 means data.

### 30.2.5.8 enum \_lcdic\_interrupt

Enumerator

*kLCDIC\_AllInterrupt* All interrupts.

### 30.2.5.9 anonymous enum

Enumerator

*kLCDIC\_AllFlag* All flags.

### 30.2.5.10 anonymous enum

Enumerator

*kLCDIC\_TeNoSync* Don't need to sync.

*kLCDIC\_TeRisingEdgeSync* Sync to TE rising edge.  
*kLCDIC\_TeFallingEdgeSync* Sync to TE falling edge.

### 30.2.5.11 anonymous enum

Enumerator

*kLCDIC\_ShortTimeout* Using short timeout.  
*kLCDIC\_LongTimeout* Using long timeout.

### 30.2.5.12 anonymous enum

Enumerator

*kLCDIC\_DataFormatByte* Byte.  
*kLCDIC\_DataFormatHalfWord* Half word (2-byte).  
*kLCDIC\_DataFormatWord* Word (4-byte).

### 30.2.5.13 anonymous enum

Enumerator

*kLCDIC\_Command* Command.  
*kLCDIC\_Data* Data.

### 30.2.5.14 anonymous enum

Enumerator

*kLCDIC\_RX* RX.  
*kLCDIC\_TX* TX.

### 30.2.5.15 enum lcdic\_xfer\_mode\_t

Enumerator

*kLCDIC\_XferCmdOnly* Only send command.  
*kLCDIC\_XferSendRepeatData* Send repeat data.  
*kLCDIC\_XferSenddataArray* Send data array.  
*kLCDIC\_XferReceivedataArray* Receive data array.

## 30.2.6 Function Documentation

### 30.2.6.1 status\_t LCDIC\_Init ( LCDIC\_Type \* *base*, const lcdic\_config\_t \* *config* )

This function initializes the LCDIC to work, it configures the LCDIC according to the config structure and enables the module. After calling this function, the peripheral is ready to work.

## Parameters

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

## Return values

<i>kStatus_Success</i>	Initialize successfully.
------------------------	--------------------------

**30.2.6.2 void LCDIC\_Deinit ( LCDIC\_Type \* *base* )**

This function disables the LCDIC, and disables peripheral clock if necessary.

## Parameters

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

**30.2.6.3 void LCDIC\_GetDefaultConfig ( lcdic\_config\_t \* *config* )**

The default configuration value is:

```

config->mode          = kLCDIC_3WireSPI;
config->endian        = kLCDIC_BigEndian;
config->rxThreshold  = kLCDIC_RxThreshold0Word;
config->txThreshold  = kLCDIC_TxThreshold3Word;

config->timerRatio0  = 8;
config->timerRatio1  = 9;

config->resetPulseWidth_Timer0 = 20;
config->resetSequence      = 0;
config->resetSequencePulseNum = 1;
config->resetPolarity       = kLCDIC_ResetActiveLow;

config->i8080CtrlFlags = kLCDIC_I8080_CsActiveLow |
    kLCDIC_I8080_DcCmdLow | kLCDIC_I8080_RdActiveLow |
    kLCDIC_I8080_WrActiveLow |
    kLCDIC_I8080_CsEnableIdleOff;

config->csWaitTime      = 2;
config->csSetupTime     = 2;
config->csHoldTime      = 2;
config->dcSetupTime     = 2;
config->dcHoldTime      = 2;
config->writeDataSetupTime = 2;
config->writeDataHoldTime = 2;
config->writeEnableActiveWidth = 6;
config->writeEnableInactiveWidth = 6;
config->readEnableActiveWidth = 15;
config->readEnableInactiveWidth = 15;

config->spiCtrlFlags =
    kLCDIC_SPI_MsbFirst | kLCDIC_SPI_ClkActiveHigh |
    kLCDIC_SPI_ClkPhaseFirstEdge | kLCDIC_SPI_DcCmdLow;

config->teTimeoutTime_Timer1 = 16;

```

```
config->teSyncWaitTime_Timer1 = 0;
config->cmdShortTimeout_Timer0 = 1;
config->cmdLongTimeout_Timer1 = 16;
```

Parameters

<i>config</i>	Pointer to the LCDIC configuration.
---------------	-------------------------------------

#### 30.2.6.4 void LCDIC\_ResetState ( LCDIC\_Type \* *base* )

This function resets the LCDIC state. After calling this function, all data in TX\_FIFO and RX\_FIFO will be cleared and all transactions on LCD interface will restart despite of formal status.

The configurations will not be reset.

Parameters

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

#### 30.2.6.5 static void LCDIC\_EnableInterrupts ( LCDIC\_Type \* *base*, uint32\_t *interrupts* ) [inline], [static]

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>interrupts</i>	The interrupts to enable, pass in as OR'ed value of <a href="#">_lcdic_interrupt</a> .

#### 30.2.6.6 static void LCDIC\_DisableInterrupts ( LCDIC\_Type \* *base*, uint32\_t *interrupts* ) [inline], [static]

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>interrupts</i>	The interrupts to disable, pass in as OR'ed value of <a href="#">_lcdic_interrupt</a> .

#### 30.2.6.7 static uint32\_t LCDIC\_GetInterruptStatus ( LCDIC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The interrupt pending status.

Note

The interrupt must be enabled, otherwise the interrupt flags will not assert.

### 30.2.6.8 static uint32\_t LCDIC\_GetInterruptRawStatus ( LCDIC\_Type \* *base* ) [inline], [static]

This function gets the raw interrupt pending flags, it is not affected by interrupt enabled status.

Parameters

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

Returns

The raw interrupt status.

### 30.2.6.9 static void LCDIC\_ClearInterruptStatus ( LCDIC\_Type \* *base*, uint32\_t *interrupts* ) [inline], [static]

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>interrupts</i>	The interrupt status to clear , pass in as OR'ed value of <a href="#">_lcdic_interrupt</a> .

### 30.2.6.10 static uint32\_t LCDIC\_GetStatusFlags ( LCDIC\_Type \* *base* ) [inline], [static]

Note

The interval between two times calling this function shall be larger than one LCDIC function clock.

Parameters

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

Returns

The status flags, it is OR'ed value of \_ldic\_flags.

### 30.2.6.11 static uint32\_t LCDIC\_GetProcessingTrxCmd ( LCDIC\_Type \* *base* ) [inline], [static]

Note

The interval between two times calling this function shall be larger than one LCDIC function clock.

Parameters

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

Returns

The TRX-CMD on-going.

### 30.2.6.12 static void LCDIC\_SetTxThreshold ( LCDIC\_Type \* *base*, lcdic\_tx\_threshold\_t *threshold* ) [inline], [static]

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>threshold</i>	TX threshold.

### 30.2.6.13 static void LCDIC\_SetRxThreshold ( LCDIC\_Type \* *base*, lcdic\_rx\_threshold\_t *threshold* ) [inline], [static]

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>threshold</i>	RX threshold.

### 30.2.6.14 status\_t LCDIC\_WriteTxFifoBlocking ( LCDIC\_Type \* *base*, const uint32\_t \* *data*, uint32\_t *dataLen\_Word* )

This function waits for empty slot in TX FIFO and fill the data to TX FIFO.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>data</i>	Data to send, the data length must be dividable by 4.
<i>dataLen_Word</i>	Data length in word.

Return values

<i>kStatus_Success</i>	Write successfully.
<i>kStatus_Timeout</i>	Timeout happened.

### 30.2.6.15 status\_t LCDIC\_ReadRxFifoBlocking ( LCDIC\_Type \* *base*, uint32\_t \* *data*, uint32\_t *dataLen\_Word* )

This function waits for valid data in RX FIFO and read them.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>data</i>	Array for received data, the data length must be dividable by 4.
<i>dataLen_Word</i>	Data length in word.

Return values

<i>kStatus_Success</i>	Read successfully.
<i>kStatus_Timeout</i>	Timeout happened.

### 30.2.6.16 static void LCDIC\_SendResetSequence ( LCDIC\_Type \* *base* ) [inline], [static]

The function sends reset to reset pin, to reset the external panel. The reset sequence parameters are configued by [lc\(dic\)\\_config\\_t](#).

Parameters

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

### 30.2.6.17 void LCDIC\_SetResetSequenceDoneCallback ( lcdic\_reset\_done\_callback\_t callback )

Parameters

<i>callback</i>	The callback to set.
-----------------	----------------------

### 30.2.6.18 static void LCDIC\_EnableDMA ( LCDIC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>enable</i>	Use true to enable, false to disable.

### 30.2.6.19 status\_t LCDIC\_SendCommandBlocking ( LCDIC\_Type \* *base*, uint8\_t *cmd* )

This function sends out command and waits until send finished.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>cmd</i>	Command to send.

Return values

<i>kStatus_Success</i>	Command sent successfully.
------------------------	----------------------------

### 30.2.6.20 status\_t LCDIC\_SendRepeatDataBlocking ( LCDIC\_Type \* *base*, const lcdic\_repeat\_tx\_xfer\_t \* *xfer* )

This function sends out command and the repeat data, then waits until send finished or timeout happened.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>xfer</i>	Pointer to the transfer configuration.

Return values

<i>kStatus_Success</i>	Sent successfully.
<i>kStatus_Timeout</i>	Timeout happened.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 30.2.6.21 status\_t LCDIC\_SendDataArrayBlocking ( LCDIC\_Type \* *base*, const lcdic\_tx\_xfer\_t \* *xfer* )

This function sends out command and the data array, then waits until send finished or timeout happened.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>xfer</i>	Pointer to the transfer configuration.

Return values

<i>kStatus_Success</i>	Sent successfully.
<i>kStatus_Timeout</i>	Timeout happened.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 30.2.6.22 status\_t LCDIC\_ReadDataArrayBlocking ( LCDIC\_Type \* *base*, const lcdic\_rx\_xfer\_t \* *xfer* )

This function sends out command and read the data array, then waits until send finished or timeout happened.

Parameters

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

<i>xfer</i>	Pointer to the transfer configuration.
-------------	--

Return values

<i>kStatus_Success</i>	Sent successfully.
<i>kStatus_Timeout</i>	Timeout happened.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 30.2.6.23 status\_t LCDIC\_TransferBlocking ( LCDIC\_Type \* *base*, const lcdic\_xfer\_t \* *xfer* )

This function sends command only, or sends repeat data, or sends data array, or reads data array based on the transfer structure. It uses blocking way, only returns when transfer successed or failed.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>xfer</i>	Pointer to the transfer configuration.

Return values

<i>kStatus_Success</i>	Sent successfully.
<i>kStatus_Timeout</i>	Timeout happened.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 30.2.6.24 status\_t LCDIC\_TransferCreateHandle ( LCDIC\_Type \* *base*, lcdic\_handle\_t \* *handle*, lcdic\_transfer\_callback\_t *callback*, void \* *userData* )

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the lcdic_handle_t structure to store the transfer state.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

Return values

<i>kStatus_Success</i>	Successfully created the handle.
------------------------	----------------------------------

### 30.2.6.25 status\_t LCDIC\_TransferNonBlocking ( *LCDIC\_Type* \* *base*, *Icdic\_handle\_t* \* *handle*, *Icdic\_xfer\_t* \* *xfer* )

This function transfer data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, or timeout happened, the callback function is called.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the <i>Icdic_handle_t</i> structure to store the transfer state.
<i>xfer</i>	LCDIC transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_Busy</i>	LCDIC driver is busy with another transfer.

### 30.2.6.26 status\_t LCDIC\_SendCommandNonBlocking ( *LCDIC\_Type* \* *base*, *Icdic\_handle\_t* \* *handle*, *uint8\_t* *cmd* )

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the <i>Icdic_handle_t</i> structure to store the transfer state.
<i>cmd</i>	Command to send.

Return values

<i>kStatus_Success</i>	Command sent successfully.
<i>kStatus_Busy</i>	LCDIC driver is busy with another transfer.

### 30.2.6.27 status\_t LCDIC\_SendRepeatDataNonBlocking ( *LCDIC\_Type* \* *base*, *Icdic\_handle\_t* \* *handle*, *const Icdic\_repeat\_tx\_xfer\_t* \* *xfer* )

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the lcdic_handle_t structure to store the transfer state.
<i>xfer</i>	LCDIC transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_Busy</i>	LCDIC driver is busy with another transfer.

### 30.2.6.28 status\_t LCDIC\_SendDataArrayNonBlocking ( LCDIC\_Type \* *base*, lcdic\_handle\_t \* *handle*, const lcdic\_tx\_xfer\_t \* *xfer* )

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the lcdic_handle_t structure to store the transfer state.
<i>xfer</i>	LCDIC transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_Busy</i>	LCDIC driver is busy with another transfer.

### 30.2.6.29 status\_t LCDIC\_ReadDataArrayNonBlocking ( LCDIC\_Type \* *base*, lcdic\_handle\_t \* *handle*, const lcdic\_rx\_xfer\_t \* *xfer* )

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the lcdic_handle_t structure to store the transfer state.
<i>xfer</i>	LCDIC transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_Busy</i>	LCDIC driver is busy with another transfer.

### 30.2.6.30 void LCDIC\_TransferHandleIRQ ( LCDIC\_Type \* *base*, void \* *handle* )

IRQ handler to work with [LCDIC\\_TransferNonBlocking](#).

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the lcdic_handle_t structure to store the transfer state.

### 30.2.6.31 void LCDIC\_TransferInstallIRQHandler ( uint32\_t *instance*, void \* *handle*, lcdic\_transfer\_irq\_handler\_t *handler* )

Install IRQ handler for specific instance.

Parameters

<i>instance</i>	LCDIC instance.
<i>handle</i>	Driver handle, it will be used as IRQ handler parameter.
<i>handler</i>	IRQ handler to instance.

### 30.2.6.32 uint32\_t LCDIC.GetInstance ( LCDIC\_Type \* *base* )

Parameters

<i>base</i>	LCDIC peripheral base address
-------------	-------------------------------

Returns

The LCDIC module instance

### 30.2.6.33 IRQn\_Type LCDIC\_GetIRQn ( uint32\_t *instance* )

Parameters

<i>instance</i>	LCDIC instance.
-----------------	-----------------

Returns

The LCDIC IRQn.

### 30.2.6.34 `uint32_t LCDIC_FillByteToWord ( const uint8_t * bytes, uint8_t len )`

LCDIC data registers only accept 4-byte data, but the user passed data might be not 4-byte size aligned. This function is used to construct the unaligned part to a word, to write to LCDIC register.

Parameters

<i>bytes</i>	The byte array.
<i>len</i>	Length of the byte array.

Returns

The construct word.

### 30.2.6.35 `void LCDIC_ExtractByteFromWord ( uint32_t word, uint8_t * bytes, uint8_t len )`

LCDIC data registers only accept 4-byte data, but the user passed data might be not 4-byte size aligned. This function is used to get desired bytes from the word read from LCDIC register, and save to the user data array.

Parameters

<i>word</i>	Word data read from LCDIC register.
<i>bytes</i>	The byte array.
<i>len</i>	Length of the byte array.

### 30.2.6.36 `status_t LCDIC_PreparesSendCommand ( LCDIC_Type * base, uint8_t cmd )`

Fill the TRX command and command to TX FIFO, after calling this function, user should wait for transfer done by checking status or IRQ.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>cmd</i>	Command to send.

Return values

<i>kStatus_Success</i>	Operation successed.
------------------------	----------------------

### 30.2.6.37 status\_t LCDIC\_PreparesendRepeatData ( LCDIC\_Type \* *base*, const lcdic\_repeat\_tx\_xfer\_t \* *xfer* )

Fill the required data to TX FIFO, after calling this function, user should wait for transfer done by checking status or IRQ.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>xfer</i>	Transfer structure.

Return values

<i>kStatus_Success</i>	Operation successed.
<i>kStatus_InvalidArgument</i>	Invalid argument.

### 30.2.6.38 status\_t LCDIC\_PreparesenddataArray ( LCDIC\_Type \* *base*, const lcdic\_tx\_xfer\_t \* *xfer*, uint32\_t \* *xferSizeWordAligned*, uint8\_t \* *xferSizeWordUnaligned*, uint32\_t \* *wordUnalignedData* )

Fill the required command data to TX FIFO, after calling this function, user should fill the *xfer->txData* to TX FIFO based on FIFO status.

## Parameters

<i>base</i>	LCDIC peripheral base address.
<i>xfer</i>	Transfer structure.
<i>xferSizeWord-Aligned</i>	The word size aligned part of the transfer data.
<i>xferSizeWord-Unaligned</i>	The word size unaligned part of the transfer data.
<i>word-UnalignedData</i>	Word to save the word size unaligned data, it should be sent after all word size aligned data write finished.

## Return values

<i>kStatus_Success</i>	Operation successed.
<i>kStatus_InvalidArgument</i>	Invalid argument.

**30.2.6.39 status\_t LCDIC\_PrepReadDataArray ( LCDIC\_Type \* *base*, const lcdic\_rx\_xfer\_t \* *xfer*, uint32\_t \* *xferSizeWordAligned*, uint8\_t \* *xferSizeWordUnaligned* )**

Fill the required command data to TX FIFO, after calling this function, user should read RX FIFO to *xfer->rxData* based on FIFO status.

## Parameters

<i>base</i>	LCDIC peripheral base address.
<i>xfer</i>	Transfer structure.
<i>xferSizeWord-Aligned</i>	The word size aligned part of the transfer data.
<i>xferSizeWord-Unaligned</i>	The word size unaligned part of the transfer data.

## Return values

<i>kStatus_Success</i>	Operation successed.
<i>kStatus_InvalidArgument</i>	Invalid argument.

## 30.3 LCDIC DMA Driver

### 30.3.1 Overview

#### Data Structures

- struct `lcdic_dma_handle_t`

*LCDIC DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* `lcdic_dma_callback_t`)(LCDIC\_Type \*base, lcdic\_dma\_handle\_t \*handle, `status_t` status, void \*userData)

*LCDIC DMA callback called at the end of transfer.*

#### Driver version

- #define `FSL_LCDIC_DMA_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 0)`)

#### DMA Transactional

- `status_t LCDIC_TransferCreateHandleDMA` (LCDIC\_Type \*base, lcdic\_dma\_handle\_t \*handle, `lcdic_dma_callback_t` callback, void \*userData, `dma_handle_t` \*txDmaHandle, `dma_handle_t` \*rxDmaHandle, `dma_descriptor_t` dmaDesc[2])  
*Initialize the LCDIC DMA handle.*
- `status_t LCDIC_TransferDMA` (LCDIC\_Type \*base, lcdic\_dma\_handle\_t \*handle, const `lcdic_xfer_t` \*xfer)  
*Perform a non-blocking LCDIC transfer using DMA.*
- `status_t LCDIC_SendDataArrayDMA` (LCDIC\_Type \*base, lcdic\_dma\_handle\_t \*handle, const `lcdic_tx_xfer_t` \*xfer)  
*Send data array using DMA.*
- `status_t LCDIC_ReadDataArrayDMA` (LCDIC\_Type \*base, lcdic\_dma\_handle\_t \*handle, const `lcdic_rx_xfer_t` \*xfer)  
*Read data array using DMA.*
- void `LCDIC_TransferHandleIRQDMA` (LCDIC\_Type \*base, void \*handle)  
*LCDIC IRQ handler function work with DMA transactional APIs.*

### 30.3.2 Data Structure Documentation

#### 30.3.2.1 struct \_lcdic\_dma\_handle

##### Data Fields

- volatile bool `xferInProgress`

- *Transfer in progress.*
- **lcdic\_xfer\_mode\_t xferMode**  
*On-going transfer mode.*
- **dma\_handle\_t \* txDmaHandle**  
*DMA handler for send.*
- **dma\_handle\_t \* rxDmaHandle**  
*DMA handler for receive.*
- **lcdic\_dma\_callback\_t callback**  
*Callback when transfer finished.*
- **void \* userData**  
*User Data for callback.*
- **uint32\_t xferSizeWordAligned**  
*4-byte size aligned part or the transfer data size.*
- **uint8\_t xferSizeWordUnaligned**  
*4-byte size unaligned part of the transfer data size.*
- **uint8\_t rxSizeWordUnaligned**  
*Same as xferSizeWordUnaligned, it is only used for RX.*
- **uint32\_t tmpData**  
*To save temporary data during transfer.*
- **dma\_descriptor\_t \* dmaDesc**  
*Pointer to two DMA descriptor, should be 16-byte aligned.*
- **const uint8\_t \* txData**  
*Pointer to the TX data.*
- **uint8\_t \* rxData**  
*Pointer to the RX data.*

## Field Documentation

- (1) `lcdic_xfer_mode_t lcdic_dma_handle_t::xferMode`
- (2) `lcdic_dma_callback_t lcdic_dma_handle_t::callback`
- (3) `const uint8_t* lcdic_dma_handle_t::txData`
- (4) `uint8_t* lcdic_dma_handle_t::rxData`
- (5) `uint32_t lcdic_dma_handle_t::xferSizeWordAligned`
- (6) `uint8_t lcdic_dma_handle_t::xferSizeWordUnaligned`
- (7) `uint8_t lcdic_dma_handle_t::rxSizeWordUnaligned`
- (8) `uint32_t lcdic_dma_handle_t::tmpData`
- (9) `dma_descriptor_t* lcdic_dma_handle_t::dmaDesc`

### 30.3.3 Typedef Documentation

**30.3.3.1 `typedef void(* lcdic_dma_callback_t)(LCDIC_Type *base, lcdic_dma_handle_t *handle, status_t status, void *userData)`**

### 30.3.4 Function Documentation

**30.3.4.1 `status_t LCDIC_TransferCreateHandleDMA ( LCDIC_Type * base,  
lcdic_dma_handle_t * handle, lcdic_dma_callback_t callback, void * userData,  
dma_handle_t * txDmaHandle, dma_handle_t * rxDmaHandle, dma_descriptor_t  
dmaDesc[2] )`**

This function initializes the LCDIC DMA handle which can be used for other LCDIC transactional APIs. Usually, for a specified LCDIC instance, user need only call this API once to get the initialized handle.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	LCDIC handle pointer.
<i>callback</i>	User callback function called at the end of a transfer.
<i>userData</i>	User data for callback.
<i>txDmaHandle</i>	DMA handle pointer for LCDIC Tx, the handle shall be static allocated by users.
<i>rxDmaHandle</i>	DMA handle pointer for LCDIC Rx, the handle shall be static allocated by users.
<i>dmaDesc</i>	User allocated dma descriptor, it should be in non-cacheable region and 16-byte aligned.

### 30.3.4.2 status\_t LCDIC\_TransferDMA ( LCDIC\_Type \* *base*, lcdic\_dma\_handle\_t \* *handle*, const lcdic\_xfer\_t \* *xfer* )

This function returned immediately after transfer initiates, monitor the transfer done by callback.

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	LCDIC 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_Busy</i>	LCDIC is not idle, is running another transfer.

### 30.3.4.3 status\_t LCDIC\_SendDataArrayDMA ( LCDIC\_Type \* *base*, lcdic\_dma\_handle\_t \* *handle*, const lcdic\_tx\_xfer\_t \* *xfer* )

Parameters

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

<i>handle</i>	Pointer to the lcdic_handle_t structure to store the transfer state.
<i>xfer</i>	LCDIC transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_Busy</i>	LCDIC driver is busy with another transfer.

### 30.3.4.4 status\_t LCDIC\_ReadDataArrayDMA ( LCDIC\_Type \* *base*, lcdic\_dma\_handle\_t \* *handle*, const lcdic\_rx\_xfer\_t \* *xfer* )

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the lcdic_handle_t structure to store the transfer state.
<i>xfer</i>	LCDIC transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_Busy</i>	LCDIC driver is busy with another transfer.

### 30.3.4.5 void LCDIC\_TransferHandleIRQDMA ( LCDIC\_Type \* *base*, void \* *handle* )

IRQ handler to work with [LCDIC\\_TransferDMA](#).

Parameters

<i>base</i>	LCDIC peripheral base address.
<i>handle</i>	Pointer to the lcdic_dma_handle_t structure to store the transfer state.

# Chapter 31

## MRT: Multi-Rate Timer

### 31.1 Overview

The MCUXpresso SDK provides a driver for the Multi-Rate Timer (MRT) of MCUXpresso SDK devices.

### 31.2 Function groups

The MRT driver supports operating the module as a time counter.

#### 31.2.1 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.

#### 31.2.2 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

#### 31.2.3 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.

### 31.2.4 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.

### 31.2.5 Status

Provides functions to get and clear the PIT status.

### 31.2.6 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 31.3 Typical use case

### 31.3.1 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.*

## 31.4 Data Structure Documentation

### 31.4.1 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*

## 31.5 Enumeration Type Documentation

### 31.5.1 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.

### 31.5.2 enum mrt\_timer\_mode\_t

Enumerator

- [kMRT\\_RepeatMode](#)** Repeat Interrupt mode.
- [kMRT\\_OneShotMode](#)** One-shot Interrupt mode.
- [kMRT\\_OneShotStallMode](#)** One-shot stall mode.

### 31.5.3 enum mrt\_interrupt\_enable\_t

Enumerator

*kMRT\_TimerInterruptEnable* Timer interrupt enable.

### 31.5.4 enum mrt\_status\_flags\_t

Enumerator

*kMRT\_TimerInterruptFlag* Timer interrupt flag.

*kMRT\_TimerRunFlag* Indicates state of the timer.

## 31.6 Function Documentation

### 31.6.1 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.

### 31.6.2 void MRT\_Deinit ( MRT\_Type \* *base* )

Parameters

<i>base</i>	Multi-Rate timer peripheral base address
-------------	--

### 31.6.3 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.
---------------	---

### 31.6.4 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.

### 31.6.5 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>

### 31.6.6 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>

### 31.6.7 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](#)

### 31.6.8 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](#)

### 31.6.9 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>

### 31.6.10 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

### 31.6.11 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

### 31.6.12 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.

### 31.6.13 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.

### 31.6.14 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
-------------	--

### 31.6.15 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 32

## OSTIMER: OS Event Timer Driver

### 32.1 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.

### 32.2 Function groups

The OSTIMER driver supports operating the module as a time counter.

#### 32.2.1 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.

#### 32.2.2 OSTIMER status

The function [OSTIMER\\_GetStatusFlags\(\)](#) will get the current status flag of OSTIMER. The function [OSTIMER\\_ClearStatusFlag\(\)](#) will help clear the status flags.

#### 32.2.3 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.

#### 32.2.4 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\(\)](#).

### 32.3 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, 2)`)  
*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.*

## 32.4 Macro Definition Documentation

### 32.4.1 #define FSL\_OSTIMER\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 2))

## 32.5 Typedef Documentation

### 32.5.1 **typedef void(\* ostimer\_callback\_t)(void)**

## 32.6 Enumeration Type Documentation

### 32.6.1 **enum \_ostimer\_flags**

Enumerator

*kOSTIMER\_MatchInterruptFlag* Match interrupt flag bit, sets if the match value was reached.

## 32.7 Function Documentation

### 32.7.1 **void OSTIMER\_Init ( OSTIMER\_Type \* base )**

### 32.7.2 **void OSTIMER\_Deinit ( OSTIMER\_Type \* base )**

This function shuts down OSTIMER bus clock

Parameters

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

### 32.7.3 **uint64\_t OSTIMER\_GrayToDecimal ( uint64\_t gray )**

Parameters

<i>gray</i>	The gray value input.
-------------	-----------------------

Returns

The decimal value.

### 32.7.4 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.

### 32.7.5 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

### 32.7.6 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

### 32.7.7 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.

### 32.7.8 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.

### 32.7.9 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>value</i>	OSTIMER timer match value (Value is gray-code format).

### 32.7.10 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.
-------------	----------------------------------

### 32.7.11 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.
-------------	----------------------------------

### 32.7.12 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.

### 32.7.13 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.

### 32.7.14 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.

### **32.7.15 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.

### **32.7.16 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 33

## PINT: Pin Interrupt and Pattern Match Driver

### 33.1 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.

### 33.2 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.

#### 33.2.1 Pin Interrupt use case

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

#### 33.2.2 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, 13))

## 33.3 Typedef Documentation

### 33.3.1 **typedef void(\* pint\_cb\_t)(pint\_pin\_int\_t pintr, uint32\_t pmatch\_status)**

## 33.4 Enumeration Type Documentation

### 33.4.1 **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.

### 33.4.2 **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.

### 33.4.3 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.

### 33.4.4 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.

### 33.4.5 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.

## 33.5 Function Documentation

### 33.5.1 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>
--------------

### 33.5.2 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>
--------------

### 33.5.3 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>
--------------

### 33.5.4 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>
--------------

### 33.5.5 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.
---------------	--

### 33.5.6 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>
--------------

### 33.5.7 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.
---------------	---

### 33.5.8 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>
--------------

### 33.5.9 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.
-------------	---

### **33.5.10 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>	
--------------	--

### **33.5.11 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.
--------------	--

**33.5.12 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>
--------------

### 33.5.13 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.
-------------	---

### 33.5.14 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>
--------------

### 33.5.15 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.
--------------	---

### 33.5.16 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>
--------------

### 33.5.17 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>
--------------

### 33.5.18 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.
---------------	---

### 33.5.19 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.
---------------	---

### 33.5.20 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.
-----------------	--

### 33.5.21 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>
--------------

### 33.5.22 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>
--------------

### 33.5.23 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>
--------------

### **33.5.24 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>
--------------

### **33.5.25 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>
--------------

### **33.5.26 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>
--------------

### 33.5.27 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>
--------------

### 33.5.28 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>
--------------

### 33.5.29 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 34

## POWERQUAD: PowerQuad hardware accelerator

### 34.1 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.

## 34.2 Function groups

### 34.2.1 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, 2, 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)

- Processing function for the 16-bit integer vectorised natural exponent with negative parameter.  
• void [PQ\\_VectorBiquadDf2F32](#) (float \*pSrc, float \*pDst, int32\_t length)
  - Processing function for the floating-point vectorised biquad direct form II.
- void [PQ\\_VectorBiquadDf2Fixed32](#) (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)
  - Processing function for the 32-bit integer vectorised biquad direct form II.
- void [PQ\\_VectorBiquadDf2Fixed16](#) (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)
  - Processing function for the 16-bit integer vectorised biquad direct form II.
- void [PQ\\_VectorBiquadCascadeDf2F32](#) (float \*pSrc, float \*pDst, int32\_t length)
  - Processing function for the floating-point vectorised biquad direct form II.
- void [PQ\\_VectorBiquadCascadeDf2Fixed32](#) (int32\_t \*pSrc, int32\_t \*pDst, int32\_t length)
  - Processing function for the 32-bit integer vectorised biquad direct form II.
- void [PQ\\_VectorBiquadCascadeDf2Fixed16](#) (int16\_t \*pSrc, int16\_t \*pDst, int32\_t length)
  - Processing function for the 16-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 fixed inverse trigonometric.
- 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.
- int32\_t [PQ\\_Arctan2Fixed](#) (POWERQUAD\_Type \*base, int32\_t x, int32\_t y, [pq\\_cordic\\_iter\\_t](#) iteration)
  - Processing function for the fixed inverse trigonometric.
- static int32\_t [PQ\\_Biquad1Fixed](#) (int32\_t val)
  - Processing function for the fixed biquad.
- void [PQ\\_TransformCFFT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
  - Processing function for the complex FFT.
- void [PQ\\_TransformRFFT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
  - Processing function for the real FFT.
- void [PQ\\_TransformIFFT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
  - Processing function for the inverse complex FFT.
- void [PQ\\_TransformCDCT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
  - Processing function for the inverse complex DCT.
- void [PQ\\_TransformRDCT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
  - Processing function for the real DCT.
- void [PQ\\_TransformIDCT](#) (POWERQUAD\_Type \*base, uint32\_t length, void \*pData, void \*pResult)
  - Processing function for the inverse complex DCT.
- void [PQ\\_BiquadBackUpInternalState](#) (POWERQUAD\_Type \*base, int32\_t biquad\_num, [pq\\_biquad\\_state\\_t](#) \*state)
  - Processing function for backup biquad context.
- void [PQ\\_BiquadRestoreInternalState](#) (POWERQUAD\_Type \*base, int32\_t biquad\_num, [pq\\_biquad\\_state\\_t](#) \*state)
  - Processing function for restore 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.

- Initialization function for the direct form II Biquad cascade filter.
- void **PQ\_BiquadCascadeDf2F32** (const **pq\_biquad\_cascade\_df2\_instance** \*S, float \*pSrc, float \*pDst, uint32\_t blockSize)
  - 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.

### 34.3 Data Structure Documentation

#### 34.3.1 struct pq\_prescale\_t

##### Data Fields

- int8\_t **inputPrescale**

- `int8_t outputPrescale`  
*Input prescale.*
- `int8_t outputSaturate`  
*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`

### 34.3.2 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  $1.5 \times 2^n$  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`

### 34.3.3 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

**34.3.4 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

**34.3.5 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.

**34.3.6 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

**34.4 Macro Definition Documentation****34.4.1 #define FSL\_POWERQUAD\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))****34.4.2 #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.

**34.4.3 #define PQ\_End\_Vector\_Func( ) \_\_asm volatile("POP {r2-r7}")**

This function should be called after vector calculation.

**34.4.4 #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.

#### 34.4.5 #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        \
     %[psrc] "r"(PSRC),       \
     %[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.

#### 34.4.6 #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.

**34.4.7 #define PQ\_EndVector( ) \_\_asm volatile("POP {r3-r10} \n")**

This function should be called after vector calculation.

**34.4.8 #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\_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
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();
```

**34.4.9 #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();
```

### 34.4.10 #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();
```

### 34.4.11 #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();
```

### 34.4.12 #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();

```

### 34.4.13 #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();

```

### 34.4.14 #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();

```

### 34.4.15 #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();

```

### 34.4.16 #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();

```

**34.4.17 #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();
```

**34.4.18 #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 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_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FX(false, false);
PQ_DF2_Cascade_Vector8_FX(true, true);
PQ_End_Vector_Func();

```

### 34.4.19 #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();

```

**34.4.20 #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 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_Vector8BiquadDf2CascadeFixed32();
PQ_EndVector();
```

**34.4.21 #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();

```

**34.4.22 #define POWERQUAD\_MAKE\_MATRIX\_LEN( mat1Row, mat1Col,  
mat2Col ) (((uint32\_t)(mat1Row) << 0U) | ((uint32\_t)(mat1Col) << 8U) |  
((uint32\_t)(mat2Col) << 16U))**

**34.4.23 #define PQ\_Q31\_2\_FLOAT( x ) (((float)(x)) / 2147483648.0f)**

**34.4.24 #define PQ\_Q15\_2\_FLOAT( x ) (((float)(x)) / 32768.0f)**

## 34.5 Enumeration Type Documentation

### 34.5.1 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.

### 34.5.2 enum pq\_format\_t

Enumerator

- kPQ\_16Bit* Int16 Fixed point.
- kPQ\_32Bit* Int32 Fixed point.
- kPQ\_Float* Float point.

### 34.5.3 enum pq\_cordic\_iter\_t

Enumerator

- kPQ\_Iteration\_8* Iterate 8 times.
- kPQ\_Iteration\_16* Iterate 16 times.
- kPQ\_Iteration\_24* Iterate 24 times.

## 34.6 Function Documentation

### 34.6.1 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.
---------------	-------------------------------------

### 34.6.2 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.

### 34.6.3 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.

#### 34.6.4 void PQ\_Init ( POWERQUAD\_Type \* *base* )

Parameters

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

#### 34.6.5 void PQ\_Deinit ( POWERQUAD\_Type \* *base* )

Parameters

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

#### 34.6.6 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

#### 34.6.7 static void PQ\_WaitDone ( POWERQUAD\_Type \* *base* ) [inline], [static]

Parameters

---

<i>base</i>	POWERQUAD peripheral base address
-------------	-----------------------------------

**34.6.8 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

**34.6.9 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

**34.6.10 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

**34.6.11 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
--------------	------------------------------------

### 34.6.12 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

### 34.6.13 static void PQ\_EtonxF32( 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

### 34.6.14 static void PQ\_SinF32( float \* *pSrc*, float \* *pDst* ) [inline], [static]

Parameters

<i>*pSrc</i>	points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
<i>*pDst</i>	points to the block of output data

### 34.6.15 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

#### 34.6.16 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

#### 34.6.17 static void PQ\_DivF32 ( `float * x1, float * x2, float * pDst` ) [inline], [static]

Get  $x1 / x2$ .

Parameters

<code>x1</code>	$x1$
<code>x2</code>	$x2$
<code>*pDst</code>	points to the block of output data

#### 34.6.18 static void PQ\_Biquad1F32 ( `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

#### 34.6.19 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  $\ln(\text{val})$ .

#### 34.6.20 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  $\text{inv}(\text{val})$ .

#### 34.6.21 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  $\sqrt{\text{val}}$ .

#### 34.6.22 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  $1/\sqrt{\text{val}}$ .

#### 34.6.23 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).

#### 34.6.24 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).

#### 34.6.25 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).

#### 34.6.26 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).

#### 34.6.27 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).

#### 34.6.28 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 sin(val).

#### 34.6.29 static int32\_t PQ\_BiquadFixed ( int32\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated
------------	------------------------

Returns

returns biquad(val).

#### 34.6.30 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.

**34.6.31 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.

**34.6.32 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.

**34.6.33 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.

**34.6.34 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.

**34.6.35 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.

**34.6.36 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.

**34.6.37 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.

**34.6.38 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.

34.6.39 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.

#### 34.6.40 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.

#### 34.6.41 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.

#### 34.6.42 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.
---------------	--------------------------

#### 34.6.43 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.

#### 34.6.44 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.

#### 34.6.45 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.

#### 34.6.46 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.

#### 34.6.47 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.

#### 34.6.48 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.

#### 34.6.49 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.

#### 34.6.50 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.

#### 34.6.51 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.

#### 34.6.52 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.

#### 34.6.53 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.

#### 34.6.54 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.

#### 34.6.55 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

#### 34.6.56 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

#### 34.6.57 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

34.6.58 **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

### 34.6.59 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

### 34.6.60 int32\_t PQ\_ArctanFixed ( POWERQUAD\_Type \* *base*, int32\_t *x*, int32\_t *y*, pq\_cordic\_iter\_t *iteration* )

Get the inverse tangent, the behavior is like c function atan.

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 -2^26 to 2^26, which means -pi/2 to pi/2.

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).

34.6.61 **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 radians,  $2^{27}$  means pi. The range is -1.118 to 1.118 radians.

## 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).

### 34.6.62 int32\_t PQ\_Arctan2Fixed ( POWERQUAD\_Type \* *base*, int32\_t *x*, int32\_t *y*, pq\_cordic\_iter\_t *iteration* )

Get the inverse tangent, it calculates the angle in radians for the quadrant. The behavior is like c function atan2.

## 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\_Arctan2Fixed(POWERQUAD, 100000, 200000, kPQ\_Iteration\_24) is more accurate than PQ\_Arctan2Fixed(POWERQUAD, 1, 2, kPQ\_Iteration\_24).

34.6.63 static int32\_t PQ\_Biquad1Fixed( int32\_t *val* ) [inline], [static]

Parameters

<i>val</i>	value to be calculated
------------	------------------------

Returns

returns biquad(val).

#### 34.6.64 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.

#### 34.6.65 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.

#### 34.6.66 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.

**34.6.67 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.

**34.6.68 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.

**34.6.69 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.

**34.6.70 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.

**34.6.71 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.

**34.6.72 void PQ\_BiquadCascadeDf2Init ( pq\_biquad\_cascade\_df2\_instance \* *S*, uint8\_t *numStages*, pq\_biquad\_state\_t \* *pState* )**

Parameters

<i>in,out</i>	* <i>S</i>	points to an instance of the filter data structure.
<i>in</i>	<i>numStages</i>	number of 2nd order stages in the filter.

in	<i>*pState</i>	points to the state buffer.
----	----------------	-----------------------------

**34.6.73 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.

**34.6.74 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.

**34.6.75 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.

**34.6.76 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.

**34.6.77 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

**34.6.78 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.

**34.6.79 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.

**34.6.80 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.

**34.6.81 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.

**34.6.82 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.

**34.6.83 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.

**34.6.84 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.

**34.6.85 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 35

## RTC: Real Time Clock

### 35.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC).

### 35.2 Function groups

The RTC driver supports operating the module as a time counter.

#### 35.2.1 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.

#### 35.2.2 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.

#### 35.2.3 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.

#### 35.2.4 Start & Stop timer

The function [RTC\\_StartTimer\(\)](#) starts the RTC time counter.

The function [RTC\\_StopTimer\(\)](#) stops the RTC time counter.

### 35.2.5 Status

Provides functions to get and clear the RTC status.

### 35.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

### 35.2.7 High resolution timer

Provides functions to enable high resolution timer and set and get the wake time.

## 35.3 Typical use case

### 35.3.1 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, 2, 0))  
*Version 2.2.0.*

## 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).*

## SUBSEC counter

- static void **RTC\_EnableSubsecCounter** (RTC\_Type \*base, bool enable)  
*Enable the RTC Sub-second counter (32KHZ).*
- static uint32\_t **RTC\_GetSubsecValue** (const RTC\_Type \*base)  
*A read of 32KHZ sub-seconds counter.*

## 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.*

## 35.4 Data Structure Documentation

### 35.4.1 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`

**35.5 Enumeration Type Documentation****35.5.1 enum rtc\_interrupt\_enable\_t**

Enumerator

`kRTC_AlarmInterruptEnable` Alarm interrupt.  
`kRTC_WakeupInterruptEnable` Wake-up interrupt.

**35.5.2 enum rtc\_status\_flags\_t**

Enumerator

`kRTC_AlarmFlag` Alarm flag.  
`kRTC_WakeupFlag` 1kHz wake-up timer flag

**35.6 Function Documentation****35.6.1 void RTC\_Init ( RTC\_Type \* *base* )**

Note

This API should be called at the beginning of the application using the RTC driver.

Parameters

<code>base</code>	RTC peripheral base address
-------------------	-----------------------------

**35.6.2 static void RTC\_Deinit ( RTC\_Type \* *base* ) [inline], [static]**

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

### 35.6.3 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

### 35.6.4 void RTC\_GetDatetime ( 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.

### 35.6.5 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

### 35.6.6 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.

### 35.6.7 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>

### 35.6.8 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).

**35.6.9 static void RTC\_EnableSubsecCounter ( RTC\_Type \* *base*, bool *enable* )  
[inline], [static]**

Note

Only enable sub-second counter after RTC\_ENA bit has been set to 1.

Parameters

<i>base</i>	RTC peripheral base address
<i>enable</i>	Enable/Disable RTC sub-second counter. <ul style="list-style-type: none"><li>• true: Enable RTC sub-second counter.</li><li>• false: Disable RTC sub-second counter.</li></ul>

**35.6.10 static uint32\_t RTC\_GetSubsecValue ( const RTC\_Type \* *base* )  
[inline], [static]**

Parameters

<i>base</i>	RTC peripheral base address
-------------	-----------------------------

Returns

Current value of the SUBSEC register

**35.6.11 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

**35.6.12 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.

### 35.6.13 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

### 35.6.14 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.

### 35.6.15 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).
--------------------	--

### 35.6.16 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).

### 35.6.17 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>

### 35.6.18 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>

### 35.6.19 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>

### 35.6.20 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>

### 35.6.21 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](#)

### 35.6.22 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](#)

### 35.6.23 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>

### 35.6.24 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>

### 35.6.25 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
-------------	-----------------------------

### 35.6.26 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
-------------	-----------------------------

### 35.6.27 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 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, 5, 1)`)
- 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 void **SCTIMER\_GetEventInState** (SCT\_Type \*base, uint32\_t event, uint32\_t state)
 

*Clear the state mask bit field of EV\_STATE register.*
- static bool **SCTIMER\_GetCaptureValue** (SCT\_Type \*base, **sctimer\_counter\_t** whichCounter, uint8\_t capChannel)
 

*Get the value of capture register.*
- void **SCTIMER\_EventHandleIRQ** (SCT\_Type \*base)
 

*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**

- **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.

<b><i>kSCTIMER_Event1Flag</i></b>	Event 1 Flag.
<b><i>kSCTIMER_Event2Flag</i></b>	Event 2 Flag.
<b><i>kSCTIMER_Event3Flag</i></b>	Event 3 Flag.
<b><i>kSCTIMER_Event4Flag</i></b>	Event 4 Flag.
<b><i>kSCTIMER_Event5Flag</i></b>	Event 5 Flag.
<b><i>kSCTIMER_Event6Flag</i></b>	Event 6 Flag.
<b><i>kSCTIMER_Event7Flag</i></b>	Event 7 Flag.
<b><i>kSCTIMER_Event8Flag</i></b>	Event 8 Flag.
<b><i>kSCTIMER_Event9Flag</i></b>	Event 9 Flag.
<b><i>kSCTIMER_Event10Flag</i></b>	Event 10 Flag.
<b><i>kSCTIMER_Event11Flag</i></b>	Event 11 Flag.
<b><i>kSCTIMER_Event12Flag</i></b>	Event 12 Flag.
<b><i>kSCTIMER_BusErrorLFlag</i></b>	Bus error due to write when L counter was not halted.
<b><i>kSCTIMER_BusErrorHFlag</i></b>	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>dutyCyclePercent</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 <code>sctimer_interrupt_enable_t</code>
<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 static uint32\_t SCTIMER\_GetCaptureValue ( **SCT\_Type** \* *base*, **sctimer\_counter\_t** *whichCounter*, **uint8\_t** *capChannel* ) [inline], [static]

This function returns the captured value upon occurrence of the events selected by the corresponding Capture Control registers occurred.

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>capChannel</i>	SCTimer capture register of capture channel.

Returns

The SCTimer counter value at which this register was last captured.

### 36.9.39 void SCTIMER\_EventHandleIRQ ( **SCT\_Type** \* *base* )

Parameters

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

# Chapter 37

## SPI: Serial Peripheral Interface Driver

### 37.1 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.

### 37.2 Typical use case

#### 37.2.1 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)
{
}

// ...
}

```

### 37.2.2 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 Driver

## 37.3 SPI Driver

### 37.3.1 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](#) (0x00U)  
*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 }

```

*SPI interrupt sources.*

- 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` []

*SPI default SSEL COUNT.*

## Driver version

- #define `FSL_SPI_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 2)`)  
*SPI driver version.*

### 37.3.2 Data Structure Documentation

#### 37.3.2.1 `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`

#### 37.3.2.2 `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**

### 37.3.2.3 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*

### 37.3.2.4 struct spi\_transfer\_t

#### Data Fields

- **const 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`

#### 37.3.2.5 struct spi\_half\_duplex\_transfer\_t

### Data Fields

- `const 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`

#### 37.3.2.6 struct spi\_config\_t

#### 37.3.2.7 struct \_spi\_master\_handle

Master handle type.

### Data Fields

- `const 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`

- **Number of data to be received [in bytes].**  
• volatile int8\_t **toReceiveCount**  
    *The number of data expected to receive in data width.*
- size\_t **totalByteCount**  
    *A number of transfer bytes.*
- volatile uint32\_t **state**  
    *SPI internal state.*
- spi\_master\_callback\_t **callback**  
    *SPI callback.*
- void \* **userData**  
    *Callback parameter.*
- uint8\_t **dataWidth**  
    *Width of the data [Valid values: 1 to 16].*
- uint8\_t **sselNum**  
    *Slave select number to be asserted when transferring data [Valid values: 0 to 3].*
- uint32\_t **configFlags**  
    *Additional option to control transfer.*
- uint8\_t **txWatermark**  
    *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.

### 37.3.3 Macro Definition Documentation

37.3.3.1 `#define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 3, 2))`

37.3.3.2 `#define SPI_DUMMYDATA (0x00U)`

37.3.3.3 `#define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

### 37.3.4 Typedef Documentation

37.3.4.1 `typedef void(* flexcomm_spi_master_irq_handler_t)(SPI_Type *base, spi_master_handle_t *handle)`

37.3.4.2 `typedef void(* flexcomm_spi_slave_irq_handler_t)(SPI_Type *base, spi_slave_handle_t *handle)`

### 37.3.5 Enumeration Type Documentation

37.3.5.1 `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.

37.3.5.2 `enum spi_shift_direction_t`

Enumerator

*kSPI\_MsbFirst* Data transfers start with most significant bit.

*kSPI\_LsbFirst* Data transfers start with least significant bit.

37.3.5.3 `enum spi_clock_polarity_t`

Enumerator

*kSPI\_ClockPolarityActiveHigh* Active-high SPI clock (idles low).

*kSPI\_ClockPolarityActiveLow* Active-low SPI clock (idles high).

### 37.3.5.4 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.

### 37.3.5.5 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.

### 37.3.5.6 enum spi\_rxfifo\_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.

### 37.3.5.7 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

<i>kSPI_Data8Bits</i>	8 bits data width
<i>kSPI_Data9Bits</i>	9 bits data width
<i>kSPI_Data10Bits</i>	10 bits data width
<i>kSPI_Data11Bits</i>	11 bits data width
<i>kSPI_Data12Bits</i>	12 bits data width
<i>kSPI_Data13Bits</i>	13 bits data width
<i>kSPI_Data14Bits</i>	14 bits data width
<i>kSPI_Data15Bits</i>	15 bits data width
<i>kSPI_Data16Bits</i>	16 bits data width

### 37.3.5.8 enum spi\_ssel\_t

Enumerator

<i>kSPI_Ssel0</i>	Slave select 0.
<i>kSPI_Ssel1</i>	Slave select 1.
<i>kSPI_Ssel2</i>	Slave select 2.
<i>kSPI_Ssel3</i>	Slave select 3.

### 37.3.5.9 anonymous enum

Enumerator

<i>kStatus_SPI_Busy</i>	SPI bus is busy.
<i>kStatus_SPI_Idle</i>	SPI is idle.
<i>kStatus_SPI_Error</i>	SPI error.
<i>kStatus_SPI_BaudrateNotSupport</i>	Baudrate is not support in current clock source.
<i>kStatus_SPI_Timeout</i>	SPI timeout polling status flags.

### 37.3.5.10 enum \_spi\_interrupt\_enable

Enumerator

<i>kSPI_RxLvlIrq</i>	Rx level interrupt.
<i>kSPI_TxLvlIrq</i>	Tx level interrupt.

### 37.3.5.11 enum \_spi\_statusflags

Enumerator

<i>kSPI_TxEmptyFlag</i>	txFifo is empty
<i>kSPI_TxNotFullFlag</i>	txFifo is not full
<i>kSPI_RxNotEmptyFlag</i>	rxFIFO is not empty
<i>kSPI_RxFullFlag</i>	rxFIFO is full

### 37.3.6 Variable Documentation

#### 37.3.6.1 volatile uint8\_t s\_dummyData[]

Global variable for dummy data value setting.

# Chapter 38

## USART: Universal Synchronous/Asynchronous Receiver-/Transmitter Driver

### 38.1 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.

## 38.2 Typical use case

### 38.2.1 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);
}

```

### 38.2.2 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)
{
}

// ...
}

```

### 38.2.3 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)
    {
        ;
    }
}

// ...
}

```

### 38.2.4 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 Driver

## 38.3 USART Driver

### 38.3.1 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 `uart_parity_mode_t` {
 

```
kUSART_ParityDisabled = 0x0U,
```

```
kUSART_ParityEven = 0x2U,
```

```
kUSART_ParityOdd = 0x3U }
```

*USART parity mode.*
- enum `uart_stop_bit_count_t` {
 

```
kUSART_OneStopBit = 0U,
```

```
kUSART_TwoStopBit = 1U }
```

*USART stop bit count.*
- enum `uart_data_len_t` {
 

```
kUSART_7BitsPerChar = 0U,
```

```
kUSART_8BitsPerChar = 1U }
```

*USART data size.*
- enum `uart_clock_polarity_t` {
 

```
kUSART_RxSampleOnFallingEdge = 0x0U,
```

```
kUSART_RxSampleOnRisingEdge = 0x1U }
```

*USART clock polarity configuration, used in sync mode.*
- enum `uart_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 `uart_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, 4))`  
*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.*

### 38.3.2 Data Structure Documentation

#### 38.3.2.1 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`

#### 38.3.2.2 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**

### 38.3.2.3 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`

### 38.3.3 Macro Definition Documentation

#### 38.3.3.1 #define FSL\_USART\_DRIVER\_VERSION (MAKE\_VERSION(2, 8, 4))

#### 38.3.3.2 #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.

### 38.3.4 Ttypedef Documentation

38.3.4.1 **typedef void(\* usart\_transfer\_callback\_t)(USART\_Type \*base, usart\_handle\_t \*handle, status\_t status, void \*userData)**

38.3.4.2 **typedef void(\* flexcomm\_usart\_irq\_handler\_t)(USART\_Type \*base, usart\_handle\_t \*handle)**

### 38.3.5 Enumeration Type Documentation

#### 38.3.5.1 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.

#### 38.3.5.2 enum usart\_sync\_mode\_t

Enumerator

**kUSART\_SyncModeDisabled** Asynchronous mode.

**kUSART\_SyncModeSlave** Synchronous slave mode.

**kUSART\_SyncModeMaster** Synchronous master mode.

#### 38.3.5.3 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.

### 38.3.5.4 enum usart\_stop\_bit\_count\_t

Enumerator

*kUSART\_OneStopBit* One stop bit.

*kUSART\_TwoStopBit* Two stop bits.

### 38.3.5.5 enum usart\_data\_len\_t

Enumerator

*kUSART\_7BitsPerChar* Seven bit mode.

*kUSART\_8BitsPerChar* Eight bit mode.

### 38.3.5.6 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.

### 38.3.5.7 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.

### 38.3.5.8 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.

### 38.3.5.9 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.

### 38.3.5.10 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.

### 38.3.6 Function Documentation

**38.3.6.1 uint32\_t USART\_GetInstance ( USART\_Type \* *base* )**

**38.3.6.2 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                  |

**38.3.6.3 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. |
|-------------|--------------------------------|

**38.3.6.4 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. |
|---------------|-------------------------------------|

### 38.3.6.5 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.               |

### 38.3.6.6 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.               |

### 38.3.6.7 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. |

### 38.3.6.8 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.           |

### 38.3.6.9 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. |

### 38.3.6.10 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](#).

### 38.3.6.11 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.    |

**38.3.6.12 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> . |

**38.3.6.13 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> . |

**38.3.6.14 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. |
|-------------|--------------------------------|

### 38.3.6.15 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. |

### 38.3.6.16 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. |

### 38.3.6.17 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. |

### 38.3.6.18 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.             |

### 38.3.6.19 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.             |

### 38.3.6.20 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.             |

### 38.3.6.21 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.

### 38.3.6.22 static uint8\_t USART\_GetRxFifoCount ( USART\_Type \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

Returns

rx FIFO data count.

### 38.3.6.23 static uint8\_t USART\_GetTxFifoCount ( USART\_Type \* *base* ) [inline], [static]

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

Returns

tx FIFO data count.

### 38.3.6.24 void USART\_SendAddress ( USART\_Type \* *base*, uint8\_t *address* )

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | USART peripheral base address. |
| <i>address</i> | USART slave address.           |

### 38.3.6.25 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.            |

### 38.3.6.26 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.                 |

### 38.3.6.27 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. |

### 38.3.6.28 `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

|                                      |                                                                                    |
|--------------------------------------|------------------------------------------------------------------------------------|
| <code>kStatus_Success</code>         | Successfully start the data transmission.                                          |
| <code>kStatus_USART_TxBusy</code>    | Previous transmission still not finished, data not all written to TX register yet. |
| <code>kStatus_InvalidArgument</code> | Invalid argument.                                                                  |

### 38.3.6.29 `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.                                                                         |

### 38.3.6.30 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.          |

### 38.3.6.31 size\_t USART\_TransferGetRxRingBufferLength ( usart\_handle\_t \* *handle* )

Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | USART handle pointer. |
|---------------|-----------------------|

Returns

Length of received data in RX ring buffer.

### 38.3.6.32 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.          |

**38.3.6.33 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> ; |

### 38.3.6.34 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.                                    |

### 38.3.6.35 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.          |

### 38.3.6.36 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> ; |

### 38.3.6.37 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.          |

# 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->enableLockOscilllator = 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 |
|-------------|------------------------------|

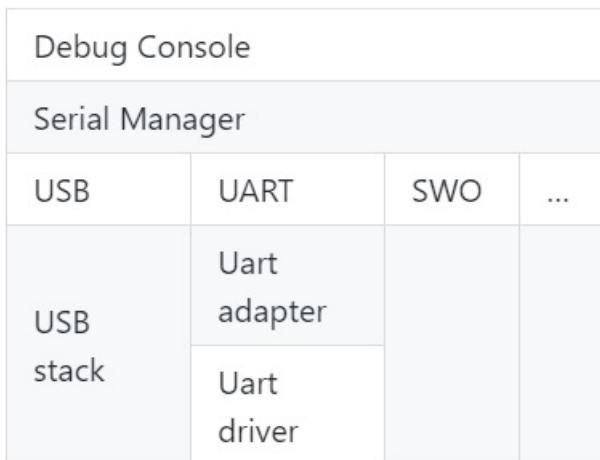
# Chapter 41

## 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                              |

| specifier | Description |
|-----------|-------------|
|-----------|-------------|

- 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_-<br/>REDIRECT_TO_SDK</code>             | defined                            | Low level peripheral* | Low level peripheral |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_SDK</code>             | undefined                          | Low level peripheral* | semihost             |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_TO-<br/>OLCHAIN</code> | defined                            | Low level peripheral* | Low level peripheral |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_TO-<br/>OLCHAIN</code> | undefined                          | semihost              | semihost             |
| <code>DEBUGCONSOLE_-<br/>DISABLE</code>                     | defined                            | No output             | Low level peripheral |
| <code>DEBUGCONSOLE_-<br/>DISABLE</code>                     | undefined                          | No output             | semihost             |

|                         |                              |               |               |
|-------------------------|------------------------------|---------------|---------------|
| <b>SDK_DEBUGCONSOLE</b> | <b>SDK_DEBUGCONSOLE_UART</b> | <b>PRINTF</b> | <b>printf</b> |
|-------------------------|------------------------------|---------------|---------------|

\* 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**

- **debug console configuration**  
*The configuration is used for debug console only.*

**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.
- status\_t **DbgConsole\_TryGetchar** (char \*ch)
  - Debug console try to get char This function provides a API which will not block current task, if character is available return it, otherwise return fail.

## 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.

Returns

|                   |                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <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\_B-LOCKING 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\_B-LOCKING 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.

41.5.13 **status\_t DbgConsole\_TryGetchar ( char \* *ch* )**

### Parameters

|           |                                |
|-----------|--------------------------------|
| <i>ch</i> | the address of char to receive |
|-----------|--------------------------------|

### Returns

Indicates get char was successful or not.

## 41.6 debug console configuration

The configuration is used for debug console only.

### 41.6.1 Overview

Please note, it is not sued for debug console lite.

#### Macros

- `#define DEBUG_CONSOLE_TRANSMIT_BUFFER_LEN (512U)`  
*If Non-blocking mode is needed, please define it at project setting, otherwise blocking mode is the default transfer mode.*
- `#define DEBUG_CONSOLE_RECEIVE_BUFFER_LEN (1024U)`  
*define the receive buffer length which is used to store the user input, buffer is enabled automatically when non-blocking transfer is using, This value will affect the RAM's ultilization, should be set per paltform's capability and software requirement.*
- `#define DEBUG_CONSOLE_TX_RELIABLE_ENABLE (1U)`  
*Whether enable the reliable TX function If the macro is zero, the reliable TX function of the debug console is disabled.*
- `#define DEBUG_CONSOLE_RX_ENABLE (1U)`  
*Whether enable the RX function If the macro is zero, the receive function of the debug console is disabled.*
- `#define DEBUG_CONSOLE_PRINTF_MAX_LOG_LEN (128U)`  
*define the MAX log length debug console support , that is when you call printf("log", x);, the log length can not bigger than this value.*
- `#define DEBUG_CONSOLE_SCANF_MAX_LOG_LEN (20U)`  
*define the buffer support buffer scanf log length, that is when you call scanf("log", &x);, the log length can not bigger than this value.*
- `#define DEBUG_CONSOLE_SYNCHRONIZATION_BM 0`  
*Debug console synchronization User should not change these macro for synchronization mode, but add the corresponding synchronization mechanism per different software environment.*
- `#define DEBUG_CONSOLE_SYNCHRONIZATION_FREERTOS 1`  
*synchronization for freertos software*
- `#define DEBUG_CONSOLE_SYNCHRONIZATION_MODE DEBUG_CONSOLE_SYNCHRONIZATION_BM`  
*RTOS synchronization mechanism disable If not defined, default is enable, to avoid multitask log print mess.*
- `#define DEBUG_CONSOLE_ENABLE_ECHO_FUNCTION 0`  
*echo function support If you want to use the echo function,please define DEBUG\_CONSOLE\_ENABLE\_ECHO at your project setting.*
- `#define BOARD_USE_VIRTUALCOM 0U`  
*Definition to select virtual com(USB CDC) as the debug console.*

## 41.6.2 Macro Definition Documentation

### 41.6.2.1 #define DEBUG\_CONSOLE\_TRANSMIT\_BUFFER\_LEN (512U)

Warning: If you want to use non-blocking transfer, please make sure the corresponding IO interrupt is enable, otherwise there is no output. And non-blocking is combine with buffer, no matter bare-metal or rtos. Below shows how to configure in your project if you want to use non-blocking mode. For IAR, right click project and select "Options", define it in "C/C++ Compiler->Preprocessor->Defined symbols". For KEIL, click "Options for Target...", define it in "C/C++->Preprocessor Symbols->Define". For ARM-GCC, open CmakeLists.txt and add the following lines, "SET(CMAKE\_C\_FLAGS\_DEBUG "\${CMAKE\_C\_FLAGS\_DEBUG} -DDEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING")" for debug target. "SET(CMAKE\_C\_FLAGS\_RELEASE "\${CMAKE\_C\_FLAGS\_RELEASE} -DDEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING")" for release target. For MCUXpresso, right click project and select "Properties", define it in "C/C++ Build->Settings->MCU C Complier->Preprocessor".

define the transmit buffer length which is used to store the multi task log, buffer is enabled automatically when non-blocking transfer is using, This value will affect the RAM's utilization, should be set per platform's capability and software requirement. If it is configured too small, log maybe missed , because the log will not be buffered if the buffer is full, and the print will return immediately with -1. And this value should be multiple of 4 to meet memory alignment.

### 41.6.2.2 #define DEBUG\_CONSOLE\_RECEIVE\_BUFFER\_LEN (1024U)

If it is configured too small, log maybe missed, because buffer will be overwritten if buffer is too small. And this value should be multiple of 4 to meet memory alignment.

### 41.6.2.3 #define DEBUG\_CONSOLE\_TX\_RELIABLE\_ENABLE (1U)

When the macro is zero, the string of PRINTF will be thrown away after the transmit buffer is full.

### 41.6.2.4 #define DEBUG\_CONSOLE\_PRINTF\_MAX\_LOG\_LEN (128U)

This macro decide the local log buffer length, the buffer locate at stack, the stack maybe overflow if the buffer is too big and current task stack size not big enough.

### 41.6.2.5 #define DEBUG\_CONSOLE\_SCANF\_MAX\_LOG\_LEN (20U)

As same as the DEBUG\_CONSOLE\_BUFFER\_PRINTF\_MAX\_LOG\_LEN.

#### 41.6.2.6 #define DEBUG\_CONSOLE\_SYNCHRONIZATION\_BM 0

Such as, if another RTOS is used, add: #define DEBUG\_CONSOLE\_SYNCHRONIZATION\_XXXX 3 in this configuration file and implement the synchronization in fsl.log.c.

synchronization for baremetal software

#### 41.6.2.7 #define DEBUG\_CONSOLE\_SYNCHRONIZATION\_MODE DEBUG\_CONSOLE\_SYNCHRONIZATION\_BM

If other RTOS is used, you can implement the RTOS's specific synchronization mechanism in fsl.log.c If synchronization is disabled, log maybe messed on terminal.

#### 41.6.2.8 #define BOARD\_USE\_VIRTUALCOM 0U

# Chapter 42

## Debug Console Lite

### 42.1 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.

### 42.2 Function groups

#### 42.2.1 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);
```

Selects the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_None = 0U,
    kSerialPort_Uart = 1U,
} serial_port_type_t;
```

After the initialization is successful, `stdout` and `stdin` are connected to the selected peripheral. The debug console state is stored in the `debug_console_state_t` structure, such as shown here.

```
typedef struct DebugConsoleState
{
    uint8_t uartHandleBuffer[HAL_UART_HANDLE_SIZE];
    hal_uart_status_t (*putChar)(hal_uart_handle_t handle, const uint8_t *data, size_t length);
    hal_uart_status_t (*getChar)(hal_uart_handle_t handle, uint8_t *data, size_t length);
    serial_port_type_t type;
} debug_console_state_t;
```

This example shows how to call the [DbgConsole\\_Init\(\)](#) given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_USART_INSTANCE, BOARD_DEBUG_USART_BAUDRATE,
    BOARD_DEBUG_USART_TYPE,
    BOARD_DEBUG_USART_CLK_FREQ);
```

## 42.2.2 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                              |

| specifier | Description |
|-----------|-------------|
|-----------|-------------|

- 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 */

```

### 42.2.3 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 UART. 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_-<br/>REDIRECT_TO_SDK</code>             | defined                            | UART                | UART                |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_SDK</code>             | undefined                          | UART                | semihost            |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_TO-<br/>OLCHAIN</code> | defined                            | UART                | UART                |
| <code>DEBUGCONSOLE_-<br/>REDIRECT_TO_TO-<br/>OLCHAIN</code> | undefined                          | semihost            | semihost            |
| <code>DEBUGCONSOLE_-<br/>DISABLE</code>                     | defined                            | No output           | UART                |
| <code>DEBUGCONSOLE_-<br/>DISABLE</code>                     | undefined                          | No output           | semihost            |

|                         |                              |               |               |
|-------------------------|------------------------------|---------------|---------------|
| <b>SDK_DEBUGCONSOLE</b> | <b>SDK_DEBUGCONSOLE_UART</b> | <b>PRINTF</b> | <b>printf</b> |
|-------------------------|------------------------------|---------------|---------------|

## 42.3 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.

## 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_FLOAT_ENABLE 0U`  
*Definition to printf the float number.*
- `#define SCANF_FLOAT_ENABLE 0U`  
*Definition to scanf the float number.*
- `#define PRINTF_ADVANCED_ENABLE 0U`  
*Definition to support advanced format specifier for printf.*
- `#define SCANF_ADVANCED_ENABLE 0U`  
*Definition to support advanced format specifier for scanf.*
- `#define PRINTF_DbgConsole_Printf`  
*Definition to select redirect toolchain printf, scanf to uart or not.*

## 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 dbgConsoleCh)`  
*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.*

## 42.4 Macro Definition Documentation

### 42.4.1 #define DEBUGCONSOLE\_REDIRECT\_TO\_TOOLCHAIN 0U

Select toolchain printf and scanf.

**42.4.2 #define DEBUGCONSOLE\_REDIRECT\_TO\_SDK 1U**

**42.4.3 #define DEBUGCONSOLE\_DISABLE 2U**

**42.4.4 #define SDK\_DEBUGCONSOLE DEBUGCONSOLE\_REDIRECT\_TO\_SDK**

**42.4.5 #define PRINTF\_FLOAT\_ENABLE 0U**

**42.4.6 #define SCANF\_FLOAT\_ENABLE 0U**

**42.4.7 #define PRINTF\_ADVANCED\_ENABLE 0U**

**42.4.8 #define SCANF\_ADVANCED\_ENABLE 0U**

**42.4.9 #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.

## 42.5 Function Documentation

**42.5.1 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, frequency of peripheral source clock, and base address at the specified baud rate. 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></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 |
| <i>kStatus_Fail</i>    | Execution failure      |

#### 42.5.2 status\_t DbgConsole\_Deinit ( void )

Call this function to disable debug log messages to be output via the specified peripheral base address and at the specified baud rate.

Returns

Indicates whether de-initialization was successful or not.

#### 42.5.3 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.

#### 42.5.4 status\_t DbgConsole\_ExitLowpower ( void )

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

#### 42.5.5 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.

#### 42.5.6 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.

#### 42.5.7 int DbgConsole\_Putchar ( int *dbgConsoleCh* )

Call this function to write a character to stdout.

Parameters

|                     |                          |
|---------------------|--------------------------|
| <i>dbgConsoleCh</i> | Character to be written. |
|---------------------|--------------------------|

Returns

Returns the character written.

#### 42.5.8 int DbgConsole\_Scanf ( char \* *fmt\_s*, ... )

Call this function to read formatted data from the standard input stream.

Parameters

|              |                        |
|--------------|------------------------|
| <i>fmt_s</i> | Format control string. |
|--------------|------------------------|

Returns

Returns the number of fields successfully converted and assigned.

#### 42.5.9 **int DbgConsole\_Getchar ( void )**

Call this function to read a character from standard input.

Returns

Returns the character read.

# 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_PRINT_COPYRIGHT` (1U)  
*Whether print copyright.*
- #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\_PRINT\_COPYRIGHT (1U)****43.4.11 #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. |
|-------------|--------------------------------------|

**43.4.12 #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.                                                                              |

#### 43.4.13 #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_Open-WriteHandleFailed</i> | Open the write handle failed.                    |
| <i>kStatus_SHELL_Open-ReadHandleFailed</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

## CODEC Driver

### 44.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

### Modules

- [CODEC Common Driver](#)
- [CODEC I2C Driver](#)
- [Wm8904](#)

## 44.2 CODEC Common Driver

### 44.2.1 Overview

The codec common driver provides a codec control abstraction interface.

### Modules

- [CODEC Adapter](#)
- [Wm8904\\_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.*

### 44.2.2 Data Structure Documentation

#### 44.2.2.1 struct codec\_config\_t

##### Data Fields

- `uint32_t codecDevType`  
*codec type*
- `void *codecDevConfig`  
*Codec device specific configuration.*

#### 44.2.2.2 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*

#### 44.2.2.3 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*

#### 44.2.3 Macro Definition Documentation

##### 44.2.3.1 #define FSL\_CODEC\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

#### 44.2.4 Enumeration Type Documentation

##### 44.2.4.1 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.

#### 44.2.4.2 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.

#### 44.2.4.3 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.

#### 44.2.4.4 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

#### 44.2.4.5 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

#### 44.2.4.6 enum codec\_module\_ctrl\_cmd\_t

Enumerator

*kCODEC\_ModuleSwitchI2SInInterface* module digital interface siwtch.

#### 44.2.4.7 anonymous enum

Enumerator

*kCODEC\_ModuleI2SInInterfacePCM* Pcm interface.  
*kCODEC\_ModuleI2SInInterfaceDSD* DSD interface.

#### 44.2.4.8 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

#### 44.2.4.9 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

#### 44.2.4.10 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

#### 44.2.4.11 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

#### 44.2.4.12 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

#### 44.2.4.13 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

## 44.2.5 Function Documentation

### 44.2.5.1 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.

### 44.2.5.2 status\_t CODEC\_Deinit ( ***codec\_handle\_t \* handle*** )

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

kStatus\_Success is success, else de-initial failed.

### 44.2.5.3 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.

#### 44.2.5.4 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.

#### 44.2.5.5 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.

#### 44.2.5.6 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.

#### 44.2.5.7 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.

#### 44.2.5.8 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.

#### 44.2.5.9 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.

#### 44.2.5.10 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.

## 44.3 CODEC I2C Driver

### 44.3.1 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.*

## 44.3.2 Data Structure Documentation

### 44.3.2.1 struct codec\_i2c\_config\_t

#### Data Fields

- `uint32_t codecI2CInstance`  
*i2c bus instance*
- `uint32_t codecI2CSourceClock`  
*i2c bus source clock frequency*

## 44.3.3 Enumeration Type Documentation

### 44.3.3.1 enum codec\_reg\_addr\_t

Enumerator

`kCODEC_RegAddr8Bit` 8-bit register address.  
`kCODEC_RegAddr16Bit` 16-bit register address.

### 44.3.3.2 enum codec\_reg\_width\_t

Enumerator

`kCODEC_RegWidth8Bit` 8-bit register width.  
`kCODEC_RegWidth16Bit` 16-bit register width.  
`kCODEC_RegWidth32Bit` 32-bit register width.

## 44.3.4 Function Documentation

### 44.3.4.1 status\_t CODEC\_I2C\_Init ( void \* handle, uint32\_t i2cInstance, uint32\_t i2cBaudrate, uint32\_t i2cSourceClockHz )

Parameters

|                          |                                                                     |
|--------------------------|---------------------------------------------------------------------|
| <code>handle</code>      | i2c master handle.                                                  |
| <code>i2cInstance</code> | 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.

#### 44.3.4.2 status\_t CODEC\_I2C\_Deinit ( void \* *handle* )

Parameters

|               |                    |
|---------------|--------------------|
| <i>handle</i> | i2c master handle. |
|---------------|--------------------|

Returns

kStatus\_HAL\_I2cSuccess is success, else deinitial failed.

#### 44.3.4.3 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.

#### 44.3.4.4 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.

# Chapter 45

## Serial Manager

### 45.1 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\\_uart](#)
- [Serial\\_port\\_usb](#)

### 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](#) (1U)  
*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\\_PORT\\_TYPE\\_BLE\\_WU](#) (0U)  
*Enable or disable BLE WU port (1 - enable, 0 - disable)*

- #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** (44U)  
*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 + 124U)  
*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 int32\_t(\* **serial\_manager\_lowpower\_critical\_callback\_t** )(int32\_t power\_mode)  
*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`,
   
    `kSerialPort_None` = 0U,
   
    `kSerialPort_Uart` = 1U,
   
    `kSerialPort_UsbCdc`,
   
    `kSerialPort_Swo`,
   
    `kSerialPort_Virtual`,
   
    `kSerialPort_Rpmsg`,
   
    `kSerialPort_UartDma`,
   
    `kSerialPort_SpiMaster`,
   
    `kSerialPort_SpiSlave`,
   
    `kSerialPort_BleWu` }
   
        *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 *serialConfig)`
  
          *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_WriteNonBlocking (serial_write_handle_t writeHandle, uint8_t *buffer, uint32_t length)`  
*Transmits data with the non-blocking mode.*
- `serial_manager_status_t SerialManager_ReadNonBlocking (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length)`  
*Reads data with the non-blocking mode.*
- `serial_manager_status_t SerialManager_TryRead (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length, uint32_t *receivedLength)`  
*Tries to read data.*
- `serial_manager_status_t SerialManager_CancelWriting (serial_write_handle_t writeHandle)`  
*Cancels unfinished send transmission.*
- `serial_manager_status_t SerialManager_CancelReading (serial_read_handle_t readHandle)`  
*Cancels unfinished receive transmission.*
- `serial_manager_status_t SerialManager_InstallTxCallback (serial_write_handle_t writeHandle, serial_manager_callback_t callback, void *callbackParam)`  
*Installs a TX callback and callback parameter.*
- `serial_manager_status_t SerialManager_InstallRxCallback (serial_read_handle_t readHandle, serial_manager_callback_t callback, void *callbackParam)`  
*Installs a RX callback and callback parameter.*
- `static bool SerialManager_needPollingIsr (void)`  
*Check if need polling ISR.*
- `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.*

## 45.2 Data Structure Documentation

### 45.2.1 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.

### 45.2.2 struct serial\_manager\_callback\_message\_t

#### Data Fields

- `uint8_t * buffer`  
*Transferred buffer.*
- `uint32_t length`  
*Transferred data length.*

## 45.3 Macro Definition Documentation

### 45.3.1 #define SERIAL\_MANAGER\_WRITE\_TIME\_DELAY\_DEFAULT\_VALUE (1U)

### 45.3.2 #define SERIAL\_MANAGER\_READ\_TIME\_DELAY\_DEFAULT\_VALUE (1U)

### 45.3.3 #define SERIAL\_MANAGER\_USE\_COMMON\_TASK (0U)

Macro to determine whether use common task.

**45.3.4 #define SERIAL\_MANAGER\_HANDLE\_SIZE (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 124U)****45.3.5 #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. |
|-------------|-----------------------------------------------|

**45.3.6 #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. |
|-------------|-----------------------------------------------------|

**45.3.7 #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. |
|-------------|----------------------------------------------------|

**45.3.8 #define SERIAL\_MANAGER\_TASK\_PRIORITY (2U)**

**45.3.9 #define SERIAL\_MANAGER\_TASK\_STACK\_SIZE (1000U)**

## 45.4 Enumeration Type Documentation

### 45.4.1 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.
- 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.  
*kSerialPort\_BleWu* Serial port BLE WU.

#### 45.4.2 enum serial\_manager\_type\_t

Enumerator

*kSerialManager\_NonBlocking* None blocking handle.  
*kSerialManager\_Blocking* Blocking handle.

#### 45.4.3 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.

### 45.5 Function Documentation

#### 45.5.1 serial\_manager\_status\_t SerialManager\_Init ( serial\_handle\_t serialHandle, const serial\_manager\_config\_t \* serialConfig )

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);
*

```

Example below shows how to use this API to configure the Serial Manager task configuration. For example if user need do specfical configuration(s\_os\_thread\_def\_serialmanager)for the serial manager task,

```

* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
* const osa_task_def_t s_os_thread_def_serialmanager = {
*     .tpriority = 4,
*     .instances = 1,
*     .stacksize = 2048,
* };
* 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;
* config.serialTaskConfig = (osa_task_def_t *)&s_os_thread_def_serialmanager,
* 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);
*
```

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | Pointer to point to a memory space of size <b>SERIAL_MANAGER_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>SERIAL_MANAGER_HANDLE_DEFINE(serialHandle)</b> ; or <b>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</b> |
| <i>serialConfig</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. |

**45.5.2 *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. |

### 45.5.3 `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

|                           |                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>serialHandle</code> | The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.                                                                                                                                                                                                                                                    |
| <code>writeHandle</code>  | 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: <code>SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle)</code> ; or <code>uint32_t writeHandle[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code> |

#### Return values

|                                                   |                                |
|---------------------------------------------------|--------------------------------|
| <code>kStatus_SerialManager_Error</code>          | An error occurred.             |
| <code>kStatus_SerialManager_HandleConflict</code> | The writing handle was opened. |
| <code>kStatus_SerialManager_Success</code>        | 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);
*

```

#### 45.5.4 **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. |
|---------------------------------------|-------------------------------|

#### 45.5.5 **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));
*
```

#### 45.5.6 **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. |
|---------------------------------------|-------------------------------|

#### 45.5.7 **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.                                               |

#### 45.5.8 `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

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer.             |
| <i>buffer</i>     | Start address of the data to store the received data. |
| <i>length</i>     | The length of the data to be received.                |

Return values

|                                       |                                                                      |
|---------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_-Success</i> | Successfully received all data.                                      |
| <i>kStatus_SerialManager_-Busy</i>    | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_-Error</i>   | An error occurred.                                                   |

#### 45.5.9 `serial_manager_status_t SerialManager_WriteNonBlocking ( serial_write_handle_t writeHandle, uint8_t * buffer, uint32_t length )`

This is a non-blocking function, which returns directly without waiting for all data to be sent. When all data is sent, the module notifies the upper layer through a TX callback function and passes the status parameter `kStatus_SerialManager_Success`. 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 TX callback is mandatory before the function could be used.

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.                                               |

#### 45.5.10 `serial_manager_status_t SerialManager_ReadNonBlocking ( serial_read_handle_t readHandle, uint8_t * buffer, uint32_t length )`

This is a non-blocking function, which returns directly without waiting for all data to be received. When all data is received, the module driver notifies the upper layer through a RX callback function and passes the status parameter `kStatus_SerialManager_Success`. 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 RX callback is mandatory before the function could be used.

##### 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.                                                   |

#### 45.5.11 `serial_manager_status_t SerialManager_TryRead ( serial_read_handle_t readHandle, uint8_t * buffer, uint32_t length, uint32_t * receivedLength )`

The function tries to read data from internal ring buffer. If the ring buffer is not empty, the data will be copied from ring buffer to up layer buffer. The copied length is the minimum of the ring buffer and up layer length. After the data is copied, the actual data length is passed by the parameter length. And There can only one buffer for receiving for the reading handle at the same time.

Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>readHandle</i>     | The serial manager module handle pointer.             |
| <i>buffer</i>         | Start address of the data to store the received data. |
| <i>length</i>         | The length of the data to be received.                |
| <i>receivedLength</i> | Length received from the ring buffer directly.        |

Return values

|                                       |                                                                      |
|---------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_-Success</i> | Successfully received all data.                                      |
| <i>kStatus_SerialManager_-Busy</i>    | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_-Error</i>   | An error occurred.                                                   |

#### 45.5.12 **serial\_manager\_status\_t SerialManager\_CancelWriting ( serial\_write\_handle\_t writeHandle )**

The function cancels unfinished send transmission. When the transfer is canceled, the module notifies the upper layer through a TX callback function and passes the status parameter [kStatus\\_SerialManager\\_-Canceled](#).

Note

The function [SerialManager\\_CancelWriting](#) cannot be used to abort the transmission of the function [SerialManager\\_WriteBlocking](#).

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
|--------------------|-------------------------------------------|

Return values

|                                       |                                     |
|---------------------------------------|-------------------------------------|
| <i>kStatus_SerialManager_-Success</i> | Get successfully abort the sending. |
| <i>kStatus_SerialManager_-Error</i>   | An error occurred.                  |

#### 45.5.13 `serial_manager_status_t SerialManager_CancelReading ( serial_read_handle_t readHandle )`

The function cancels unfinished receive transmission. When the transfer is canceled, the module notifies the upper layer through a RX callback function and passes the status parameter `kStatus_SerialManager_Canceled`.

##### Note

The function `SerialManager_CancelReading` cannot be used to abort the transmission of the function `SerialManager_ReadBlocking`.

##### Parameters

|                         |                                           |
|-------------------------|-------------------------------------------|
| <code>readHandle</code> | The serial manager module handle pointer. |
|-------------------------|-------------------------------------------|

##### Return values

|                                            |                                       |
|--------------------------------------------|---------------------------------------|
| <code>kStatus_SerialManager_Success</code> | Get successfully abort the receiving. |
| <code>kStatus_SerialManager_Error</code>   | An error occurred.                    |

#### 45.5.14 `serial_manager_status_t SerialManager_InstallTxCallback ( serial_write_handle_t writeHandle, serial_manager_callback_t callback, void * callbackParam )`

This function is used to install the TX callback and callback parameter for the serial manager module. When any status of TX transmission changed, the driver will notify the upper layer by the installed callback function. And the status is also passed as status parameter when the callback is called.

##### Parameters

|                            |                                           |
|----------------------------|-------------------------------------------|
| <code>writeHandle</code>   | The serial manager module handle pointer. |
| <code>callback</code>      | The callback function.                    |
| <code>callbackParam</code> | The parameter of the callback function.   |

Return values

|                                      |                                    |
|--------------------------------------|------------------------------------|
| <i>kStatus_SerialManager_Success</i> | Successfully install the callback. |
|--------------------------------------|------------------------------------|

#### 45.5.15 **serial\_manager\_status\_t SerialManager\_InstallRxCallback ( serial\_read\_handle\_t *readHandle*, serial\_manager\_callback\_t *callback*, void \* *callbackParam* )**

This function is used to install the RX callback and callback parameter for the serial manager module. When any status of RX transmission changed, the driver will notify the upper layer by the installed callback function. And the status is also passed as status parameter when the callback is called.

Parameters

|                      |                                           |
|----------------------|-------------------------------------------|
| <i>readHandle</i>    | The serial manager module handle pointer. |
| <i>callback</i>      | The callback function.                    |
| <i>callbackParam</i> | The parameter of the callback function.   |

Return values

|                                      |                                    |
|--------------------------------------|------------------------------------|
| <i>kStatus_SerialManager_Success</i> | Successfully install the callback. |
|--------------------------------------|------------------------------------|

#### 45.5.16 **static bool SerialManager\_needPollingIsr ( void ) [inline], [static]**

This function is used to check if need polling ISR.

Return values

|             |                  |
|-------------|------------------|
| <i>TRUE</i> | if need polling. |
|-------------|------------------|

#### 45.5.17 **serial\_manager\_status\_t SerialManager\_EnterLowpower ( serial\_handle\_t *serialHandle* )**

This function is used to prepare to enter low power consumption.

Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

Return values

|                                       |                       |
|---------------------------------------|-----------------------|
| <i>kStatus_SerialManager_-Success</i> | Successful operation. |
|---------------------------------------|-----------------------|

#### 45.5.18 **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. |
|---------------------------------------|-----------------------|

#### 45.5.19 **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. |
|-------------------|-------------------------------------------------------------------|

# **Chapter 46**

## **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>.

### **46.1 Function groups**

#### **46.1.1 SPI CMSIS GetVersion Operation**

This function group will return the SPI CMSIS Driver version to user.

#### **46.1.2 SPI CMSIS GetCapabilities Operation**

This function group will return the capabilities of this driver.

#### **46.1.3 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.

#### **46.1.4 SPI CMSIS Transfer Operation**

This function group controls the transfer, master send/receive data, and slave send/receive data.

#### **46.1.5 SPI CMSIS Status Operation**

This function group gets the SPI transfer status.

## 46.1.6 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.

## 46.2 Typical use case

### 46.2.1 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 uninitialize */
DRIVER_MASTER_SPI.Uninitialize();
```

### 46.2.2 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 uninitialize */
DRIVER_SLAVE_SPI.Uninitialize();
```

# Chapter 47

## 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.

### 47.1 I2C CMSIS Driver

#### 47.1.1 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;
```

#### 47.1.2 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;

```

### 47.1.3 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 48

## 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.

### 48.1 USART Send Methods

#### 48.1.1 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)
{
}
```

#### 48.1.2 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 49

## CDOG

### 49.1 Overview

#### Files

- file `fsl_cdog.h`

#### Driver version

- `#define FSL_CDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`  
*Defines CDOG driver version 2.1.3.*

### CDOG Functional Operation

- `status_t CDOG_Init (CDOG_Type *base, cdog_config_t *conf)`  
*Initialize CDOG.*
- `void CDOG_Deinit (CDOG_Type *base)`  
*Deinitialize CDOG.*
- `void CDOG_GetDefaultConfig (cdog_config_t *conf)`  
*Sets the default configuration of CDOG.*
- `void CDOG_Stop (CDOG_Type *base, uint32_t stop)`  
*Stops secure counter and instruction timer.*
- `void CDOG_Start (CDOG_Type *base, uint32_t reload, uint32_t start)`  
*Sets secure counter and instruction timer values.*
- `void CDOG_Check (CDOG_Type *base, uint32_t check)`  
*Checks secure counter.*
- `void CDOG_Set (CDOG_Type *base, uint32_t stop, uint32_t reload, uint32_t start)`  
*Sets secure counter and instruction timer values.*
- `void CDOG_Add (CDOG_Type *base, uint32_t add)`  
*Add value to secure counter.*
- `void CDOG_Add1 (CDOG_Type *base)`  
*Add 1 to secure counter.*
- `void CDOG_Add16 (CDOG_Type *base)`  
*Add 16 to secure counter.*
- `void CDOG_Add256 (CDOG_Type *base)`  
*Add 256 to secure counter.*
- `void CDOG_Sub (CDOG_Type *base, uint32_t sub)`  
*brief Subtract value to secure counter*
- `void CDOG_Sub1 (CDOG_Type *base)`  
*Subtract 1 from secure counter.*
- `void CDOG_Sub16 (CDOG_Type *base)`  
*Subtract 16 from secure counter.*
- `void CDOG_Sub256 (CDOG_Type *base)`  
*Subtract 256 from secure counter.*
- `void CDOG_WritePersistent (CDOG_Type *base, uint32_t value)`

- *Set the CDOG persistent word.*
- `uint32_t CDOG_ReadPersistent (CDOG_Type *base)`  
*Get the CDOG persistent word.*

## 49.2 Macro Definition Documentation

### 49.2.1 `#define FSL_CDOG_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`

Change log:

- Version 2.1.3
  - Re-design multiple instance IRQs and Clocks
  - Add fix for RESTART command errata
- Version 2.1.2
  - Support multiple IRQs
  - Fix default CONTROL values
- Version 2.1.1
  - Remove bit CONTROL[CONTROL\_CTRL]
- Version 2.1.0
  - Rename CWT to CDOG
- Version 2.0.2
  - Fix MISRA-2012 issues
- Version 2.0.1
  - Fix doxygen issues
- Version 2.0.0
  - initial version

## 49.3 Function Documentation

### 49.3.1 `status_t CDOG_Init ( CDOG_Type * base, cdog_config_t * conf )`

This function initializes CDOG block and setting.

Parameters

|                   |                              |
|-------------------|------------------------------|
| <code>base</code> | CDOG peripheral base address |
| <code>conf</code> | CDOG configuration structure |

Returns

Status of the init operation

### 49.3.2 `void CDOG_Deinit ( CDOG_Type * base )`

This function deinitializes CDOG secure counter.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | CDOG peripheral base address |
|-------------|------------------------------|

### 49.3.3 void CDOG\_GetDefaultConfig ( *cdog\_config\_t* \* *conf* )

This function initialize CDOG config structure to default values.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>conf</i> | CDOG configuration structure |
|-------------|------------------------------|

### 49.3.4 void CDOG\_Stop ( *CDOG\_Type* \* *base*, *uint32\_t* *stop* )

This function stops instruction timer and secure counter. This also change state od CDOG to IDLE.

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | CDOG peripheral base address                                       |
| <i>stop</i> | expected value which will be compared with value of secure counter |

### 49.3.5 void CDOG\_Start ( *CDOG\_Type* \* *base*, *uint32\_t* *reload*, *uint32\_t* *start* )

This function sets value in RELOAD and START registers for instruction timer and secure counter

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | CDOG peripheral base address |
| <i>reload</i> | reload value                 |
| <i>start</i>  | start value                  |

### 49.3.6 void CDOG\_Check ( *CDOG\_Type* \* *base*, *uint32\_t* *check* )

This function compares stop value in handler with secure counter value by writting to RELOAD refister.

Parameters

|              |                              |
|--------------|------------------------------|
| <i>base</i>  | CDOG peripheral base address |
| <i>check</i> | expected (stop) value        |

#### 49.3.7 void CDOG\_Set ( CDOG\_Type \* *base*, uint32\_t *stop*, uint32\_t *reload*, uint32\_t *start* )

This function sets value in STOP, RELOAD and START registers for instruction timer and secure counter.

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | CDOG peripheral base address                                       |
| <i>stop</i>   | expected value which will be compared with value of secure counter |
| <i>reload</i> | reload value for instruction timer                                 |
| <i>start</i>  | start value for secure timer                                       |

#### 49.3.8 void CDOG\_Add ( CDOG\_Type \* *base*, uint32\_t *add* )

This function add specified value to secure counter.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CDOG peripheral base address. |
| <i>add</i>  | Value to be added.            |

#### 49.3.9 void CDOG\_Add1 ( CDOG\_Type \* *base* )

This function add 1 to secure counter.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CDOG peripheral base address. |
|-------------|-------------------------------|

#### 49.3.10 void CDOG\_Add16 ( CDOG\_Type \* *base* )

This function add 16 to secure counter.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CDOG peripheral base address. |
|-------------|-------------------------------|

#### 49.3.11 void CDOG\_Add256 ( CDOG\_Type \* *base* )

This function add 256 to secure counter.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CDOG peripheral base address. |
|-------------|-------------------------------|

#### 49.3.12 void CDOG\_Sub ( CDOG\_Type \* *base*, uint32\_t *sub* )

This function substract specified value to secure counter.

param base CDOG peripheral base address. param sub Value to be substracted.

#### 49.3.13 void CDOG\_Sub1 ( CDOG\_Type \* *base* )

This function substract specified 1 from secure counter.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CDOG peripheral base address. |
|-------------|-------------------------------|

#### 49.3.14 void CDOG\_Sub16 ( CDOG\_Type \* *base* )

This function substract specified 16 from secure counter.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CDOG peripheral base address. |
|-------------|-------------------------------|

#### 49.3.15 void CDOG\_Sub256 ( CDOG\_Type \* *base* )

This function substract specified 256 from secure counter.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CDOG peripheral base address. |
|-------------|-------------------------------|

#### 49.3.16 void CDOG\_WritePersistent ( **CDOG\_Type** \* *base*, **uint32\_t** *value* )

Parameters

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | CDOG peripheral base address. |
| <i>value</i> | The value to be written.      |

#### 49.3.17 **uint32\_t** CDOG\_ReadPersistent ( **CDOG\_Type** \* *base* )

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CDOG peripheral base address. |
|-------------|-------------------------------|

Returns

The persistent word.

# Chapter 50

## Dmic\_dma\_driver

### 50.1 Overview

#### Files

- file [fsl\\_dmic\\_dma.h](#)

#### Data Structures

- struct [dmic\\_transfer\\_t](#)  
*DMIC transfer structure.* [More...](#)
- struct [dmic\\_dma\\_handle\\_t](#)  
*DMIC DMA handle.* [More...](#)

#### Typedefs

- typedef void(\* [dmic\\_dma\\_transfer\\_callback\\_t](#) )(DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*DMIC transfer callback function.*

#### DMIC DMA version

- #define [FSL\\_DMIC\\_DMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 4, 0))  
*DMIC DMA driver version 2.4.0.*

#### DMA transactional

- [status\\_t DMIC\\_TransferCreateHandleDMA](#) (DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle, [dmic\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*rxDmaHandle)  
*Initializes the DMIC handle which is used in transactional functions.*
- [status\\_t DMIC\\_TransferReceiveDMA](#) (DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle, [dmic\\_transfer\\_t](#) \*xfer, uint32\_t channel)  
*Receives data using DMA.*
- [void DMIC\\_TransferAbortReceiveDMA](#) (DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle)  
*Aborts the received data using DMA.*
- [status\\_t DMIC\\_TransferGetReceiveCountDMA](#) (DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*
- [void DMIC\\_InstallDMADescriptorMemory](#) (dmic\_dma\_handle\_t \*handle, void \*linkAddr, size\_t linkNum)  
*Install DMA descriptor memory.*

## 50.2 Data Structure Documentation

### 50.2.1 struct dmic\_transfer\_t

#### Data Fields

- void \* **data**  
*The buffer of data to be transfer.*
- uint8\_t **dataWidth**  
*DMIC support 16bit/32bit.*
- size\_t **dataSize**  
*The byte count to be transfer.*
- uint8\_t **dataAddrInterleaveSize**  
*destination address interleave size*
- struct \_dmic\_transfer \* **linkTransfer**  
*use to support link transfer*

#### Field Documentation

- (1) **void\* dmic\_transfer\_t::data**
- (2) **size\_t dmic\_transfer\_t::dataSize**

### 50.2.2 struct \_dmic\_dma\_handle

#### Data Fields

- DMIC\_Type \* **base**  
*DMIC peripheral base address.*
- **dma\_handle\_t \* rxDmaHandle**  
*The DMA RX channel used.*
- **dmic\_dma\_transfer\_callback\_t callback**  
*Callback function.*
- void \* **userData**  
*DMIC callback function parameter.*
- size\_t **transferSize**  
*Size of the data to receive.*
- volatile uint8\_t **state**  
*Internal state of DMIC DMA transfer.*
- uint32\_t **channel**  
*DMIC channel used.*
- bool **isChannelValid**  
*DMIC channel initialization flag.*
- **dma\_descriptor\_t \* desLink**  
*descriptor pool pointer*
- size\_t **linkNum**  
*number of descriptor in descriptors pool*

**Field Documentation**

- (1) `DMIC_Type* dmic_dma_handle_t::base`
- (2) `dma_handle_t* dmic_dma_handle_t::rxDmaHandle`
- (3) `dmic_dma_transfer_callback_t dmic_dma_handle_t::callback`
- (4) `void* dmic_dma_handle_t::userData`
- (5) `size_t dmic_dma_handle_t::transferSize`
- (6) `uint32_t dmic_dma_handle_t::channel`

**50.3 Typedef Documentation**

**50.3.1 `typedef void(* dmic_dma_transfer_callback_t)(DMIC_Type *base, dmic_dma_handle_t *handle, status_t status, void *userData)`**

**50.4 Function Documentation**

**50.4.1 `status_t DMIC_TransferCreateHandleDMA ( DMIC_Type * base, dmic_dma_handle_t * handle, dmic_dma_transfer_callback_t callback, void * userData, dma_handle_t * rxDmaHandle )`**

## Parameters

|                    |                                                      |
|--------------------|------------------------------------------------------|
| <i>base</i>        | DMIC peripheral base address.                        |
| <i>handle</i>      | Pointer to <code>dmic_dma_handle_t</code> structure. |
| <i>callback</i>    | Callback function.                                   |
| <i>userData</i>    | User data.                                           |
| <i>rxDmaHandle</i> | User-requested DMA handle for RX DMA transfer.       |

**50.4.2 `status_t DMIC_TransferReceiveDMA ( DMIC_Type * base, dmic_dma_handle_t * handle, dmic_transfer_t * xfer, uint32_t channel )`**

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>    | DMIC DMA transfer structure. See <a href="#">dmic_transfer_t</a> . |
| <i>channel</i> | DMIC start channel number.                                         |

Return values

|                        |  |
|------------------------|--|
| <i>kStatus_Success</i> |  |
|------------------------|--|

#### 50.4.3 void DMIC\_TransferAbortReceiveDMA ( ***DMIC\_Type \* base,*** ***dmc\_dma\_handle\_t \* handle*** )

This function aborts the received data using DMA.

Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>base</i>   | DMIC peripheral base address          |
| <i>handle</i> | Pointer to dmc_dma_handle_t structure |

#### 50.4.4 status\_t DMIC\_TransferGetReceiveCountDMA ( ***DMIC\_Type \* base,*** ***dmc\_dma\_handle\_t \* handle, uint32\_t \* count*** )

This function gets the number of bytes that have been received.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | DMIC peripheral base address. |
| <i>handle</i> | DMIC handle pointer.          |
| <i>count</i>  | Receive 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; |

#### 50.4.5 void DMIC\_InstallDMADescriptorMemory ( *dmic\_dma\_handle\_t \* handle,* *void \* linkAddr, size\_t linkNum* )

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, it should be called after DMI-C\_TransferCreateHandleDMA. 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 DMA channel transfer handle. |
| <i>linkAddr</i> | DMA link descriptor address.            |
| <i>linkNum</i>  | DMA link descriptor number.             |

# Chapter 51

## Flexspi\_dma

### 51.1 Overview

#### Data Structures

- struct `flexspi_dma_handle_t`

*FLEXSPI DMA transfer handle, users should not touch the content of the handle.* [More...](#)

#### Typedefs

- typedef void(\* `flexspi_dma_callback_t`)`(FLEXSPI_Type *base, flexspi_dma_handle_t *handle, status_t status, void *userData)`

*FLEXSPI dma transfer callback function for finish and error.*

#### Enumerations

- enum `flexspi_dma_transfer_nsize_t` {  
  `kFLEXPSI_DMANSize1Bytes` = 0x1U,  
  `kFLEXPSI_DMANSize2Bytes` = 0x2U,  
  `kFLEXPSI_DMANSize4Bytes` = 0x4U }  
*dma transfer configuration*

#### Driver version

- #define `FSL_FLEXSPI_DMA_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 1)`)  
*FLEXSPI DMA driver version 2.2.1.*

#### FLEXSPI dma Transactional

- void `FLEXSPI_TransferCreateHandleDMA` (`FLEXSPI_Type *base, flexspi_dma_handle_t *handle, flexspi_dma_callback_t callback, void *userData, dma_handle_t *txDmaHandle, dma_handle_t *rxDmaHandle)`  
*Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.*
- void `FLEXSPI_TransferUpdateSizeDMA` (`FLEXSPI_Type *base, flexspi_dma_handle_t *handle, flexspi_dma_transfer_nsize_t nsize)`  
*Update FLEXSPI DMA transfer source data transfer size(SSIZE) and destination data transfer size(DSI-ZE).*
- `status_t FLEXSPI_TransferDMA` (`FLEXSPI_Type *base, flexspi_dma_handle_t *handle, flexspi_transfer_t *xfer)`  
*Transfers FLEXSPI data using an dma non-blocking method.*
- void `FLEXSPI_TransferAbortDMA` (`FLEXSPI_Type *base, flexspi_dma_handle_t *handle)`  
*Aborts the transfer data using dma.*
- `status_t FLEXSPI_TransferGetTransferCountDMA` (`FLEXSPI_Type *base, flexspi_dma_handle_t *handle, size_t *count)`

*Gets the transferred counts of transfer.*

## 51.2 Data Structure Documentation

### 51.2.1 struct \_flexspi\_dma\_handle

#### Data Fields

- **dma\_handle\_t \* txDmaHandle**  
*dma handler for FLEXSPI Tx.*
- **dma\_handle\_t \* rxDmaHandle**  
*dma handler for FLEXSPI Rx.*
- **size\_t transferSize**  
*Bytes need to transfer.*
- **flexspi\_dma\_transfer\_nsize\_t nsize**  
*dma SSIZE/DSIZE in each transfer.*
- **uint8\_t nbytes**  
*dma minor byte transfer count initially configured.*
- **uint8\_t count**  
*The transfer data count in a DMA request.*
- **uint32\_t state**  
*Internal state for FLEXSPI dma transfer.*
- **flexspi\_dma\_callback\_t completionCallback**  
*A callback function called after the dma transfer is finished.*
- **void \* userData**  
*User callback parameter.*

#### Field Documentation

- (1) **dma\_handle\_t\* flexspi\_dma\_handle\_t::txDmaHandle**
- (2) **dma\_handle\_t\* flexspi\_dma\_handle\_t::rxDmaHandle**
- (3) **size\_t flexspi\_dma\_handle\_t::transferSize**
- (4) **flexspi\_dma\_transfer\_nsize\_t flexspi\_dma\_handle\_t::nsize**
- (5) **uint8\_t flexspi\_dma\_handle\_t::nbytes**
- (6) **uint8\_t flexspi\_dma\_handle\_t::count**
- (7) **uint32\_t flexspi\_dma\_handle\_t::state**
- (8) **flexspi\_dma\_callback\_t flexspi\_dma\_handle\_t::completionCallback**

## 51.3 Macro Definition Documentation

### 51.3.1 #define FSL\_FLEXSPI\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 1))

## 51.4 Enumeration Type Documentation

### 51.4.1 enum flexspi\_dma\_transfer\_nsize\_t

Enumerator

*kFLEXSPI\_DMAnSize1Bytes* Source/Destination data transfer size is 1 byte every time.

*kFLEXSPI\_DMAnSize2Bytes* Source/Destination data transfer size is 2 bytes every time.

*kFLEXSPI\_DMAnSize4Bytes* Source/Destination data transfer size is 4 bytes every time.

## 51.5 Function Documentation

### 51.5.1 void FLEXSPI\_TransferCreateHandleDMA ( *FLEXSPI\_Type* \* *base*, *flexspi\_dma\_handle\_t* \* *handle*, *flexspi\_dma\_callback\_t* *callback*, *void* \* *userData*, *dma\_handle\_t* \* *txDmaHandle*, *dma\_handle\_t* \* *rxDmaHandle* )

Parameters

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | FLEXSPI peripheral base address                |
| <i>handle</i>      | Pointer to flexspi_dma_handle_t structure      |
| <i>callback</i>    | FLEXSPI callback, NULL means no callback.      |
| <i>userData</i>    | User callback function data.                   |
| <i>txDmaHandle</i> | User requested DMA handle for TX DMA transfer. |
| <i>rxDmaHandle</i> | User requested DMA handle for RX DMA transfer. |

### 51.5.2 void FLEXSPI\_TransferUpdateSizeDMA ( *FLEXSPI\_Type* \* *base*, *flexspi\_dma\_handle\_t* \* *handle*, *flexspi\_dma\_transfer\_nsize\_t* *nsize* )

Parameters

|               |                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address                                                                                 |
| <i>handle</i> | Pointer to flexspi_dma_handle_t structure                                                                       |
| <i>nsize</i>  | FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_DMAnSize1Bytes(one byte). |

See Also

[flexspi\\_dma\\_transfer\\_nsize\\_t](#).

### 51.5.3 status\_t FLEXSPI\_TransferDMA ( **FLEXSPI\_Type** \* *base*, **flexspi\_dma\_handle\_t** \* *handle*, **flexspi\_transfer\_t** \* *xfer* )

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.          |
| <i>handle</i> | Pointer to flexspi_dma_handle_t structure |
| <i>xfer</i>   | FLEXSPI transfer structure.               |

Return values

|                                |                                                                                                             |
|--------------------------------|-------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FLEXSPI_Busy</i>    | FLEXSPI is busy transfer.                                                                                   |
| <i>kStatus_InvalidArgument</i> | The watermark configuration is invalid, the watermark should be power of 2 to do successfully DMA transfer. |
| <i>kStatus_Success</i>         | FLEXSPI successfully start dma transfer.                                                                    |

#### 51.5.4 void FLEXSPI\_TransferAbortDMA ( **FLEXSPI\_Type** \* *base*, **flexspi\_dma\_handle\_t** \* *handle* )

This function aborts the transfer data using dma.

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.          |
| <i>handle</i> | Pointer to flexspi_dma_handle_t structure |

#### 51.5.5 status\_t FLEXSPI\_TransferGetTransferCountDMA ( **FLEXSPI\_Type** \* *base*, **flexspi\_dma\_handle\_t** \* *handle*, **size\_t** \* *count* )

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.           |
| <i>handle</i> | Pointer to flexspi_dma_handle_t structure. |
| <i>count</i>  | Bytes transfer.                            |

Return values

## Function Documentation

|                                      |                                                                |
|--------------------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>               | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferIn-Progress</i> | There is not a non-blocking transaction currently in progress. |

# Chapter 52

## I2c\_dma\_driver

### 52.1 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.*

## 52.2 Data Structure Documentation

### 52.2.1 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`

## 52.3 Macro Definition Documentation

### 52.3.1 #define FSL\_I2C\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))

## 52.4 Typedef Documentation

52.4.1 **typedef void(\* i2c\_master\_dma\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

52.4.2 **typedef void(\* flexcomm\_i2c\_dma\_master\_irq\_handler\_t)(I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle)**

## 52.5 Function Documentation

52.5.1 **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                           |

52.5.2 **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. |

### 52.5.3 **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. |

### 52.5.4 **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 |

# Chapter 53

## I2s\_dma\_driver

### 53.1 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\(2, 3, 3\)\)](#)  
*I2S DMA driver version 2.3.3.*

#### 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.*

## 53.2 Data Structure Documentation

### 53.2.1 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*

## 53.3 Macro Definition Documentation

### 53.3.1 #define FSL\_I2S\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 3))

## 53.4 Typedef Documentation

### 53.4.1 `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. |

## 53.5 Function Documentation

53.5.1 **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.                        |

53.5.2 **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. |
|-------------------------|------------------------------------------------------|

### 53.5.3 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. |

### 53.5.4 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.                        |

### 53.5.5 *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. |

### 53.5.6 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>         |                                  |

### 53.5.7 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.              |

### 53.5.8 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> |
|------------------------|

### 53.5.9 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 54

## Spi\_dma\_driver

### 54.1 Overview

#### 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, 2, 1))  
*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.

## 54.2 Data Structure Documentation

### 54.2.1 struct \_spi\_dma\_handle

#### Data Fields

- volatile bool **txInProgress**  
*Send transfer finished.*
- volatile bool **rxInProgress**  
*Receive transfer finished.*
- uint8\_t **bytesPerFrame**  
*Bytes in a frame for SPI transfer.*
- uint8\_t **lastwordBytes**  
*The Bytes of lastword for master.*
- **dma\_handle\_t \* txHandle**  
*DMA handler for SPI send.*
- **dma\_handle\_t \* rxHandle**  
*DMA handler for SPI receive.*
- **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.*
- uint32\_t **instance**  
*Index of SPI instance.*
- const uint8\_t \* **txNextData**  
*The pointer of next time tx data.*
- const uint8\_t \* **txEndData**  
*The pointer of end of data.*
- uint8\_t \* **rxNextData**  
*The pointer of next time rx data.*
- uint8\_t \* **rxEndData**  
*The pointer of end of rx data.*
- uint32\_t **dataBytesEveryTime**  
*Bytes in a time for DMA transfer, default is DMA\_MAX\_TRANSFER\_COUNT.*

## 54.3 Macro Definition Documentation

### 54.3.1 #define FSL\_SPI\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 1))

## 54.4 Typedef Documentation

**54.4.1 `typedef void(* spi_dma_callback_t)(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)`**

## 54.5 Function Documentation

**54.5.1 `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. |

**54.5.2 `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. |

#### 54.5.3 **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.

#### 54.5.4 **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. |

#### 54.5.5 **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. |

#### 54.5.6 **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.      |

#### 54.5.7 **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.

#### 54.5.8 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.      |

#### 54.5.9 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 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.

# Chapter 55

## Usart\_dma\_driver

### 55.1 Overview

#### Files

- file [fsl\\_usart\\_dma.h](#)

#### Data Structures

- struct [usart\\_dma\\_handle\\_t](#)  
*UART 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.*

## 55.2 Data Structure Documentation

### 55.2.1 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`

## 55.3 Macro Definition Documentation

### 55.3.1 #define FSL\_USART\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 6, 0))

## 55.4 Typedef Documentation

**55.4.1 `typedef void(* usart_dma_transfer_callback_t)(USART_Type *base, usart_dma_handle_t *handle, status_t status, void *userData)`**

## 55.5 Function Documentation

**55.5.1 `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 usart_dma_handle_t 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. |

**55.5.2 `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.           |

### 55.5.3 **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.           |

### 55.5.4 **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 |

### 55.5.5 **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 |

### 55.5.6 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> ; |

### 55.5.7 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; |

## 55.5.8 CODEC Adapter

### 55.5.8.1 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,  
  kCODEC\_PCM512X,  
  kCODEC\_PCM186X }  
  *codec type*

### 55.5.8.2 Enumeration Type Documentation

#### 55.5.8.2.1 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  
**kCODEC\_PCM512X** pcm512x  
**kCODEC\_PCM186X** pcm186x

## 55.5.9 Wm8904\_adapter

### 55.5.9.1 Overview

#### 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 initilization.*
- `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)  
*set audio codec module power.*
- 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.*

### 55.5.9.2 Function Documentation

#### 55.5.9.2.1 `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.

#### 55.5.9.2.2 `status_t HAL_CODEC_WM8904_Deinit( void * handle )`

Parameters

|               |               |
|---------------|---------------|
| <i>handle</i> | codec handle. |
|---------------|---------------|

Returns

`kStatus_Success` is success, else de-initial failed.

#### 55.5.9.2.3 `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.

#### 55.5.9.2.4 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.

#### 55.5.9.2.5 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.

#### 55.5.9.2.6 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.

#### 55.5.9.2.7 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.

#### 55.5.9.2.8 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.

#### 55.5.9.2.9 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.

#### 55.5.9.2.10 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.

#### 55.5.9.2.11 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.

#### 55.5.9.2.12 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.

#### 55.5.9.2.13 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.

#### 55.5.9.2.14 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.

#### 55.5.9.2.15 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.

#### 55.5.9.2.16 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.

#### 55.5.9.2.17 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.

#### 55.5.9.2.18 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.

**55.5.9.2.19 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.

**55.5.9.2.20 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.

## 55.6 Wm8904

### 55.6.1 Overview

#### 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\_EnableDACTDMMMode (wm8904\_handle\_t \*handle, wm8904\_timeslot\_t timeSlot)**  
*Enable WM8904 DAC time slot.*
- **status\_t WM8904\_EnableADCTDMMMode (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.*

## 55.6.2 Data Structure Documentation

### 55.6.2.1 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*

### 55.6.2.2 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.*

### 55.6.2.3 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*

#### 55.6.2.4 struct `wm8904_handle_t`

##### Data Fields

- `wm8904_config_t * config`  
*wm8904 config pointer*
- `uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]`  
*i2c handle*

#### 55.6.3 Macro Definition Documentation

55.6.3.1 `#define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))`

55.6.3.2 `#define WM8904_I2C_ADDRESS (0x1A)`

55.6.3.3 `#define WM8904_I2C_BITRATE (400000U)`

#### 55.6.4 Enumeration Type Documentation

##### 55.6.4.1 anonymous enum

Enumerator

*kStatus\_WM8904\_Success* Success.  
*kStatus\_WM8904\_Fail* Failure.

##### 55.6.4.2 anonymous enum

Enumerator

*kWM8904\_LRCPolarityNormal* LRC polarity normal.  
*kWM8904\_LRCPolarityInverted* LRC polarity inverted.

##### 55.6.4.3 enum `wm8904_module_t`

Enumerator

*kWM8904\_ModuleADC* moduel ADC  
*kWM8904\_ModuleDAC* module DAC

*kWM8904\_ModulePGA* module PGA  
*kWM8904\_ModuleHeadphone* module headphone  
*kWM8904\_ModuleLineout* module line out

#### 55.6.4.4 anonymous enum

#### 55.6.4.5 enum `wm8904_timeslot_t`

Enumerator

*kWM8904\_TimeSlot0* time slot0  
*kWM8904\_TimeSlot1* time slot1

#### 55.6.4.6 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.

#### 55.6.4.7 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  $1536 * \text{sample rate} * \text{frame width}$ .

#### 55.6.4.8 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

#### 55.6.4.9 enum **wm8904\_bit\_width\_t**

Enumerator

*kWM8904\_BitWidth16* 16 bits  
*kWM8904\_BitWidth20* 20 bits  
*kWM8904\_BitWidth24* 24 bits  
*kWM8904\_BitWidth32* 32 bits

#### 55.6.4.10 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

#### 55.6.4.11 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

#### 55.6.4.12 anonymous enum

Enumerator

*kWM8904\_PlaySourcePGA* play source PGA, bypass ADC

*kWM8904\_PlaySourceDAC* play source Input3

#### 55.6.4.13 enum `wm8904_sys_clk_source_t`

Enumerator

*kWM8904\_SysClkSourceMCLK* wm8904 system clock soure from MCLK

*kWM8904\_SysClkSourceFLL* wm8904 system clock soure from FLL

#### 55.6.4.14 enum `wm8904_fll_clk_source_t`

Enumerator

*kWM8904\_FLLClkSourceMCLK* wm8904 FLL clock source from MCLK

### 55.6.5 Function Documentation

#### 55.6.5.1 `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.

#### 55.6.5.2 `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.

#### 55.6.5.3 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.

#### 55.6.5.4 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. |

#### 55.6.5.5 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.

#### 55.6.5.6 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. |
|---------------|-----------------------------------|

#### 55.6.5.7 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.

#### 55.6.5.8 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.

#### 55.6.5.9 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. |

#### 55.6.5.10 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.

#### 55.6.5.11 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.

#### 55.6.5.12 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.

#### 55.6.5.13 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.

#### 55.6.5.14 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.

#### 55.6.5.15 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.

#### 55.6.5.16 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.

#### 55.6.5.17 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.

#### 55.6.5.18 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> | true is power on, false is power down. |

Returns

kStatus\_WM8904\_Success if successful, different code otherwise..

#### 55.6.5.19 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..

#### 55.6.5.20 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.

#### 55.6.5.21 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.

#### 55.6.5.22 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..

### 55.6.5.23 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..

### 55.6.5.24 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.

## 55.7 Serial\_port\_swo

### 55.7.1 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*

### 55.7.2 Data Structure Documentation

#### 55.7.2.1 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.*

### 55.7.3 Enumeration Type Documentation

#### 55.7.3.1 enum serial\_port\_swo\_protocol\_t

Enumerator

- `kSerialManager_SwoProtocolManchester` SWO Manchester protocol.  
`kSerialManager_SwoProtocolNrz` SWO UART/NRZ protocol.

## 55.8 Serial\_port\_uart

### 55.8.1 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*

### 55.8.2 Enumeration Type Documentation

#### 55.8.2.1 enum serial\_port\_uart\_parity\_mode\_t

Enumerator

- kSerialManager\_UartParityDisabled** Parity disabled.  
**kSerialManager\_UartParityEven** Parity even enabled.  
**kSerialManager\_UartParityOdd** Parity odd enabled.

#### 55.8.2.2 enum serial\_port\_uart\_stop\_bit\_count\_t

Enumerator

- kSerialManager\_UartOneStopBit** One stop bit.  
**kSerialManager\_UartTwoStopBit** Two stop bits.

## 55.9 Serial\_port\_usb

### 55.9.1 Overview

#### 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.*

### 55.9.2 Data Structure Documentation

#### 55.9.2.1 struct serial\_port\_usb\_cdc\_config\_t

##### Data Fields

- `serial_port_usb_cdc_controller_index_t controllerIndex`  
*controller index*

### 55.9.3 Enumeration Type Documentation

#### 55.9.3.1 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.

**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.

