

MCUXpresso SDK API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKIMX8QMAPIRM

Rev. 1

Apr 2019



Contents

Chapter Introduction

Chapter Driver errors status

Chapter Architectural Overview

Chapter Trademarks

Chapter CACHE: CACHE Memory Controller

5.1	Overview	11
5.2	Function groups	11
5.3	Macro Definition Documentation	13
5.3.1	FSL_CACHE_DRIVER_VERSION	13
5.3.2	L1CODEBUSCACHE_LINESIZE_BYTE	13
5.3.3	L1SYSTEMBUSCACHE_LINESIZE_BYTE	13
5.4	Function Documentation	13
5.4.1	L1CACHE_InvalidateCodeCacheByRange	13
5.4.2	L1CACHE_CleanCodeCacheByRange	13
5.4.3	L1CACHE_CleanInvalidateCodeCacheByRange	14
5.4.4	L1CACHE_EnableCodeCacheWriteBuffer	14
5.4.5	L1CACHE_InvalidateSystemCacheByRange	14
5.4.6	L1CACHE_CleanSystemCacheByRange	15
5.4.7	L1CACHE_CleanInvalidateSystemCacheByRange	15
5.4.8	L1CACHE_EnableSystemCacheWriteBuffer	16
5.4.9	L1CACHE_InvalidateICacheByRange	17
5.4.10	L1CACHE_InvalidateDCacheByRange	17
5.4.11	L1CACHE_CleanDCacheByRange	17
5.4.12	L1CACHE_CleanInvalidateDCacheByRange	18
5.4.13	ICACHE_InvalidateByRange	18
5.4.14	DCACHE_InvalidateByRange	18
5.4.15	DCACHE_CleanByRange	19
5.4.16	DCACHE_CleanInvalidateByRange	19

Section Number	Title	Page Number
Chapter	Clock Driver	
6.1	Overview	21
6.2	Macro Definition Documentation	23
6.2.1	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	23
6.2.2	FSL_CLOCK_DRIVER_VERSION	23
6.2.3	MU_CLOCKS	23
6.2.4	GPIO_CLOCKS	23
6.2.5	RGPIO_CLOCKS	24
6.2.6	FTM_CLOCKS	24
6.2.7	GPT_CLOCKS	24
6.2.8	FLEXCAN_CLOCKS	25
6.2.9	FLEXSPI_CLOCKS	25
6.2.10	LPUART_CLOCKS	25
6.2.11	LPADC_CLOCKS	25
6.2.12	INTMUX_CLOCKS	26
6.2.13	SAI_CLOCKS	26
6.2.14	SEMA42_CLOCKS	26
6.2.15	TPM_CLOCKS	26
6.2.16	LPIT_CLOCKS	27
6.2.17	LPI2C_CLOCKS	27
6.2.18	LPSPI_CLOCKS	27
6.2.19	EDMA_CLOCKS	28
6.2.20	ESAI_CLOCKS	28
6.2.21	ISI_CLOCKS	28
6.2.22	MIPI_CSI2RX_CLOCKS	28
6.2.23	MIPI_DSI_HOST_CLOCKS	28
6.2.24	ENET_CLOCKS	29
6.2.25	EMVSIM_CLOCKS	29
6.2.26	DPU_CLOCKS	29
6.2.27	LDB_CLOCKS	29
6.2.28	LPCG_TUPLE	29
6.2.29	LPCG_TUPLE_REG_BASE	30
6.2.30	LPCG_TUPLE_RSRC	30
6.2.31	NV	30
6.3	Enumeration Type Documentation	30
6.3.1	clock_ip_src_t	30
6.3.2	clock_name_t	30
6.3.3	clock_ip_name_t	30
6.4	Function Documentation	30
6.4.1	CLOCK_Init	30
6.4.2	CLOCK_EnableClockExt	30

Contents

Section Number	Title	Page Number
6.4.3	CLOCK_EnableClock	31
6.4.4	CLOCK_DisableClock	31
6.4.5	CLOCK_SetIpFreq	31
6.4.6	CLOCK_GetIpFreq	32
6.4.7	CLOCK_GetFreq	32
6.4.8	CLOCK_GetCoreSysClkFreq	32
6.4.9	CLOCK_ConfigLPCG	33
6.4.10	CLOCK_SetLpcgGate	34
Chapter Debug Console		
7.1	Overview	35
7.2	Function groups	35
7.2.1	Initialization	35
7.2.2	Advanced Feature	35
7.3	Typical use case	39
7.4	Macro Definition Documentation	41
7.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	41
7.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK	42
7.4.3	DEBUGCONSOLE_DISABLE	42
7.4.4	SDK_DEBUGCONSOLE	42
7.4.5	SDK_DEBUGCONSOLE_UART	42
7.4.6	PRINTF	42
7.5	Function Documentation	42
7.5.1	DbgConsole_Init	42
7.5.2	DbgConsole_Deinit	43
7.5.3	DbgConsole_Printf	43
7.5.4	DbgConsole_Putchar	43
7.5.5	DbgConsole_Scanf	44
7.5.6	DbgConsole_Getchar	44
7.5.7	DbgConsole_Flush	45
7.5.8	StrFormatPrintf	45
7.5.9	StrFormatScanf	45
7.6	Semihosting	47
7.6.1	Guide Semihosting for IAR	47
7.6.2	Guide Semihosting for Keil µVision	47
7.6.3	Guide Semihosting for MCUXpresso IDE	48
7.6.4	Guide Semihosting for ARMGCC	48
7.7	SWO	51
7.7.1	Guide SWO for SDK	51

Section Number	Title	Page Number
7.7.2	Guide SWO for Keil µVision	52
7.7.3	Guide SWO for MCUXpresso IDE	53
7.7.4	Guide SWO for ARMGCC	53
 Chapter Display Processing Unit (DPU)		
8.1	Overview	55
8.2	Program model	55
8.3	Path configuration	60
8.4	Data Structure Documentation	77
8.4.1	struct dpu_fetch_unit_config_t	77
8.4.2	struct dpu_coordinates_config_t	77
8.4.3	struct dpu_warp_config_t	78
8.4.4	struct dpu_src_buffer_config_t	79
8.4.5	struct dpu_clip_window_config_t	80
8.4.6	struct dpu_dst_buffer_config_t	81
8.4.7	struct dpu_layer_blend_config_t	82
8.4.8	struct dpu.blit_blend_config_t	83
8.4.9	struct dpu.rop_config_t	84
8.4.10	struct dpu.const_frame_config_t	84
8.4.11	struct dpu.display_timing_config_t	85
8.4.12	struct dpu.display_config_t	86
8.4.13	struct dpu.scaler_config_t	87
8.4.14	struct dpu.signature_config_t	88
8.4.15	struct dpu.signature_window_config_t	89
8.5	Macro Definition Documentation	89
8.5.1	FSL_DPU_DRIVER_VERSION	89
8.5.2	DPU_PALETTE_ENTRY_NUM	89
8.5.3	DPU_MAKE_SRC_REG1	89
8.5.4	DPU_MAKE_SRC_REG2	89
8.5.5	DPU_MAKE_SRC_REG3	89
8.5.6	DPU_MAKE_CONST_COLOR	90
8.5.7	DPU_FRAME_GEN_INT_DISABLE	90
8.5.8	DPU_FRAME_GEN_INT_PER_LINE	90
8.5.9	DPU_FRAME_GEN_INT_PER_FRAME	90
8.6	Enumeration Type Documentation	90
8.6.1	dpu_unit_t	90
8.6.2	_dpu_interrupt	90
8.6.3	_dpu_unit_source	91
8.6.4	dpu_pixel_format_t	92

Contents

Section Number	Title	Page Number
8.6.5	dpu_warp_coordinate_mode_t	92
8.6.6	dpu_clip_color_mode_t	93
8.6.7	dpu_alpha_mask_mode_t	93
8.6.8	dpu_blend_mode_t	93
8.6.9	dpu.blit_blend_func_t	93
8.6.10	dpu.blit_blend_mode_t	94
8.6.11	dpu.blit_blend_neutral_border_mode_t	94
8.6.12	_dpu_rop_flags	95
8.6.13	_dpu_display_timing_flags	95
8.6.14	dpu_display_mode_t	95
8.6.15	_dpu_signature_window_flags	95
8.6.16	_dpu_signature_status	96
8.7	Function Documentation	96
8.7.1	DPU_Init	96
8.7.2	DPU_Deinit	96
8.7.3	DPU_PreparePathConfig	96
8.7.4	DPU_EnableInterrupts	97
8.7.5	DPU_DisableInterrupts	97
8.7.6	DPU_GetInterruptsPendingFlags	98
8.7.7	DPU_ClearInterruptsPendingFlags	98
8.7.8	DPU_SetInterruptsPendingFlags	99
8.7.9	DPU_MaskUserInterrupts	99
8.7.10	DPU_EnableUserInterrupts	100
8.7.11	DPU_DisableUserInterrupts	100
8.7.12	DPU.GetUserInterruptsPendingFlags	100
8.7.13	DPU_ClearUserInterruptsPendingFlags	101
8.7.14	DPU_SetUserInterruptsPendingFlags	101
8.7.15	DPU_EnableShadowLoad	101
8.7.16	DPU_InitPipeline	102
8.7.17	DPU_DeinitPipeline	102
8.7.18	DPU_TriggerPipelineShadowLoad	102
8.7.19	DPU_TriggerPipelineCompleteInterrupt	103
8.7.20	DPU_SetUnitSrc	103
8.7.21	DPU_FetchUnitGetDefaultConfig	103
8.7.22	DPU_InitFetchUnit	104
8.7.23	DPU_SetColorPaletteIndexWidth	104
8.7.24	DPU_UpdateColorPalette	105
8.7.25	DPU_EnableColorPalette	105
8.7.26	DPU_CorrdinatesGetDefaultConfig	106
8.7.27	DPU_InitWarpCoordinates	106
8.7.28	DPU_FetcUnitGetDefaultWarpConfig	106
8.7.29	DPU_InitFetchUnitWarp	107
8.7.30	DPU_SrcBufferGetDefaultConfig	107
8.7.31	DPU_SetFetchUnitSrcBufferConfig	108

Contents

Section Number	Title	Page Number
8.7.32	DPU_SetFetchUnitSrcBufferAddr	108
8.7.33	DPU_SetFetchUnitFrameSize	108
8.7.34	DPU_SetFetchUnitOffset	109
8.7.35	DPU_EnableFetchUnitSrcBuffer	109
8.7.36	DPU_ClipWindowGetDefaultConfig	109
8.7.37	DPU_SetFetchUnitClipWindowConfig	110
8.7.38	DPU_EnableFetchUnitClipWindow	110
8.7.39	DPU_SetFetchUnitClipColor	110
8.7.40	DPU_InitExtDst	111
8.7.41	DPU_InitStore	111
8.7.42	DPU_SetStoreDstBufferConfig	112
8.7.43	DPU_DstBufferGetDefaultConfig	112
8.7.44	DPU_SetStoreDstBufferAddr	113
8.7.45	DPU_SetStoreOffset	113
8.7.46	DPU_StartStore	113
8.7.47	DPU_InitRop	114
8.7.48	DPU_RopGetDefaultConfig	115
8.7.49	DPU_SetRopConfig	115
8.7.50	DPU_EnableRop	115
8.7.51	DPU_InitBlitBlend	116
8.7.52	DPU_BlitBlendGetDefaultConfig	116
8.7.53	DPU_SetBlitBlendConfig	117
8.7.54	DPU_EnableBlitBlend	117
8.7.55	DPU_LayerBlendGetDefaultConfig	117
8.7.56	DPU_InitLayerBlend	118
8.7.57	DPU_SetLayerBlendConfig	119
8.7.58	DPU_EnableLayerBlend	119
8.7.59	DPU_InitConstFrame	119
8.7.60	DPU_ConstFrameGetDefaultConfig	120
8.7.61	DPU_SetConstFrameConfig	120
8.7.62	DPU_InitScaler	120
8.7.63	DPU_ScalerGetDefaultConfig	120
8.7.64	DPU_SetScalerConfig	121
8.7.65	DPU_DisplayTimingGetDefaultConfig	121
8.7.66	DPU_InitDisplayTiming	122
8.7.67	DPU_DisplayGetDefaultConfig	122
8.7.68	DPU_SetDisplayConfig	122
8.7.69	DPU_StartDisplay	123
8.7.70	DPU_StopDisplay	123
8.7.71	DPU_SetFrameGenInterruptConfig	123
8.7.72	DPU_TriggerDisplayShadowLoad	124
8.7.73	DPU_SignatureGetDefaultConfig	125
8.7.74	DPU_InitSignature	125
8.7.75	DPU_SignatureWindowGetDefaultConfig	125
8.7.76	DPU_SetSignatureWindowConfig	126

Contents

Section Number	Title	Page Number
8.7.77	DPU_EnableSignatureWindowCompute	126
8.7.78	DPU_EnableSignatureWindowCheck	126
8.7.79	DPU_GetSignatureWindowCrc	127
8.7.80	DPU_SetSignatureWindowRefCrc	127
8.7.81	DPU_GetSignatureStatus	128
8.7.82	DPU_TriggerSignatureShadowLoad	128
Chapter DPU IRQSTEER: Interrupt Request Steering Driver		
9.1	Overview	131
9.2	Function Documentation	131
9.2.1	DPU_IRQSTEER_EnableInterrupt	131
9.2.2	DPU_IRQSTEER_DisableInterrupt	131
9.2.3	DPU_IRQSTEER_IsInterruptSet	132
Chapter EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver		
10.1	Overview	133
10.2	Typical use case	133
10.2.1	EDMA Operation	133
10.3	Data Structure Documentation	138
10.3.1	struct edma_config_t	138
10.3.2	struct edma_transfer_config_t	139
10.3.3	struct edma_channel_Preemption_config_t	140
10.3.4	struct edma_minor_offset_config_t	140
10.3.5	struct edma_tcd_t	141
10.3.6	struct edma_handle_t	142
10.4	Macro Definition Documentation	143
10.4.1	FSL_EDMA_DRIVER_VERSION	143
10.5	Typedef Documentation	143
10.5.1	edma_callback	143
10.6	Enumeration Type Documentation	143
10.6.1	edma_transfer_size_t	143
10.6.2	edma_modulo_t	144
10.6.3	edma_bandwidth_t	144
10.6.4	edma_channel_link_type_t	145
10.6.5	_edma_channel_status_flags	145
10.6.6	_edma_error_status_flags	145
10.6.7	_edma_channel_sys_bus_info	145
10.6.8	edma_interrupt_enable_t	146

Contents

Section Number	Title	Page Number
10.6.9	edma_transfer_type_t	146
10.6.10	_edma_transfer_status	146
10.7	Function Documentation	146
10.7.1	EDMA_Init	146
10.7.2	EDMA_Deinit	146
10.7.3	EDMA_GetDefaultConfig	147
10.7.4	EDMA_EnableAllChannelLink	147
10.7.5	EDMA_ResetChannel	147
10.7.6	EDMA_SetTransferConfig	148
10.7.7	EDMA_SetMinorOffsetConfig	148
10.7.8	EDMA_SetChannelArbitrationGroup	149
10.7.9	EDMA_SetChannelPreemptionConfig	149
10.7.10	EDMA_GetChannelSystemBusInformation	149
10.7.11	EDMA_SetChannelLink	150
10.7.12	EDMA_SetBandWidth	150
10.7.13	EDMA_SetModulo	151
10.7.14	EDMA_EnableAsyncRequest	151
10.7.15	EDMA_EnableAutoStopRequest	151
10.7.16	EDMA_EnableChannelInterrupts	152
10.7.17	EDMA_DisableChannelInterrupts	152
10.7.18	EDMA_TcdReset	152
10.7.19	EDMA_TcdSetTransferConfig	153
10.7.20	EDMA_TcdSetMinorOffsetConfig	153
10.7.21	EDMA_TcdSetChannelLink	154
10.7.22	EDMA_TcdSetBandWidth	154
10.7.23	EDMA_TcdSetModulo	155
10.7.24	EDMA_TcdEnableAutoStopRequest	155
10.7.25	EDMA_TcdEnableInterrupts	155
10.7.26	EDMA_TcdDisableInterrupts	155
10.7.27	EDMA_EnableChannelRequest	156
10.7.28	EDMA_DisableChannelRequest	156
10.7.29	EDMA_TriggerChannelStart	156
10.7.30	EDMA_GetRemainingMajorLoopCount	157
10.7.31	EDMA_GetErrorStatusFlags	158
10.7.32	EDMA_GetChannelStatusFlags	158
10.7.33	EDMA_ClearChannelStatusFlags	159
10.7.34	EDMA_CreateHandle	159
10.7.35	EDMA_InstallTCDMemory	159
10.7.36	EDMA_SetCallback	160
10.7.37	EDMA_PreparesTransfer	160
10.7.38	EDMA_SubmitTransfer	161
10.7.39	EDMA_StartTransfer	161
10.7.40	EDMA_StopTransfer	161
10.7.41	EDMA_AbortTransfer	162

Contents

Section Number	Title	Page Number
10.7.42	EDMA_GetUnusedTCDNumber	162
10.7.43	EDMA_GetNextTCDAAddress	162
10.7.44	EDMA_HandleIRQ	163

Chapter ENET: Ethernet MAC Driver

11.1	Overview	165
11.2	Typical use case	166
11.2.1	ENET Initialization, receive, and transmit operations	166
11.3	Data Structure Documentation	176
11.3.1	struct enet_rx_bd_struct_t	176
11.3.2	struct enet_tx_bd_struct_t	177
11.3.3	struct enet_data_error_stats_t	178
11.3.4	struct enet_buffer_config_t	179
11.3.5	struct enet_ptp_time_t	181
11.3.6	struct enet_ptp_time_data_t	181
11.3.7	struct enet_ptp_time_data_ring_t	182
11.3.8	struct enet_ptp_config_t	182
11.3.9	struct enet_intcoalesce_config_t	183
11.3.10	struct enet_avb_config_t	183
11.3.11	struct enet_config_t	184
11.3.12	struct _enet_handle	186
11.4	Macro Definition Documentation	187
11.4.1	FSL_ENET_DRIVER_VERSION	187
11.4.2	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	189
11.4.3	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	189
11.4.4	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	189
11.4.5	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	189
11.4.6	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	189
11.4.7	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	189
11.4.8	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	189
11.4.9	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	189
11.4.10	ENET_BUFFDESCRIPTOR_RX_LENVLIOLATE_MASK	189
11.4.11	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	189
11.4.12	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	189
11.4.13	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	189
11.4.14	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	189
11.4.15	ENET_BUFFDESCRIPTOR_TX_READY_MASK	189
11.4.16	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK	189
11.4.17	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	189
11.4.18	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK	189
11.4.19	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	189

Contents

Section Number	Title	Page Number
11.4.20	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	189
11.4.21	ENET_BUFFDESCRIPTOR_RX_IPV4_MASK	189
11.4.22	ENET_BUFFDESCRIPTOR_RX_IPV6_MASK	189
11.4.23	ENET_BUFFDESCRIPTOR_RX_VLAN_MASK	189
11.4.24	ENET_BUFFDESCRIPTOR_RX_PROTOCOLCHECKSUM_MASK	189
11.4.25	ENET_BUFFDESCRIPTOR_RX_IPHEADCHECKSUM_MASK	189
11.4.26	ENET_BUFFDESCRIPTOR_RX_INTERRUPT_MASK	189
11.4.27	ENET_BUFFDESCRIPTOR_RX_UNICAST_MASK	189
11.4.28	ENET_BUFFDESCRIPTOR_RX_COLLISION_MASK	189
11.4.29	ENET_BUFFDESCRIPTOR_RX_PHYERR_MASK	189
11.4.30	ENET_BUFFDESCRIPTOR_RX_MACERR_MASK	189
11.4.31	ENET_BUFFDESCRIPTOR_TX_ERR_MASK	189
11.4.32	ENET_BUFFDESCRIPTOR_TX_UNDERFLOWERR_MASK	189
11.4.33	ENET_BUFFDESCRIPTOR_TX_EXCCOLLISIONERR_MASK	189
11.4.34	ENET_BUFFDESCRIPTOR_TX_FRAMEERR_MASK	189
11.4.35	ENET_BUFFDESCRIPTOR_TX_LATECOLLISIONERR_MASK	189
11.4.36	ENET_BUFFDESCRIPTOR_TX_OVERFLOWERR_MASK	189
11.4.37	ENET_BUFFDESCRIPTOR_TX_TIMESTAMPERR_MASK	189
11.4.38	ENET_BUFFDESCRIPTOR_TX_INTERRUPT_MASK	189
11.4.39	ENET_BUFFDESCRIPTOR_TX_TIMESTAMP_MASK	189
11.4.40	ENET_BUFFDESCRIPTOR_TX_USETXLAUNCHTIME_MASK	189
11.4.41	ENET_BUFFDESCRIPTOR_TX_FRAMETYPE_MASK	189
11.4.42	ENET_BUFFDESCRIPTOR_TX_FRAMETYPE_SHIFT	189
11.4.43	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	189
11.4.44	ENET_FRAME_MAX_FRAMELEN	190
11.4.45	ENET_FIFO_MIN_RX_FULL	190
11.4.46	ENET_RX_MIN_BUFSIZE	190
11.5	Typedef Documentation	190
11.5.1	enet_callback_t	190
11.6	Enumeration Type Documentation	190
11.6.1	_enet_status	190
11.6.2	enet_mii_mode_t	190
11.6.3	enet_mii_speed_t	191
11.6.4	enet_mii_duplex_t	191
11.6.5	enet_mii_write_t	191
11.6.6	enet_mii_read_t	191
11.6.7	enet_mii_extend_opcode	191
11.6.8	enet_special_control_flag_t	192
11.6.9	enet_interrupt_enable_t	192
11.6.10	enet_event_t	193
11.6.11	enet_idle_slope_t	193
11.6.12	enet_tx_accelerator_t	194
11.6.13	enet_rx_accelerator_t	194

Contents

Section Number	Title	Page Number
11.6.14	enet_ptp_event_type_t	194
11.6.15	enet_ptp_timer_channel_t	194
11.6.16	enet_ptp_timer_channel_mode_t	195
11.7	Function Documentation	195
11.7.1	ENET_GetDefaultConfig	195
11.7.2	ENET_Init	195
11.7.3	ENET_Deinit	196
11.7.4	ENET_Reset	196
11.7.5	ENET_SetMII	197
11.7.6	ENET_SetSMI	197
11.7.7	ENET_GetSMI	197
11.7.8	ENET_ReadSMIData	198
11.7.9	ENET_StartSMIRead	198
11.7.10	ENET_StartSMIWrite	198
11.7.11	ENET_StartExtC45SMIRead	199
11.7.12	ENET_StartExtC45SMIWrite	199
11.7.13	ENET_SetRGMIIClockDelay	199
11.7.14	ENET_SetMacAddr	200
11.7.15	ENET_GetMacAddr	200
11.7.16	ENET_AddMulticastGroup	200
11.7.17	ENET_LeaveMulticastGroup	200
11.7.18	ENET_AVBConfigure	201
11.7.19	ENET_ActiveRead	201
11.7.20	ENET_ActiveReadMultiRing	201
11.7.21	ENET_EnableSleepMode	202
11.7.22	ENET_GetAccelFunction	202
11.7.23	ENET_EnableInterrupts	202
11.7.24	ENET_DisableInterrupts	204
11.7.25	ENET_GetInterruptStatus	204
11.7.26	ENET_ClearInterruptStatus	204
11.7.27	ENET_SetCallback	205
11.7.28	ENET_GetRxErrBeforeReadFrame	205
11.7.29	ENET_GetTxErrAfterSendFrame	206
11.7.30	ENET_GetRxFrameSize	206
11.7.31	ENET_ReadFrame	207
11.7.32	ENET_SendFrame	208
11.7.33	ENET_GetRxErrBeforeReadFrameMultiRing	208
11.7.34	ENET_SendFrameMultiRing	209
11.7.35	ENET_GetTxErrAfterSendFrameMultiRing	209
11.7.36	ENET_GetRxFrameSizeMultiRing	210
11.7.37	ENET_ReadFrameMultiRing	211
11.7.38	ENET_TransmitIRQHandler	211
11.7.39	ENET_ReceiveIRQHandler	211
11.7.40	ENET_CommonFrame1IRQHandler	212

Contents

Section Number	Title	Page Number
11.7.41	ENET_CommonFrame2IRQHandler	213
11.7.42	ENET_ErrorIRQHandler	213
11.7.43	ENET_CommonFrame0IRQHandler	213
11.7.44	ENET_Ptp1588Configure	213
11.7.45	ENET_Ptp1588StartTimer	214
11.7.46	ENET_Ptp1588StopTimer	214
11.7.47	ENET_Ptp1588AdjustTimer	215
11.7.48	ENET_Ptp1588SetChannelMode	215
11.7.49	ENET_Ptp1588SetChannelCmpValue	215
11.7.50	ENET_Ptp1588GetChannelStatus	216
11.7.51	ENET_Ptp1588ClearChannelStatus	217
11.7.52	ENET_Ptp1588GetTimer	217
11.7.53	ENET_Ptp1588SetTimer	217
11.7.54	ENET_Ptp1588TimerIRQHandler	218
11.7.55	ENET_GetRxFrameTime	219
11.7.56	ENET_GetTxFrameTime	219

Chapter **ESAI: Enhanced Serial Audio Interface**

12.1	Overview	221
12.2	Typical use case	221
12.2.1	ESAI Send/Receive using an interrupt method	221
12.2.2	ESAI Send/receive using a DMA method	221
12.3	Data Structure Documentation	227
12.3.1	struct esai_customer_protocol_t	227
12.3.2	struct esai_config_t	228
12.3.3	struct esai_format_t	229
12.3.4	struct esai_transfer_t	229
12.3.5	struct _esai_handle	229
12.4	Macro Definition Documentation	230
12.4.1	ESAI_XFER_QUEUE_SIZE	230
12.5	Enumeration Type Documentation	230
12.5.1	_esai_status_t	230
12.5.2	esai_mode_t	230
12.5.3	esai_protocol_t	230
12.5.4	esai_master_slave_t	231
12.5.5	esai_sync_mode_t	231
12.5.6	esai_clock_polarity_t	231
12.5.7	esai_shift_direction_t	231
12.5.8	esai_clock_direction_t	232
12.5.9	_esai_interrupt_enable_t	232

Contents

Section Number	Title	Page Number
12.5.10	_esai_flags	232
12.5.11	_esai_sai_flags	232
12.5.12	esai_sample_rate_t	233
12.5.13	esai_word_width_t	233
12.5.14	esai_slot_format_t	233
12.6	Function Documentation	234
12.6.1	ESAI_Init	234
12.6.2	ESAI_GetDefaultConfig	234
12.6.3	ESAI_Deinit	235
12.6.4	ESAI_Enable	235
12.6.5	ESAI_Reset	235
12.6.6	ESAI_TxReset	235
12.6.7	ESAI_RxReset	236
12.6.8	ESAI_TxResetFIFO	236
12.6.9	ESAI_RxResetFIFO	236
12.6.10	ESAI_TxEnable	236
12.6.11	ESAI_RxEnable	237
12.6.12	ESAI_TxEnableFIFO	237
12.6.13	ESAI_RxEnableFIFO	237
12.6.14	ESAI_TxSetSlotMask	237
12.6.15	EASI_RxSetSlotMask	238
12.6.16	ESAI_GetStatusFlag	238
12.6.17	ESAI_GetSAIStatusFlag	238
12.6.18	ESAI_GetTxFIFOStatus	238
12.6.19	ESAI_GetRxFIFOStatus	239
12.6.20	ESAI_TxEnableInterrupts	239
12.6.21	ESAI_RxEnableInterrupts	239
12.6.22	ESAI_TxDisableInterrupts	240
12.6.23	ESAI_RxDisableInterrupts	240
12.6.24	ESAI_TxGetDataRegisterAddress	240
12.6.25	ESAI_RxGetDataRegisterAddress	241
12.6.26	ESAI_TxSetFormat	242
12.6.27	ESAI_RxSetFormat	242
12.6.28	ESAI_WriteBlocking	242
12.6.29	ESAI_WriteData	243
12.6.30	ESAI_ReadBlocking	243
12.6.31	ESAI_ReadData	243
12.6.32	ESAI_TransferTxCreateHandle	244
12.6.33	ESAI_TransferRxCreateHandle	244
12.6.34	ESAI_TransferTxSetFormat	245
12.6.35	ESAI_TransferRxSetFormat	246
12.6.36	ESAI_TransferSendNonBlocking	246
12.6.37	ESAI_TransferReceiveNonBlocking	247
12.6.38	ESAI_TransferGetSendCount	247

Section Number	Title	Page Number
12.6.39	ESAI_TransferGetReceiveCount	248
12.6.40	ESAI_TransferAbortSend	248
12.6.41	ESAI_TransferAbortReceive	249
12.6.42	ESAI_TransferTxHandleIRQ	249
12.6.43	ESAI_TransferRxHandleIRQ	249
12.7	ESAI eDMA Driver	250
12.7.1	Overview	250
12.7.2	Data Structure Documentation	251
12.7.3	Function Documentation	252
 Chapter FlexCAN: Flex Controller Area Network Driver		
13.1	Overview	259
13.2	FlexCAN Driver	260
13.2.1	Overview	260
13.2.2	Typical use case	260
13.2.3	Data Structure Documentation	270
13.2.4	Macro Definition Documentation	274
13.2.5	Typedef Documentation	279
13.2.6	Enumeration Type Documentation	280
13.2.7	Function Documentation	283
13.3	FlexCAN eDMA Driver	304
13.3.1	Overview	304
13.3.2	Data Structure Documentation	304
13.3.3	Macro Definition Documentation	305
13.3.4	Typedef Documentation	305
13.3.5	Function Documentation	305
 Chapter FLEXSPI: Flexible Serial Peripheral Interface Driver		
14.1	Overview	307
14.2	Data Structure Documentation	312
14.2.1	struct flexspi_config_t	312
14.2.2	struct flexspi_device_config_t	315
14.2.3	struct flexspi_transfer_t	316
14.2.4	struct _flexspi_handle	317
14.3	Macro Definition Documentation	318
14.3.1	FSL_FLEXSPI_DRIVER_VERSION	318
14.3.2	FLEXSPI_LUT_SEQ	318
14.4	Typedef Documentation	318

Section Number	Title	Page Number
14.4.1	flexspi_transfer_callback_t	318
14.5	Enumeration Type Documentation	318
14.5.1	_flexspi_status	318
14.5.2	_flexspi_command	318
14.5.3	_flexspi_pad	319
14.5.4	flexspi_flags_t	319
14.5.5	flexspi_read_sample_clock_t	320
14.5.6	flexspi_cs_interval_cycle_unit_t	320
14.5.7	flexspi_ahb_write_wait_unit_t	321
14.5.8	flexspi_ip_error_code_t	321
14.5.9	flexspi_ahb_error_code_t	321
14.5.10	flexspi_port_t	322
14.5.11	flexspi_arb_command_source_t	322
14.5.12	flexspi_command_type_t	322
14.6	Function Documentation	322
14.6.1	FLEXSPI_Init	322
14.6.2	FLEXSPI_GetDefaultConfig	322
14.6.3	FLEXSPI_Deinit	322
14.6.4	FLEXSPI_SetFlashConfig	323
14.6.5	FLEXSPI_SoftwareReset	323
14.6.6	FLEXSPI_Enable	323
14.6.7	FLEXSPI_EnableInterrupts	323
14.6.8	FLEXSPI_DisableInterrupts	324
14.6.9	FLEXSPI_EnableTxDMA	324
14.6.10	FLEXSPI_EnableRxDMA	324
14.6.11	FLEXSPI_GetTxFifoAddress	324
14.6.12	FLEXSPI_GetRxFifoAddress	325
14.6.13	FLEXSPI_ResetFifos	325
14.6.14	FLEXSPI_GetFifoCounts	325
14.6.15	FLEXSPI_GetInterruptStatusFlags	326
14.6.16	FLEXSPI_ClearInterruptStatusFlags	326
14.6.17	FLEXSPI_GetDataLearningPhase	326
14.6.18	FLEXSPI_GetArbitratorCommandSource	326
14.6.19	FLEXSPI_GetIPCommandErrorCode	327
14.6.20	FLEXSPI_GetAHBCommandErrorCode	327
14.6.21	FLEXSPI_GetBusIdleStatus	327
14.6.22	FLEXSPI_EnableIPParallelMode	328
14.6.23	FLEXSPI_EnableAHBParallelMode	328
14.6.24	FLEXSPI_UpdateLUT	328
14.6.25	FLEXSPI_WriteData	329
14.6.26	FLEXSPI_ReadData	330
14.6.27	FLEXSPI_WriteBlocking	330
14.6.28	FLEXSPI_ReadBlocking	331

Section Number	Title	Page Number
14.6.29	FLEXSPI_TransferBlocking	331
14.6.30	FLEXSPI_TransferCreateHandle	332
14.6.31	FLEXSPI_TransferNonBlocking	332
14.6.32	FLEXSPI_TransferGetCount	333
14.6.33	FLEXSPI_TransferAbort	333
14.6.34	FLEXSPI_TransferHandleIRQ	334
14.7	FLEXSPI eDMA Driver	335
14.7.1	Overview	335
14.7.2	Data Structure Documentation	336
14.7.3	Macro Definition Documentation	337
14.7.4	Enumeration Type Documentation	337
14.7.5	Function Documentation	337
 Chapter FTM: FlexTimer Driver		
15.1	Overview	341
15.2	Function groups	341
15.2.1	Initialization and deinitialization	341
15.2.2	PWM Operations	341
15.2.3	Input capture operations	341
15.2.4	Output compare operations	342
15.2.5	Quad decode	342
15.2.6	Fault operation	342
15.3	Register Update	342
15.4	Typical use case	343
15.4.1	PWM output	343
15.5	Data Structure Documentation	349
15.5.1	struct ftm_chnl_pwm_signal_param_t	349
15.5.2	struct ftm_chnl_pwm_config_param_t	350
15.5.3	struct ftm_dual_edge_capture_param_t	350
15.5.4	struct ftm_phase_params_t	350
15.5.5	struct ftm_fault_param_t	351
15.5.6	struct ftm_config_t	351
15.6	Macro Definition Documentation	352
15.6.1	FSL_FTM_DRIVER_VERSION	352
15.7	Enumeration Type Documentation	352
15.7.1	ftm_chnl_t	352
15.7.2	ftm_fault_input_t	353
15.7.3	ftm_pwm_mode_t	353

Contents

Section Number	Title	Page Number
15.7.4	ftm_pwm_level_select_t	353
15.7.5	ftm_output_compare_mode_t	353
15.7.6	ftm_input_capture_edge_t	353
15.7.7	ftm_dual_edge_capture_mode_t	354
15.7.8	ftm_quad_decode_mode_t	354
15.7.9	ftm_phase_polarity_t	354
15.7.10	ftm_deadtime_prescale_t	354
15.7.11	ftm_clock_source_t	354
15.7.12	ftm_clock_prescale_t	355
15.7.13	ftm_bdm_mode_t	355
15.7.14	ftm_fault_mode_t	355
15.7.15	ftm_external_trigger_t	355
15.7.16	ftm_pwm_sync_method_t	356
15.7.17	ftm_reload_point_t	356
15.7.18	ftm_interrupt_enable_t	357
15.7.19	ftm_status_flags_t	357
15.7.20	_ftm_quad_decoder_flags	358
15.8	Function Documentation	358
15.8.1	FTM_Init	358
15.8.2	FTM_Deinit	358
15.8.3	FTM_GetDefaultConfig	358
15.8.4	FTM_SetupPwm	359
15.8.5	FTM_UpdatePwmDutyCycle	359
15.8.6	FTM_UpdateChnlEdgeLevelSelect	360
15.8.7	FTM_SetupPwmMode	360
15.8.8	FTM_SetupInputCapture	361
15.8.9	FTM_SetupOutputCompare	361
15.8.10	FTM_SetupDualEdgeCapture	361
15.8.11	FTM_SetupFault	362
15.8.12	FTM_EnableInterrupts	362
15.8.13	FTM_DisableInterrupts	362
15.8.14	FTM_GetEnabledInterrupts	363
15.8.15	FTM_GetStatusFlags	364
15.8.16	FTM_ClearStatusFlags	364
15.8.17	FTM_SetTimerPeriod	364
15.8.18	FTM_GetCurrentTimerCount	365
15.8.19	FTM_StartTimer	365
15.8.20	FTM_StopTimer	365
15.8.21	FTM_SetSoftwareCtrlEnable	365
15.8.22	FTM_SetSoftwareCtrlVal	366
15.8.23	FTM_SetGlobalTimeBaseOutputEnable	366
15.8.24	FTM_SetOutputMask	366
15.8.25	FTM_SetPwmOutputEnable	367
15.8.26	FTM_SetFaultControlEnable	368

Contents

Section Number	Title	Page Number
15.8.27	FTM_SetDeadTimeEnable	368
15.8.28	FTM_SetComplementaryEnable	368
15.8.29	FTM_SetInvertEnable	369
15.8.30	FTM_SetupQuadDecode	369
15.8.31	FTM_GetQuadDecoderFlags	369
15.8.32	FTM_SetQuadDecoderModuloValue	370
15.8.33	FTM_GetQuadDecoderCounterValue	371
15.8.34	FTM_ClearQuadDecoderCounterValue	371
15.8.35	FTM_SetSoftwareTrigger	371
15.8.36	FTM_SetWriteProtection	371
Chapter	GPT: General Purpose Timer	
16.1	Overview	373
16.2	Function groups	373
16.2.1	Initialization and deinitialization	373
16.3	Typical use case	373
16.3.1	GPT interrupt example	373
16.4	Data Structure Documentation	376
16.4.1	struct gpt_config_t	376
16.5	Enumeration Type Documentation	377
16.5.1	gpt_clock_source_t	377
16.5.2	gpt_input_capture_channel_t	377
16.5.3	gpt_input_operation_mode_t	378
16.5.4	gpt_output_compare_channel_t	378
16.5.5	gpt_output_operation_mode_t	378
16.5.6	gpt_interrupt_enable_t	378
16.5.7	gpt_status_flag_t	379
16.6	Function Documentation	379
16.6.1	GPT_Init	379
16.6.2	GPT_Deinit	379
16.6.3	GPT_GetDefaultConfig	379
16.6.4	GPT_SoftwareReset	380
16.6.5	GPT_SetClockSource	380
16.6.6	GPT_GetClockSource	380
16.6.7	GPT_SetClockDivider	380
16.6.8	GPT_GetClockDivider	381
16.6.9	GPT_SetOscClockDivider	381
16.6.10	GPT_GetOscClockDivider	381
16.6.11	GPT_StartTimer	381

Section Number	Title	Page Number
16.6.12	GPT_StopTimer	382
16.6.13	GPT_GetCurrentTimerCount	382
16.6.14	GPT_SetInputOperationMode	382
16.6.15	GPT_GetInputOperationMode	382
16.6.16	GPT_GetInputCaptureValue	383
16.6.17	GPT_SetOutputOperationMode	383
16.6.18	GPT_GetOutputOperationMode	384
16.6.19	GPT_SetOutputCompareValue	384
16.6.20	GPT_GetOutputCompareValue	384
16.6.21	GPT_ForceOutput	385
16.6.22	GPT_EnableInterrupts	385
16.6.23	GPT_DisableInterrupts	385
16.6.24	GPT_GetEnabledInterrupts	385
16.6.25	GPT_GetStatusFlags	386
16.6.26	GPT_ClearStatusFlags	386

Chapter GPIO: General-Purpose Input/Output Driver

17.1	Overview	387
17.2	GPIO Driver	388
17.2.1	Overview	388
17.2.2	Typical use case	388
17.2.3	Data Structure Documentation	390
17.2.4	Macro Definition Documentation	390
17.2.5	Enumeration Type Documentation	390
17.2.6	Function Documentation	391

Chapter INTMUX: Interrupt Multiplexer Driver

18.1	Overview	397
18.2	Typical use case	397
18.2.1	Channel Configure	397
18.3	Macro Definition Documentation	398
18.3.1	FSL_INTMUX_DRIVER_VERSION	398
18.4	Enumeration Type Documentation	398
18.4.1	intmux_channel_logic_mode_t	398
18.5	Function Documentation	398
18.5.1	INTMUX_Init	398
18.5.2	INTMUX_Deinit	398
18.5.3	INTMUX_ResetChannel	398

Contents

Section Number	Title	Page Number
18.5.4	INTMUX_SetChannelMode	399
18.5.5	INTMUX_EnableInterrupt	399
18.5.6	INTMUX_DisableInterrupt	399
18.5.7	INTMUX_GetChannelPendingSources	400
 Chapter IRQSTEER: Interrupt Request Steering Driver		
19.1	Overview	401
19.2	Macro Definition Documentation	403
19.2.1	FSL_IRQSTEER_DRIVER_VERSION	403
19.2.2	IRQSTEER_INT_SRC_REG_WIDTH	403
19.2.3	IRQSTEER_INT_SRC_REG_INDEX	403
19.2.4	IRQSTEER_INT_SRC_BIT_OFFSET	403
19.2.5	IRQSTEER_INT_SRC_NUM	403
19.3	Enumeration Type Documentation	403
19.3.1	irqsteer_int_group_t	403
19.3.2	irqsteer_int_master_t	404
19.4	Function Documentation	404
19.4.1	IRQSTEER_Init	404
19.4.2	IRQSTEER_Deinit	404
19.4.3	IRQSTEER_EnableInterrupt	404
19.4.4	IRQSTEER_DisableInterrupt	405
19.4.5	IRQSTEER_SetInterrupt	405
19.4.6	IRQSTEER_EnableMasterInterrupt	405
19.4.7	IRQSTEER_DisableMasterInterrupt	406
19.4.8	IRQSTEER_IsInterruptSet	406
19.4.9	IRQSTEER_IsMasterInterruptSet	407
19.4.10	IRQSTEER_GetGroupInterruptStatus	407
19.4.11	IRQSTEER_GetMasterNextInterrupt	407
 Chapter ISI: Image Sensing Interface		
20.1	Overview	409
20.2	Typical use case	409
20.2.1	Output buffer	409
20.2.2	Output panic and overflow	411
20.3	Data Structure Documentation	417
20.3.1	struct isi_config_t	417
20.3.2	struct isi_esc_config_t	418
20.3.3	struct isi_crop_config_t	420

Contents

Section Number	Title	Page Number
20.3.4	<code>struct isi_region_alpha_config_t</code>	421
20.3.5	<code>struct isi_input_mem_config_t</code>	421
20.4	Macro Definition Documentation	422
20.4.1	<code>FSL_ISI_DRIVER_VERSION</code>	422
20.5	Enumeration Type Documentation	422
20.5.1	<code>_isi_interrupt</code>	422
20.5.2	<code>isi_output_format_t</code>	422
20.5.3	<code>isi_chain_mode_t</code>	424
20.5.4	<code>isi_deint_mode_t</code>	424
20.5.5	<code>isi_threshold_t</code>	425
20.5.6	<code>isi_csc_mode_t</code>	425
20.5.7	<code>isi_flip_mode_t</code>	425
20.5.8	<code>isi_input_mem_format_t</code>	425
20.6	Function Documentation	426
20.6.1	<code>ISI_Init</code>	426
20.6.2	<code>ISI_Deinit</code>	426
20.6.3	<code>ISI_Reset</code>	427
20.6.4	<code>ISI_EnableInterrupts</code>	427
20.6.5	<code>ISI_DisableInterrupts</code>	427
20.6.6	<code>ISI_GetInterruptStatus</code>	428
20.6.7	<code>ISI_ClearInterruptStatus</code>	428
20.6.8	<code>ISI_GetOverflowBytes</code>	428
20.6.9	<code>ISI_SetConfig</code>	429
20.6.10	<code>ISI_GetDefaultConfig</code>	429
20.6.11	<code>ISI_SetScalerConfig</code>	430
20.6.12	<code>ISI_SetColorSpaceConversionConfig</code>	430
20.6.13	<code>ISI_ColorSpaceConversionGetDefaultConfig</code>	430
20.6.14	<code>ISI_EnableColorSpaceConversion</code>	431
20.6.15	<code>ISI_SetCropConfig</code>	431
20.6.16	<code>ISI_CropGetDefaultConfig</code>	432
20.6.17	<code>ISI_EnableCrop</code>	432
20.6.18	<code>ISI_SetGlobalAlpha</code>	432
20.6.19	<code>ISI_EnableGlobalAlpha</code>	433
20.6.20	<code>ISI_SetRegionAlphaConfig</code>	433
20.6.21	<code>ISI_RegionAlphaGetDefaultConfig</code>	433
20.6.22	<code>ISI_EnableRegionAlpha</code>	435
20.6.23	<code>ISI_SetInputMemConfig</code>	435
20.6.24	<code>ISI_InputMemGetDefaultConfig</code>	435
20.6.25	<code>ISI_SetInputMemAddr</code>	436
20.6.26	<code>ISI_TriggerInputMemRead</code>	436
20.6.27	<code>ISI_SetFlipMode</code>	436
20.6.28	<code>ISI_SetOutputBufferAddr</code>	436

Contents

Section Number	Title	Page Number
20.6.29	ISI_Start	437
20.6.30	ISI_Stop	437
Chapter	LDB: LVDS Display Bridge	
21.1	Overview	439
21.2	Data Structure Documentation	440
21.2.1	struct ldb_channel_config_t	440
21.3	Macro Definition Documentation	440
21.3.1	FSL_LDB_DRIVER_VERSION	440
21.4	Enumeration Type Documentation	440
21.4.1	ldb_output_bus_t	440
21.4.2	_ldb_input_flag	440
21.5	Function Documentation	440
21.5.1	LDB_Init	440
21.5.2	LDB_Deinit	441
21.5.3	LDB_InitChannel	441
21.5.4	LDB_DeinitChannel	441
21.6	LDB Driver	442
Chapter	LPADC: 12-bit SAR Analog-to-Digital Converter Driver	
22.1	Overview	443
22.2	Typical use case	443
22.2.1	Polling Configuration	443
22.2.2	Interrupt Configuration	443
22.3	Data Structure Documentation	446
22.3.1	struct lpadc_config_t	446
22.3.2	struct lpadc_conv_command_config_t	448
22.3.3	struct lpadc_conv_trigger_config_t	449
22.3.4	struct lpadc_conv_result_t	450
22.4	Macro Definition Documentation	450
22.4.1	FSL_LPADC_DRIVER_VERSION	450
22.4.2	LPADC_GET_ACTIVE_COMMAND_STATUS	450
22.4.3	LPADC_GET_ACTIVE_TRIGGER_STATUE	450
22.5	Enumeration Type Documentation	450
22.5.1	_lpadc_status_flags	450

Contents

Section Number	Title	Page Number
22.5.2	<code>_lpadc_interrupt_enable</code>	451
22.5.3	<code>lpadc_sample_scale_mode_t</code>	451
22.5.4	<code>lpadc_sample_channel_mode_t</code>	451
22.5.5	<code>lpadc_hardware_average_mode_t</code>	451
22.5.6	<code>lpadc_sample_time_mode_t</code>	452
22.5.7	<code>lpadc_hardware_compare_mode_t</code>	452
22.5.8	<code>lpadc_conversion_resolution_mode_t</code>	452
22.5.9	<code>lpadc_reference_voltage_source_t</code>	453
22.5.10	<code>lpadc_power_level_mode_t</code>	453
22.5.11	<code>lpadc_trigger_priority_policy_t</code>	453
22.6	Function Documentation	454
22.6.1	<code>LPADC_Init</code>	454
22.6.2	<code>LPADC_GetDefaultConfig</code>	454
22.6.3	<code>LPADC_Deinit</code>	454
22.6.4	<code>LPADC_Enable</code>	454
22.6.5	<code>LPADC_DoResetFIFO</code>	455
22.6.6	<code>LPADC_DoResetConfig</code>	455
22.6.7	<code>LPADC_GetStatusFlags</code>	455
22.6.8	<code>LPADC_ClearStatusFlags</code>	455
22.6.9	<code>LPADC_EnableInterrupts</code>	456
22.6.10	<code>LPADC_DisableInterrupts</code>	456
22.6.11	<code>LPADC_EnableFIFOWatermarkDMA</code>	456
22.6.12	<code>LPADC_GetConvResultCount</code>	456
22.6.13	<code>LPADC_GetConvResult</code>	457
22.6.14	<code>LPADC_SetConvTriggerConfig</code>	457
22.6.15	<code>LPADC_GetDefaultConvTriggerConfig</code>	457
22.6.16	<code>LPADC_DoSoftwareTrigger</code>	458
22.6.17	<code>LPADC_SetConvCommandConfig</code>	458
22.6.18	<code>LPADC_GetDefaultConvCommandConfig</code>	458

Chapter LPI2C: Low Power I2C Driver

23.1	Overview	461
23.2	Macro Definition Documentation	461
23.2.1	<code>FSL_LPI2C_DRIVER_VERSION</code>	461
23.2.2	<code>LPI2C_WAIT_TIMEOUT</code>	461
23.3	Enumeration Type Documentation	462
23.3.1	<code>_lpi2c_status</code>	462
23.4	LPI2C Master Driver	463
23.4.1	<code>Overview</code>	463
23.4.2	<code>Data Structure Documentation</code>	466

Section Number	Contents	Page Number
	Title	
23.4.3	Typedef Documentation	470
23.4.4	Enumeration Type Documentation	470
23.4.5	Function Documentation	473
23.5	LPI2C Slave Driver	487
23.5.1	Overview	487
23.5.2	Data Structure Documentation	489
23.5.3	Typedef Documentation	493
23.5.4	Enumeration Type Documentation	494
23.5.5	Function Documentation	495
23.6	LPI2C Master DMA Driver	504
23.6.1	Overview	504
23.6.2	Data Structure Documentation	504
23.6.3	Typedef Documentation	506
23.6.4	Function Documentation	507
23.7	LPI2C FreeRTOS Driver	510
23.7.1	Overview	510
23.7.2	Macro Definition Documentation	510
23.7.3	Function Documentation	510
 Chapter LPIT: Low-Power Interrupt Timer		
24.1	Overview	513
24.2	Function groups	513
24.2.1	Initialization and deinitialization	513
24.2.2	Timer period Operations	513
24.2.3	Start and Stop timer operations	513
24.2.4	Status	514
24.2.5	Interrupt	514
24.3	Typical use case	514
24.3.1	LPIT tick example	514
24.4	Data Structure Documentation	516
24.4.1	struct lpit_chnl_params_t	516
24.4.2	struct lpit_config_t	517
24.5	Enumeration Type Documentation	517
24.5.1	lpit_chnl_t	517
24.5.2	lpit_timer_modes_t	517
24.5.3	lpit_trigger_select_t	518
24.5.4	lpit_trigger_source_t	518
24.5.5	lpit_interrupt_enable_t	518

Section Number	Title	Page Number
24.5.6	lpit_status_flags_t	519
24.6	Function Documentation	519
24.6.1	LPIT_Init	519
24.6.2	LPIT_Deinit	519
24.6.3	LPIT_GetDefaultConfig	519
24.6.4	LPIT_SetupChannel	520
24.6.5	LPIT_EnableInterrupts	520
24.6.6	LPIT_DisableInterrupts	520
24.6.7	LPIT_GetEnabledInterrupts	521
24.6.8	LPIT_GetStatusFlags	522
24.6.9	LPIT_ClearStatusFlags	522
24.6.10	LPIT_SetTimerPeriod	522
24.6.11	LPIT_GetCurrentTimerCount	523
24.6.12	LPIT_StartTimer	523
24.6.13	LPIT_StopTimer	523
24.6.14	LPIT_Reset	524

Chapter LPSPI: Low Power Serial Peripheral Interface

25.1	Overview	525
25.2	LPSPI Peripheral driver	526
25.2.1	Overview	526
25.2.2	Function groups	526
25.2.3	Typical use case	526
25.2.4	Data Structure Documentation	533
25.2.5	Macro Definition Documentation	539
25.2.6	Typedef Documentation	540
25.2.7	Enumeration Type Documentation	541
25.2.8	Function Documentation	546
25.2.9	Variable Documentation	561
25.3	LPSPI eDMA Driver	562
25.3.1	Overview	562
25.3.2	Data Structure Documentation	563
25.3.3	Macro Definition Documentation	568
25.3.4	Typedef Documentation	568
25.3.5	Function Documentation	569
25.4	LPSPI FreeRTOS Driver	574
25.4.1	Overview	574
25.4.2	Macro Definition Documentation	574
25.4.3	Function Documentation	574

Contents

Section Number	Title	Page Number
Chapter	LPUART: Low Power UART Driver	
26.1	Overview	577
26.2	LPUART Driver	578
26.2.1	Overview	578
26.2.2	Typical use case	578
26.2.3	Data Structure Documentation	583
26.2.4	Macro Definition Documentation	586
26.2.5	Typedef Documentation	586
26.2.6	Enumeration Type Documentation	586
26.2.7	Function Documentation	590
26.3	LPUART DMA Driver	604
26.4	LPUART eDMA Driver	605
26.4.1	Overview	605
26.4.2	Data Structure Documentation	606
26.4.3	Macro Definition Documentation	607
26.4.4	Typedef Documentation	607
26.4.5	Function Documentation	607
26.5	LPUART FreeRTOS Driver	611
26.5.1	Overview	611
26.5.2	Data Structure Documentation	611
26.5.3	Macro Definition Documentation	612
26.5.4	Function Documentation	612
Chapter	Memory-Mapped Cryptographic Acceleration Unit (mmCAU)	
27.1	Overview	615
27.2	Purpose	615
27.3	Library Features	615
27.4	CAU and mmCAU software library overview	615
27.5	mmCAU software library usage	616
27.6	Function Documentation	618
27.6.1	cau_aes_set_key	618
27.6.2	cau_aes_encrypt	619
27.6.3	cau_aes_decrypt	619
27.6.4	cau_des_chk parity	620
27.6.5	cau_des_encrypt	620

Contents

Section Number	Title	Page Number
27.6.6	cau_des_decrypt	621
27.6.7	cau_md5_initialize_output	621
27.6.8	cau_md5_hash_n	622
27.6.9	cau_md5_update	623
27.6.10	cau_md5_hash	623
27.6.11	cau_sha1_initialize_output	624
27.6.12	cau_sha1_hash_n	624
27.6.13	cau_sha1_update	625
27.6.14	cau_sha1_hash	626
27.6.15	cau_sha256_initialize_output	626
27.6.16	cau_sha256_hash_n	627
27.6.17	cau_sha256_update	627
27.6.18	cau_sha256_hash	628
27.6.19	MMCAU_AES_SetKey	629
27.6.20	MMCAU_AES_EncryptEcb	629
27.6.21	MMCAU_AES_DecryptEcb	630
27.6.22	MMCAU_DES_ChkParity	631
27.6.23	MMCAU_DES_EncryptEcb	631
27.6.24	MMCAU_DES_DecryptEcb	632
27.6.25	MMCAU_MD5_InitializeOutput	633
27.6.26	MMCAU_MD5_HashN	633
27.6.27	MMCAU_MD5_Update	634
27.6.28	MMCAU_SHA1_InitializeOutput	635
27.6.29	MMCAU_SHA1_HashN	635
27.6.30	MMCAU_SHA1_Update	636
27.6.31	MMCAU_SHA256_InitializeOutput	637
27.6.32	MMCAU_SHA256_HashN	637
27.6.33	MMCAU_SHA256_Update	638

Chapter MIPI CSI2 RX: MIPI CSI2 RX Driver

28.1	Overview	641
28.2	Data Structure Documentation	643
28.2.1	struct csi2rx_config_t	643
28.3	Macro Definition Documentation	644
28.3.1	FSL_CSIRX_DRIVER_VERSION	644
28.4	Enumeration Type Documentation	644
28.4.1	_csi2rx_data_lane	644
28.4.2	_csi2rx_payload	644
28.4.3	_csi2rx_bit_error	645
28.4.4	csi2rx_ppi_error_t	645
28.4.5	_csi2rx_interrupt	645

Contents

Section Number	Title	Page Number
28.4.6	_csi2rx_ulps_status	645
28.5	Function Documentation	646
28.5.1	CSI2RX_Init	646
28.5.2	CSI2RX_Deinit	647
28.5.3	CSI2RX_GetBitError	647
28.5.4	CSI2RX_GetEccBitErrorPosition	648
28.5.5	CSI2RX_GetUlpsStatus	649
28.5.6	CSI2RX_GetPpiErrorDataLanes	649
28.5.7	CSI2RX_EnableInterrupts	650
28.5.8	CSI2RX_DisableInterrupts	650
28.5.9	CSI2RX_GetInterruptStatus	651
Chapter	MIPI DSI Driver	
29.1	Overview	653
29.2	Data Structure Documentation	660
29.2.1	struct dsi_config_t	660
29.2.2	struct dsi_dpi_config_t	660
29.2.3	struct dsi_dphy_config_t	662
29.2.4	struct dsi_transfer_t	664
29.2.5	struct _dsi_handle	664
29.3	Typedef Documentation	665
29.3.1	dsi_callback_t	665
29.4	Enumeration Type Documentation	665
29.4.1	_dsi_status	665
29.4.2	dsi_dpi_color_coding_t	665
29.4.3	dsi_dpi_pixel_packet_t	666
29.4.4	_dsi_dpi_polarity_flag	666
29.4.5	dsi_dpi_video_mode_t	666
29.4.6	dsi_dpi_bllp_mode_t	666
29.4.7	_dsi_apb_status	667
29.4.8	_dsi_rx_error_status	667
29.4.9	_dsi_host_status	667
29.4.10	_dsi_interrupt	668
29.4.11	dsi_tx_data_type_t	669
29.4.12	dsi_rx_data_type_t	670
29.4.13	_dsi_transfer_flags	670
29.5	Function Documentation	670
29.5.1	DSI_Init	670
29.5.2	DSI_Deinit	670

Contents

Section Number	Title	Page Number
29.5.3	DSI_SetDefaultConfig	671
29.5.4	DSI_SetDpiConfig	671
29.5.5	DSI_InitDphy	671
29.5.6	DSI_DeinitDphy	672
29.5.7	DSI_GetDphyDefaultConfig	672
29.5.8	DSI_EnableInterrupts	672
29.5.9	DSI_DisableInterrupts	673
29.5.10	DSI_GetAndClearInterruptStatus	673
29.5.11	DSI_SetApbPacketControl	673
29.5.12	DSI_WriteApbTxPayload	674
29.5.13	DSI_ReadApbRxPayload	674
29.5.14	DSI_SendApbPacket	674
29.5.15	DSI_GetApbStatus	675
29.5.16	DSI_GetRxErrorStatus	675
29.5.17	DSI_GetEccRxErrorPosition	675
29.5.18	DSI_GetAndClearHostStatus	676
29.5.19	DSI_GetRxPacketHeader	676
29.5.20	DSI_GetRxPacketType	676
29.5.21	DSI_GetRxPacketWordCount	677
29.5.22	DSI_GetRxPacketVirtualChannel	677
29.5.23	DSI_TransferBlocking	677
29.5.24	DSI_TransferCreateHandle	678
29.5.25	DSI_TransferNonBlocking	678
29.5.26	DSI_TransferAbort	679
29.5.27	DSI_TransferHandleIRQ	679

Chapter MU: Messaging Unit

30.1	Overview	681
30.2	Function description	681
30.2.1	MU initialization	681
30.2.2	MU message	681
30.2.3	MU flags	682
30.2.4	Status and interrupt	682
30.2.5	MU misc functions	682
30.3	Macro Definition Documentation	685
30.3.1	FSL_MU_DRIVER_VERSION	685
30.4	Enumeration Type Documentation	685
30.4.1	_mu_status_flags	685
30.4.2	_mu_interrupt_enable	685
30.4.3	_mu_interrupt_trigger	686

Contents

Section Number	Title	Page Number
30.5	Function Documentation	686
30.5.1	MU_Init	686
30.5.2	MU_Deinit	686
30.5.3	MU_SendMsgNonBlocking	686
30.5.4	MU_SendMsg	687
30.5.5	MU_ReceiveMsgNonBlocking	687
30.5.6	MU_ReceiveMsg	688
30.5.7	MU_SetFlagsNonBlocking	688
30.5.8	MU_SetFlags	689
30.5.9	MU_GetFlags	689
30.5.10	MU_GetStatusFlags	689
30.5.11	MU_ClearStatusFlags	690
30.5.12	MU_EnableInterrupts	691
30.5.13	MU_DisableInterrupts	691
30.5.14	MU_TriggerInterrupts	691
30.5.15	MU_ClearNmi	692
30.5.16	MU_BootCoreB	692
30.5.17	MU_HoldCoreBReset	692
30.5.18	MU_BootOtherCore	693
30.5.19	MU_HoldOtherCoreReset	693
30.5.20	MU_ResetBothSides	693
30.5.21	MU_HardwareResetOtherCore	694
30.5.22	MU_SetClockOnOtherCoreEnable	695
30.5.23	MU_GetOtherCorePowerMode	695

Chapter **Notification Framework**

31.1	Overview	697
31.2	Notifier Overview	697
31.3	Data Structure Documentation	699
31.3.1	struct notifier_notification_block_t	699
31.3.2	struct notifier_callback_config_t	700
31.3.3	struct notifier_handle_t	700
31.4	Typedef Documentation	701
31.4.1	notifier_user_config_t	701
31.4.2	notifier_user_function_t	701
31.4.3	notifier_callback_t	702
31.5	Enumeration Type Documentation	702
31.5.1	_notifier_status	702
31.5.2	notifier_policy_t	703
31.5.3	notifier_notification_type_t	703

Contents

Section Number	Title	Page Number
31.5.4	notifier_callback_type_t	703
31.6	Function Documentation	704
31.6.1	NOTIFIER_CreateHandle	704
31.6.2	NOTIFIER_SwitchConfig	705
31.6.3	NOTIFIER_GetErrorCallbackIndex	706
Chapter	RGPIO: Rapid General-Purpose Input/Output Driver	
32.1	Overview	707
32.2	Data Structure Documentation	707
32.2.1	struct rgpio_pin_config_t	707
32.3	Macro Definition Documentation	708
32.3.1	FSL_RGPIO_DRIVER_VERSION	708
32.4	Enumeration Type Documentation	708
32.4.1	rgpio_pin_direction_t	708
32.5	RGPIO Driver	709
32.5.1	Overview	709
32.5.2	Typical use case	709
32.5.3	Function Documentation	710
32.6	FGPIO Driver	714
32.6.1	Typical use case	714
Chapter	Shell	
33.1	Overview	715
33.2	Function groups	715
33.2.1	Initialization	715
33.2.2	Advanced Feature	715
33.2.3	Shell Operation	716
33.3	Data Structure Documentation	717
33.3.1	struct shell_command_t	717
33.4	Macro Definition Documentation	718
33.4.1	SHELL_NON_BLOCKING_MODE	718
33.4.2	SHELL_AUTO_COMPLETE	718
33.4.3	SHELL_BUFFER_SIZE	718
33.4.4	SHELL_MAX_ARGS	718
33.4.5	SHELL_HISTORY_COUNT	718

Contents

Section Number	Title	Page Number
33.4.6	SHELL_HANDLE_SIZE	718
33.4.7	SHELL_COMMAND_DEFINE	718
33.4.8	SHELL_COMMAND	719
33.5	Typedef Documentation	719
33.5.1	cmd_function_t	719
33.6	Enumeration Type Documentation	719
33.6.1	shell_status_t	719
33.7	Function Documentation	719
33.7.1	SHELL_Init	719
33.7.2	SHELL_RegisterCommand	720
33.7.3	SHELL_UnregisterCommand	721
33.7.4	SHELL_Write	721
33.7.5	SHELL_Printf	721
33.7.6	SHELL_Task	722
 Chapter SEMA42: Hardware Semaphores Driver		
34.1	Overview	723
34.2	Typical use case	723
34.3	Macro Definition Documentation	725
34.3.1	SEMA42_GATE_NUM_RESET_ALL	725
34.3.2	SEMA42_GATEn	725
34.4	Enumeration Type Documentation	725
34.4.1	_sema42_status	725
34.4.2	sema42_gate_status_t	725
34.5	Function Documentation	726
34.5.1	SEMA42_Init	726
34.5.2	SEMA42_Deinit	726
34.5.3	SEMA42_TryLock	726
34.5.4	SEMA42_Lock	727
34.5.5	SEMA42_Unlock	728
34.5.6	SEMA42_GetGateStatus	728
34.5.7	SEMA42_ResetGate	728
34.5.8	SEMA42_ResetAllGates	729
 Chapter TPM: Timer PWM Module		
35.1	Overview	731

Contents

Section Number	Title	Page Number
35.2	Typical use case	732
35.2.1	PWM output	732
35.3	Data Structure Documentation	736
35.3.1	struct tpm_chnl_pwm_signal_param_t	736
35.3.2	struct tpm_dual_edge_capture_param_t	736
35.3.3	struct tpm_phase_params_t	737
35.3.4	struct tpm_config_t	737
35.4	Enumeration Type Documentation	737
35.4.1	tpm_chnl_t	737
35.4.2	tpm_pwm_mode_t	738
35.4.3	tpm_pwm_level_select_t	738
35.4.4	tpm_trigger_select_t	738
35.4.5	tpm_trigger_source_t	739
35.4.6	tpm_output_compare_mode_t	739
35.4.7	tpm_input_capture_edge_t	739
35.4.8	tpm_quad_decode_mode_t	739
35.4.9	tpm_phase_polarity_t	740
35.4.10	tpm_clock_source_t	740
35.4.11	tpm_clock_prescale_t	740
35.4.12	tpm_interrupt_enable_t	740
35.4.13	tpm_status_flags_t	741
35.5	Function Documentation	741
35.5.1	TPM_Init	741
35.5.2	TPM_Deinit	741
35.5.3	TPM_GetDefaultConfig	741
35.5.4	TPM_SetupPwm	742
35.5.5	TPM_UpdatePwmDutycycle	742
35.5.6	TPM_UpdateChnlEdgeLevelSelect	743
35.5.7	TPM_SetupInputCapture	743
35.5.8	TPM_SetupOutputCompare	743
35.5.9	TPM_SetupDualEdgeCapture	744
35.5.10	TPM_SetupQuadDecode	744
35.5.11	TPM_EnableInterrupts	744
35.5.12	TPM_DisableInterrupts	745
35.5.13	TPM_GetEnabledInterrupts	745
35.5.14	TPM_GetStatusFlags	745
35.5.15	TPM_ClearStatusFlags	745
35.5.16	TPM_SetTimerPeriod	746
35.5.17	TPM_GetCurrentTimerCount	746
35.5.18	TPM_StartTimer	747
35.5.19	TPM_StopTimer	748
35.5.20	TPM_Reset	748

Contents

Section Number	Title	Page Number
Chapter	TSTMR: Timestamp Timer Driver	
36.1	Overview	749
36.2	Function Documentation	749
36.2.1	TSTMR_ReadTimeStamp	749
36.2.2	TSTMR_DelayUs	749
Chapter	WDOG32: 32-bit Watchdog Timer	
37.1	Overview	751
37.2	Typical use case	751
37.3	Data Structure Documentation	753
37.3.1	struct wdog32_work_mode_t	753
37.3.2	struct wdog32_config_t	753
37.4	Macro Definition Documentation	754
37.4.1	FSL_WDOG32_DRIVER_VERSION	754
37.5	Enumeration Type Documentation	754
37.5.1	wdog32_clock_source_t	754
37.5.2	wdog32_clock_prescaler_t	754
37.5.3	wdog32_test_mode_t	754
37.5.4	_wdog32_interrupt_enable_t	754
37.5.5	_wdog32_status_flags_t	755
37.6	Function Documentation	755
37.6.1	WDOG32_GetDefaultConfig	755
37.6.2	AT_QUICKACCESS_SECTION_CODE	755
37.6.3	WDOG32_Deinit	756
37.6.4	WDOG32_Enable	756
37.6.5	WDOG32_Disable	756
37.6.6	WDOG32_EnableInterrupts	757
37.6.7	WDOG32_DisableInterrupts	757
37.6.8	WDOG32_GetStatusFlags	757
37.6.9	AT_QUICKACCESS_SECTION_CODE	758
37.6.10	WDOG32_SetTimeoutValue	758
37.6.11	WDOG32_SetWindowValue	759
37.6.12	WDOG32_Unlock	759
37.6.13	WDOG32_Refresh	759
37.6.14	WDOG32_GetCounterValue	760

Contents

Section Number	Title	Page Number
Chapter Serial Manager		
38.1	Overview	761
38.2	Data Structure Documentation	763
38.2.1	struct serial_manager_config_t	763
38.2.2	struct serial_manager_callback_message_t	763
38.3	Enumeration Type Documentation	763
38.3.1	serial_port_type_t	763
38.3.2	serial_manager_status_t	763
38.4	Function Documentation	764
38.4.1	SerialManager_Init	764
38.4.2	SerialManager_Deinit	765
38.4.3	SerialManager_OpenWriteHandle	765
38.4.4	SerialManager_CloseWriteHandle	766
38.4.5	SerialManager_OpenReadHandle	767
38.4.6	SerialManager_CloseReadHandle	767
38.4.7	SerialManager_WriteBlocking	768
38.4.8	SerialManager_ReadBlocking	769
38.4.9	SerialManager_EnterLowpower	769
38.4.10	SerialManager_ExitLowpower	770
38.5	Serial Port Uart	771
38.5.1	Overview	771
38.5.2	Data Structure Documentation	771
38.5.3	Enumeration Type Documentation	772
38.6	Serial Port USB	773
38.6.1	Overview	773
38.6.2	Data Structure Documentation	774
38.6.3	Enumeration Type Documentation	774
38.6.4	USB Device Configuration	775
38.7	Serial Port SWO	777
38.7.1	Overview	777
38.7.2	Data Structure Documentation	777
38.7.3	Enumeration Type Documentation	777
Chapter GenericList		
39.1	Overview	779
39.2	Data Structure Documentation	780
39.2.1	struct list_t	780

Contents

Section Number	Title	Page Number
39.2.2	<code>struct list_element_t</code>	780
39.3	Enumeration Type Documentation	780
39.3.1	<code>list_status_t</code>	780
39.4	Function Documentation	781
39.4.1	<code>LIST_Init</code>	781
39.4.2	<code>LIST_GetList</code>	781
39.4.3	<code>LIST_AddHead</code>	781
39.4.4	<code>LIST_AddTail</code>	781
39.4.5	<code>LIST_RemoveHead</code>	782
39.4.6	<code>LIST_GetHead</code>	782
39.4.7	<code>LIST_GetNext</code>	782
39.4.8	<code>LIST_GetPrev</code>	783
39.4.9	<code>LIST_RemoveElement</code>	783
39.4.10	<code>LIST_AddPrevElement</code>	783
39.4.11	<code>LIST.GetSize</code>	784
39.4.12	<code>LIST_GetAvailableSize</code>	784

Chapter **UART_Adapter**

40.1	Overview	785
40.2	Data Structure Documentation	787
40.2.1	<code>struct hal_uart_config_t</code>	787
40.2.2	<code>struct hal_uart_transfer_t</code>	787
40.3	Macro Definition Documentation	787
40.3.1	<code>HAL_UART_TRANSFER_MODE</code>	787
40.4	Typedef Documentation	788
40.4.1	<code>hal_uart_transfer_callback_t</code>	788
40.5	Enumeration Type Documentation	788
40.5.1	<code>hal_uart_status_t</code>	788
40.5.2	<code>hal_uart_parity_mode_t</code>	788
40.5.3	<code>hal_uart_stop_bit_count_t</code>	788
40.6	Function Documentation	788
40.6.1	<code>HAL_UartInit</code>	788
40.6.2	<code>HAL_UartDeinit</code>	789
40.6.3	<code>HAL_UartReceiveBlocking</code>	789
40.6.4	<code>HAL_UartSendBlocking</code>	790
40.6.5	<code>HAL_UartInstallCallback</code>	791
40.6.6	<code>HAL_UartReceiveNonBlocking</code>	791
40.6.7	<code>HAL_UartSendNonBlocking</code>	792

Section Number	Contents	Title	Page Number
40.6.8	HAL_UartGetReceiveCount	HAL_UartGetReceiveCount	793
40.6.9	HAL_UartGetSendCount	HAL_UartGetSendCount	793
40.6.10	HAL_UartAbortReceive	HAL_UartAbortReceive	793
40.6.11	HAL_UartAbortSend	HAL_UartAbortSend	794
40.6.12	HAL_UartIsrFunction	HAL_UartIsrFunction	794

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 FreeRTOSTM. In addition to the base enablement, the MCUXpresso SDK is augmented with demo applications, driver example projects, and API documentation to help users quickly leverage the support provided by MCUXpresso SDK. The [MCUXpresso SDK Web Builder](#) is available to provide access to all MCUXpresso SDK packages. See the *MCUXpresso Software Development Kit (SDK) Release Notes* (document MCUXSDKRNN) in the Supported Devices section at [MCUXpresso-SDK: Software Development Kit for MCUXpresso](#) for details.

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

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

All demo applications and driver examples are provided with projects for the following toolchains:

- IAR Embedded Workbench
- GNU Arm Embedded Toolchain

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/](#).

Deliverable	Location
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

Table 2: MCUXpresso SDK Folder Structure

Chapter 2

Driver errors status

- `kStatus_EDMA_QueueFull` = 5100
- `kStatus_EDMA_Busy` = 5101
- `kStatus_ENET_RxFrameError` = 4000
- `kStatus_ENET_RxFrameFail` = 4001
- `kStatus_ENET_RxFrameEmpty` = 4002
- `kStatus_ENET_TxFrameOverLen` = 4003
- `kStatus_ENET_TxFrameBusy` = 4004
- `kStatus_ENET_TxFrameFail` = 4005
- `kStatus_ENET_PtpTsRingFull` = 4006
- `kStatus_ENET_PtpTsRingEmpty` = 4007
- `kStatus_ESAI_TxBusy` = 6900
- `kStatus_ESAI_RxBusy` = 6901
- `kStatus_ESAI_TxError` = 6902
- `kStatus_ESAI_RxError` = 6903
- `kStatus_ESAI_QueueFull` = 6904
- `kStatus_ESAI_TxIdle` = 6905
- `kStatus_ESAI_RxIdle` = 6906
- `kStatus_FLEXCAN_TxBusy` = 5300
- `kStatus_FLEXCAN_TxIdle` = 5301
- `kStatus_FLEXCAN_TxSwitchToRx` = 5302
- `kStatus_FLEXCAN_RxBusy` = 5303
- `kStatus_FLEXCAN_RxIdle` = 5304
- `kStatus_FLEXCAN_RxOverflow` = 5305
- `kStatus_FLEXCAN_RxFifoBusy` = 5306
- `kStatus_FLEXCAN_RxFifoIdle` = 5307
- `kStatus_FLEXCAN_RxFifoOverflow` = 5308
- `kStatus_FLEXCAN_RxFifoWarning` = 5309
- `kStatus_FLEXCAN_ErrorStatus` = 5310
- `kStatus_FLEXCAN_UnHandled` = 5311
- `#kStatus_FLEXSPI_Idle` = 7000
- `kStatus_FLEXSPI_Busy` = 7001
- `kStatus_FLEXSPI_SequenceExecutionTimeout` = 7002
- `kStatus_FLEXSPI_IpCommandSequenceError` = 7003
- `kStatus_FLEXSPI_IpCommandGrantTimeout` = 7004
- `kStatus_LPI2C_Busy` = 900
- `kStatus_LPI2C_Idle` = 901
- `kStatus_LPI2C_Nak` = 902
- `kStatus_LPI2C_FifoError` = 903

- `kStatus_LPI2C_BitError` = 904
- `kStatus_LPI2C_ArbitrationLost` = 905
- `kStatus_LPI2C_PinLowTimeout` = 906
- `kStatus_LPI2C_NoTransferInProgress` = 907
- `kStatus_LPI2C_DmaRequestFail` = 908
- `kStatus_LPI2C_Timeout` = 909
- `kStatus_LPSPI_Busy` = 400
- `kStatus_LPSPI_Error` = 401
- `kStatus_LPSPI_Idle` = 402
- `kStatus_LPSPI_OutOfRange` = 403
- `kStatus_LPUART_TxBusy` = 1300
- `kStatus_LPUART_RxBusy` = 1301
- `kStatus_LPUART_TxIdle` = 1302
- `kStatus_LPUART_RxIdle` = 1303
- `kStatus_LPUART_TxWatermarkTooLarge` = 1304
- `kStatus_LPUART_RxWatermarkTooLarge` = 1305
- `kStatus_LPUART_FlagCannotClearManually` = 1306
- `kStatus_LPUART_Error` = 1307
- `kStatus_LPUART_RxRingBufferOverrun` = 1308
- `kStatus_LPUART_RxHardwareOverrun` = 1309
- `kStatus_LPUART_NoiseError` = 1310
- `kStatus_LPUART_FramingError` = 1311
- `kStatus_LPUART_ParityError` = 1312
- `kStatus_LPUART_BaudrateNotSupport` = 1313
- `kStatus_LPUART_IdleLineDetected` = 1314
- `kStatus_DSI_Busy` = 3000
- `kStatus_DSI_RxDataError` = 3001
- `kStatus_DSI_ErrorReportReceived` = 3002
- `kStatus_DSI_NotSupported` = 3003
- `kStatus_NOTIFIER_ErrorNotificationBefore` = 9800
- `kStatus_NOTIFIER_ErrorNotificationAfter` = 9801
- `kStatus_SEMA42_Busy` = 1600
- `kStatus_SEMA42_Reseting` = 1601

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

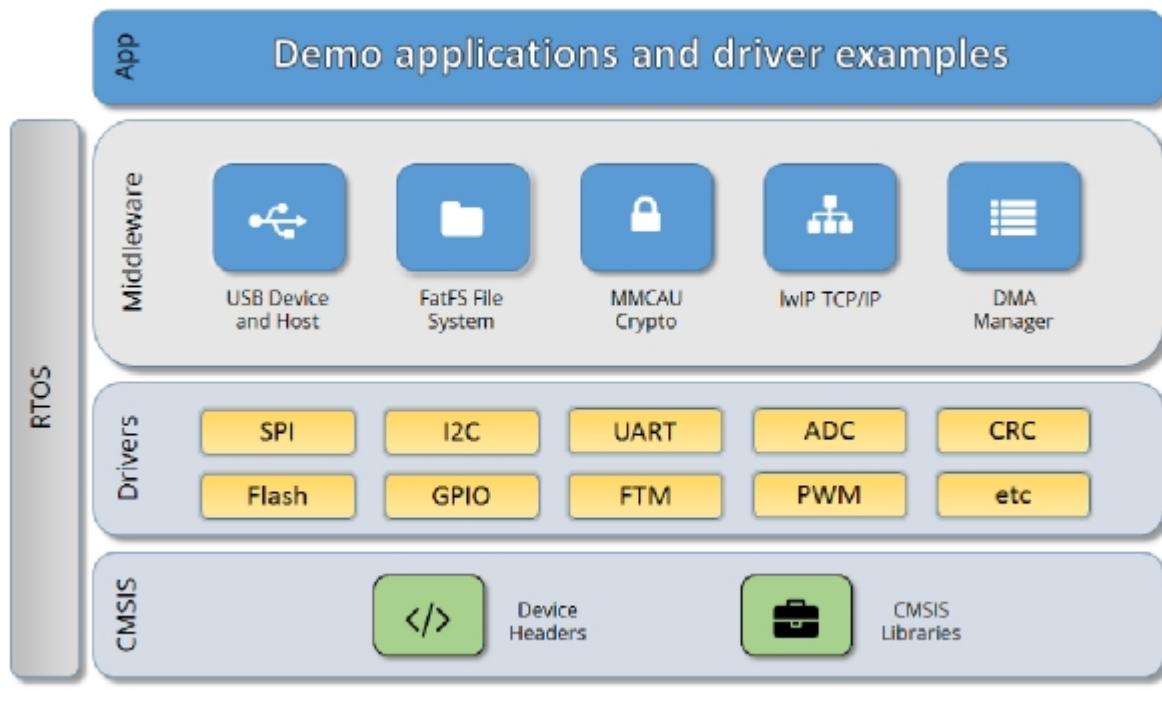


Figure 1: MCUXpresso SDK Block Diagram

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 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 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.

How to Reach Us:

Home Page: nxp.com

Web Support: 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: www.nxp.com/SalesTermsandConditions {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-MOBILE, 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.

© 2019 NXP B.V.

Chapter 5

CACHE: CACHE Memory Controller

5.1 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 L1 cache driver API. This level provides the level 1 caches controller drivers. The L1 caches in this arch is the previous the local memory controller (LMEM).

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, USDHC, 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 only L1 driver APIs.

5.2 Function groups

L1 CACHE Operation {#L1CACHE_MaintainOperation}

The L1 CACHE has both code cache and data cache. This function group provides two independent API groups for both code cache and data cache. There are Enable/Disable APIs for code cache and data cache control and cache maintenance operations as Invalidate/Clean/CleanInvalidate by all and by address range.

Macros

- #define `L1CODEBUSCACHE_LINESIZE_BYTE` FSL_FEATURE_L1ICACHE_LINESIZE_BY-
TE
code bus cache line size is equal to system bus line size, so the unified I/D cache line size equals too.
- #define `L1SYSTEMBUSCACHE_LINESIZE_BYTE` `L1CODEBUSCACHE_LINESIZE_BYTE`
The system bus CACHE line size is 16B = 128b.

Driver version

- #define `FSL_CACHE_DRIVER_VERSION` (MAKE_VERSION(2, 0, 1))
cache driver version 2.0.1.

cache control for L1 cache (local memory controller for code/system bus cache)

- void `L1CACHE_EnableCodeCache` (void)
Enables the processor code bus cache.
- void `L1CACHE_DisableCodeCache` (void)
Disables the processor code bus cache.

Function groups

- void **L1CACHE_InvalidateCodeCache** (void)
Invalidates the processor code bus cache.
- void **L1CACHE_InvalidateCodeCacheByRange** (uint32_t address, uint32_t size_byte)
Invalidates processor code bus cache by range.
- void **L1CACHE_CleanCodeCache** (void)
Cleans the processor code bus cache.
- void **L1CACHE_CleanCodeCacheByRange** (uint32_t address, uint32_t size_byte)
Cleans processor code bus cache by range.
- void **L1CACHE_CleanInvalidateCodeCache** (void)
Cleans and invalidates the processor code bus cache.
- void **L1CACHE_CleanInvalidateCodeCacheByRange** (uint32_t address, uint32_t size_byte)
Cleans and invalidate processor code bus cache by range.
- static void **L1CACHE_EnableCodeCacheWriteBuffer** (bool enable)
Enables/disables the processor code bus write buffer.
- void **L1CACHE_EnableSystemCache** (void)
Enables the processor system bus cache.
- void **L1CACHE_DisableSystemCache** (void)
Disables the processor system bus cache.
- void **L1CACHE_InvalidateSystemCache** (void)
Invalidates the processor system bus cache.
- void **L1CACHE_InvalidateSystemCacheByRange** (uint32_t address, uint32_t size_byte)
Invalidates processor system bus cache by range.
- void **L1CACHE_CleanSystemCache** (void)
Cleans the processor system bus cache.
- void **L1CACHE_CleanSystemCacheByRange** (uint32_t address, uint32_t size_byte)
Cleans processor system bus cache by range.
- void **L1CACHE_CleanInvalidateSystemCache** (void)
Cleans and invalidates the processor system bus cache.
- void **L1CACHE_CleanInvalidateSystemCacheByRange** (uint32_t address, uint32_t size_byte)
Cleans and Invalidates processor system bus cache by range.
- static void **L1CACHE_EnableSystemCacheWriteBuffer** (bool enable)
Enables/disables the processor system bus write buffer.

cache control for unified L1 cache driver

- void **L1CACHE_InvalidateICacheByRange** (uint32_t address, uint32_t size_byte)
Invalidates cortex-m4 L1 instrument cache by range.
- static void **L1CACHE_InvalidateDCacheByRange** (uint32_t address, uint32_t size_byte)
Invalidates cortex-m4 L1 data cache by range.
- void **L1CACHE_CleanDCacheByRange** (uint32_t address, uint32_t size_byte)
Cleans cortex-m4 L1 data cache by range.
- void **L1CACHE_CleanInvalidateDCacheByRange** (uint32_t address, uint32_t size_byte)
Cleans and Invalidates cortex-m4 L1 data cache by range.

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)

- static void **DCACHE_CleanInvalidateByRange** (uint32_t address, uint32_t size_byte)
Cleans and Invalidates data cache by range.

5.3 Macro Definition Documentation

5.3.1 #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

5.3.2 #define L1CODEBUSCACHE_LINESIZE_BYTE FSL_FEATURE_L1ICACHE_LINESIZE_BYTE

The code bus CACHE line size is 16B = 128b.

5.3.3 #define L1SYSTEMBUSCACHE_LINESIZE_BYTE L1CODEBUSCACHE_LINESIZE_BYTE

5.4 Function Documentation

5.4.1 void L1CACHE_InvalidateCodeCacheByRange (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.

Note

Address and size should be aligned to "L1CODECACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_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.

5.4.2 void L1CACHE_CleanCodeCacheByRange (uint32_t address, uint32_t size_byte)

Function Documentation

Parameters

<i>address</i>	The physical address of cache.
<i>size_byte</i>	size of the memory to be cleaned.

Note

Address and size should be aligned to "L1CODEBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_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.

5.4.3 void L1CACHE_CleanInvalidateCodeCacheByRange (**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.

Note

Address and size should be aligned to "L1CODEBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_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.

5.4.4 static void L1CACHE_EnableCodeCacheWriteBuffer (**bool enable**) [**inline**], [**static**]

Parameters

<i>enable</i>	The enable or disable flag. true - enable the code bus write buffer. false - disable the code bus write buffer.
---------------	---

5.4.5 void L1CACHE_InvalidateSystemCacheByRange (**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.

Note

Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The start-Addr here will be forced to align to L1SYSTEMBUSCACHE_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.

5.4.6 void L1CACHE_CleanSystemCacheByRange (**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.

Note

Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The start-Addr here will be forced to align to L1SYSTEMBUSCACHE_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.

5.4.7 void L1CACHE_CleanInvalidateSystemCacheByRange (**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 Clean and Invalidated.

Note

Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The start-Addr here will be forced to align to L1SYSTEMBUSCACHE_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.

Function Documentation

5.4.8 **static void L1CACHE_EnableSystemCacheWriteBuffer (bool *enable*)**
[**inline**], [**static**]

Parameters

<i>enable</i>	The enable or disable flag. true - enable the code bus write buffer. false - disable the code bus write buffer.
---------------	---

5.4.9 void L1CACHE_InvalidateCacheByRange (*uint32_t address, uint32_t size_byte*)

Parameters

<i>address</i>	The start address of the memory to be invalidated.
<i>size_byte</i>	The memory size.

Note

The start address and size_byte should be 16-Byte(FSL FEATURE_L1ICACHE_LINESIZE_BY-TE) aligned.

5.4.10 static void L1CACHE_InvalidateDCacheByRange (*uint32_t address, uint32_t size_byte*) [inline], [static]

Parameters

<i>address</i>	The start address of the memory to be invalidated.
<i>size_byte</i>	The memory size.

Note

The start address and size_byte should be 16-Byte(FSL FEATURE_L1DCACHE_LINESIZE_BY-TE) aligned.

5.4.11 void L1CACHE_CleanDCacheByRange (*uint32_t address, uint32_t size_byte*)

Function Documentation

Parameters

<i>address</i>	The start address of the memory to be cleaned.
<i>size_byte</i>	The memory size.

Note

The start address and size_byte should be 16-Byte(FSL FEATURE_L1DCACHE_LINESIZE_BY-TE) aligned.

5.4.12 void L1CACHE_CleanInvalidateDCacheByRange (uint32_t *address*, uint32_t *size_byte*)

Parameters

<i>address</i>	The start address of the memory to be clean and invalidated.
<i>size_byte</i>	The memory size.

Note

The start address and size_byte should be 16-Byte(FSL FEATURE_L1DCACHE_LINESIZE_BY-TE) aligned.

5.4.13 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.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL FEATURE_L1ICACHE_LINESIZE_BYT. 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.

5.4.14 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.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_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.

5.4.15 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.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_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.

5.4.16 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.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_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.

Function Documentation

Chapter 6

Clock Driver

6.1 Overview

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

Macros

- #define **FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL** 0
Configure whether driver controls clock.
- #define **MU_CLOCKS**
Clock ip name array for MU.
- #define **GPIO_CLOCKS**
Clock ip name array for GPIO.
- #define **RGPIO_CLOCKS**
Clock ip name array for RGPIO.
- #define **FTM_CLOCKS**
Clock ip name array for FTM.
- #define **GPT_CLOCKS**
Clock ip name array for GPT.
- #define **FLEXCAN_CLOCKS**
Clock ip name array for FLEXCAN.
- #define **FLEXSPI_CLOCKS**
Clock ip name array for FLEXSPI.
- #define **LPUART_CLOCKS**
Clock ip name array for LPUART.
- #define **LPADC_CLOCKS**
Clock ip name array for LPADC.
- #define **INTMUX_CLOCKS**
Clock ip name array for INTMUX.
- #define **SAI_CLOCKS**
Clock ip name array for SAI.
- #define **SEMA42_CLOCKS**
Clock ip name array for SEMA42.
- #define **TPM_CLOCKS**
Clock ip name array for TPM.
- #define **LPIT_CLOCKS**
Clock ip name array for LPIT.
- #define **LPI2C_CLOCKS**
Clock ip name array for LPI2C.
- #define **LPSPI_CLOCKS**
Clock ip name array for LPSPI.
- #define **EDMA_CLOCKS**
Clock ip name array for EDMA.
- #define **ESAI_CLOCKS**
Clock ip name array for ESAI.

Overview

- #define **ISI_CLOCKS**
Clock ip name array for ISI.
- #define **MIPI_CSI2RX_CLOCKS**
Clock ip name array for MIPI CSI2 RX.
- #define **MIPI_DSI_HOST_CLOCKS**
Clock ip name array for MIPI DSI host.
- #define **ENET_CLOCKS**
Clock ip name array for ENET.
- #define **EMVSIM_CLOCKS**
Clock ip name array for EMVSIM.
- #define **DPU_CLOCKS**
Clock ip name array for DPU.
- #define **LDB_CLOCKS**
Clock ip name array for LVDS display bridge(LDB).
- #define **LPCG_TUPLE(rsrcc, base)** ((uint32_t) (((base) >> 12U) << 10U) | rsrcc))
LPCG TUPLE macros to map corresponding ip clock name, SCFWAPI resource index and LPCG Register base address.
- #define **LPCG_TUPLE_REG_BASE(tuple)** ((volatile uint32_t *) (((uint32_t)(tuple) >> 10U) & 0xFFFFFU) << 12U))
Get the LPCG REG base address.
- #define **LPCG_TUPLE_RSRC(tuple)** ((sc_rsrc_t)((uint32_t)(tuple)&0x3FFU))
Get the resource index.
- #define **NV (0U)**
LPCG Cell not available.

Enumerations

- enum **clock_ip_src_t** {
 kCLOCK_IpSrcNone = 0U,
 kCLOCK_IpSrcDummy = 1U }
Clock source for peripherals that support various clock selections.
- enum **clock_name_t** {
 kCLOCK_CoreSysClk,
 kCLOCK_CONNECTIVITY_AhbClk }
Clock name used to get clock frequency.
- enum **clock_ip_name_t**
Peripheral clock name definition used for clock gate, clock source and clock divider setting.

Functions

- void **CLOCK_Init** (sc_ipc_t ipc)
Initialize Clock module.
- void **CLOCK_Deinit** (void)
Deinitialize Clock module.
- bool **CLOCK_EnableClockExt** (clock_ip_name_t name, uint32_t gate)
Enable the clock for specific IP, with gate setting.
- static bool **CLOCK_EnableClock** (clock_ip_name_t name)
Enable the clock for specific IP.
- bool **CLOCK_DisableClock** (clock_ip_name_t name)
Disable the clock for specific IP.

- `uint32_t CLOCK_SetIpFreq (clock_ip_name_t name, uint32_t freq)`
Set the clock frequency for specific IP module.
- `uint32_t CLOCK_GetIpFreq (clock_ip_name_t name)`
Get the clock frequency for a specific IP module.
- `uint32_t CLOCK_GetFreq (clock_name_t name)`
Gets the clock frequency for a specific clock name.
- `uint32_t CLOCK_GetCoreSysClkFreq (void)`
Get the core clock or system clock frequency.
- `void CLOCK_ConfigLPCG (clock_ip_name_t name, uint32_t swGate, uint32_t hwGate)`
Config the LPCG cell for specific IP.
- `void CLOCK_SetLpcgGate (volatile uint32_t *regBase, uint32_t swGate, uint32_t hwGate, uint32_t bitsMask)`
Set LPCG gate for specific LPCG.

Driver version

- `#define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`
CLOCK driver version 2.1.0.

6.2 Macro Definition Documentation

6.2.1 #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.

6.2.2 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

6.2.3 #define MU_CLOCKS

6.2.4 #define GPIO_CLOCKS

Value:

```
{
    \KCLOCK_IpInvalid,
    \KCLOCK_IpInvalid,
    \KCLOCK_IpInvalid,
    \KCLOCK_IpInvalid,
    \KCLOCK_IpInvalid,
    \KCLOCK_HSIO_Gpio,
```

Macro Definition Documentation

```
kCLOCK_LSIO_Gpio0, \
kCLOCK_LSIO_Gpio1, \
kCLOCK_LSIO_Gpio2, \
kCLOCK_LSIO_Gpio3, \
kCLOCK_LSIO_Gpio4, \
kCLOCK_LSIO_Gpio5, \
kCLOCK_LSIO_Gpio6, \
kCLOCK_LSIO_Gpio7, \
kCLOCK_IpInvalid, \
kCLOCK_IpInvalid, \
kCLOCK_IpInvalid, \
}
```

6.2.5 #define RGPIO_CLOCKS

Value:

```
{ \
    kCLOCK_M4_0_Rgpio, \
    kCLOCK_M4_1_Rgpio, \
}
```

6.2.6 #define FTM_CLOCKS

Value:

```
{ \
    kCLOCK_DMA_Ftm0, \
    kCLOCK_DMA_Ftm1, \
}
```

6.2.7 #define GPT_CLOCKS

Value:

```
{ \
    kCLOCK_AUDIO_Gpt0, \
    kCLOCK_AUDIO_Gpt1, \
    kCLOCK_AUDIO_Gpt2, \
    kCLOCK_AUDIO_Gpt3, \
    kCLOCK_AUDIO_Gpt4, \
    kCLOCK_AUDIO_Gpt5, \
    kCLOCK_LSI0_Gpt0, \
    kCLOCK_LSI0_Gpt1, \
    kCLOCK_LSI0_Gpt2, \
    kCLOCK_LSI0_Gpt3, \
    kCLOCK_LSI0_Gpt4, \
}
```

6.2.8 #define FLEXCAN_CLOCKS

Value:

```
{
    kCLOCK_DMA_Can0, \
    kCLOCK_DMA_Can1, \
    kCLOCK_DMA_Can2,
}
```

6.2.9 #define FLEXSPI_CLOCKS

Value:

```
{
    kCLOCK_LSIO_Flexspi0, \
    kCLOCK_LSIO_Flexspi1,
}
```

6.2.10 #define LPUART_CLOCKS

Value:

```
{
    kCLOCK_M4_0_Lpuart, \
    kCLOCK_M4_1_Lpuart, \
    kCLOCK_DMA_Lpuart0, \
    kCLOCK_DMA_Lpuart1, \
    kCLOCK_DMA_Lpuart2, \
    kCLOCK_DMA_Lpuart3, \
    kCLOCK_DMA_Lpuart4, \
    kCLOCK_SCU_Lpuart, \
}
```

6.2.11 #define LPADC_CLOCKS

Value:

```
{
    kCLOCK_DMA_Lpadc0, \
    kCLOCK_DMA_Lpadc1,
}
```

Macro Definition Documentation

6.2.12 #define INTMUX_CLOCKS

Value:

```
{\n    kCLOCK_M4_0_Intmux,\n    kCLOCK_M4_1_Intmux,\n    kCLOCK_TpInvalid,\n}
```

6.2.13 #define SAI_CLOCKS

Value:

```
{\n    kCLOCK_AUDIO_Sai0,\n    kCLOCK_AUDIO_Sai1,\n    kCLOCK_AUDIO_Sai2,\n    kCLOCK_AUDIO_Sai3,\n    kCLOCK_AUDIO_Sai4,\n    kCLOCK_AUDIO_Sai5,\n    kCLOCK_AUDIO_Sai6,\n    kCLOCK_AUDIO_Sai7,\n}
```

6.2.14 #define SEMA42_CLOCKS

Value:

```
{\n    kCLOCK_M4_0_Sema42,\n    kCLOCK_M4_1_Sema42,\n    kCLOCK_SCU_Sema42,\n}
```

6.2.15 #define TPM_CLOCKS

Value:

```
{\n    kCLOCK_M4_0_Tpm,\n    kCLOCK_M4_1_Tpm,\n    kCLOCK_SCU_Tpm,\n}
```

6.2.16 #define LPIT_CLOCKS

Value:

```
{
    kCLOCK_M4_0_Lpit,
    kCLOCK_M4_1_Lpit,
    kCLOCK_SCU_Lpit,
}
```

6.2.17 #define LPI2C_CLOCKS

Value:

```
{
    kCLOCK_M4_0_Lpi2c,
    kCLOCK_M4_1_Lpi2c,
    kCLOCK_HDMI_Lpi2c0,
    kCLOCK_LVDS_0_Lpi2c1,
    kCLOCK_LVDS_0_Lpi2c0,
    kCLOCK_LVDS_1_Lpi2c1,
    kCLOCK_LVDS_1_Lpi2c0,
    kCLOCK_MIPI_0_Lpi2c0,
    kCLOCK_MIPI_0_Lpi2c1,
    kCLOCK_MIPI_1_Lpi2c0,
    kCLOCK_MIPI_1_Lpi2c1,
    kCLOCK_DMA_Lpi2c0,
    kCLOCK_DMA_Lpi2c1,
    kCLOCK_DMA_Lpi2c2,
    kCLOCK_DMA_Lpi2c3,
    kCLOCK_DMA_Lpi2c4,
    kCLOCK_CSI_0_Lpi2c0,
    kCLOCK_CSI_1_Lpi2c0,
    kCLOCK_HDMI_RX_Lpi2c0,
    kCLOCK_SCU_Lpi2c,
}
```

6.2.18 #define LPSPI_CLOCKS

Value:

```
{
    kCLOCK_DMA_Lpspi0,
    kCLOCK_DMA_Lpspi1,
    kCLOCK_DMA_Lpspi2,
    kCLOCK_DMA_Lpspi3,
}
```

Macro Definition Documentation

6.2.19 #define EDMA_CLOCKS

Value:

```
{\n    kCLOCK_DMA_Dma0,\n}
```

6.2.20 #define ESAI_CLOCKS

Value:

```
{\n    kCLOCK_AUDIO_Esai0,\n    kCLOCK_AUDIO_Esai1\n}
```

6.2.21 #define ISI_CLOCKS

Value:

```
{\n    kCLOCK_IMAGING_Isi0,\n    kCLOCK_IMAGING_Isi1,\n    kCLOCK_IMAGING_Isi2,\n    kCLOCK_IMAGING_Isi3,\n    kCLOCK_IMAGING_Isi4,\n    kCLOCK_IMAGING_Isi5,\n    kCLOCK_IMAGING_Isi6,\n    kCLOCK_IMAGING_Isi7,\n}
```

6.2.22 #define MIPI_CSI2RX_CLOCKS

Value:

```
{\n    kCLOCK_MipiCsi2Rx0,\n    kCLOCK_MipiCsi2Rx1\n}
```

6.2.23 #define MIPI_DSI_HOST_CLOCKS

Value:

```
{\n    kCLOCK_MipiDsiHost0,\n    kCLOCK_MipiDsiHost1\n}
```

6.2.24 #define ENET_CLOCKS**Value:**

```
{
    \_kCLOCK_CONNECTIVITY_Enet0, \_
    \_kCLOCK_CONNECTIVITY_Enet1 \_
}
```

6.2.25 #define EMVSIM_CLOCKS**Value:**

```
{
    \_kCLOCK_DMA_EmvSim0, \_
    \_kCLOCK_DMA_EmvSim1, \_
}
```

6.2.26 #define DPU_CLOCKS**Value:**

```
{
    \_kCLOCK_Dpu0, \_
    \_kCLOCK_Dpu1, \_
}
```

6.2.27 #define LDB_CLOCKS**Value:**

```
{
    \_kCLOCK_Ldb0, \_
    \_kCLOCK_Ldb1 \_
}
```

6.2.28 #define LPCG_TUPLE(rsrc, base) ((uint32_t)((base) >> 12U) << 10U) | rsrc)

The LPCG base should be 4KB aligned, if not it will be truncated.

Function Documentation

6.2.29 `#define LPCG_TUPLE_REG_BASE(tuple) ((volatile uint32_t *)((((uint32_t)(tuple) >> 10U) & 0xFFFFFU) << 12U))`

6.2.30 `#define LPCG_TUPLE_RSRC(tuple) ((sc_rsrc_t)((uint32_t)(tuple)&0x3FFU))`

6.2.31 `#define NV (0U)`

6.3 Enumeration Type Documentation

6.3.1 enum clock_ip_src_t

Enumerator

kCLOCK_IpSrcNone Clock is off.

kCLOCK_IpSrcDummy Clock option 1.

6.3.2 enum clock_name_t

Enumerator

kCLOCK_CoreSysClk Core/system clock for M4.

kCLOCK_Connectivity_AhbClk AHB clock in Connectivity subsystem.

6.3.3 enum clock_ip_name_t

It is defined as the corresponding register address.

6.4 Function Documentation

6.4.1 void CLOCK_Init (sc_ipc_t ipc)

Parameters

<i>ipc</i>	IPC handle for communication with SCU, see sc_ipc_t.
------------	--

6.4.2 bool CLOCK_EnableClockExt (clock_ip_name_t name, uint32_t gate)

Parameters

<i>name</i>	Which clock to enable, see clock_ip_name_t .
<i>gate</i>	0: clock always on, 1: HW auto clock gating.

Returns

true if success, false if failure.

6.4.3 static bool CLOCK_EnableClock (`clock_ip_name_t name`) [inline], [static]

Parameters

<i>name</i>	Which clock to enable, see clock_ip_name_t .
-------------	--

Returns

true for success, false for failure.

6.4.4 bool CLOCK_DisableClock (`clock_ip_name_t name`)

Parameters

<i>name</i>	Which clock to disable, see clock_ip_name_t .
-------------	---

Returns

true for success, false for failure.

6.4.5 uint32_t CLOCK_SetIpFreq (`clock_ip_name_t name, uint32_t freq`)

This function sets the IP module clock frequency.

Function Documentation

Parameters

<i>name</i>	Which peripheral to check, see clock_ip_name_t .
<i>freq</i>	Target clock frequency value in hertz.

Returns

the Real clock frequency value in hertz, or 0 if failed

6.4.6 **uint32_t CLOCK_GetIpFreq ([clock_ip_name_t name](#))**

This function gets the IP module clock frequency.

Parameters

<i>name</i>	Which peripheral to get, see clock_ip_name_t .
-------------	--

Returns

Clock frequency value in hertz, or 0 if failed

6.4.7 **uint32_t CLOCK_GetFreq ([clock_name_t name](#))**

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in [clock_name_t](#).

Parameters

<i>clockName</i>	Clock names defined in clock_name_t
------------------	---

Returns

Clock frequency value in hertz

6.4.8 **uint32_t CLOCK_GetCoreSysClkFreq (void)**

Returns

Clock frequency in Hz.

6.4.9 **void CLOCK_ConfigLPCG (*clock_ip_name_t name*, *uint32_t swGate*,
uint32_t hwGate)**

Function Documentation

Parameters

<i>name</i>	Which clock to enable, see clock_ip_name_t .
<i>swGate</i>	Software clock gating. 0: clock is gated; 1: clock is enabled
<i>swGate</i>	Hardware auto gating. 0: disable the HW clock gate control; 1: HW clock gating is enabled

6.4.10 void CLOCK_SetLpcgGate (volatile uint32_t * *regBase*, uint32_t *swGate*, uint32_t *hwGate*, uint32_t *bitsMask*)

Parameters

<i>regBase</i>	LPCG register base address.
<i>swGate</i>	Software clock gating. 0: clock is gated; 1: clock is enabled
<i>swGate</i>	Hardware auto gating. 0: disable the HW clock gate control; 1: HW clock gating is enabled
<i>bitsMask</i>	The available bits in LPCG register. Each bit indicate the corresponding bit is available or not.

Chapter 7

Debug Console

7.1 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.

7.2 Function groups

7.2.1 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);
```

7.2.2 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

Function groups

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

Function groups

- 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 *

specifier	Qualifying Input	Type of argument
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      /* Select printf, scanf, putchar, getchar of SDK version. */
#define PRINTF            DbgConsole_Printf
#define SCANF              DbgConsole_Scanf
#define PUTCHAR            DbgConsole_Putchar
#define GETCHAR            DbgConsole_Getchar
#else                      /* Select printf, scanf, putchar, getchar of toolchain. */
#define PRINTF            printf
#define SCANF              scanf
#define PUTCHAR            putchar
#define GETCHAR            getchar
#endif /* SDK_DEBUGCONSOLE */
```

7.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

Typical use case

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\r\nTime: %u ticks %2.5f milliseconds\r\nDONE\r", "1 day", 86400, 86.4);
```

Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

Print out failure messages using MCUXpresso SDK __assert_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
    PRINTF("ASSERT ERROR \"% %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
           , line, func);
    for (;;) {}
}
```

Note:

To use 'printf' and 'scanf' for GNUC Base, add file '**fsl_sbrk.c**' in path: ..\{package}\devices\{subset}\utilities\fsl-sbrk.c to your project.

Modules

- [SWO](#)
- /*!
- [Semihosting](#)

Macros

- `#define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U`
Definition select redirect toolchain printf, scanf or not.
- `#define DEBUGCONSOLE_REDIRECT_TO_SDK 1U`
Select SDK version printf, scanf.
- `#define DEBUGCONSOLE_DISABLE 2U`
Disable debugconsole function.
- `#define SDK_DEBUGCONSOLE 1U`
Definition to select sdk or toolchain printf, scanf.
- `#define SDK_DEBUGCONSOLE_UART`
Definition to select redirect toolchain printf, scanf to uart or not.
- `#define PRINTF DbgConsole_Printf`
Definition to select redirect toolchain printf, scanf to uart or not.

Typedefs

- `typedef void(* printfCb)(char *buf, int32_t *indicator, char val, int len)`
A function pointer which is used when format printf log.

Functions

- `int StrFormatPrintf (const char *fmt, va_list ap, char *buf, printfCb cb)`
This function outputs its parameters according to a formatted string.
- `int StrFormatScanf (const char *line_ptr, char *format, va_list args_ptr)`
Converts an input line of ASCII characters based upon a provided string format.

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.
- `int DbgConsole_Printf (const char *formatString,...)`
Writes formatted output to the standard output stream.
- `int DbgConsole_Putchar (int ch)`
Writes a character to stdout.
- `int DbgConsole_Scanf (char *formatString,...)`
Reads formatted data from the standard input stream.
- `int DbgConsole_Getchar (void)`
Reads a character from standard input.
- `status_t DbgConsole_Flush (void)`
Debug console flush.

7.4 Macro Definition Documentation

7.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

Function Documentation

7.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

7.4.3 #define DEBUGCONSOLE_DISABLE 2U

7.4.4 #define SDK_DEBUGCONSOLE 1U

The macro only support to be redefined in project setting.

7.4.5 #define SDK_DEBUGCONSOLE_UART

7.4.6 #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.

7.5 Function Documentation

7.5.1 status_t DbgConsole_Init (uint8_t *instance*, uint32_t *baudRate*, serial_port_type_t *device*, uint32_t *clkSrcFreq*)

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

<i>instance</i>	The instance of the module.
<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">• kSerialPort_Uart,• kSerialPort_UsbCdc.

<i>clkSrcFreq</i>	Frequency of peripheral source clock.
-------------------	---------------------------------------

Returns

Indicates whether initialization was successful or not.

Return values

<i>kStatus_Success</i>	Execution successfully
------------------------	------------------------

7.5.2 **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.

7.5.3 **int DbgConsole_Printf(const char * *formatString*, ...)**

Call this function to write a formatted output to the standard output stream.

Parameters

<i>formatString</i>	Format control string.
---------------------	------------------------

Returns

Returns the number of characters printed or a negative value if an error occurs.

7.5.4 **int DbgConsole_Putchar(int *ch*)**

Call this function to write a character to stdout.

Function Documentation

Parameters

<i>ch</i>	Character to be written.
-----------	--------------------------

Returns

Returns the character written.

7.5.5 int DbgConsole_Scanf (*char * formatString*, ...)

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>formatString</i>	Format control string.
---------------------	------------------------

Returns

Returns the number of fields successfully converted and assigned.

7.5.6 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.

7.5.7 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.

7.5.8 int StrFormatPrintf (const char * *fmt*, va_list *ap*, char * *buf*, printfCb *cb*)

Note

I/O is performed by calling given function pointer using following (*func_ptr)(c);

Parameters

in	<i>fmt</i>	Format string for printf.
in	<i>ap</i>	Arguments to printf.
in	<i>buf</i>	pointer to the buffer
	<i>cb</i>	print callbk function pointer

Returns

Number of characters to be print

7.5.9 int StrFormatScanf (const char * *line_ptr*, char * *format*, va_list *args_ptr*)

Parameters

in	<i>line_ptr</i>	The input line of ASCII data.
in	<i>format</i>	Format first points to the format string.
in	<i>args_ptr</i>	The list of parameters.

Returns

Number of input items converted and assigned.

Function Documentation

Return values

<i>IO_EOF</i>	When line_ptr is empty string "".
---------------	-----------------------------------

7.6 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

7.6.1 Guide Semihosting for IAR

NOTE: After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the `SDK_DEBUGCONSOLE` is disabled.

Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

Step 3: Starting semihosting

1. Choose "Semihosting_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting.
1. Make sure the `SDK_DEBUGCONSOLE_UART` is not defined, remove the default definition in `fsl_debug_console.h`.
1. Start the project by choosing Project>Download and Debug.
2. Choose View>Terminal I/O to display the output from the I/O operations.

7.6.2 Guide Semihosting for Keil µVision

NOTE: Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

7.6.3 Guide Semihosting for MCUXpresso IDE

Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK_DEBUGCONSOLE=0, if set SDK_DEBUGCONSOLE=1, the log will be redirect to the UART.

Step 2: Building the project

1. Compile and link the project.

Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

7.6.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
 - "Host Name (or IP address)" : localhost
 - "Port" :2333
 - "Connection type" : Telet.
 - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__stack_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__stack_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --defsym=__heap_size__=0x2000")
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} --defsym=__heap_size__=0x2000")
```

Step 2: Building the project

1. Change "CMakeLists.txt":

```
Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=nano.specs")"
to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=rdimon.specs")"
```

Replace paragraph

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
```

To

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
```

Remove

```
target_link_libraries(semihosting_ARMGCC.elf debug nosys)
```

2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

- (a) Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

- (b) After the setting, press "enter". The PuTTY window now shows the printf() output.

7.7 SWO

/*!

Serial wire output is a mechanism for ARM targets to output signal from core through a single pin. Some IDE also support SWO, such IAR and KEIL, both input and output are supported, reference below for detail.

7.7.1 Guide SWO for SDK

NOTE: After the setting both "printf" and "PRINTF" are available for debugging, JlinkSWOViewer can be used to capture the output log.

Step 1: Setting up the environment

1. Define SERIAL_PORT_TYPE_SWO in your project settings.
2. Prepare code, the port and baudrate can be decided by application, clkSrcFreq should be mcu core clock frequency:

```
DbgConsole_Init(instance, baudRate, SERIAL_PORT_TYPE_SWO, clkSrcFreq);
```

3. Use PRINTF or printf to print some thing in application.

Step 2: Building the project

Step 3: Download and run project

7.7.1.1 Guide SWO for IAR

NOTE: After the setting both "printf" and "scanf" are available for debugging.

Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK, and click the Activate button to enable trace data collection.

SWO

6. There are three cases for this SDK_DEBUGCONSOLE_UART whether or not defined. a: if use uppercase PRINTF to output log,The SDK_DEBUGCONSOLE_UART defined or not defined will not effect debug function. b: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to zero,then debug function ok. c: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to one,then debug function ok.

NOTE: Case a or c only apply at example which enable swo function,the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h. For case a and c, Do and not do the above third step will be not affect function.

1. Start the project by choosing Project>Download and Debug.

Step 2: Building the project

Step 3: Starting swo

1. Download and debug application.
2. Choose View -> Terminal I/O to display the output from the I/O operations.
3. Run application.

7.7.2 Guide SWO for Keil µVision

NOTE: After the setting both "printf" and "scanf" are available for debugging.

Step 1: Setting up the environment

1. There are three cases for this SDK_DEBUGCONSOLE_UART whether or not defined. a: if use uppercase PRINTF to output log,the SDK_DEBUGCONSOLE_UART definition does not affect the functionality and skip the second step directly. b: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to zero,then start the second step. c: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to one,then skip the second step directly.

NOTE: Case a or c only apply at example which enable swo function,the SDK_DEBUGCONSOLE_UART definition in fsl_debug_console.h.

1. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
2. Open Project>Options for target or using Alt+F7 or click.
3. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
4. Select “Debug” tab and choose Port:SW, then select “Trace” tab, choose “Enable” and click OK, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7.

Step 4: Run the project

1. Choose "Debug" on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.

7.7.3 Guide SWO for MCUXpresso IDE

NOTE: MCUX support SWO for LPC-Link2 debug probe only.

7.7.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 8

Display Processing Unit (DPU)

8.1 Overview

The SDK provides a peripheral driver for the DPU.

The DPU module consists of many processing units, such as FetchDecode, LayerBlend, and so on. The SDK DPU driver provides separate functions for these processing units.

For a processing unit, there are three kinds of functions:

1. The initialize functions. These functions are named as DPU_InitXxx. For example, [DPU_InitStore](#). These functions should only be used before display started to initialize the processing units.
2. The configure functions. These functions are named as DPU_XxxSetYyyConfig. For example, [DPU_SetStoreDstBufferConfig](#). These functions can be used before the display starts to setup configuration. Additionally, they can be used after the display starts to make some runtime changes.
3. The function to get default configuration.

In the DPU driver, the pipeline is also treated as a processing unit. For example, the unit kDPU_Pipeline-ExtDst0 means the pipeline with unit ExtDst0 as its endpoint. Accordingly, there are functions to initialize the pipeline and configure the pipeline.

8.2 Program model

The DPU module provides the shadow registers. The software can write to shadow registers instead of to the active configuration. When a new configuration is completed, the software can trigger the shadowed configuration to be the active configuration.

The DPU driver uses this feature. The shadow load function is enabled during the unit initialization. After all configurations in a pipeline are finished, the function [DPU_TriggerPipelineShadowLoad](#) can be called to activate the shadowed configurations. After this, the upper layer should monitor the interrupt status to make sure the shadow load is finished before a new configuration.

The program workflow is like this:

Program model

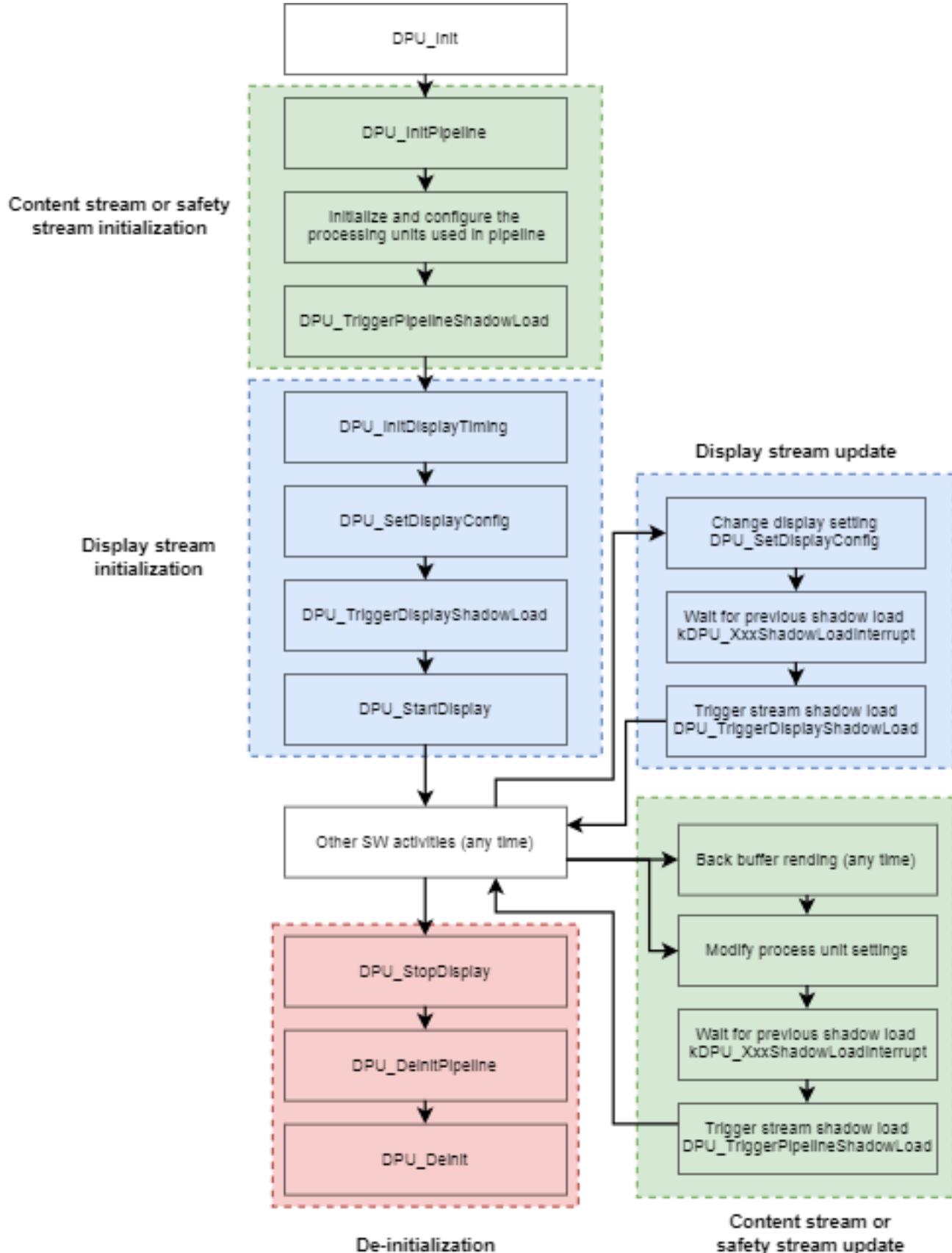


Figure 8.2.1: DPU display workflow
MCUXpresso SDK API Reference Manual

For the blot engine, the driver supports two kinds of methods.

Method 1: Configure and start operation when a previous process finishes. The software workflow is:

1. Configure the blot engine units.
2. Trigger the blot engine pipeline shadow load using [DPU_TriggerPipelineShadowLoad](#).
3. Start the process using [DPU_StartStore](#).
4. Monitor the DPU store frame complete interrupt.
5. Repeat from step 1 for a new process.

The workflow flow is:

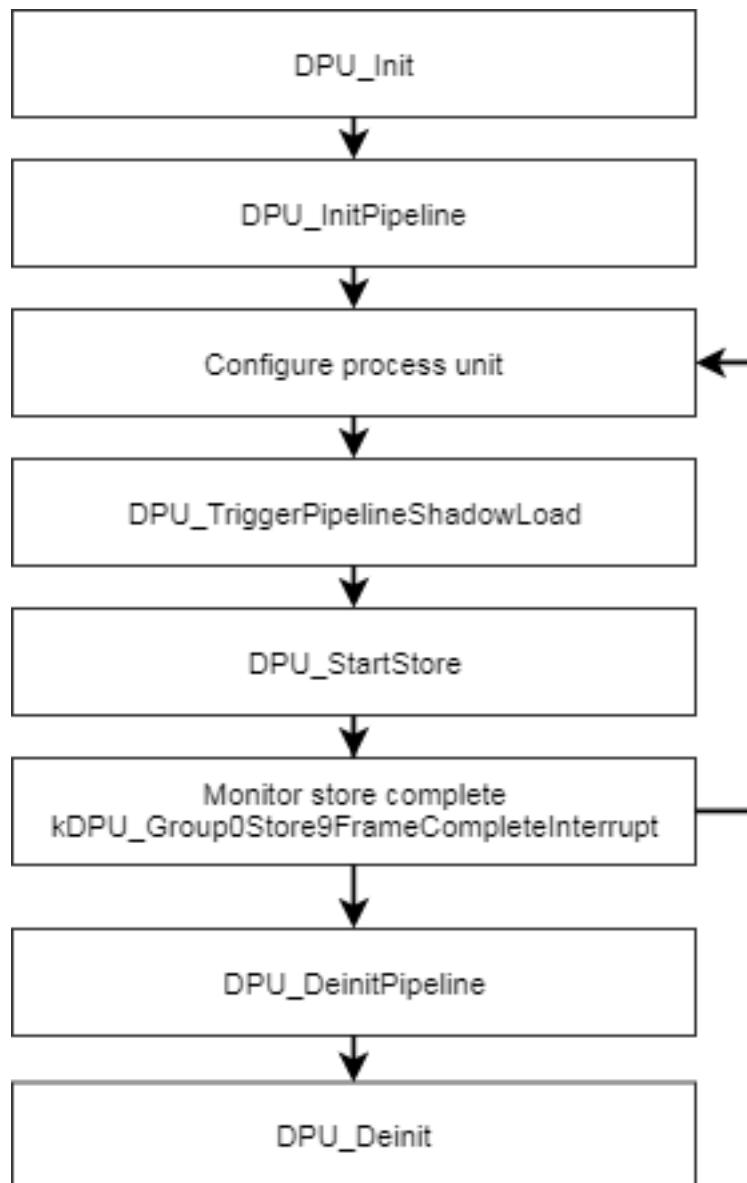


Figure 8.2.2: DPU blot engine workflow 1

Program model

Method 2: Set the new configuration when the previous process is still on-going. In this case, the software cannot use the store frame complete interrupt to make sure all processes are finished because the software cannot distinguish which frame completion asserts this interrupt. The function [DPU_TriggerPipelineCompleteInterrupt](#) should be used in this case. The workflow is:

1. Configure the blot engine units.
2. Trigger the blot engine pipeline shadow load using [DPU_TriggerPipelineShadowLoad](#).
3. Start process using [DPU_StartStore](#).
4. Monitor the DPU pipeline shadow load interrupt.
5. If there is new process, then repeat from step 1.
6. If there is not a new process or the software wants to make sure all processes are finished, call [DPU_TriggerPipelineCompleteInterrupt](#) and monitor the pipeline sequence complete interrupt.

The workflow flow is:

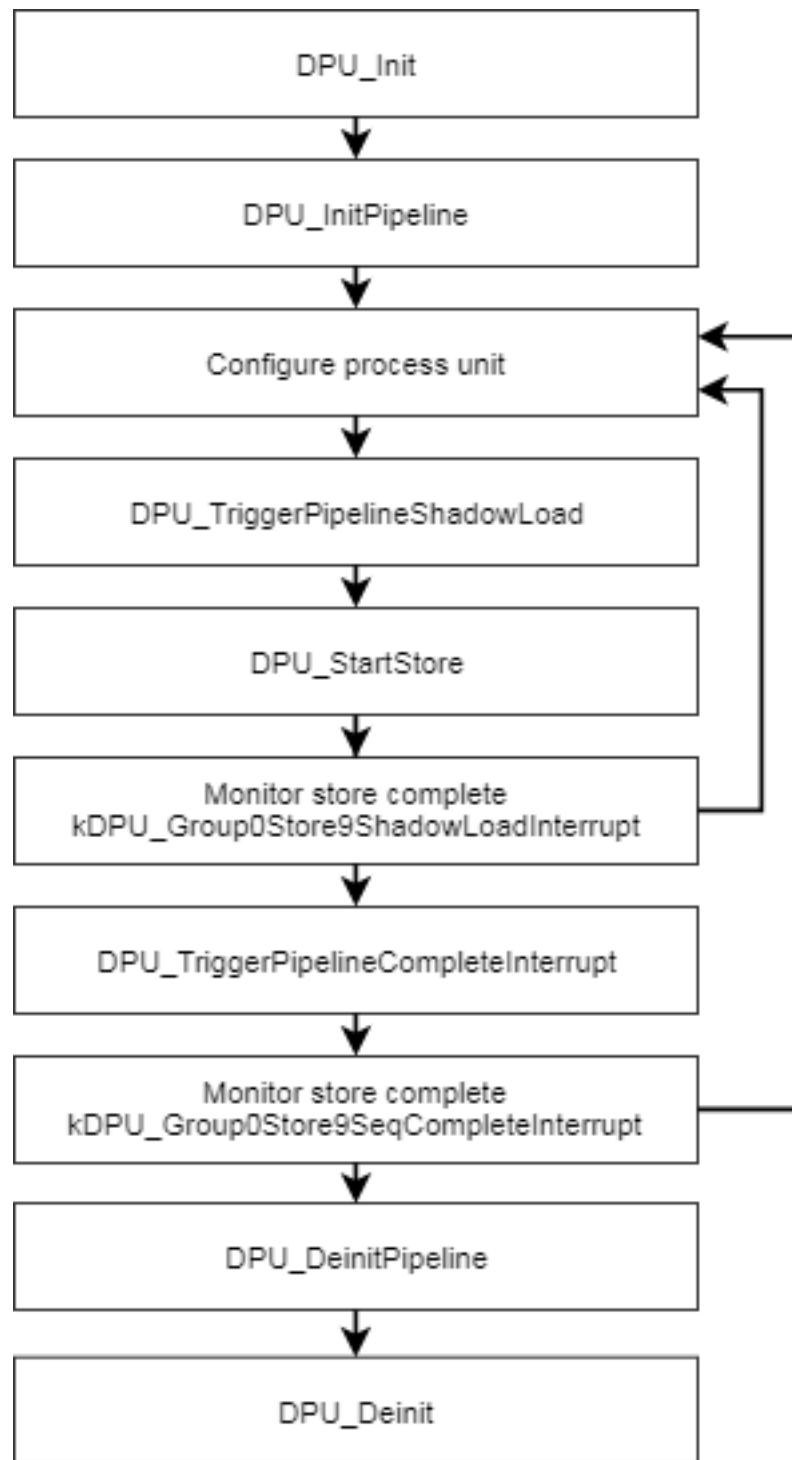


Figure 8.2.3: DPU blit engine workflow 2

Path configuration

8.3 Path configuration

The DPU consists of many processing units. The pipeline path should be configured carefully for special use cases.

The blot engine diagram is:

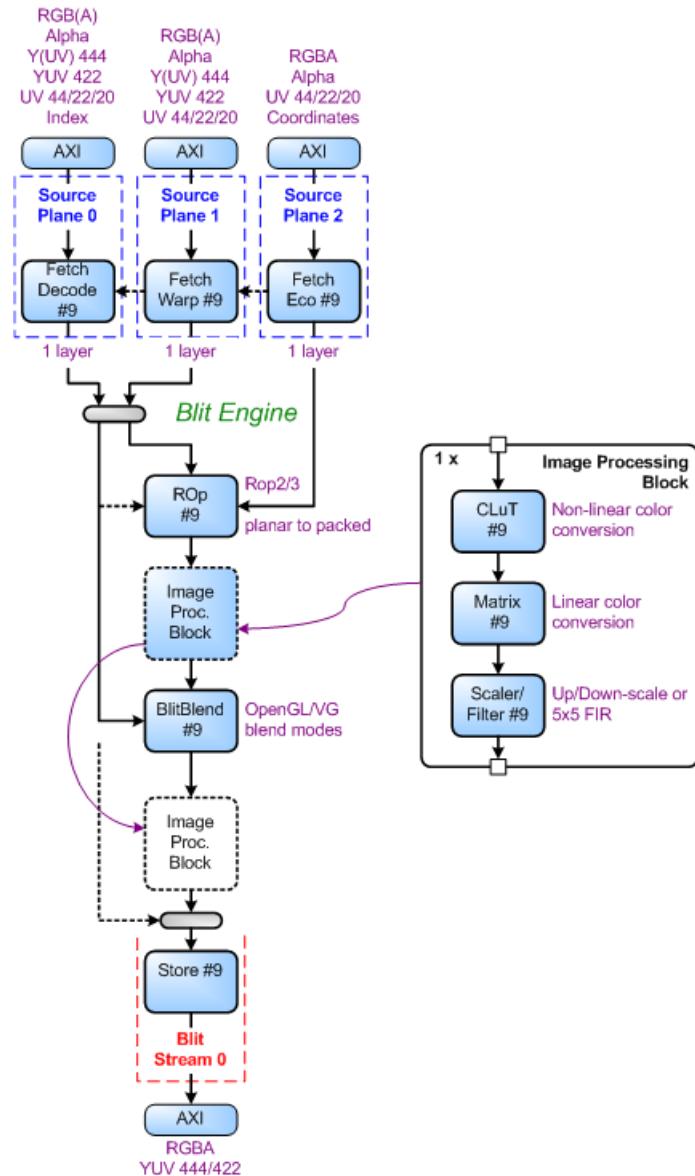


Figure 8.3.1: Blot Engine Block Diagram

The display controller block diagram is:

Path configuration

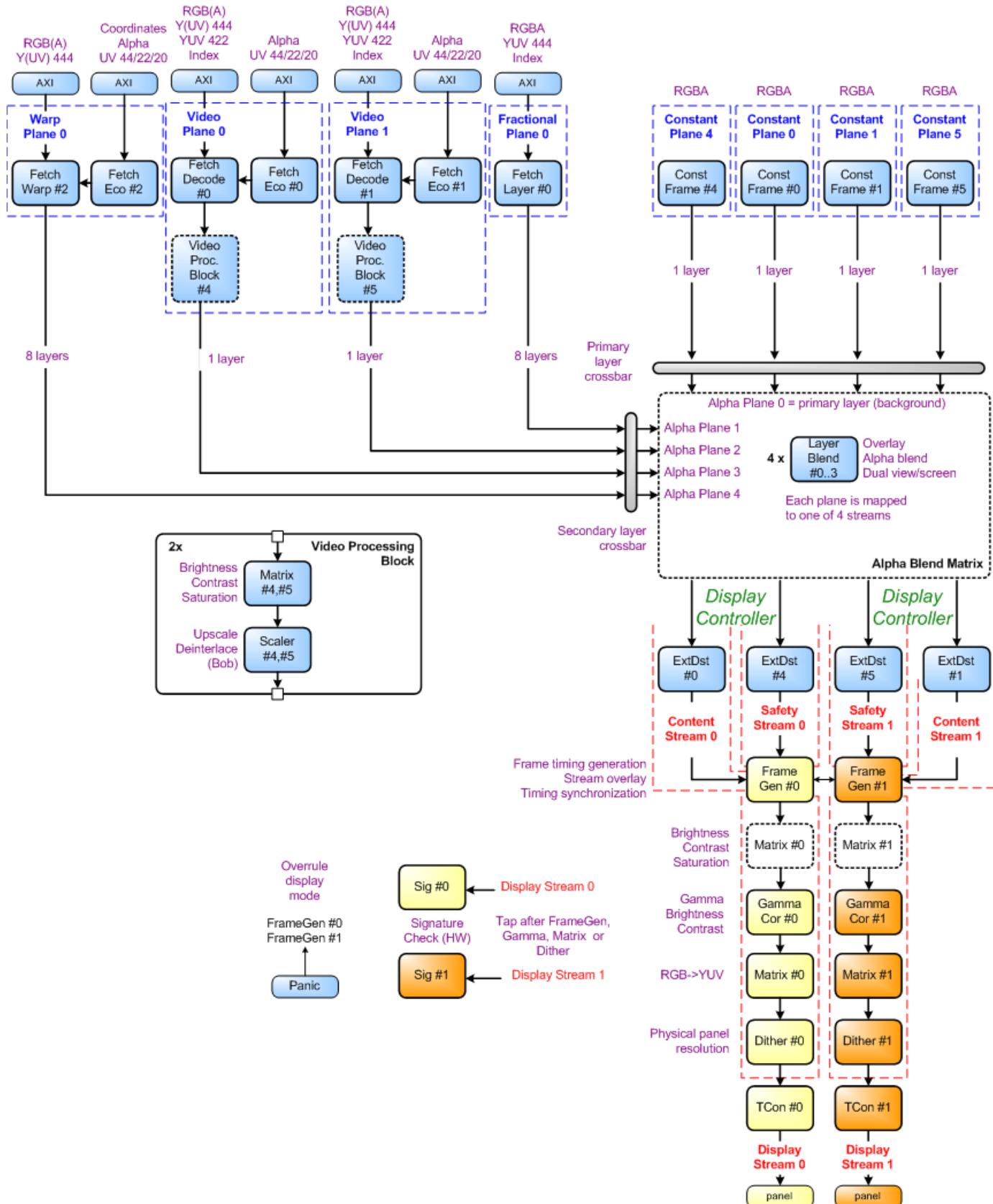


Figure 8.3.2: Display Controller Block Diagram
MCUXpresso SDK API Reference Manual

Path configuration

Processing units have their primary input (named src or prim) connected to the top side and their secondary input port (named sec), if present, connected to the left or right side in the diagram. The ROP#9 unit has its secondary input right and tertiary (named tert) left side connected.

Note

An active unit must at least have its primary port connected, while secondary and tertiary ports are optional

Note

When both horizontal and vertical scaling is active, then the sequence of both units in the Pixelbus configuration should be

- > HScaler -> VScaler -> when down-scaling horizontally
- > VScaler -> HScaler -> when up-scaling horizontally

The default path configuration after reset is:

Path configuration

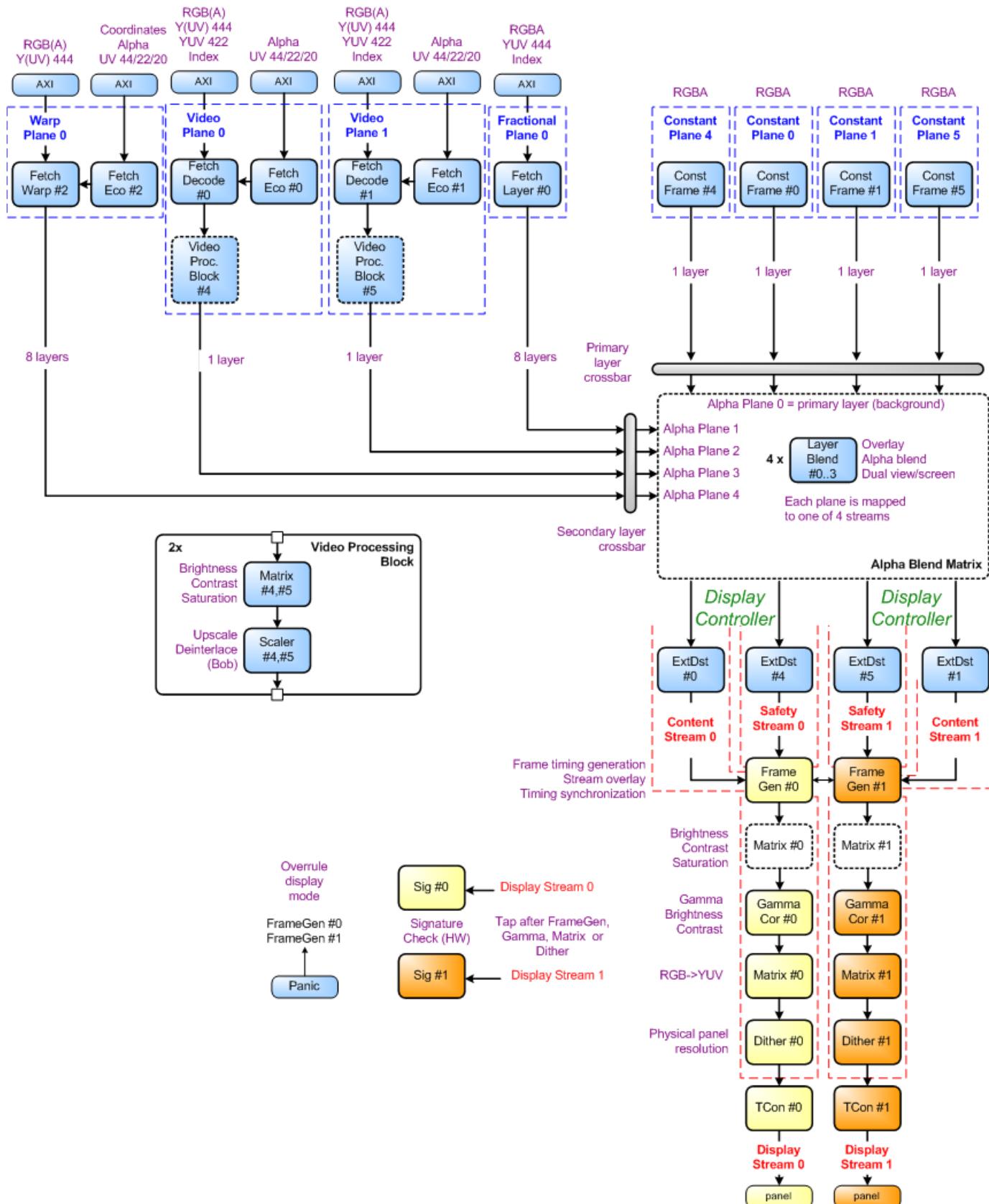


Figure 8.3.3: Default Path Configuration
MCUXpresso SDK API Reference Manual

Path configuration

Data Structures

- struct `dpu_fetch_unit_config_t`
Configuration structure for fetch units. [More...](#)
- struct `dpu_coordinates_config_t`
Configuration structure for the arbitrary warping re-sampling coordinates. [More...](#)
- struct `dpu_warp_config_t`
Warp configuration structure for FetchWarp unit. [More...](#)
- struct `dpu_src_buffer_config_t`
Fetch unit source buffer configuration structure. [More...](#)
- struct `dpu_clip_window_config_t`
Fetch unit clip window configuration structure. [More...](#)
- struct `dpu_dst_buffer_config_t`
Store unit Destination buffer configuration structure. [More...](#)
- struct `dpu_layer_blend_config_t`
LayerBlend unit configuration structure. [More...](#)
- struct `dpu_blt_blend_config_t`
BlitBlend unit configuration structure. [More...](#)
- struct `dpu_rop_config_t`
ROp unit configuration structure. [More...](#)
- struct `dpu_const_frame_config_t`
ConstFrame unit configuration structure. [More...](#)
- struct `dpu_display_timing_config_t`
Display timing configuration structure. [More...](#)
- struct `dpu_display_config_t`
Display mode configuration structure. [More...](#)
- struct `dpu_scaler_config_t`
VScaler and HScaler configuration structure. [More...](#)
- struct `dpu_signature_config_t`
Signature unit static configuration. [More...](#)
- struct `dpu_signature_window_config_t`
Signature unit evaluation window configuration. [More...](#)

Macros

- #define `DPU_PALETTE_ENTRY_NUM` (256)
DPU palette entry number.
- #define `DPU_MAKE_CONST_COLOR`(red, green, blue, alpha) (((uint32_t)(red)) << 24U) |
((uint32_t)(green)) << 16U) | ((uint32_t)(blue)) << 8U) | ((uint32_t)(alpha)))
Define the const value that write to <unit>_ConstantColor.

Enumerations

- enum `dpu_unit_t`
DPU units.
- enum `_dpu_interrupt` {

```

kDPU_Group0Store9ShadowLoadInterrupt = (1U << 0U),
kDPU_Group0Store9FrameCompleteInterrupt = (1U << 1U),
kDPU_Group0Store9SeqCompleteInterrupt = (1U << 2U),
kDPU_Group0ExtDst0ShadowLoadInterrupt = (1U << 3U),
kDPU_Group0ExtDst0FrameCompleteInterrupt = (1U << 4U),
kDPU_Group0ExtDst0SeqCompleteInterrupt = (1U << 5U),
kDPU_Group0ExtDst4ShadowLoadInterrupt = (1U << 6U),
kDPU_Group0ExtDst4FrameCompleteInterrupt = (1U << 7U),
kDPU_Group0ExtDst4SeqCompleteInterrupt = (1U << 8U),
kDPU_Group0ExtDst1ShadowLoadInterrupt = (1U << 9U),
kDPU_Group0ExtDst1FrameCompleteInterrupt = (1U << 10U),
kDPU_Group0ExtDst1SeqCompleteInterrupt = (1U << 11U),
kDPU_Group0ExtDst5ShadowLoadInterrupt = (1U << 12U),
kDPU_Group0ExtDst5FrameCompleteInterrupt = (1U << 13U),
kDPU_Group0ExtDst5SeqCompleteInterrupt = (1U << 14U),
kDPU_Group0Display0ShadowLoadInterrupt = (1U << 15U),
kDPU_Group0Display0FrameCompleteInterrupt = (1U << 16U),
kDPU_Group0Display0SeqCompleteInterrupt = (1U << 17U),
kDPU_Group0FrameGen0Int0Interrupt = (1U << 18U),
kDPU_Group0FrameGen0Int1Interrupt = (1U << 19U),
kDPU_Group0FrameGen0Int2Interrupt = (1U << 20U),
kDPU_Group0FrameGen0Int3Interrupt = (1U << 21U),
kDPU_Group0Sig0ShadowLoadInterrupt = (1U << 22U),
kDPU_Group0Sig0ValidInterrupt = (1U << 23U),
kDPU_Group0Sig0ErrorInterrupt = (1U << 24U),
kDPU_Group0Display1ShadowLoadInterrupt = (1U << 25U),
kDPU_Group0Display1FrameCompleteInterrupt = (1U << 26U),
kDPU_Group0Display1SeqCompleteInterrupt = (1U << 27U),
kDPU_Group0FrameGen1Int0Interrupt = (1U << 28U),
kDPU_Group0FrameGen1Int1Interrupt = (1U << 29U),
kDPU_Group0FrameGen1Int2Interrupt = (1U << 30U),
kDPU_Group0FrameGen1Int3Interrupt = (1U << 31U),
kDPU_Group1Sig1ShadowLoadInterrupt = (1U << 0U),
kDPU_Group1Sig1ValidInterrupt = (1U << 1U),
kDPU_Group1Sig1ErrorInterrupt = (1U << 2U),
kDPU_Group1CmdSeqErrorInterrupt = (1U << 4U),
kDPU_Group1SoftwareInt0Interrupt = (1U << 5U),
kDPU_Group1SoftwareInt1Interrupt = (1U << 6U),
kDPU_Group1SoftwareInt2Interrupt = (1U << 7U),
kDPU_Group1SoftwareInt3Interrupt = (1U << 8U),
kDPU_Group1FrameGen0PrimSyncOnInterrupt = (1U << 9U),
kDPU_Group1FrameGen0PrimSyncOffInterrupt = (1U << 10U),
kDPU_Group1FrameGen0SecSyncOnInterrupt = (1U << 11U),
kDPU_Group1FrameGen0SecSyncOffInterrupt = (1U << 12U),
kDPU_Group1FrameGen1PrimSyncOnInterrupt = (1U << 13U),
kDPU_Group1FrameGen1PrimSyncOffInterrupt = (1U << 14U),
kDPU_Group1FrameGen1SecSyncOnInterrupt = (1U << 15U)

```

Path configuration

```
kDPU_Group1FrameGen1SecSyncOffInterrupt = (1U << 16U) }  
    DPU interrupt.  
• enum _dpu_unit_source {  
    kDPU_UnitSrcNone = 0,  
    kDPU_UnitSrcFetchDecode9 = 1U,  
    kDPU_UnitSrcFetchWarp9 = 2U,  
    kDPU_UnitSrcFetchEco9 = 3U,  
    kDPU_UnitSrcRop9 = 4U,  
    kDPU_UnitSrcClut9 = 5U,  
    kDPU_UnitSrcMatrix9 = 6U,  
    kDPU_UnitSrcHScaler9 = 7U,  
    kDPU_UnitSrcVScaler9 = 8U,  
    kDPU_UnitSrcFilter9 = 9U,  
    kDPU_UnitSrcBlitBlend9 = 10U,  
    kDPU_UnitSrcStore9 = 11U,  
    kDPU_UnitSrcConstFrame0 = 12U,  
    kDPU_UnitSrcConstFrame1 = 16U,  
    kDPU_UnitSrcConstFrame4 = 14U,  
    kDPU_UnitSrcConstFrame5 = 18U,  
    kDPU_UnitSrcFetchWarp2 = 20U,  
    kDPU_UnitSrcFetchEco2 = 21U,  
    kDPU_UnitSrcFetchDecode0 = 22U,  
    kDPU_UnitSrcFetchEco0 = 23U,  
    kDPU_UnitSrcFetchDecode1 = 24U,  
    kDPU_UnitSrcFetchEco1 = 25U,  
    kDPU_UnitSrcFetchLayer0 = 26U,  
    kDPU_UnitSrcMatrix4 = 27U,  
    kDPU_UnitSrcHScaler4 = 28U,  
    kDPU_UnitSrcVScaler4 = 29U,  
    kDPU_UnitSrcMatrix5 = 30U,  
    kDPU_UnitSrcHScaler5 = 31U,  
    kDPU_UnitSrcVScaler5 = 32U,  
    kDPU_UnitSrcLayerBlend0 = 33U,  
    kDPU_UnitSrcLayerBlend1 = 34U,  
    kDPU_UnitSrcLayerBlend2 = 35U,  
    kDPU_UnitSrcLayerBlend3 = 36U }  
    DPU unit input source.  
• enum dpu_pixel_format_t {  
    kDPU_PixelFormatGray8 = 0,  
    kDPU_PixelFormatRGB565 = 1,  
    kDPU_PixelFormatARGB8888 = 2,  
    kDPU_PixelFormatRGB888 = 3,  
    kDPU_PixelFormatARGB1555 = 4 }  
    DPU pixel format.  
• enum dpu_warp_coordinate_mode_t {
```

```
kDPU_WarpCoordinateModePNT = 0U,
kDPU_WarpCoordinateModeDPNT = 1U,
kDPU_WarpCoordinateModeDDPNT = 2U }
```

FetchWarp unit warp coordinate mode.

- enum `dpu_clip_color_mode_t` {
 `kDPU_ClipColorNull`,
 `kDPU_ClipColorSublayer` }

Define the color to take for pixels that do not lie inside the clip window of any layer.

- enum `dpu_alpha_mask_mode_t` {
 `kDPU_AlphaMaskPrim`,
 `kDPU_AlphaMaskSec`,
 `kDPU_AlphaMaskPrimOrSec`,
 `kDPU_AlphaMaskPrimAndSec`,
 `kDPU_AlphaMaskPrimInv`,
 `kDPU_AlphaMaskSecInv`,
 `kDPU_AlphaMaskPrimOrSecInv`,
 `kDPU_AlphaMaskPrimAndSecInv` }

LayerBlend unit AlphaMask mode.

- enum `dpu_blend_mode_t` {
 `kDPU_BlendZero`,
 `kDPU_BlendOne`,
 `kDPU_BlendPrimAlpha`,
 `kDPU_BlendPrimAlphaInv`,
 `kDPU_BlendSecAlpha`,
 `kDPU_BlendSecAlphaInv`,
 `kDPU_BlendConstAlpha`,
 `kDPU_BlendConstAlphaInv` }

LayerBlend unit alpha blend mode.

- enum `dpu_blt_blend_func_t` {
 `kDPU_BlitBlendFuncGlZero` = 0,
 `kDPU_BlitBlendFuncGlOne` = 1,
 `kDPU_BlitBlendFuncGlSrcColor` = 0x0300,
 `kDPU_BlitBlendFuncGlOneMinusSrcColor` = 0x0301,
 `kDPU_BlitBlendFuncGlSrcAlpha` = 0x0302,
 `kDPU_BlitBlendFuncGlOneMinusSrcAlpha` = 0x0303,
 `kDPU_BlitBlendFuncGlDstAlpha` = 0x0304,
 `kDPU_BlitBlendFuncGlOneMinusDstAlpha` = 0x0305,
 `kDPU_BlitBlendFuncGlDstColor` = 0x0306,
 `kDPU_BlitBlendFuncGlOneMinusDstColor` = 0x0307,
 `kDPU_BlitBlendFuncGlSrcAlphaSaturate` = 0x0308,
 `kDPU_BlitBlendFuncGlConstColor` = 0x8001,
 `kDPU_BlitBlendFuncGlOneMinusConstColor` = 0x8002,
 `kDPU_BlitBlendFuncGlConstAlpha` = 0x8003,
 `kDPU_BlitBlendFuncGlOneMinusConstAlpha` = 0x8004 }

BlitBlend blend function.

- enum `dpu_blt_blend_mode_t` {

Path configuration

```
kDPU_BlitBlendModeGlFuncAdd = 0x8006,  
kDPU_BlitBlendModeGlMin = 0x8007,  
kDPU_BlitBlendModeGlMax = 0x8008,  
kDPU_BlitBlendModeGlFuncSubtract = 0x800A,  
kDPU_BlitBlendModeGlFuncReverseSubtract = 0x800B,  
kDPU_BlitBlendModeVgBlendSrc = 0x2000,  
kDPU_BlitBlendModeVgBlendSrcOver = 0x2001,  
kDPU_BlitBlendModeVgBlendDstOver = 0x2002,  
kDPU_BlitBlendModeVgBlendSrcIn = 0x2003,  
kDPU_BlitBlendModeVgBlendDstIn = 0x2004,  
kDPU_BlitBlendModeVgBlendMultiply = 0x2005,  
kDPU_BlitBlendModeVgBlendScreen = 0x2006,  
kDPU_BlitBlendModeVgBlendDarken = 0x2007,  
kDPU_BlitBlendModeVgBlendLighten = 0x2008,  
kDPU_BlitBlendModeVgBlendAdditive = 0x2009 }
```

BlitBlend blend mode.

- enum `dpu_blt_blend_neutral_border_mode_t` {
 `kDPU_BlitBlendNeutralBorderPrim` = 0,
 `kDPU_BlitBlendNeutralBorderSec` = 0 }

BlitBlend neutral border mode.

- enum `_dpu_rop_flags` {
 `kDPU_RopAddRed`,
 `kDPU_RopAddGreen`,
 `kDPU_RopAddBlue`,
 `kDPU_RopAddAlpha`,
 `kDPU_RopTertDiv2` = `DPU_ROP_CONTROL_TertDiv2_MASK`,
 `kDPU_RopSecDiv2` = `DPU_ROP_CONTROL_SecDiv2_MASK`,
 `kDPU_RopPrimDiv2` = `DPU_ROP_CONTROL_PrimDiv2_MASK` }

ROp unit control flags.

- enum `_dpu_display_timing_flags` {
 `kDPU_DisplayPixelActiveHigh` = 0,
 `kDPU_DisplayPixelActiveLow` = `DPU_DISENGCONF_POLARITYCTRL_PixInv_MASK`,
 `kDPU_DisplayDataEnableActiveHigh`,
 `kDPU_DisplayDataEnableActiveLow` = 0,
 `kDPU_DisplayHsyncActiveHigh` = `DPU_DISENGCONF_POLARITYCTRL_PolHs_MASK`,
 `kDPU_DisplayHsyncActiveLow` = 0,
 `kDPU_DisplayVsyncActiveHigh` = `DPU_DISENGCONF_POLARITYCTRL_PolVs_MASK`,
 `kDPU_DisplayVsyncActiveLow` = 0 }

Display timing configuration flags.

- enum `dpu_display_mode_t` {
 `kDPU_DisplayBlackBackground`,
 `kDPU_DisplayConstBackground`,
 `kDPU_DisplayOnlyPrim`,
 `kDPU_DisplayOnlySec`,
 `kDPU_DisplayPrimOnTop`,
 `kDPU_DisplaySecOnTop`,

```
kDPU_DisplayTest }
```

Display mode, safety stream is the primary input, content stream is the secondary input.

- enum `_dpu_signature_window_flags` {

 `kDPU_SignatureWindowEnableGlobalPanic` = DPU_SIG_EVALCONTROL_EnGlobalPanic_MASK,

 `kDPU_SignatureWindowEnableLocalPanic` = DPU_SIG_EVALCONTROL_EnLocalPanic_MASK,

 `kDPU_SignatureWindowEnableAlphaMask` = DPU_SIG_EVALCONTROL_AlphaMask_MASK,

 `kDPU_SignatureWindowInvertAlpha` = DPU_SIG_EVALCONTROL_AlphaInv_MASK }

 Signature unit evaluation window control flags.
- enum `_dpu_signature_status` {

 `kDPU_SignatureIdle` = DPU_SIG_STATUS_StsSigIdle_MASK,

 `kDPU_SignatureValid` = DPU_SIG_STATUS_StsSigValid_MASK }

 Signature unit status.

Driver version

- #define `FSL_DPU_DRIVER_VERSION` (MAKE_VERSION(2, 0, 0))

DPU driver version 2.0.0.

Macros for the DPU unit input source.

The DPU unit input source is controlled by the register `pixencfg_<unit>_dynamic`, the macros `DPU_MAKE_SRC_REG1`, `DPU_MAKE_SRC_REG2`, and `DPU_MAKE_SRC_REG3` are used to define the register value of `pixencfg_<unit>_dynamic`.

`DPU_MAKE_SRC_REG1` defines register for DPU unit that has one input source. Accordingly, `DPU_MAKE_SRC_REG2` and `DPU_MAKE_SRC_REG3` are used to define the register for units that have two and three input source. See `_dpu_unit_source` for the input source details.

- #define `DPU_MAKE_SRC_REG1`(src) (((uint32_t)(src)) & 0x3FU)

Macro for one input source unit.
- #define `DPU_MAKE_SRC_REG2`(primSrc, secSrc) (((((uint32_t)(primSrc)) & 0x3FU) | (((uint32_t)(secSrc)) & 0x3FU) << 0x8U)))

Macro for two input source unit.
- #define `DPU_MAKE_SRC_REG3`(primSrc, secSrc, tertSrc)

Macro for three input source unit.

Macros define the FrameGen interrupt mode.

These macros are used by the function `DPU_SetFrameGenInterruptConfig` to set the FrameGen interrupt mode.

- #define `DPU_FRAME_GEN_INT_DISABLE` 0U

Disable FrameGen interrupt.
- #define `DPU_FRAME_GEN_INT_PER_LINE`(colNum) ((1U << 31U) | (1U << 15U) | (((uint32_t)colNum) & (0x3FFFU)))

Generate FrameGen interrupt every line at the column colNum.
- #define `DPU_FRAME_GEN_INT_PER_FRAME`(rowNum) ((1U << 31U) | (((uint32_t)rowNum) & 0x3FFF0000U))

Generate FrameGen interrupt every frame at the row rowNum.

Path configuration

Generate FrameGen interrupt every frame at the row rowNum.

DPU Initialization and de-initialization

- void [DPU_Init](#) (IRIS_MVPL_Type *base)
Initializes the DPU peripheral.
- void [DPU_Deinit](#) (IRIS_MVPL_Type *base)
Deinitializes the DPU peripheral.
- void [DPU_PrepPathConfig](#) (IRIS_MVPL_Type *base)
Prepare the unit path configuration.

DPU interrupts

- void [DPU_EnableInterrupts](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Enable the selected DPU interrupts.
- void [DPU_DisableInterrupts](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Disable the selected DPU interrupts.
- uint32_t [DPU_GetInterruptsPendingFlags](#) (IRIS_MVPL_Type *base, uint8_t group)
Get the DPU interrupts pending status.
- void [DPU_ClearInterruptsPendingFlags](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Clear the specified DPU interrupts pending status.
- void [DPU_SetInterruptsPendingFlags](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Set the specified DPU interrupts pending status.
- void [DPU_MaskUserInterrupts](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Mask the selected DPU user interrupts.
- void [DPU_EnableUserInterrupts](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Enable the selected DPU user interrupts.
- void [DPU_DisableUserInterrupts](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Disable the selected DPU user interrupts.
- uint32_t [DPU.GetUserInterruptsPendingFlags](#) (IRIS_MVPL_Type *base, uint8_t group)
Get the DPU user interrupts pending status.
- void [DPU_ClearUserInterruptsPendingFlags](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Clear the specified DPU user interrupts pending status.
- void [DPU_SetUserInterruptsPendingFlags](#) (IRIS_MVPL_Type *base, uint8_t group, uint32_t mask)
Set the specified DPU user interrupts pending status.

Shadow load related.

- status_t [DPU_EnableShadowLoad](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, bool enable)
Enable or disable the register shadowing for the DPU process units.

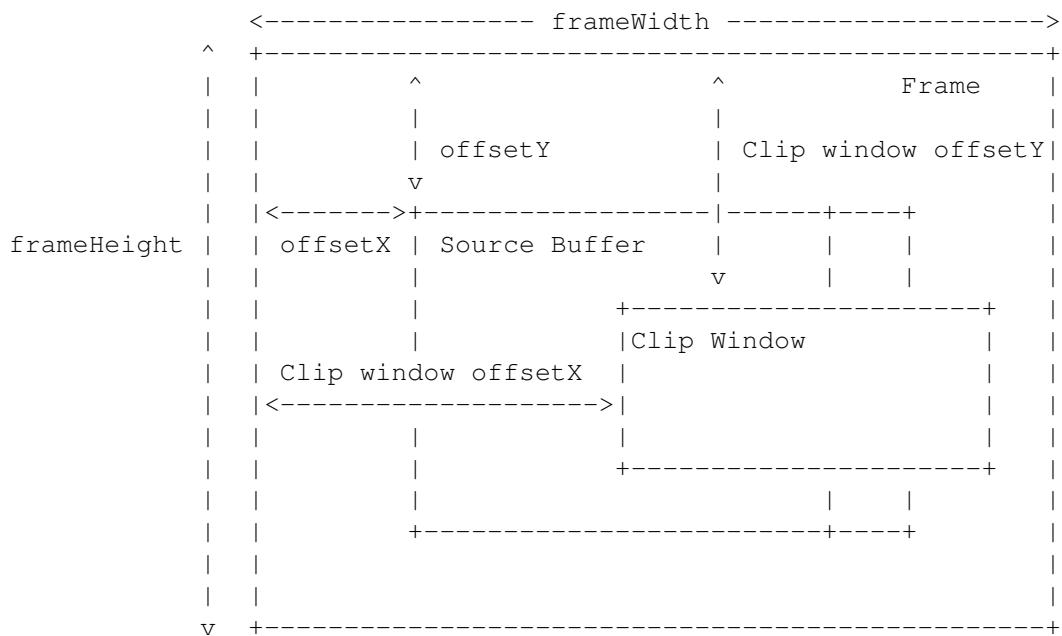
Pipline.

- void [DPU_InitPipeline](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit)
Initialize the pipeline.
- void [DPU_DeinitPipeline](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit)
Deinitializes the pipeline.
- void [DPU_TriggerPipelineShadowLoad](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit)

- void [DPU_TriggerPipelineCompleteInterrupt](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit)
Trigger the pipeline.
 - void [DPU_SetUnitSrc](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, uint32_t srcReg)
Set the DPU unit input source selection.

Fetch Units

The Fetch unit input frame buffer is used like this:



- void **DPU_FetchUnitGetDefaultConfig** (**dpu_fetch_unit_config_t** *config)
Get the default configuration for fetch unit.
 - void **DPU_InitFetchUnit** (**IRIS_MVPL_Type** *base, **dpu_unit_t** unit, const **dpu_fetch_unit_config_t** *config)
Initialize the fetch unit.
 - status_t **DPU_SetColorPaletteIndexWidth** (**IRIS_MVPL_Type** *base, **dpu_unit_t** unit, **uint8_t** indexWidth)
Set the color palette index width for fetch unit.
 - status_t **DPU_UpdateColorPalette** (**IRIS_MVPL_Type** *base, **dpu_unit_t** unit, **uint32_t** startIndex, const **uint32_t** *palette, **uint32_t** count)
Updates the color palette for fetch unit.
 - void **DPU_EnableColorPalette** (**IRIS_MVPL_Type** *base, **dpu_unit_t** unit, **uint8_t** sublayer, **bool** enable)
Enable or disable color palette for some sublayer.
 - void **DPU_CorrdinatesGetDefaultConfig** (**dpu_coordinates_config_t** *config)
Get the default configuration structure for arbitrary warping re-sampling coordinates.
 - status_t **DPU_InitWarpCoordinates** (**IRIS_MVPL_Type** *base, **dpu_unit_t** unit, const **dpu_coordinates_config_t** *config)
Initialize the arbitrary warping coordinates.
 - void **DPU_FetcUnitGetDefaultWarpConfig** (**dpu_warp_config_t** *config)

Path configuration

Get the default warp configuration for FetchWarp unit.

- status_t **DPU_InitFetchUnitWarp** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, const **dpu_warp_config_t** *config)
Initialize the Warp function for FetchWarp unit.

- void **DPU_SrcBufferGetDefaultConfig** (**dpu_src_buffer_config_t** *config)
Get default configuration structure for fetch unit source buffer.

- status_t **DPU_SetFetchUnitSrcBufferConfig** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint8_t sublayer, const **dpu_src_buffer_config_t** *config)
Set the fetch unit sublayer source buffer.

- void **DPU_SetFetchUnitSrcBufferAddr** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint8_t sublayer, uint32_t baseAddr)
Set the fetch unit sublayer source buffer base address.

- void **DPU_SetFetchUnitFrameSize** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint16_t height, uint16_t width)
Set the fetch unit frame size.

- void **DPU_SetFetchUnitOffset** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint8_t sublayer, uint16_t offsetX, uint16_t offsetY)
Set the fetch unit sublayer offset.

- void **DPU_EnableFetchUnitSrcBuffer** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint8_t sublayer, bool enable)
Enable or disable fetch unit sublayer source buffer.

- void **DPU_ClipWindowGetDefaultConfig** (**dpu_clip_window_config_t** *config)
Get default configuration structure for clip window.

- void **DPU_SetFetchUnitClipWindowConfig** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint8_t sublayer, const **dpu_clip_window_config_t** *config)
Set the fetch unit sublayer clip window.

- void **DPU_EnableFetchUnitClipWindow** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint8_t sublayer, bool enable)
Enable or disable the fetch unit sublayer clip window.

- void **DPU_SetFetchUnitClipColor** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, **dpu_clip_color_mode_t** clipColorMode, uint8_t sublayer)
Set the fetch unit clip color mode.

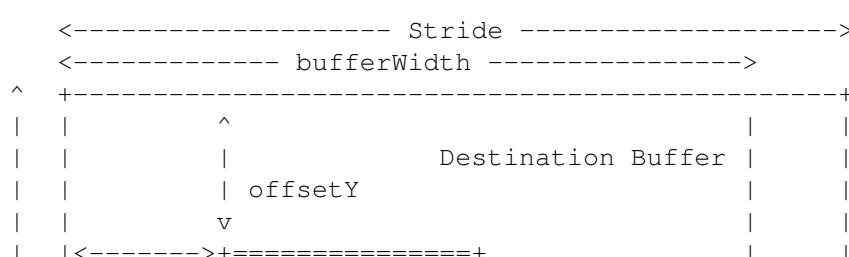
ExtDst Units

- void **DPU_InitExtDst** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint32_t srcReg)

Initialize the ExtDst unit.

Store Units

The Store unit output buffer is like this:



```

bufferHeight |   | offsetX "
|   |      " Input Frame "
|   |
|   |      "
|   +=====+
|   |
|   |
|   |
|   v +-----+

```

- void **DPU_InitStore** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint32_t srcReg)
Initialize the Store unit.
- status_t **DPU_SetStoreDstBufferConfig** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, const **dpu_dst_buffer_config_t** *config)
Set the Store unit Destination buffer configuration.
- void **DPU_DstBufferGetDefaultConfig** (**dpu_dst_buffer_config_t** *config)
Get the default configuration for Store unit.
- void **DPU_SetStoreDstBufferAddr** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint32_t baseAddr)
Set the Store unit Destination buffer base address.
- void **DPU_SetStoreOffset** (IRIS_MVPL_Type *base, **dpu_unit_t** unit, uint16_t offsetX, uint16_t offsetY)
Set the Store unit output offset.
- void **DPU_StartStore** (IRIS_MVPL_Type *base, **dpu_unit_t** unit)
Start the Store unit.

Rop units

Rop unit combines up to three input frames to a single output frame, all having the same dimension.

It supports:

1. Logic Operations Each bit of the RGBA input code is combined with the same bit from the same pixel from the other inputs by any logical operation (= 3 to 1 bit function). The input and output relationship is:

Tertiary Input	Secondary Input	Primary Input	Output
0	0	0	operation index[0]
0	0	1	operation index[1]
0	1	0	operation index[2]
0	1	1	operation index[3]
1	0	0	operation index[4]
1	0	1	operation index[5]
1	1	0	operation index[6]
1	1	1	operation index[7]

2. Arithmetic Operations Input RGBA codes can simply be added for each pixel, optionally with an factor 0.5 being applied for averaging two frames.

Path configuration

- void [DPU_InitRop](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, uint32_t srcReg)
Initialize the ROp unit.
- void [DPU_RopGetDefaultConfig](#) ([dpu_rop_config_t](#) *config)
Get the default ROp unit configuration.
- void [DPU_SetRopConfig](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, const [dpu_rop_config_t](#) *config)
Set the ROp unit configuration.
- void [DPU_EnableRop](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, bool enable)
Enable or disable the ROp unit.

BlitBlend units

- void [DPU_InitBlitBlend](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, uint32_t srcReg)
Initialize the BlitBlend unit.
- void [DPU_BlitBlendGetDefaultConfig](#) ([dpu.blit_blend_config_t](#) *config)
Get the default BlitBlend unit configuration.
- void [DPU_SetBlitBlendConfig](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, const [dpu.blit_blend_config_t](#) *config)
Set the BlitBlend unit configuration.
- void [DPU_EnableBlitBlend](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, bool enable)
Enable or disable the BlitBlend unit.

LayerBlend units

- void [DPU_LayerBlendGetDefaultConfig](#) ([dpu.layer_blend_config_t](#) *config)
Get default configuration structure for LayerBlend.
- void [DPU_InitLayerBlend](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, uint32_t srcReg)
Initialize the LayerBlend.
- void [DPU_SetLayerBlendConfig](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, const [dpu.layer_blend_config_t](#) *config)
Set the LayerBlend unit configuration.
- void [DPU_EnableLayerBlend](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, bool enable)
Enable or disable the LayerBlend unit.

ConstFrame units

- void [DPU_InitConstFrame](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit)
Initialize the ConstFrame unit.
- void [DPU_ConstFrameGetDefaultConfig](#) ([dpu.const_frame_config_t](#) *config)
Get default configuration structure for ConstFrame unit.
- void [DPU_SetConstFrameConfig](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, const [dpu.const_frame_config_t](#) *config)
Set the ConstFrame unit configuration.

VScaler and HScaler units

Note

When both horizontal and vertical scaling is active, then the sequence of both units in the Pixelbus configuration should be

-> HScaler -> VScaler -> when down-scaling horizontally
 -> VScaler -> HScaler -> when up-scaling horizontally

- void [DPU_InitScaler](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit)
Initialize the VScaler or HScaler unit.
- void [DPU_ScalerGetDefaultConfig](#) ([dpu_scaler_config_t](#) *config)
Get default configuration structure for VScaler and HScaler.
- void [DPU_SetScalerConfig](#) (IRIS_MVPL_Type *base, [dpu_unit_t](#) unit, const [dpu_scaler_config_t](#) *config)
Set the VScaler or HScaler units configuration.

Display engine

- void [DPU_DisplayTimingGetDefaultConfig](#) ([dpu_display_timing_config_t](#) *config)
Get default configuration structure for display mode.
- void [DPU_InitDisplayTiming](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, const [dpu_display_timing_config_t](#) *config)
Initialize the display timing.
- void [DPU_DisplayGetDefaultConfig](#) ([dpu_display_config_t](#) *config)
Get default configuration structure for display frame mode.
- void [DPU_SetDisplayConfig](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, const [dpu_display_config_t](#) *config)
Set the display mode.
- void [DPU_StartDisplay](#) (IRIS_MVPL_Type *base, uint8_t displayIndex)
Start the display.
- void [DPU_StopDisplay](#) (IRIS_MVPL_Type *base, uint8_t displayIndex)
Stop the display.
- void [DPU_SetFrameGenInterruptConfig](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, uint8_t interruptIndex, uint32_t intConfig)
Clear the FrameGen unit status flags.
- void [DPU_TriggerDisplayShadowLoad](#) (IRIS_MVPL_Type *base, uint8_t displayIndex)
Trigger the display stream shadow load token.

Signature unit

The Signature unit could compute the CRC value of interested region and compare to the reference value to detect incorrect display output.

Up to 8 evaluation windows can be setup. Signature computation and reference check is done individually for each window.

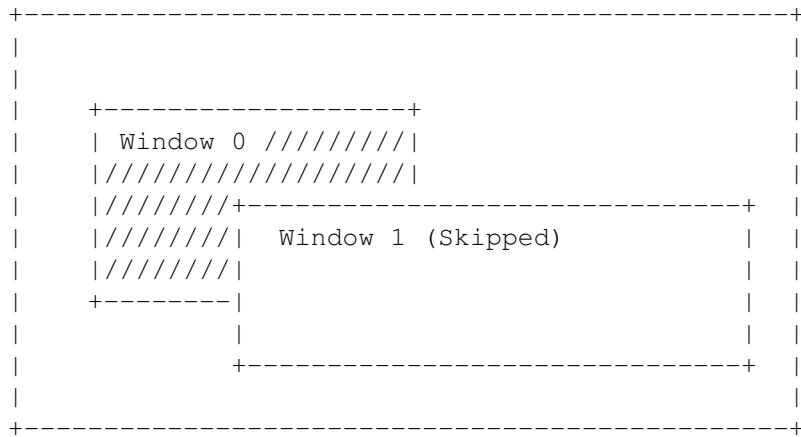
A pixel of the input frame does not contribute to more than one window. In case of overlapping windows, the window with larger index is on top.

Alpha mask could be involved into the signature evaluation, thus any kind of shape could be monitored.

Note that the mask is considered for checksum computation only, not for assignment of individual pixels to a certain evaluation window. So, a non-rectangular overlap between different windows is not possible.

Path configuration

An evaluation window could be configured as skipped. This provides another method for monitoring non-rectangular windows. For example:



In this example, window 1 is skipped, in this case, only the shadow part of window 0 is monitored.

- void [DPU_SignatureGetDefaultConfig](#) (*dpu_signature_config_t* *config)
Get Signature unit default configuration.
- void [DPU_InitSignature](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, const *dpu_signature_config_t* *config)
Initialize the Signature unit.
- void [DPU_SignatureWindowGetDefaultConfig](#) (*dpu_signature_window_config_t* *config)
Get Signature unit validate window default configuration.
- void [DPU_SetSignatureWindowConfig](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, uint8_t windowIndex, const *dpu_signature_window_config_t* *config)
Set the Signature unit evaluation window configuration.
- void [DPU_EnableSignatureWindowCompute](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, uint8_t windowIndex, bool enable)
Enable or disable the Signature unit evaluation window CRC value computation.
- void [DPU_EnableSignatureWindowCheck](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, uint8_t windowIndex, bool enable)
Enable or disable the Signature unit evaluation window CRC value check.
- void [DPU_GetSignatureWindowCrc](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, uint8_t windowIndex, uint32_t *redCRC, uint32_t *greenCRC, uint32_t *blueCRC)
Get the measured signature value of the evaluation window.
- void [DPU_SetSignatureWindowRefCrc](#) (IRIS_MVPL_Type *base, uint8_t displayIndex, uint8_t windowIndex, uint32_t redCRC, uint32_t greenCRC, uint32_t blueCRC)
Set the reference signature value of the evaluation window.
- uint32_t [DPU_GetSignatureStatus](#) (IRIS_MVPL_Type *base, uint8_t displayIndex)
Get the signature unit status.
- void [DPU_TriggerSignatureShadowLoad](#) (IRIS_MVPL_Type *base, uint8_t displayIndex)
Trigger the Signature unit configuration shadow load.

8.4 Data Structure Documentation

8.4.1 struct dpu_fetch_unit_config_t

Data Fields

- `uint32_t srcReg`
This value will be set to register pixengcfg_fetchX_dynamic to set the unit input source, see [DPU_MAKE_SRC_REGI](#).
- `uint16_t frameHeight`
Frame height.
- `uint16_t frameWidth`
Frame width.

8.4.1.0.0.1 Field Documentation

8.4.1.0.0.1.1 `uint32_t dpu_fetch_unit_config_t::srcReg`

8.4.1.0.0.1.2 `uint16_t dpu_fetch_unit_config_t::frameHeight`

8.4.1.0.0.1.3 `uint16_t dpu_fetch_unit_config_t::frameWidth`

8.4.2 struct dpu_coordinates_config_t

The coordinate layer supports:

- 32 bpp: 2 x s12.4 (signed fix-point)
- 24 bpp: 2 x s8.
- 16 bpp: 2 x s4.4
- 8 bpp: 2 x s0.4
- 4 bpp: 2 x s(-2).4 (means total value size = 2 bits and lowest bit = 2^{-4})
- 2 bpp: 2 x s(-3).4
- 1 bpp: 1 x s(-3).4 (x and y alternating)

Data Fields

- `uint8_t bitsPerPixel`
Number of bits per pixel in the source buffer.
- `uint16_t strideBytes`
Source buffer stride in bytes.
- `uint32_t baseAddr`
Source buffer base address.
- `uint16_t frameHeight`
Frame height.
- `uint16_t frameWidth`
Frame width.

Data Structure Documentation

8.4.2.0.0.2 Field Documentation

8.4.2.0.0.2.1 uint8_t dpu_coordinates_config_t::bitsPerPixel

Must be 1, 2, 4, 8, 16, 32.

8.4.2.0.0.2.2 uint16_t dpu_coordinates_config_t::strideBytes

8.4.2.0.0.2.3 uint32_t dpu_coordinates_config_t::baseAddr

8.4.2.0.0.2.4 uint16_t dpu_coordinates_config_t::frameHeight

8.4.2.0.0.2.5 uint16_t dpu_coordinates_config_t::frameWidth

8.4.3 struct dpu_warp_config_t

Data Fields

- uint32_t **srcReg**
This value will be set to register pixengcfg_fetchX_dynamic to set the unit input source, see [DPU_MAKE_SRC_REG1](#).
- uint16_t **frameHeight**
Frame height.
- uint16_t **frameWidth**
Frame width.
- uint8_t **warpBitsPerPixel**
Pixel bits of the coordinate layer.
- bool **enableSymmetricOffset**
Enables symmetric range for negative and positive coordinate values by adding an offset of +0.03125 internally to all coordinate input values.
- **dpu_warp_coordinate_mode_t coordMode**
Coordinate layer mode.
- uint32_t **arbStartX**
X of start point position.
- uint32_t **arbStartY**
Y of start point position.
- uint8_t **arbDeltaYY**
Y of vector between start and first sample point.
- uint8_t **arbDeltaYX**
X of vector between start and first sample point.
- uint8_t **arbDeltaXY**
Y of vector between first and second sample point.
- uint8_t **arbDeltaXX**
X of vector between first and second sample point.

8.4.3.0.0.3 Field Documentation

8.4.3.0.0.3.1 uint32_t dpu_warp_config_t::srcReg

8.4.3.0.0.3.2 uint16_t dpu_warp_config_t::frameHeight

8.4.3.0.0.3.3 uint16_t dpu_warp_config_t::frameWidth

8.4.3.0.0.3.4 uint8_t dpu_warp_config_t::warpBitsPerPixel

8.4.3.0.0.3.5 bool dpu_warp_config_t::enableSymmetricOffset

Recommended for small coordinate formats in DD_PNT mode.

8.4.3.0.0.3.6 dpu_warp_coordinate_mode_t dpu_warp_config_t::coordMode

8.4.3.0.0.3.7 uint32_t dpu_warp_config_t::arbStartX

Signed 16.5 fix-point. Used in D_PNT and DD_PNT.

8.4.3.0.0.3.8 uint32_t dpu_warp_config_t::arbStartY

Signed 16.5 fix-point. Used in D_PNT and DD_PNT.

8.4.3.0.0.3.9 uint8_t dpu_warp_config_t::arbDeltaYY

Signed 3.5 fix-point. Used in DD_PNT.

8.4.3.0.0.3.10 uint8_t dpu_warp_config_t::arbDeltaYX

Signed 3.5 fix-point. Used in DD_PNT.

8.4.3.0.0.3.11 uint8_t dpu_warp_config_t::arbDeltaXY

Signed 3.5 fix-point. Used in DD_PNT.

8.4.3.0.0.3.12 uint8_t dpu_warp_config_t::arbDeltaXX

Signed 3.5 fix-point. Used in DD_PNT.

8.4.4 struct dpu_src_buffer_config_t

Base address and stride alignment restrictions: 32 bpp: Base address and stride must be a multiple of 4 bytes. 16 bpp: Base address and stride must be a multiple of 2 bytes. others: any byte alignment allowed

Generally, the [bitsPerPixel](#) and [pixelFormat](#) specify the pixel format in frame buffer, they should match. But when the color palette is used, the [bitsPerPixel](#) specify the format in framebuffer, the [pixelFormat](#) specify the format in color palette entry.

Data Structure Documentation

Data Fields

- `uint32_t baseAddr`
Source buffer base address, see alignment restrictions.
- `uint16_t strideBytes`
Source buffer stride in bytes, see alignment restrictions.
- `uint8_t bitsPerPixel`
Bits per pixel in frame buffer.
- `dpu_pixel_format_t pixelFormat`
Pixel format.
- `uint16_t bufferHeight`
Buffer height.
- `uint16_t bufferWidth`
Buffer width.
- `uint32_t constColor`
Const color shown in the region out of frame buffer, see [DPU_MAKE_CONST_COLOR](#).

8.4.4.0.0.4 Field Documentation

8.4.4.0.0.4.1 `uint32_t dpu_src_buffer_config_t::baseAddr`

8.4.4.0.0.4.2 `uint16_t dpu_src_buffer_config_t::strideBytes`

8.4.4.0.0.4.3 `uint8_t dpu_src_buffer_config_t::bitsPerPixel`

8.4.4.0.0.4.4 `dpu_pixel_format_t dpu_src_buffer_config_t::pixelFormat`

8.4.4.0.0.4.5 `uint16_t dpu_src_buffer_config_t::bufferHeight`

8.4.4.0.0.4.6 `uint16_t dpu_src_buffer_config_t::bufferWidth`

8.4.4.0.0.4.7 `uint32_t dpu_src_buffer_config_t::constColor`

8.4.5 struct dpu_clip_window_config_t

Data Fields

- `uint16_t windowOffsetX`
Horizontal offset of the clip window.
- `uint16_t windowOffsetY`
Vertical offset of the clip window.
- `uint16_t windowHeight`
Height of the clip window.
- `uint16_t windowWidth`
Width of the clip window.

8.4.5.0.0.5 Field Documentation

8.4.5.0.0.5.1 `uint16_t dpu_clip_window_config_t::windowOffsetX`

8.4.5.0.0.5.2 `uint16_t dpu_clip_window_config_t::windowOffsetY`

8.4.5.0.0.5.3 `uint16_t dpu_clip_window_config_t::windowHeight`

8.4.5.0.0.5.4 `uint16_t dpu_clip_window_config_t::windowWidth`

8.4.6 `struct dpu_dst_buffer_config_t`

Base address and stride alignment restrictions: 32 bpp: Base address and stride must be a multiple of 4 bytes. 16 bpp: Base address and stride must be a multiple of 2 bytes. others: any byte alignment allowed

Data Fields

- `uint32_t baseAddr`
Destination buffer base address, see alignment restrictions.
- `uint16_t strideBytes`
Destination buffer stride in bytes, see alignment restrictions.
- `uint8_t bitsPerPixel`
Bits per pixel.
- `dpu_pixel_format_t pixelFormat`
Pixel format.
- `uint16_t bufferHeight`
Buffer height.
- `uint16_t bufferWidth`
Buffer width.

Data Structure Documentation

8.4.6.0.0.6 Field Documentation

- 8.4.6.0.0.6.1 `uint32_t dpu_dst_buffer_config_t::baseAddr`
- 8.4.6.0.0.6.2 `uint16_t dpu_dst_buffer_config_t::strideBytes`
- 8.4.6.0.0.6.3 `uint8_t dpu_dst_buffer_config_t::bitsPerPixel`
- 8.4.6.0.0.6.4 `dpu_pixel_format_t dpu_dst_buffer_config_t::pixelFormat`
- 8.4.6.0.0.6.5 `uint16_t dpu_dst_buffer_config_t::bufferHeight`
- 8.4.6.0.0.6.6 `uint16_t dpu_dst_buffer_config_t::bufferWidth`

8.4.7 struct `dpu_layer_blend_config_t`

Data Fields

- `uint8_t constAlpha`
The const alpha value used in blend.
- `dpu_blend_mode_t secAlphaBlendMode`
Secondary (overlay) input alpha blending function.
- `dpu_blend_mode_t primAlphaBlendMode`
Primary (background) input alpha blending function.
- `dpu_blend_mode_t secColorBlendMode`
Secondary (overlay) input color blending function.
- `dpu_blend_mode_t primColorBlendMode`
Primary (background) input color blending function.
- `uint32_t srcReg`
This value will be set to pixengcfg_layerblendX_dynamic to set the unit input source, see [DPU_MAKE_SRC_REG2](#).
- `bool enableAlphaMask`
Enable AlphaMask feature.
- `dpu_alpha_mask_mode_t alphaMaskMode`
AlphaMask mode, only valid when enableAlphaMask is true.

8.4.7.0.0.7 Field Documentation

8.4.7.0.0.7.1 `uint8_t dpu_layer_blend_config_t::constAlpha`

8.4.7.0.0.7.2 `dpu_blend_mode_t dpu_layer_blend_config_t::secAlphaBlendMode`

8.4.7.0.0.7.3 `dpu_blend_mode_t dpu_layer_blend_config_t::primAlphaBlendMode`

8.4.7.0.0.7.4 `dpu_blend_mode_t dpu_layer_blend_config_t::secColorBlendMode`

8.4.7.0.0.7.5 `dpu_blend_mode_t dpu_layer_blend_config_t::primColorBlendMode`

8.4.7.0.0.7.6 `uint32_t dpu_layer_blend_config_t::srcReg`

8.4.7.0.0.7.7 `bool dpu_layer_blend_config_t::enableAlphaMask`

8.4.7.0.0.7.8 `dpu_alpha_mask_mode_t dpu_layer_blend_config_t::alphaMaskMode`

8.4.8 struct `dpu.blit_blend_config_t`

Data Fields

- `uint8_t neutralBorderRightPixels`
Number of neutral right border pixels.
- `uint8_t neutralBorderLeftPixels`
Number of neutral left border pixels.
- `dpu.blit_blend_neutral_border_mode_t neutralBorderMode`
Neutral border mode.
- `uint32_t constColor`
Const color used for blit blend, see [DPU_MAKE_CONST_COLOR](#).
- `dpu.blit_blend_func_t redBlendFuncSrc`
Red component source blend function.
- `dpu.blit_blend_func_t redBlendFuncDst`
Red component destination blend function.
- `dpu.blit_blend_func_t greenBlendFuncSrc`
Green component source blend function.
- `dpu.blit_blend_func_t greenBlendFuncDst`
Green component destination blend function.
- `dpu.blit_blend_func_t blueBlendFuncSrc`
Blue component source blend function.
- `dpu.blit_blend_func_t blueBlendFuncDst`
Blue component destination blend function.
- `dpu.blit_blend_func_t alphaBlendFuncSrc`
Alpha component source blend function.
- `dpu.blit_blend_func_t alphaBlendFuncDst`
Alpha component destination blend function.
- `dpu.blit_blend_mode_t redBlendMode`
Red component blend mode.
- `dpu.blit_blend_mode_t greenBlendMode`
Green component blend mode.

Data Structure Documentation

- `dpu_blit_blend_mode_t blueBlendMode`
Blue component blend mode.
- `dpu_blit_blend_mode_t alphaBlendMode`
Alpha component blend mode.

8.4.8.0.0.8 Field Documentation

8.4.8.0.0.8.1 `uint8_t dpu_blit_blend_config_t::neutralBorderRightPixels`

8.4.8.0.0.8.2 `uint8_t dpu_blit_blend_config_t::neutralBorderLeftPixels`

8.4.8.0.0.8.3 `dpu_blit_blend_neutral_border_mode_t dpu_blit_blend_config_t::neutralBorderMode`

8.4.8.0.0.8.4 `uint32_t dpu_blit_blend_config_t::constColor`

8.4.9 struct `dpu_rop_config_t`

Data Fields

- `uint32_t controlFlags`
Control flags, see [_dpu_rop_flags](#).
- `uint8_t alphaIndex`
Alpha operation index.
- `uint8_t blueIndex`
Blue operation index.
- `uint8_t greenIndex`
Green operation index.
- `uint8_t redIndex`
Red operation index.

8.4.9.0.0.9 Field Documentation

8.4.9.0.0.9.1 `uint32_t dpu_rop_config_t::controlFlags`

8.4.9.0.0.9.2 `uint8_t dpu_rop_config_t::alphaIndex`

8.4.9.0.0.9.3 `uint8_t dpu_rop_config_t::blueIndex`

8.4.9.0.0.9.4 `uint8_t dpu_rop_config_t::greenIndex`

8.4.9.0.0.9.5 `uint8_t dpu_rop_config_t::redIndex`

8.4.10 struct `dpu_const_frame_config_t`

Data Fields

- `uint16_t frameHeight`
Frame height.

- `uint16_t frameWidth`
Frame width.
- `uint32_t constColor`
See [DPU_MAKE_CONST_COLOR](#).

8.4.10.0.0.10 Field Documentation

8.4.10.0.0.10.1 `uint16_t dpu_const_frame_config_t::frameHeight`

8.4.10.0.0.10.2 `uint16_t dpu_const_frame_config_t::frameWidth`

8.4.10.0.0.10.3 `uint32_t dpu_const_frame_config_t::constColor`

8.4.11 `struct dpu_display_timing_config_t`

Data Fields

- `uint16_t flags`
OR'ed value of [_dpu_display_timing_flags](#).
- `uint16_t width`
Active width.
- `uint16_t hsw`
HSYNC pulse width.
- `uint16_t hfp`
Horizontal front porch.
- `uint16_t hbp`
Horizontal back porch.
- `uint16_t height`
Active height.
- `uint16_t vsw`
VSYNC pulse width.
- `uint16_t vfp`
Vertical front porch.
- `uint16_t vbp`
Vertical back porch.

Data Structure Documentation

8.4.11.0.0.11 Field Documentation

8.4.11.0.0.11.1 `uint16_t dpu_display_timing_config_t::flags`

8.4.11.0.0.11.2 `uint16_t dpu_display_timing_config_t::width`

8.4.11.0.0.11.3 `uint16_t dpu_display_timing_config_t::hsw`

8.4.11.0.0.11.4 `uint16_t dpu_display_timing_config_t::hfp`

8.4.11.0.0.11.5 `uint16_t dpu_display_timing_config_t::hbp`

8.4.11.0.0.11.6 `uint16_t dpu_display_timing_config_t::height`

8.4.11.0.0.11.7 `uint16_t dpu_display_timing_config_t::vsw`

8.4.11.0.0.11.8 `uint16_t dpu_display_timing_config_t::vfp`

8.4.11.0.0.11.9 `uint16_t dpu_display_timing_config_t::vbp`

8.4.12 struct `dpu_display_config_t`

Data Fields

- `bool enablePrimAlpha`
Enable primary input alpha for screen composition.
- `bool enableSecAlpha`
Enable secondary input alpha for screen composition.
- `dpu_display_mode_t displayMode`
Display mode.
- `bool enablePrimAlphaInPanic`
Enable primary input alpha for screen composition in panic mode.
- `bool enableSecAlphaInPanic`
Enable secondary input alpha for screen composition in panic mode.
- `dpu_display_mode_t displayModeInPanic`
Display mode in panic mode.
- `uint16_t constRed`
Const red value, 10-bit.
- `uint16_t constGreen`
Const green value, 10-bit.
- `uint16_t constBlue`
Const green value, 10-bit.
- `uint8_t constAlpha`
Const alpha value, 1-bit.
- `uint16_t primAreaStartX`
Primary screen upper left corner, x component.
- `uint16_t primAreaStartY`
Primary screen upper left corner, y component.
- `uint16_t secAreaStartX`
Secondary screen upper left corner, x component.

- `uint16_t secAreaStartY`
Secondary screen upper left corner, y component.

8.4.12.0.0.12 Field Documentation

8.4.12.0.0.12.1 `bool dpu_display_config_t::enablePrimAlpha`

8.4.12.0.0.12.2 `bool dpu_display_config_t::enableSecAlpha`

8.4.12.0.0.12.3 `dpu_display_mode_t dpu_display_config_t::displayMode`

8.4.12.0.0.12.4 `bool dpu_display_config_t::enablePrimAlphaInPanic`

8.4.12.0.0.12.5 `bool dpu_display_config_t::enableSecAlphaInPanic`

8.4.12.0.0.12.6 `dpu_display_mode_t dpu_display_config_t::displayModeInPanic`

8.4.12.0.0.12.7 `uint16_t dpu_display_config_t::constRed`

8.4.12.0.0.12.8 `uint16_t dpu_display_config_t::constGreen`

8.4.12.0.0.12.9 `uint16_t dpu_display_config_t::constBlue`

8.4.12.0.0.12.10 `uint8_t dpu_display_config_t::constAlpha`

8.4.12.0.0.12.11 `uint16_t dpu_display_config_t::primAreaStartX`

14-bit , start from 1.

8.4.12.0.0.12.12 `uint16_t dpu_display_config_t::primAreaStartY`

14-bit, start from 1.

8.4.12.0.0.12.13 `uint16_t dpu_display_config_t::secAreaStartX`

14-bit, start from 1.

8.4.12.0.0.12.14 `uint16_t dpu_display_config_t::secAreaStartY`

14-bit, start from 1.

8.4.13 `struct dpu_scaler_config_t`

Data Fields

- `uint32_t srcReg`
This value will be set to register pixengcfg_slacer_dynamic to set the unit input source, see [DPU_MAKE_SRC_REG1](#).
- `uint16_t inputSize`

Data Structure Documentation

- `uint16_t outputSize`
For HScaler, it is frame width, for VScaler, it is frame height.

8.4.13.0.0.13 Field Documentation

8.4.13.0.0.13.1 `uint32_t dpu_scaler_config_t::srcReg`

When down-scaling horizontally, the path should be -> HScaler -> VScaler ->, When up-scaling horizontally, the path should be -> VScaler -> HScaler ->.

8.4.13.0.0.13.2 `uint16_t dpu_scaler_config_t::inputSize`

8.4.13.0.0.13.3 `uint16_t dpu_scaler_config_t::outputSize`

8.4.14 `struct dpu_signature_config_t`

Data Fields

- `uint8_t errorThreshold`
Number of frames with signature violation before signature error is set for an evaluation window.
- `uint8_t errorResetThreshold`
Number of consecutive frames without signature violation before signature error is reset for an evaluation window.
- `uint8_t panicRed`
Constant color shown in the window when local panic happens.
- `uint8_t panicGreen`
Constant color shown in the window when local panic happens.
- `uint8_t panicBlue`
Constant color shown in the window when local panic happens.
- `uint8_t panicAlpha`
Constant color shown in the window when local panic happens.

8.4.14.0.0.14 Field Documentation

8.4.14.0.0.14.1 `uint8_t dpu_signature_config_t::errorThreshold`

8.4.14.0.0.14.2 `uint8_t dpu_signature_config_t::errorResetThreshold`

8.4.14.0.0.14.3 `uint8_t dpu_signature_config_t::panicRed`

8.4.14.0.0.14.4 `uint8_t dpu_signature_config_t::panicGreen`

8.4.14.0.0.14.5 `uint8_t dpu_signature_config_t::panicBlue`

8.4.14.0.0.14.6 `uint8_t dpu_signature_config_t::panicAlpha`

Must be 0 or 1

8.4.15 struct dpu_signature_window_config_t

Data Fields

- `uint32_t controlFlags`
Control flags, OR'ed value of `_dpu_signature_window_flags`.
- `uint16_t upperLeftX`
X coordinate of the upper left corner.
- `uint16_t upperLeftY`
Y coordinate of the upper left corner.
- `uint16_t lowerRightX`
X coordinate of the lower right corner.
- `uint16_t lowerRightY`
Y coordinate of the lower right corner.

8.4.15.0.0.15 Field Documentation

8.4.15.0.0.15.1 `uint32_t dpu_signature_window_config_t::controlFlags`

8.4.15.0.0.15.2 `uint16_t dpu_signature_window_config_t::upperLeftX`

8.4.15.0.0.15.3 `uint16_t dpu_signature_window_config_t::upperLeftY`

8.4.15.0.0.15.4 `uint16_t dpu_signature_window_config_t::lowerRightX`

8.4.15.0.0.15.5 `uint16_t dpu_signature_window_config_t::lowerRightY`

8.5 Macro Definition Documentation

8.5.1 `#define FSL_DPU_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

8.5.2 `#define DPU_PALETTE_ENTRY_NUM (256)`

8.5.3 `#define DPU_MAKE_SRC_REG1(src) (((uint32_t)(src)) & 0x3FU)`

8.5.4 `#define DPU_MAKE_SRC_REG2(primSrc, secSrc) (((((uint32_t)(primSrc)) & 0x3FU) | (((((uint32_t)(secSrc)) & 0x3FU) << 0x8U)))`

8.5.5 `#define DPU_MAKE_SRC_REG3(primSrc, secSrc, tertSrc)`

Value:

```
((((uint32_t)(primSrc)) & 0x3FU) | (((((uint32_t)(secSrc)) & 0x3FU) << 0x8U) | \
(((uint32_t)(tertSrc)) & 0x3FU) << 0x10U)))
```

Enumeration Type Documentation

- 8.5.6 `#define DPU_MAKE_CONST_COLOR(red, green, blue, alpha) (((uint32_t)(red)) << 24U) | (((uint32_t)(green)) << 16U) | (((uint32_t)(blue)) << 8U) | ((uint32_t)(alpha)))`
- 8.5.7 `#define DPU_FRAME_GEN_INT_DISABLE 0U`
- 8.5.8 `#define DPU_FRAME_GEN_INT_PER_LINE(colNum) ((1U << 31U) | (1U << 15U) | (((uint32_t)colNum) & (0x3FFF)))`
- 8.5.9 `#define DPU_FRAME_GEN_INT_PER_FRAME(rowNum) ((1U << 31U) | (((uint32_t)rowNum) & 0x3FFF0000U))`

8.6 Enumeration Type Documentation

8.6.1 enum dpu_unit_t

8.6.2 enum _dpu_interrupt

Enumerator

kDPU_Group0Store9ShadowLoadInterrupt Store9 shadow load interrupt.
kDPU_Group0Store9FrameCompleteInterrupt Store9 frame complete interrupt.
kDPU_Group0Store9SeqCompleteInterrupt Store9 sequence complete interrupt.
kDPU_Group0ExtDst0ShadowLoadInterrupt ExtDst0 shadow load interrupt.
kDPU_Group0ExtDst0FrameCompleteInterrupt ExtDst0 frame complete interrupt.
kDPU_Group0ExtDst0SeqCompleteInterrupt ExtDst0 sequence complete interrupt.
kDPU_Group0ExtDst4ShadowLoadInterrupt ExtDst4 shadow load interrupt.
kDPU_Group0ExtDst4FrameCompleteInterrupt ExtDst4 frame complete interrupt.
kDPU_Group0ExtDst4SeqCompleteInterrupt ExtDst4 sequence complete interrupt.
kDPU_Group0ExtDst1ShadowLoadInterrupt ExtDst1 shadow load interrupt.
kDPU_Group0ExtDst1FrameCompleteInterrupt ExtDst1 frame complete interrupt.
kDPU_Group0ExtDst1SeqCompleteInterrupt ExtDst1 sequence complete interrupt.
kDPU_Group0ExtDst5ShadowLoadInterrupt ExtDst5 shadow load interrupt.
kDPU_Group0ExtDst5FrameCompleteInterrupt ExtDst5 frame complete interrupt.
kDPU_Group0ExtDst5SeqCompleteInterrupt ExtDst5 sequence complete interrupt.
kDPU_Group0Display0ShadowLoadInterrupt Display stream 0 shadow load interrupt.
kDPU_Group0Display0FrameCompleteInterrupt Display stream 0 frame complete interrupt.
kDPU_Group0Display0SeqCompleteInterrupt Display stream 0 sequence complete interrupt.
kDPU_Group0FrameGen0Int0Interrupt FrameGen 0 interrupt 0.
kDPU_Group0FrameGen0Int1Interrupt FrameGen 0 interrupt 1.
kDPU_Group0FrameGen0Int2Interrupt FrameGen 0 interrupt 2.
kDPU_Group0FrameGen0Int3Interrupt FrameGen 0 interrupt 3.
kDPU_Group0Sig0ShadowLoadInterrupt Sig0 shadow load interrupt.
kDPU_Group0Sig0ValidInterrupt Sig0 measurement valid interrupt.

kDPU_Group0Sig0ErrorInterrupt Sig0 error interrupt.
kDPU_Group0Display1ShadowLoadInterrupt Display stream 1 shadow load interrupt.
kDPU_Group0Display1FrameCompleteInterrupt Display stream 1 frame complete interrupt.
kDPU_Group0Display1SeqCompleteInterrupt Display stream 1 sequence complete interrupt.
kDPU_Group0FrameGen1Int0Interrupt FrameGen 1 interrupt 0.
kDPU_Group0FrameGen1Int1Interrupt FrameGen 1 interrupt 1.
kDPU_Group0FrameGen1Int2Interrupt FrameGen 1 interrupt 2.
kDPU_Group0FrameGen1Int3Interrupt FrameGen 1 interrupt 3.
kDPU_Group1Sig1ShadowLoadInterrupt Sig1 shadow load interrupt.
kDPU_Group1Sig1ValidInterrupt Sig1 measurement valid interrupt.
kDPU_Group1Sig1ErrorInterrupt Sig1 error interrupt.
kDPU_Group1CmdSeqErrorInterrupt Command sequencer error interrupt.
kDPU_Group1SoftwareInt0Interrupt Common software interrupt 0.
kDPU_Group1SoftwareInt1Interrupt Common software interrupt 1.
kDPU_Group1SoftwareInt2Interrupt Common software interrupt 2.
kDPU_Group1SoftwareInt3Interrupt Common software interrupt 3.
kDPU_Group1FrameGen0PrimSyncOnInterrupt Safety stream 0 synchronized interrupt.
kDPU_Group1FrameGen0PrimSyncOffInterrupt Safety stream 0 loss synchronization interrupt.
kDPU_Group1FrameGen0SecSyncOnInterrupt Content stream 0 synchronized interrupt.
kDPU_Group1FrameGen0SecSyncOffInterrupt Content stream 0 loss synchronization interrupt.
kDPU_Group1FrameGen1PrimSyncOnInterrupt Safety stream 1 synchronized interrupt.
kDPU_Group1FrameGen1PrimSyncOffInterrupt Safety stream 1 loss synchronization interrupt.
kDPU_Group1FrameGen1SecSyncOnInterrupt Content stream 1 synchronized interrupt.
kDPU_Group1FrameGen1SecSyncOffInterrupt Content stream 1 loss synchronization interrupt.

8.6.3 enum _dpu_unit_source

Enumerator

kDPU_UnitSrcNone Disable the input source.
kDPU_UnitSrcFetchDecode9 The input source is fetch decode 9.
kDPU_UnitSrcFetchWarp9 The input source is fetch warp 9.
kDPU_UnitSrcFetchEco9 The input source is fetch eco 9.
kDPU_UnitSrcRop9 The input source is rop 9.
kDPU_UnitSrcClut9 The input source is CLUT 9.
kDPU_UnitSrcMatrix9 The input source is matrix 9.
kDPU_UnitSrcHScaler9 The input source is HScaler 9.
kDPU_UnitSrcVScaler9 The input source is VScaler 9.
kDPU_UnitSrcFilter9 The input source is Filter 9.
kDPU_UnitSrcBlitBlend9 The input source is BlitBlend 9.
kDPU_UnitSrcStore9 The input source is Store 9.
kDPU_UnitSrcConstFrame0 The input source is ConstFrame 0.
kDPU_UnitSrcConstFrame1 The input source is ConstFrame 1.
kDPU_UnitSrcConstFrame4 The input source is ConstFrame 4.

Enumeration Type Documentation

kDPU_UnitSrcConstFrame5 The input source is ConstFrame 5.
kDPU_UnitSrcFetchWarp2 The input source is FetchWarp 2.
kDPU_UnitSrcFetchEco2 The input source is FetchEco 2.
kDPU_UnitSrcFetchDecode0 The input source is FetchDecode 0.
kDPU_UnitSrcFetchEco0 The input source is FetchEco 0.
kDPU_UnitSrcFetchDecode1 The input source is FetchDecode 1.
kDPU_UnitSrcFetchEco1 The input source is FetchEco 1.
kDPU_UnitSrcFetchLayer0 The input source is FetchLayer 0.
kDPU_UnitSrcMatrix4 The input source is Matrix 4.
kDPU_UnitSrcHScaler4 The input source is HScaler 4.
kDPU_UnitSrcVScaler4 The input source is VScaler 4.
kDPU_UnitSrcMatrix5 The input source is Matrix 5.
kDPU_UnitSrcHScaler5 The input source is HScaler 5.
kDPU_UnitSrcVScaler5 The input source is VScaler 5.
kDPU_UnitSrcLayerBlend0 The input source is LayerBlend 0.
kDPU_UnitSrcLayerBlend1 The input source is LayerBlend 1.
kDPU_UnitSrcLayerBlend2 The input source is LayerBlend 2.
kDPU_UnitSrcLayerBlend3 The input source is LayerBlend 3.

8.6.4 enum dpu_pixel_format_t

To support more pixel format, enhance this enum and the array s_dpuColorComponentFormats.

Enumerator

kDPU_PixelFormatGray8 8-bit gray.
kDPU_PixelFormatRGB565 RGB565, 16-bit per pixel.
kDPU_PixelFormatARGB8888 ARGB8888, 32-bit per pixel.
kDPU_PixelFormatRGB888 RGB888, 24-bit per pixel.
kDPU_PixelFormatARGB1555 ARGB1555, 16-bit per pixel.

8.6.5 enum dpu_warp_coordinate_mode_t

Enumerator

kDPU_WarpCoordinateModePNT Sample points positions are read from coordinate layer.
kDPU_WarpCoordinateModeDPNT Sample points start position and delta are read from coordinate layer.
kDPU_WarpCoordinateModeDDPNT Sample points initial value and delta increase value are read from coordinate layer.

8.6.6 enum dpu_clip_color_mode_t

Enumerator

kDPU_ClipColorNull Use null color.

kDPU_ClipColorSublayer Use color of sublayer.

8.6.7 enum dpu_alpha_mask_mode_t

Enumerator

kDPU_AlphaMaskPrim Areas with primary input alpha > 128 mapped to alpha 255, the rest mapped to 0.

kDPU_AlphaMaskSec Areas with secondary input alpha > 128 mapped to alpha 255, the rest mapped to 0.

kDPU_AlphaMaskPrimOrSec Primary and secondary OR'ed together.

kDPU_AlphaMaskPrimAndSec Primary and secondary AND'ed together.

kDPU_AlphaMaskPrimInv Primary input alpha inverted.

kDPU_AlphaMaskSecInv Secondary input alpha inverted.

kDPU_AlphaMaskPrimOrSecInv Primary and inverted secondary OR'ed together.

kDPU_AlphaMaskPrimAndSecInv Primary and inverted secondary AND'ed together.

8.6.8 enum dpu_blend_mode_t

Enumerator

kDPU_BitBlendFuncGlZero OUT = IN * 0.

kDPU_BitBlendFuncGlOne OUT = IN * 1.

kDPU_BitBlendFuncGlPrimAlpha OUT = IN * ALPHA_primary.

kDPU_BitBlendFuncGlPrimAlphaInv OUT = IN * (1 - ALPHA_primary).

kDPU_BitBlendFuncGlSecAlpha OUT = IN * ALPHA_secondary.

kDPU_BitBlendFuncGlSecAlphaInv OUT = IN * (1 - ALPHA_secondary).

kDPU_BitBlendFuncGlConstAlpha OUT = IN * ALPHA_const.

kDPU_BitBlendFuncGlConstAlphaInv OUT = IN * (1 - ALPHA_const).

8.6.9 enum dpu.blit_blend_func_t

Enumerator

kDPU_BitBlendFuncGlZero GL_ZERO.

kDPU_BitBlendFuncGlOne GL_ONE.

kDPU_BitBlendFuncGlSrcColor GL_SRC_COLOR.

Enumeration Type Documentation

kDPU_BlitBlendFuncGlOneMinusSrcColor GL_ONE_MINUS_SRC_COLOR.
kDPU_BlitBlendFuncGlSrcAlpha GL_SRC_ALPHA.
kDPU_BlitBlendFuncGlOneMinusSrcAlpha GL_ONE_MINUS_SRC_ALPHA.
kDPU_BlitBlendFuncGlDstAlpha GL_DST_ALPHA.
kDPU_BlitBlendFuncGlOneMinusDstAlpha GL_ONE_MINUS_DST_ALPHA.
kDPU_BlitBlendFuncGlDstColor GL_DST_COLOR.
kDPU_BlitBlendFuncGlOneMinusDstColor GL_ONE_MINUS_DST_COLOR.
kDPU_BlitBlendFuncGlSrcAlphaSaturate GL_SRC_ALPHA_SATURATE.
kDPU_BlitBlendFuncGlConstColor GL_CONSTANT_COLOR.
kDPU_BlitBlendFuncGlOneMinusConstColor GL_ONE_MINUS_CONSTANT_COLOR.
kDPU_BlitBlendFuncGlConstAlpha GL_CONSTANT_ALPHA.
kDPU_BlitBlendFuncGlOneMinusConstAlpha GL_ONE_MINUS_CONSTANT_ALPHA.

8.6.10 enum dpu_blend_mode_t

Enumerator

kDPU_BlitBlendModeGlFuncAdd GL_FUNC_ADD.
kDPU_BlitBlendModeGlMin GL_MIN.
kDPU_BlitBlendModeGlMax GL_MAX.
kDPU_BlitBlendModeGlFuncSubtract GL_FUNC_SUBTRACT.
kDPU_BlitBlendModeGlFuncReverseSubtract GL_FUNC_REVERSE_SUBTRACT.
kDPU_BlitBlendModeVgBlendSrc VG_BLEND_SRC.
kDPU_BlitBlendModeVgBlendSrcOver VG_BLEND_SRC_OVER.
kDPU_BlitBlendModeVgBlendDstOver VG_BLEND_DST_OVER.
kDPU_BlitBlendModeVgBlendSrcIn VG_BLEND_SRC_IN.
kDPU_BlitBlendModeVgBlendDstIn VG_BLEND_DST_IN.
kDPU_BlitBlendModeVgBlendMultiply VG_BLEND_MULTIPLY.
kDPU_BlitBlendModeVgBlendScreen VG_BLEND_SCREEN.
kDPU_BlitBlendModeVgBlendDarken VG_BLEND_DARKEN.
kDPU_BlitBlendModeVgBlendLighten VG_BLEND_LIGHTEN.
kDPU_BlitBlendModeVgBlendAdditive VG_BLEND_ADDITIVE.

8.6.11 enum dpu_blend_neutral_border_mode_t

Enumerator

kDPU_BlitBlendNeutralBorderPrim Bypasses primary pixel.
kDPU_BlitBlendNeutralBorderSec Bypasses secondary pixel.

8.6.12 enum _dpu_rop_flags

Enumerator

- kDPU_RopAddRed* Set to add the red component, otherwise raster with operation index.
- kDPU_RopAddGreen* Set to add the green component, otherwise raster with operation index.
- kDPU_RopAddBlue* Set to add the blue component, otherwise raster with operation index.
- kDPU_RopAddAlpha* Set to add the alpha component, otherwise raster with operation index.
- kDPU_RopTertDiv2* In add mode, set this to divide tertiary port input by 2.
- kDPU_RopSecDiv2* In add mode, set this to divide secondary port input by 2.
- kDPU_RopPrimDiv2* In add mode, set this to divide primary port input by 2.

8.6.13 enum _dpu_display_timing_flags

Enumerator

- kDPU_DisplayPixelActiveHigh* Pixel data active high.
- kDPU_DisplayPixelActiveLow* Pixel data active low.
- kDPU_DisplayDataEnableActiveHigh* Set to make data enable high active.
- kDPU_DisplayDataEnableActiveLow* Set to make data enable high low.
- kDPU_DisplayHsyncActiveHigh* Set to make HSYNC high active.
- kDPU_DisplayHsyncActiveLow* Set to make HSYNC low active.
- kDPU_DisplayVsyncActiveHigh* Set to make VSYNC high active.
- kDPU_DisplayVsyncActiveLow* Set to make VSYNC low active.

8.6.14 enum dpu_display_mode_t

Enumerator

- kDPU_DisplayBlackBackground* Black background is shown.
- kDPU_DisplayConstBackground* Const color background is shown.
- kDPU_DisplayOnlyPrim* Only primary input is shown.
- kDPU_DisplayOnlySec* Only secondary input is shown.
- kDPU_DisplayPrimOnTop* Both inputs overlaid with primary on top.
- kDPU_DisplaySecOnTop* Both inputs overlaid with secondary on top.
- kDPU_DisplayTest* White background with test pattern shown.

8.6.15 enum _dpu_signature_window_flags

Enumerator

- kDPU_SignatureWindowEnableGlobalPanic* When enabled the window error will activate display stream the panic mode.

Function Documentation

kDPU_SignatureWindowEnableLocalPanic When enabled the window error will replace pixels in window to the const panic color.

kDPU_SignatureWindowEnableAlphaMask When enabled pixels with alpha bit = 0 are ignored for signature computation.

kDPU_SignatureWindowInvertAlpha When enabled pixels with alpha bit = 1 are ignored for signature computation.

8.6.16 enum _dpu_signature_status

Enumerator

kDPU_SignatureIdle Signature unit is in idle status.

kDPU_SignatureValid Signature unit is in idle status.

8.7 Function Documentation

8.7.1 void DPU_Init (IRIS_MVPL_Type * *base*)

This function ungates the DPU clock.

Parameters

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

8.7.2 void DPU_Deinit (IRIS_MVPL_Type * *base*)

This function gates the DPU clock.

Parameters

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

8.7.3 void DPU_PreparePathConfig (IRIS_MVPL_Type * *base*)

The DPU has a default path configuration. Before changing the configuration, this function could be used to break all the original path. This make sure one pixel engine unit is not used in multiple pipelines.

Parameters

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

8.7.4 void DPU_EnableInterrupts (IRIS_MVPL_Type * *base*, uint8_t *group*, uint32_t *mask*)

For example, to enable Store9 shadow load interrupt and Store9 frame complete interrupt, use like this:

```
DPU_EnableInterrupts(DPU, 0,
    kDPU_Group0Store9ShadowLoadInterrupt |
        kDPU_Group0Store9FrameCompleteInterrupt
);
```

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupts to enable, this is a logical OR of members in _dpu_interrupt .

Note

Only the members in the same group could be OR'ed, at the same time, the parameter *group* should be passed in correctly.

8.7.5 void DPU_DisableInterrupts (IRIS_MVPL_Type * *base*, uint8_t *group*, uint32_t *mask*)

For example, to disable Store9 shadow load interrupt and Store9 frame complete interrupt, use like this:

```
DPU_DisableInterrupts(DPU, 0,
    kDPU_Group0Store9ShadowLoadInterrupt |
        kDPU_Group0Store9FrameCompleteInterrupt
);
```

Parameters

Function Documentation

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupts to disable, this is a logical OR of members in _dpu_interrupt .

Note

Only the members in the same group could be OR'ed, at the same time, the parameter *group* should be passed in correctly.

8.7.6 **uint32_t DPU_GetInterruptsPendingFlags (IRIS_MVPL_Type * *base*, uint8_t *group*)**

The pending status are returned as mask. For example, to check the Store9 shadow load interrupt and Store9 frame complete interrupt pending status, use like this.

```
uint32_t pendingStatus = DPU_GetInterruptsPendingFlags(DPU, 0);
if (pendingStatus & kDPU_Group0Store9ShadowLoadInterrupt)
{
    // Store9 shadow load interrupt occurs, handle it.
}
if (pendingStatus & kDPU_Group0Store9FrameCompleteInterrupt)
{
    // Store9 frame complete interrupt occurs, handle it.
}
```

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.

Returns

The interrupts pending status mask value, see [_dpu_interrupt](#).

8.7.7 **void DPU_ClearInterruptsPendingFlags (IRIS_MVPL_Type * *base*, uint8_t *group*, uint32_t *mask*)**

For example, to disable Store9 shadow load interrupt and Store9 frame complete interrupt pending status, use like this:

```
DPU_ClearInterruptsPendingFlags(DPU, 0, kDPU_Group0Store9ShadowLoadInterrupt
| kDPU_Group0Store9FrameCompleteInterrupt);
```

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupt pending flags to clear, this is a logical OR of members in _dpu_interrupt .

Note

Only the members in the same group could be OR'ed, at the same time, the parameter *group* should be passed in correctly.

8.7.8 void DPU_SetInterruptsPendingFlags (IRIS_MVPL_Type * *base*, uint8_t *group*, uint32_t *mask*)

This function sets the interrupts pending flags, this is a method to trigger interrupts by software.

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupt pending flags to set, this is a logical OR of members in _dpu_interrupt .

Note

Only the members in the same group could be OR'ed, at the same time, the parameter *group* should be passed in correctly.

8.7.9 void DPU_MaskUserInterrupts (IRIS_MVPL_Type * *base*, uint8_t *group*, uint32_t *mask*)

The only difference between DPU user interrupt and normal interrupt is user interrupts could be masked by [DPU_MaskUserInterrupts](#). All other APIs useage are the same.

Parameters

Function Documentation

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupts to mask, this is a logical OR of members in _dpu_interrupt .

8.7.10 void DPU_EnableUserInterrupts (**IRIS_MVPL_Type** * *base*, **uint8_t** *group*, **uint32_t** *mask*)

The only difference between DPU user interrupt and normal interrupt is user interrupts could be masked by [DPU_MaskUserInterrupts](#).

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupts to enable, this is a logical OR of members in _dpu_interrupt .

8.7.11 void DPU_DisableUserInterrupts (**IRIS_MVPL_Type** * *base*, **uint8_t** *group*, **uint32_t** *mask*)

The only difference between DPU user interrupt and normal interrupt is user interrupts could be masked by [DPU_MaskUserInterrupts](#).

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupts to disable, this is a logical OR of members in _dpu_interrupt .

8.7.12 **uint32_t** DPU_GetUserInterruptsPendingFlags (**IRIS_MVPL_Type** * *base*, **uint8_t** *group*)

The only difference between DPU user interrupt and normal interrupt is user interrupts could be masked by [DPU_MaskUserInterrupts](#).

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.

Returns

The interrupts pending status mask value, see [_dpu_interrupt](#).

8.7.13 void DPU_ClearUserInterruptsPendingFlags (IRIS_MVPL_Type * *base*, uint8_t *group*, uint32_t *mask*)

The only difference between DPU user interrupt and normal interrupt is user interrupts could be masked by [DPU_MaskUserInterrupts](#).

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupt pending flags to clear, this is a logical OR of members in _dpu_interrupt .

8.7.14 void DPU_SetUserInterruptsPendingFlags (IRIS_MVPL_Type * *base*, uint8_t *group*, uint32_t *mask*)

The only difference between DPU user interrupt and normal interrupt is user interrupts could be masked by [DPU_MaskUserInterrupts](#).

Parameters

<i>base</i>	DPU peripheral base address.
<i>group</i>	Interrupt group index.
<i>mask</i>	The interrupt pending flags to set, this is a logical OR of members in _dpu_interrupt .

8.7.15 status_t DPU_EnableShadowLoad (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, bool *enable*)

For example:

```
// To enable the shadowing of all RWS registers of the pipeline with endpoint Store9.
DPU_EnableShadowLoad(DPU, kDPU_PipelineStore9, true);
```

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	The unit whose shadow load to enable or disable, see dpu_unit_t .
<i>enable</i>	True to enable, false to disable.

Return values

<i>kStatus_Success</i>	The shadow load is enabled or disabled successfully.
<i>kStatus_InvalidArgument</i>	The unit does not support shadow load.

8.7.16 void DPU_InitPipeline (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	The DPU pipeline unit.

8.7.17 void DPU_DeinitPipeline (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*)

Power down the pipeline and disable the shadow load feature.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	The DPU pipeline unit.

8.7.18 void DPU_TriggerPipelineShadowLoad (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*)

This function triggers the pipeline reconfiguration.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	The DPU pipeline unit.

8.7.19 void DPU_TriggerPipelineCompleteInterrupt (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*)

This function triggers the pipeline sequence complete interrupt. After triggered, this interrupt occurs when the pipeline is empty and no more operations are pending. It will occur immediately, when this is the case already during activation of the trigger. Generally this is used for the blit operation, to make sure all operations finished.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	The DPU pipeline unit.

8.7.20 void DPU_SetUnitSrc (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint32_t *srcReg*)

Sets the DPU unit input source, the input source is controlled by the register <unit>_dynamic in "Pixel Engin Top Level". This function writes the register <unit>_dynamic directly, please check the reference manual for the register details. This function only changes the input source control bits in register.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	The DPU pipeline unit.
<i>srcReg</i>	The value written to register <unit>_dynamic. Could be generated using DPU_MAKE_SRC_REG1 , DPU_MAKE_SRC_REG2 , and DPU_MAKE_SRC_REG3 .

8.7.21 void DPU_FetchUnitGetDefaultConfig (dpu_fetch_unit_config_t * *config*)

The default value is:

```
config->srcReg = 0U;
config->frameHeight = 320U;
config->frameWidth = 480U;
```

Function Documentation

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.22 void DPU_InitFetchUnit (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, const dpu_fetch_unit_config_t * *config*)

This function initializes the fetch unit for the basic use, for other use case such as arbitrary warping, use the functions [DPU_InitFetchUnitWarp](#) and [DPU_InitWarpCoordinates](#).

The input source of fetch unit could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcFetchWarp9](#)
- [kDPU_UnitSrcFetchEco2](#)
- [kDPU_UnitSrcFetchEco9](#)
- [kDPU_UnitSrcFetchEco0](#)
- [kDPU_UnitSrcFetchEco1](#)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>config</i>	Pointer to the configuration structure.

8.7.23 status_t DPU_SetColorPaletteIndexWidth (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint8_t *indexWidth*)

The palette index width could be 1 to 8. Note the difference between palette index width and the pixel width in framebuffer.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be FetchDecode or FetchLayer here.
<i>indexWidth</i>	The palette index width.

Return values

<i>kStatus_Success</i>	Initialization success.
<i>kStatus_InvalidArgument</i>	Wrong argument.

8.7.24 **status_t DPU_UpdateColorPalette (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint32_t *startIndex*, const uint32_t * *palette*, uint32_t *count*)**

This function updates the fetch unit color palette, the palette values specified by *palette* are loaded to fetch unit from *startIndex*. The load count is specified by *count*.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be FetchDecode or FetchLayer here.
<i>startIndex</i>	The start index of the fetch unit palette to update.
<i>palette</i>	Pointer to the palette.
<i>count</i>	Count of <i>palette</i> .

Return values

<i>kStatus_Success</i>	Initialization success.
<i>kStatus_InvalidArgument</i>	Wrong argument.

8.7.25 **void DPU_EnableColorPalette (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint8_t *sublayer*, bool *enable*)**

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be FetchDecode or FetchLayer here.
<i>sublayer</i>	Sublayer index, should be 0 to 7.
<i>enable</i>	True to enable, false to disable.

Function Documentation

8.7.26 void DPU_CorrdinatesGetDefaultConfig (*dpu_coordinates_config_t* * *config*)

The default value is:

```
config->bitsPerPixel = 0U;
config->strideBytes = 0x500U;
config->baseAddr = 0U;
config->frameHeight = 320U;
config->frameWidth = 480U;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.27 status_t DPU_InitWarpCoordinates (*IRIS_MVPL_Type* * *base*, *dpu_unit_t* *unit*, *const dpu_coordinates_config_t* * *config*)

This function initializes the FetchEco unit, so that it could be used as the arbitrary warping coordinates.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be FetchEco here.
<i>config</i>	Pointer to the configuration structure.

Return values

<i>kStatus_Success</i>	Initialization success.
<i>kStatus_InvalidArgument</i>	Wrong argument.

8.7.28 void DPU_FetcUnitGetDefaultWarpConfig (*dpu_warp_config_t* * *config*)

The default value is:

```
config->srcReg = 0U;
config->frameHeight = 320U;
config->frameWidth = 480U;
config->warpBitsPerPixel = 0U;
config->enableSymmetricOffset = false;
config->coordMode = KDPU\_WarpCoordinateModePNT;
config->arbStartX = 0U;
config->arbStartY = 0U;
config->arbDeltaYY = 0U;
config->arbDeltaYX = 0U;
config->arbDeltaXY = 0U;
config->arbDeltaXX = 0U;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.29 **status_t DPU_InitFetchUnitWarp (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, const dpu_warp_config_t * *config*)**

This function initializes the FetchWarp unit for the arbitrary warping.

The valid source of fetch warp unit could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcFetchEco2](#)
- [kDPU_UnitSrcFetchEco9](#)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be FetchWarp unit here.
<i>config</i>	Pointer to the configuration structure.

Return values

<i>kStatus_Success</i>	Initialization success.
<i>kStatus_InvalidArgument</i>	Wrong argument.

8.7.30 **void DPU_SrcBufferGetDefaultConfig (dpu_src_buffer_config_t * *config*)**

The default value is:

```
config->baseAddr = 0U;
config->strideBytes = 0x500U;
config->bitsPerPixel = 32U;
config->pixelFormat = kDPU\_PixelFormatARGB8888;
config->bufferHeight = 0U;
config->bufferWidth = 0U;
config->constColor = DPU\_MAKE\_CONST\_COLOR\(0, 0, 0, 0\);
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

Function Documentation

8.7.31 **status_t DPU_SetFetchUnitSrcBufferConfig (IRIS_MVPL_Type * *base*,
dpu_unit_t *unit*, uint8_t *sublayer*, const dpu_src_buffer_config_t * *config*)**

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>sublayer</i>	Sublayer index, should be 0 to 7.
<i>config</i>	Pointer to the configuration structure.

Return values

<i>kStatus_Success</i>	Initialization success.
<i>kStatus_InvalidArgument</i>	Wrong argument.

8.7.32 void DPU_SetFetchUnitSrcBufferAddr (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint8_t *sublayer*, uint32_t *baseAddr*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>sublayer</i>	Sublayer index, should be 0 to 7.
<i>baseAddr</i>	Source buffer base address.

8.7.33 void DPU_SetFetchUnitFrameSize (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint16_t *height*, uint16_t *width*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>height</i>	Frame height.
<i>width</i>	Frame width.

8.7.34 void DPU_SetFetchUnitOffset (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint8_t *sublayer*, uint16_t *offsetX*, uint16_t *offsetY*)

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>sublayer</i>	Sublayer index, should be 0 to 7.
<i>offsetX</i>	Horizontal offset.
<i>offsetY</i>	Vertical offset.

8.7.35 void DPU_EnableFetchUnitSrcBuffer (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint8_t *sublayer*, bool *enable*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>sublayer</i>	Sublayer index, should be 0 to 7.
<i>enable</i>	True to enable, false to disable.

8.7.36 void DPU_ClipWindowGetDefaultConfig (dpu_clip_window_config_t * *config*)

The default value is:

```
config->windowOffsetX = 0U;
config->windowOffsetY = 0U;
config->windowHeight = 0U;
config->>windowWidth = 0U;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.37 void DPU_SetFetchUnitClipWindowConfig (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint8_t *sublayer*, const dpu_clip_window_config_t * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>sublayer</i>	Sublayer index, should be 0 to 7.
<i>config</i>	Pointer to the configuration structure.

8.7.38 void DPU_EnableFetchUnitClipWindow (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*, **uint8_t** *sublayer*, **bool** *enable*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>sublayer</i>	Sublayer index, should be 0 to 7.
<i>enable</i>	True to enable, false to disable.

8.7.39 void DPU_SetFetchUnitClipColor (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*, **dpu_clip_color_mode_t** *clipColorMode*, **uint8_t** *sublayer*)

This function selects which color to take for pixels that do not lie inside the clip window of any layer.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be fetch unit here.
<i>clipColorMode</i>	Select null color or use sublayer color.
<i>sublayer</i>	Select which sublayer's color to use when <i>clipColorMode</i> is kDPU_ClipColorSublayer .

8.7.40 void DPU_InitExtDst (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*, **uint32_t** *srcReg*)

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be ExtDst unit here.
<i>srcReg</i>	Input source selecte register value, pixencfg_extdstX_dynamic see DPU_MAKE_SRC_REG1 . The valid source: <ul style="list-style-type: none">• kDPU_UnitSrcNone• kDPU_UnitSrcBlitBlend9• kDPU_UnitSrcConstFrame0• kDPU_UnitSrcConstFrame1• kDPU_UnitSrcConstFrame4• kDPU_UnitSrcConstFrame5• kDPU_UnitSrcHScaler4• kDPU_UnitSrcVScaler4• kDPU_UnitSrcHScaler5• kDPU_UnitSrcVScaler5• kDPU_UnitSrcLayerBlend0• kDPU_UnitSrcLayerBlend1• kDPU_UnitSrcLayerBlend2• kDPU_UnitSrcLayerBlend3

8.7.41 void DPU_InitStore (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*, **uint32_t** *srcReg*)

The valid input source of the store unit could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcHScaler9](#)
- [kDPU_UnitSrcVScaler9](#)
- [kDPU_UnitSrcVScaler9](#)
- [kDPU_UnitSrcFilter9](#)
- [kDPU_UnitSrcBlitBlend9](#)
- [kDPU_UnitSrcFetchDecode9](#)
- [kDPU_UnitSrcFetchWarp9](#)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be Store unit here.
<i>srcReg</i>	Input source selecte register value, pixencfg_extdstX_dynamic see DPU_MAKE_SRC_REG1 .

8.7.42 **status_t DPU_SetStoreDstBufferConfig (IRIS_MVPL_Type * *base*,
dpu_unit_t *unit*, const dpu_dst_buffer_config_t * *config*)**

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be Store unit here.
<i>config</i>	Pointer to the configuration.

Return values

<i>kStatus_Success</i>	Initialization success.
<i>kStatus_InvalidArgument</i>	Wrong argument.

8.7.43 void DPU_DstBufferGetDefaultConfig ([dpu_dst_buffer_config_t](#) * *config*)

The default value is:

```
config->baseAddr = 0U;
config->strideBytes = 0x500U;
config->bitsPerPixel = 32U;
config->pixelFormat = kDPU\_PixelFormatARGB8888;
config->bufferHeight = 0U;
config->bufferWidth = 0U;
```

Parameters

<i>config</i>	Pointer to the configuration.
---------------	-------------------------------

8.7.44 void DPU_SetStoreDstBufferAddr ([IRIS_MVPL_Type](#) * *base*, [dpu_unit_t](#) *unit*, [uint32_t](#) *baseAddr*)

This function is run time used for better performance.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be Store unit here.
<i>baseAddr</i>	Base address of the Destination buffer to set.

8.7.45 void DPU_SetStoreOffset ([IRIS_MVPL_Type](#) * *base*, [dpu_unit_t](#) *unit*, [uint16_t](#) *offsetX*, [uint16_t](#) *offsetY*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be Store unit here.
<i>offsetX</i>	Horizontal offset.
<i>offsetY</i>	Vertical offset.

Note

The horizontal offset has limitations for some formats. It must be a multiple of - 8 for 1 bpp buffers

- 4 for 2 bpp and 18 bpp buffers
- 2 for 4 bpp buffers

8.7.46 void DPU_StartStore ([IRIS_MVPL_Type](#) * *base*, [dpu_unit_t](#) *unit*)

This function starts the Store unit to save the frame to output buffer. When the frame store completed, the interrupt flag [kDPU_Group0Store9FrameCompleteInterrupt](#) asserts.

This is an example shows how to use Store unit:

```
* // Initialize the Store unit, use FetchDecode9 output as its input.
DPU_InitStore(DPU, kDPU_Store9, DPU_MAKE_SRC_REG1(
    kDPU_UnitSrcFetchDecode9));

// Configure the Store unit output buffer.
DPU_SetStoreDstBufferConfig(DPU, kDPU_Store9, &DstBufferConfig);

// Configure FetchDecode9 unit, including source buffer setting and so on.
//
// CODE FOR FETCHDECODE9
//

// Initialize the Store9 pipeline
DPU_InitPipeline(DPU, kDPU_PipelineStore9);

DPU_ClearUserInterruptsPendingFlags(DPU,
    kDPU_Group0Store9ShadowLoadInterrupt);

// Trigger the shadow load
DPU_TriggerPipelineShadowLoad(DPU, kDPU_PipelineStore9);

DPU_ClearUserInterruptsPendingFlags(DPU,
    kDPU_Group0Store9FrameCompleteInterrupt);

// Start the Store9 to convert and output.
DPU_StartStore(DPU, kDPU_Store9);

// Wait for Store 9 completed, this could also be monitored by interrupt.
while (! (kDPU_Group0Store9FrameCompleteInterrupt &
    DPU_GetUserInterruptsPendingFlags(DPU, 0))
{
}
```

For better performance, it is allowed to set next operation while current is still in progress. Upper layer could set next operation immediately after shadow load finished.

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be Store unit here.

8.7.47 void DPU_InitRop (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*, **uint32_t** *srcReg*)

The primary input source of the unit could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcFetchDecode9](#)
- [kDPU_UnitSrcFetchWarp9](#)

The secondary input source of the unit could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcFetchEco9](#)

The tert input source of the unit could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcFetchDecode9](#)
- [kDPU_UnitSrcFetchWarp9](#)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be Rop unit here.
<i>srcReg</i>	Unit source selection, see DPU_MAKE_SRC_REG3 .

8.7.48 void DPU_RopGetDefaultConfig (**dpu_rop_config_t** * *config*)

The default configuration is:

```
config->controlFlags = 0U;
config->alphaIndex = 0U;
config->blueIndex = 0U;
config->greenIndex = 0U;
config->redIndex = 0U;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.49 void DPU_SetRopConfig (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, const dpu_rop_config_t * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be Rop unit here.
<i>config</i>	Pointer to the configuration structure.

8.7.50 void DPU_EnableRop (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, bool *enable*)

If disabled, only the primary input is output.

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be Rop unit here.
<i>enable</i>	Pass true to enable, false to disable.

8.7.51 void DPU_InitBlitBlend (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint32_t *srcReg*)

The valid input primary source could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcHScaler9](#)
- [kDPU_UnitSrcVScaler9](#)
- [kDPU_UnitSrcFilter9](#)
- [kDPU_UnitSrcRop9](#)

The valid input secondary source could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcFetchDecode9](#)
- [kDPU_UnitSrcFetchWarp9](#)

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be BlitBlend unit here.
<i>srcReg</i>	Unit source selection, see DPU_MAKE_SRC_REG2 .

8.7.52 void DPU_BlitBlendGetDefaultConfig ([dpu_blend_config_t](#) * *config*)

The default configuration is:

```
config->neutralBorderRightPixels = 0U;
config->neutralBorderLeftPixels = 0U;
config->neutralBorderMode = kDPU\_BlitBlendNeutralBorderPrim;
config->constColor = DPU\_MAKE\_CONST\_COLOR\(0, 0, 0, 0\);
config->redBlendFuncSrc = kDPU\_BlitBlendFuncG1SrcColor;
config->redBlendFuncDst = kDPU\_BlitBlendFuncG1SrcColor;
config->greenBlendFuncSrc = kDPU\_BlitBlendFuncG1SrcColor;
config->greenBlendFuncDst = kDPU\_BlitBlendFuncG1SrcColor;
config->blueBlendFuncSrc = kDPU\_BlitBlendFuncG1SrcColor;
config->blueBlendFuncDst = kDPU\_BlitBlendFuncG1SrcColor;
config->alphaBlendFuncSrc = kDPU\_BlitBlendFuncG1SrcColor;
config->alphaBlendFuncDst = kDPU\_BlitBlendFuncG1SrcColor;
config->redBlendMode = kDPU\_BlitBlendModeG1FuncAdd;
config->greenBlendMode = kDPU\_BlitBlendModeG1FuncAdd;
config->blueBlendMode = kDPU\_BlitBlendModeG1FuncAdd;
config->alphaBlendMode = kDPU\_BlitBlendModeG1FuncAdd;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.53 void DPU_SetBlitBlendConfig ([IRIS_MVPL_Type](#) * *base*, [dpu_unit_t](#) *unit*, [const dpu_blend_config_t](#) * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be BlitBlend unit here.
<i>config</i>	Pointer to the configuration structure.

8.7.54 void DPU_EnableBlitBlend (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, bool *enable*)

The BlitBlend unit could be runtime enabled or disabled, when disabled, the primary input is output directly.

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be BlitBlend unit here.
<i>enable</i>	Pass true to enable, false to disable.

8.7.55 void DPU_LayerBlendGetDefaultConfig ([dpu_layer_blend_config_t](#) * *config*)

The default value is:

```
config->constAlpha = 0U;
config->secAlphaBlendMode = kDPU\_BlendOne;
config->primAlphaBlendMode = kDPU\_BlendZero;
config->secColorBlendMode = kDPU\_BlendOne;
config->primColorBlendMode = kDPU\_BlendZero;
config->enableAlphaMask = true;
config->alphaMaskMode = kDPU\_AlphaMaskPrim;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.56 void DPU_InitLayerBlend ([IRIS_MVPL_Type](#) * *base*, [dpu_unit_t](#) *unit*, [uint32_t](#) *srcReg*)

The valid primary source:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcConstFrame0](#)
- [kDPU_UnitSrcConstFrame1](#)
- [kDPU_UnitSrcConstFrame4](#)
- [kDPU_UnitSrcConstFrame5](#)
- [kDPU_UnitSrcHScaler4](#)
- [kDPU_UnitSrcVScaler4](#)
- [kDPU_UnitSrcHScaler5](#)
- [kDPU_UnitSrcVScaler5](#)
- [kDPU_UnitSrcMatrix4](#)
- [kDPU_UnitSrcMatrix5](#)
- [kDPU_UnitSrcLayerBlend0](#)
- [kDPU_UnitSrcLayerBlend1](#)
- [kDPU_UnitSrcLayerBlend2](#)
- [kDPU_UnitSrcLayerBlend3](#)

The valid secondary source:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcConstFrame0](#)
- [kDPU_UnitSrcConstFrame1](#)
- [kDPU_UnitSrcConstFrame4](#)
- [kDPU_UnitSrcConstFrame5](#)
- [kDPU_UnitSrcHScaler4](#)
- [kDPU_UnitSrcVScaler4](#)
- [kDPU_UnitSrcHScaler5](#)
- [kDPU_UnitSrcVScaler5](#)
- [kDPU_UnitSrcMatrix4](#)
- [kDPU_UnitSrcMatrix5](#)
- [kDPU_UnitSrcLayerBlend0](#)
- [kDPU_UnitSrcLayerBlend1](#)
- [kDPU_UnitSrcLayerBlend2](#)
- [kDPU_UnitSrcLayerBlend3](#)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be LayerBlend unit here.
<i>srcReg</i>	Unit source selection, see DPU_MAKE_SRC_REG2 .

8.7.57 void DPU_SetLayerBlendConfig (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*, **const dpu_layer_blend_config_t** * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be LayerBlend unit here.
<i>config</i>	Pointer to the configuration structure.

8.7.58 void DPU_EnableLayerBlend (**IRIS_MVPL_Type** * *base*, **dpu_unit_t** *unit*, **bool enable**)

If enabled, the blend result is output, otherwise, the primary input is output.

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be LayerBlend unit here.
<i>enable</i>	Pass true to enable, false to disable.

8.7.59 void DPU_InitConstFrame (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be ConstFrame unit here.

8.7.60 void DPU_ConstFrameGetDefaultConfig (dpu_const_frame_config_t * *config*)

The default value is:

```
config->frameHeight = 320U;
config->frameWidth = 480U;
config->constColor = DPU\_MAKE\_CONST\_COLOR\(0xFF, 0xFF, 0xFF, 0xFF\);
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.61 void DPU_SetConstFrameConfig (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, const dpu_const_frame_config_t * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be ConstFrame unit here.
<i>config</i>	Pointer to the configuration structure.

8.7.62 void DPU_InitScaler (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be HScaler or VScaler unit here.

8.7.63 void DPU_ScalerGetDefaultConfig ([dpu_scaler_config_t](#) * *config*)

The default value is:

```
config->srcReg = 0U;
config->inputSize = 0U;
config->outputSize = 0U;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.64 void DPU_SetScalerConfig ([IRIS_MVPL_Type](#) * *base*, [dpu_unit_t](#) *unit*, [const dpu_scaler_config_t](#) * *config*)

The valid input source could be:

- [kDPU_UnitSrcNone](#)
- [kDPU_UnitSrcFetchDecode0](#)
- [kDPU_UnitSrcMatrix4](#)
- [kDPU_UnitSrcVScaler4](#)
- [kDPU_UnitSrcHScaler4](#)
- [kDPU_UnitSrcFetchDecode1](#)
- [kDPU_UnitSrcMatrix5](#)
- [kDPU_UnitSrcVScaler5](#)
- [kDPU_UnitSrcHScaler5](#)
- [kDPU_UnitSrcVScaler9](#)
- [kDPU_UnitSrcHScaler9](#)
- [kDPU_UnitSrcFilter9](#)
- [kDPU_UnitSrcMatrix9](#)

Parameters

<i>base</i>	DPU peripheral base address.
<i>unit</i>	DPU unit, see dpu_unit_t , must be HScaler or VScaler unit here.
<i>config</i>	Pointer to the configuration structure.

Function Documentation

8.7.65 void DPU_DisplayTimingGetDefaultConfig (*dpu_display_timing_config_t* * *config*)

The default value is:

```
config->flags = kDPU_DisplayDeActiveHigh;
config->width = 320U;
config->hsw = 32U;
config->hfp = 8U;
config->hbp = 40U;
config->height = 240U;
config->vsw = 4U;
config->vfp = 13U;
config->vbp = 6U;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.66 void DPU_InitDisplayTiming (*IRIS_MVPL_Type* * *base*, *uint8_t* *displayIndex*, *const dpu_display_timing_config_t* * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Index of the display.
<i>config</i>	Pointer to the configuration structure.

8.7.67 void DPU_DisplayGetDefaultConfig (*dpu_display_config_t* * *config*)

The default value is:

```
config->enablePrimAlpha = false;
config->enableSecAlpha = false;
config->displayMode = kDPU_DisplayTest;
config->enablePrimAlphaInPanic = false;
config->enableSecAlphaInPanic = false;
config->displayModeInPanic = kDPU_DisplayTest;
config->constRed = 0x3FFU;
config->constGreen = 0x3FFU;
config->constBlue = 0x3FFU;
config->constAlpha = 1U;
config->primAreaStartX = 1U;
config->primAreaStartY = 1U;
config->secAreaStartX = 1U;
config->secAreaStartY = 1U;
```

Parameters

<i>config</i>	Pointer to the configuration structure.
---------------	---

8.7.68 void DPU_SetDisplayConfig (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*, const dpu_display_config_t * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Index of the display.
<i>config</i>	Pointer to the configuration structure.

8.7.69 void DPU_StartDisplay (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Index of the display.

8.7.70 void DPU_StopDisplay (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*)

This function stops the display and wait the sequence complete.

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Index of the display.

8.7.71 void DPU_SetFrameGenInterruptConfig (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*, uint8_t *interruptIndex*, uint32_t *intConfig*)

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.
<i>interruptIndex</i>	Interrupt index, there could be 4 interrupts for each display.
<i>intConfig</i>	Interrupt mode, could be one of DPU_FRAME_GEN_INT_DISABLE, DPU_FRAME_GEN_INT_PER_LINE, and DPU_FRAME_GEN_INT_PER_FRAME.

8.7.72 void DPU_TriggerDisplayShadowLoad (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*)

Trigger the display stream shadow load token, then the shadow register will be loaded at the begining of next frame.

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.

8.7.73 void DPU_SignatureGetDefaultConfig (dpu_signature_config_t * *config*)

The default configuration is:

```
config->errorThreshold = 0U;
config->errorResetThreshold = 8U;
config->panicRed = 0U;
config->panicGreen = 0U;
config->panicBlue = 0U;
config->panicAlpha = 0U;
```

Parameters

<i>config</i>	Pointer to the configuration.
---------------	-------------------------------

8.7.74 void DPU_InitSignature (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*, const dpu_signature_config_t * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.
<i>config</i>	Pointer to the configuration.

8.7.75 void DPU_SignatureWindowGetDefaultConfig (**dpu_signature_window_config_t** * *config*)

The default configuration is:

```
config->controlFlags = 0U;
config->upperLeftX = 0U;
config->upperLeftY = 0U;
config->lowerRightX = 0U;
config->lowerRightY = 0U;
```

Parameters

<i>config</i>	Pointer to the configuration.
---------------	-------------------------------

8.7.76 void DPU_SetSignatureWindowConfig (**IRIS_MVPL_Type** * *base*, **uint8_t** *displayIndex*, **uint8_t** *windowIndex*, **const dpu_signature_window_config_t** * *config*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.
<i>windowIndex</i>	Evaluation window index, should be 0 to 7.
<i>config</i>	Pointer to the configuration.

8.7.77 void DPU_EnableSignatureWindowCompute (**IRIS_MVPL_Type** * *base*, **uint8_t** *displayIndex*, **uint8_t** *windowIndex*, **bool enable**)

When enabled, a CRC signature is computed for all pixels inside this evaluation window, When disabled, the internal status for this window is reset (StsSigError bit and frame counters)

Function Documentation

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.
<i>windowIndex</i>	Evaluation window index, should be 0 to 7.
<i>enable</i>	Pass true to enable, false to disable.

8.7.78 void DPU_EnableSignatureWindowCheck (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*, uint8_t *windowIndex*, bool *enable*)

When enabled, the measured signature is checked against a reference value.

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.
<i>windowIndex</i>	Evaluation window index, should be 0 to 7.
<i>enable</i>	Pass true to enable, false to disable.

8.7.79 void DPU_GetSignatureWindowCrc (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*, uint8_t *windowIndex*, uint32_t * *redCRC*, uint32_t * *greenCRC*, uint32_t * *blueCRC*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.
<i>windowIndex</i>	Evaluation window index, should be 0 to 7.
<i>redCRC</i>	Measured signature value of red.
<i>greenCRC</i>	Measured signature value of green.
<i>blueCRC</i>	Measured signature value of blue.

8.7.80 void DPU_SetSignatureWindowRefCrc (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*, uint8_t *windowIndex*, uint32_t *redCRC*, uint32_t *greenCRC*, uint32_t *blueCRC*)

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.
<i>windowIndex</i>	Evaluation window index, should be 0 to 7.
<i>redCRC</i>	Reference signature value of red.
<i>greenCRC</i>	Reference signature value of green.
<i>blueCRC</i>	Reference signature value of blue.

8.7.81 uint32_t DPU_GetSignatureStatus (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*)

This function returns the signature unit status. The return value could be compared to check the status defined in [_dpu_signature_status](#). For example:

```
uint32_t status = DPU_GetSignatureStatus(DPU, 0);

if (kDPU_SignatureValid & status)
{
    // signature measure finished, could read the value.
    DPU_GetSignatureWindowCrc(...);
}
```

The error flags are also returned as an mask value, upper layer could get specific window status by checking the returned bit accordingly. For example,

```
uint32_t status = DPU_GetSignatureStatus(DPU, 0);

if ((1<<3) & status)
{
    // Window 3 error detected.
}

if ((1<<5) & status)
{
    // Window 5 error detected.
}
```

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.

Returns

Mask value of status.

Function Documentation

8.7.82 void DPU_TriggerSignatureShadowLoad (IRIS_MVPL_Type * *base*, uint8_t *displayIndex*)

When new configuration set by [DPU_SetSignatureWindowConfig](#), [DPU_EnableSignatureWindowCheck](#), [DPU_EnableSignatureWindowCompute](#), and [DPU_SetSignatureWindowRefCrc](#), use this function to trigger the shadow load, then the new configuration takes effect.

Upper layer should monitor the [kDPU_Group0Sig0ShadowLoadInterrupt](#) or [kDPU_Group1Sig1ShadowLoadInterrupt](#) to wait shadow load finished. New configurations should only be set after shadow load finished.

Parameters

<i>base</i>	DPU peripheral base address.
<i>displayIndex</i>	Display index.

Chapter 9

DPU IRQSTEER: Interrupt Request Steering Driver

9.1 Overview

The MCUXpresso SDK provides a peripheral driver for the DPU Interrupt Request Steering (IRQSTEER) module of MCUXpresso SDK devices. The DPU IRQSTEER combines and routes the interrupts in DPU to other subsystems.

Functions

- static void [DPU_IRQSTEER_EnableInterrupt](#) (IRQSTEER_Type *base, DPU_IRQSTEER_IRQn_Type irq)
Enables an interrupt source.
- static void [DPU_IRQSTEER_DisableInterrupt](#) (IRQSTEER_Type *base, DPU_IRQSTEER_IRQn_Type irq)
Disables an interrupt source.
- static bool [DPU_IRQSTEER_IsInterruptSet](#) (IRQSTEER_Type *base, DPU_IRQSTEER_IRQn_Type irq)
Checks the status of one specific DPU IRQSTEER interrupt.

Driver version

- #define **FSL_DPU_IRQSTEER_DRIVER_VERSION** (MAKE_VERSION(2, 0, 0))

9.2 Function Documentation

9.2.1 static void DPU_IRQSTEER_EnableInterrupt (IRQSTEER_Type * *base*, DPU_IRQSTEER_IRQn_Type *irq*) [inline], [static]

Parameters

<i>base</i>	DPU IRQSTEER peripheral base address.
<i>irq</i>	Interrupt to be enabled.

9.2.2 static void DPU_IRQSTEER_DisableInterrupt (IRQSTEER_Type * *base*, DPU_IRQSTEER_IRQn_Type *irq*) [inline], [static]

Function Documentation

Parameters

<i>base</i>	DPU IRQSTEER peripheral base address.
<i>irq</i>	Interrupt to be disabled.

**9.2.3 static bool DPU_IRQSTEER_IsInterruptSet (IRQSTEER_Type * *base*,
DPU_IRQSTEER_IRQn_Type *irq*) [inline], [static]**

Parameters

<i>base</i>	DPU IRQSTEER peripheral base address.
<i>irq</i>	Interrupt source status to be checked. The interrupt must be an IRQSTEER source.

Returns

The interrupt status. "true" means interrupt set. "false" means not.

Chapter 10

EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

10.1 Overview

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

10.2 Typical use case

10.2.1 EDMA Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/edma

Data Structures

- struct `edma_config_t`
eDMA global configuration structure. [More...](#)
- struct `edma_transfer_config_t`
eDMA transfer configuration [More...](#)
- struct `edma_channel_Preemption_config_t`
eDMA channel priority configuration [More...](#)
- struct `edma_minor_offset_config_t`
eDMA minor offset configuration [More...](#)
- struct `edma_tcd_t`
eDMA TCD. [More...](#)
- struct `edma_handle_t`
eDMA transfer handle structure [More...](#)

Typedefs

- `typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)`
Define callback function for eDMA.
- `typedef uint32_t(* edma_memroymap_callback)(uint32_t addr)`
Memroy map function callback for DMA.

Enumerations

- enum `edma_transfer_size_t` {
 `kEDMA_TransferSize1Bytes` = 0x0U,
 `kEDMA_TransferSize2Bytes` = 0x1U,
 `kEDMA_TransferSize4Bytes` = 0x2U,
 `kEDMA_TransferSize16Bytes` = 0x4U,
 `kEDMA_TransferSize32Bytes` = 0x5U,
 `kEDMA_TransferSize64Bytes` = 0x6U }

Typical use case

eDMA transfer configuration

- enum `edma_modulo_t` {
 `kEDMA_ModuloDisable` = 0x0U,
 `kEDMA_Modulo2bytes`,
 `kEDMA_Modulo4bytes`,
 `kEDMA_Modulo8bytes`,
 `kEDMA_Modulo16bytes`,
 `kEDMA_Modulo32bytes`,
 `kEDMA_Modulo64bytes`,
 `kEDMA_Modulo128bytes`,
 `kEDMA_Modulo256bytes`,
 `kEDMA_Modulo512bytes`,
 `kEDMA_Modulo1Kbytes`,
 `kEDMA_Modulo2Kbytes`,
 `kEDMA_Modulo4Kbytes`,
 `kEDMA_Modulo8Kbytes`,
 `kEDMA_Modulo16Kbytes`,
 `kEDMA_Modulo32Kbytes`,
 `kEDMA_Modulo64Kbytes`,
 `kEDMA_Modulo128Kbytes`,
 `kEDMA_Modulo256Kbytes`,
 `kEDMA_Modulo512Kbytes`,
 `kEDMA_Modulo1Mbytes`,
 `kEDMA_Modulo2Mbytes`,
 `kEDMA_Modulo4Mbytes`,
 `kEDMA_Modulo8Mbytes`,
 `kEDMA_Modulo16Mbytes`,
 `kEDMA_Modulo32Mbytes`,
 `kEDMA_Modulo64Mbytes`,
 `kEDMA_Modulo128Mbytes`,
 `kEDMA_Modulo256Mbytes`,
 `kEDMA_Modulo512Mbytes`,
 `kEDMA_Modulo1Gbytes`,
 `kEDMA_Modulo2Gbytes` }

eDMA modulo configuration

- enum `edma_bandwidth_t` {
 `kEDMA_BandwidthStallNone` = 0x0U,
 `kEDMA_BandwidthStall4Cycle` = 0x2U,
 `kEDMA_BandwidthStall8Cycle` = 0x3U }

Bandwidth control.

- enum `edma_channel_link_type_t` {
 `kEDMA_LinkNone` = 0x0U,
 `kEDMA_MinorLink`,
 `kEDMA_MajorLink` }

Channel link type.

- enum `_edma_channel_status_flags` {

- ```
kEDMA_DoneFlag = 0x1U,
kEDMA_ErrorFlag = 0x2U,
kEDMA_InterruptFlag = 0x4U }
```

*eDMA channel status flags.*
- enum `_edma_error_status_flags` {

```
kEDMA_DestinationBusErrorFlag = DMA_MP_ES_DBE_MASK,
kEDMA_SourceBusErrorFlag = DMA_MP_ES_SBE_MASK,
kEDMA_ScatterGatherErrorFlag = DMA_MP_ES_SGE_MASK,
kEDMA_NbytesErrorFlag = DMA_MP_ES_NCE_MASK,
kEDMA_DestinationOffsetErrorFlag = DMA_MP_ES_DOE_MASK,
kEDMA_DestinationAddressErrorFlag,
kEDMA_SourceOffsetErrorFlag = DMA_MP_ES_SOE_MASK,
kEDMA_SourceAddressErrorFlag = DMA_MP_ES_SAE_MASK,
kEDMA_TransferCanceledFlag = DMA_MP_ES_ECX_MASK,
kEDMA_ErrorChannelFlag = DMA_MP_ES_ERRCHN_MASK,
kEDMA_ValidFlag = DMA_MP_ES_VLD_MASK }
```

*eDMA channel error status flags.*
- enum `_edma_channel_sys_bus_info` {

```
kEDMA_AttributeOutput = DMA_CH_SBR_ATTR_MASK,
kEDMA_PrivilegedAccessLevel = DMA_CH_SBR_PAL_MASK,
kEDMA_MasterId }
```

*eDMA channel system bus information.*
- enum `edma_interrupt_enable_t` {

```
kEDMA_ErrorInterruptEnable = 0x1U,
kEDMA_MajorInterruptEnable = DMA_TCD_CSR_INTMAJOR_MASK,
kEDMA_HalfInterruptEnable = DMA_TCD_CSR_INTHALF_MASK }
```

*eDMA interrupt source*
- enum `edma_transfer_type_t` {

```
kEDMA_MemoryToMemory = 0x0U,
kEDMA_PeripheralToMemory,
kEDMA_MemoryToPeripheral }
```

*eDMA transfer type*
- enum `_edma_transfer_status` {

```
kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),
kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }
```

*eDMA transfer status*

## Driver version

- #define `FSL_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 0))  
*eDMA driver version*

## eDMA initialization and de-initialization

- void `EDMA_Init` (DMA\_Type \*base, const `edma_config_t` \*config)  
*Initializes the eDMA peripheral.*
- void `EDMA_Deinit` (DMA\_Type \*base)

## Typical use case

- void **EDMA\_GetDefaultConfig** (*edma\_config\_t* \*config)  
*Gets the eDMA default configuration structure.*
- static void **EDMA\_EnableAllChannelLink** (DMA\_Type \*base, bool enable)  
*Enables/disables all channel linking.*

## eDMA Channel Operation

- void **EDMA\_ResetChannel** (DMA\_Type \*base, uint32\_t channel)  
*Sets all TCD registers to default values.*
- void **EDMA\_SetTransferConfig** (DMA\_Type \*base, uint32\_t channel, const *edma\_transfer\_config\_t* \*config, *edma\_tcd\_t* \*nextTcd)  
*Configures the eDMA transfer attribute.*
- void **EDMA\_SetMinorOffsetConfig** (DMA\_Type \*base, uint32\_t channel, const *edma\_minor\_offset\_config\_t* \*config)  
*Configures the eDMA minor offset feature.*
- static void **EDMA\_SetChannelArbitrationGroup** (DMA\_Type \*base, uint32\_t channel, uint32\_t group)  
*Configures the eDMA channel arbitration group.*
- static void **EDMA\_SetChannelPreemptionConfig** (DMA\_Type \*base, uint32\_t channel, const *edma\_channel\_Preemption\_config\_t* \*config)  
*Configures the eDMA channel preemption feature.*
- static uint32\_t **EDMA\_GetChannelSystemBusInformation** (DMA\_Type \*base, uint32\_t channel)  
*Gets the eDMA channel identification and attribute information on the system bus interface.*
- void **EDMA\_SetChannelLink** (DMA\_Type \*base, uint32\_t channel, *edma\_channel\_link\_type\_t* type, uint32\_t linkedChannel)  
*Sets the channel link for the eDMA transfer.*
- void **EDMA\_SetBandWidth** (DMA\_Type \*base, uint32\_t channel, *edma\_bandwidth\_t* bandWidth)  
*Sets the bandwidth for the eDMA transfer.*
- void **EDMA\_SetModulo** (DMA\_Type \*base, uint32\_t channel, *edma\_modulo\_t* srcModulo, *edma\_modulo\_t* destModulo)  
*Sets the source modulo and the destination modulo for the eDMA transfer.*
- static void **EDMA\_EnableAsyncRequest** (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables an async request for the eDMA transfer.*
- static void **EDMA\_EnableAutoStopRequest** (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables an auto stop request for the eDMA transfer.*
- void **EDMA\_EnableChannelInterrupts** (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Enables the interrupt source for the eDMA transfer.*
- void **EDMA\_DisableChannelInterrupts** (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Disables the interrupt source for the eDMA transfer.*

## eDMA TCD Operation

- void **EDMA\_TcdReset** (*edma\_tcd\_t* \*tcd)  
*Sets all fields to default values for the TCD structure.*
- void **EDMA\_TcdSetTransferConfig** (*edma\_tcd\_t* \*tcd, const *edma\_transfer\_config\_t* \*config, *edma\_tcd\_t* \*nextTcd)  
*Configures the eDMA TCD transfer attribute.*
- void **EDMA\_TcdSetMinorOffsetConfig** (*edma\_tcd\_t* \*tcd, const *edma\_minor\_offset\_config\_t* \*config)

- Configures the eDMA TCD minor offset feature.  
void [EDMA\\_TcdSetChannelLink](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_channel\\_link\\_type\\_t](#) type, [uint32\\_t](#) linkedChannel)  
*Sets the channel link for the eDMA TCD.*
- static void [EDMA\\_TcdSetBandWidth](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_bandwidth\\_t](#) bandwidth)  
*Sets the bandwidth for the eDMA TCD.*
- void [EDMA\\_TcdSetModulo](#) ([edma\\_tcd\\_t](#) \*tcd, [edma\\_modulo\\_t](#) srcModulo, [edma\\_modulo\\_t](#) destModulo)  
*Sets the source modulo and the destination modulo for the eDMA TCD.*
- static void [EDMA\\_TcdEnableAutoStopRequest](#) ([edma\\_tcd\\_t](#) \*tcd, bool enable)  
*Sets the auto stop request for the eDMA TCD.*
- void [EDMA\\_TcdEnableInterrupts](#) ([edma\\_tcd\\_t](#) \*tcd, [uint32\\_t](#) mask)  
*Enables the interrupt source for the eDMA TCD.*
- void [EDMA\\_TcdDisableInterrupts](#) ([edma\\_tcd\\_t](#) \*tcd, [uint32\\_t](#) mask)  
*Disables the interrupt source for the eDMA TCD.*

## eDMA Channel Transfer Operation

- static void [EDMA\\_EnableChannelRequest](#) ([DMA\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Enables the eDMA hardware channel request.*
- static void [EDMA\\_DisableChannelRequest](#) ([DMA\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Disables the eDMA hardware channel request.*
- static void [EDMA\\_TriggerChannelStart](#) ([DMA\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Starts the eDMA transfer by using the software trigger.*

## eDMA Channel Status Operation

- [uint32\\_t EDMA\\_GetRemainingMajorLoopCount](#) ([DMA\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Gets the Remaining major loop count from the eDMA current channel TCD.*
- static [uint32\\_t EDMA\\_GetErrorStatusFlags](#) ([DMA\\_Type](#) \*base)  
*Gets the eDMA channel error status flags.*
- [uint32\\_t EDMA\\_GetChannelStatusFlags](#) ([DMA\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Gets the eDMA channel status flags.*
- void [EDMA\\_ClearChannelStatusFlags](#) ([DMA\\_Type](#) \*base, [uint32\\_t](#) channel, [uint32\\_t](#) mask)  
*Clears the eDMA channel status flags.*

## eDMA Transactional Operation

- void [EDMA\\_CreateHandle](#) ([edma\\_handle\\_t](#) \*handle, [DMA\\_Type](#) \*base, [uint32\\_t](#) channel)  
*Creates the eDMA handle.*
- void [EDMA\\_InstallTCMDMemory](#) ([edma\\_handle\\_t](#) \*handle, [edma\\_tcd\\_t](#) \*tcdPool, [uint32\\_t](#) tcdSize)  
*Installs the TCDs memory pool into the eDMA handle.*
- void [EDMA\\_SetCallback](#) ([edma\\_handle\\_t](#) \*handle, [edma\\_callback](#) callback, void \*userData)  
*Installs a callback function for the eDMA transfer.*
- void [EDMA\\_PrepTransfer](#) ([edma\\_transfer\\_config\\_t](#) \*config, void \*srcAddr, [uint32\\_t](#) srcWidth, void \*destAddr, [uint32\\_t](#) destWidth, [uint32\\_t](#) bytesEachRequest, [uint32\\_t](#) transferBytes, [edma\\_transfer\\_type\\_t](#) type)  
*Prepares the eDMA transfer structure.*
- status\_t [EDMA\\_SubmitTransfer](#) ([edma\\_handle\\_t](#) \*handle, const [edma\\_transfer\\_config\\_t](#) \*config)  
*Submits the eDMA transfer request.*

## Data Structure Documentation

- void **EDMA\_StartTransfer** (edma\_handle\_t \*handle)  
*eDMA starts transfer.*
- void **EDMA\_StopTransfer** (edma\_handle\_t \*handle)  
*eDMA stops transfer.*
- void **EDMA\_AbortTransfer** (edma\_handle\_t \*handle)  
*eDMA aborts transfer.*
- static uint32\_t **EDMA\_GetUnusedTCDNumber** (edma\_handle\_t \*handle)  
*Get unused TCD slot number.*
- static uint32\_t **EDMA\_GetNextTCDAddress** (edma\_handle\_t \*handle)  
*Get the next tcd address.*
- void **EDMA\_HandleIRQ** (edma\_handle\_t \*handle)  
*eDMA IRQ handler for the current major loop transfer completion.*

### 10.3 Data Structure Documentation

#### 10.3.1 struct edma\_config\_t

##### Data Fields

- bool **enableMasterIdReplication**  
*Enable (true) master ID replication.*
- bool **enableHaltOnError**  
*Enable (true) transfer halt on error.*
- bool **enableRoundRobinArbitration**  
*Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection.*
- bool **enableDebugMode**  
*Enable(true) eDMA debug mode.*
- bool **enableBufferedWrites**  
*Enable(true) buffered writes.*

##### 10.3.1.0.0.16 Field Documentation

###### 10.3.1.0.0.16.1 bool edma\_config\_t::enableMasterIdReplication

If Master ID replication is disabled, the privileged protection level (supervisor mode) for DMA transfers is used.

###### 10.3.1.0.0.16.2 bool edma\_config\_t::enableHaltOnError

Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

###### 10.3.1.0.0.16.3 bool edma\_config\_t::enableDebugMode

When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

### 10.3.1.0.0.16.4 bool edma\_config\_t::enableBufferedWrites

When buffered writes are enabled, all writes except for the last write sequence of the minor loop are signaled by the eDMA as bufferable.

## 10.3.2 struct edma\_transfer\_config\_t

This structure configures the source/destination transfer attribute.

### Data Fields

- `uint32_t srcAddr`  
*Source data address.*
- `uint32_t destAddr`  
*Destination data address.*
- `edma_transfer_size_t srcTransferSize`  
*Source data transfer size.*
- `edma_transfer_size_t destTransferSize`  
*Destination data transfer size.*
- `int16_t srcOffset`  
*Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.*
- `int16_t destOffset`  
*Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.*
- `uint32_t minorLoopBytes`  
*Bytes to transfer in a minor loop.*
- `uint32_t majorLoopCounts`  
*Major loop iteration count.*

## Data Structure Documentation

### 10.3.2.0.0.17 Field Documentation

10.3.2.0.0.17.1 `uint32_t edma_transfer_config_t::srcAddr`

10.3.2.0.0.17.2 `uint32_t edma_transfer_config_t::destAddr`

10.3.2.0.0.17.3 `edma_transfer_size_t edma_transfer_config_t::srcTransferSize`

10.3.2.0.0.17.4 `edma_transfer_size_t edma_transfer_config_t::destTransferSize`

10.3.2.0.0.17.5 `int16_t edma_transfer_config_t::srcOffset`

10.3.2.0.0.17.6 `int16_t edma_transfer_config_t::destOffset`

10.3.2.0.0.17.7 `uint32_t edma_transfer_config_t::majorLoopCounts`

### 10.3.3 struct `edma_channel_Preemption_config_t`

#### Data Fields

- `bool enableChannelPreemption`  
*If true: a channel can be suspended by other channel with higher priority.*
- `bool enablePreemptAbility`  
*If true: a channel can suspend other channel with low priority.*
- `uint8_t channelPriority`  
*Channel priority.*

### 10.3.4 struct `edma_minor_offset_config_t`

#### Data Fields

- `bool enableSrcMinorOffset`  
*Enable(true) or Disable(false) source minor loop offset.*
- `bool enableDestMinorOffset`  
*Enable(true) or Disable(false) destination minor loop offset.*
- `uint32_t minorOffset`  
*Offset for a minor loop mapping.*

#### 10.3.4.0.0.18 Field Documentation

10.3.4.0.0.18.1 `bool edma_minor_offset_config_t::enableSrcMinorOffset`

10.3.4.0.0.18.2 `bool edma_minor_offset_config_t::enableDestMinorOffset`

10.3.4.0.0.18.3 `uint32_t edma_minor_offset_config_t::minorOffset`

#### 10.3.5 struct `edma_tcd_t`

This structure is same as TCD register which is described in reference manual, and is used to configure the scatter/gather feature as a next hardware TCD.

#### Data Fields

- `__IO uint32_t SADDR`  
*SADDR register, used to save source address.*
- `__IO uint16_t SOFF`  
*SOFF register, save offset bytes every transfer.*
- `__IO uint16_t ATTR`  
*ATTR register, source/destination transfer size and modulo.*
- `__IO uint32_t NBYTES`  
*Nbytes register, minor loop length in bytes.*
- `__IO uint32_t SLAST`  
*SLAST register.*
- `__IO uint32_t DADDR`  
*DADDR register, used for destination address.*
- `__IO uint16_t DOFF`  
*DOFF register, used for destination offset.*
- `__IO uint16_t CITER`  
*CITER register, current minor loop numbers, for unfinished minor loop.*
- `__IO uint32_t DLAST_SGA`  
*DLASTSGA register, next stcd address used in scatter-gather mode.*
- `__IO uint16_t CSR`  
*CSR register, for TCD control status.*
- `__IO uint16_t BITER`  
*BITER register, begin minor loop count.*

## Data Structure Documentation

### 10.3.5.0.0.19 Field Documentation

10.3.5.0.0.19.1 `__IO uint16_t edma_tcd_t::CITER`

10.3.5.0.0.19.2 `__IO uint16_t edma_tcd_t::BITER`

### 10.3.6 struct `edma_handle_t`

#### Data Fields

- `edma_callback callback`  
*Callback function for major count exhausted.*
- `void * userData`  
*Callback function parameter.*
- `DMA_Type * base`  
*eDMA peripheral base address.*
- `edma_tcd_t * tcdPool`  
*Pointer to memory stored TCDs.*
- `uint8_t channel`  
*eDMA channel number.*
- `volatile int8_t header`  
*The first TCD index.*
- `volatile int8_t tail`  
*The last TCD index.*
- `volatile int8_t tcdUsed`  
*The number of used TCD slots.*
- `volatile int8_t tcdSize`  
*The total number of TCD slots in the queue.*
- `uint8_t flags`  
*The status of the current channel.*
- `edma_memmap_callback memoryCallback`  
*Callback function for memory map convert in complex system.*

**10.3.6.0.0.20 Field Documentation****10.3.6.0.0.20.1** `edma_callback` `edma_handle_t::callback`**10.3.6.0.0.20.2** `void*` `edma_handle_t::userData`**10.3.6.0.0.20.3** `DMA_Type*` `edma_handle_t::base`**10.3.6.0.0.20.4** `edma_tcd_t*` `edma_handle_t::tcdPool`**10.3.6.0.0.20.5** `uint8_t` `edma_handle_t::channel`**10.3.6.0.0.20.6** `volatile int8_t` `edma_handle_t::header`**10.3.6.0.0.20.7** `volatile int8_t` `edma_handle_t::tail`**10.3.6.0.0.20.8** `volatile int8_t` `edma_handle_t::tcdUsed`**10.3.6.0.0.20.9** `volatile int8_t` `edma_handle_t::tcdSize`**10.3.6.0.0.20.10** `uint8_t` `edma_handle_t::flags`**10.4 Macro Definition Documentation****10.4.1** `#define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

Version 2.1.0.

**10.5 Typedef Documentation****10.5.1** `typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)`**10.6 Enumeration Type Documentation****10.6.1** `enum edma_transfer_size_t`

Enumerator

|                                        |                                                               |
|----------------------------------------|---------------------------------------------------------------|
| <code>kEDMA_TransferSize1Bytes</code>  | Source/Destination data transfer size is 1 byte every time.   |
| <code>kEDMA_TransferSize2Bytes</code>  | Source/Destination data transfer size is 2 bytes every time.  |
| <code>kEDMA_TransferSize4Bytes</code>  | Source/Destination data transfer size is 4 bytes every time.  |
| <code>kEDMA_TransferSize16Bytes</code> | Source/Destination data transfer size is 16 bytes every time. |
| <code>kEDMA_TransferSize32Bytes</code> | Source/Destination data transfer size is 32 bytes every time. |
| <code>kEDMA_TransferSize64Bytes</code> | Source/Destination data transfer size is 64 bytes every time. |

## Enumeration Type Documentation

### 10.6.2 enum edma\_modulo\_t

Enumerator

|                              |                                      |
|------------------------------|--------------------------------------|
| <i>kEDMA_ModuloDisable</i>   | Disable modulo.                      |
| <i>kEDMA_Modulo2bytes</i>    | Circular buffer size is 2 bytes.     |
| <i>kEDMA_Modulo4bytes</i>    | Circular buffer size is 4 bytes.     |
| <i>kEDMA_Modulo8bytes</i>    | Circular buffer size is 8 bytes.     |
| <i>kEDMA_Modulo16bytes</i>   | Circular buffer size is 16 bytes.    |
| <i>kEDMA_Modulo32bytes</i>   | Circular buffer size is 32 bytes.    |
| <i>kEDMA_Modulo64bytes</i>   | Circular buffer size is 64 bytes.    |
| <i>kEDMA_Modulo128bytes</i>  | Circular buffer size is 128 bytes.   |
| <i>kEDMA_Modulo256bytes</i>  | Circular buffer size is 256 bytes.   |
| <i>kEDMA_Modulo512bytes</i>  | Circular buffer size is 512 bytes.   |
| <i>kEDMA_Modulo1Kbytes</i>   | Circular buffer size is 1 K bytes.   |
| <i>kEDMA_Modulo2Kbytes</i>   | Circular buffer size is 2 K bytes.   |
| <i>kEDMA_Modulo4Kbytes</i>   | Circular buffer size is 4 K bytes.   |
| <i>kEDMA_Modulo8Kbytes</i>   | Circular buffer size is 8 K bytes.   |
| <i>kEDMA_Modulo16Kbytes</i>  | Circular buffer size is 16 K bytes.  |
| <i>kEDMA_Modulo32Kbytes</i>  | Circular buffer size is 32 K bytes.  |
| <i>kEDMA_Modulo64Kbytes</i>  | Circular buffer size is 64 K bytes.  |
| <i>kEDMA_Modulo128Kbytes</i> | Circular buffer size is 128 K bytes. |
| <i>kEDMA_Modulo256Kbytes</i> | Circular buffer size is 256 K bytes. |
| <i>kEDMA_Modulo512Kbytes</i> | Circular buffer size is 512 K bytes. |
| <i>kEDMA_Modulo1Mbytes</i>   | Circular buffer size is 1 M bytes.   |
| <i>kEDMA_Modulo2Mbytes</i>   | Circular buffer size is 2 M bytes.   |
| <i>kEDMA_Modulo4Mbytes</i>   | Circular buffer size is 4 M bytes.   |
| <i>kEDMA_Modulo8Mbytes</i>   | Circular buffer size is 8 M bytes.   |
| <i>kEDMA_Modulo16Mbytes</i>  | Circular buffer size is 16 M bytes.  |
| <i>kEDMA_Modulo32Mbytes</i>  | Circular buffer size is 32 M bytes.  |
| <i>kEDMA_Modulo64Mbytes</i>  | Circular buffer size is 64 M bytes.  |
| <i>kEDMA_Modulo128Mbytes</i> | Circular buffer size is 128 M bytes. |
| <i>kEDMA_Modulo256Mbytes</i> | Circular buffer size is 256 M bytes. |
| <i>kEDMA_Modulo512Mbytes</i> | Circular buffer size is 512 M bytes. |
| <i>kEDMA_Modulo1Gbytes</i>   | Circular buffer size is 1 G bytes.   |
| <i>kEDMA_Modulo2Gbytes</i>   | Circular buffer size is 2 G bytes.   |

### 10.6.3 enum edma\_bandwidth\_t

Enumerator

|                                   |                                                        |
|-----------------------------------|--------------------------------------------------------|
| <i>kEDMA_BandwidthStallNone</i>   | No eDMA engine stalls.                                 |
| <i>kEDMA_BandwidthStall4Cycle</i> | eDMA engine stalls for 4 cycles after each read/write. |
| <i>kEDMA_BandwidthStall8Cycle</i> | eDMA engine stalls for 8 cycles after each read/write. |

#### 10.6.4 enum edma\_channel\_link\_type\_t

Enumerator

*kEDMA\_LinkNone* No channel link.

*kEDMA\_MinorLink* Channel link after each minor loop.

*kEDMA\_MajorLink* Channel link while major loop count exhausted.

#### 10.6.5 enum \_edma\_channel\_status\_flags

Enumerator

*kEDMA\_DoneFlag* DONE flag, set while transfer finished, CITER value exhausted.

*kEDMA\_ErrorFlag* eDMA error flag, an error occurred in a transfer

*kEDMA\_InterruptFlag* eDMA interrupt flag, set while an interrupt occurred of this channel

#### 10.6.6 enum \_edma\_error\_status\_flags

Enumerator

*kEDMA\_DestinationBusErrorFlag* Bus error on destination address.

*kEDMA\_SourceBusErrorFlag* Bus error on the source address.

*kEDMA\_ScatterGatherErrorFlag* Error on the Scatter/Gather address, not 32byte aligned.

*kEDMA\_NbytesErrorFlag* NBYTES/CITER configuration error.

*kEDMA\_DestinationOffsetErrorFlag* Destination offset not aligned with destination size.

*kEDMA\_DestinationAddressErrorFlag* Destination address not aligned with destination size.

*kEDMA\_SourceOffsetErrorFlag* Source offset not aligned with source size.

*kEDMA\_SourceAddressErrorFlag* Source address not aligned with source size.

*kEDMA\_TransferCanceledFlag* Transfer cancelled.

*kEDMA\_ErrorChannelFlag* Error channel number of the cancelled channel number.

*kEDMA\_ValidFlag* No error occurred, this bit is 0. Otherwise, it is 1.

#### 10.6.7 enum \_edma\_channel\_sys\_bus\_info

Enumerator

*kEDMA\_AttributeOutput* DMA's AHB system bus attribute output value.

*kEDMA\_PrivilegedAccessLevel* Privileged Access Level for DMA transfers. 0b - User protection level; 1b - Privileged protection level.

*kEDMA\_MasterId* DMA's master ID when channel is active and master ID replication is enabled.

## Function Documentation

### 10.6.8 enum edma\_interrupt\_enable\_t

Enumerator

*kEDMA\_ErrorInterruptEnable* Enable interrupt while channel error occurs.

*kEDMA\_MajorInterruptEnable* Enable interrupt while major count exhausted.

*kEDMA\_HalfInterruptEnable* Enable interrupt while major count to half value.

### 10.6.9 enum edma\_transfer\_type\_t

Enumerator

*kEDMA\_MemoryToMemory* Transfer from memory to memory.

*kEDMA\_PeripheralToMemory* Transfer from peripheral to memory.

*kEDMA\_MemoryToPeripheral* Transfer from memory to peripheral.

### 10.6.10 enum \_edma\_transfer\_status

Enumerator

*kStatus\_EDMA\_QueueFull* TCD queue is full.

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

## 10.7 Function Documentation

### 10.7.1 void EDMA\_Init ( DMA\_Type \* *base*, const edma\_config\_t \* *config* )

This function ungates the eDMA clock and configures the eDMA peripheral according to the configuration structure.

Parameters

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <i>base</i>   | eDMA peripheral base address.                                  |
| <i>config</i> | A pointer to the configuration structure, see "edma_config_t". |

Note

This function enables the minor loop map feature.

### 10.7.2 void EDMA\_Deinit ( DMA\_Type \* *base* )

This function gates the eDMA clock.

Parameters

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

### 10.7.3 void EDMA\_GetDefaultConfig ( **edma\_config\_t \* config** )

This function sets the configuration structure to default values. The default configuration is set to the following values:

```
* config.enableMasterIdReplication = true;
* config.enableHaltOnError = true;
* config.enableRoundRobinArbitration = false;
* config.enableDebugMode = false;
* config.enableBufferedWrites = false;
*
```

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>config</i> | A pointer to the eDMA configuration structure. |
|---------------|------------------------------------------------|

### 10.7.4 static void EDMA\_EnableAllChannelLink ( **DMA\_Type \* base, bool enable** ) [inline], [static]

This function enables/disables all channel linking in the management page. For specific channel linking enablement & configuration, please refer to EDMA\_SetChannelLink and EDMA\_TcdSetChannelLink APIs.

For example, to disable all channel linking in the DMA0 management page:

```
* EDMA_EnableAllChannelLink(DMA0, false);
*
```

Parameters

|               |                                                                                                      |
|---------------|------------------------------------------------------------------------------------------------------|
| <i>base</i>   | eDMA peripheral base address.                                                                        |
| <i>enable</i> | Switcher of the channel linking feature for all channels. "true" means to enable. "false" means not. |

### 10.7.5 void EDMA\_ResetChannel ( **DMA\_Type \* base, uint32\_t channel** )

This function sets TCD registers for this channel to default values.

## Function Documentation

### Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | eDMA peripheral base address. |
| <i>channel</i> | eDMA channel number.          |

### Note

This function must not be called while the channel transfer is ongoing or it causes unpredictable results.

This function enables the auto stop request feature.

### **10.7.6 void EDMA\_SetTransferConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_transfer\_config\_t \* *config*, edma\_tcd\_t \* *nextTcd* )**

This function configures the transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the TCD address. Example:

```
* edma_transfer_config_t config;
* edma_tcd_t tcd;
* config.srcAddr = ...;
* config.destAddr = ...;
*
* ...
* EDMA_SetTransferConfig(DMA0, channel, &config, &tcd);
*
```

### Parameters

|                |                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                 |
| <i>channel</i> | eDMA channel number.                                                                          |
| <i>config</i>  | Pointer to eDMA transfer configuration structure.                                             |
| <i>nextTcd</i> | Point to TCD structure. It can be NULL if users do not want to enable scatter/gather feature. |

### Note

If nextTcd is not NULL, it means scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the eDMA\_ResetChannel.

### **10.7.7 void EDMA\_SetMinorOffsetConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_minor\_offset\_config\_t \* *config* )**

The minor offset means that the signed-extended value is added to the source address or destination address after each minor loop.

Parameters

|                |                                                        |
|----------------|--------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                          |
| <i>channel</i> | eDMA channel number.                                   |
| <i>config</i>  | A pointer to the minor offset configuration structure. |

#### 10.7.8 static void EDMA\_SetChannelArbitrationGroup ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *group* ) [inline], [static]

This function configures the channel arbitration group. The arbitration group priorities are evaluated by numeric value from highest group number to lowest.

Parameters

|                |                                                          |
|----------------|----------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                            |
| <i>channel</i> | eDMA channel number                                      |
| <i>group</i>   | Fixed-priority arbitration group number for the channel. |

#### 10.7.9 static void EDMA\_SetChannelPreemptionConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const edma\_channel\_Preemption\_config\_t \* *config* ) [inline], [static]

This function configures the channel preemption attribute and the priority of the channel.

Parameters

|                |                                                              |
|----------------|--------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                |
| <i>channel</i> | eDMA channel number                                          |
| <i>config</i>  | A pointer to the channel preemption configuration structure. |

#### 10.7.10 static uint32\_t EDMA\_GetChannelSystemBusInformation ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

## Function Documentation

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | eDMA peripheral base address. |
| <i>channel</i> | eDMA channel number.          |

Returns

The mask of the channel system bus information. Users need to use the \_edma\_channel\_sys\_bus\_info type to decode the return variables.

### 10.7.11 void EDMA\_SetChannelLink ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_channel\_link\_type\_t *type*, uint32\_t *linkedChannel* )

This function configures either the minor link or the major link mode. The minor link means that the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Parameters

|                      |                                                                                                                                                                              |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | eDMA peripheral base address.                                                                                                                                                |
| <i>channel</i>       | eDMA channel number.                                                                                                                                                         |
| <i>type</i>          | A channel link type, which can be one of the following: <ul style="list-style-type: none"><li>• kEDMA_LinkNone</li><li>• kEDMA_MinorLink</li><li>• kEDMA_MajorLink</li></ul> |
| <i>linkedChannel</i> | The linked channel number.                                                                                                                                                   |

Note

Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

### 10.7.12 void EDMA\_SetBandWidth ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_bandwidth\_t *bandWidth* )

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

|                  |                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | eDMA peripheral base address.                                                                                                                                                                                 |
| <i>channel</i>   | eDMA channel number.                                                                                                                                                                                          |
| <i>bandWidth</i> | A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"> <li>• kEDMABandwidthStallNone</li> <li>• kEDMABandwidthStall4Cycle</li> <li>• kEDMABandwidthStall8Cycle</li> </ul> |

#### 10.7.13 void EDMA\_SetModulo ( DMA\_Type \* *base*, uint32\_t *channel*, edma\_modulo\_t *srcModulo*, edma\_modulo\_t *destModulo* )

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | eDMA peripheral base address. |
| <i>channel</i>    | eDMA channel number.          |
| <i>srcModulo</i>  | A source modulo value.        |
| <i>destModulo</i> | A destination modulo value.   |

#### 10.7.14 static void EDMA\_EnableAsyncRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                    |
| <i>channel</i> | eDMA channel number.                             |
| <i>enable</i>  | The command to enable (true) or disable (false). |

#### 10.7.15 static void EDMA\_EnableAutoStopRequest ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

## Function Documentation

Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                    |
| <i>channel</i> | eDMA channel number.                             |
| <i>enable</i>  | The command to enable (true) or disable (false). |

### 10.7.16 void EDMA\_EnableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )

Parameters

|                |                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                       |
| <i>channel</i> | eDMA channel number.                                                                                |
| <i>mask</i>    | The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type. |

### 10.7.17 void EDMA\_DisableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )

Parameters

|                |                                                                                           |
|----------------|-------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                             |
| <i>channel</i> | eDMA channel number.                                                                      |
| <i>mask</i>    | The mask of the interrupt source to be set. Use the defined edma_interrupt_enable_t type. |

### 10.7.18 void EDMA\_TcdReset ( edma\_tcd\_t \* *tcd* )

This function sets all fields for this TCD structure to default value.

Parameters

|            |                               |
|------------|-------------------------------|
| <i>tcd</i> | Pointer to the TCD structure. |
|------------|-------------------------------|

## Note

This function enables the auto stop request feature.

### 10.7.19 void EDMA\_TcdSetTransferConfig ( *edma\_tcd\_t* \* *tcd*, *const edma\_transfer\_config\_t* \* *config*, *edma\_tcd\_t* \* *nextTcd* )

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The STCD is used in the scatter-gather mode. This function configures the TCD transfer attribute, including source address, destination address, transfer size, address offset, and so on. It also configures the scatter gather feature if the user supplies the next TCD address. Example:

```
* edma_transfer_config_t config = {
* ...
* }
* edma_tcd_t tcd __aligned(32);
* edma_tcd_t nextTcd __aligned(32);
* EDMA_TcdSetTransferConfig(&tcd, &config, &nextTcd);
*
```

## Parameters

|                |                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------|
| <i>tcd</i>     | Pointer to the TCD structure.                                                                            |
| <i>config</i>  | Pointer to eDMA transfer configuration structure.                                                        |
| <i>nextTcd</i> | Pointer to the next TCD structure. It can be NULL if users do not want to enable scatter/gather feature. |

## Note

TCD address should be 32 bytes aligned or it causes an eDMA error.

If the nextTcd is not NULL, the scatter gather feature is enabled and DREQ bit is cleared in the previous transfer configuration, which is set in the EDMA\_TcdReset.

### 10.7.20 void EDMA\_TcdSetMinorOffsetConfig ( *edma\_tcd\_t* \* *tcd*, *const edma\_minor\_offset\_config\_t* \* *config* )

A minor offset is a signed-extended value added to the source address or a destination address after each minor loop.

## Function Documentation

Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>tcd</i>    | A point to the TCD structure.                          |
| <i>config</i> | A pointer to the minor offset configuration structure. |

### **10.7.21 void EDMA\_TcdSetChannelLink ( *edma\_tcd\_t* \* *tcd*, *edma\_channel\_link\_type\_t* *type*, *uint32\_t* *linkedChannel* )**

This function configures either a minor link or a major link. The minor link means the channel link is triggered every time CITER decreases by 1. The major link means that the channel link is triggered when the CITER is exhausted.

Note

Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

Parameters

|                      |                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>           | Point to the TCD structure.                                                                                                                               |
| <i>type</i>          | Channel link type, it can be one of: <ul style="list-style-type: none"><li>• kEDMA_LinkNone</li><li>• kEDMA_MinorLink</li><li>• kEDMA_MajorLink</li></ul> |
| <i>linkedChannel</i> | The linked channel number.                                                                                                                                |

### **10.7.22 static void EDMA\_TcdSetBandWidth ( *edma\_tcd\_t* \* *tcd*, *edma\_bandwidth\_t* *bandWidth* ) [inline], [static]**

Because the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. The bandwidth forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

Parameters

|                  |                                                                                                                                                                                                           |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>       | A pointer to the TCD structure.                                                                                                                                                                           |
| <i>bandWidth</i> | A bandwidth setting, which can be one of the following: <ul style="list-style-type: none"><li>• kEDMABandwidthStallNone</li><li>• kEDMABandwidthStall4Cycle</li><li>• kEDMABandwidthStall8Cycle</li></ul> |

### 10.7.23 void EDMA\_TcdSetModulo ( *edma\_tcd\_t \* tcd*, *edma\_modulo\_t srcModulo*, *edma\_modulo\_t destModulo* )

This function defines a specific address range specified to be the value after (SADDR + SOFF)/(DADDR + DOFF) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>tcd</i>        | A pointer to the TCD structure. |
| <i>srcModulo</i>  | A source modulo value.          |
| <i>destModulo</i> | A destination modulo value.     |

### 10.7.24 static void EDMA\_TcdEnableAutoStopRequest ( *edma\_tcd\_t \* tcd*, *bool enable* ) [inline], [static]

If enabling the auto stop request, the eDMA hardware automatically disables the hardware channel request.

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>tcd</i>    | A pointer to the TCD structure.                  |
| <i>enable</i> | The command to enable (true) or disable (false). |

### 10.7.25 void EDMA\_TcdEnableInterrupts ( *edma\_tcd\_t \* tcd*, *uint32\_t mask* )

Parameters

|             |                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------|
| <i>tcd</i>  | Point to the TCD structure.                                                                                |
| <i>mask</i> | The mask of interrupt source to be set. Users need to use the defined <i>edma_interrupt_enable_t</i> type. |

### 10.7.26 void EDMA\_TcdDisableInterrupts ( *edma\_tcd\_t \* tcd*, *uint32\_t mask* )

## Function Documentation

Parameters

|             |                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------|
| <i>tcd</i>  | Point to the TCD structure.                                                                         |
| <i>mask</i> | The mask of interrupt source to be set. Users need to use the defined edma_interrupt_enable_t type. |

### 10.7.27 static void EDMA\_EnableChannelRequest ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function enables the hardware channel request.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | eDMA peripheral base address. |
| <i>channel</i> | eDMA channel number.          |

### 10.7.28 static void EDMA\_DisableChannelRequest ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function disables the hardware channel request.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | eDMA peripheral base address. |
| <i>channel</i> | eDMA channel number.          |

### 10.7.29 static void EDMA\_TriggerChannelStart ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This function starts a minor loop transfer.

Parameters

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

|                |                      |
|----------------|----------------------|
| <i>channel</i> | eDMA channel number. |
|----------------|----------------------|

### 10.7.30 **uint32\_t EDMA\_GetRemainingMajorLoopCount ( DMA\_Type \* *base*, uint32\_t *channel* )**

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the number of major loop count that has not finished.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | eDMA peripheral base address. |
| <i>channel</i> | eDMA channel number.          |

Returns

Major loop count which has not been transferred yet for the current TCD.

Note

1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.
1. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA\_TCDn\_NBYTES\_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma\_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount \* NBYTES(initially configured)

### 10.7.31 **static uint32\_t EDMA\_GetErrorStatusFlags ( DMA\_Type \* *base* ) [inline], [static]**

## Function Documentation

Parameters

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

Returns

The mask of error status flags. Users need to use the \_edma\_error\_status\_flags type to decode the return variables.

### 10.7.32 **uint32\_t EDMA\_GetChannelStatusFlags ( DMA\_Type \* *base*, uint32\_t *channel* )**

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | eDMA peripheral base address. |
| <i>channel</i> | eDMA channel number.          |

Returns

The mask of channel status flags. Users need to use the \_edma\_channel\_status\_flags type to decode the return variables.

### 10.7.33 **void EDMA\_ClearChannelStatusFlags ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *mask* )**

Parameters

|                |                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------|
| <i>base</i>    | eDMA peripheral base address.                                                                            |
| <i>channel</i> | eDMA channel number.                                                                                     |
| <i>mask</i>    | The mask of channel status to be cleared. Users need to use the defined _edma_channel_status_flags type. |

### 10.7.34 **void EDMA\_CreateHandle ( edma\_handle\_t \* *handle*, DMA\_Type \* *base*, uint32\_t *channel* )**

This function is called if using the transactional API for eDMA. This function initializes the internal state of the eDMA handle.

Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>handle</i>  | eDMA handle pointer. The eDMA handle stores callback function and parameters. |
| <i>base</i>    | eDMA peripheral base address.                                                 |
| <i>channel</i> | eDMA channel number.                                                          |

#### **10.7.35 void EDMA\_InstallTCDMemory( edma\_handle\_t \* *handle*, edma\_tcd\_t \* *tcdPool*, uint32\_t *tcdSize* )**

This function is called after the EDMA\_CreateHandle to use scatter/gather feature.

Parameters

|                |                                                           |
|----------------|-----------------------------------------------------------|
| <i>handle</i>  | eDMA handle pointer.                                      |
| <i>tcdPool</i> | A memory pool to store TCDs. It must be 32 bytes aligned. |
| <i>tcdSize</i> | The number of TCD slots.                                  |

#### **10.7.36 void EDMA\_SetCallback( edma\_handle\_t \* *handle*, edma\_callback *callback*, void \* *userData* )**

This callback is called in the eDMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>handle</i>   | eDMA handle pointer.                   |
| <i>callback</i> | eDMA callback function pointer.        |
| <i>userData</i> | A parameter for the callback function. |

#### **10.7.37 void EDMA\_PreparesTransfer( edma\_transfer\_config\_t \* *config*, void \* *srcAddr*, uint32\_t *srcWidth*, void \* *destAddr*, uint32\_t *destWidth*, uint32\_t *bytesEachRequest*, uint32\_t *transferBytes*, edma\_transfer\_type\_t *type* )**

This function prepares the transfer configuration structure according to the user input.

## Function Documentation

Parameters

|                         |                                                                                   |
|-------------------------|-----------------------------------------------------------------------------------|
| <i>config</i>           | The user configuration structure of type <a href="#">edma_transfer_config_t</a> . |
| <i>srcAddr</i>          | eDMA transfer source address.                                                     |
| <i>srcWidth</i>         | eDMA transfer source address width(bytes).                                        |
| <i>destAddr</i>         | eDMA transfer destination address.                                                |
| <i>destWidth</i>        | eDMA transfer destination address width(bytes).                                   |
| <i>bytesEachRequest</i> | eDMA transfer bytes per channel request.                                          |
| <i>transferBytes</i>    | eDMA transfer bytes to be transferred.                                            |
| <i>type</i>             | eDMA transfer type.                                                               |

Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, the source address must be 4 bytes aligned, or it results in source address error (SAE).

### 10.7.38 **status\_t EDMA\_SubmitTransfer ( *edma\_handle\_t \* handle*, *const edma\_transfer\_config\_t \* config* )**

This function submits the eDMA transfer request according to the transfer configuration structure. If submitting 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> | eDMA handle pointer.                              |
| <i>config</i> | Pointer to eDMA transfer configuration structure. |

Return values

|                                |                                                                     |
|--------------------------------|---------------------------------------------------------------------|
| <i>kStatus_EDMA_Success</i>    | It means submit transfer request succeed.                           |
| <i>kStatus_EDMA_Queue-Full</i> | It means TCD queue is full. Submit transfer request is not allowed. |
| <i>kStatus_EDMA_Busy</i>       | It means the given channel is busy, need to submit request later.   |

**10.7.39 void EDMA\_StartTransfer ( *edma\_handle\_t* \* *handle* )**

This function enables the channel request. Users can call this function after submitting the transfer request or before submitting the transfer request.

## Function Documentation

Parameters

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

### 10.7.40 void EDMA\_StopTransfer ( *edma\_handle\_t \* handle* )

This function disables the channel request to pause the transfer. Users can call [EDMA\\_StartTransfer\(\)](#) again to resume the transfer.

Parameters

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

### 10.7.41 void EDMA\_AbortTransfer ( *edma\_handle\_t \* handle* )

This function disables the channel request and clear transfer status bits. Users can submit another transfer after calling this API.

Parameters

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

### 10.7.42 static uint32\_t EDMA\_GetUnusedTCDNumber ( *edma\_handle\_t \* handle* ) [**inline**], [**static**]

This function gets current tcd index which is run. If the TCD pool pointer is NULL, it will return 0.

Parameters

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

Returns

The unused tcd slot number.

### 10.7.43 static uint32\_t EDMA\_GetNextTCDAccount ( *edma\_handle\_t \* handle* ) [**inline**], [**static**]

This function gets the next tcd address. If this is last TCD, return 0.

Parameters

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

Returns

The next TCD address.

#### **10.7.44 void EDMA\_HandleIRQ ( edma\_handle\_t \* *handle* )**

This function clears the channel major interrupt flag and calls the callback function if it is not NULL.

Parameters

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

## Function Documentation

# Chapter 11

## ENET: Ethernet MAC Driver

### 11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 10/100 Mbps Ethernet MAC (ENET) module of MCUXpresso SDK devices.

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.

This group sets/gets the ENET mac address and the multicast group address filter. [ENET\\_AddMulticast-Group\(\)](#) should be called to add the ENET MAC to the multicast group. The IEEE 1588 feature requires receiving the PTP message.

This group has the receive active API [ENET\\_ActiveRead\(\)](#) and [ENET\\_ActiveReadMultiRing\(\)](#) 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.

#### 1. For single ring

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.

#### 1. For multiple-ring supported

The ENET driver now added a series transactional APIs with postfix "MultiRing" to support the extended multiple-ring for AVB feature. There are extended multiple-ring functions for receive side: [ENET\\_GetRxErrBeforeReadFrameMultiRing\(\)](#), [ENET\\_GetRxFrameSizeMultiRing\(\)](#), and [ENET\\_ReadFrameMultiRing\(\)](#). They are the similar to the single ring receive APIs and only add the "ringId" input param to identify the different ring index. For TX side add the [ENET\\_SendFrameMultiRing](#), [ENET\\_GetTxErr-](#)

## Typical use case

After [SendFrameMultiRing\(\)](#). They are similar to the single ring transmit APIs and only add the "ring-Id" input param to identify the different ring index.

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. The [ENET\\_GetRxFrameTime\(\)](#) and [ENET\\_GetTxFrameTime\(\)](#) functions are called by the PTP stack to get the timestamp captured by the ENET driver.

## 11.2 Typical use case

### 11.2.1 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\\_buffer\\_config\\_t](#)  
*Defines the receive buffer descriptor configuration structure. [More...](#)*
- struct [enet\\_ptp\\_time\\_t](#)  
*Defines the ENET PTP time stamp structure. [More...](#)*
- struct [enet\\_ptp\\_time\\_data\\_t](#)  
*Defines the structure for the ENET PTP message data and timestamp data. [More...](#)*
- struct [enet\\_ptp\\_time\\_data\\_ring\\_t](#)  
*Defines the ENET PTP ring buffer structure for the PTP message timestamp store. [More...](#)*
- struct [enet\\_ptp\\_config\\_t](#)  
*Defines the ENET PTP configuration structure. [More...](#)*
- struct [enet\\_intcoalesce\\_config\\_t](#)  
*Defines the interrupt coalescing configure structure. [More...](#)*
- struct [enet\\_avb\\_config\\_t](#)  
*Defines the ENET AVB Configure structure. [More...](#)*
- struct [enet\\_config\\_t](#)  
*Defines the basic configuration structure for the ENET device. [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\_callback\_t )(ENET\_Type \*base, enet\_handle\_t \*handle, uint32\_t ringId, enet\_event\_t event, void \*userData)  
*ENET callback function.*

## Enumerations

- enum \_enet\_status {
   
   kStatus\_ENET\_RxFrameError = MAKE\_STATUS(kStatusGroup\_ENET, 0U),
   kStatus\_ENET\_RxFrameFail = MAKE\_STATUS(kStatusGroup\_ENET, 1U),
   kStatus\_ENET\_RxFrameEmpty = MAKE\_STATUS(kStatusGroup\_ENET, 2U),
   kStatus\_ENET\_TxFrameOverLen = MAKE\_STATUS(kStatusGroup\_ENET, 3U),
   kStatus\_ENET\_TxFrameBusy = MAKE\_STATUS(kStatusGroup\_ENET, 4U),
   kStatus\_ENET\_TxFrameFail = MAKE\_STATUS(kStatusGroup\_ENET, 5U),
   kStatus\_ENET\_PtpTsRingFull = MAKE\_STATUS(kStatusGroup\_ENET, 6U),
   kStatus\_ENET\_PtpTsRingEmpty = MAKE\_STATUS(kStatusGroup\_ENET, 7U) }
   
*Defines the status return codes for transaction.*
- enum enet\_mii\_mode\_t {
   
   kENET\_MiiMode = 0U,
   kENET\_RmiiMode = 1U,
   kENET\_RgmiiMode = 2U }
   
*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,
   kENET\_MiiSpeed1000M = 2U }
   
*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,

## Typical use case

```
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,  
    kENET\_ControlSVLANEnable = 0x0400U,  
    kENET\_ControlVLANUseSecondTag = 0x0800U }

*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\_RxFlush2Interrupt = ENET\_EIR\_RXFLUSH\_2\_MASK,  
    kENET\_RxFlush1Interrupt = ENET\_EIR\_RXFLUSH\_1\_MASK,  
    kENET\_RxFlush0Interrupt = ENET\_EIR\_RXFLUSH\_0\_MASK,  
    kENET\_TxFrame2Interrupt = ENET\_EIR\_TXF2\_MASK,  
    kENET\_TxBuffer2Interrupt = ENET\_EIR\_TXB2\_MASK,  
    kENET\_RxFrame2Interrupt = ENET\_EIR\_RXF2\_MASK,  
    kENET\_RxBuffer2Interrupt = ENET\_EIR\_RXB2\_MASK,  
    kENET\_TxFrame1Interrupt = ENET\_EIR\_TXF1\_MASK,  
    kENET\_TxBuffer1Interrupt = ENET\_EIR\_TXB1\_MASK,  
    kENET\_RxFrame1Interrupt = ENET\_EIR\_RXF1\_MASK,  
    kENET\_RxBuffer1Interrupt = ENET\_EIR\_RXB1\_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_idle_slope_t` {
   
kENET\_IdleSlope1 = 1U,
   
kENET\_IdleSlope2 = 2U,
   
kENET\_IdleSlope4 = 4U,
   
kENET\_IdleSlope8 = 8U,
   
kENET\_IdleSlope16 = 16U,
   
kENET\_IdleSlope32 = 32U,
   
kENET\_IdleSlope64 = 64U,
   
kENET\_IdleSlope128 = 128U,
   
kENET\_IdleSlope256 = 256U,
   
kENET\_IdleSlope384 = 384U,
   
kENET\_IdleSlope512 = 512U,
   
kENET\_IdleSlope640 = 640U,
   
kENET\_IdleSlope768 = 768U,
   
kENET\_IdleSlope896 = 896U,
   
kENET\_IdleSlope1024 = 1024U,
   
kENET\_IdleSlope1152 = 1152U,
   
kENET\_IdleSlope1280 = 1280U,
   
kENET\_IdleSlope1408 = 1408U,
   
kENET\_IdleSlope1536 = 1536U }

*Defines certain idle slope for bandwidth fraction.*

- 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.*

- enum `enet_ptp_event_type_t` {
   
kENET\_PtpEventMsgType = 3U,
   
kENET\_PtpSrcPortIdLen = 10U,
   
kENET\_PtpEventPort = 319U,
   
kENET\_PtpGnrlPort = 320U }

## Typical use case

- *Defines the ENET PTP message related constant.*
  - enum `enet_ptp_timer_channel_t` {  
    kENET\_PtpTimerChannel1 = 0U,  
    kENET\_PtpTimerChannel2,  
    kENET\_PtpTimerChannel3,  
    kENET\_PtpTimerChannel4 }
- *Defines the IEEE 1588 PTP timer channel numbers.*
  - enum `enet_ptp_timer_channel_mode_t` {  
    kENET\_PtpChannelDisable = 0U,  
    kENET\_PtpChannelRisingCapture = 1U,  
    kENET\_PtpChannelFallingCapture = 2U,  
    kENET\_PtpChannelBothCapture = 3U,  
    kENET\_PtpChannelSoftCompare = 4U,  
    kENET\_PtpChannelToggleCompare = 5U,  
    kENET\_PtpChannelClearCompare = 6U,  
    kENET\_PtpChannelSetCompare = 7U,  
    kENET\_PtpChannelClearCompareSetOverflow = 10U,  
    kENET\_PtpChannelSetCompareClearOverflow = 11U,  
    kENET\_PtpChannelPulseLowonCompare = 14U,  
    kENET\_PtpChannelPulseHighonCompare = 15U }
- *Defines the capture or compare mode for IEEE 1588 PTP timer channels.*

## Driver version

- #define `FSL_ENET_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 3))  
*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\_SOFTOWENER1\_MASK 0x4000U  
*Software owner one mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_WRAP\_MASK 0x2000U  
*Wrap buffer descriptor mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_SOFTOWENER2\_MASK 0x1000U  
*Software owner two mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_LAST\_MASK 0x0800U  
*Last BD of the frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_TRANMICRC\_MASK 0x0400U  
*Transmit CRC mask.*

## First extended control region bit masks of the receive buffer descriptor.

- #define ENET\_BUFFDESCRIPTOR\_RX\_IPV4\_MASK 0x0001U  
*Ipv4 frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_IPV6\_MASK 0x0002U  
*Ipv6 frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_VLAN\_MASK 0x0004U  
*VLAN frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_PROTOCOLCHECKSUM\_MASK 0x0010U  
*Protocol checksum error mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_IPHEADCHECKSUM\_MASK 0x0020U  
*IP header checksum error mask.*

## Second extended control region bit masks of the receive buffer descriptor.

- #define ENET\_BUFFDESCRIPTOR\_RX\_INTERRUPT\_MASK 0x0080U  
*BD interrupt mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_UNICAST\_MASK 0x0100U  
*Unicast frame mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_COLLISION\_MASK 0x0200U  
*BD collision mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_PHYERR\_MASK 0x0400U  
*PHY error mask.*
- #define ENET\_BUFFDESCRIPTOR\_RX\_MACERR\_MASK 0x8000U  
*Mac error mask.*

## Typical use case

### First extended control region bit masks of the transmit buffer descriptor.

- #define ENET\_BUFFDESCRIPTOR\_TX\_ERR\_MASK 0x8000U  
*Transmit error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_UNDERFLOWERR\_MASK 0x2000U  
*Underflow error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_EXCCOLLISIONERR\_MASK 0x1000U  
*Excess collision error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_FRAMEERR\_MASK 0x0800U  
*Frame error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_LATECOLLISIONERR\_MASK 0x0400U  
*Late collision error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_OVERFLOWERR\_MASK 0x0200U  
*Overflow error mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_TIMESTAMPERR\_MASK 0x0100U  
*Timestamp error mask.*

### Second extended control region bit masks of the transmit buffer descriptor.

- #define ENET\_BUFFDESCRIPTOR\_TX\_INTERRUPT\_MASK 0x4000U  
*Interrupt mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_TIMESTAMP\_MASK 0x2000U  
*Timestamp flag mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_USETXLAUNCHTIME\_MASK 0x0100U  
*Use the transmit launch time.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_FRAMETYPE\_MASK 0x00F0U  
*Frame type mask.*
- #define ENET\_BUFFDESCRIPTOR\_TX\_FRAMETYPE\_SHIFT 4U  
*Frame type shift.*
- #define ENET\_BD\_FTYPE(n) ((n << ENET\_BUFFDESCRIPTOR\_TX\_FRAMETYPE\_SHIFT) & ENET\_BUFFDESCRIPTOR\_TX\_FRAMETYPE\_MASK)

### Defines some Ethernet parameters.

- #define ENET\_FRAME\_MAX\_FRAMELEN 1518U  
*Default maximum Ethernet frame size.*
- #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\_SHIFT)
- #define ENET\_TX\_INTERRUPT
- #define ENET\_RX\_INTERRUPT
- #define ENET\_TS\_INTERRUPT (kENET\_TsTimerInterrupt | kENET\_TsAvailInterrupt)
- #define ENET\_ERR\_INTERRUPT
- #define ENET\_ERR\_INTERRUPT

### Initialization and De-initialization

- void ENET\_GetDefaultConfig (enet\_config\_t \*config)  
*Gets the ENET default configuration structure.*

- void `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_Deinit` (ENET\_Type \*base)  
*Deinitializes the ENET module.*
- static void `ENET_Reset` (ENET\_Type \*base)  
*Resets the ENET module.*

## MII interface operation

- void `ENET_SetMII` (ENET\_Type \*base, `enet_mii_speed_t` speed, `enet_mii_duplex_t` duplex)  
*Sets the ENET MII speed and duplex.*
- void `ENET_SetSMI` (ENET\_Type \*base, uint32\_t srcClock\_Hz, bool isPreambleDisabled)  
*Sets the ENET SMI(serial management interface)- MII management interface.*
- static bool `ENET_GetSMI` (ENET\_Type \*base)  
*Gets the ENET SMI- MII management interface configuration.*
- static uint32\_t `ENET_ReadSMIData` (ENET\_Type \*base)  
*Reads data from the PHY register through an SMI interface.*
- void `ENET_StartSMIRead` (ENET\_Type \*base, uint32\_t phyAddr, uint32\_t phyReg, `enet_mii_read_t` operation)  
*Starts an SMI (Serial Management Interface) read command.*
- void `ENET_StartSMIWrite` (ENET\_Type \*base, uint32\_t phyAddr, uint32\_t phyReg, `enet_mii_write_t` operation, uint32\_t data)  
*Starts an SMI write command.*
- void `ENET_StartExtC45SMIRead` (ENET\_Type \*base, uint32\_t phyAddr, uint32\_t phyReg)  
*Starts the extended IEEE802.3 Clause 45 MDIO format SMI read command.*
- void `ENET_StartExtC45SMIWrite` (ENET\_Type \*base, uint32\_t phyAddr, uint32\_t phyReg, uint32\_t data)  
*Starts the extended IEEE802.3 Clause 45 MDIO format SMI write command.*
- static void `ENET_SetRGMIIClockDelay` (ENET\_Type \*base, bool txEnabled, bool rxEnabled)  
*Control the usage of the delayed tx/rx RGMI clock.*

## 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

- void `ENET_AVBConfigure` (ENET\_Type \*base, enet\_handle\_t \*handle, const `enet_avb_config_t` \*config)  
*Sets the ENET AVB feature.*
- static void `ENET_ActiveRead` (ENET\_Type \*base)  
*Activates ENET read or receive.*

## Typical use case

- static void [ENET\\_ActiveReadMultiRing](#) (ENET\_Type \*base)  
*Activates ENET read or receive for multiple-queue/ring.*
- 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.*

## Transactional operation

- void [ENET\\_SetCallback](#) (enet\_handle\_t \*handle, [enet\\_callback\\_t](#) callback, void \*userData)  
*Sets the callback function.*
- void [ENET\\_GetRxErrBeforeReadFrame](#) (enet\_handle\_t \*handle, [enet\\_data\\_error\\_stats\\_t](#) \*eErrorStatic)  
*Gets the error statistics of a received frame for ENET single ring.*
- status\_t [ENET\\_GetTxErrAfterSendFrame](#) (enet\_handle\_t \*handle, [enet\\_data\\_error\\_stats\\_t](#) \*eErrorStatic)  
*Gets the ENET transmit frame statistics after the data send for single ring.*
- status\_t [ENET\\_GetRxFrameSize](#) (enet\_handle\_t \*handle, uint32\_t \*length)  
*Gets the size of the read frame for single ring.*
- status\_t [ENET\\_ReadFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t \*data, uint32\_t length)  
*Reads a frame from the ENET device for single ring.*
- status\_t [ENET\\_SendFrame](#) (ENET\_Type \*base, enet\_handle\_t \*handle, const uint8\_t \*data, uint32\_t length)  
*Transmits an ENET frame for single ring.*
- void [ENET\\_GetRxErrBeforeReadFrameMultiRing](#) (enet\_handle\_t \*handle, [enet\\_data\\_error\\_stats\\_t](#) \*eErrorStatic, uint32\_t ringId)  
*Gets the error statistics of received frame for extended multi-ring.*
- status\_t [ENET\\_SendFrameMultiRing](#) (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t \*data, uint32\_t length, uint32\_t ringId)  
*Transmits an ENET frame for extended multi-ring.*
- status\_t [ENET\\_GetTxErrAfterSendFrameMultiRing](#) (enet\_handle\_t \*handle, [enet\\_data\\_error\\_stats\\_t](#) \*eErrorStatic, uint32\_t ringId)  
*Gets the ENET transmit frame statistics after the data send for extended multi-ring.*
- status\_t [ENET\\_GetRxFrameSizeMultiRing](#) (enet\_handle\_t \*handle, uint32\_t \*length, uint32\_t ringId)  
*Gets the size of the read frame for extended mutli-ring.*

- status\_t **ENET\_ReadFrameMultiRing** (ENET\_Type \*base, enet\_handle\_t \*handle, uint8\_t \*data, uint32\_t length, uint32\_t ringId)
 

*Reads a frame from the ENET device for multi-ring.*
- void **ENET\_TransmitIRQHandler** (ENET\_Type \*base, enet\_handle\_t \*handle, uint32\_t ringId)
 

*The transmit IRQ handler.*
- void **ENET\_ReceiveIRQHandler** (ENET\_Type \*base, enet\_handle\_t \*handle, uint32\_t ringId)
 

*The receive IRQ handler.*
- void **ENET\_CommonFrame1IRQHandler** (ENET\_Type \*base)
 

*the common IRQ handler for the tx/rx irq handler.*
- void **ENET\_CommonFrame2IRQHandler** (ENET\_Type \*base)
 

*the common IRQ handler for the tx/rx 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\_CommonFrame0IRQHandler** (ENET\_Type \*base)
 

*the common IRQ handler for the tx/rx/error etc irq handler.*

## ENET PTP 1588 function operation

- void **ENET\_Ptp1588Configure** (ENET\_Type \*base, enet\_handle\_t \*handle, enet\_ptp\_config\_t \*ptpConfig)
 

*Configures the ENET PTP IEEE 1588 feature with the basic configuration.*
- void **ENET\_Ptp1588StartTimer** (ENET\_Type \*base, uint32\_t ptptimeSrc)
 

*Starts the ENET PTP 1588 Timer.*
- static void **ENET\_Ptp1588StopTimer** (ENET\_Type \*base)
 

*Stops the ENET PTP 1588 Timer.*
- void **ENET\_Ptp1588AdjustTimer** (ENET\_Type \*base, uint32\_t corrIncrease, uint32\_t corrPeriod)
 

*Adjusts the ENET PTP 1588 timer.*
- static void **ENET\_Ptp1588SetChannelMode** (ENET\_Type \*base, enet\_ptp\_timer\_channel\_t channel, enet\_ptp\_timer\_channel\_mode\_t mode, bool intEnable)
 

*Sets the ENET PTP 1588 timer channel mode.*
- static void **ENET\_Ptp1588SetChannelCmpValue** (ENET\_Type \*base, enet\_ptp\_timer\_channel\_t channel, uint32\_t cmpValue)
 

*Sets the ENET PTP 1588 timer channel comparison value.*
- static bool **ENET\_Ptp1588GetChannelStatus** (ENET\_Type \*base, enet\_ptp\_timer\_channel\_t channel)
 

*Gets the ENET PTP 1588 timer channel status.*
- static void **ENET\_Ptp1588ClearChannelStatus** (ENET\_Type \*base, enet\_ptp\_timer\_channel\_t channel)
 

*Clears the ENET PTP 1588 timer channel status.*
- void **ENET\_Ptp1588GetTimer** (ENET\_Type \*base, enet\_handle\_t \*handle, enet\_ptp\_time\_t \*ptpTime)
 

*Gets the current ENET time from the PTP 1588 timer.*
- void **ENET\_Ptp1588SetTimer** (ENET\_Type \*base, enet\_handle\_t \*handle, enet\_ptp\_time\_t \*ptpTime)
 

*Sets the ENET PTP 1588 timer to the assigned time.*
- void **ENET\_Ptp1588TimerIRQHandler** (ENET\_Type \*base, enet\_handle\_t \*handle)
 

*The IEEE 1588 PTP time stamp interrupt handler.*
- status\_t **ENET\_GetRxFrameTime** (enet\_handle\_t \*handle, enet\_ptp\_time\_data\_t \*ptpTimeData)
 

*Gets the time stamp of the received frame.*
- status\_t **ENET\_GefTxFrameTime** (enet\_handle\_t \*handle, enet\_ptp\_time\_data\_t \*ptpTimeData)

## Data Structure Documentation

*Gets the time stamp of the transmit frame.*

### 11.3 Data Structure Documentation

#### 11.3.1 struct enet\_rx\_bd\_struct\_t

##### Data Fields

- `uint16_t length`  
*Buffer descriptor data length.*
- `uint16_t control`  
*Buffer descriptor control and status.*
- `uint8_t * buffer`  
*Data buffer pointer.*
- `uint16_t controlExtend0`  
*Extend buffer descriptor control0.*
- `uint16_t controlExtend1`  
*Extend buffer descriptor control1.*
- `uint16_t payloadCheckSum`  
*Internal payload checksum.*
- `uint8_t headerLength`  
*Header length.*
- `uint8_t protocolType`  
*Protocol type.*
- `uint16_t controlExtend2`  
*Extend buffer descriptor control2.*
- `uint32_t timestamp`  
*Timestamp.*

### 11.3.1.0.0.21 Field Documentation

- 11.3.1.0.0.21.1 `uint16_t enet_rx_bd_struct_t::length`
- 11.3.1.0.0.21.2 `uint16_t enet_rx_bd_struct_t::control`
- 11.3.1.0.0.21.3 `uint8_t* enet_rx_bd_struct_t::buffer`
- 11.3.1.0.0.21.4 `uint16_t enet_rx_bd_struct_t::controlExtend0`
- 11.3.1.0.0.21.5 `uint16_t enet_rx_bd_struct_t::controlExtend1`
- 11.3.1.0.0.21.6 `uint16_t enet_rx_bd_struct_t::payloadCheckSum`
- 11.3.1.0.0.21.7 `uint8_t enet_rx_bd_struct_t::headerLength`
- 11.3.1.0.0.21.8 `uint8_t enet_rx_bd_struct_t::protocolType`
- 11.3.1.0.0.21.9 `uint16_t enet_rx_bd_struct_t::controlExtend2`
- 11.3.1.0.0.21.10 `uint32_t enet_rx_bd_struct_t::timestamp`

## 11.3.2 struct enet\_tx\_bd\_struct\_t

### Data Fields

- `uint16_t length`  
*Buffer descriptor data length.*
- `uint16_t control`  
*Buffer descriptor control and status.*
- `uint8_t * buffer`  
*Data buffer pointer.*
- `uint16_t controlExtend0`  
*Extend buffer descriptor control0.*
- `uint16_t controlExtend1`  
*Extend buffer descriptor control1.*
- `int8_t * txLaunchTime`  
*Transmit launch time.*
- `uint16_t controlExtend2`  
*Extend buffer descriptor control2.*
- `uint32_t timestamp`  
*Timestamp.*

## Data Structure Documentation

### 11.3.2.0.0.22 Field Documentation

- 11.3.2.0.0.22.1 `uint16_t enet_tx_bd_struct_t::length`
- 11.3.2.0.0.22.2 `uint16_t enet_tx_bd_struct_t::control`
- 11.3.2.0.0.22.3 `uint8_t* enet_tx_bd_struct_t::buffer`
- 11.3.2.0.0.22.4 `uint16_t enet_tx_bd_struct_t::controlExtend0`
- 11.3.2.0.0.22.5 `uint16_t enet_tx_bd_struct_t::controlExtend1`
- 11.3.2.0.0.22.6 `int8_t* enet_tx_bd_struct_t::txLaunchTime`
- 11.3.2.0.0.22.7 `uint16_t enet_tx_bd_struct_t::controlExtend2`
- 11.3.2.0.0.22.8 `uint32_t enet_tx_bd_struct_t::timestamp`

### 11.3.3 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.*
- `uint32_t statsRxProtocolChecksumErr`  
*Receive protocol checksum error.*
- `uint32_t statsRxIpHeadChecksumErr`  
*Receive IP header checksum error.*
- `uint32_t statsRxMacErr`  
*Receive Mac error.*
- `uint32_t statsRxPhyErr`  
*Receive PHY error.*
- `uint32_t statsRxCollisionErr`  
*Receive collision.*
- `uint32_t statsTxErr`  
*The error happen when transmit the frame.*
- `uint32_t statsTxFameErr`  
*The transmit frame is error.*
- `uint32_t statsTxOverFlowErr`  
*Transmit overflow.*
- `uint32_t statsTxLateCollisionErr`  
*Transmit late collision.*

- `uint32_t statsTxExcessCollisionErr`  
*Transmit excess collision.*
- `uint32_t statsTxUnderFlowErr`  
*Transmit under flow error.*
- `uint32_t statsTxTsErr`  
*Transmit time stamp error.*

### 11.3.3.0.0.23 Field Documentation

- 11.3.3.0.0.23.1 `uint32_t enet_data_error_stats_t::statsRxLenGreaterErr`
- 11.3.3.0.0.23.2 `uint32_t enet_data_error_stats_t::statsRxFcsErr`
- 11.3.3.0.0.23.3 `uint32_t enet_data_error_stats_t::statsRxOverRunErr`
- 11.3.3.0.0.23.4 `uint32_t enet_data_error_stats_t::statsRxTruncateErr`
- 11.3.3.0.0.23.5 `uint32_t enet_data_error_stats_t::statsRxProtocolChecksumErr`
- 11.3.3.0.0.23.6 `uint32_t enet_data_error_stats_t::statsRxIpHeadChecksumErr`
- 11.3.3.0.0.23.7 `uint32_t enet_data_error_stats_t::statsRxMacErr`
- 11.3.3.0.0.23.8 `uint32_t enet_data_error_stats_t::statsRxPhyErr`
- 11.3.3.0.0.23.9 `uint32_t enet_data_error_stats_t::statsRxCollisionErr`
- 11.3.3.0.0.23.10 `uint32_t enet_data_error_stats_t::statsTxErr`
- 11.3.3.0.0.23.11 `uint32_t enet_data_error_stats_t::statsTxFrameErr`
- 11.3.3.0.0.23.12 `uint32_t enet_data_error_stats_t::statsTxOverFlowErr`
- 11.3.3.0.0.23.13 `uint32_t enet_data_error_stats_t::statsTxLateCollisionErr`
- 11.3.3.0.0.23.14 `uint32_t enet_data_error_stats_t::statsTxExcessCollisionErr`
- 11.3.3.0.0.23.15 `uint32_t enet_data_error_stats_t::statsTxUnderFlowErr`
- 11.3.3.0.0.23.16 `uint32_t enet_data_error_stats_t::statsTxTsErr`

### 11.3.4 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

## Data Structure Documentation

- 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\_BUF\_F\_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.*
- `uint32_t rxBuffSizeAlign`  
*Aligned receive data buffer size.*
- `uint32_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.*

**11.3.4.0.0.24 Field Documentation**

- 11.3.4.0.0.24.1 uint16\_t enet\_buffer\_config\_t::rxBdNumber**
- 11.3.4.0.0.24.2 uint16\_t enet\_buffer\_config\_t::txBdNumber**
- 11.3.4.0.0.24.3 uint32\_t enet\_buffer\_config\_t::rxBuffSizeAlign**
- 11.3.4.0.0.24.4 uint32\_t enet\_buffer\_config\_t::txBuffSizeAlign**
- 11.3.4.0.0.24.5 volatile enet\_rx\_bd\_struct\_t\* enet\_buffer\_config\_t::rxBdStartAddrAlign**
- 11.3.4.0.0.24.6 volatile enet\_tx\_bd\_struct\_t\* enet\_buffer\_config\_t::txBdStartAddrAlign**
- 11.3.4.0.0.24.7 uint8\_t\* enet\_buffer\_config\_t::rxBufferAlign**
- 11.3.4.0.0.24.8 uint8\_t\* enet\_buffer\_config\_t::txBufferAlign**

**11.3.5 struct enet\_ptp\_time\_t****Data Fields**

- **uint64\_t second**  
*Second.*
- **uint32\_t nanosecond**  
*Nanosecond.*

**11.3.5.0.0.25 Field Documentation**

- 11.3.5.0.0.25.1 uint64\_t enet\_ptp\_time\_t::second**
- 11.3.5.0.0.25.2 uint32\_t enet\_ptp\_time\_t::nanosecond**

**11.3.6 struct enet\_ptp\_time\_data\_t****Data Fields**

- **uint8\_t version**  
*PTP version.*
- **uint8\_t sourcePortId [kENET\_PtpSrcPortIdLen]**  
*PTP source port ID.*
- **uint16\_t sequenceId**  
*PTP sequence ID.*
- **uint8\_t messageType**  
*PTP message type.*
- **enet\_ptp\_time\_t timeStamp**  
*PTP timestamp.*

## Data Structure Documentation

### 11.3.6.0.0.26 Field Documentation

11.3.6.0.0.26.1 `uint8_t enet_ptp_time_data_t::version`

11.3.6.0.0.26.2 `uint8_t enet_ptp_time_data_t::sourcePortId[kENET_PtpSrcPortIdLen]`

11.3.6.0.0.26.3 `uint16_t enet_ptp_time_data_t::sequenceId`

11.3.6.0.0.26.4 `uint8_t enet_ptp_time_data_t::messageType`

11.3.6.0.0.26.5 `enet_ptp_time_t enet_ptp_time_data_t::timeStamp`

### 11.3.7 `struct enet_ptp_time_data_ring_t`

#### Data Fields

- `uint32_t front`  
*The first index of the ring.*
- `uint32_t end`  
*The end index of the ring.*
- `uint32_t size`  
*The size of the ring.*
- `enet_ptp_time_data_t * ptpTsData`  
*PTP message data structure.*

### 11.3.7.0.0.27 Field Documentation

11.3.7.0.0.27.1 `uint32_t enet_ptp_time_data_ring_t::front`

11.3.7.0.0.27.2 `uint32_t enet_ptp_time_data_ring_t::end`

11.3.7.0.0.27.3 `uint32_t enet_ptp_time_data_ring_t::size`

11.3.7.0.0.27.4 `enet_ptp_time_data_t* enet_ptp_time_data_ring_t::ptpTsData`

### 11.3.8 `struct enet_ptp_config_t`

#### Data Fields

- `uint8_t ptpRxBuffNum`  
*Receive 1588 timestamp buffer number.*
- `uint8_t ptpTxBuffNum`  
*Transmit 1588 timestamp buffer number.*
- `enet_ptp_time_data_t * rxPtpTsData`  
*The start address of 1588 receive timestamp buffers.*
- `enet_ptp_time_data_t * txPtpTsData`  
*The start address of 1588 transmit timestamp buffers.*
- `enet_ptp_timer_channel_t channel`  
*Used for ERRATA\_2579: the PTP 1588 timer channel for time interrupt.*

- `uint32_t ptp1588ClockSrc_Hz`  
*The clock source of the PTP 1588 timer.*

#### 11.3.8.0.0.28 Field Documentation

11.3.8.0.0.28.1 `enet_ptp_timer_channel_t enet_ptp_config_t::channel`

11.3.8.0.0.28.2 `uint32_t enet_ptp_config_t::ptp1588ClockSrc_Hz`

#### 11.3.9 `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.*

#### 11.3.9.0.0.29 Field Documentation

11.3.9.0.0.29.1 `uint8_t enet_intcoalesce_config_t::txCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`

11.3.9.0.0.29.2 `uint16_t enet_intcoalesce_config_t::txCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`

11.3.9.0.0.29.3 `uint8_t enet_intcoalesce_config_t::rxCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`

11.3.9.0.0.29.4 `uint16_t enet_intcoalesce_config_t::rxCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`

#### 11.3.10 `struct enet_avb_config_t`

This is used for to configure the extended ring 1 and ring 2.

1. The classification match format is  $(\text{CMP3} \ll 12) | (\text{CMP2} \ll 8) | (\text{CMP1} \ll 4) | \text{CMP0}$ . composed of four 3-bit compared VLAN priority field cmp0~cmp3, cm0 ~ cmp3 are used in parallel.

If CMP1,2,3 are not unused, please set them to the same value as CMP0.

1. The idleSlope is used to calculate the Band Width fraction,  $\text{BW fraction} = 1 / (1 + 512/\text{idleSlope})$ . For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

## Data Structure Documentation

### Data Fields

- `uint16_t rxClassifyMatch` [FSL\_FEATURE\_ENET\_QUEUE-1]  
*The classification match value for the ring.*
- `enet_idle_slope_t idleSlope` [FSL\_FEATURE\_ENET\_QUEUE-1]  
*The idle slope for certain bandwidth fraction.*

#### 11.3.10.0.0.30 Field Documentation

11.3.10.0.0.30.1 `uint16_t enet_avb_config_t::rxClassifyMatch[FSL_FEATURE_ENET_QUEUE-1]`

11.3.10.0.0.30.2 `enet_idle_slope_t enet_avb_config_t::idleSlope[FSL_FEATURE_ENET_QUEUE-1]`

### 11.3.11 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 is 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**  
*MII Speed.*
- **enet\_mii\_duplex\_t miiDuplex**  
*MII duplex.*
- **uint8\_t rxAccelerConfig**  
*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 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.*
- **uint8\_t ringNum**  
*Number of used rings.*

### 11.3.11.0.0.31 Field Documentation

#### 11.3.11.0.0.31.1 uint32\_t enet\_config\_t::macSpecialConfig

A logical OR of "enet\_special\_control\_flag\_t".

#### 11.3.11.0.0.31.2 uint32\_t enet\_config\_t::interrupt

A logical OR of "enet\_interrupt\_enable\_t".

## Data Structure Documentation

- 11.3.11.0.0.31.3 `uint16_t enet_config_t::rxMaxFrameLen`
- 11.3.11.0.0.31.4 `enet_mii_mode_t enet_config_t::miiMode`
- 11.3.11.0.0.31.5 `enet_mii_speed_t enet_config_t::miiSpeed`
- 11.3.11.0.0.31.6 `enet_mii_duplex_t enet_config_t::miiDuplex`
- 11.3.11.0.0.31.7 `uint8_t enet_config_t::rxAccelerConfig`
- 11.3.11.0.0.31.8 `uint8_t enet_config_t::txAccelerConfig`
- 11.3.11.0.0.31.9 `uint16_t enet_config_t::pauseDuration`
- 11.3.11.0.0.31.10 `uint8_t enet_config_t::rxFifoEmptyThreshold`
- 11.3.11.0.0.31.11 `uint8_t enet_config_t::rxFifoFullThreshold`
- 11.3.11.0.0.31.12 `uint8_t enet_config_t::txFifoWatermark`
- 11.3.11.0.0.31.13 `uint8_t enet_config_t::ringNum`

default with 1 – single ring.

## 11.3.12 struct \_enet\_handle

### Data Fields

- volatile `enet_rx_bd_struct_t * rxBdBase` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer descriptor base address pointer.*
- volatile `enet_rx_bd_struct_t * rxBdCurrent` [FSL\_FEATURE\_ENET\_QUEUE]  
*The current available receive buffer descriptor pointer.*
- volatile `enet_tx_bd_struct_t * txBdBase` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer descriptor base address pointer.*
- volatile `enet_tx_bd_struct_t * txBdCurrent` [FSL\_FEATURE\_ENET\_QUEUE]  
*The current available transmit buffer descriptor pointer.*
- `uint32_t rxBuffSizeAlign` [FSL\_FEATURE\_ENET\_QUEUE]  
*Receive buffer size alignment.*
- `uint32_t txBuffSizeAlign` [FSL\_FEATURE\_ENET\_QUEUE]  
*Transmit buffer size alignment.*
- `uint8_t ringNum`  
*Number of used rings.*
- `enet_callback_t callback`  
*Callback function.*
- `void * userData`  
*Callback function parameter.*
- volatile `enet_tx_bd_struct_t * txBdDirtyStatic` [FSL\_FEATURE\_ENET\_QUEUE]  
*The dirty transmit buffer descriptor for error static update.*
- volatile `enet_tx_bd_struct_t * txBdDirtyTime` [FSL\_FEATURE\_ENET\_QUEUE]

- `uint64_t msTimerSecond`  
*The dirty transmit buffer descriptor for time stamp update.*
- `enet_ptp_time_data_ring_t rxPtpTsDataRing`  
*The second for Master PTP timer .*
- `enet_ptp_time_data_ring_t txPtpTsDataRing`  
*Receive PTP 1588 time stamp data ring buffer.*
- `enet_ptp_time_data_ring_t txPtpTsDataRing`  
*Transmit PTP 1588 time stamp data ring buffer.*

### 11.3.12.0.0.32 Field Documentation

- 11.3.12.0.0.32.1 `volatile enet_rx_bd_struct_t* enet_handle_t::rxBdBase[FSL_FEATURE_ENET_QUEUE]`
- 11.3.12.0.0.32.2 `volatile enet_rx_bd_struct_t* enet_handle_t::rxBdCurrent[FSL_FEATURE_ENET_QUEUE]`
- 11.3.12.0.0.32.3 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdBase[FSL_FEATURE_ENET_QUEUE]`
- 11.3.12.0.0.32.4 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdCurrent[FSL_FEATURE_ENET_QUEUE]`
- 11.3.12.0.0.32.5 `uint32_t enet_handle_t::rxBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]`
- 11.3.12.0.0.32.6 `uint32_t enet_handle_t::txBuffSizeAlign[FSL_FEATURE_ENET_QUEUE]`
- 11.3.12.0.0.32.7 `uint8_t enet_handle_t::ringNum`
- 11.3.12.0.0.32.8 `enet_callback_t enet_handle_t::callback`
- 11.3.12.0.0.32.9 `void* enet_handle_t::userData`
- 11.3.12.0.0.32.10 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdDirtyStatic[FSL_FEATURE_ENET_QUEUE]`
- 11.3.12.0.0.32.11 `volatile enet_tx_bd_struct_t* enet_handle_t::txBdDirtyTime[FSL_FEATURE_ENET_QUEUE]`
- 11.3.12.0.0.32.12 `uint64_t enet_handle_t::msTimerSecond`
- 11.3.12.0.0.32.13 `enet_ptp_time_data_ring_t enet_handle_t::rxPtpTsDataRing`
- 11.3.12.0.0.32.14 `enet_ptp_time_data_ring_t enet_handle_t::txPtpTsDataRing`

## 11.4 Macro Definition Documentation

- 11.4.1 `#define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))`

Version 2.2.3.



- 11.4.2 `#define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U`
- 11.4.3 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U`
- 11.4.4 `#define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U`
- 11.4.5 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U`
- 11.4.6 `#define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U`
- 11.4.7 `#define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U`
- 11.4.8 `#define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U`
- 11.4.9 `#define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U`
- 11.4.10 `#define ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK 0x0020U`
- 11.4.11 `#define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U`
- 11.4.12 `#define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U`
- 11.4.13 `#define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U`
- 11.4.14 `#define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U`
- 11.4.15 `#define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U`
- 11.4.16 `#define ENET_BUFFDESCRIPTOR_TX_SOFTOWNER1_MASK 0x4000U`
- 11.4.17 `#define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U`
- 11.4.18 `#define ENET_BUFFDESCRIPTOR_TX_SOFTOWNER2_MASK 0x1000U`
- 11.4.19 `#define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U`
- 11.4.20 `#define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U`
- 11.4.21 `#define ENET_BUFFDESCRIPTOR_RX_IPV4_MASK 0x0001U`
- 11.4.22 `#define ENET_BUFFDESCRIPTOR_RX_IPV6_MASK 0x0002U`

## Enumeration Type Documentation

```
(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)
```

**11.4.44 #define ENET\_FRAME\_MAX\_FRAMELEN 1518U**

**11.4.45 #define ENET\_FIFO\_MIN\_RX\_FULL 5U**

**11.4.46 #define ENET\_RX\_MIN\_BUFFERSIZE 256U**

## 11.5 Typedef Documentation

**11.5.1 typedef void(\* enet\_callback\_t)(ENET\_Type \*base, enet\_handle\_t \*handle, uint32\_t ringId, enet\_event\_t event, void \*userData)**

## 11.6 Enumeration Type Documentation

### 11.6.1 enum \_enet\_status

Enumerator

*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\_TxFrameOverLen* Tx frame over length.

*kStatus\_ENET\_TxFrameBusy* Tx buffer descriptors are under process.

*kStatus\_ENET\_TxFrameFail* Transmit frame fail.

*kStatus\_ENET\_PtpTsRingFull* Timestamp ring full.

*kStatus\_ENET\_PtpTsRingEmpty* Timestamp ring empty.

### 11.6.2 enum enet\_mii\_mode\_t

Enumerator

*kENET\_MiiMode* MII mode for data interface.

*kENET\_RmiiMode* RMII mode for data interface.

*kENET\_RgmiiMode* RGMII mode for data interface.

### 11.6.3 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.

*kENET\_MiiSpeed1000M* Speed 1000M bps.

### 11.6.4 enum enet\_mii\_duplex\_t

Enumerator

*kENET\_MiiHalfDuplex* Half duplex mode.

*kENET\_MiiFullDuplex* Full duplex mode.

### 11.6.5 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.

### 11.6.6 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.

### 11.6.7 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.

## Enumeration Type Documentation

### 11.6.8 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).
- kENET\_ControlSVLANEnable* Enable S-VLAN.
- kENET\_ControlVLANUseSecondTag* Enable extracting the second vlan tag for further processing.

### 11.6.9 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.

|                                 |                                          |
|---------------------------------|------------------------------------------|
| <i>kENET_RxFlush2Interrupt</i>  | Rx DMA ring2 flush indication.           |
| <i>kENET_RxFlush1Interrupt</i>  | Rx DMA ring1 flush indication.           |
| <i>kENET_RxFlush0Interrupt</i>  | RX DMA ring0 flush indication.           |
| <i>kENET_TxFrame2Interrupt</i>  | Tx frame interrupt for Tx ring/class 2.  |
| <i>kENET_TxBuffer2Interrupt</i> | Tx buffer interrupt for Tx ring/class 2. |
| <i>kENET_RxFrame2Interrupt</i>  | Rx frame interrupt for Rx ring/class 2.  |
| <i>kENET_RxBuffer2Interrupt</i> | Rx buffer interrupt for Rx ring/class 2. |
| <i>kENET_TxFrame1Interrupt</i>  | Tx frame interrupt for Tx ring/class 1.  |
| <i>kENET_TxBuffer1Interrupt</i> | Tx buffer interrupt for Tx ring/class 1. |
| <i>kENET_RxFrame1Interrupt</i>  | Rx frame interrupt for Rx ring/class 1.  |
| <i>kENET_RxBuffer1Interrupt</i> | Rx buffer interrupt for Rx ring/class 1. |
| <i>kENET_TsAvailInterrupt</i>   | TS AVAIL interrupt source for PTP.       |
| <i>kENET_TsTimerInterrupt</i>   | TS WRAP interrupt source for PTP.        |

### 11.6.10 enum enet\_event\_t

Enumerator

|                                  |                                             |
|----------------------------------|---------------------------------------------|
| <i>kENET_RxEvent</i>             | Receive event.                              |
| <i>kENET_TxEvent</i>             | Transmit event.                             |
| <i>kENET_ErrEvent</i>            | Error event: BABR/BABT/EBERR/LC/RL/UN/PLR . |
| <i>kENET_WakeUpEvent</i>         | Wake up from sleep mode event.              |
| <i>kENET_TimeStampEvent</i>      | Time stamp event.                           |
| <i>kENET_TimeStampAvailEvent</i> | Time stamp available event.                 |

### 11.6.11 enum enet\_idle\_slope\_t

Enumerator

|                           |                                        |
|---------------------------|----------------------------------------|
| <i>kENET_IdleSlope1</i>   | The bandwidth fraction is about 0.002. |
| <i>kENET_IdleSlope2</i>   | The bandwidth fraction is about 0.003. |
| <i>kENET_IdleSlope4</i>   | The bandwidth fraction is about 0.008. |
| <i>kENET_IdleSlope8</i>   | The bandwidth fraction is about 0.02.  |
| <i>kENET_IdleSlope16</i>  | The bandwidth fraction is about 0.03.  |
| <i>kENET_IdleSlope32</i>  | The bandwidth fraction is about 0.06.  |
| <i>kENET_IdleSlope64</i>  | The bandwidth fraction is about 0.11.  |
| <i>kENET_IdleSlope128</i> | The bandwidth fraction is about 0.20.  |
| <i>kENET_IdleSlope256</i> | The bandwidth fraction is about 0.33.  |
| <i>kENET_IdleSlope384</i> | The bandwidth fraction is about 0.43.  |
| <i>kENET_IdleSlope512</i> | The bandwidth fraction is about 0.50.  |
| <i>kENET_IdleSlope640</i> | The bandwidth fraction is about 0.56.  |
| <i>kENET_IdleSlope768</i> | The bandwidth fraction is about 0.60.  |
| <i>kENET_IdleSlope896</i> | The bandwidth fraction is about 0.64.  |

## Enumeration Type Documentation

- kENET\_IdleSlope1024*** The bandwidth fraction is about 0.67.
- kENET\_IdleSlope1152*** The bandwidth fraction is about 0.69.
- kENET\_IdleSlope1280*** The bandwidth fraction is about 0.71.
- kENET\_IdleSlope1408*** The bandwidth fraction is about 0.73.
- kENET\_IdleSlope1536*** The bandwidth fraction is about 0.75.

### 11.6.12 enum enet\_tx\_accelerator\_t

Enumerator

- kENET\_TxAccelIsShift16Enabled*** Transmit FIFO shift-16.
- kENET\_TxAccelIpCheckEnabled*** Insert IP header checksum.
- kENET\_TxAccelProtoCheckEnabled*** Insert protocol checksum.

### 11.6.13 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.

### 11.6.14 enum enet\_ptp\_event\_type\_t

Enumerator

- kENET\_PtpEventMsgType*** PTP event message type.
- kENET\_PtpSrcPortIdLen*** PTP message sequence id length.
- kENET\_PtpEventPort*** PTP event port number.
- kENET\_PtpGnrlPort*** PTP general port number.

### 11.6.15 enum enet\_ptp\_timer\_channel\_t

Enumerator

- kENET\_PtpTimerChannel1*** IEEE 1588 PTP timer Channel 1.
- kENET\_PtpTimerChannel2*** IEEE 1588 PTP timer Channel 2.
- kENET\_PtpTimerChannel3*** IEEE 1588 PTP timer Channel 3.
- kENET\_PtpTimerChannel4*** IEEE 1588 PTP timer Channel 4.

### 11.6.16 enum enet\_ptp\_timer\_channel\_mode\_t

Enumerator

*kENET\_PtpChannelDisable* Disable timer channel.  
*kENET\_PtpChannelRisingCapture* Input capture on rising edge.  
*kENET\_PtpChannelFallingCapture* Input capture on falling edge.  
*kENET\_PtpChannelBothCapture* Input capture on both edges.  
*kENET\_PtpChannelSoftCompare* Output compare software only.  
*kENET\_PtpChannelToggleCompare* Toggle output on compare.  
*kENET\_PtpChannelClearCompare* Clear output on compare.  
*kENET\_PtpChannelSetCompare* Set output on compare.  
*kENET\_PtpChannelClearCompareSetOverflow* Clear output on compare, set output on overflow.  
*kENET\_PtpChannelSetCompareClearOverflow* Set output on compare, clear output on overflow.  
*kENET\_PtpChannelPulseLowonCompare* Pulse output low on compare for one IEEE 1588 clock cycle.  
*kENET\_PtpChannelPulseHighonCompare* Pulse output high on compare for one IEEE 1588 clock cycle.

## 11.7 Function Documentation

### 11.7.1 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. |
|---------------|----------------------------------------------------------|

### 11.7.2 void 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.

## Function Documentation

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.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

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\(\)](#).

### 11.7.3 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. |
|-------------|-------------------------------|

### 11.7.4 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. |
|-------------|-------------------------------|

### 11.7.5 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.  |

### 11.7.6 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. Normally it's the system clock. See clock distribution.                                                  |
| <i>isPreamble-Disabled</i> | The preamble disable flag. <ul style="list-style-type: none"> <li>• true Enables the preamble.</li> <li>• false Disables the preamble.</li> </ul> |

### 11.7.7 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

## Function Documentation

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

Returns

The SMI setup status true or false.

### **11.7.8 static uint32\_t ENET\_ReadSMIData ( ENET\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The data read from PHY

### **11.7.9 void ENET\_StartSMIRead ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, enet\_mii\_read\_t *operation* )**

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

|                  |                                      |
|------------------|--------------------------------------|
| <i>base</i>      | ENET peripheral base address.        |
| <i>phyAddr</i>   | The PHY address.                     |
| <i>phyReg</i>    | The PHY register. Range from 0 ~ 31. |
| <i>operation</i> | The read operation.                  |

### **11.7.10 void ENET\_StartSMIWrite ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, enet\_mii\_write\_t *operation*, uint32\_t *data* )**

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

|                  |                                      |
|------------------|--------------------------------------|
| <i>base</i>      | ENET peripheral base address.        |
| <i>phyAddr</i>   | The PHY address.                     |
| <i>phyReg</i>    | The PHY register. Range from 0 ~ 31. |
| <i>operation</i> | The write operation.                 |
| <i>data</i>      | The data written to PHY.             |

### 11.7.11 void ENET\_StartExtC45SMIRead ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg* )

Parameters

|                |                                                                                                                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                                                                                                                          |
| <i>phyAddr</i> | The PHY address.                                                                                                                                                                                       |
| <i>phyReg</i>  | The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16)   regAddr. |

### 11.7.12 void ENET\_StartExtC45SMIWrite ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, uint32\_t *data* )

Parameters

|                |                                                                                                                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                                                                                                                          |
| <i>phyAddr</i> | The PHY address.                                                                                                                                                                                       |
| <i>phyReg</i>  | The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16)   regAddr. |
| <i>data</i>    | The data written to PHY.                                                                                                                                                                               |

### 11.7.13 static void ENET\_SetRGMIIClockDelay ( ENET\_Type \* *base*, bool *txEnabled*, bool *rxEnabled* ) [inline], [static]

## Function Documentation

Parameters

|                  |                                                                 |
|------------------|-----------------------------------------------------------------|
| <i>base</i>      | ENET peripheral base address.                                   |
| <i>txEnabled</i> | Enable or disable to generate the delayed version of RGMII_TXC. |
| <i>rxEnabled</i> | Enable or disable to use the delayed version of RGMII_RXC.      |

### 11.7.14 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. |

### 11.7.15 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. |

### 11.7.16 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. |

### 11.7.17 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. |

### 11.7.18 void ENET\_AVBConfigure ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, const enet\_avb\_config\_t \* *config* )

ENET AVB feature configuration, set the Receive classification match and transmit bandwidth. This API is called when the AVB feature is required.

Note: The AVB frames transmission scheme is credit-based tx scheme and it's only supported with the Enhanced buffer descriptors. so the AVB configuration should only done with Enhanced buffer descriptor. so when the AVB feature is required, please make sure the the "ENET\_ENHANCEDBUFFERDESCRIPTOR\_MODE" is defined.

Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                 |
| <i>handle</i> | ENET handler pointer.                         |
| <i>config</i> | The ENET AVB feature configuration structure. |

### 11.7.19 static void ENET\_ActiveRead ( ENET\_Type \* *base* ) [inline], [static]

This function is to active the enet read process. It is used for single descriptor ring/queue.

Parameters

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

Note

This must be called after the MAC configuration and state are ready. It must be called after the [ENET\\_Init\(\)](#) and [ENET\\_Ptp1588Configure\(\)](#). This should be called when the ENET receive required.

### 11.7.20 static void ENET\_ActiveReadMultiRing ( ENET\_Type \* *base* ) [inline], [static]

This function is to active the enet read process. It is used for extended multiple descriptor rings/queues.

## Function Documentation

Parameters

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

Note

This must be called after the MAC configuration and state are ready. It must be called after the [ENET\\_Init\(\)](#) and [ENET\\_Ptp1588Configure\(\)](#). This should be called when the ENET receive required.

### **11.7.21 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. |

### **11.7.22 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.  |

### **11.7.23 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);
*
```

## Function Documentation

### Parameters

|             |                                                                                                |
|-------------|------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                  |
| <i>mask</i> | ENET interrupts to enable. This is a logical OR of the enumeration :: enet_interrupt_enable_t. |

### 11.7.24 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 :: enet_interrupt_enable_t. |

### 11.7.25 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.

### 11.7.26 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 :: enet_interrupt_enable_t. |

### 11.7.27 void ENET\_SetCallback ( *enet\_handle\_t* \* *handle*, *enet\_callback\_t* *callback*, *void* \* *userData* )

This API is provided for the application callback required case when ENET interrupt is enabled. This API should be called after calling ENET\_Init.

## Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>handle</i>   | ENET handler pointer. Should be provided by application. |
| <i>callback</i> | The ENET callback function.                              |
| <i>userData</i> | The callback function parameter.                         |

### 11.7.28 void ENET\_GetRxErrBeforeReadFrame ( *enet\_handle\_t* \* *handle*, *enet\_data\_error\_stats\_t* \* *eErrorStatic* )

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);
* if (status == kStatus_ENET_RxFrameError)
* {
* // Get the error information of the received frame.
* ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic);
* // update the receive buffer.
* ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
* }
*
```

## Function Documentation

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.                                                     |

### 11.7.29 **status\_t ENET\_GetTxErrAfterSendFrame ( enet\_handle\_t \* *handle*, enet\_data\_error\_stats\_t \* *eErrorStatic* )**

This interface gets the error statistics of the transmit frame. Because the error information is reported by the uDMA after the data delivery, this interface should be called after the data transmit API. It is recommended to call this function on transmit interrupt handler. After calling the ENET\_SendFrame, the transmit interrupt notifies the transmit completion.

Parameters

|                     |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <i>handle</i>       | The PTP handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>eErrorStatic</i> | The error statistics structure pointer.                                          |

Returns

The execute status.

### 11.7.30 **status\_t ENET\_GetRxFrameSize ( enet\_handle\_t \* *handle*, uint32\_t \* *length* )**

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 update the receive buffers 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.                                             |

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. |

### 11.7.31 **status\_t ENET\_ReadFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint8\_t \* *data*, uint32\_t *length* )**

This function reads a frame (both the data and the length) from the ENET buffer descriptors. The ENET\_GetRxFrameSize should be used to get the size of the prepared data buffer. This is an example:

```

* uint32_t length;
* enet_handle_t g_handle;
* //Get the received frame size firstly.
* status = ENET_GetRxFrameSize(&g_handle, &length);
* if (length != 0)
* {
* //Allocate memory here with the size of "length"
* uint8_t *data = memory allocate interface;
* if (!data)
* {
* ENET_ReadFrame(ENET, &g_handle, NULL, 0);
* //Add the console warning log.
* }
* else
* {
* status = ENET_ReadFrame(ENET, &g_handle, data, length);
* //Call stack input API to deliver the data to stack
* }
* }
* else if (status == kStatus_ENET_RxFrameError)
* {
* //Update the received buffer when a error frame is received.
* ENET_ReadFrame(ENET, &g_handle, NULL, 0);
* }
*
```

Parameters

## Function Documentation

|               |                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------|
| <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.                       |

Returns

The execute status, successful or failure.

### 11.7.32 **status\_t ENET\_SendFrame ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, const uint8\_t \* *data*, uint32\_t *length* )**

Note

The CRC is automatically appended to the data. Input the data to send without the CRC.

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 be send.                                      |
| <i>length</i> | The length of the data to be send.                                                |

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> . |

### 11.7.33 **void ENET\_GetRxErrBeforeReadFrameMultiRing ( enet\_handle\_t \* *handle*, enet\_data\_error\_stats\_t \* *eErrorStatic*, uint32\_t *ringId* )**

This API must be called after the ENET\_GetRxFrameSizeMultiRing and before the [ENET\\_ReadFrameMultiRing\(\)](#). If the ENET\_GetRxFrameSizeMultiRing returns *kStatus\_ENET\_RxFrameError*, the ENET\_GetRxErrBeforeReadFrameMultiRing can be used to get the exact error statistics.

## 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_QUEUE - 1.                                  |

**11.7.34 status\_t ENET\_SendFrameMultiRing ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint8\_t \* *data*, uint32\_t *length*, uint32\_t *ringId* )**

## Note

The CRC is automatically appended to the data. Input the data to send without the CRC.

In this API, multiple-ring are mainly used for extended avb frames are supported. The transmit scheme for avb frames is the credit-based scheme, the AVB class A, AVB class B and the non-AVB frame are transmitted in ring 1, ring 2 and ring 0 independently. So application should care about the transmit ring index when use multiple-ring 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>data</i>   | The data buffer provided by user to be send.                                      |
| <i>length</i> | The length of the data to be send.                                                |
| <i>ringId</i> | The ring index for transmission.                                                  |

## 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> . |

**11.7.35 status\_t ENET\_GetTxErrAfterSendFrameMultiRing ( enet\_handle\_t \* *handle*, enet\_data\_error\_stats\_t \* *eErrorStatic*, uint32\_t *ringId* )**

This interface gets the error statistics of the transmit frame. Because the error information is reported by the uDMA after the data delivery, this interface should be called after the data transmit API and shall be

## Function Documentation

called by transmit interrupt handler. After calling the ENET\_SendFrame, the transmit interrupt notifies the transmit completion.

Parameters

|                     |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <i>handle</i>       | The PTP handler 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.                                                                  |

Returns

The execute status.

### 11.7.36 **status\_t ENET\_GetRxFrameSizeMultiRing ( enet\_handle\_t \* *handle*, uint32\_t \* *length*, uint32\_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\_GetRxFrameSizeMultiRing, [ENET\\_ReadFrameMultiRing\(\)](#) should be called to update the receive buffers If the result is not "kStatus\_ENET\_RxFrameEmpty". The usage is the same to the single ring, refer to ENET\_GetRxFrameSize.

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_ReadFrameMultiRing to read frame. |
|----------------------------------|---------------------------------------------------------------------------|

|                                   |                                                                                                                                      |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_ENET_RxFrame-Error</i> | Data error happens. ENET_ReadFrameMultiRing 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. |

### 11.7.37 **status\_t ENET\_ReadFrameMultiRing ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint8\_t \* *data*, uint32\_t *length*, uint32\_t *ringId* )**

This function reads a frame (both the data and the length) from the ENET buffer descriptors. The ENET\_GetRxFrameSizeMultiRing should be used to get the size of the prepared data buffer. This usage is the same as the single ring, refer to ENET\_ReadFrame.

#### 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;                                                                     |

#### Returns

The execute status, successful or failure.

### 11.7.38 **void ENET\_TransmitIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint32\_t *ringId* )**

#### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |
| <i>ringId</i> | The ring id or ring number.   |

### 11.7.39 **void ENET\_ReceiveIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, uint32\_t *ringId* )**

## Function Documentation

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |
| <i>ringId</i> | The ring id or ring number.   |

### 11.7.40 void ENET\_CommonFrame1IRQHandler ( ENET\_Type \* *base* )

This is used for the combined tx/rx interrupt for multi-ring (frame 1).

Parameters

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

### 11.7.41 void ENET\_CommonFrame2IRQHandler ( ENET\_Type \* *base* )

This is used for the combined tx/rx interrupt for multi-ring (frame 2).

Parameters

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

### 11.7.42 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.     |

### 11.7.43 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. |
|-------------|-------------------------------|

**11.7.44 void ENET\_Ptp1588Configure ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_ptp\_config\_t \* *ptpConfig* )**

The function sets the clock for PTP 1588 timer and enables time stamp interrupts and transmit interrupts for PTP 1588 features. This API should be called when the 1588 feature is enabled or the ENET\_E-NHANCEDBUFFERDESCRIPTOR\_MODE is defined. ENET\_Init should be called before calling this API.

## Note

The PTP 1588 time-stamp second increase though time-stamp interrupt handler and the transmit time-stamp store is done through transmit interrupt handler. As a result, the TS interrupt and TX interrupt are enabled when you call this API.

## Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>base</i>      | ENET peripheral base address.   |
| <i>handle</i>    | ENET handler pointer.           |
| <i>ptpConfig</i> | The ENET PTP1588 configuration. |

**11.7.45 void ENET\_Ptp1588StartTimer ( ENET\_Type \* *base*, uint32\_t *ptpClkSrc* )**

This function is used to initialize the PTP timer. After the PTP starts, the PTP timer starts running.

## Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>base</i>      | ENET peripheral base address.      |
| <i>ptpClkSrc</i> | The clock source of the PTP timer. |

**11.7.46 static void ENET\_Ptp1588StopTimer ( ENET\_Type \* *base* ) [inline], [static]**

This function is used to stops the ENET PTP timer.

## Function Documentation

Parameters

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

**11.7.47 void ENET\_Ptp1588AdjustTimer ( ENET\_Type \* *base*, uint32\_t *corrIncrease*, uint32\_t *corrPeriod* )**

Parameters

|                     |                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | ENET peripheral base address.                                                                                                                                                                                                                              |
| <i>corrIncrease</i> | The correction increment value. This value is added every time the correction timer expires. A value less than the PTP timer frequency(1/ptpClkSrc) slows down the timer, a value greater than the 1/ptpClkSrc speeds up the timer.                        |
| <i>corrPeriod</i>   | The PTP timer correction counter wrap-around value. This defines after how many timer clock the correction counter should be reset and trigger a correction increment on the timer. A value of 0 disables the correction counter and no correction occurs. |

**11.7.48 static void ENET\_Ptp1588SetChannelMode ( ENET\_Type \* *base*, enet\_ptp\_timer\_channel\_t *channel*, enet\_ptp\_timer\_channel\_mode\_t *mode*, bool *intEnable* ) [inline], [static]**

Parameters

|                  |                                                                  |
|------------------|------------------------------------------------------------------|
| <i>base</i>      | ENET peripheral base address.                                    |
| <i>channel</i>   | The ENET PTP timer channel number.                               |
| <i>mode</i>      | The PTP timer channel mode, see "enet_ptp_timer_channel_mode_t". |
| <i>intEnable</i> | Enables or disables the interrupt.                               |

**11.7.49 static void ENET\_Ptp1588SetChannelCmpValue ( ENET\_Type \* *base*, enet\_ptp\_timer\_channel\_t *channel*, uint32\_t *cmpValue* ) [inline], [static]**

Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>base</i>     | ENET peripheral base address.                          |
| <i>channel</i>  | The PTP timer channel, see "enet_ptp_timer_channel_t". |
| <i>cmpValue</i> | The compare value for the compare setting.             |

### 11.7.50 static bool ENET\_Ptp1588GetChannelStatus ( ENET\_Type \* *base*, enet\_ptp\_timer\_channel\_t *channel* ) [inline], [static]

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | ENET peripheral base address.       |
| <i>channel</i> | The IEEE 1588 timer channel number. |

Returns

True or false, Compare or capture operation status

### 11.7.51 static void ENET\_Ptp1588ClearChannelStatus ( ENET\_Type \* *base*, enet\_ptp\_timer\_channel\_t *channel* ) [inline], [static]

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | ENET peripheral base address.       |
| <i>channel</i> | The IEEE 1588 timer channel number. |

### 11.7.52 void ENET\_Ptp1588GetTimer ( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_ptp\_time\_t \* *ptpTime* )

Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                 |
| <i>handle</i>  | The ENET state pointer. This is the same state pointer used in the ENET_Init. |
| <i>ptpTime</i> | The PTP timer structure.                                                      |

## Function Documentation

11.7.53 **void ENET\_Ptp1588SetTimer( ENET\_Type \* *base*, enet\_handle\_t \* *handle*, enet\_ptp\_time\_t \* *ptpTime* )**

Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                 |
| <i>handle</i>  | The ENET state pointer. This is the same state pointer used in the ENET_Init. |
| <i>ptpTime</i> | The timer to be set to the PTP timer.                                         |

#### 11.7.54 void ENET\_Ptp1588TimerIRQHandler ( ENET\_Type \* *base*, enet\_handle\_t \* *handle* )

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                                                 |
| <i>handle</i> | The ENET state pointer. This is the same state pointer used in the ENET_Init. |

#### 11.7.55 status\_t ENET\_GetRxFrameTime ( enet\_handle\_t \* *handle*, enet\_ptp\_time\_data\_t \* *ptpTimeData* )

This function is used for PTP stack to get the timestamp captured by the ENET driver.

Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>handle</i>      | The ENET handler pointer. This is the same state pointer used in ENET_Init. |
| <i>ptpTimeData</i> | The special PTP timestamp data for search the receive timestamp.            |

Return values

|                                     |                             |
|-------------------------------------|-----------------------------|
| <i>kStatus_Success</i>              | Get 1588 timestamp success. |
| <i>kStatus_ENET_PtpTs-RingEmpty</i> | 1588 timestamp ring empty.  |
| <i>kStatus_ENET_PtpTs-RingFull</i>  | 1588 timestamp ring full.   |

#### 11.7.56 status\_t ENET\_GetTxFrameTime ( enet\_handle\_t \* *handle*, enet\_ptp\_time\_data\_t \* *ptpTimeData* )

This function is used for PTP stack to get the timestamp captured by the ENET driver.

## Function Documentation

Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>handle</i>      | The ENET handler pointer. This is the same state pointer used in ENET_Init. |
| <i>ptpTimeData</i> | The special PTP timestamp data for search the receive timestamp.            |

Return values

|                                     |                             |
|-------------------------------------|-----------------------------|
| <i>kStatus_Success</i>              | Get 1588 timestamp success. |
| <i>kStatus_ENET_PtpTs-RingEmpty</i> | 1588 timestamp ring empty.  |
| <i>kStatus_ENET_PtpTs-RingFull</i>  | 1588 timestamp ring full.   |

# Chapter 12

## ESAI: Enhanced Serial Audio Interface

### 12.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Enhanced Serial Audio Interface (ESAI) module of MCUXpresso SDK devices.

ESAI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low-level APIs. Functional APIs can be used for ESAI initialization/configuration/operation for the optimization/customization purpose. Using the functional API requires the knowledge of the ESAI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. ESAI 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 esai\_handle\_t as the first parameter. Initialize the handle by calling the [ESAI\\_TransferTxCreateHandle\(\)](#) or [ESAI\\_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [ESAI\\_TransferSendNonBlocking\(\)](#) and [ESAI\\_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus\_ESAI\_TxIdle and kStatus\_ESAI\_RxIdle status.

### 12.2 Typical use case

#### 12.2.1 ESAI Send/Receive using an interrupt method

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

#### 12.2.2 ESAI Send/receive using a DMA method

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

## Modules

- [ESAI eDMA Driver](#)

## Data Structures

- struct [esai\\_customer\\_protocol\\_t](#)

## Typical use case

- *ESAI customer defined audio format.* [More...](#)
- struct `esai_config_t`  
*ESAI user configuration structure.* [More...](#)
- struct `esai_format_t`  
*esai transfer format* [More...](#)
- struct `esai_transfer_t`  
*ESAI transfer structure.* [More...](#)
- struct `esai_handle_t`  
*ESAI handle structure.* [More...](#)

## Macros

- `#define ESAI_XFER_QUEUE_SIZE (4)`  
*ESAI transfer queue size, user can refine it according to use case.*

## Typedefs

- `typedef void(* esai_transfer_callback_t )(ESAI_Type *base, esai_handle_t *handle, status_t status, void *userData)`  
*ESAI transfer callback prototype.*

## Enumerations

- enum `_esai_status_t` {  
  `kStatus_ESAI_TxBusy` = MAKE\_STATUS(kStatusGroup\_ESAI, 0),  
  `kStatus_ESAI_RxBusy` = MAKE\_STATUS(kStatusGroup\_ESAI, 1),  
  `kStatus_ESAI_TxError` = MAKE\_STATUS(kStatusGroup\_ESAI, 2),  
  `kStatus_ESAI_RxError` = MAKE\_STATUS(kStatusGroup\_ESAI, 3),  
  `kStatus_ESAI_QueueFull` = MAKE\_STATUS(kStatusGroup\_ESAI, 4),  
  `kStatus_ESAI_TxIdle` = MAKE\_STATUS(kStatusGroup\_ESAI, 5),  
  `kStatus_ESAI_RxIdle` = MAKE\_STATUS(kStatusGroup\_ESAI, 6) }  
*ESAI return status.*
- enum `esai_mode_t` {  
  `kESAI_NormalMode` = 0x0U,  
  `kESAI_NetworkMode` }  
*Define the ESAI bus type.*
- enum `esai_protocol_t` {  
  `kESAI_BusLeftJustified` = 0x0U,  
  `kESAI_BusRightJustified`,  
  `kESAI_BusI2S`,  
  `kESAI_BusPCMA`,  
  `kESAI_BusPCMB`,  
  `kESAI_BusTDM`,  
  `kESAI_BusCustomerNormal`,  
  `kESAI_BusCustomerNetwork` }  
*Define the ESAI bus type.*
- enum `esai_master_slave_t` {  
  `kESAI_Master` = 0x0U,

- ```

kESAI_Slave = 0x1U }

Master or slave mode.
• enum esai_sync_mode_t {
    kESAI_ModeAsync = 0x0U,
    kESAI_ModeSync }

Synchronous or asynchronous mode.
• enum esai_hclk_source_t
    Mater clock source.
• enum esai_clock_polarity_t {
    kESAI_ClockActiveHigh = 0x0U,
    kESAI_ClockActiveLow }

Bit clock source.
• enum esai_shift_direction_t {
    kESAI_ShifterMSB = 0x0,
    kESAI_ShifterLSB = 0x1 }
• enum esai_clock_direction_t {
    kESAI_ClockInput = 0x0,
    kESAI_ClockOutput = 0x1 }
• enum _esai_interrupt_enable_t {
    kESAI_LastSlotInterruptEnable,
    kESAI_TransmitInterruptEnable = ESAI_TCR_TIE_MASK,
    kESAI_EvenSlotDataInterruptEnable = ESAI_TCR_TEDIE_MASK,
    kESAI_ExceptionInterruptEnable = ESAI_TCR_TEIE_MASK }

The ESAI interrupt enable flag.
• enum _esai_flags {
    kESAI_TransmitInitFlag = ESAI_ESR_TINIT_MASK,
    kESAI_ReceiveFIFOFullFlag = ESAI_ESR_RFF_MASK,
    kESAI_TransmitFIFOEmptyFlag = ESAI_ESR_TFE_MASK,
    kESAI_TransmitLastSlotFlag = ESAI_ESR_TLS_MASK,
    kESAI_TransmitDataExceptionFlag = ESAI_ESR_TDE_MASK,
    kESAI_TransmitEvenDataFlag = ESAI_ESR_TED_MASK,
    kESAI_TransmitDataFlag = ESAI_ESR_TD_MASK,
    kESAI_ReceiveLastSlot = ESAI_ESR_RLS_MASK,
    kESAI_ReceiveDataException = ESAI_ESR_RDE_MASK,
    kESAI_ReceiveEvenData = ESAI_ESR_RED_MASK,
    kESAI_ReceiveData = ESAI_ESR_RD_MASK }

The ESAI status flag.
• enum _esai_sai_flags {

```

Typical use case

```
kESAI_TransmitOddRegEmpty = ESAI_SAISR_TODFE_MASK,  
kESAI_TransmitEvenRegEmpty = ESAI_SAISR_TEDE_MASK,  
kESAI_TransmitRegEmpty = ESAI_SAISR_TDE_MASK,  
kESAI_TransmitUnderrunError = ESAI_SAISR_TUE_MASK,  
kESAI_TransmitFrameSync = ESAI_SAISR_TFS_MASK,  
kESAI_RecceiveOddRegFull = ESAI_SAISR_RODF_MASK,  
kESAI_ReceiveEvenRegFull = ESAI_SAISR_RDF_MASK,  
kESAI_RecceiveOverrunError = ESAI_SAISR_ROE_MASK,  
kESAI_ReceiveFrameSync = ESAI_SAISR_RFS_MASK,  
kESAI_SerialInputFlag2 = ESAI_SAISR_IF2_MASK,  
kESAI_SerialInputFlag1 = ESAI_SAISR_IF1_MASK,  
kESAI_SerialInputFlag0 = ESAI_SAISR_IF0_MASK }
```

SAI interface port status flag.

- enum `esai_sample_rate_t` {
 kESAI_SampleRate8KHz = 8000U,
 kESAI_SampleRate11025Hz = 11025U,
 kESAI_SampleRate12KHz = 12000U,
 kESAI_SampleRate16KHz = 16000U,
 kESAI_SampleRate22050Hz = 22050U,
 kESAI_SampleRate24KHz = 24000U,
 kESAI_SampleRate32KHz = 32000U,
 kESAI_SampleRate44100Hz = 44100U,
 kESAI_SampleRate48KHz = 48000U,
 kESAI_SampleRate96KHz = 96000U }

Audio sample rate.

- enum `esai_word_width_t` {
 kESAI_WordWidth8bits = 8U,
 kESAI_WordWidth16bits = 16U,
 kESAI_WordWidth24bits = 24U,
 kESAI_WordWidth32bits = 32U }

Audio word width.

- enum `esai_slot_format_t` {

```

kESAI_SlotLen8WordLen8 = 0x0U,
kESAI_SlotLen12WordLen8 = 0x04U,
kESAI_SlotLen12WordLen12 = 0x01U,
kESAI_SlotLen16WordLen8 = 0x08U,
kESAI_SlotLen16WordLen12 = 0x05U,
kESAI_SlotLen16WordLen16 = 0x02U,
kESAI_SlotLen20WordLen8 = 0x0CU,
kESAI_SlotLen20WordLen12 = 0x09U,
kESAI_SlotLen20WordLen16 = 0x06U,
kESAI_SlotLen20WordLen20 = 0x03U,
kESAI_SlotLen24WordLen8 = 0x10U,
kESAI_SlotLen24WordLen12 = 0x0DU,
kESAI_SlotLen24WordLen16 = 0x0AU,
kESAI_SlotLen24WordLen20 = 0x07U,
kESAI_SlotLen24WordLen24 = 0x1EU,
kESAI_SlotLen32WordLen8 = 0x18U,
kESAI_SlotLen32WordLen12 = 0x15U,
kESAI_SlotLen32WordLen16 = 0x12U,
kESAI_SlotLen32WordLen20 = 0x0FU,
kESAI_SlotLen32WordLen24 = 0x1FU }

```

Driver version

- #define **FSL_ESAI_DRIVER_VERSION** (MAKE_VERSION(2, 0, 1))
Version 2.0.1.

Initialization and deinitialization

- void **ESAI_Init** (ESAI_Type *base, esai_config_t *config)
Initializes the ESAI peripheral.
- void **ESAI_GetDefaultConfig** (esai_config_t *config)
Sets the ESAI configuration structure to default values.
- void **ESAI_Deinit** (ESAI_Type *base)
De-initializes the ESAI peripheral.
- static void **ESAI_Enable** (ESAI_Type *base, bool enable)
Enable/Disable the ESAI peripheral internal logic.
- static void **ESAI_Reset** (ESAI_Type *base)
Reset ESAI internal logic.
- void **ESAI_TxReset** (ESAI_Type *base)
Reset ESAI all tx sections.
- void **ESAI_RxReset** (ESAI_Type *base)
Reset ESAI all rx sections.
- static void **ESAI_TxResetFIFO** (ESAI_Type *base)
Resets the ESAI Tx FIFO.
- static void **ESAI_RxResetFIFO** (ESAI_Type *base)
Resets the ESAI Rx FIFO.
- void **ESAI_TxEnable** (ESAI_Type *base, uint8_t partMap)
Enables/disables ESAI Tx.

Typical use case

- void **ESAI_RxEnable** (ESAI_Type *base, uint8_t partMap)
Enables/disables ESAI Rx.
- static void **ESAI_TxEnableFIFO** (ESAI_Type *base, bool enable)
Enables/disables ESAI Tx FIFO.
- static void **ESAI_RxEnableFIFO** (ESAI_Type *base, bool enable)
Enables/disables ESAI Rx FIFO.
- static void **ESAI_TxSetSlotMask** (ESAI_Type *base, uint32_t slot)
Set ESAI Tx slot mask value.
- static void **ESAI_RxSetSlotMask** (ESAI_Type *base, uint32_t slot)
Set ESAI Rx slot mask value.

Status

- static uint32_t **ESAI_GetStatusFlag** (ESAI_Type *base)
Gets the ESAI status flag state.
- static uint32_t **ESAI_GetSAIStatusFlag** (ESAI_Type *base)
Gets the ESAI SAI port status flag state.
- static uint32_t **ESAI_GetTxFIFOStatus** (ESAI_Type *base)
Gets the ESAI Tx FIFO state.
- static uint32_t **ESAI_GetRxFIFOStatus** (ESAI_Type *base)
Gets the ESAI Rx status flag state.

Interrupts

- static void **ESAI_TxEnableInterrupts** (ESAI_Type *base, uint32_t mask)
Enables ESAI Tx interrupt requests.
- static void **ESAI_RxEnableInterrupts** (ESAI_Type *base, uint32_t mask)
Enables ESAI Rx interrupt requests.
- static void **ESAI_TxDisableInterrupts** (ESAI_Type *base, uint32_t mask)
Disables ESAI Tx interrupt requests.
- static void **ESAI_RxDisableInterrupts** (ESAI_Type *base, uint32_t mask)
Disables ESAI Rx interrupt requests.

DMA Control

- static uint32_t **ESAI_TxGetDataRegisterAddress** (ESAI_Type *base)
Gets the ESAI Tx data register address.
- static uint32_t **ESAI_RxGetDataRegisterAddress** (ESAI_Type *base)
Gets the ESAI Rx data register address.

Bus Operations

- void **ESAI_TxSetFormat** (ESAI_Type *base, **esai_format_t** *format, uint32_t hckClockHz, uint32_t hckSourceClockHz)
Configures the ESAI Tx audio format.
- void **ESAI_RxSetFormat** (ESAI_Type *base, **esai_format_t** *format, uint32_t hckClockHz, uint32_t hckSourceClockHz)
Configures the ESAI Rx audio format.
- void **ESAI_WriteBlocking** (ESAI_Type *base, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Sends data using a blocking method.

- static void [ESAI_WriteData](#) (ESAI_Type *base, uint32_t data)
Writes data into ESAI FIFO.
- void [ESAI_ReadBlocking](#) (ESAI_Type *base, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Receives data using a blocking method.
- static uint32_t [ESAI_ReadData](#) (ESAI_Type *base, uint32_t channel)
Reads data from ESAI FIFO.

Transactional

- void [ESAI_TransferTxCreateHandle](#) (ESAI_Type *base, esai_handle_t *handle, [esai_transfer_callback_t](#) callback, void *userData)
Initializes the ESAI Tx handle.
- void [ESAI_TransferRxCreateHandle](#) (ESAI_Type *base, esai_handle_t *handle, [esai_transfer_callback_t](#) callback, void *userData)
Initializes the ESAI Rx handle.
- status_t [ESAI_TransferTxSetFormat](#) (ESAI_Type *base, esai_handle_t *handle, [esai_format_t](#) *format, uint32_t hckClockHz, uint32_t hckSourceClockHz)
Configures the ESAI Tx audio format.
- status_t [ESAI_TransferRxSetFormat](#) (ESAI_Type *base, esai_handle_t *handle, [esai_format_t](#) *format, uint32_t hckClockHz, uint32_t hckSourceClockHz)
Configures the ESAI Rx audio format.
- status_t [ESAI_TransferSendNonBlocking](#) (ESAI_Type *base, esai_handle_t *handle, [esai_transfer_t](#) *xfer)
Performs an interrupt non-blocking send transfer on ESAI.
- status_t [ESAI_TransferReceiveNonBlocking](#) (ESAI_Type *base, esai_handle_t *handle, [esai_transfer_t](#) *xfer)
Performs an interrupt non-blocking receive transfer on ESAI.
- status_t [ESAI_TransferGetSendCount](#) (ESAI_Type *base, esai_handle_t *handle, size_t *count)
Gets a set byte count.
- status_t [ESAI_TransferGetReceiveCount](#) (ESAI_Type *base, esai_handle_t *handle, size_t *count)
Gets a received byte count.
- void [ESAI_TransferAbortSend](#) (ESAI_Type *base, esai_handle_t *handle)
Aborts the current send.
- void [ESAI_TransferAbortReceive](#) (ESAI_Type *base, esai_handle_t *handle)
Aborts the current IRQ receive.
- void [ESAI_TransferTxHandleIRQ](#) (ESAI_Type *base, esai_handle_t *handle)
Tx interrupt handler.
- void [ESAI_TransferRxHandleIRQ](#) (ESAI_Type *base, esai_handle_t *handle)
Rx interrupt handler.

12.3 Data Structure Documentation

12.3.1 struct esai_customer_protocol_t

Data Fields

- [esai_mode_t](#) mode
ESAI mode, network, normal or on demand mode.
- [esai_shift_direction_t](#) shiftDirection

Data Structure Documentation

- bool **fsEarly**
Data shift direction, MSB or LSB.
- bool **ifZeroPading**
If the frame sync one bit early.
- bool **dataAlign**
If padding zero.
- bool **fsOneBit**
Data left aligned or right aligned.
- uint8_t **slotNum**
If the frame sync one word length or one bit length.
- Slot number for the audio format.

12.3.2 struct esai_config_t

Data Fields

- esai_sync_mode_t **syncMode**
ESAI sync mode, control Tx/Rx clock sync.
- esai_protocol_t **txProtocol**
Use which kind of protocol.
- esai_protocol_t **rxProtocol**
Use which kind of protocol.
- esai_customer_protocol_t **txCustomer**
Audio protocol customer uses for tx.
- esai_customer_protocol_t **rxCustomer**
Audio protocol customer uses for rx.
- esai_master_slave_t **master**
Master or slave.
- esai_clock_direction_t **txHckDirection**
Tx HCK direction, input or output.
- esai_clock_direction_t **rxHckDirection**
Rx HCK direction, input or output.
- esai_hclk_source_t **txHckSource**
Tx HCK input clock source.
- esai_hclk_source_t **rxHckSource**
Rx HCK input clock source.
- esai_hclk_source_t **txHckOutputSource**
Tx HCK pin output clock source.
- esai_hclk_source_t **rxHckOutputSource**
Rx HCK pin output clock source.
- esai_clock_polarity_t **txHckPolarity**
Tx HCK polarity.
- esai_clock_polarity_t **txFsPolarity**
Tx frame sync polarity.
- esai_clock_polarity_t **txSckPolarity**
Tx bit clock polarity.
- esai_clock_polarity_t **rxHckPolarity**
Rx HCK polarity.
- esai_clock_polarity_t **rxFsPolarity**

- **esai_clock_polarity_t rxSckPolarity**
Rx frame sync polarity.
- **uint8_t txWatermark**
Rx bit clock polarity.
- **uint8_t rxWatermark**
Tx transfer watermark.
- **uint8_t partMap**
Rx receive watermark.

12.3.3 struct esai_format_t

Data Fields

- **esai_sample_rate_t sampleRate_Hz**
Sample rate of audio data.
- **esai_slot_format_t slotType**
Slot format for audio format.
- **uint8_t partMap**
The sections enabled, 0x1 means TE0 enabled, 0x2 means TE1 enabled, 0x4 means TE2, etc.

12.3.4 struct esai_transfer_t

Data Fields

- **uint8_t * data**
Data start address to transfer.
- **size_t dataSize**
Transfer size.

12.3.4.0.0.33 Field Documentation

12.3.4.0.0.33.1 uint8_t* esai_transfer_t::data

12.3.4.0.0.33.2 size_t esai_transfer_t::dataSize

12.3.5 struct _esai_handle

Data Fields

- **uint32_t state**
Transfer status.
- **esai_transfer_callback_t callback**
Callback function called at transfer event.
- **void * userData**
Callback parameter passed to callback function.
- **uint8_t bitWidth**
Bit width for transfer, 8/16/24/32 bits.

Enumeration Type Documentation

- `uint8_t slotLen`
Slot length of the audio data.
- `uint8_t partMap`
Enabled part map.
- `esai_transfer_t esaiQueue [ESAI_XFER_QUEUE_SIZE]`
Transfer queue storing queued transfer.
- `size_t transferSize [ESAI_XFER_QUEUE_SIZE]`
Data bytes need to transfer.
- `volatile uint8_t queueUser`
Index for user to queue transfer.
- `volatile uint8_t queueDriver`
Index for driver to get the transfer data and size.
- `uint8_t watermark`
Watermark value.

12.4 Macro Definition Documentation

12.4.1 #define `ESAI_XFER_QUEUE_SIZE` (4)

12.5 Enumeration Type Documentation

12.5.1 enum `_esai_status_t`

Enumerator

- `kStatus_ESAI_TxBusy` ESAI Tx is busy.
- `kStatus_ESAI_RxBusy` ESAI Rx is busy.
- `kStatus_ESAI_TxError` ESAI Tx FIFO error.
- `kStatus_ESAI_RxError` ESAI Rx FIFO error.
- `kStatus_ESAI_QueueFull` ESAI transfer queue is full.
- `kStatus_ESAI_TxIdle` ESAI Tx is idle.
- `kStatus_ESAI_RxIdle` ESAI Rx is idle.

12.5.2 enum `esai_mode_t`

Enumerator

- `kESAI_NormalMode` Use normal mode.
- `kESAI_NetworkMode` Network mode.

12.5.3 enum `esai_protocol_t`

Enumerator

- `kESAI_BusLeftJustified` Uses left justified format.

kESAI_BusRightJustified Uses right justified format.

kESAI_BusI2S Uses I2S format.

kESAI_BusPCMA Uses I2S PCM A format.

kESAI_BusPCMB Uses I2S PCM B format.

kESAI_BusTDM Use TDM mode.

kESAI_BusCustomerNormal Customer defined normal mode.

kESAI_BusCustomerNetwork Customer defined network mode.

12.5.4 enum esai_master_slave_t

Enumerator

kESAI_Master Master mode.

kESAI_Slave Slave mode.

12.5.5 enum esai_sync_mode_t

Enumerator

kESAI_ModeAsync Asynchronous mode.

kESAI_ModeSync Synchronous mode (with receiver or transmit)

12.5.6 enum esai_clock_polarity_t

Enumerator

kESAI_ClockActiveHigh Clock active while high.

kESAI_ClockActiveLow Clock active while low.

12.5.7 enum esai_shift_direction_t

Enumerator

kESAI_ShifterMSB Data is shifted MSB first.

kESAI_ShifterLSB Data is shifted LSB first.

Enumeration Type Documentation

12.5.8 enum esai_clock_direction_t

Enumerator

kESAI_ClockInput Clock direction is input.

kESAI_ClockOutput Clock direction is output.

12.5.9 enum _esai_interrupt_enable_t

Enumerator

kESAI_LastSlotInterruptEnable Enable interrupt at the beginning of last slot of frame in network mode.

kESAI_TransmitInterruptEnable Transmit/receive even slot data interrupt.

kESAI_EvenSlotDataInterruptEnable Transmit/receive even slot data interrupt.

kESAI_ExceptionInterruptEnable FIFO error flag.

12.5.10 enum _esai_flags

Enumerator

kESAI_TransmitInitFlag Indicates transmit FIFO is writing the first word.

kESAI_ReceiveFIFOFullFlag Receive FIFO full flag.

kESAI_TransmitFIFOEmptyFlag Transmit FIFO empty.

kESAI_TransmitLastSlotFlag Transmit last slot.

kESAI_TransmitDataExceptionFlag Transmit data exception.

kESAI_TransmitEvenDataFlag Transmit even data.

kESAI_TransmitDataFlag Transmit data.

kESAI_ReceiveLastSlot Receive last slot.

kESAI_ReceiveDataException Receive data exception.

kESAI_ReceiveEvenData Receive even data.

kESAI_ReceiveData Receive data.

12.5.11 enum _esai_sai_flags

Enumerator

kESAI_TransmitOddRegEmpty Enabled transmitter register empty at odd slot.

kESAI_TransmitEvenRegEmpty Enabled transmitter register empty at even slot.

kESAI_TransmitRegEmpty All data in enabled transmitter register send to shifter.

kESAI_TransmitUnderrunError Serial shifter empty and a transmit slot begins.

kESAI_TransmitFrameSync A transmit frame sync occurred in the current time slot.

kESAI_ReceiveOddRegFull Enabled receiver register full at odd slot.
kESAI_ReceiveEvenRegFull Enabled receiver register full at even slot.
kESAI_RecceiveOverrunError Receive data register overrun flag.
kESAI_ReceiveFrameSync Receive frame sync flag, indicate a frame sync occurs.
kESAI_SerialInputFlag2 Serial input flag 2.
kESAI_SerialInputFlag1 Serial in out flag 1.
kESAI_SerialInputFlag0 Serial input flag 0.

12.5.12 enum esai_sample_rate_t

Enumerator

kESAI_SampleRate8KHz Sample rate 8000 Hz.
kESAI_SampleRate11025KHz Sample rate 11025 Hz.
kESAI_SampleRate12KHz Sample rate 12000 Hz.
kESAI_SampleRate16KHz Sample rate 16000 Hz.
kESAI_SampleRate22050KHz Sample rate 22050 Hz.
kESAI_SampleRate24KHz Sample rate 24000 Hz.
kESAI_SampleRate32KHz Sample rate 32000 Hz.
kESAI_SampleRate44100KHz Sample rate 44100 Hz.
kESAI_SampleRate48KHz Sample rate 48000 Hz.
kESAI_SampleRate96KHz Sample rate 96000 Hz.

12.5.13 enum esai_word_width_t

Enumerator

kESAI_WordWidth8bits Audio data width 8 bits.
kESAI_WordWidth16bits Audio data width 16 bits.
kESAI_WordWidth24bits Audio data width 24 bits.
kESAI_WordWidth32bits Audio data width 32 bits.

12.5.14 enum esai_slot_format_t

Enumerator

kESAI_SlotLen8WordLen8 Slot length 8 bits, word length 8 bits.
kESAI_SlotLen12WordLen8 Slot length 12 bits, word length 8 bits.
kESAI_SlotLen12WordLen12 Slot length 12 bits, word length 12 bits.
kESAI_SlotLen16WordLen8 Slot length 16 bits, word length 8 bits.
kESAI_SlotLen16WordLen12 Slot length 16 bits, word length 12 bits.

Function Documentation

kESAI_SlotLen16WordLen16 Slot length 16 bits, word length 16 bits.
kESAI_SlotLen20WordLen8 Slot length 20 bits, word length 8 bits.
kESAI_SlotLen20WordLen12 Slot length 20 bits, word length 12 bits.
kESAI_SlotLen20WordLen16 Slot length 20 bits, word length 16 bits.
kESAI_SlotLen20WordLen20 Slot length 20 bits, word length 20 bits.
kESAI_SlotLen24WordLen8 Slot length 24 bits, word length 8 bits.
kESAI_SlotLen24WordLen12 Slot length 24 bits, word length 12 bits.
kESAI_SlotLen24WordLen16 Slot length 24 bits, word length 16 bits.
kESAI_SlotLen24WordLen20 Slot length 24 bits, word length 20 bits.
kESAI_SlotLen24WordLen24 Slot length 24 bits, word length 24 bits.
kESAI_SlotLen32WordLen8 Slot length 32 bits, word length 8 bits.
kESAI_SlotLen32WordLen12 Slot length 32 bits, word length 12 bits.
kESAI_SlotLen32WordLen16 Slot length 32 bits, word length 16 bits.
kESAI_SlotLen32WordLen20 Slot length 32 bits, word length 20 bits.
kESAI_SlotLen32WordLen24 Slot length 32 bits, word length 24 bits.

12.6 Function Documentation

12.6.1 void **ESAI_Init(** *ESAI_Type* * *base*, *esai_config_t* * *config* **)**

Ungates the ESAI clock, resets the module, and configures ESAI with a configuration structure. The configuration structure can be custom filled or set with default values by [ESAI_GetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the ESAI driver. Otherwise, accessing the ESAI module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	ESAI base pointer
<i>config</i>	ESAI configuration structure.

12.6.2 void **ESAI_GetDefaultConfig(** *esai_config_t* * *config* **)**

This API initializes the configuration structure for use in *ESAI_TxConfig()*. The initialized structure can remain unchanged in [ESAI_Init\(\)](#), or it can be modified before calling [ESAI_Init\(\)](#).

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

12.6.3 void **ESAI_Deinit** (**ESAI_Type** * *base*)

This API gates the ESAI clock. The ESAI module can't operate unless **ESAI_Init** is called to enable the clock.

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

12.6.4 static void **ESAI_Enable** (**ESAI_Type** * *base*, **bool** *enable*) [inline], [static]

Parameters

<i>base</i>	ESAI base pointer
<i>enable</i>	True means enable, false means disable.

12.6.5 static void **ESAI_Reset** (**ESAI_Type** * *base*) [inline], [static]

This API only resets the core logic, including the configuration registers, but not the ESAI FIFOs, users still needs to reset the ESAI fifo by calling **ESAI_TxResetFIFO** and **ESAI_RxResetFIFO**.

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

12.6.6 void **ESAI_TxReset** (**ESAI_Type** * *base*)

This API only resets the core logic of tx and all tx sections.

Function Documentation

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

12.6.7 void **ESAI_RxReset**(**ESAI_Type** * *base*)

This API only resets the core logic of rx and all rx sections.

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

12.6.8 static void **ESAI_TxResetFIFO**(**ESAI_Type** * *base*) [inline], [static]

This function only resets the ESAI Tx FIFO.

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

12.6.9 static void **ESAI_RxResetFIFO**(**ESAI_Type** * *base*) [inline], [static]

This function only resets the ESAI Rx FIFO.

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

12.6.10 void **ESAI_TxEnable**(**ESAI_Type** * *base*, **uint8_t** *partMap*)

Parameters

<i>base</i>	ESAI base pointer
<i>partMap</i>	Which parts need to be enabled. 0 means all part disabled. This parameter can be a combination of each parts, every part N is 2^N in part map.

12.6.11 void **ESAI_RxEnable** (**ESAI_Type** * *base*, **uint8_t** *partMap*)

Parameters

<i>base</i>	ESAI base pointer
<i>partMap</i>	Which parts need to be enabled. 0 means all part disabled. This parameter can be a combination of each parts, every part N is 2^N in part map.

12.6.12 static void **ESAI_TxEnableFIFO** (**ESAI_Type** * *base*, **bool** *enable*) [**inline**], [**static**]

Parameters

<i>base</i>	ESAI base pointer
<i>enable</i>	True means enable ESAI Tx, false means disable.

12.6.13 static void **ESAI_RxEnableFIFO** (**ESAI_Type** * *base*, **bool** *enable*) [**inline**], [**static**]

Parameters

<i>base</i>	ESAI base pointer
<i>enable</i>	True means enable ESAI Rx, false means disable.

12.6.14 static void **ESAI_TxSetSlotMask** (**ESAI_Type** * *base*, **uint32_t** *slot*) [**inline**], [**static**]

Function Documentation

Parameters

<i>base</i>	ESAI base pointer
<i>slot</i>	Slot number need to be masked for Tx.

12.6.15 static void EASI_RxSetSlotMask (**ESAI_Type * *base*, **uint32_t** *slot*) [inline], [static]**

Parameters

<i>base</i>	ESAI base pointer
<i>slot</i>	Slot number need to be masked for Rx

12.6.16 static uint32_t EASI_GetStatusFlag (**ESAI_Type * *base*) [inline], [static]**

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

Returns

ESAI status flag value. Use status flag to AND [_esai_flags](#) to get the related status.

12.6.17 static uint32_t EASI_GetSAIStatusFlag (**ESAI_Type * *base*) [inline], [static]**

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

Returns

ESAI status flag value. Use status flag to AND [_esai_sai_flags](#) to get the related status.

12.6.18 static uint32_t EASI_GetTxFIFOStatus (**ESAI_Type * *base*) [inline], [static]**

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

Returns

ESAI Tx status flag value.

12.6.19 static uint32_t **ESAI_GetRxFIFOStatus** (**ESAI_Type** * *base*) [inline], [static]

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

Returns

ESAI Rx status flag value.

12.6.20 static void **ESAI_TxEnableInterrupts** (**ESAI_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

<i>base</i>	ESAI base pointer
<i>mask</i>	interrupt source. The parameter can be a combination of elements in _esai_interrupt_enable_t.

12.6.21 static void **ESAI_RxEnableInterrupts** (**ESAI_Type** * *base*, **uint32_t** *mask*) [inline], [static]

Parameters

Function Documentation

<i>base</i>	ESAI base pointer
<i>mask</i>	interrupt source. The parameter can be a combination of elements in _esai_interrupt_enable_t.

**12.6.22 static void ESAI_TxDisableInterrupts (ESAI_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	ESAI base pointer
<i>mask</i>	interrupt source. The parameter can be a combination of elements in _esai_interrupt_enable_t.

**12.6.23 static void ESAI_RxDisableInterrupts (ESAI_Type * *base*, uint32_t *mask*)
[inline], [static]**

Parameters

<i>base</i>	ESAI base pointer
<i>mask</i>	interrupt source. The parameter can be a combination of elements in _esai_interrupt_enable_t.

**12.6.24 static uint32_t ESAI_TxGetDataRegisterAddress (ESAI_Type * *base*)
[inline], [static]**

This API is used to provide a transfer address for ESAI DMA transfer configuration.

Parameters

<i>base</i>	ESAI base pointer.
-------------	--------------------

Returns

data register address.

**12.6.25 static uint32_t ESAI_RxGetDataRegisterAddress (*ESAI_Type* * *base*)
[inline], [static]**

This API is used to provide a transfer address for ESAI DMA transfer configuration.

Function Documentation

Parameters

<i>base</i>	ESAI base pointer.
-------------	--------------------

Returns

data register address.

12.6.26 void ESAI_TxSetFormat (*ESAI_Type* * *base*, *esai_format_t* * *format*, *uint32_t* *hckClockHz*, *uint32_t* *hckSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	ESAI base pointer.
<i>format</i>	Pointer to ESAI audio data format structure.
<i>hckClockHz</i>	HCK clock frequency in Hz.
<i>hckSource-ClockHz</i>	HCK source clock frequency in Hz.

12.6.27 void ESAI_RxSetFormat (*ESAI_Type* * *base*, *esai_format_t* * *format*, *uint32_t* *hckClockHz*, *uint32_t* *hckSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	ESAI base pointer.
<i>format</i>	Pointer to ESAI audio data format structure.
<i>hckClockHz</i>	HCK clock frequency in Hz.
<i>hckSource-ClockHz</i>	HCK source clock frequency in Hz.

12.6.28 void ESAI_WriteBlocking (*ESAI_Type* * *base*, *uint32_t* *bitWidth*, *uint8_t* * *buffer*, *uint32_t* *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	ESAI base pointer.
<i>bitWidth</i>	How many bits in a audio word, usually 8/16/24 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

**12.6.29 static void ESAI_WriteData (*ESAI_Type* * *base*, *uint32_t* *data*)
[inline], [static]**

Parameters

<i>base</i>	ESAI base pointer.
<i>data</i>	Data needs to be written.

12.6.30 void ESAI_ReadBlocking (*ESAI_Type* * *base*, *uint32_t* *bitWidth*, *uint8_t* * *buffer*, *uint32_t* *size*)

Note

This function blocks by polling until data is ready to be sent.

Parameters

<i>base</i>	ESAI base pointer.
<i>bitWidth</i>	How many bits in a audio word, usually 8/16/24 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

**12.6.31 static *uint32_t* ESAI_ReadData (*ESAI_Type* * *base*, *uint32_t* *channel*)
[inline], [static]**

Function Documentation

Parameters

<i>base</i>	ESAI base pointer.
<i>channel</i>	Data channel used.

Returns

Data in ESAI FIFO.

12.6.32 void ESAI_TransferTxCreateHandle (*ESAI_Type* * *base*, *esai_handle_t* * *handle*, *esai_transfer_callback_t* *callback*, *void* * *userData*)

This function initializes the Tx handle for ESAI Tx transactional APIs. Call this function one time to get the handle initialized.

Parameters

<i>base</i>	ESAI base pointer
<i>handle</i>	ESAI handle pointer.
<i>callback</i>	pointer to user callback function
<i>userData</i>	user parameter passed to the callback function

12.6.33 void ESAI_TransferRxCreateHandle (*ESAI_Type* * *base*, *esai_handle_t* * *handle*, *esai_transfer_callback_t* *callback*, *void* * *userData*)

This function initializes the Rx handle for ESAI Rx transactional APIs. Call this function one time to get the handle initialized.

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI handle pointer.
<i>callback</i>	pointer to user callback function
<i>userData</i>	user parameter passed to the callback function

12.6.34 **status_t ESAI_TransferTxSetFormat(*ESAI_Type * base*, *esai_handle_t * handle*, *esai_format_t * format*, *uint32_t hckClockHz*, *uint32_t hckSourceClockHz*)**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Function Documentation

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI handle pointer.
<i>format</i>	Pointer to ESAI audio data format structure.
<i>hckClockHz</i>	HCK clock frequency in Hz.
<i>hckSource-ClockHz</i>	HCK clock source frequency in Hz.

Returns

Status of this function. Return value is one of status_t.

12.6.35 status_t ESAI_TransferRxSetFormat (ESAI_Type * *base*, esai_handle_t * *handle*, esai_format_t * *format*, uint32_t *hckClockHz*, uint32_t *hckSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI handle pointer.
<i>format</i>	Pointer to ESAI audio data format structure.
<i>hckClockHz</i>	HCK clock frequency in Hz.
<i>hckSource-ClockHz</i>	HCK clock source frequency in Hz.

Returns

Status of this function. Return value is one of status_t.

12.6.36 status_t ESAI_TransferSendNonBlocking (ESAI_Type * *base*, esai_handle_t * *handle*, esai_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the ESAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_ESAI_Busy, the transfer is finished.

Parameters

<i>base</i>	ESAI base pointer
<i>handle</i>	pointer to esai_handle_t structure which stores the transfer state
<i>xfer</i>	pointer to esai_transfer_t structure

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_ESAI_TxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

12.6.37 status_t ESAI_TransferReceiveNonBlocking (ESAI_Type * *base*, esai_handle_t * *handle*, esai_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the ESAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not *kStatus_ESAI_Busy*, the transfer is finished.

Parameters

<i>base</i>	ESAI base pointer
<i>handle</i>	pointer to esai_handle_t structure which stores the transfer state
<i>xfer</i>	pointer to esai_transfer_t structure

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_ESAI_RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

12.6.38 status_t ESAI_TransferGetSendCount (ESAI_Type * *base*, esai_handle_t * *handle*, size_t * *count*)

Function Documentation

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	pointer to esai_handle_t structure which stores the transfer state.
<i>count</i>	Bytes count sent.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

12.6.39 **status_t ESAI_TransferGetReceiveCount (*ESAI_Type * base, esai_handle_t * handle, size_t * count*)**

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	pointer to esai_handle_t structure which stores the transfer state.
<i>count</i>	Bytes count received.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

12.6.40 **void ESAI_TransferAbortSend (*ESAI_Type * base, esai_handle_t * handle*)**

Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	pointer to esai_handle_t structure which stores the transfer state.

12.6.41 void **ESAI_TransferAbortReceive** (**ESAI_Type** * *base*, **esai_handle_t** * *handle*)

Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	ESAI base pointer
<i>handle</i>	pointer to esai_handle_t structure which stores the transfer state.

12.6.42 void **ESAI_TransferTxHandleIRQ** (**ESAI_Type** * *base*, **esai_handle_t** * *handle*)

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	pointer to esai_handle_t structure.

12.6.43 void **ESAI_TransferRxHandleIRQ** (**ESAI_Type** * *base*, **esai_handle_t** * *handle*)

Parameters

<i>base</i>	ESAI base pointer.
-------------	--------------------

Function Documentation

<i>handle</i>	pointer to esai_handle_t structure.
---------------	-------------------------------------

12.7 ESAI eDMA Driver

12.7.1 Overview

Data Structures

- struct `esai_edma_handle_t`

ESAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

TypeDefs

- typedef void(* `esai_edma_callback_t`)(ESAI_Type *base, esai_edma_handle_t *handle, status_t status, void *userData)

ESAI eDMA transfer callback function for finish and error.

Driver version

- #define `FSL_ESAI_EDMA_DRIVER_VERSION` (MAKE_VERSION(2, 0, 0))
Version 2.0.0.

eDMA Transactional

- void `ESAI_TransferTxCreateHandleEDMA` (ESAI_Type *base, esai_edma_handle_t *handle, `esai_edma_callback_t` callback, void *userData, `edma_handle_t` *dmaHandle)
Initializes the ESAI eDMA handle.
- void `ESAI_TransferRxCreateHandleEDMA` (ESAI_Type *base, esai_edma_handle_t *handle, `esai_edma_callback_t` callback, void *userData, `edma_handle_t` *dmaHandle)
Initializes the ESAI Rx eDMA handle.
- void `ESAI_TransferTxSetFormatEDMA` (ESAI_Type *base, esai_edma_handle_t *handle, `esai_format_t` *format, uint32_t hckClockHz, uint32_t hclkSourceClockHz)
Configures the ESAI Tx audio format.
- void `ESAI_TransferRxSetFormatEDMA` (ESAI_Type *base, esai_edma_handle_t *handle, `esai_format_t` *format, uint32_t hckClockHz, uint32_t hclkSourceClockHz)
Configures the ESAI Rx audio format.
- status_t `ESAI_TransferSendEDMA` (ESAI_Type *base, esai_edma_handle_t *handle, `esai_transfer_t` *xfer)
Performs a non-blocking ESAI transfer using DMA.
- status_t `ESAI_TransferReceiveEDMA` (ESAI_Type *base, esai_edma_handle_t *handle, `esai_transfer_t` *xfer)
Performs a non-blocking ESAI receive using eDMA.
- void `ESAI_TransferAbortSendEDMA` (ESAI_Type *base, esai_edma_handle_t *handle)
Aborts a ESAI transfer using eDMA.
- void `ESAI_TransferAbortReceiveEDMA` (ESAI_Type *base, esai_edma_handle_t *handle)
Aborts a ESAI receive using eDMA.

ESAI eDMA Driver

- status_t [ESAI_TransferGetSendCountEDMA](#) (ESAI_Type *base, esai_edma_handle_t *handle, size_t *count)
Gets byte count sent by ESAI.
- status_t [ESAI_TransferGetReceiveCountEDMA](#) (ESAI_Type *base, esai_edma_handle_t *handle, size_t *count)
Gets byte count received by ESAI.

12.7.2 Data Structure Documentation

12.7.2.1 struct _esai_edma_handle

Data Fields

- **edma_handle_t * dmaHandle**
DMA handler for ESAI send.
- **uint8_t nbytes**
eDMA minor byte transfer count initially configured.
- **uint8_t bitWidth**
Bit width for transfer, 8/16/24/32 bits.
- **uint8_t slotLen**
Slot length of the audio data.
- **uint8_t count**
The transfer data count in a DMA request.
- **uint8_t chapterMap**
Section enabled for transfer.
- **uint32_t state**
Internal state for ESAI eDMA transfer.
- **esai_edma_callback_t callback**
Callback for users while transfer finish or error occurs.
- **void * userData**
User callback parameter.
- **edma_tcd_t tcd [ESAI_XFER_QUEUE_SIZE+1U]**
TCD pool for eDMA transfer.
- **esai_transfer_t esaiQueue [ESAI_XFER_QUEUE_SIZE]**
Transfer queue storing queued transfer.
- **size_t transferSize [ESAI_XFER_QUEUE_SIZE]**
Data bytes need to transfer.
- **volatile uint8_t queueUser**
Index for user to queue transfer.
- **volatile uint8_t queueDriver**
Index for driver to get the transfer data and size.

12.7.2.1.0.34 Field Documentation

12.7.2.1.0.34.1 `uint8_t esai_edma_handle_t::nbytes`

12.7.2.1.0.34.2 `edma_tcd_t esai_edma_handle_t::tcd[ESAI_XFER_QUEUE_SIZE+1U]`

12.7.2.1.0.34.3 `esai_transfer_t esai_edma_handle_t::esaiQueue[ESAI_XFER_QUEUE_SIZE]`

12.7.2.1.0.34.4 `volatile uint8_t esai_edma_handle_t::queueUser`

12.7.3 Function Documentation

12.7.3.1 `void ESAI_TransferTxCreateHandleEDMA (ESAI_Type * base,
esai_edma_handle_t * handle, esai_edma_callback_t callback, void * userData,
edma_handle_t * dmaHandle)`

This function initializes the ESAI master DMA handle, which can be used for other ESAI master transactional APIs. Usually, for a specified ESAI instance, call this API once to get the initialized handle.

ESAI eDMA Driver

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI eDMA handle pointer.
<i>base</i>	ESAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>dmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

**12.7.3.2 void ESAI_TransferRxCreateHandleEDMA (*ESAI_Type* * *base*,
 esai_edma_handle_t * *handle*, *esai_edma_callback_t* *callback*, *void* * *userData*,
 edma_handle_t * *dmaHandle*)**

This function initializes the ESAI slave DMA handle, which can be used for other ESAI master transactional APIs. Usually, for a specified ESAI instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI eDMA handle pointer.
<i>base</i>	ESAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>dmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

**12.7.3.3 void ESAI_TransferTxSetFormatEDMA (*ESAI_Type* * *base*, *esai_edma_handle_t*
 * *handle*, *esai_format_t* * *format*, *uint32_t* *hckClockHz*, *uint32_t*
 hclkSourceClockHz)**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

<i>base</i>	ESAI base pointer.
-------------	--------------------

<i>handle</i>	ESAI eDMA handle pointer.
<i>format</i>	Pointer to ESAI audio data format structure.
<i>hckClockHz</i>	HCK clock frequency in Hz.
<i>hclkSource-ClockHz</i>	HCK clock source frequency in Hz.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

12.7.3.4 void ESAI_TransferRxSetFormatEDMA (*ESAI_Type* * *base*, *esai_edma_handle_t* * *handle*, *esai_format_t* * *format*, *uint32_t* *hckClockHz*, *uint32_t* *hclkSourceClockHz*)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI eDMA handle pointer.
<i>format</i>	Pointer to ESAI audio data format structure.
<i>hckClockHz</i>	HCK clock frequency in Hz.
<i>hclkSource-ClockHz</i>	HCK clock source frequency in Hz.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

12.7.3.5 *status_t* ESAI_TransferSendEDMA (*ESAI_Type* * *base*, *esai_edma_handle_t* * *handle*, *esai_transfer_t* * *xfer*)

Note

This interface returns immediately after the transfer initiates. Call ESAI_GetTransferStatus to poll the transfer status and check whether the ESAI transfer is finished.

ESAI eDMA Driver

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a ESAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_TxBusy</i>	ESAI is busy sending data.

12.7.3.6 **status_t ESAI_TransferReceiveEDMA (*ESAI_Type * base, esai_edma_handle_t * handle, esai_transfer_t * xfer*)**

Note

This interface returns immediately after the transfer initiates. Call the `ESAI_GetReceiveRemainingBytes` to poll the transfer status and check whether the ESAI transfer is finished.

Parameters

<i>base</i>	ESAI base pointer
<i>handle</i>	ESAI eDMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a ESAI eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_RxBusy</i>	ESAI is busy receiving data.

12.7.3.7 **void ESAI_TransferAbortSendEDMA (*ESAI_Type * base, esai_edma_handle_t * handle*)**

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI eDMA handle pointer.

12.7.3.8 void ESAI_TransferAbortReceiveEDMA (**ESAI_Type** * *base*, **esai_edma_handle_t** * *handle*)

Parameters

<i>base</i>	ESAI base pointer
<i>handle</i>	ESAI eDMA handle pointer.

12.7.3.9 status_t ESAI_TransferGetSendCountEDMA (**ESAI_Type** * *base*, **esai_edma_handle_t** * *handle*, **size_t** * *count*)

Parameters

<i>base</i>	ESAI base pointer.
<i>handle</i>	ESAI eDMA handle pointer.
<i>count</i>	Bytes count sent by ESAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

12.7.3.10 status_t ESAI_TransferGetReceiveCountEDMA (**ESAI_Type** * *base*, **esai_edma_handle_t** * *handle*, **size_t** * *count*)

Parameters

<i>base</i>	ESAI base pointer
-------------	-------------------

ESAI eDMA Driver

<i>handle</i>	ESAI eDMA handle pointer.
<i>count</i>	Bytes count received by ESAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferIn-Progress</i>	There is no non-blocking transaction in progress.

Chapter 13

FlexCAN: Flex Controller Area Network Driver

13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flex Controller Area Network (FlexCAN) module of MCUXpresso SDK devices.

Modules

- [FlexCAN Driver](#)
- [FlexCAN eDMA Driver](#)

FlexCAN Driver

13.2 FlexCAN Driver

13.2.1 Overview

This section describes the programming interface of the FlexCAN driver. The FlexCAN driver configures FlexCAN module and provides functional and transactional interfaces to build the FlexCAN application.

13.2.2 Typical use case

13.2.2.1 Message Buffer Send Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/flexcan

13.2.2.2 Message Buffer Receive Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/flexcan

13.2.2.3 Receive FIFO Operation

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/flexcan

Data Structures

- struct [flexcan_frame_t](#)
FlexCAN message frame structure. [More...](#)
- struct [flexcan_fd_frame_t](#)
CAN FDmessage frame structure. [More...](#)
- struct [flexcan_timing_config_t](#)
FlexCAN protocol timing characteristic configuration structure. [More...](#)
- struct [flexcan_config_t](#)
FlexCAN module configuration structure. [More...](#)
- struct [flexcan_rx_mb_config_t](#)
FlexCAN Receive Message Buffer configuration structure. [More...](#)
- struct [flexcan_rx_fifo_config_t](#)
FlexCAN Rx FIFO configuration structure. [More...](#)
- struct [flexcan_mb_transfer_t](#)
FlexCAN Message Buffer transfer. [More...](#)
- struct [flexcan_fifo_transfer_t](#)
FlexCAN Rx FIFO transfer. [More...](#)
- struct [flexcan_handle_t](#)
FlexCAN handle structure. [More...](#)

Macros

- #define **FLEXCAN_ID_STD**(id) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)

FlexCAN Frame ID helper macro.
- #define **FLEXCAN_ID_EXT**(id)

Extend Frame ID helper macro.
- #define **FLEXCAN_RX_MB_STD_MASK**(id, rtr, ide)

FlexCAN Rx Message Buffer Mask helper macro.
- #define **FLEXCAN_RX_MB_EXT_MASK**(id, rtr, ide)

Extend Rx Message Buffer Mask helper macro.
- #define **FLEXCAN_RX_FIFO_STD_MASK_TYPE_A**(id, rtr, ide)

FlexCAN Rx FIFO Mask helper macro.
- #define **FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH**(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B upper part helper macro.
- #define **FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW**(id, rtr, ide)

Standard Rx FIFO Mask helper macro Type B lower part helper macro.
- #define **FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH**(id) (((uint32_t)(id)&0x7F8) << 21)

Standard Rx FIFO Mask helper macro Type C upper part helper macro.
- #define **FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH**(id) (((uint32_t)(id)&0x7F8) << 13)

Standard Rx FIFO Mask helper macro Type C mid-upper part helper macro.
- #define **FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW**(id) (((uint32_t)(id)&0x7F8) << 5)

Standard Rx FIFO Mask helper macro Type C mid-lower part helper macro.
- #define **FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW**(id) (((uint32_t)(id)&0x7F8) >> 3)

Standard Rx FIFO Mask helper macro Type C lower part helper macro.
- #define **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A**(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type A helper macro.
- #define **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH**(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B upper part helper macro.
- #define **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW**(id, rtr, ide)

Extend Rx FIFO Mask helper macro Type B lower part helper macro.
- #define **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH**(id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)

Extend Rx FIFO Mask helper macro Type C upper part helper macro.
- #define **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH**(id)

Extend Rx FIFO Mask helper macro Type C mid-upper part helper macro.
- #define **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW**(id)

Extend Rx FIFO Mask helper macro Type C mid-lower part helper macro.
- #define **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW**(id) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)

Extend Rx FIFO Mask helper macro Type C lower part helper macro.
- #define **FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A**(id, rtr, ide) **FLEXCAN_RX_FIFO_STD_MASK_TYPE_A**(id, rtr, ide)

FlexCAN Rx FIFO Filter helper macro.
- #define **FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH**(id, rtr, ide)

Standard Rx FIFO Filter helper macro Type B upper part helper macro.
- #define **FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW**(id, rtr, ide)

FlexCAN Driver

- Standard Rx FIFO Filter helper macro Type B lower part helper macro.
 - #define **FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH**(id)
- Standard Rx FIFO Filter helper macro Type C upper part helper macro.
 - #define **FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH**(id)
- Standard Rx FIFO Filter helper macro Type C mid-upper part helper macro.
 - #define **FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW**(id)
- Standard Rx FIFO Filter helper macro Type C mid-lower part helper macro.
 - #define **FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW**(id)
- Standard Rx FIFO Filter helper macro Type C lower part helper macro.
 - #define **FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A**(id, rtr, ide) **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A**(id, rtr, ide)
 - Extend Rx FIFO Filter helper macro Type A helper macro.
 - #define **FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH**(id, rtr, ide)
 - Extend Rx FIFO Filter helper macro Type B upper part helper macro.
 - #define **FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW**(id, rtr, ide)
 - Extend Rx FIFO Filter helper macro Type B lower part helper macro.
 - #define **FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH**(id)
 - Extend Rx FIFO Filter helper macro Type C upper part helper macro.
 - #define **FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH**(id)
 - Extend Rx FIFO Filter helper macro Type C mid-upper part helper macro.
 - #define **FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW**(id)
 - Extend Rx FIFO Filter helper macro Type C mid-lower part helper macro.
 - #define **FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW**(id) **FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW**(id)
 - Extend Rx FIFO Filter helper macro Type C lower part helper macro.

Typedefs

- **typedef void(* flexcan_transfer_callback_t)**(CAN_Type *base, flexcan_handle_t *handle, status_t status, uint32_t result, void *userData)
- FlexCAN transfer callback function.*

Enumerations

- enum `_flexcan_status` {

 `kStatus_FLEXCAN_TxBusy` = MAKE_STATUS(kStatusGroup_FLEXCAN, 0),

 `kStatus_FLEXCAN_TxIdle` = MAKE_STATUS(kStatusGroup_FLEXCAN, 1),

 `kStatus_FLEXCAN_TxSwitchToRx`,

 `kStatus_FLEXCAN_RxBusy` = MAKE_STATUS(kStatusGroup_FLEXCAN, 3),

 `kStatus_FLEXCAN_RxIdle` = MAKE_STATUS(kStatusGroup_FLEXCAN, 4),

 `kStatus_FLEXCAN_RxOverflow` = MAKE_STATUS(kStatusGroup_FLEXCAN, 5),

 `kStatus_FLEXCAN_RxFifoBusy` = MAKE_STATUS(kStatusGroup_FLEXCAN, 6),

 `kStatus_FLEXCAN_RxFifoIdle` = MAKE_STATUS(kStatusGroup_FLEXCAN, 7),

 `kStatus_FLEXCAN_RxFifoOverflow` = MAKE_STATUS(kStatusGroup_FLEXCAN, 8),

 `kStatus_FLEXCAN_RxFifoWarning` = MAKE_STATUS(kStatusGroup_FLEXCAN, 9),

 `kStatus_FLEXCAN_ErrorStatus` = MAKE_STATUS(kStatusGroup_FLEXCAN, 10),

 `kStatus_FLEXCAN_WakeUp` = MAKE_STATUS(kStatusGroup_FLEXCAN, 11),

 `kStatus_FLEXCAN_UnHandled` = MAKE_STATUS(kStatusGroup_FLEXCAN, 12) }
- FlexCAN transfer status.*
- enum `flexcan_frame_format_t` {

 `kFLEXCAN_FrameFormatStandard` = 0x0U,

 `kFLEXCAN_FrameFormatExtend` = 0x1U }
- FlexCAN frame format.*
- enum `flexcan_frame_type_t` {

 `kFLEXCAN_FrameTypeData` = 0x0U,

 `kFLEXCAN_FrameTypeRemote` = 0x1U }
- FlexCAN frame type.*
- enum `flexcan_clock_source_t` {

 `kFLEXCAN_ClkSrcOsc` = 0x0U,

 `kFLEXCAN_ClkSrcPeri` = 0x1U }
- FlexCAN clock source.*
- enum `flexcan_wake_up_source_t` {

 `kFLEXCAN_WakeupSrcUnfiltered` = 0x0U,

 `kFLEXCAN_WakeupSrcFiltered` = 0x1U }
- FlexCAN wake up source.*
- enum `flexcan_rx_fifo_filter_type_t` {

 `kFLEXCAN_RxFifoFilterTypeA` = 0x0U,

 `kFLEXCAN_RxFifoFilterTypeB`,

 `kFLEXCAN_RxFifoFilterTypeC`,

 `kFLEXCAN_RxFifoFilterTypeD` = 0x3U }
- FlexCAN Rx Fifo Filter type.*
- enum `flexcan_mb_size_t` {

 `kFLEXCAN_8BperMB` = 0x0U,

 `kFLEXCAN_16BperMB` = 0x1U,

 `kFLEXCAN_32BperMB` = 0x2U,

 `kFLEXCAN_64BperMB` = 0x3U }
- FlexCAN Message Buffer Data Size.*
- enum `flexcan_rx_fifo_priority_t` {

 `kFLEXCAN_RxFifoPrioLow` = 0x0U,

FlexCAN Driver

- ```
kFLEXCAN_RxFifoPrioHigh = 0x1U }
```
- FlexCAN Rx FIFO priority.*
- enum \_flexcan\_interrupt\_enable {  
 kFLEXCAN\_BusOffInterruptEnable = CAN\_CTRL1\_BOFFMSK\_MASK,  
 kFLEXCAN\_ErrorInterruptEnable = CAN\_CTRL1\_ERRMSK\_MASK,  
 kFLEXCAN\_RxWarningInterruptEnable = CAN\_CTRL1\_RWRNMSK\_MASK,  
 kFLEXCAN\_TxWarningInterruptEnable = CAN\_CTRL1\_TWRNMSK\_MASK,  
 kFLEXCAN\_WakeUpInterruptEnable = CAN\_MCR\_WAKMSK\_MASK }
- FlexCAN interrupt configuration structure, default settings all disabled.*
- enum \_flexcan\_flags {  
 kFLEXCAN\_FDErrorIntFlag = CAN\_ESR1\_ERRINT\_FAST\_MASK,  
 kFLEXCAN\_BusoffDoneIntFlag = CAN\_ESR1\_BOFFDONEINT\_MASK,  
 kFLEXCAN\_SynchFlag = CAN\_ESR1\_SYNCH\_MASK,  
 kFLEXCAN\_TxWarningIntFlag = CAN\_ESR1\_TWRNINT\_MASK,  
 kFLEXCAN\_RxWarningIntFlag = CAN\_ESR1\_RWRNINT\_MASK,  
 kFLEXCAN\_TxErrorWarningFlag = CAN\_ESR1\_TXWRN\_MASK,  
 kFLEXCAN\_RxErrorWarningFlag = CAN\_ESR1\_RXWRN\_MASK,  
 kFLEXCAN\_IdleFlag = CAN\_ESR1\_IDLE\_MASK,  
 kFLEXCAN\_FaultConfinementFlag = CAN\_ESR1\_FLTCNF\_MASK,  
 kFLEXCAN\_TransmittingFlag = CAN\_ESR1\_TX\_MASK,  
 kFLEXCAN\_ReceivingFlag = CAN\_ESR1\_RX\_MASK,  
 kFLEXCAN\_BusOffIntFlag = CAN\_ESR1\_BOFFINT\_MASK,  
 kFLEXCAN\_ErrorIntFlag = CAN\_ESR1\_ERRINT\_MASK,  
 kFLEXCAN\_WakeUpIntFlag = CAN\_ESR1\_WAKINT\_MASK }
- FlexCAN status flags.*
- enum \_flexcan\_error\_flags {  
 kFLEXCAN\_FDStuffingError = CAN\_ESR1\_STFERR\_FAST\_MASK,  
 kFLEXCAN\_FDFormError = CAN\_ESR1\_FRMERR\_FAST\_MASK,  
 kFLEXCAN\_FDCrcError = CAN\_ESR1\_CRCERR\_FAST\_MASK,  
 kFLEXCAN\_FDBit0Error = CAN\_ESR1\_BIT0ERR\_FAST\_MASK,  
 kFLEXCAN\_FDBit1Error = (int)CAN\_ESR1\_BIT1ERR\_FAST\_MASK,  
 kFLEXCAN\_OverrunError = CAN\_ESR1\_ERROVR\_MASK,  
 kFLEXCAN\_StuffingError = CAN\_ESR1\_STFERR\_MASK,  
 kFLEXCAN\_FormError = CAN\_ESR1\_FRMERR\_MASK,  
 kFLEXCAN\_CrcError = CAN\_ESR1\_CRCERR\_MASK,  
 kFLEXCAN\_AckError = CAN\_ESR1\_ACKERR\_MASK,  
 kFLEXCAN\_Bit0Error = CAN\_ESR1\_BIT0ERR\_MASK,  
 kFLEXCAN\_Bit1Error = CAN\_ESR1\_BIT1ERR\_MASK }
- FlexCAN error status flags.*
- enum \_flexcan\_rx\_fifo\_flags {  
 kFLEXCAN\_RxFifoOverflowFlag = CAN\_IFLAG1\_BUF7I\_MASK,  
 kFLEXCAN\_RxFifoWarningFlag = CAN\_IFLAG1\_BUF6I\_MASK,  
 kFLEXCAN\_RxFifoFrameAvlFlag = CAN\_IFLAG1\_BUF5I\_MASK }
- FlexCAN Rx FIFO status flags.*

## Driver version

- #define **FSL\_FLEXCAN\_DRIVER\_VERSION** (MAKE\_VERSION(2, 3, 1))  
*FlexCAN driver version 2.3.3.*

## Initialization and deinitialization

- uint32\_t **FLEXCANGetInstance** (CAN\_Type \*base)  
*Get the FlexCAN instance from peripheral base address.*
- bool **FLEXCAN\_CalculateImprovedTimingValues** (uint32\_t baudRate, uint32\_t sourceClock\_Hz, flexcan\_timing\_config\_t \*pconfig)  
*Calculates the improved timing values by specific baudrates for classical CAN.*
- void **FLEXCAN\_Init** (CAN\_Type \*base, const flexcan\_config\_t \*config, uint32\_t sourceClock\_Hz)  
*Initializes a FlexCAN instance.*
- bool **FLEXCAN\_FDCalculateImprovedTimingValues** (uint32\_t baudRate, uint32\_t baudRateFD, uint32\_t sourceClock\_Hz, flexcan\_timing\_config\_t \*pconfig)  
*Calculates the improved timing values by specific baudrates for CANFD.*
- void **FLEXCAN\_FDInit** (CAN\_Type \*base, const flexcan\_config\_t \*config, uint32\_t sourceClock\_Hz, flexcan\_mb\_size\_t dataSize, bool brs)  
*Initializes a FlexCAN instance.*
- void **FLEXCAN\_Deinit** (CAN\_Type \*base)  
*De-initializes a FlexCAN instance.*
- void **FLEXCAN\_GetDefaultConfig** (flexcan\_config\_t \*config)  
*Gets the default configuration structure.*

## Configuration.

- void **FLEXCAN\_SetTimingConfig** (CAN\_Type \*base, const flexcan\_timing\_config\_t \*config)  
*Sets the FlexCAN protocol timing characteristic.*
- void **FLEXCAN\_SetFDTimingConfig** (CAN\_Type \*base, const flexcan\_timing\_config\_t \*config)  
*Sets the FlexCAN FD protocol timing characteristic.*
- void **FLEXCAN\_SetRxMbGlobalMask** (CAN\_Type \*base, uint32\_t mask)  
*Sets the FlexCAN receive message buffer global mask.*
- void **FLEXCAN\_SetRxFifoGlobalMask** (CAN\_Type \*base, uint32\_t mask)  
*Sets the FlexCAN receive FIFO global mask.*
- void **FLEXCAN\_SetRxIndividualMask** (CAN\_Type \*base, uint8\_t maskIdx, uint32\_t mask)  
*Sets the FlexCAN receive individual mask.*
- void **FLEXCAN\_SetTxMbConfig** (CAN\_Type \*base, uint8\_t mbIdx, bool enable)  
*Configures a FlexCAN transmit message buffer.*
- void **FLEXCAN\_SetFDTxMbConfig** (CAN\_Type \*base, uint8\_t mbIdx, bool enable)  
*Configures a FlexCAN transmit message buffer.*
- void **FLEXCAN\_SetRxMbConfig** (CAN\_Type \*base, uint8\_t mbIdx, const flexcan\_rx\_mb\_config\_t \*config, bool enable)  
*Configures a FlexCAN Receive Message Buffer.*
- void **FLEXCAN\_SetFDRxMbConfig** (CAN\_Type \*base, uint8\_t mbIdx, const flexcan\_rx\_mb\_config\_t \*config, bool enable)  
*Configures a FlexCAN Receive Message Buffer.*

## FlexCAN Driver

- void [FLEXCAN\\_SetRx\\_fifoConfig](#) (CAN\_Type \*base, const [flexcan\\_rx\\_fifo\\_config\\_t](#) \*config, bool enable)  
*Configures the FlexCAN Rx FIFO.*

## Status

- static uint32\_t [FLEXCAN\\_GetStatusFlags](#) (CAN\_Type \*base)  
*Gets the FlexCAN module interrupt flags.*
- static void [FLEXCAN\\_ClearStatusFlags](#) (CAN\_Type \*base, uint32\_t mask)  
*Clears status flags with the provided mask.*
- static void [FLEXCAN\\_GetBusErrCount](#) (CAN\_Type \*base, uint8\_t \*txErrBuf, uint8\_t \*rxErrBuf)  
*Gets the FlexCAN Bus Error Counter value.*
- static uint64\_t [FLEXCAN\\_GetMbStatusFlags](#) (CAN\_Type \*base, uint64\_t mask)  
*Gets the FlexCAN Message Buffer interrupt flags.*
- static void [FLEXCAN\\_ClearMbStatusFlags](#) (CAN\_Type \*base, uint64\_t mask)  
*Clears the FlexCAN Message Buffer interrupt flags.*

## Interrupts

- static void [FLEXCAN\\_EnableInterrupts](#) (CAN\_Type \*base, uint32\_t mask)  
*Enables FlexCAN interrupts according to the provided mask.*
- static void [FLEXCAN\\_DisableInterrupts](#) (CAN\_Type \*base, uint32\_t mask)  
*Disables FlexCAN interrupts according to the provided mask.*
- static void [FLEXCAN\\_EnableMbInterrupts](#) (CAN\_Type \*base, uint64\_t mask)  
*Enables FlexCAN Message Buffer interrupts.*
- static void [FLEXCAN\\_DisableMbInterrupts](#) (CAN\_Type \*base, uint64\_t mask)  
*Disables FlexCAN Message Buffer interrupts.*

## DMA Control

- void [FLEXCAN\\_EnableRx\\_fifoDMA](#) (CAN\_Type \*base, bool enable)  
*Enables or disables the FlexCAN Rx FIFO DMA request.*
- static uint32\_t [FLEXCAN\\_GetRx\\_fifoHeadAddr](#) (CAN\_Type \*base)  
*Gets the Rx FIFO Head address.*

## Bus Operations

- static void [FLEXCAN\\_Enable](#) (CAN\_Type \*base, bool enable)  
*Enables or disables the FlexCAN module operation.*
- status\_t [FLEXCAN\\_WriteTxMb](#) (CAN\_Type \*base, uint8\_t mbIdx, const [flexcan\\_frame\\_t](#) \*txFrame)  
*Writes a FlexCAN Message to the Transmit Message Buffer.*
- status\_t [FLEXCAN\\_ReadRxMb](#) (CAN\_Type \*base, uint8\_t mbIdx, [flexcan\\_frame\\_t](#) \*rxFrame)  
*Reads a FlexCAN Message from Receive Message Buffer.*

- status\_t **FLEXCAN\_WriteFDTxMb** (CAN\_Type \*base, uint8\_t mbIdx, const flexcan\_fd\_frame\_t \*txFrame)
 

*Writes a FlexCAN FD Message to the Transmit Message Buffer.*
- status\_t **FLEXCAN\_ReadFDRxMb** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_fd\_frame\_t \*rxFrame)
 

*Reads a FlexCAN FD Message from Receive Message Buffer.*
- status\_t **FLEXCAN\_ReadRxFifo** (CAN\_Type \*base, flexcan\_frame\_t \*rxFrame)
 

*Reads a FlexCAN Message from Rx FIFO.*

## Transactional

- status\_t **FLEXCAN\_TransferFDSendBlocking** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_fd\_frame\_t \*txFrame)
 

*Performs a polling send transaction on the CAN bus.*
- status\_t **FLEXCAN\_TransferFDReceiveBlocking** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_fd\_frame\_t \*rxFrame)
 

*Performs a polling receive transaction on the CAN bus.*
- status\_t **FLEXCAN\_TransferFDSendNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*xfer)
 

*Sends a message using IRQ.*
- status\_t **FLEXCAN\_TransferFDReceiveNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*xfer)
 

*Receives a message using IRQ.*
- void **FLEXCAN\_TransferFDAbortSend** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)
 

*Aborts the interrupt driven message send process.*
- void **FLEXCAN\_TransferFDAbortReceive** (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)
 

*Aborts the interrupt driven message receive process.*
- status\_t **FLEXCAN\_TransferSendBlocking** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_frame\_t \*txFrame)
 

*Performs a polling send transaction on the CAN bus.*
- status\_t **FLEXCAN\_TransferReceiveBlocking** (CAN\_Type \*base, uint8\_t mbIdx, flexcan\_frame\_t \*rxFrame)
 

*Performs a polling receive transaction on the CAN bus.*
- status\_t **FLEXCAN\_TransferReceiveFifoBlocking** (CAN\_Type \*base, flexcan\_frame\_t \*rxFrame)
 

*Performs a polling receive transaction from Rx FIFO on the CAN bus.*
- void **FLEXCAN\_TransferCreateHandle** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_transfer\_callback\_t callback, void \*userData)
 

*Initializes the FlexCAN handle.*
- status\_t **FLEXCAN\_TransferSendNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*xfer)
 

*Sends a message using IRQ.*
- status\_t **FLEXCAN\_TransferReceiveNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*xfer)
 

*Receives a message using IRQ.*
- status\_t **FLEXCAN\_TransferReceiveFifoNonBlocking** (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_fifo\_transfer\_t \*xfer)

## FlexCAN Driver

*Receives a message from Rx FIFO using IRQ.*

- void [FLEXCAN\\_TransferAbortSend](#) (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)  
*Aborts the interrupt driven message send process.*
- void [FLEXCAN\\_TransferAbortReceive](#) (CAN\_Type \*base, flexcan\_handle\_t \*handle, uint8\_t mbIdx)  
*Aborts the interrupt driven message receive process.*
- void [FLEXCAN\\_TransferAbortReceiveFifo](#) (CAN\_Type \*base, flexcan\_handle\_t \*handle)  
*Aborts the interrupt driven message receive from Rx FIFO process.*
- void [FLEXCAN\\_TransferHandleIRQ](#) (CAN\_Type \*base, flexcan\_handle\_t \*handle)  
*FlexCAN IRQ handle function.*



## FlexCAN Driver

### 13.2.3 Data Structure Documentation

#### 13.2.3.1 struct flexcan\_frame\_t

##### 13.2.3.1.0.35 Field Documentation

13.2.3.1.0.35.1 uint32\_t flexcan\_frame\_t::timestamp

13.2.3.1.0.35.2 uint32\_t flexcan\_frame\_t::length

13.2.3.1.0.35.3 uint32\_t flexcan\_frame\_t::type

13.2.3.1.0.35.4 uint32\_t flexcan\_frame\_t::format

13.2.3.1.0.35.5 uint32\_t flexcan\_frame\_t::\_\_pad0\_\_

13.2.3.1.0.35.6 uint32\_t flexcan\_frame\_t::idhit

13.2.3.1.0.35.7 uint32\_t flexcan\_frame\_t::id

13.2.3.1.0.35.8 uint32\_t flexcan\_frame\_t::dataWord0

13.2.3.1.0.35.9 uint32\_t flexcan\_frame\_t::dataWord1

13.2.3.1.0.35.10 uint8\_t flexcan\_frame\_t::dataByte3

13.2.3.1.0.35.11 uint8\_t flexcan\_frame\_t::dataByte2

13.2.3.1.0.35.12 uint8\_t flexcan\_frame\_t::dataByte1

13.2.3.1.0.35.13 uint8\_t flexcan\_frame\_t::dataByte0

13.2.3.1.0.35.14 uint8\_t flexcan\_frame\_t::dataByte7

13.2.3.1.0.35.15 uint8\_t flexcan\_frame\_t::dataByte6

13.2.3.1.0.35.16 uint8\_t flexcan\_frame\_t::dataByte5

13.2.3.1.0.35.17 uint8\_t flexcan\_frame\_t::dataByte4

#### 13.2.3.2 struct flexcan\_fd\_frame\_t

##### 13.2.3.2.0.36 Field Documentation

13.2.3.2.0.36.1 uint32\_t flexcan\_fd\_frame\_t::timestamp

13.2.3.2.0.36.2 uint32\_t flexcan\_fd\_frame\_t::length

13.2.3.2.0.36.3 uint32\_t flexcan\_fd\_frame\_t::type

13.2.3.2.0.36.4 uint32\_t flexcan\_fd\_frame\_t::format

13.2.3.2.0.36.5 uint32\_t flexcan\_fd\_frame\_t::fdAPI Reference Manual

- *Clock Pre-scaler Division Factor.*
- `uint8_t rJumpwidth`  
*Re-sync Jump Width.*
- `uint8_t phaseSeg1`  
*Phase Segment 1.*
- `uint8_t phaseSeg2`  
*Phase Segment 2.*
- `uint8_t propSeg`  
*Propagation Segment.*
- `uint16_t fpreDivider`  
*Fast Clock Pre-scaler Division Factor.*
- `uint8_t frJumpwidth`  
*Fast Re-sync Jump Width.*
- `uint8_t fphaseSeg1`  
*Fast Phase Segment 1.*
- `uint8_t fphaseSeg2`  
*Fast Phase Segment 2.*
- `uint8_t fpropSeg`  
*Fast Propagation Segment.*

### 13.2.3.3.0.37 Field Documentation

**13.2.3.3.0.37.1 `uint16_t flexcan_timing_config_t::preDivider`**

**13.2.3.3.0.37.2 `uint8_t flexcan_timing_config_t::rJumpwidth`**

**13.2.3.3.0.37.3 `uint8_t flexcan_timing_config_t::phaseSeg1`**

**13.2.3.3.0.37.4 `uint8_t flexcan_timing_config_t::phaseSeg2`**

**13.2.3.3.0.37.5 `uint8_t flexcan_timing_config_t::propSeg`**

**13.2.3.3.0.37.6 `uint16_t flexcan_timing_config_t::fpreDivider`**

**13.2.3.3.0.37.7 `uint8_t flexcan_timing_config_t::frJumpwidth`**

**13.2.3.3.0.37.8 `uint8_t flexcan_timing_config_t::fphaseSeg1`**

**13.2.3.3.0.37.9 `uint8_t flexcan_timing_config_t::fphaseSeg2`**

**13.2.3.3.0.37.10 `uint8_t flexcan_timing_config_t::fpropSeg`**

### 13.2.3.4 `struct flexcan_config_t`

#### Data Fields

- `uint32_t baudRate`  
*FlexCAN baud rate in bps.*
- `uint32_t baudRateFD`  
*FlexCAN FD baud rate in bps.*
- `flexcan_clock_source_t clkSrc`

## FlexCAN Driver

- *Clock source for FlexCAN Protocol Engine.*
- **flexcan\_wake\_up\_source\_t wakeupSrc**
  - Wake up source selection.
- **uint8\_t maxMbNum**
  - The maximum number of Message Buffers used by user.
- **bool enableLoopBack**
  - Enable or Disable Loop Back Self Test Mode.
- **bool enableTimerSync**
  - Enable or Disable Timer Synchronization.
- **bool enableSelfWakeup**
  - Enable or Disable Self Wakeup Mode.
- **bool enableIndividMask**
  - Enable or Disable Rx Individual Mask.

### 13.2.3.4.0.38 Field Documentation

**13.2.3.4.0.38.1 uint32\_t flexcan\_config\_t::baudRate**

**13.2.3.4.0.38.2 uint32\_t flexcan\_config\_t::baudRateFD**

**13.2.3.4.0.38.3 flexcan\_clock\_source\_t flexcan\_config\_t::clkSrc**

**13.2.3.4.0.38.4 flexcan\_wake\_up\_source\_t flexcan\_config\_t::wakeupSrc**

**13.2.3.4.0.38.5 uint8\_t flexcan\_config\_t::maxMbNum**

**13.2.3.4.0.38.6 bool flexcan\_config\_t::enableLoopBack**

**13.2.3.4.0.38.7 bool flexcan\_config\_t::enableTimerSync**

**13.2.3.4.0.38.8 bool flexcan\_config\_t::enableSelfWakeup**

**13.2.3.4.0.38.9 bool flexcan\_config\_t::enableIndividMask**

### 13.2.3.5 struct flexcan\_rx\_mb\_config\_t

This structure is used as the parameter of [FLEXCAN\\_SetRxMbConfig\(\)](#) function. The [FLEXCAN\\_SetRxMbConfig\(\)](#) function is used to configure FlexCAN Receive Message Buffer. The function abort previous receiving process, clean the Message Buffer and activate the Rx Message Buffer using given Message Buffer setting.

### Data Fields

- **uint32\_t id**
  - CAN Message Buffer Frame Identifier, should be set using [FLEXCAN\\_ID\\_EXT\(\)](#) or [FLEXCAN\\_ID\\_STD\(\)](#) macro.*
- **flexcan\_frame\_format\_t format**
  - CAN Frame Identifier format(Standard or Extend).*
- **flexcan\_frame\_type\_t type**
  - CAN Frame Type(Data or Remote).*

**13.2.3.5.0.39 Field Documentation****13.2.3.5.0.39.1 uint32\_t flexcan\_rx\_mb\_config\_t::id****13.2.3.5.0.39.2 flexcan\_frame\_format\_t flexcan\_rx\_mb\_config\_t::format****13.2.3.5.0.39.3 flexcan\_frame\_type\_t flexcan\_rx\_mb\_config\_t::type****13.2.3.6 struct flexcan\_rx\_fifo\_config\_t****Data Fields**

- `uint32_t * idFilterTable`  
*Pointer to the FlexCAN Rx FIFO identifier filter table.*
- `uint8_t idFilterNum`  
*The quantity of filter elements.*
- `flexcan_rx_fifo_filter_type_t idFilterType`  
*The FlexCAN Rx FIFO Filter type.*
- `flexcan_rx_fifo_priority_t priority`  
*The FlexCAN Rx FIFO receive priority.*

**13.2.3.6.0.40 Field Documentation****13.2.3.6.0.40.1 uint32\_t\* flexcan\_rx\_fifo\_config\_t::idFilterTable****13.2.3.6.0.40.2 uint8\_t flexcan\_rx\_fifo\_config\_t::idFilterNum****13.2.3.6.0.40.3 flexcan\_rx\_fifo\_filter\_type\_t flexcan\_rx\_fifo\_config\_t::idFilterType****13.2.3.6.0.40.4 flexcan\_rx\_fifo\_priority\_t flexcan\_rx\_fifo\_config\_t::priority****13.2.3.7 struct flexcan\_mb\_transfer\_t****Data Fields**

- `flexcan_frame_t * frame`  
*The buffer of CAN Message to be transfer.*
- `uint8_t mbIdx`  
*The index of Message buffer used to transfer Message.*

**13.2.3.7.0.41 Field Documentation****13.2.3.7.0.41.1 flexcan\_frame\_t\* flexcan\_mb\_transfer\_t::frame****13.2.3.7.0.41.2 uint8\_t flexcan\_mb\_transfer\_t::mbIdx****13.2.3.8 struct flexcan\_fifo\_transfer\_t****Data Fields**

- `flexcan_frame_t * frame`

## FlexCAN Driver

*The buffer of CAN Message to be received from Rx FIFO.*

### 13.2.3.8.0.42 Field Documentation

#### 13.2.3.8.0.42.1 `flexcan_frame_t* flexcan_fifo_transfer_t::frame`

#### 13.2.3.9 `struct _flexcan_handle`

FlexCAN handle structure definition.

### Data Fields

- `flexcan_transfer_callback_t callback`  
*Callback function.*
- `void *userData`  
*FlexCAN callback function parameter.*
- `flexcan_fd_frame_t *volatile mbFDFrameBuf [CAN_WORD1_COUNT]`  
*The buffer for received data from Message Buffers.*
- `flexcan_frame_t *volatile rxFifoFrameBuf`  
*The buffer for received data from Rx FIFO.*
- `volatile uint8_t mbState [CAN_WORD1_COUNT]`  
*Message Buffer transfer state.*
- `volatile uint8_t rxFifoState`  
*Rx FIFO transfer state.*

### 13.2.3.9.0.43 Field Documentation

#### 13.2.3.9.0.43.1 `flexcan_transfer_callback_t flexcan_handle_t::callback`

#### 13.2.3.9.0.43.2 `void* flexcan_handle_t::userData`

#### 13.2.3.9.0.43.3 `flexcan_fd_frame_t* volatile flexcan_handle_t::mbFDFrameBuf[CAN_WORD1_CO- UNT]`

#### 13.2.3.9.0.43.4 `flexcan_frame_t* volatile flexcan_handle_t::rxFifoFrameBuf`

#### 13.2.3.9.0.43.5 `volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]`

#### 13.2.3.9.0.43.6 `volatile uint8_t flexcan_handle_t::rxFifoState`

### 13.2.4 Macro Definition Documentation

#### 13.2.4.1 `#define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))`

#### 13.2.4.2 `#define FLEXCAN_ID_STD( id ) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT) & CAN_ID_STD_MASK)`

Standard Frame ID helper macro.

**13.2.4.3 #define FLEXCAN\_ID\_EXT( *id* )****Value:**

```
((uint32_t)((uint32_t)(id) << CAN_ID_EXT_SHIFT)) & \
(CAN_ID_EXT_MASK | CAN_ID_STD_MASK)
```

**13.2.4.4 #define FLEXCAN\_RX\_MB\_STD\_MASK( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
FLEXCAN_ID_STD(id))
```

Standard Rx Message Buffer Mask helper macro.

**13.2.4.5 #define FLEXCAN\_RX\_MB\_EXT\_MASK( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
FLEXCAN_ID_EXT(id))
```

**13.2.4.6 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
(FLEXCAN_ID_STD(id) << 1))
```

Standard Rx FIFO Mask helper macro Type A helper macro.

**13.2.4.7 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
(((uint32_t)(id)&0x7FF) << 19))
```

# FlexCAN Driver

**13.2.4.8 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_B\_LOW( id, rtr, ide )**

## Value:

```
((uint32_t) ((uint32_t) (rtr) << 15) | (uint32_t) ((uint32_t) (ide) << 14)) | \
 (((uint32_t) (id)&0x7FF) << 3))
```

13.2.4.9 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH( id ) (((uint32\_t)(id)&0x7F8) << 21)

13.2.4.10 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH( id ) (((uint32) t)(id)&0x7F8) << 13)

```
13.2.4.11 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(id) (((uint32_t)(id)&0x7F8) << 5)
```

13.2.4.12 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW( id ) (((uint32\_t)(id)&0x7F8) >> 3)

13.2.4.13 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A( *id*, *rtr*, *ide* )

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
 (FLEXCAN_ID_EXT(id) << 1))
```

13.2.4.14 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )

**Value:**

```
(\((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
(\u03c0FLEXCAN_ID_EXT(id) & 0x1FFF8000)
<< 1))
```

13.2.4.15 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW( *id*, *rrt*, *ide* )

**Value:**

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
 ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) >>
 15))
```

**13.2.4.16 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH( *id* )**  
 $((\text{FLEXCAN\_ID\_EXT}(\text{id}) \& 0x1FE00000) << 3)$

**13.2.4.17 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH( *id* )**

**Value:**

```
((\text{FLEXCAN_ID_EXT}(\text{id}) \& 0x1FE00000) >> 5) \
```

**13.2.4.18 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW( *id* )**

**Value:**

```
((\text{FLEXCAN_ID_EXT}(\text{id}) \& 0x1FE00000) >> 13) \
```

**13.2.4.19 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW( *id* )**  
 $((\text{FLEXCAN\_ID\_EXT}(\text{id}) \& 0x1FE00000) >> 21)$

**13.2.4.20 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_A( *id*, *rtr*, *ide* )**  
**FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_A(*id*, *rtr*, *ide*)**

Standard Rx FIFO Filter helper macro Type A helper macro.

**13.2.4.21 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )**

**Value:**

```
\text{FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH}(\text{id}, \text{rtr}, \text{ide}) \
```

**13.2.4.22 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )**

**Value:**

```
\text{FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW}(\text{id}, \text{rtr}, \text{ide}) \
```

## FlexCAN Driver

**13.2.4.23 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(\
 id)
```

**13.2.4.24 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(\
 id)
```

**13.2.4.25 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_LOW( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(\
 id)
```

**13.2.4.26 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_LOW( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(\
 id)
```

**13.2.4.27 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_A( *id*, *rtr*, *ide* ) FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A(*id*, *rtr*, *ide*)**

**13.2.4.28 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(\
 id, rtr, ide)
```

**13.2.4.29 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_LOW( id, rtr, ide )****Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(
 id, rtr, ide) \
```

**13.2.4.30 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_HIGH( id )****Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(
 id) \
```

**13.2.4.31 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_HIGH( id )****Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(
 id) \
```

**13.2.4.32 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_LOW( id )****Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(
 id) \
```

**13.2.4.33 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_LOW( id ) FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW(id)****13.2.5 Typedef Documentation****13.2.5.1 typedef void(\* flexcan\_transfer\_callback\_t)(CAN\_Type \*base, flexcan\_handle\_t \*handle, status\_t status, uint32\_t result, void \*userData)**

The FlexCAN transfer callback returns a value from the underlying layer. If the status equals to kStatus\_FLEXCAN\_ErrorStatus, the result parameter is the Content of FlexCAN status register which can be used to get the working status(or error status) of FlexCAN module. If the status equals to other FlexCAN Message Buffer transfer status, the result is the index of Message Buffer that generate transfer event. If the status equals to other FlexCAN Message Buffer transfer status, the result is meaningless and should be Ignored.

### 13.2.6 Enumeration Type Documentation

#### 13.2.6.1 enum \_flexcan\_status

Enumerator

*kStatus\_FLEXCAN\_TxBusy* Tx Message Buffer is Busy.

*kStatus\_FLEXCAN\_TxIdle* Tx Message Buffer is Idle.

*kStatus\_FLEXCAN\_TxSwitchToRx* Remote Message is send out and Message buffer changed to Receive one.

*kStatus\_FLEXCAN\_RxBusy* Rx Message Buffer is Busy.

*kStatus\_FLEXCAN\_RxIdle* Rx Message Buffer is Idle.

*kStatus\_FLEXCAN\_RxOverflow* Rx Message Buffer is Overflowed.

*kStatus\_FLEXCAN\_RxFifoBusy* Rx Message FIFO is Busy.

*kStatus\_FLEXCAN\_RxFifoIdle* Rx Message FIFO is Idle.

*kStatus\_FLEXCAN\_RxFifoOverflow* Rx Message FIFO is overflowed.

*kStatus\_FLEXCAN\_RxFifoWarning* Rx Message FIFO is almost overflowed.

*kStatus\_FLEXCAN\_ErrorStatus* FlexCAN Module Error and Status.

*kStatus\_FLEXCAN\_WakeUp* FlexCAN is waken up from STOP mode.

*kStatus\_FLEXCAN\_UnHandled* UnHadled Interrupt asserted.

#### 13.2.6.2 enum flexcan\_frame\_format\_t

Enumerator

*kFLEXCAN\_FrameFormatStandard* Standard frame format attribute.

*kFLEXCAN\_FrameFormatExtend* Extend frame format attribute.

#### 13.2.6.3 enum flexcan\_frame\_type\_t

Enumerator

*kFLEXCAN\_FrameTypeData* Data frame type attribute.

*kFLEXCAN\_FrameTypeRemote* Remote frame type attribute.

#### 13.2.6.4 enum flexcan\_clock\_source\_t

Enumerator

*kFLEXCAN\_ClkSrcOsc* FlexCAN Protocol Engine clock from Oscillator.

*kFLEXCAN\_ClkSrcPeri* FlexCAN Protocol Engine clock from Peripheral Clock.

### 13.2.6.5 enum flexcan\_wake\_up\_source\_t

Enumerator

***kFLEXCAN\_WakeupSrcUnfiltered*** FlexCAN uses unfiltered Rx input to detect edge.

***kFLEXCAN\_WakeupSrcFiltered*** FlexCAN uses filtered Rx input to detect edge.

### 13.2.6.6 enum flexcan\_rx\_fifo\_filter\_type\_t

Enumerator

***kFLEXCAN\_RxFifoFilterTypeA*** One full ID (standard and extended) per ID Filter element.

***kFLEXCAN\_RxFifoFilterTypeB*** Two full standard IDs or two partial 14-bit ID slices per ID Filter Table element.

***kFLEXCAN\_RxFifoFilterTypeC*** Four partial 8-bit Standard or extended ID slices per ID Filter Table element.

***kFLEXCAN\_RxFifoFilterTypeD*** All frames rejected.

### 13.2.6.7 enum flexcan\_mb\_size\_t

Enumerator

***kFLEXCAN\_8BperMB*** Selects 8 bytes per Message Buffer.

***kFLEXCAN\_16BperMB*** Selects 16 bytes per Message Buffer.

***kFLEXCAN\_32BperMB*** Selects 32 bytes per Message Buffer.

***kFLEXCAN\_64BperMB*** Selects 64 bytes per Message Buffer.

### 13.2.6.8 enum flexcan\_rx\_fifo\_priority\_t

The matching process starts from the Rx MB(or Rx FIFO) with higher priority. If no MB(or Rx FIFO) is satisfied, the matching process goes on with the Rx FIFO(or Rx MB) with lower priority.

Enumerator

***kFLEXCAN\_RxFifoPrioLow*** Matching process start from Rx Message Buffer first.

***kFLEXCAN\_RxFifoPrioHigh*** Matching process start from Rx FIFO first.

### 13.2.6.9 enum \_flexcan\_interrupt\_enable

This structure contains the settings for all of the FlexCAN Module interrupt configurations. Note: FlexCAN Message Buffers and Rx FIFO have their own interrupts.

## FlexCAN Driver

Enumerator

***kFLEXCAN\_BusOffInterruptEnable*** Bus Off interrupt.  
***kFLEXCAN\_ErrorInterruptEnable*** Error interrupt.  
***kFLEXCAN\_RxWarningInterruptEnable*** Rx Warning interrupt.  
***kFLEXCAN\_TxWarningInterruptEnable*** Tx Warning interrupt.  
***kFLEXCAN\_WakeUpInterruptEnable*** Wake Up interrupt.

### 13.2.6.10 enum \_flexcan\_flags

This provides constants for the FlexCAN status flags for use in the FlexCAN functions. Note: The CPU read action clears FLEXCAN\_ErrorFlag, therefore user need to read FLEXCAN\_ErrorFlag and distinguish which error is occur using [\\_flexcan\\_error\\_flags](#) enumerations.

Enumerator

***kFLEXCAN\_FDErrorIntFlag*** Error Overrun Status.  
***kFLEXCAN\_BusoffDoneIntFlag*** Error Overrun Status.  
***kFLEXCAN\_SynchFlag*** CAN Synchronization Status.  
***kFLEXCAN\_TxWarningIntFlag*** Tx Warning Interrupt Flag.  
***kFLEXCAN\_RxWarningIntFlag*** Rx Warning Interrupt Flag.  
***kFLEXCAN\_TxErrorWarningFlag*** Tx Error Warning Status.  
***kFLEXCAN\_RxErrorWarningFlag*** Rx Error Warning Status.  
***kFLEXCAN\_IdleFlag*** CAN IDLE Status Flag.  
***kFLEXCAN\_FaultConfinementFlag*** Fault Confinement State Flag.  
***kFLEXCAN\_TransmittingFlag*** FlexCAN In Transmission Status.  
***kFLEXCAN\_ReceivingFlag*** FlexCAN In Reception Status.  
***kFLEXCAN\_BusOffIntFlag*** Bus Off Interrupt Flag.  
***kFLEXCAN\_ErrorIntFlag*** Error Interrupt Flag.  
***kFLEXCAN\_WakeUpIntFlag*** Wake-Up Interrupt Flag.

### 13.2.6.11 enum \_flexcan\_error\_flags

The FlexCAN Error Status enumerations is used to report current error of the FlexCAN bus. This enumerations should be used with KFLEXCAN\_ErrorFlag in [\\_flexcan\\_flags](#) enumerations to determine which error is generated.

Enumerator

***kFLEXCAN\_FDStuffingError*** Stuffing Error.  
***kFLEXCAN\_FDFormError*** Form Error.  
***kFLEXCAN\_FDCrcError*** Cyclic Redundancy Check Error.  
***kFLEXCAN\_FDBit0Error*** Unable to send dominant bit.  
***kFLEXCAN\_FDBit1Error*** Unable to send recessive bit.  
***kFLEXCAN\_OverrunError*** Error Overrun Status.

- kFLEXCAN\_StuffingError*** Stuffing Error.
- kFLEXCAN\_FormError*** Form Error.
- kFLEXCAN\_CrcError*** Cyclic Redundancy Check Error.
- kFLEXCAN\_AckError*** Received no ACK on transmission.
- kFLEXCAN\_Bit0Error*** Unable to send dominant bit.
- kFLEXCAN\_Bit1Error*** Unable to send recessive bit.

### 13.2.6.12 enum \_flexcan\_rx\_fifo\_flags

The FlexCAN Rx FIFO Status enumerations are used to determine the status of the Rx FIFO. Because Rx FIFO occupy the MB0 ~ MB7 (Rx Fifo filter also occupies more Message Buffer space), Rx FIFO status flags are mapped to the corresponding Message Buffer status flags.

Enumerator

- kFLEXCAN\_RxFifoOverflowFlag*** Rx FIFO overflow flag.
- kFLEXCAN\_RxFifoWarningFlag*** Rx FIFO almost full flag.
- kFLEXCAN\_RxFifoFrameAvlFlag*** Frames available in Rx FIFO flag.

### 13.2.7 Function Documentation

#### 13.2.7.1 uint32\_t FLEXCAN\_GetInstance ( CAN\_Type \* base )

Parameters

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

Returns

FlexCAN instance.

#### 13.2.7.2 bool FLEXCAN\_CalculateImprovedTimingValues ( uint32\_t baudRate, uint32\_t sourceClock\_Hz, flexcan\_timing\_config\_t \* pconfig )

Parameters

## FlexCAN Driver

|                       |                                                                        |
|-----------------------|------------------------------------------------------------------------|
| <i>baudRate</i>       | The classical CAN speed in bps defined by user                         |
| <i>sourceClock_Hz</i> | The Source clock data speed in bps. Zero to disable baudrate switching |
| <i>pconfig</i>        | Pointer to the FlexCAN timing configuration structure.                 |

Returns

TRUE if timing configuration found, FALSE if failed to find configuration

### 13.2.7.3 void FLEXCAN\_Init ( CAN\_Type \* *base*, const flexcan\_config\_t \* *config*, uint32\_t *sourceClock\_Hz* )

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the `flexcan_config_t` parameters and how to call the `FLEXCAN_Init` function by passing in these parameters.

```
* flexcan_config_t flexcanConfig;
* flexcanConfig.clkSrc = kFLEXCAN_ClkSrcOsc;
* flexcanConfig.baudRate = 1000000U;
* flexcanConfig.maxMbNum = 16;
* flexcanConfig.enableLoopBack = false;
* flexcanConfig.enableSelfWakeUp = false;
* flexcanConfig.enableIndividMask = false;
* flexcanConfig.enableDoze = false;
* flexcanConfig.timingConfig = timingConfig;
* FLEXCAN_Init(CAN0, &flexcanConfig, 8000000UL);
*
```

Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                      |
| <i>config</i>         | Pointer to the user-defined configuration structure.  |
| <i>sourceClock_Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz. |

### 13.2.7.4 bool FLEXCAN\_FDCalculateImprovedTimingValues ( uint32\_t *baudRate*, uint32\_t *baudRateFD*, uint32\_t *sourceClock\_Hz*, flexcan\_timing\_config\_t \* *pconfig* )

## Parameters

|                       |                                                                        |
|-----------------------|------------------------------------------------------------------------|
| <i>baudRate</i>       | The CANFD bus control speed in bps defined by user                     |
| <i>baudRateFD</i>     | The CANFD bus data speed in bps defined by user                        |
| <i>sourceClock_Hz</i> | The Source clock data speed in bps. Zero to disable baudrate switching |
| <i>pconfig</i>        | Pointer to the FlexCAN timing configuration structure.                 |

## Returns

TRUE if timing configuration found, FALSE if failed to find configuration

### 13.2.7.5 void FLEXCAN\_FDInit ( CAN\_Type \* *base*, const flexcan\_config\_t \* *config*, uint32\_t *sourceClock\_Hz*, flexcan\_mb\_size\_t *dataSize*, bool *brs* )

This function initializes the FlexCAN module with user-defined settings. This example shows how to set up the `flexcan_config_t` parameters and how to call the `FLEXCAN_FDInit` function by passing in these parameters.

```
* flexcan_config_t flexcanConfig;
* flexcanConfig.clkSrc = kFLEXCAN_ClkSrcOsc;
* flexcanConfig.baudRate = 1000000U;
* flexcanConfig.baudRateFD = 2000000U;
* flexcanConfig.maxMbNum = 16;
* flexcanConfig.enableLoopBack = false;
* flexcanConfig.enableSelfWakeUp = false;
* flexcanConfig.enableIndividMask = false;
* flexcanConfig.enableDoze = false;
* flexcanConfig.timingConfig = timingConfig;
* FLEXCAN_FDInit(CAN0, &flexcanConfig, 8000000UL,
* kFLEXCAN_16BperMB, false);
*
```

## Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                      |
| <i>config</i>         | Pointer to the user-defined configuration structure.  |
| <i>sourceClock_Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz. |

## FlexCAN Driver

|                 |                                          |
|-----------------|------------------------------------------|
| <i>dataSize</i> | FlexCAN FD frame payload size.           |
| <i>brs</i>      | If bitrate switch is enabled in FD mode. |

### 13.2.7.6 void FLEXCAN\_Deinit ( CAN\_Type \* *base* )

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

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

### 13.2.7.7 void FLEXCAN\_GetDefaultConfig ( flexcan\_config\_t \* *config* )

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. `flexcanConfig->clkSrc = kFLEXCAN_ClkSrcOsc;` `flexcanConfig->baudRate = 1000000U;` `flexcanConfig->baudRateFD = 2000000U;` `flexcanConfig->maxMbNum = 16;` `flexcanConfig->enableLoopBack = false;` `flexcanConfig->enableSelfWakeup = false;` `flexcanConfig->enableIndividMask = false;` `flexcanConfig->enableDoze = false;` `flexcanConfig.timingConfig = timingConfig;`

Parameters

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

### 13.2.7.8 void FLEXCAN\_SetTimingConfig ( CAN\_Type \* *base*, const flexcan\_timing\_config\_t \* *config* )

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN\\_Init\(\)](#) and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling [FLEXCAN\\_SetTimingConfig\(\)](#) overrides the baud rate set in [FLEXCAN\\_Init\(\)](#).

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.               |
| <i>config</i> | Pointer to the timing configuration structure. |

### 13.2.7.9 void FLEXCAN\_SetFDTimingConfig ( CAN\_Type \* *base*, const flexcan\_timing\_config\_t \* *config* )

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN\\_Init\(\)](#) and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling [FLEXCAN\\_SetFDTimingConfig\(\)](#) overrides the baud rate set in [FLEXCAN\\_Init\(\)](#).

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.               |
| <i>config</i> | Pointer to the timing configuration structure. |

### 13.2.7.10 void FLEXCAN\_SetRxMbGlobalMask ( CAN\_Type \* *base*, uint32\_t *mask* )

This function sets the global mask for the FlexCAN message buffer in a matching process. The configuration is only effective when the Rx individual mask is disabled in the [FLEXCAN\\_Init\(\)](#).

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.     |
| <i>mask</i> | Rx Message Buffer Global Mask value. |

### 13.2.7.11 void FLEXCAN\_SetRxFifoGlobalMask ( CAN\_Type \* *base*, uint32\_t *mask* )

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FlexCAN peripheral base address. |
| <i>mask</i> | Rx Fifo Global Mask value.       |

### 13.2.7.12 void FLEXCAN\_SetRxIndividualMask ( CAN\_Type \* *base*, uint8\_t *maskIdx*, uint32\_t *mask* )

This function sets the individual mask for the FlexCAN matching process. The configuration is only effective when the Rx individual mask is enabled in the [FLEXCAN\\_Init\(\)](#). If the Rx FIFO is disabled, the individual mask is applied to the corresponding Message Buffer. If the Rx FIFO is enabled, the individual mask for Rx FIFO occupied Message Buffer is applied to the Rx Filter with the same index. Note that only the first 32 individual masks can be used as the Rx FIFO filter mask.

## FlexCAN Driver

Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>base</i>    | FlexCAN peripheral base address. |
| <i>maskIdx</i> | The Index of individual Mask.    |
| <i>mask</i>    | Rx Individual Mask value.        |

### 13.2.7.13 void FLEXCAN\_SetTxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, bool *enable* )

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

|               |                                                                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                       |
| <i>enable</i> | Enable/disable Tx Message Buffer. <ul style="list-style-type: none"><li>• true: Enable Tx Message Buffer.</li><li>• false: Disable Tx Message Buffer.</li></ul> |

### 13.2.7.14 void FLEXCAN\_SetFDTxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, bool *enable* )

This function aborts the previous transmission, cleans the Message Buffer, and configures it as a Transmit Message Buffer.

Parameters

|               |                                                                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                       |
| <i>enable</i> | Enable/disable Tx Message Buffer. <ul style="list-style-type: none"><li>• true: Enable Tx Message Buffer.</li><li>• false: Disable Tx Message Buffer.</li></ul> |

### 13.2.7.15 void FLEXCAN\_SetRxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_rx\_mb\_config\_t \* *config*, bool *enable* )

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

|               |                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                   |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                          |
| <i>config</i> | Pointer to the FlexCAN Message Buffer configuration structure.                                                                                                     |
| <i>enable</i> | Enable/disable Rx Message Buffer. <ul style="list-style-type: none"> <li>• true: Enable Rx Message Buffer.</li> <li>• false: Disable Rx Message Buffer.</li> </ul> |

### 13.2.7.16 void FLEXCAN\_SetFDRxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_rx\_mb\_config\_t \* *config*, bool *enable* )

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

|               |                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                                                   |
| <i>mbIdx</i>  | The Message Buffer index.                                                                                                                                          |
| <i>config</i> | Pointer to the FlexCAN Message Buffer configuration structure.                                                                                                     |
| <i>enable</i> | Enable/disable Rx Message Buffer. <ul style="list-style-type: none"> <li>• true: Enable Rx Message Buffer.</li> <li>• false: Disable Rx Message Buffer.</li> </ul> |

### 13.2.7.17 void FLEXCAN\_SetRx\_fifoConfig ( CAN\_Type \* *base*, const flexcan\_rx\_fifo\_config\_t \* *config*, bool *enable* )

This function configures the Rx FIFO with given Rx FIFO configuration.

Parameters

|               |                                                                                                                                      |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                                                                     |
| <i>config</i> | Pointer to the FlexCAN Rx FIFO configuration structure.                                                                              |
| <i>enable</i> | Enable/disable Rx FIFO. <ul style="list-style-type: none"> <li>• true: Enable Rx FIFO.</li> <li>• false: Disable Rx FIFO.</li> </ul> |

## FlexCAN Driver

### 13.2.7.18 static uint32\_t FLEXCAN\_GetStatusFlags ( CAN\_Type \* *base* ) [inline], [static]

This function gets all FlexCAN status flags. The flags are returned as the logical OR value of the enumerators [\\_flexcan\\_flags](#). To check the specific status, compare the return value with enumerators in [\\_flexcan\\_flags](#).

Parameters

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

Returns

FlexCAN status flags which are ORed by the enumerators in the [\\_flexcan\\_flags](#).

### 13.2.7.19 static void FLEXCAN\_ClearStatusFlags ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function clears the FlexCAN status flags with a provided mask. An automatically cleared flag can't be cleared by this function.

Parameters

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                           |
| <i>mask</i> | The status flags to be cleared, it is logical OR value of <a href="#">_flexcan_flags</a> . |

### 13.2.7.20 static void FLEXCAN\_GetBusErrCount ( CAN\_Type \* *base*, uint8\_t \* *txErrBuf*, uint8\_t \* *rxErrBuf* ) [inline], [static]

This function gets the FlexCAN Bus Error Counter value for both Tx and Rx direction. These values may be needed in the upper layer error handling.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.        |
| <i>txErrBuf</i> | Buffer to store Tx Error Counter value. |
| <i>rxErrBuf</i> | Buffer to store Rx Error Counter value. |

### 13.2.7.21 static uint64\_t FLEXCAN\_GetMbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

This function gets the interrupt flags of a given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

Returns

The status of given Message Buffers.

### 13.2.7.22 static void FLEXCAN\_ClearMbStatusFlags ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]

This function clears the interrupt flags of a given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

### 13.2.7.23 static void FLEXCAN\_EnableInterrupts ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function enables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see [\\_flexcan\\_interrupt\\_enable](#).

Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

### 13.2.7.24 static void FLEXCAN\_DisableInterrupts ( CAN\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function disables the FlexCAN interrupts according to the provided mask. The mask is a logical OR of enumeration members, see [\\_flexcan\\_interrupt\\_enable](#).

## FlexCAN Driver

Parameters

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

**13.2.7.25 static void FLEXCAN\_EnableMbInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

This function enables the interrupts of given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

**13.2.7.26 static void FLEXCAN\_DisableMbInterrupts ( CAN\_Type \* *base*, uint64\_t *mask* ) [inline], [static]**

This function disables the interrupts of given Message Buffers.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

**13.2.7.27 void FLEXCAN\_EnableRxFifoDMA ( CAN\_Type \* *base*, bool *enable* )**

This function enables or disables the DMA feature of FlexCAN build-in Rx FIFO.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>enable</i> | true to enable, false to disable. |

**13.2.7.28 static uint32\_t FLEXCAN\_GetRxFifoHeadAddr ( CAN\_Type \* *base* ) [inline], [static]**

This function returns the FlexCAN Rx FIFO Head address, which is mainly used for the DMA/eDMA use case.

Parameters

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

Returns

FlexCAN Rx FIFO Head address.

### 13.2.7.29 static void FLEXCAN\_Enable ( CAN\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the FlexCAN module.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN base pointer.             |
| <i>enable</i> | true to enable, false to disable. |

### 13.2.7.30 status\_t FLEXCAN\_WriteTxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_frame\_t \* *txFrame* )

This function writes a CAN Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN Message transmit. After that the function returns immediately.

Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.         |
| <i>mbIdx</i>   | The FlexCAN Message Buffer index.        |
| <i>txFrame</i> | Pointer to CAN message frame to be sent. |

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

### 13.2.7.31 status\_t FLEXCAN\_ReadRxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *rxFrame* )

This function reads a CAN message from a specified Receive Message Buffer. The function fills a receive CAN message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

## FlexCAN Driver

Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                      |
| <i>mbIdx</i>   | The FlexCAN Message Buffer index.                     |
| <i>rxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

### 13.2.7.32 **status\_t FLEXCAN\_WriteFDTxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_fd\_frame\_t \* *txFrame* )**

This function writes a CAN FD Message to the specified Transmit Message Buffer and changes the Message Buffer state to start CAN FD Message transmit. After that the function returns immediately.

Parameters

|                |                                             |
|----------------|---------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.            |
| <i>mbIdx</i>   | The FlexCAN FD Message Buffer index.        |
| <i>txFrame</i> | Pointer to CAN FD message frame to be sent. |

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

### 13.2.7.33 **status\_t FLEXCAN\_ReadFDRxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_fd\_frame\_t \* *rxFrame* )**

This function reads a CAN FD message from a specified Receive Message Buffer. The function fills a receive CAN FD message frame structure with just received data and activates the Message Buffer again. The function returns immediately.

Parameters

|                |                                                          |
|----------------|----------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                         |
| <i>mbIdx</i>   | The FlexCAN FD Message Buffer index.                     |
| <i>rxFrame</i> | Pointer to CAN FD message frame structure for reception. |

Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

### 13.2.7.34 **status\_t FLEXCAN\_ReadRxFifo ( CAN\_Type \* *base*, flexcan\_frame\_t \* *rxFrame* )**

This function reads a CAN message from the FlexCAN build-in Rx FIFO.

Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base address.                      |
| <i>rxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

### 13.2.7.35 **status\_t FLEXCAN\_TransferFDSendBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_fd\_frame\_t \* *txFrame* )**

Note that a transfer handle does not need to be created before calling this API.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FlexCAN peripheral base pointer. |
|-------------|----------------------------------|

## FlexCAN Driver

|                |                                             |
|----------------|---------------------------------------------|
| <i>mbIdx</i>   | The FlexCAN FD Message Buffer index.        |
| <i>txFrame</i> | Pointer to CAN FD message frame to be sent. |

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

### 13.2.7.36 status\_t FLEXCAN\_TransferFDReceiveBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_fd\_frame\_t \* *rxFrame* )

Note that a transfer handle does not need to be created before calling this API.

Parameters

|                |                                                          |
|----------------|----------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base pointer.                         |
| <i>mbIdx</i>   | The FlexCAN FD Message Buffer index.                     |
| <i>rxFrame</i> | Pointer to CAN FD message frame structure for reception. |

Return values

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_Success</i>             | - Rx Message Buffer is full and has been read successfully.               |
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

### 13.2.7.37 status\_t FLEXCAN\_TransferFDSendNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *xfer* )

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                              |
| <i>handle</i> | FlexCAN handle pointer.                                                                       |
| <i>xfer</i>   | FlexCAN FD Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                       |
|--------------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>         | Start Tx Message Buffer sending process successfully. |
| <i>kStatus_Fail</i>            | Write Tx Message Buffer failed.                       |
| <i>kStatus_FLEXCAN_Tx-Busy</i> | Tx Message Buffer is in use.                          |

### 13.2.7.38 **status\_t FLEXCAN\_TransferFDReceiveNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *xfer* )**

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                              |
| <i>handle</i> | FlexCAN handle pointer.                                                                       |
| <i>xfer</i>   | FlexCAN FD Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                           |
|--------------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>         | - Start Rx Message Buffer receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-Busy</i> | - Rx Message Buffer is in use.                            |

### 13.2.7.39 **void FLEXCAN\_TransferFDAbortSend ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )**

This function aborts the interrupt driven message send process.

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.     |
| <i>handle</i> | FlexCAN handle pointer.              |
| <i>mbIdx</i>  | The FlexCAN FD Message Buffer index. |

### 13.2.7.40 **void FLEXCAN\_TransferFDAbortReceive ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )**

This function aborts the interrupt driven message receive process.

## FlexCAN Driver

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.     |
| <i>handle</i> | FlexCAN handle pointer.              |
| <i>mbIdx</i>  | The FlexCAN FD Message Buffer index. |

### 13.2.7.41 status\_t FLEXCAN\_TransferSendBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *txFrame* )

Note that a transfer handle does not need to be created before calling this API.

Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base pointer.         |
| <i>mbIdx</i>   | The FlexCAN Message Buffer index.        |
| <i>txFrame</i> | Pointer to CAN message frame to be sent. |

Return values

|                        |                                          |
|------------------------|------------------------------------------|
| <i>kStatus_Success</i> | - Write Tx Message Buffer Successfully.  |
| <i>kStatus_Fail</i>    | - Tx Message Buffer is currently in use. |

### 13.2.7.42 status\_t FLEXCAN\_TransferReceiveBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *rxFrame* )

Note that a transfer handle does not need to be created before calling this API.

Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base pointer.                      |
| <i>mbIdx</i>   | The FlexCAN Message Buffer index.                     |
| <i>rxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                        |                                                             |
|------------------------|-------------------------------------------------------------|
| <i>kStatus_Success</i> | - Rx Message Buffer is full and has been read successfully. |
|------------------------|-------------------------------------------------------------|

|                                    |                                                                           |
|------------------------------------|---------------------------------------------------------------------------|
| <i>kStatus_FLEXCAN_Rx-Overflow</i> | - Rx Message Buffer is already overflowed and has been read successfully. |
| <i>kStatus_Fail</i>                | - Rx Message Buffer is empty.                                             |

### 13.2.7.43 status\_t FLEXCAN\_TransferReceiveFifoBlocking ( CAN\_Type \* *base*, flexcan\_frame\_t \* *rxFrame* )

Note that a transfer handle does not need to be created before calling this API.

Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | FlexCAN peripheral base pointer.                      |
| <i>rxFrame</i> | Pointer to CAN message frame structure for reception. |

Return values

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>kStatus_Success</i> | - Read Message from Rx FIFO successfully. |
| <i>kStatus_Fail</i>    | - Rx FIFO is not enabled.                 |

### 13.2.7.44 void FLEXCAN\_TransferCreateHandle ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the FlexCAN handle, which can be used for other FlexCAN transactional APIs. Usually, for a specified FlexCAN instance, call this API once to get the initialized handle.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.        |
| <i>handle</i>   | FlexCAN handle pointer.                 |
| <i>callback</i> | The callback function.                  |
| <i>userData</i> | The parameter of the callback function. |

### 13.2.7.45 status\_t FLEXCAN\_TransferSendNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *xfer* )

This function sends a message using IRQ. This is a non-blocking function, which returns right away. When messages have been sent out, the send callback function is called.

## FlexCAN Driver

Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                           |
| <i>handle</i> | FlexCAN handle pointer.                                                                    |
| <i>xfer</i>   | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                       |
|--------------------------------|-------------------------------------------------------|
| <i>kStatus_Success</i>         | Start Tx Message Buffer sending process successfully. |
| <i>kStatus_Fail</i>            | Write Tx Message Buffer failed.                       |
| <i>kStatus_FLEXCAN_Tx-Busy</i> | Tx Message Buffer is in use.                          |

### 13.2.7.46 status\_t FLEXCAN\_TransferReceiveNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *xfer* )

This function receives a message using IRQ. This is non-blocking function, which returns right away. When the message has been received, the receive callback function is called.

Parameters

|               |                                                                                            |
|---------------|--------------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                           |
| <i>handle</i> | FlexCAN handle pointer.                                                                    |
| <i>xfer</i>   | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

Return values

|                                |                                                           |
|--------------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>         | - Start Rx Message Buffer receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-Busy</i> | - Rx Message Buffer is in use.                            |

### 13.2.7.47 status\_t FLEXCAN\_TransferReceiveFifoNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_fifo\_transfer\_t \* *xfer* )

This function receives a message using IRQ. This is a non-blocking function, which returns right away. When all messages have been received, the receive callback function is called.

Parameters

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                      |
| <i>handle</i> | FlexCAN handle pointer.                                                               |
| <i>xfer</i>   | FlexCAN Rx FIFO transfer structure. See the <a href="#">flexcan_fifo_transfer_t</a> . |

Return values

|                                    |                                                 |
|------------------------------------|-------------------------------------------------|
| <i>kStatus_Success</i>             | - Start Rx FIFO receiving process successfully. |
| <i>kStatus_FLEXCAN_Rx-FifoBusy</i> | - Rx FIFO is currently in use.                  |

### 13.2.7.48 void FLEXCAN\_TransferAbortSend ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )

This function aborts the interrupt driven message send process.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mbIdx</i>  | The FlexCAN Message Buffer index. |

### 13.2.7.49 void FLEXCAN\_TransferAbortReceive ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, uint8\_t *mbIdx* )

This function aborts the interrupt driven message receive process.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.  |
| <i>handle</i> | FlexCAN handle pointer.           |
| <i>mbIdx</i>  | The FlexCAN Message Buffer index. |

### 13.2.7.50 void FLEXCAN\_TransferAbortReceiveFifo ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )

This function aborts the interrupt driven message receive from Rx FIFO process.

## FlexCAN Driver

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address. |
| <i>handle</i> | FlexCAN handle pointer.          |

### 13.2.7.51 void FLEXCAN\_TransferHandleIRQ ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )

This function handles the FlexCAN Error, the Message Buffer, and the Rx FIFO IRQ request.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address. |
| <i>handle</i> | FlexCAN handle pointer.          |

## 13.3 FlexCAN eDMA Driver

### 13.3.1 Overview

#### Data Structures

- struct `flexcan_edma_handle_t`  
*FlexCAN eDMA handle. [More...](#)*

#### TypeDefs

- typedef void(\* `flexcan_edma_transfer_callback_t`)  
(CAN\_Type \*base, flexcan\_edma\_handle\_t  
\*handle, status\_t status, void \*userData)  
*FlexCAN transfer callback function.*

#### Driver version

- #define `FSL_FLEXCAN_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 3, 1))  
*FlexCAN EDMA driver version 2.3.1.*

#### eDMA transactional

- void `FLEXCAN_TransferCreateHandleEDMA` (CAN\_Type \*base, flexcan\_edma\_handle\_t  
\*handle, `flexcan_edma_transfer_callback_t` callback, void \*userData, `edma_handle_t` \*rxFifoEdmaHandle)  
*Initializes the FlexCAN handle, which is used in transactional functions.*
- status\_t `FLEXCAN_TransferReceiveFifoEDMA` (CAN\_Type \*base, flexcan\_edma\_handle\_t  
\*handle, `flexcan_fifo_transfer_t` \*xfer)  
*Receives the CAN Message from the Rx FIFO using eDMA.*
- void `FLEXCAN_TransferAbortReceiveFifoEDMA` (CAN\_Type \*base, flexcan\_edma\_handle\_t  
\*handle)  
*Aborts the receive process which used eDMA.*

### 13.3.2 Data Structure Documentation

#### 13.3.2.1 struct \_flexcan\_edma\_handle

##### Data Fields

- `flexcan_edma_transfer_callback_t` `callback`  
*Callback function.*
- void \* `userData`  
*FlexCAN callback function parameter.*
- `edma_handle_t` \* `rxFifoEdmaHandle`

## FlexCAN eDMA Driver

- volatile uint8\_t **rxFifoState**  
*Rx FIFO transfer state.*

### 13.3.2.1.0.44 Field Documentation

13.3.2.1.0.44.1 **flexcan\_edma\_transfer\_callback\_t flexcan\_edma\_handle\_t::callback**

13.3.2.1.0.44.2 **void\* flexcan\_edma\_handle\_t::userData**

13.3.2.1.0.44.3 **edma\_handle\_t\* flexcan\_edma\_handle\_t::rxFifoEdmaHandle**

13.3.2.1.0.44.4 **volatile uint8\_t flexcan\_edma\_handle\_t::rxFifoState**

### 13.3.3 Macro Definition Documentation

13.3.3.1 **#define FSL\_FLEXCAN\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 1))**

### 13.3.4 Typedef Documentation

13.3.4.1 **typedef void(\* flexcan\_edma\_transfer\_callback\_t)(CAN\_Type \*base,  
flexcan\_edma\_handle\_t \*handle, status\_t status, void \*userData)**

### 13.3.5 Function Documentation

13.3.5.1 **void FLEXCAN\_TransferCreateHandleEDMA ( CAN\_Type \* base,  
flexcan\_edma\_handle\_t \* handle, flexcan\_edma\_transfer\_callback\_t callback,  
void \* userData, edma\_handle\_t \* rxFifoEdmaHandle )**

Parameters

|                         |                                                     |
|-------------------------|-----------------------------------------------------|
| <i>base</i>             | FlexCAN peripheral base address.                    |
| <i>handle</i>           | Pointer to flexcan_edma_handle_t structure.         |
| <i>callback</i>         | The callback function.                              |
| <i>userData</i>         | The parameter of the callback function.             |
| <i>rxFifoEdmaHandle</i> | User-requested DMA handle for Rx FIFO DMA transfer. |

13.3.5.2 **status\_t FLEXCAN\_TransferReceiveFifoEDMA ( CAN\_Type \* base,  
flexcan\_edma\_handle\_t \* handle, flexcan\_fifo\_transfer\_t \* xfer )**

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.                                                       |
| <i>handle</i> | Pointer to flexcan_edma_handle_t structure.                                            |
| <i>xfer</i>   | FlexCAN Rx FIFO EDMA transfer structure, see <a href="#">flexcan_fifo_transfer_t</a> . |

Return values

|                                    |                            |
|------------------------------------|----------------------------|
| <i>kStatus_Success</i>             | if succeed, others failed. |
| <i>kStatus_FLEXCAN_Rx-FifoBusy</i> | Previous transfer ongoing. |

### 13.3.5.3 void FLEXCAN\_TransferAbortReceiveFifoEDMA ( CAN\_Type \* *base*, flexcan\_edma\_handle\_t \* *handle* )

This function aborts the receive process which used eDMA.

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | FlexCAN peripheral base address.            |
| <i>handle</i> | Pointer to flexcan_edma_handle_t structure. |



# Chapter 14

## FLEXSPI: Flexible Serial Peripheral Interface Driver

### 14.1 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.

### Modules

- [FLEXSPI eDMA Driver](#)

### 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.*

## Overview

### 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 _flexspi_status {`  
  `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 _flexspi_command {`  
  `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.*
- `enum _flexspi_pad {`

```
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\_IpCommandExcutionDoneFlag,
   
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 writting.*

- enum `flexspi_ip_error_code_t` {

## Overview

```
kFLEXSPI_IpCmdError.NoError = 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\_AhbCmdError.NoError = 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`  
*Trigger source of current command sequence granted by arbitrator.*
- enum `flexspi_command_type_t` {  
    kFLEXSPI\_Command,  
    kFLEXSPI\_Config }

## Driver version

- #define `FSL_FLEXSPI_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 5))  
*FLEXSPI driver version 2.0.5.*

## Initialization and deinitialization

- 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_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

- static void [FLEXSPI\\_EnableIPParallelMode](#) (FLEXSPI\_Type \*base, bool enable)

## Data Structure Documentation

- static void **FLEXSPI\_EnableAHBParallelMode** (FLEXSPI\_Type \*base, bool enable)  
*Enables/disables the FLEXSPI IP command parallel mode.*
- 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, uint32\_t \*buffer, size\_t size)  
*Sends a buffer of data bytes using blocking method.*
- status\_t **FLEXSPI\_ReadBlocking** (FLEXSPI\_Type \*base, uint32\_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.*

## 14.2 Data Structure Documentation

### 14.2.1 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.
- **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 excution finished.
- **bool enableAHBCachable**  
Enable AHB bus cachable read access support.

## Data Structure Documentation

### 14.2.1.0.0.45 Field Documentation

14.2.1.0.0.45.1 `flexspi_read_sample_clock_t flexspi_config_t::rxSampleClock`

14.2.1.0.0.45.2 `bool flexspi_config_t::enableSckFreeRunning`

14.2.1.0.0.45.3 `bool flexspi_config_t::enableCombination`

14.2.1.0.0.45.4 `bool flexspi_config_t::enableDoze`

14.2.1.0.0.45.5 `bool flexspi_config_t::enableHalfSpeedAccess`

14.2.1.0.0.45.6 `bool flexspi_config_t::enableSckBDiffOpt`

14.2.1.0.0.45.7 `bool flexspi_config_t::enableSameConfigForAll`

14.2.1.0.0.45.8 `uint16_t flexspi_config_t::seqTimeoutCycle`

14.2.1.0.0.45.9 `uint8_t flexspi_config_t::ipGrantTimeoutCycle`

14.2.1.0.0.45.10 `uint8_t flexspi_config_t::txWatermark`

14.2.1.0.0.45.11 `uint8_t flexspi_config_t::rxWatermark`

14.2.1.0.0.45.12 `bool flexspi_config_t::enableAHBWriteIpTxFifo`

14.2.1.0.0.45.13 `bool flexspi_config_t::enableAHBWriteIpRxFifo`

14.2.1.0.0.45.14 `uint8_t flexspi_config_t::ahbGrantTimeoutCycle`

14.2.1.0.0.45.15 `uint16_t flexspi_config_t::ahbBusTimeoutCycle`

14.2.1.0.0.45.16 `uint8_t flexspi_config_t::resumeWaitCycle`

14.2.1.0.0.45.17 `flexspi_ahbBuffer_config_t flexspi_config_t::buffer[FSL_FEATURE_FLEXSPI_A-HB_BUFFER_COUNT]`

14.2.1.0.0.45.18 `bool flexspi_config_t::enableClearAHBBufferOpt`

14.2.1.0.0.45.19 `bool flexspi_config_t::enableReadAddressOpt`

when eanble, there is no AHB read burst start address alignment limitation.

- 14.2.1.0.0.45.20 **bool flexspi\_config\_t::enableAHBPrefetch**
- 14.2.1.0.0.45.21 **bool flexspi\_config\_t::enableAHBBufferable**
- 14.2.1.0.0.45.22 **bool flexspi\_config\_t::enableAHBCachable**

## 14.2.2 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.*
- **flexspi\_cs\_interval\_cycle\_unit\_t CSIntervalUnit**  
*CS interval unit, 1 or 256 cycle.*
- **uint16\_t CSInterval**  
*CS line assert interval, mutiply 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, mutiply 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.*

## Data Structure Documentation

### 14.2.2.0.0.46 Field Documentation

14.2.2.0.0.46.1 `uint32_t flexspi_device_config_t::flexspiRootClk`

14.2.2.0.0.46.2 `bool flexspi_device_config_t::isSck2Enabled`

14.2.2.0.0.46.3 `uint32_t flexspi_device_config_t::flashSize`

14.2.2.0.0.46.4 `flexspi_cs_interval_cycle_unit_t flexspi_device_config_t::CSIntervalUnit`

14.2.2.0.0.46.5 `uint16_t flexspi_device_config_t::CSInterval`

14.2.2.0.0.46.6 `uint8_t flexspi_device_config_t::CSHoldTime`

14.2.2.0.0.46.7 `uint8_t flexspi_device_config_t::CSSetupTime`

14.2.2.0.0.46.8 `uint8_t flexspi_device_config_t::dataValidTime`

14.2.2.0.0.46.9 `uint8_t flexspi_device_config_t::columnspace`

14.2.2.0.0.46.10 `bool flexspi_device_config_t::enableWordAddress`

14.2.2.0.0.46.11 `uint8_t flexspi_device_config_t::AWRSeqIndex`

14.2.2.0.0.46.12 `uint8_t flexspi_device_config_t::AWRSeqNumber`

14.2.2.0.0.46.13 `uint8_t flexspi_device_config_t::ARDSeqIndex`

14.2.2.0.0.46.14 `uint8_t flexspi_device_config_t::ARDSeqNumber`

14.2.2.0.0.46.15 `flexspi_ahb_write_wait_unit_t flexspi_device_config_t::AHBWriteWaitUnit`

14.2.2.0.0.46.16 `uint16_t flexspi_device_config_t::AHBWriteWaitInterval`

14.2.2.0.0.46.17 `bool flexspi_device_config_t::enableWriteMask`

### 14.2.3 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.*

#### 14.2.3.0.0.47 Field Documentation

14.2.3.0.0.47.1 `uint32_t flexspi_transfer_t::deviceAddress`

14.2.3.0.0.47.2 `flexspi_port_t flexspi_transfer_t::port`

14.2.3.0.0.47.3 `flexspi_command_type_t flexspi_transfer_t::cmdType`

14.2.3.0.0.47.4 `uint8_t flexspi_transfer_t::seqIndex`

14.2.3.0.0.47.5 `uint8_t flexspi_transfer_t::SeqNumber`

14.2.3.0.0.47.6 `uint32_t* flexspi_transfer_t::data`

14.2.3.0.0.47.7 `size_t flexspi_transfer_t::dataSize`

#### 14.2.4 struct \_flexspi\_handle

#### Data Fields

- `uint32_t state`  
*Internal state for FLEXSPI transfer.*
- `uint32_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.*

#### 14.2.4.0.0.48 Field Documentation

14.2.4.0.0.48.1 `uint32_t* flexspi_handle_t::data`

14.2.4.0.0.48.2 `size_t flexspi_handle_t::dataSize`

14.2.4.0.0.48.3 `size_t flexspi_handle_t::transferTotalSize`

14.2.4.0.0.48.4 `void* flexspi_handle_t::userData`

## Enumeration Type Documentation

### 14.3 Macro Definition Documentation

14.3.1 `#define FSL_FLEXSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))`

14.3.2 `#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))
```

### 14.4 Typedef Documentation

14.4.1 `typedef void(* flexspi_transfer_callback_t)(FLEXSPI_Type *base,  
flexspi_handle_t *handle, status_t status, void *userData)`

### 14.5 Enumeration Type Documentation

#### 14.5.1 enum \_flexspi\_status

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.

#### 14.5.2 enum \_flexspi\_command

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.

### 14.5.3 enum \_flexspi\_pad

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].

### 14.5.4 enum flexspi\_flags\_t

Enumerator

***kFLEXSPI\_SequenceExecutionTimeoutFlag*** Sequence execution timeout.

## Enumeration Type Documentation

***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.

### 14.5.5 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.

### 14.5.6 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.

### 14.5.7 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.

### 14.5.8 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.

### 14.5.9 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.

## Function Documentation

### 14.5.10 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.

### 14.5.11 enum flexspi\_arb\_command\_source\_t

### 14.5.12 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.

## 14.6 Function Documentation

### 14.6.1 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.     |

### 14.6.2 void FLEXSPI\_GetDefaultConfig( flexspi\_config\_t \* *config* )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>config</i> | FLEXSPI configuration structure. |
|---------------|----------------------------------|

### 14.6.3 void FLEXSPI\_Deinit( FLEXSPI\_Type \* *base* )

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

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

#### 14.6.4 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.          |

#### 14.6.5 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. |
|-------------|----------------------------------|

#### 14.6.6 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. |

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

## Function Documentation

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

**14.6.8 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.        |

**14.6.9 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. |

**14.6.10 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. |

**14.6.11 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. |
|------------|------------------|

#### 14.6.12 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. |
|------------|------------------|

#### 14.6.13 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.      |

#### 14.6.14 static void FLEXSPI\_GetFifoCounts ( **FLEXSPI\_Type** \* *base*, size\_t \* *txCount*, size\_t \* *rxCount* ) [inline], [static]

Parameters

## Function Documentation

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <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.  |

**14.6.15 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 <a href="#">flexspi_flags_t</a> could get the related status. |
|------------------|---------------------------------------------------------------------------------------------------|

**14.6.16 static void FLEXSPI\_ClearInterruptStatusFlags ( FLEXSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|                  |                                  |
|------------------|----------------------------------|
| <i>base</i>      | FLEXSPI peripheral base address. |
| <i>interrupt</i> | status flag.                     |

**14.6.17 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 uint8_t type variable to receive the selected clock phase on PORTA. |
| <i>portBPhase</i> | Pointer to a uint8_t type variable to receive the selected clock phase on PORTB. |

14.6.18 **static flexspi\_arb\_command\_source\_t FLEXSPI\_GetArbitrator-CommandSource ( FLEXSPI\_Type \* *base* ) [inline], [static]**

## Function Documentation

Parameters

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

Return values

|                |                                     |
|----------------|-------------------------------------|
| <i>trigger</i> | source of current command sequence. |
|----------------|-------------------------------------|

**14.6.19 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. |
|--------------|--------------------------------------|

**14.6.20 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. |
|--------------|---------------------------------------|

**14.6.21 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. |

#### 14.6.22 static void FLEXSPI\_EnableIIParallelMode ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                    |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

#### 14.6.23 static void FLEXSPI\_EnableAHBParallelMode ( FLEXSPI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                    |
| <i>enable</i> | True means enable parallel mode, false means disable parallel mode. |

#### 14.6.24 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. |

## Function Documentation

|              |                         |
|--------------|-------------------------|
| <i>cmd</i>   | Command sequence array. |
| <i>count</i> | Number of sequences.    |

**14.6.25 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.         |

**14.6.26 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.

**14.6.27 status\_t FLEXSPI\_WriteBlocking ( FLEXSPI\_Type \* *base*, uint32\_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_IpCommandSequenceError</i>   | IP command sequencen error detected |
| <i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>    | IP command grant timeout detected   |

#### 14.6.28 **status\_t FLEXSPI\_ReadBlocking ( FLEXSPI\_Type \* *base*, uint32\_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_IpCommandSequenceError</i>   | IP command sequencen error detected |
| <i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>    | IP command grant timeout detected   |

#### 14.6.29 **status\_t FLEXSPI\_TransferBlocking ( FLEXSPI\_Type \* *base*, flexspi\_transfer\_t \* *xfer* )**

## Function Documentation

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      |

**14.6.30 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.                    |

**14.6.31 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. |

#### 14.6.32 **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> | <i>count</i> is Invalid.       |
| <i>kStatus_Success</i>         | Successfully return the count. |

#### 14.6.33 **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

## Function Documentation

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                      |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state |

**14.6.34 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. |

## 14.7 FLEXSPI eDMA Driver

### 14.7.1 Overview

#### Data Structures

- struct `flexspi_edma_handle_t`

*FLEXSPI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### TypeDefs

- typedef void(\* `flexspi_edma_callback_t` )(FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, status\_t status, void \*userData)

*FLEXSPI eDMA transfer callback function for finish and error.*

#### Enumerations

- enum `flexspi_edma_transfer_nsize_t` {
   
kFLEXPSI\_EDMA\_NSize1Bytes = 0x1U,
   
kFLEXPSI\_EDMA\_NSize2Bytes = 0x2U,
   
kFLEXPSI\_EDMA\_NSize4Bytes = 0x4U,
   
kFLEXPSI\_EDMA\_NSize8Bytes = 0x8U,
   
kFLEXPSI\_EDMA\_NSize32Bytes = 0x20U }

*eDMA transfer configuration*

#### Driver version

- #define `FSL_FLEXSPI_EDMA_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))

*FLEXSPI EDMA driver version 2.0.2.*

#### FLEXSPI eDMA Transactional

- void `FLEXSPI_TransferCreateHandleEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, `flexspi_edma_callback_t` callback, void \*userData, `edma_handle_t` \*txDmaHandle, `edma_handle_t` \*rxDmaHandle)
   
*Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.*
- void `FLEXSPI_TransferUpdateSizeEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, `flexspi_edma_transfer_nsize_t` nsize)
   
*Update FLEXSPI EDMA transfer source data transfer size(SSIZE) and destination data transfer size(DS-IZE).*
- status\_t `FLEXSPI_TransferEDMA` (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, `flexspi_transfer_t` \*xfer)
   
*Transfers FLEXSPI data using an eDMA non-blocking method.*

## FLEXSPI eDMA Driver

- void **FLEXSPI\_TransferAbortEDMA** (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle)  
*Aborts the transfer data using eDMA.*
- status\_t **FLEXSPI\_TransferGetCountEDMA** (FLEXSPI\_Type \*base, flexspi\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the transferred counts of transfer.*

### 14.7.2 Data Structure Documentation

#### 14.7.2.1 struct \_flexspi\_edma\_handle

##### Data Fields

- **edma\_handle\_t \* txDmaHandle**  
*eDMA handler for FLEXSPI Tx.*
- **edma\_handle\_t \* rxDmaHandle**  
*eDMA handler for FLEXSPI Rx.*
- **size\_t transferSize**  
*Bytes need to transfer.*
- **flexspi\_edma\_transfer\_nsize\_t nsize**  
*eDMA SSIZE/DSIZE in each transfer.*
- **uint8\_t nbytes**  
*eDMA minor byte transfer count initially configured.*
- **uint8\_t count**  
*The transfer data count in a DMA request.*
- **uint32\_t state**  
*Internal state for FLEXSPI eDMA transfer.*
- **flexspi\_edma\_callback\_t completionCallback**  
*A callback function called after the eDMA transfer is finished.*
- **void \* userData**  
*User callback parameter.*

#### 14.7.2.1.0.49 Field Documentation

- 14.7.2.1.0.49.1 `edma_handle_t* flexspi_edma_handle_t::txDmaHandle`
- 14.7.2.1.0.49.2 `edma_handle_t* flexspi_edma_handle_t::rxDmaHandle`
- 14.7.2.1.0.49.3 `size_t flexspi_edma_handle_t::transferSize`
- 14.7.2.1.0.49.4 `flexspi_edma_transfer_nsize_t flexspi_edma_handle_t::nsize`
- 14.7.2.1.0.49.5 `uint8_t flexspi_edma_handle_t::nbytes`
- 14.7.2.1.0.49.6 `uint8_t flexspi_edma_handle_t::count`
- 14.7.2.1.0.49.7 `uint32_t flexspi_edma_handle_t::state`
- 14.7.2.1.0.49.8 `flexspi_edma_callback_t flexspi_edma_handle_t::completionCallback`

#### 14.7.3 Macro Definition Documentation

- 14.7.3.1 `#define FSL_FLEXSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

#### 14.7.4 Enumeration Type Documentation

##### 14.7.4.1 enum flexspi\_edma\_transfer\_nsize\_t

Enumerator

- `kFLEXSPI_EDMA_N_SIZE_1BYTES` Source/Destination data transfer size is 1 byte every time.
- `kFLEXSPI_EDMA_N_SIZE_2BYTES` Source/Destination data transfer size is 2 bytes every time.
- `kFLEXSPI_EDMA_N_SIZE_4BYTES` Source/Destination data transfer size is 4 bytes every time.
- `kFLEXSPI_EDMA_N_SIZE_8BYTES` Source/Destination data transfer size is 8 bytes every time.
- `kFLEXSPI_EDMA_N_SIZE_32BYTES` Source/Destination data transfer size is 32 bytes every time.

#### 14.7.5 Function Documentation

- 14.7.5.1 `void FLEXSPI_TransferCreateHandleEDMA ( FLEXSPI_Type * base,  
flexspi_edma_handle_t * handle, flexspi_edma_callback_t callback, void *  
userData, edma_handle_t * txDmaHandle, edma_handle_t * rxDmaHandle )`

Parameters

---

## FLEXSPI eDMA Driver

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | FLEXSPI peripheral base address                |
| <i>handle</i>      | Pointer to flexspi_edma_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. |

**14.7.5.2 void FLEXSPI\_TransferUpdateSizeEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_handle\_t \* *handle*, flexspi\_edma\_transfer\_nsize\_t *nsize* )**

Parameters

|               |                                                                                                                  |
|---------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address                                                                                  |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure                                                                       |
| <i>nsize</i>  | FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_EDMAnSize1Bytes(one byte). |

See Also

[flexspi\\_edma\\_transfer\\_nsize\\_t](#) .

**14.7.5.3 status\_t FLEXSPI\_TransferEDMA ( FLEXSPI\_Type \* *base*, flexspi\_edma\_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_edma_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 EDMA transfer. |
| <i>kStatus_Success</i>         | FLEXSPI successfully start edma transfer.                                                                    |

#### 14.7.5.4 void FLEXSPI\_TransferAbortEDMA ( **FLEXSPI\_Type** \* *base*, **flexspi\_edma\_handle\_t** \* *handle* )

This function aborts the transfer data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.           |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure |

#### 14.7.5.5 **status\_t** FLEXSPI\_TransferGetCountEDMA ( **FLEXSPI\_Type** \* *base*, **flexspi\_edma\_handle\_t** \* *handle*, **size\_t** \* *count* )

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.            |
| <i>handle</i> | Pointer to flexspi_edma_handle_t structure. |
| <i>count</i>  | Bytes transfer.                             |

Return values

|                                      |                                                                |
|--------------------------------------|----------------------------------------------------------------|
| <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 15

## FTM: FlexTimer Driver

### 15.1 Overview

The MCUXpresso SDK provides a driver for the FlexTimer Module (FTM) of MCUXpresso SDK devices.

### 15.2 Function groups

The FTM driver supports the generation of PWM signals, input capture, dual edge capture, output compare, and quadrature decoder modes. The driver also supports configuring each of the FTM fault inputs.

#### 15.2.1 Initialization and deinitialization

The function [FTM\\_Init\(\)](#) initializes the FTM with specified configurations. The function [FTM\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the FTM for the requested register update mode for registers with buffers. It also sets up the FTM's fault operation mode and FTM behavior in the BDM mode.

The function [FTM\\_Deinit\(\)](#) disables the FTM counter and turns off the module clock.

#### 15.2.2 PWM Operations

The function [FTM\\_SetupPwm\(\)](#) sets up FTM channels for the PWM output. The function sets up the PWM signal properties for multiple channels. Each channel has its own duty cycle and level-mode specified. However, the same PWM period and PWM mode 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 [FTM\\_UpdatePwmDutycycle\(\)](#) updates the PWM signal duty cycle of a particular FTM channel.

The function [FTM\\_UpdateChnlEdgeLevelSelect\(\)](#) updates the level select bits of a particular FTM channel. This can be used to disable the PWM output when making changes to the PWM signal.

#### 15.2.3 Input capture operations

The function [FTM\\_SetupInputCapture\(\)](#) sets up an FTM channel for the input capture. The user can specify the capture edge and a filter value to be used when processing the input signal.

The function [FTM\\_SetupDualEdgeCapture\(\)](#) can be used to measure the pulse width of a signal. A channel pair is used during capture with the input signal coming through a channel n. The user can specify whether

## Register Update

to use one-shot or continuous capture, the capture edge for each channel, and any filter value to be used when processing the input signal.

### 15.2.4 Output compare operations

The function [FTM\\_SetupOutputCompare\(\)](#) sets up an FTM channel for the output comparison. The user can specify the channel output on a successful comparison and a comparison value.

### 15.2.5 Quad decode

The function [FTM\\_SetupQuadDecode\(\)](#) sets up FTM channels 0 and 1 for quad decoding. The user can specify the quad decoding mode, polarity, and filter properties for each input signal.

### 15.2.6 Fault operation

The function [FTM\\_SetupFault\(\)](#) sets up the properties for each fault. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

## 15.3 Register Update

Some of the FTM registers have buffers. The driver supports various methods to update these registers with the content of the register buffer. The registers can be updated using the PWM synchronized loading or an intermediate point loading. The update mechanism for register with buffers can be specified through the following fields available in the configuration structure. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/ftmMultiple PWM synchronization update modes can be used by providing an OR'ed list of options available in the enumeration [ftm\\_pwm\\_sync\\_method\\_t](#) to the pwmSyncMode field.

When using an intermediate reload points, the PWM synchronization is not required. Multiple reload points can be used by providing an OR'ed list of options available in the enumeration [ftm\\_reload\\_point\\_t](#) to the reloadPoints field.

The driver initialization function sets up the appropriate bits in the FTM module based on the register update options selected.

If software PWM synchronization is used, the below function can be used to initiate a software trigger. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/ftm

## 15.4 Typical use case

### 15.4.1 PWM output

Output a PWM signal on two FTM channels with different duty cycles. Periodically update the PWM signal duty cycle. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/ftm

## Data Structures

- struct `ftm_chnl_pwm_signal_param_t`  
*Options to configure a FTM channel's PWM signal.* [More...](#)
- struct `ftm_chnl_pwm_config_param_t`  
*Options to configure a FTM channel using precise setting.* [More...](#)
- struct `ftm_dual_edge_capture_param_t`  
*FlexTimer dual edge capture parameters.* [More...](#)
- struct `ftm_phase_params_t`  
*FlexTimer quadrature decode phase parameters.* [More...](#)
- struct `ftm_fault_param_t`  
*Structure is used to hold the parameters to configure a FTM fault.* [More...](#)
- struct `ftm_config_t`  
*FTM configuration structure.* [More...](#)

## Enumerations

- enum `ftm_chnl_t` {
   
kFTM\_Chnl\_0 = 0U,
   
kFTM\_Chnl\_1,
   
kFTM\_Chnl\_2,
   
kFTM\_Chnl\_3,
   
kFTM\_Chnl\_4,
   
kFTM\_Chnl\_5,
   
kFTM\_Chnl\_6,
   
kFTM\_Chnl\_7
 }
   
*List of FTM channels.*
- enum `ftm_fault_input_t` {
   
kFTM\_Fault\_0 = 0U,
   
kFTM\_Fault\_1,
   
kFTM\_Fault\_2,
   
kFTM\_Fault\_3
 }
   
*List of FTM faults.*
- enum `ftm_pwm_mode_t` {
   
kFTM\_EdgeAlignedPwm = 0U,
   
kFTM\_CenterAlignedPwm,
   
kFTM\_CombinedPwm
 }
   
*FTM PWM operation modes.*
- enum `ftm_pwm_level_select_t` {

## Typical use case

```
kFTM_NoPwmSignal = 0U,
kFTM_LowTrue,
kFTM_HighTrue }
```

*FTM PWM output pulse mode: high-true, low-true or no output.*

- enum `ftm_output_compare_mode_t`{  
  kFTM\_NoOutputSignal = (1U << FTM\_CnSC\_MSA\_SHIFT),  
  kFTM\_ToggleOnMatch = ((1U << FTM\_CnSC\_MSA\_SHIFT) | (1U << FTM\_CnSC\_ELSA\_S-HIFT)),  
  kFTM\_ClearOnMatch = ((1U << FTM\_CnSC\_MSA\_SHIFT) | (2U << FTM\_CnSC\_ELSA\_SHIFT)),  
  kFTM\_SetOnMatch = ((1U << FTM\_CnSC\_MSA\_SHIFT) | (3U << FTM\_CnSC\_ELSA\_SHIFT)) }

*FlexTimer output compare mode.*

- enum `ftm_input_capture_edge_t`{  
  kFTM\_RisingEdge = (1U << FTM\_CnSC\_ELSA\_SHIFT),  
  kFTM\_FallingEdge = (2U << FTM\_CnSC\_ELSA\_SHIFT),  
  kFTM\_RiseAndFallEdge = (3U << FTM\_CnSC\_ELSA\_SHIFT) }

*FlexTimer input capture edge.*

- enum `ftm_dual_edge_capture_mode_t`{  
  kFTM\_OneShot = 0U,  
  kFTM\_Continuous = (1U << FTM\_CnSC\_MSA\_SHIFT) }

*FlexTimer dual edge capture modes.*

- enum `ftm_quad_decode_mode_t`{  
  kFTM\_QuadPhaseEncode = 0U,  
  kFTM\_QuadCountAndDir }
- enum `ftm_phase_polarity_t`{  
  kFTM\_QuadPhaseNormal = 0U,  
  kFTM\_QuadPhaseInvert }

*FlexTimer quadrature phase polarities.*

- enum `ftm_deadtime_prescale_t`{  
  kFTM\_Deadtime\_Prescale\_1 = 1U,  
  kFTM\_Deadtime\_Prescale\_4,  
  kFTM\_Deadtime\_Prescale\_16 }

*FlexTimer pre-scaler factor for the dead time insertion.*

- enum `ftm_clock_source_t`{  
  kFTM\_SystemClock = 1U,  
  kFTM\_FixedClock,  
  kFTM\_ExternalClock }

*FlexTimer clock source selection.*

- enum `ftm_clock_prescale_t`{

```
kFTM_Prescale_Divide_1 = 0U,
kFTM_Prescale_Divide_2,
kFTM_Prescale_Divide_4,
kFTM_Prescale_Divide_8,
kFTM_Prescale_Divide_16,
kFTM_Prescale_Divide_32,
kFTM_Prescale_Divide_64,
kFTM_Prescale_Divide_128 }
```

*FlexTimer pre-scaler factor selection for the clock source.*

- enum `ftm_bdm_mode_t` {
   
kFTM\_BdmMode\_0 = 0U,
   
kFTM\_BdmMode\_1,
   
kFTM\_BdmMode\_2,
   
kFTM\_BdmMode\_3 }

*Options for the FlexTimer behaviour in BDM Mode.*

- enum `ftm_fault_mode_t` {
   
kFTM\_Fault\_Disable = 0U,
   
kFTM\_Fault\_EvenChnls,
   
kFTM\_Fault\_AllChnlsMan,
   
kFTM\_Fault\_AllChnlsAuto }

*Options for the FTM fault control mode.*

- enum `ftm_external_trigger_t` {
   
kFTM\_Chnl0Trigger = (1U << 4),
   
kFTM\_Chnl1Trigger = (1U << 5),
   
kFTM\_Chnl2Trigger = (1U << 0),
   
kFTM\_Chnl3Trigger = (1U << 1),
   
kFTM\_Chnl4Trigger = (1U << 2),
   
kFTM\_Chnl5Trigger = (1U << 3),
   
kFTM\_InitTrigger = (1U << 6),
   
kFTM\_ReloadInitTrigger = (1U << 7) }

*FTM external trigger options.*

- enum `ftm_pwm_sync_method_t` {
   
kFTM\_SoftwareTrigger = FTM\_SYNC\_SWSYNC\_MASK,
   
kFTM\_HardwareTrigger\_0 = FTM\_SYNC\_TRIG0\_MASK,
   
kFTM\_HardwareTrigger\_1 = FTM\_SYNC\_TRIG1\_MASK,
   
kFTM\_HardwareTrigger\_2 = FTM\_SYNC\_TRIG2\_MASK }

*FlexTimer PWM sync options to update registers with buffer.*

- enum `ftm_reload_point_t` {

## Typical use case

```
kFTM_Chnl0Match = (1U << 0),
kFTM_Chnl1Match = (1U << 1),
kFTM_Chnl2Match = (1U << 2),
kFTM_Chnl3Match = (1U << 3),
kFTM_Chnl4Match = (1U << 4),
kFTM_Chnl5Match = (1U << 5),
kFTM_Chnl6Match = (1U << 6),
kFTM_Chnl7Match = (1U << 7),
kFTM_CntMax = (1U << 8),
kFTM_CntMin = (1U << 9),
kFTM_HalfCycMatch = (1U << 10) }
```

*FTM options available as loading point for register reload.*

- enum `ftm_interrupt_enable_t` {  
    kFTM\_Chnl0InterruptEnable = (1U << 0),  
    kFTM\_Chnl1InterruptEnable = (1U << 1),  
    kFTM\_Chnl2InterruptEnable = (1U << 2),  
    kFTM\_Chnl3InterruptEnable = (1U << 3),  
    kFTM\_Chnl4InterruptEnable = (1U << 4),  
    kFTM\_Chnl5InterruptEnable = (1U << 5),  
    kFTM\_Chnl6InterruptEnable = (1U << 6),  
    kFTM\_Chnl7InterruptEnable = (1U << 7),  
    kFTM\_FaultInterruptEnable = (1U << 8),  
    kFTM\_TimeOverflowInterruptEnable = (1U << 9),  
    kFTM\_ReloadInterruptEnable = (1U << 10) }

*List of FTM interrupts.*

- enum `ftm_status_flags_t` {  
    kFTM\_Chnl0Flag = (1U << 0),  
    kFTM\_Chnl1Flag = (1U << 1),  
    kFTM\_Chnl2Flag = (1U << 2),  
    kFTM\_Chnl3Flag = (1U << 3),  
    kFTM\_Chnl4Flag = (1U << 4),  
    kFTM\_Chnl5Flag = (1U << 5),  
    kFTM\_Chnl6Flag = (1U << 6),  
    kFTM\_Chnl7Flag = (1U << 7),  
    kFTM\_FaultFlag = (1U << 8),  
    kFTM\_TimeOverflowFlag = (1U << 9),  
    kFTM\_ChnlTriggerFlag = (1U << 10),  
    kFTM\_ReloadFlag = (1U << 11) }

*List of FTM flags.*

- enum `_ftm_quad_decoder_flags` {  
    kFTM\_QuadDecoderCountingIncreaseFlag = FTM\_QDCTRL\_QUADIR\_MASK,  
    kFTM\_QuadDecoderCountingOverflowOnTopFlag = FTM\_QDCTRL\_TOFDIR\_MASK }

*List of FTM Quad Decoder flags.*

## Functions

- void **FTM\_SetupFault** (FTM\_Type \*base, **ftm\_fault\_input\_t** faultNumber, const **ftm\_fault\_param\_t** \*faultParams)
 

*Sets up the working of the FTM fault protection.*
- static void **FTM\_SetGlobalTimeBaseOutputEnable** (FTM\_Type \*base, bool enable)
 

*Enables or disables the FTM global time base signal generation to other FTMs.*
- static void **FTM\_SetOutputMask** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool mask)
 

*Sets the FTM peripheral timer channel output mask.*
- static void **FTM\_SetPwmOutputEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool value)
 

*Allows users to enable an output on an FTM channel.*
- static void **FTM\_SetSoftwareTrigger** (FTM\_Type \*base, bool enable)
 

*Enables or disables the FTM software trigger for PWM synchronization.*
- static void **FTM\_SetWriteProtection** (FTM\_Type \*base, bool enable)
 

*Enables or disables the FTM write protection.*

## Driver version

- #define **FSL\_FTM\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 0))
 

*FTM driver version 2.1.0.*

## Initialization and deinitialization

- status\_t **FTM\_Init** (FTM\_Type \*base, const **ftm\_config\_t** \*config)
 

*Ungates the FTM clock and configures the peripheral for basic operation.*
- void **FTM\_Deinit** (FTM\_Type \*base)
 

*Gates the FTM clock.*
- void **FTM\_GetDefaultConfig** (**ftm\_config\_t** \*config)
 

*Fills in the FTM configuration structure with the default settings.*

## Channel mode operations

- status\_t **FTM\_SetupPwm** (FTM\_Type \*base, const **ftm\_chnl\_pwm\_signal\_param\_t** \*chnlParams, uint8\_t numOfChnls, **ftm\_pwm\_mode\_t** mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)
 

*Configures the PWM signal parameters.*
- void **FTM\_UpdatePwmDutyCycle** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, **ftm\_pwm\_mode\_t** currentPwmMode, uint8\_t dutyCyclePercent)
 

*Updates the duty cycle of an active PWM signal.*
- void **FTM\_UpdateChnlEdgeLevelSelect** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, uint8\_t level)
 

*Updates the edge level selection for a channel.*
- status\_t **FTM\_SetupPwmMode** (FTM\_Type \*base, const **ftm\_chnl\_pwm\_config\_param\_t** \*chnlParams, uint8\_t numOfChnls, **ftm\_pwm\_mode\_t** mode)
 

*Configures the PWM mode parameters.*
- void **FTM\_SetupInputCapture** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, **ftm\_input\_capture\_edge\_t** captureMode, uint32\_t filterValue)
 

*Enables capturing an input signal on the channel using the function parameters.*
- void **FTM\_SetupOutputCompare** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, **ftm\_output\_compare\_mode\_t** compareMode, uint32\_t compareValue)
 

*Configures the FTM to generate timed pulses.*
- void **FTM\_SetupDualEdgeCapture** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, const **ftm\_dual\_edge\_capture\_param\_t** \*edgeParam, uint32\_t filterValue)
 

*Configures the FTM to generate dual edge capture pulses.*

## Typical use case

*Configures the dual edge capture mode of the FTM.*

## Interrupt Interface

- void **FTM\_EnableInterrupts** (FTM\_Type \*base, uint32\_t mask)  
*Enables the selected FTM interrupts.*
- void **FTM\_DisableInterrupts** (FTM\_Type \*base, uint32\_t mask)  
*Disables the selected FTM interrupts.*
- uint32\_t **FTM\_GetEnabledInterrupts** (FTM\_Type \*base)  
*Gets the enabled FTM interrupts.*

## Status Interface

- uint32\_t **FTM\_GetStatusFlags** (FTM\_Type \*base)  
*Gets the FTM status flags.*
- void **FTM\_ClearStatusFlags** (FTM\_Type \*base, uint32\_t mask)  
*Clears the FTM status flags.*

## Read and write the timer period

- static void **FTM\_SetTimerPeriod** (FTM\_Type \*base, uint32\_t ticks)  
*Sets the timer period in units of ticks.*
- static uint32\_t **FTM\_GetCurrentTimerCount** (FTM\_Type \*base)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void **FTM\_StartTimer** (FTM\_Type \*base, **ftm\_clock\_source\_t** clockSource)  
*Starts the FTM counter.*
- static void **FTM\_StopTimer** (FTM\_Type \*base)  
*Stops the FTM counter.*

## Software output control

- static void **FTM\_SetSoftwareCtrlEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool value)  
*Enables or disables the channel software output control.*
- static void **FTM\_SetSoftwareCtrlVal** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool value)  
*Sets the channel software output control value.*

## Channel pair operations

- static void **FTM\_SetFaultControlEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, bool value)  
*This function enables/disables the fault control in a channel pair.*
- static void **FTM\_SetDeadTimeEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, bool value)  
*This function enables/disables the dead time insertion in a channel pair.*
- static void **FTM\_SetComplementaryEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, bool value)  
*This function enables/disables complementary mode in a channel pair.*
- static void **FTM\_SetInvertEnable** (FTM\_Type \*base, **ftm\_chnl\_t** chnlPairNumber, bool value)  
*This function enables/disables inverting control in a channel pair.*

## Quad Decoder

- void [FTM\\_SetupQuadDecode](#) (FTM\_Type \*base, const [ftm\\_phase\\_params\\_t](#) \*phaseAParams, const [ftm\\_phase\\_params\\_t](#) \*phaseBParams, [ftm\\_quad\\_decode\\_mode\\_t](#) quadMode)  
*Configures the parameters and activates the quadrature decoder mode.*
- static uint32\_t [FTM\\_GetQuadDecoderFlags](#) (FTM\_Type \*base)  
*Gets the FTM Quad Decoder flags.*
- static void [FTM\\_SetQuadDecoderModuloValue](#) (FTM\_Type \*base, uint32\_t startValue, uint32\_t overValue)  
*Sets the modulo values for Quad Decoder.*
- static uint32\_t [FTM\\_GetQuadDecoderCounterValue](#) (FTM\_Type \*base)  
*Gets the current Quad Decoder counter value.*
- static void [FTM\\_ClearQuadDecoderCounterValue](#) (FTM\_Type \*base)  
*Clears the current Quad Decoder counter value.*

## 15.5 Data Structure Documentation

### 15.5.1 struct [ftm\\_chnl\\_pwm\\_signal\\_param\\_t](#)

#### Data Fields

- [ftm\\_chnl\\_t chnlNumber](#)  
*The channel/channel pair number.*
- [ftm\\_pwm\\_level\\_select\\_t level](#)  
*PWM output active level select.*
- uint8\_t [dutyCyclePercent](#)  
*PWM pulse width, value should be between 0 to 100 0 = inactive signal(0% duty cycle)...*
- uint8\_t [firstEdgeDelayPercent](#)  
*Used only in combined PWM mode to generate an asymmetrical PWM.*

#### 15.5.1.0.0.50 Field Documentation

##### 15.5.1.0.0.50.1 [ftm\\_chnl\\_t ftm\\_chnl\\_pwm\\_signal\\_param\\_t::chnlNumber](#)

In combined mode, this represents the channel pair number.

##### 15.5.1.0.0.50.2 [ftm\\_pwm\\_level\\_select\\_t ftm\\_chnl\\_pwm\\_signal\\_param\\_t::level](#)

##### 15.5.1.0.0.50.3 [uint8\\_t ftm\\_chnl\\_pwm\\_signal\\_param\\_t::dutyCyclePercent](#)

100 = always active signal (100% duty cycle).

##### 15.5.1.0.0.50.4 [uint8\\_t ftm\\_chnl\\_pwm\\_signal\\_param\\_t::firstEdgeDelayPercent](#)

Specifies the delay to the first edge in a PWM period. If unsure leave as 0; Should be specified as a percentage of the PWM period

## Data Structure Documentation

### 15.5.2 struct ftm\_chnl\_pwm\_config\_param\_t

#### Data Fields

- **ftm\_chnl\_t chnlNumber**  
*The channel/channel pair number.*
- **ftm\_pwm\_level\_select\_t level**  
*PWM output active level select.*
- **uint16\_t dutyValue**  
*PWM pulse width, the uint of this value is timer ticks.*
- **uint16\_t firstEdgeValue**  
*Used only in combined PWM mode to generate an asymmetrical PWM.*

#### 15.5.2.0.0.51 Field Documentation

##### 15.5.2.0.0.51.1 ftm\_chnl\_t ftm\_chnl\_pwm\_config\_param\_t::chnlNumber

In combined mode, this represents the channel pair number.

##### 15.5.2.0.0.51.2 ftm\_pwm\_level\_select\_t ftm\_chnl\_pwm\_config\_param\_t::level

##### 15.5.2.0.0.51.3 uint16\_t ftm\_chnl\_pwm\_config\_param\_t::dutyValue

##### 15.5.2.0.0.51.4 uint16\_t ftm\_chnl\_pwm\_config\_param\_t::firstEdgeValue

Specifies the delay to the first edge in a PWM period. If unsure leave as 0, uint of this value is timer ticks.

### 15.5.3 struct ftm\_dual\_edge\_capture\_param\_t

#### Data Fields

- **ftm\_dual\_edge\_capture\_mode\_t mode**  
*Dual Edge Capture mode.*
- **ftm\_input\_capture\_edge\_t currChanEdgeMode**  
*Input capture edge select for channel n.*
- **ftm\_input\_capture\_edge\_t nextChanEdgeMode**  
*Input capture edge select for channel n+1.*

### 15.5.4 struct ftm\_phase\_params\_t

#### Data Fields

- **bool enablePhaseFilter**  
*True: enable phase filter; false: disable filter.*
- **uint32\_t phaseFilterVal**

- **ftm\_phase\_polarity\_t phasePolarity**  
*Phase polarity.*
- Filter value, used only if phase filter is enabled.*

### 15.5.5 struct ftm\_fault\_param\_t

#### Data Fields

- bool **enableFaultInput**  
*True: Fault input is enabled; false: Fault input is disabled.*
- bool **faultLevel**  
*True: Fault polarity is active low; in other words, '0' indicates a fault; False: Fault polarity is active high.*
- bool **useFaultFilter**  
*True: Use the filtered fault signal; False: Use the direct path from fault input.*

### 15.5.6 struct ftm\_config\_t

This structure holds the configuration settings for the FTM peripheral. To initialize this structure to reasonable defaults, call the [FTM\\_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

- **ftm\_clock\_prescale\_t prescale**  
*FTM clock prescale value.*
- **ftm\_bdm\_mode\_t bdmMode**  
*FTM behavior in BDM mode.*
- uint32\_t **pwmSyncMode**  
*Synchronization methods to use to update buffered registers; Multiple update modes can be used by providing an OR'ed list of options available in enumeration [ftm\\_pwm\\_sync\\_method\\_t](#).*
- uint32\_t **reloadPoints**  
*FTM reload points; When using this, the PWM synchronization is not required.*
- **ftm\_fault\_mode\_t faultMode**  
*FTM fault control mode.*
- uint8\_t **faultFilterValue**  
*Fault input filter value.*
- **ftm\_deadtime\_prescale\_t deadTimePrescale**  
*The dead time prescalar value.*
- uint32\_t **deadTimeValue**  
*The dead time value deadTimeValue's available range is 0-1023 when register has DTVALEX, otherwise its available range is 0-63.*
- uint32\_t **extTriggers**  
*External triggers to enable.*
- uint8\_t **chnlInitState**

## Enumeration Type Documentation

- `uint8_t chnlPolarity`  
*Defines the initialization value of the channels in OUTINT register.*
- `bool useGlobalTimeBase`  
*Defines the output polarity of the channels in POL register.*  
*True: Use of an external global time base is enabled; False: disabled.*

### 15.5.6.0.0.52 Field Documentation

#### 15.5.6.0.0.52.1 `uint32_t ftm_config_t::pwmSyncMode`

#### 15.5.6.0.0.52.2 `uint32_t ftm_config_t::reloadPoints`

Multiple reload points can be used by providing an OR'ed list of options available in enumeration [ftm\\_reload\\_point\\_t](#).

#### 15.5.6.0.0.52.3 `uint32_t ftm_config_t::deadTimeValue`

#### 15.5.6.0.0.52.4 `uint32_t ftm_config_t::extTriggers`

Multiple trigger sources can be enabled by providing an OR'ed list of options available in enumeration [ftm\\_external\\_trigger\\_t](#).

## 15.6 Macro Definition Documentation

### 15.6.1 `#define FSL_FTM_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

## 15.7 Enumeration Type Documentation

### 15.7.1 `enum ftm_chnl_t`

Note

Actual number of available channels is SoC dependent

Enumerator

- `kFTM_Chnl_0` FTM channel number 0.
- `kFTM_Chnl_1` FTM channel number 1.
- `kFTM_Chnl_2` FTM channel number 2.
- `kFTM_Chnl_3` FTM channel number 3.
- `kFTM_Chnl_4` FTM channel number 4.
- `kFTM_Chnl_5` FTM channel number 5.
- `kFTM_Chnl_6` FTM channel number 6.
- `kFTM_Chnl_7` FTM channel number 7.

### 15.7.2 enum ftm\_fault\_input\_t

Enumerator

- kFTM\_Fault\_0* FTM fault 0 input pin.
- kFTM\_Fault\_1* FTM fault 1 input pin.
- kFTM\_Fault\_2* FTM fault 2 input pin.
- kFTM\_Fault\_3* FTM fault 3 input pin.

### 15.7.3 enum ftm\_pwm\_mode\_t

Enumerator

- kFTM\_EdgeAlignedPwm* Edge-aligned PWM.
- kFTM\_CenterAlignedPwm* Center-aligned PWM.
- kFTM\_CombinedPwm* Combined PWM.

### 15.7.4 enum ftm\_pwm\_level\_select\_t

Enumerator

- kFTM\_NoPwmSignal* No PWM output on pin.
- kFTM\_LowTrue* Low true pulses.
- kFTM\_HighTrue* High true pulses.

### 15.7.5 enum ftm\_output\_compare\_mode\_t

Enumerator

- kFTM\_NoOutputSignal* No channel output when counter reaches CnV.
- kFTM\_ToggleOnMatch* Toggle output.
- kFTM\_ClearOnMatch* Clear output.
- kFTM\_SetOnMatch* Set output.

### 15.7.6 enum ftm\_input\_capture\_edge\_t

Enumerator

- kFTM\_RisingEdge* Capture on rising edge only.
- kFTM\_FallingEdge* Capture on falling edge only.
- kFTM\_RiseAndFallEdge* Capture on rising or falling edge.

## Enumeration Type Documentation

### 15.7.7 enum ftm\_dual\_edge\_capture\_mode\_t

Enumerator

*kFTM\_OneShot* One-shot capture mode.

*kFTM\_Continuous* Continuous capture mode.

### 15.7.8 enum ftm\_quad\_decode\_mode\_t

Enumerator

*kFTM\_QuadPhaseEncode* Phase A and Phase B encoding mode.

*kFTM\_QuadCountAndDir* Count and direction encoding mode.

### 15.7.9 enum ftm\_phase\_polarity\_t

Enumerator

*kFTM\_QuadPhaseNormal* Phase input signal is not inverted.

*kFTM\_QuadPhaseInvert* Phase input signal is inverted.

### 15.7.10 enum ftm\_deadtime\_prescale\_t

Enumerator

*kFTM\_Deadtime\_Prescale\_1* Divide by 1.

*kFTM\_Deadtime\_Prescale\_4* Divide by 4.

*kFTM\_Deadtime\_Prescale\_16* Divide by 16.

### 15.7.11 enum ftm\_clock\_source\_t

Enumerator

*kFTM\_SystemClock* System clock selected.

*kFTM\_FixedClock* Fixed frequency clock.

*kFTM\_ExternalClock* External clock.

### 15.7.12 enum ftm\_clock\_prescale\_t

Enumerator

- kFTM\_Prescale\_Divide\_1* Divide by 1.
- kFTM\_Prescale\_Divide\_2* Divide by 2.
- kFTM\_Prescale\_Divide\_4* Divide by 4.
- kFTM\_Prescale\_Divide\_8* Divide by 8.
- kFTM\_Prescale\_Divide\_16* Divide by 16.
- kFTM\_Prescale\_Divide\_32* Divide by 32.
- kFTM\_Prescale\_Divide\_64* Divide by 64.
- kFTM\_Prescale\_Divide\_128* Divide by 128.

### 15.7.13 enum ftm\_bdm\_mode\_t

Enumerator

- kFTM\_BdmMode\_0* FTM counter stopped, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers.
- kFTM\_BdmMode\_1* FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are forced to their safe value , writes to MOD,CNTIN and C(n)V registers bypass the register buffers.
- kFTM\_BdmMode\_2* FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are frozen when chip enters in BDM mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers.
- kFTM\_BdmMode\_3* FTM counter in functional mode, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers is in fully functional mode.

### 15.7.14 enum ftm\_fault\_mode\_t

Enumerator

- kFTM\_Fault\_Disable* Fault control is disabled for all channels.
- kFTM\_Fault\_EvenChnls* Enabled for even channels only(0,2,4,6) with manual fault clearing.
- kFTM\_Fault\_AllChnlsMan* Enabled for all channels with manual fault clearing.
- kFTM\_Fault\_AllChnlsAuto* Enabled for all channels with automatic fault clearing.

### 15.7.15 enum ftm\_external\_trigger\_t

## Enumeration Type Documentation

Note

Actual available external trigger sources are SoC-specific

Enumerator

- kFTM\_Chnl0Trigger*** Generate trigger when counter equals chnl 0 CnV reg.
- kFTM\_Chnl1Trigger*** Generate trigger when counter equals chnl 1 CnV reg.
- kFTM\_Chnl2Trigger*** Generate trigger when counter equals chnl 2 CnV reg.
- kFTM\_Chnl3Trigger*** Generate trigger when counter equals chnl 3 CnV reg.
- kFTM\_Chnl4Trigger*** Generate trigger when counter equals chnl 4 CnV reg.
- kFTM\_Chnl5Trigger*** Generate trigger when counter equals chnl 5 CnV reg.
- kFTM\_InitTrigger*** Generate Trigger when counter is updated with CNTIN.
- kFTM\_ReloadInitTrigger*** Available on certain SoC's, trigger on reload point.

### 15.7.16 enum ftm\_pwm\_sync\_method\_t

Enumerator

- kFTM\_SoftwareTrigger*** Software triggers PWM sync.
- kFTM\_HardwareTrigger\_0*** Hardware trigger 0 causes PWM sync.
- kFTM\_HardwareTrigger\_1*** Hardware trigger 1 causes PWM sync.
- kFTM\_HardwareTrigger\_2*** Hardware trigger 2 causes PWM sync.

### 15.7.17 enum ftm\_reload\_point\_t

Note

Actual available reload points are SoC-specific

Enumerator

- kFTM\_Chnl0Match*** Channel 0 match included as a reload point.
- kFTM\_Chnl1Match*** Channel 1 match included as a reload point.
- kFTM\_Chnl2Match*** Channel 2 match included as a reload point.
- kFTM\_Chnl3Match*** Channel 3 match included as a reload point.
- kFTM\_Chnl4Match*** Channel 4 match included as a reload point.
- kFTM\_Chnl5Match*** Channel 5 match included as a reload point.
- kFTM\_Chnl6Match*** Channel 6 match included as a reload point.
- kFTM\_Chnl7Match*** Channel 7 match included as a reload point.
- kFTM\_CntMax*** Use in up-down count mode only, reload when counter reaches the maximum value.

***kFTM\_CntMin*** Use in up-down count mode only, reload when counter reaches the minimum value.

***kFTM\_HalfCycMatch*** Available on certain SoC's, half cycle match reload point.

**15.7.18 enum ftm\_interrupt\_enable\_t**

Note

Actual available interrupts are SoC-specific

Enumerator

- kFTM\_Chnl0InterruptEnable*** Channel 0 interrupt.
- kFTM\_Chnl1InterruptEnable*** Channel 1 interrupt.
- kFTM\_Chnl2InterruptEnable*** Channel 2 interrupt.
- kFTM\_Chnl3InterruptEnable*** Channel 3 interrupt.
- kFTM\_Chnl4InterruptEnable*** Channel 4 interrupt.
- kFTM\_Chnl5InterruptEnable*** Channel 5 interrupt.
- kFTM\_Chnl6InterruptEnable*** Channel 6 interrupt.
- kFTM\_Chnl7InterruptEnable*** Channel 7 interrupt.
- kFTM\_FaultInterruptEnable*** Fault interrupt.
- kFTM\_TimeOverflowInterruptEnable*** Time overflow interrupt.
- kFTM\_ReloadInterruptEnable*** Reload interrupt; Available only on certain SoC's.

**15.7.19 enum ftm\_status\_flags\_t**

Note

Actual available flags are SoC-specific

Enumerator

- kFTM\_Chnl0Flag*** Channel 0 Flag.
- kFTM\_Chnl1Flag*** Channel 1 Flag.
- kFTM\_Chnl2Flag*** Channel 2 Flag.
- kFTM\_Chnl3Flag*** Channel 3 Flag.
- kFTM\_Chnl4Flag*** Channel 4 Flag.
- kFTM\_Chnl5Flag*** Channel 5 Flag.
- kFTM\_Chnl6Flag*** Channel 6 Flag.
- kFTM\_Chnl7Flag*** Channel 7 Flag.
- kFTM\_FaultFlag*** Fault Flag.
- kFTM\_TimeOverflowFlag*** Time overflow Flag.
- kFTM\_ChnlTriggerFlag*** Channel trigger Flag.
- kFTM\_ReloadFlag*** Reload Flag; Available only on certain SoC's.

## Function Documentation

### 15.7.20 enum \_ftm\_quad\_decoder\_flags

Enumerator

**kFTM\_QquadDecoderCountingIncreaseFlag** Counting direction is increasing (FTM counter increment), or the direction is decreasing.

**kFTM\_QquadDecoderCountingOverflowOnTopFlag** Indicates if the TOF bit was set on the top or the bottom of counting.

## 15.8 Function Documentation

### 15.8.1 status\_t FTM\_Init ( FTM\_Type \* *base*, const ftm\_config\_t \* *config* )

Note

This API should be called at the beginning of the application which is using the FTM driver.

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | FTM peripheral base address                  |
| <i>config</i> | Pointer to the user configuration structure. |

Returns

kStatus\_Success indicates success; Else indicates failure.

### 15.8.2 void FTM\_Deinit ( FTM\_Type \* *base* )

Parameters

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

### 15.8.3 void FTM\_GetDefaultConfig ( ftm\_config\_t \* *config* )

The default values are:

```
* config->prescale = kFTM_Prescale_Divide_1;
* config->bdmMode = kFTM_BdmMode_0;
* config->pwmSyncMode = kFTM_SoftwareTrigger;
* config->reloadPoints = 0;
* config->faultMode = kFTM_Fault_Disable;
* config->faultFilterValue = 0;
* config->deadTimePrescale = kFTM_Deadtime_Prescale_1;
* config->deadTimeValue = 0;
```

```
* config->extTriggers = 0;
* config->chnlInitState = 0;
* config->chnlPolarity = 0;
* config->useGlobalTimeBase = false;
*
```

## Parameters

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

#### 15.8.4 **status\_t FTM\_SetupPwm ( FTM\_Type \* *base*, const ftm\_chnl\_pwm\_signal\_param\_t \* *chnlParams*, uint8\_t *numOfChnls*, ftm\_pwm\_mode\_t *mode*, uint32\_t *pwmFreq\_Hz*, uint32\_t *srcClock\_Hz* )**

Call this function to configure the PWM signal period, mode, duty cycle, and edge. Use this function to configure all FTM channels that are used to output a PWM signal.

## Parameters

|                    |                                                                                     |
|--------------------|-------------------------------------------------------------------------------------|
| <i>base</i>        | FTM peripheral base address                                                         |
| <i>chnlParams</i>  | Array of PWM channel parameters to configure the channel(s)                         |
| <i>numOfChnls</i>  | Number of channels to configure; This should be the size of the array passed in     |
| <i>mode</i>        | PWM operation mode, options available in enumeration <a href="#">ftm_pwm_mode_t</a> |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz                                                          |
| <i>srcClock_Hz</i> | FTM counter clock in Hz                                                             |

## Returns

kStatus\_Success if the PWM setup was successful kStatus\_Error on failure

#### 15.8.5 **void FTM\_UpdatePwmDutycycle ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, ftm\_pwm\_mode\_t *currentPwmMode*, uint8\_t *dutyCyclePercent* )**

## Parameters

## Function Documentation

|                          |                                                                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | FTM peripheral base address                                                                                                       |
| <i>chnlNumber</i>        | The channel/channel pair number. In combined mode, this represents the channel pair number                                        |
| <i>currentPwm-Mode</i>   | The current PWM mode set during PWM setup                                                                                         |
| <i>dutyCycle-Percent</i> | New PWM pulse width; The value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |

**15.8.6 void FTM\_UpdateChnlEdgeLevelSelect ( FTM\_Type \* *base*, **ftm\_chnl\_t chnlNumber, uint8\_t level** )**

Parameters

|                   |                                                                                                                                                   |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                                                                                                       |
| <i>chnlNumber</i> | The channel number                                                                                                                                |
| <i>level</i>      | The level to be set to the ELSnB:ELSnA field; Valid values are 00, 01, 10, 11. See the Kinetis SoC reference manual for details about this field. |

**15.8.7 status\_t FTM\_SetupPwmMode ( FTM\_Type \* *base*, const **ftm\_chnl\_pwm\_config\_param\_t \* chnlParams, uint8\_t numOfChnls, ftm\_pwm\_mode\_t mode** )**

Call this function to configure the PWM signal mode, duty cycle in ticks, and edge. Use this function to configure all FTM channels that are used to output a PWM signal. Please note that: This API is similar with [FTM\\_SetupPwm\(\)](#) API, but will not set the timer period, and this API will set channel match value in timer ticks, not period percent.

Parameters

|                   |                                                                                     |
|-------------------|-------------------------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                                         |
| <i>chnlParams</i> | Array of PWM channel parameters to configure the channel(s)                         |
| <i>numOfChnls</i> | Number of channels to configure; This should be the size of the array passed in     |
| <i>mode</i>       | PWM operation mode, options available in enumeration <a href="#">ftm_pwm_mode_t</a> |

Returns

kStatus\_Success if the PWM setup was successful kStatus\_Error on failure

### 15.8.8 void FTM\_SetupInputCapture ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlNumber*, **ftm\_input\_capture\_edge\_t** *captureMode*, **uint32\_t** *filterValue* )

When the edge specified in the captureMode argument occurs on the channel, the FTM counter is captured into the CnV register. The user has to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only for channels 0, 1, 2, 3.

Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>base</i>        | FTM peripheral base address                                                 |
| <i>chnlNumber</i>  | The channel number                                                          |
| <i>captureMode</i> | Specifies which edge to capture                                             |
| <i>filterValue</i> | Filter value, specify 0 to disable filter. Available only for channels 0-3. |

### 15.8.9 void FTM\_SetupOutputCompare ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlNumber*, **ftm\_output\_compare\_mode\_t** *compareMode*, **uint32\_t** *compareValue* )

When the FTM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| <i>base</i>         | FTM peripheral base address                                            |
| <i>chnlNumber</i>   | The channel number                                                     |
| <i>compareMode</i>  | Action to take on the channel output when the compare condition is met |
| <i>compareValue</i> | Value to be programmed in the CnV register.                            |

### 15.8.10 void FTM\_SetupDualEdgeCapture ( **FTM\_Type** \* *base*, **ftm\_chnl\_t** *chnlPairNumber*, **const ftm\_dual\_edge\_capture\_param\_t** \* *edgeParam*, **uint32\_t** *filterValue* )

This function sets up the dual edge capture mode on a channel pair. The capture edge for the channel pair and the capture mode (one-shot or continuous) is specified in the parameter argument. The filter function is disabled if the filterVal argument passed is zero. The filter function is available only on channels 0 and 2. The user has to read the channel CnV registers separately to get the capture values.

## Function Documentation

Parameters

|                        |                                                                                     |
|------------------------|-------------------------------------------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                                                         |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3                                 |
| <i>edgeParam</i>       | Sets up the dual edge capture function                                              |
| <i>filterValue</i>     | Filter value, specify 0 to disable filter. Available only for channel pair 0 and 1. |

**15.8.11 void FTM\_SetupFault ( FTM\_Type \* *base*, ftm\_fault\_input\_t *faultNumber*, const ftm\_fault\_param\_t \* *faultParams* )**

FTM can have up to 4 fault inputs. This function sets up fault parameters, fault level, and a filter.

Parameters

|                    |                                          |
|--------------------|------------------------------------------|
| <i>base</i>        | FTM peripheral base address              |
| <i>faultNumber</i> | FTM fault to configure.                  |
| <i>faultParams</i> | Parameters passed in to set up the fault |

**15.8.12 void FTM\_EnableInterrupts ( FTM\_Type \* *base*, uint32\_t *mask* )**

Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | FTM peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">ftm_interrupt_enable_t</a> |

**15.8.13 void FTM\_DisableInterrupts ( FTM\_Type \* *base*, uint32\_t *mask* )**

Parameters

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

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">ftm_interrupt_enable_t</a> |
|-------------|---------------------------------------------------------------------------------------------------------------------|

**15.8.14 uint32\_t FTM\_GetEnabledInterrupts ( FTM\_Type \* *base* )**

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [ftm\\_interrupt\\_enable\\_t](#)

**15.8.15 uint32\_t FTM\_GetStatusFlags ( FTM\_Type \* *base* )**

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [ftm\\_status\\_flags\\_t](#)

**15.8.16 void FTM\_ClearStatusFlags ( FTM\_Type \* *base*, uint32\_t *mask* )**

Parameters

|             |                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | FTM peripheral base address                                                                                      |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">ftm_status_flags_t</a> |

**15.8.17 static void FTM\_SetTimerPeriod ( FTM\_Type \* *base*, uint32\_t *ticks* )  
[inline], [static]**

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

## Function Documentation

Note

1. This API allows the user to use the FTM module as a timer. Do not mix usage of this API with FTM's PWM setup API's.
2. Call the utility macros provided in the fsl\_common.h to convert usec or msec to ticks.

Parameters

|              |                                                                            |
|--------------|----------------------------------------------------------------------------|
| <i>base</i>  | FTM peripheral base address                                                |
| <i>ticks</i> | A timer period in units of ticks, which should be equal or greater than 1. |

### **15.8.18 static uint32\_t FTM\_GetCurrentTimerCount ( FTM\_Type \* *base* ) [inline], [static]**

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the fsl\_common.h to convert ticks to usec or msec.

Parameters

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

Returns

The current counter value in ticks

### **15.8.19 static void FTM\_StartTimer ( FTM\_Type \* *base*, ftm\_clock\_source\_t *clockSource* ) [inline], [static]**

Parameters

|                    |                                                                              |
|--------------------|------------------------------------------------------------------------------|
| <i>base</i>        | FTM peripheral base address                                                  |
| <i>clockSource</i> | FTM clock source; After the clock source is set, the counter starts running. |

### **15.8.20 static void FTM\_StopTimer ( FTM\_Type \* *base* ) [inline], [static]**

Parameters

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

**15.8.21 static void FTM\_SetSoftwareCtrlEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, bool *value* ) [inline], [static]**

Parameters

|                   |                                                                                                                            |
|-------------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                                                                                |
| <i>chnlNumber</i> | Channel to be enabled or disabled                                                                                          |
| <i>value</i>      | true: channel output is affected by software output control false: channel output is unaffected by software output control |

**15.8.22 static void FTM\_SetSoftwareCtrlVal ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, bool *value* ) [inline], [static]**

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | FTM peripheral base address.  |
| <i>chnlNumber</i> | Channel to be configured      |
| <i>value</i>      | true to set 1, false to set 0 |

**15.8.23 static void FTM\_SetGlobalTimeBaseOutputEnable ( FTM\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FTM peripheral base address      |
| <i>enable</i> | true to enable, false to disable |

**15.8.24 static void FTM\_SetOutputMask ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, bool *mask* ) [inline], [static]**

## Function Documentation

Parameters

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                            |
| <i>chnlNumber</i> | Channel to be configured                                               |
| <i>mask</i>       | true: masked, channel is forced to its inactive state; false: unmasked |

### 15.8.25 static void FTM\_SetPwmOutputEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, bool *value* ) [inline], [static]

To enable the PWM channel output call this function with val=true. For input mode, call this function with val=false.

Parameters

|                   |                                                                    |
|-------------------|--------------------------------------------------------------------|
| <i>base</i>       | FTM peripheral base address                                        |
| <i>chnlNumber</i> | Channel to be configured                                           |
| <i>value</i>      | true: enable output; false: output is disabled, used in input mode |

### 15.8.26 static void FTM\_SetFaultControlEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]

Parameters

|                        |                                                                           |
|------------------------|---------------------------------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                                               |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3                       |
| <i>value</i>           | true: Enable fault control for this channel pair; false: No fault control |

### 15.8.27 static void FTM\_SetDeadTimeEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]

Parameters

|                        |                                                                           |
|------------------------|---------------------------------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                                               |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3                       |
| <i>value</i>           | true: Insert dead time in this channel pair; false: No dead time inserted |

**15.8.28 static void FTM\_SetComplementaryEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]**

Parameters

|                        |                                                                    |
|------------------------|--------------------------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                                        |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3                |
| <i>value</i>           | true: enable complementary mode; false: disable complementary mode |

**15.8.29 static void FTM\_SetInvertEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]**

Parameters

|                        |                                                     |
|------------------------|-----------------------------------------------------|
| <i>base</i>            | FTM peripheral base address                         |
| <i>chnlPair-Number</i> | The FTM channel pair number; options are 0, 1, 2, 3 |
| <i>value</i>           | true: enable inverting; false: disable inverting    |

**15.8.30 void FTM\_SetupQuadDecode ( FTM\_Type \* *base*, const ftm\_phase\_params\_t \* *phaseAParams*, const ftm\_phase\_params\_t \* *phaseBParams*, ftm\_quad\_decode\_mode\_t *quadMode* )**

Parameters

|                     |                                                       |
|---------------------|-------------------------------------------------------|
| <i>base</i>         | FTM peripheral base address                           |
| <i>phaseAParams</i> | Phase A configuration parameters                      |
| <i>phaseBParams</i> | Phase B configuration parameters                      |
| <i>quadMode</i>     | Selects encoding mode used in quadrature decoder mode |

**15.8.31 static uint32\_t FTM\_GetQuadDecoderFlags ( FTM\_Type \* *base* ) [inline], [static]**

## Function Documentation

Parameters

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

Returns

Flag mask of FTM Quad Decoder, see [\\_ftm\\_quad\\_decoder\\_flags](#).

### 15.8.32 static void FTM\_SetQuadDecoderModuloValue ( FTM\_Type \* *base*, uint32\_t *startValue*, uint32\_t *overValue* ) [inline], [static]

The modulo values configure the minimum and maximum values that the Quad decoder counter can reach. After the counter goes over, the counter value goes to the other side and decrease/increase again.

Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>base</i>       | FTM peripheral base address.                   |
| <i>startValue</i> | The low limit value for Quad Decoder counter.  |
| <i>overValue</i>  | The high limit value for Quad Decoder counter. |

### 15.8.33 static uint32\_t FTM\_GetQuadDecoderCounterValue ( FTM\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Current quad Decoder counter value.

### 15.8.34 static void FTM\_ClearQuadDecoderCounterValue ( FTM\_Type \* *base* ) [inline], [static]

The counter is set as the initial value.

Parameters

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

**15.8.35 static void FTM\_SetSoftwareTrigger ( FTM\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | FTM peripheral base address                                                 |
| <i>enable</i> | true: software trigger is selected, false: software trigger is not selected |

**15.8.36 static void FTM\_SetWriteProtection ( FTM\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FTM peripheral base address                                            |
| <i>enable</i> | true: Write-protection is enabled, false: Write-protection is disabled |

## Function Documentation

# Chapter 16

## GPT: General Purpose Timer

### 16.1 Overview

The MCUXpresso SDK provides a driver for the General Purpose Timer (GPT) of MCUXpresso SDK devices.

### 16.2 Function groups

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

#### 16.2.1 Initialization and deinitialization

The function [GPT\\_Init\(\)](#) initializes the gpt with specified configurations. The function [GPT\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the restart/free-run mode and input selection when running.

The function [GPT\\_Deinit\(\)](#) stops the timer and turns off the module clock.

### 16.3 Typical use case

#### 16.3.1 GPT interrupt example

Set up a channel to trigger a periodic interrupt after every 1 second. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/gpt

## Data Structures

- struct [gpt\\_config\\_t](#)  
*Structure to configure the running mode. [More...](#)*

## Enumerations

- enum [gpt\\_clock\\_source\\_t](#) {  
    kGPT\_ClockSource\_Off = 0U,  
    kGPT\_ClockSource\_Periph = 1U,  
    kGPT\_ClockSource\_HighFreq = 2U,  
    kGPT\_ClockSource\_Ext = 3U,  
    kGPT\_ClockSource\_LowFreq = 4U,  
    kGPT\_ClockSource\_Osc = 5U }

*List of clock sources.*

## Typical use case

- enum `gpt_input_capture_channel_t` {  
    `kGPT_InputCapture_Channel1` = 0U,  
    `kGPT_InputCapture_Channel2` = 1U }  
    *List of input capture channel number.*
- enum `gpt_input_operation_mode_t` {  
    `kGPT_InputOperation_Disabled` = 0U,  
    `kGPT_InputOperation_RiseEdge` = 1U,  
    `kGPT_InputOperation_FallEdge` = 2U,  
    `kGPT_InputOperation_BothEdge` = 3U }  
    *List of input capture operation mode.*
- enum `gpt_output_compare_channel_t` {  
    `kGPT_OutputCompare_Channel1` = 0U,  
    `kGPT_OutputCompare_Channel2` = 1U,  
    `kGPT_OutputCompare_Channel3` = 2U }  
    *List of output compare channel number.*
- enum `gpt_output_operation_mode_t` {  
    `kGPT_OutputOperation_Disconnected` = 0U,  
    `kGPT_OutputOperation_Toggle` = 1U,  
    `kGPT_OutputOperation_Clear` = 2U,  
    `kGPT_OutputOperation_Set` = 3U,  
    `kGPT_OutputOperation_Activelow` = 4U }  
    *List of output compare operation mode.*
- enum `gpt_interrupt_enable_t` {  
    `kGPT_OutputCompare1InterruptEnable` = `GPT_IR_OF1IE_MASK`,  
    `kGPT_OutputCompare2InterruptEnable` = `GPT_IR_OF2IE_MASK`,  
    `kGPT_OutputCompare3InterruptEnable` = `GPT_IR_OF3IE_MASK`,  
    `kGPT_InputCapture1InterruptEnable` = `GPT_IR_IF1IE_MASK`,  
    `kGPT_InputCapture2InterruptEnable` = `GPT_IR_IF2IE_MASK`,  
    `kGPT_RollOverFlagInterruptEnable` = `GPT_IR_ROVIE_MASK` }  
    *List of GPT interrupts.*
- enum `gpt_status_flag_t` {  
    `kGPT_OutputCompare1Flag` = `GPT_SR_OF1_MASK`,  
    `kGPT_OutputCompare2Flag` = `GPT_SR_OF2_MASK`,  
    `kGPT_OutputCompare3Flag` = `GPT_SR_OF3_MASK`,  
    `kGPT_InputCapture1Flag` = `GPT_SR_IF1_MASK`,  
    `kGPT_InputCapture2Flag` = `GPT_SR_IF2_MASK`,  
    `kGPT_RollOverFlag` = `GPT_SR_ROV_MASK` }  
    *Status flag.*

## Driver version

- #define `FSL_GPT_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- void `GPT_Init` (`GPT_Type` \*base, const `gpt_config_t` \*initConfig)

- Initialize GPT to reset state and initialize running mode.
- void **GPT\_Deinit** (GPT\_Type \*base)  
*Disables the module and gates the GPT clock.*
- void **GPT\_GetDefaultConfig** (gpt\_config\_t \*config)  
*Fills in the GPT configuration structure with default settings.*

## Software Reset

- static void **GPT\_SoftwareReset** (GPT\_Type \*base)  
*Software reset of GPT module.*

## Clock source and frequency control

- static void **GPT\_SetClockSource** (GPT\_Type \*base, gpt\_clock\_source\_t source)  
*Set clock source of GPT.*
- static gpt\_clock\_source\_t **GPT\_GetClockSource** (GPT\_Type \*base)  
*Get clock source of GPT.*
- static void **GPT\_SetClockDivider** (GPT\_Type \*base, uint32\_t divider)  
*Set pre scaler of GPT.*
- static uint32\_t **GPT\_GetClockDivider** (GPT\_Type \*base)  
*Get clock divider in GPT module.*
- static void **GPT\_SetOscClockDivider** (GPT\_Type \*base, uint32\_t divider)  
*OSC 24M pre-scaler before selected by clock source.*
- static uint32\_t **GPT\_GetOscClockDivider** (GPT\_Type \*base)  
*Get OSC 24M clock divider in GPT module.*

## Timer Start and Stop

- static void **GPT\_StartTimer** (GPT\_Type \*base)  
*Start GPT timer.*
- static void **GPT\_StopTimer** (GPT\_Type \*base)  
*Stop GPT timer.*

## Read the timer period

- static uint32\_t **GPT\_GetCurrentTimerCount** (GPT\_Type \*base)  
*Reads the current GPT counting value.*

## GPT Input/Output Signal Control

- static void **GPT\_SetInputOperationMode** (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel, gpt\_input\_operation\_mode\_t mode)  
*Set GPT operation mode of input capture channel.*
- static gpt\_input\_operation\_mode\_t **GPT\_GetInputOperationMode** (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel)  
*Get GPT operation mode of input capture channel.*
- static uint32\_t **GPT\_GetInputCaptureValue** (GPT\_Type \*base, gpt\_input\_capture\_channel\_t channel)  
*Get GPT input capture value of certain channel.*

## Data Structure Documentation

- static void `GPT_SetOutputOperationMode` (GPT\_Type \*base, `gpt_output_compare_channel_t` channel, `gpt_output_operation_mode_t` mode)  
*Set GPT operation mode of output compare channel.*
- static `gpt_output_operation_mode_t` `GPT_GetOutputOperationMode` (GPT\_Type \*base, `gpt_output_compare_channel_t` channel)  
*Get GPT operation mode of output compare channel.*
- static void `GPT_SetOutputCompareValue` (GPT\_Type \*base, `gpt_output_compare_channel_t` channel, uint32\_t value)  
*Set GPT output compare value of output compare channel.*
- static uint32\_t `GPT_GetOutputCompareValue` (GPT\_Type \*base, `gpt_output_compare_channel_t` channel)  
*Get GPT output compare value of output compare channel.*
- static void `GPT_ForceOutput` (GPT\_Type \*base, `gpt_output_compare_channel_t` channel)  
*Force GPT output action on output compare channel, ignoring comparator.*

## GPT Interrupt and Status Interface

- static void `GPT_EnableInterrupts` (GPT\_Type \*base, uint32\_t mask)  
*Enables the selected GPT interrupts.*
- static void `GPT_DisableInterrupts` (GPT\_Type \*base, uint32\_t mask)  
*Disables the selected GPT interrupts.*
- static uint32\_t `GPT_GetEnabledInterrupts` (GPT\_Type \*base)  
*Gets the enabled GPT interrupts.*

## Status Interface

- static uint32\_t `GPT_GetStatusFlags` (GPT\_Type \*base, `gpt_status_flag_t` flags)  
*Get GPT status flags.*
- static void `GPT_ClearStatusFlags` (GPT\_Type \*base, `gpt_status_flag_t` flags)  
*Clears the GPT status flags.*

## 16.4 Data Structure Documentation

### 16.4.1 struct gpt\_config\_t

#### Data Fields

- `gpt_clock_source_t` `clockSource`  
*clock source for GPT module.*
- `uint32_t` `divider`  
*clock divider (prescaler+1) from clock source to counter.*
- `bool` `enableFreeRun`  
*true: FreeRun mode, false: Restart mode.*
- `bool` `enableRunInWait`  
*GPT enabled in wait mode.*
- `bool` `enableRunInStop`  
*GPT enabled in stop mode.*
- `bool` `enableRunInDoze`  
*GPT enabled in doze mode.*

- bool `enableRunInDbg`  
*GPT enabled in debug mode.*
- bool `enableMode`  
`true:` counter reset to 0 when enabled;  
`false:` counter retain its value when enabled.

#### 16.4.1.0.0.53 Field Documentation

16.4.1.0.0.53.1 `gpt_clock_source_t gpt_config_t::clockSource`

16.4.1.0.0.53.2 `uint32_t gpt_config_t::divider`

16.4.1.0.0.53.3 `bool gpt_config_t::enableFreeRun`

16.4.1.0.0.53.4 `bool gpt_config_t::enableRunInWait`

16.4.1.0.0.53.5 `bool gpt_config_t::enableRunInStop`

16.4.1.0.0.53.6 `bool gpt_config_t::enableRunInDoze`

16.4.1.0.0.53.7 `bool gpt_config_t::enableRunInDbg`

16.4.1.0.0.53.8 `bool gpt_config_t::enableMode`

### 16.5 Enumeration Type Documentation

#### 16.5.1 enum `gpt_clock_source_t`

Note

Actual number of clock sources is SoC dependent

Enumerator

*kGPT\_ClockSource\_Off* GPT Clock Source Off.

*kGPT\_ClockSource\_Periph* GPT Clock Source from Peripheral Clock.

*kGPT\_ClockSource\_HighFreq* GPT Clock Source from High Frequency Reference Clock.

*kGPT\_ClockSource\_Ext* GPT Clock Source from external pin.

*kGPT\_ClockSource\_LowFreq* GPT Clock Source from Low Frequency Reference Clock.

*kGPT\_ClockSource\_Osc* GPT Clock Source from Crystal oscillator.

#### 16.5.2 enum `gpt_input_capture_channel_t`

Enumerator

*kGPT\_InputCapture\_Channel1* GPT Input Capture Channel1.

*kGPT\_InputCapture\_Channel2* GPT Input Capture Channel2.

## Enumeration Type Documentation

### 16.5.3 enum gpt\_input\_operation\_mode\_t

Enumerator

*kGPT\_InputOperation\_Disabled* Don't capture.

*kGPT\_InputOperation\_RiseEdge* Capture on rising edge of input pin.

*kGPT\_InputOperation\_FallEdge* Capture on falling edge of input pin.

*kGPT\_InputOperation\_BothEdge* Capture on both edges of input pin.

### 16.5.4 enum gpt\_output\_compare\_channel\_t

Enumerator

*kGPT\_OutputCompare\_Channel1* Output Compare Channel1.

*kGPT\_OutputCompare\_Channel2* Output Compare Channel2.

*kGPT\_OutputCompare\_Channel3* Output Compare Channel3.

### 16.5.5 enum gpt\_output\_operation\_mode\_t

Enumerator

*kGPT\_OutputOperation\_Disconnected* Don't change output pin.

*kGPT\_OutputOperation\_Toggle* Toggle output pin.

*kGPT\_OutputOperation\_Clear* Set output pin low.

*kGPT\_OutputOperation\_Set* Set output pin high.

*kGPT\_OutputOperation\_Activelow* Generate a active low pulse on output pin.

### 16.5.6 enum gpt\_interrupt\_enable\_t

Enumerator

*kGPT\_OutputCompare1InterruptEnable* Output Compare Channel1 interrupt enable.

*kGPT\_OutputCompare2InterruptEnable* Output Compare Channel2 interrupt enable.

*kGPT\_OutputCompare3InterruptEnable* Output Compare Channel3 interrupt enable.

*kGPT\_InputCapture1InterruptEnable* Input Capture Channel1 interrupt enable.

*kGPT\_InputCapture2InterruptEnable* Input Capture Channel1 interrupt enable.

*kGPT\_RollOverFlagInterruptEnable* Counter rolled over interrupt enable.

### 16.5.7 enum gpt\_status\_flag\_t

Enumerator

- kGPT\_OutputCompare1Flag* Output compare channel 1 event.
- kGPT\_OutputCompare2Flag* Output compare channel 2 event.
- kGPT\_OutputCompare3Flag* Output compare channel 3 event.
- kGPT\_InputCapture1Flag* Input Capture channel 1 event.
- kGPT\_InputCapture2Flag* Input Capture channel 2 event.
- kGPT\_RollOverFlag* Counter reaches maximum value and rolled over to 0 event.

## 16.6 Function Documentation

### 16.6.1 void GPT\_Init ( *GPT\_Type* \* *base*, *const gpt\_config\_t* \* *initConfig* )

Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>base</i>       | GPT peripheral base address.    |
| <i>initConfig</i> | GPT mode setting configuration. |

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

Parameters

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

### 16.6.3 void GPT\_GetDefaultConfig ( *gpt\_config\_t* \* *config* )

The default values are:

```
* config->clockSource = kGPT_ClockSource_Periph;
* config->divider = 1U;
* config->enableRunInStop = true;
* config->enableRunInWait = true;
* config->enableRunInDoze = false;
* config->enableRunInDbg = false;
* config->enableFreeRun = true;
* config->enableMode = true;
*
```

## Function Documentation

Parameters

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

**16.6.4 static void GPT\_SoftwareReset ( *GPT\_Type* \* *base* ) [inline], [static]**

Parameters

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

**16.6.5 static void GPT\_SetClockSource ( *GPT\_Type* \* *base*, *gpt\_clock\_source\_t source* ) [inline], [static]**

Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | GPT peripheral base address.                                               |
| <i>source</i> | Clock source (see <a href="#">gpt_clock_source_t</a> typedef enumeration). |

**16.6.6 static *gpt\_clock\_source\_t* GPT\_GetClockSource ( *GPT\_Type* \* *base* ) [inline], [static]**

Parameters

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

Returns

clock source (see [gpt\\_clock\\_source\\_t](#) typedef enumeration).

**16.6.7 static void GPT\_SetClockDivider ( *GPT\_Type* \* *base*, *uint32\_t divider* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | GPT peripheral base address. |
| <i>divider</i> | Divider of GPT (1-4096).     |

#### 16.6.8 static uint32\_t GPT\_GetClockDivider ( GPT\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

clock divider in GPT module (1-4096).

#### 16.6.9 static void GPT\_SetOscClockDivider ( GPT\_Type \* *base*, uint32\_t *divider* ) [inline], [static]

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | GPT peripheral base address. |
| <i>divider</i> | OSC Divider(1-16).           |

#### 16.6.10 static uint32\_t GPT\_GetOscClockDivider ( GPT\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

OSC clock divider in GPT module (1-16).

#### 16.6.11 static void GPT\_StartTimer ( GPT\_Type \* *base* ) [inline], [static]

## Function Documentation

Parameters

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

### 16.6.12 static void GPT\_StopTimer ( **GPT\_Type** \* *base* ) [inline], [static]

Parameters

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

### 16.6.13 static uint32\_t GPT\_GetCurrentTimerCount ( **GPT\_Type** \* *base* ) [inline], [static]

Parameters

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

Returns

Current GPT counter value.

### 16.6.14 static void GPT\_SetInputOperationMode ( **GPT\_Type** \* *base*, **gpt\_input\_capture\_channel\_t** *channel*, **gpt\_input\_operation\_mode\_t** *mode* ) [inline], [static]

Parameters

|                |                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                           |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration).             |
| <i>mode</i>    | GPT input capture operation mode (see <a href="#">gpt_input_operation_mode_t</a> typedef enumeration). |

### 16.6.15 static **gpt\_input\_operation\_mode\_t** GPT\_GetInputOperationMode ( **GPT\_Type** \* *base*, **gpt\_input\_capture\_channel\_t** *channel* ) [inline], [static]

Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                               |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration). |

Returns

GPT input capture operation mode (see [gpt\\_input\\_operation\\_mode\\_t](#) typedef enumeration).

### **16.6.16 static uint32\_t GPT\_GetInputCaptureValue ( GPT\_Type \* *base*, gpt\_input\_capture\_channel\_t *channel* ) [inline], [static]**

Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                               |
| <i>channel</i> | GPT capture channel (see <a href="#">gpt_input_capture_channel_t</a> typedef enumeration). |

Returns

GPT input capture value.

### **16.6.17 static void GPT\_SetOutputOperationMode ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel*, gpt\_output\_operation\_mode\_t *mode* ) [inline], [static]**

Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |
| <i>mode</i>    | GPT output operation mode (see <a href="#">gpt_output_operation_mode_t</a> typedef enumeration).   |

### **16.6.18 static gpt\_output\_operation\_mode\_t GPT\_GetOutputOperationMode ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel* ) [inline], [static]**

## Function Documentation

Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

Returns

GPT output operation mode (see [gpt\\_output\\_operation\\_mode\\_t](#) typedef enumeration).

**16.6.19 static void GPT\_SetOutputCompareValue ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel*, uint32\_t *value* ) [inline], [static]**

Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |
| <i>value</i>   | GPT output compare value.                                                                          |

**16.6.20 static uint32\_t GPT\_GetOutputCompareValue ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel* ) [inline], [static]**

Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

Returns

GPT output compare value.

**16.6.21 static void GPT\_ForceOutput ( GPT\_Type \* *base*, gpt\_output\_compare\_channel\_t *channel* ) [inline], [static]**

Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | GPT peripheral base address.                                                                       |
| <i>channel</i> | GPT output compare channel (see <a href="#">gpt_output_compare_channel_t</a> typedef enumeration). |

**16.6.22 static void GPT\_EnableInterrupts ( GPT\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | GPT peripheral base address.                                                                                        |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">gpt_interrupt_enable_t</a> |

**16.6.23 static void GPT\_DisableInterrupts ( GPT\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | GPT peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">gpt_interrupt_enable_t</a> |

**16.6.24 static uint32\_t GPT\_GetEnabledInterrupts ( GPT\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [gpt\\_interrupt\\_enable\\_t](#)

## Function Documentation

16.6.25 **static uint32\_t GPT\_GetStatusFlags ( GPT\_Type \* *base*, gpt\_status\_flag\_t *flags* ) [inline], [static]**

Parameters

|              |                                                                                  |
|--------------|----------------------------------------------------------------------------------|
| <i>base</i>  | GPT peripheral base address.                                                     |
| <i>flags</i> | GPT status flag mask (see <a href="#">gpt_status_flag_t</a> for bit definition). |

Returns

GPT status, each bit represents one status flag.

### 16.6.26 static void GPT\_ClearStatusFlags ( GPT\_Type \* *base*, gpt\_status\_flag\_t *flags* ) [inline], [static]

Parameters

|              |                                                                                  |
|--------------|----------------------------------------------------------------------------------|
| <i>base</i>  | GPT peripheral base address.                                                     |
| <i>flags</i> | GPT status flag mask (see <a href="#">gpt_status_flag_t</a> for bit definition). |

## Function Documentation

# Chapter 17

## GPIO: General-Purpose Input/Output Driver

### 17.1 Overview

#### Modules

- [GPIO Driver](#)

### 17.2 GPIO Driver

#### 17.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of MCUXpresso SDK devices.

#### 17.2.2 Typical use case

##### 17.2.2.1 Input Operation

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

### Data Structures

- struct `gpio_pin_config_t`  
*GPIO Init structure definition. [More...](#)*

### Enumerations

- enum `gpio_pin_direction_t` {  
  `kGPIO_DigitalInput` = 0U,  
  `kGPIO_DigitalOutput` = 1U }  
*GPIO direction definition.*
- enum `gpio_interrupt_mode_t` {  
  `kGPIO_NoIntmode` = 0U,  
  `kGPIO_IntLowLevel` = 1U,  
  `kGPIO_IntHighLevel` = 2U,  
  `kGPIO_IntRisingEdge` = 3U,  
  `kGPIO_IntFallingEdge` = 4U,  
  `kGPIO_IntRisingOrFallingEdge` = 5U }  
*GPIO interrupt mode definition.*

### Driver version

- #define `FSL_GPIO_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 1))  
*GPIO driver version 2.0.1.*

### GPIO Initialization and Configuration functions

- void `GPIO_PinInit` (GPIO\_Type \*base, uint32\_t pin, const `gpio_pin_config_t` \*Config)  
*Initializes the GPIO peripheral according to the specified parameters in the initConfig.*

## GPIO Reads and Write Functions

- void [GPIO\\_PinWrite](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void [GPIO\\_WritePinOutput](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the individual GPIO pin to logic 1 or 0.*
- static void [GPIO\\_PortSet](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_SetPinsOutput](#) (GPIO\_Type \*base, 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 mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_ClearPinsOutput](#) (GPIO\_Type \*base, 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 mask)  
*Reverses the current output logic of the multiple GPIO pins.*
- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*
- static uint32\_t [GPIO\\_ReadPinInput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the GPIO port.*

## GPIO Reads Pad Status Functions

- static uint8\_t [GPIO\\_PinReadPadStatus](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*
- static uint8\_t [GPIO\\_ReadPadStatus](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current GPIO pin pad status.*

## Interrupts and flags management functions

- void [GPIO\\_PinSetInterruptConfig](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_interrupt\\_mode\\_t](#) pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void [GPIO\\_SetPinInterruptConfig](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_interrupt\\_mode\\_t](#) pinInterruptMode)  
*Sets the current pin interrupt mode.*
- static void [GPIO\\_PortEnableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void [GPIO\\_EnableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Enables the specific pin interrupt.*
- static void [GPIO\\_PortDisableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static void [GPIO\\_DisableInterrupts](#) (GPIO\_Type \*base, uint32\_t mask)  
*Disables the specific pin interrupt.*
- static uint32\_t [GPIO\\_PortGetInterruptFlags](#) (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*
- static uint32\_t [GPIO\\_GetPinsInterruptFlags](#) (GPIO\_Type \*base)  
*Reads individual pin interrupt status.*

## GPIO Driver

- static void [GPIO\\_PortClearInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears pin interrupt flag.*
- static void [GPIO\\_ClearPinsInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears pin interrupt flag.*

### 17.2.3 Data Structure Documentation

#### 17.2.3.1 struct gpio\_pin\_config\_t

##### Data Fields

- [gpio\\_pin\\_direction\\_t](#) direction  
*Specifies the pin direction.*
- uint8\_t [outputLogic](#)  
*Set a default output logic, which has no use in input.*
- [gpio\\_interrupt\\_mode\\_t](#) interruptMode  
*Specifies the pin interrupt mode, a value of [gpio\\_interrupt\\_mode\\_t](#).*

##### 17.2.3.1.0.54 Field Documentation

###### 17.2.3.1.0.54.1 [gpio\\_pin\\_direction\\_t](#) gpio\_pin\_config\_t::direction

###### 17.2.3.1.0.54.2 [gpio\\_interrupt\\_mode\\_t](#) gpio\_pin\_config\_t::interruptMode

### 17.2.4 Macro Definition Documentation

#### 17.2.4.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 17.2.5 Enumeration Type Documentation

#### 17.2.5.1 enum gpio\_pin\_direction\_t

Enumerator

*kGPIO\_DigitalInput* Set current pin as digital input.

*kGPIO\_DigitalOutput* Set current pin as digital output.

#### 17.2.5.2 enum gpio\_interrupt\_mode\_t

Enumerator

*kGPIO\_NoIntmode* Set current pin general IO functionality.

*kGPIO\_IntLowLevel* Set current pin interrupt is low-level sensitive.

*kGPIO\_IntHighLevel* Set current pin interrupt is high-level sensitive.

*kGPIO\_IntRisingEdge* Set current pin interrupt is rising-edge sensitive.

***kGPIO\_IntFallingEdge*** Set current pin interrupt is falling-edge sensitive.

***kGPIO\_IntRisingOrFallingEdge*** Enable the edge select bit to override the ICR register's configuration.

## 17.2.6 Function Documentation

### 17.2.6.1 void GPIO\_PinInit ( ***GPIO\_Type*** \* *base*, ***uint32\_t*** *pin*, const ***gpio\_pin\_config\_t*** \* *Config* )

Parameters

|                   |                                                                                                              |
|-------------------|--------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | GPIO base pointer.                                                                                           |
| <i>pin</i>        | Specifies the pin number                                                                                     |
| <i>initConfig</i> | pointer to a <a href="#"><b>gpio_pin_config_t</b></a> structure that contains the configuration information. |

### 17.2.6.2 void GPIO\_PinWrite ( ***GPIO\_Type*** \* *base*, ***uint32\_t*** *pin*, ***uint8\_t*** *output* )

Parameters

|               |                                                                                                                                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | GPIO base pointer.                                                                                                                                                                    |
| <i>pin</i>    | GPIO port pin number.                                                                                                                                                                 |
| <i>output</i> | GPIOpin 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> |

### 17.2.6.3 static void GPIO\_WritePinOutput ( ***GPIO\_Type*** \* *base*, ***uint32\_t*** *pin*, ***uint8\_t*** *output* ) [inline], [static]

### 17.2.6.4 static void GPIO\_PortSet ( ***GPIO\_Type*** \* *base*, ***uint32\_t*** *mask* ) [inline], [static]

Parameters

---

## GPIO Driver

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**17.2.6.5 static void GPIO\_SetPinsOutput ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* )  
[inline], [static]**

**17.2.6.6 static void GPIO\_PortClear ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline],  
[static]**

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**17.2.6.7 static void GPIO\_ClearPinsOutput ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* )  
[inline], [static]**

**17.2.6.8 static void GPIO\_PortToggle ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [inline],  
[static]**

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIO1, GPIO2, GPIO3, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**17.2.6.9 static **uint32\_t** GPIO\_PinRead ( **GPIO\_Type** \* *base*, **uint32\_t** *pin* ) [inline],  
[static]**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | GPIO base pointer.    |
| <i>pin</i>  | GPIO port pin number. |

Return values

|             |                   |
|-------------|-------------------|
| <i>GPIO</i> | port input value. |
|-------------|-------------------|

**17.2.6.10 static uint32\_t GPIO\_ReadPinInput ( *GPIO\_Type* \* *base*, *uint32\_t pin* )  
[inline], [static]**

**17.2.6.11 static uint8\_t GPIO\_PinReadPadStatus ( *GPIO\_Type* \* *base*, *uint32\_t pin* )  
[inline], [static]**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | GPIO base pointer.    |
| <i>pin</i>  | GPIO port pin number. |

Return values

|             |                       |
|-------------|-----------------------|
| <i>GPIO</i> | pin pad status value. |
|-------------|-----------------------|

**17.2.6.12 static uint8\_t GPIO\_ReadPadStatus ( *GPIO\_Type* \* *base*, *uint32\_t pin* )  
[inline], [static]**

**17.2.6.13 void GPIO\_PinSetInterruptConfig ( *GPIO\_Type* \* *base*, *uint32\_t pin*,  
*gpio\_interrupt\_mode\_t pinInterruptMode* )**

Parameters

|                               |                                                                                                   |
|-------------------------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>                   | GPIO base pointer.                                                                                |
| <i>pin</i>                    | GPIO port pin number.                                                                             |
| <i>pininterrupt-<br/>Mode</i> | pointer to a <i>gpio_interrupt_mode_t</i> structure that contains the interrupt mode information. |

**17.2.6.14 static void GPIO\_SetPinInterruptConfig ( *GPIO\_Type* \* *base*, *uint32\_t pin*,  
*gpio\_interrupt\_mode\_t pinInterruptMode* ) [inline], [static]**

**17.2.6.15 static void GPIO\_PortEnableInterrupts ( *GPIO\_Type* \* *base*, *uint32\_t mask* )  
[inline], [static]**

## GPIO Driver

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**17.2.6.16 static void GPIO\_EnableInterrupts ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**17.2.6.17 static void GPIO\_PortDisableInterrupts ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**17.2.6.18 static void GPIO\_DisableInterrupts ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

**17.2.6.19 static uint32\_t GPIO\_PortGetInterruptFlags ( GPIO\_Type \* *base* ) [inline],  
[static]**

Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | GPIO base pointer. |
|-------------|--------------------|

Return values

|                |                            |
|----------------|----------------------------|
| <i>current</i> | pin interrupt status flag. |
|----------------|----------------------------|

**17.2.6.20 static uint32\_t GPIO\_GetPinsInterruptFlags ( GPIO\_Type \* *base* ) [inline],  
[static]**

Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | GPIO base pointer. |
|-------------|--------------------|

Return values

|                |                            |
|----------------|----------------------------|
| <i>current</i> | pin interrupt status flag. |
|----------------|----------------------------|

### 17.2.6.21 static void GPIO\_PortClearInterruptFlags ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [**inline**], [**static**]

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

### 17.2.6.22 static void GPIO\_ClearPinsInterruptFlags ( **GPIO\_Type** \* *base*, **uint32\_t** *mask* ) [**inline**], [**static**]

Status flags are cleared by writing a 1 to the corresponding bit position.

Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |



# Chapter 18

## INTMUX: Interrupt Multiplexer Driver

### 18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Interrupt Multiplexer (INTMUX) module of MCUXpresso SDK devices.

### 18.2 Typical use case

#### 18.2.1 Channel Configure

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

### Enumerations

- enum `intmux_channel_logic_mode_t` {  
  `kINTMUX_ChannelLogicOR` = 0x0U,  
  `kINTMUX_ChannelLogicAND` }  
*INTMUX channel logic mode.*

### Driver version

- #define `FSL_INTMUX_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 1))  
*< Version 2.0.1.*

### Initialization and deinitialization

- void `INTMUX_Init` (INTMUX\_Type \*base)  
*Initializes the INTMUX module.*
- void `INTMUX_Deinit` (INTMUX\_Type \*base)  
*Deinitializes an INTMUX instance for operation.*
- static void `INTMUX_ResetChannel` (INTMUX\_Type \*base, uint32\_t channel)  
*Resets an INTMUX channel.*
- static void `INTMUX_SetChannelMode` (INTMUX\_Type \*base, uint32\_t channel, `intmux_channel_logic_mode_t` logic)  
*Sets the logic mode for an INTMUX channel.*

### Sources

- static void `INTMUX_EnableInterrupt` (INTMUX\_Type \*base, uint32\_t channel, IRQn\_Type irq)  
*Enables an interrupt source on an INTMUX channel.*
- static void `INTMUX_DisableInterrupt` (INTMUX\_Type \*base, uint32\_t channel, IRQn\_Type irq)  
*Disables an interrupt source on an INTMUX channel.*

## Function Documentation

### Status

- static uint32\_t [INTMUX\\_GetChannelPendingSources](#) (INTMUX\_Type \*base, uint32\_t channel)  
*Gets INTMUX pending interrupt sources for a specific channel.*

### 18.3 Macro Definition Documentation

#### 18.3.1 #define FSL\_INTMUX\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 18.4 Enumeration Type Documentation

#### 18.4.1 enum intmux\_channel\_logic\_mode\_t

Enumerator

*kINTMUX\_ChannelLogicOR* Logic OR all enabled interrupt inputs.

*kINTMUX\_ChannelLogicAND* Logic AND all enabled interrupt inputs.

### 18.5 Function Documentation

#### 18.5.1 void INTMUX\_Init ( INTMUX\_Type \* *base* )

This function enables the clock gate for the specified INTMUX. It then resets all channels, so that no interrupt sources are routed and the logic mode is set to default of [kINTMUX\\_ChannelLogicOR](#). Finally, the NVIC vectors for all the INTMUX output channels are enabled.

Parameters

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

#### 18.5.2 void INTMUX\_Deinit ( INTMUX\_Type \* *base* )

The clock gate for the specified INTMUX is disabled and the NVIC vectors for all channels are disabled.

Parameters

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

#### 18.5.3 static void INTMUX\_ResetChannel ( INTMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Sets all register values in the specified channel to their reset value. This function disables all interrupt sources for the channel.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | INTMUX peripheral base address. |
| <i>channel</i> | The INTMUX channel number.      |

#### 18.5.4 static void INTMUX\_SetChannelMode ( INTMUX\_Type \* *base*, uint32\_t *channel*, intmux\_channel\_logic\_mode\_t *logic* ) [inline], [static]

INTMUX channels can be configured to use one of the two logic modes that control how pending interrupt sources on the channel trigger the output interrupt.

- [kINTMUX\\_ChannelLogicOR](#) means any source pending triggers the output interrupt.
- [kINTMUX\\_ChannelLogicAND](#) means all selected sources on the channel must be pending before the channel output interrupt triggers.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | INTMUX peripheral base address. |
| <i>channel</i> | The INTMUX channel number.      |
| <i>logic</i>   | The INTMUX channel logic mode.  |

#### 18.5.5 static void INTMUX\_EnableInterrupt ( INTMUX\_Type \* *base*, uint32\_t *channel*, IRQn\_Type *irq* ) [inline], [static]

Parameters

|                |                                                                                              |
|----------------|----------------------------------------------------------------------------------------------|
| <i>base</i>    | INTMUX peripheral base address.                                                              |
| <i>channel</i> | Index of the INTMUX channel on which the specified interrupt is enabled.                     |
| <i>irq</i>     | Interrupt to route to the specified INTMUX channel. The interrupt must be an INT-MUX source. |

#### 18.5.6 static void INTMUX\_DisableInterrupt ( INTMUX\_Type \* *base*, uint32\_t *channel*, IRQn\_Type *irq* ) [inline], [static]

## Function Documentation

Parameters

|                |                                                                           |
|----------------|---------------------------------------------------------------------------|
| <i>base</i>    | INTMUX peripheral base address.                                           |
| <i>channel</i> | Index of the INTMUX channel on which the specified interrupt is disabled. |
| <i>irq</i>     | Interrupt number. The interrupt must be an INTMUX source.                 |

**18.5.7 static uint32\_t INTMUX\_GetChannelPendingSources ( INTMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | INTMUX peripheral base address. |
| <i>channel</i> | The INTMUX channel number.      |

Returns

The mask of pending interrupt bits. Bit[n] set means INTMUX source n is pending.

# Chapter 19

## IRQSTEER: Interrupt Request Steering Driver

### 19.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Interrupt Request Steering (IRQSTEER) module of MCUXpresso SDK devices. The IrqSteer module redirects/steers the incoming interrupts to output interrupts of a selected/designated channel as specified by a set of configuration registers.

### Macros

- #define `IRQSTEER_INT_SRC_REG_WIDTH` 32U  
*IRQSTEER interrupt source register width.*
- #define `IRQSTEER_INT_SRC_REG_INDEX`(irq)  
*IRQSTEER interrupt source mapping register index.*
- #define `IRQSTEER_INT_SRC_BIT_OFFSET`(irq) ((irq - FSL\_FEATURE\_IRQ\_ST-ART\_INDEX) % `IRQSTEER_INT_SRC_REG_WIDTH`)  
*IRQSTEER interrupt source mapping bit offset.*
- #define `IRQSTEER_INT_SRC_NUM`(regIndex, bitOffset) (((FSL\_FEATURE\_CHn-MASK\_COUNT - 1U - (regIndex)) \* `IRQSTEER_INT_SRC_REG_WIDTH`) + (bitOffset))  
*IRQSTEER interrupt source number.*

### Enumerations

- enum `irqsteer_int_group_t` {  
    kIRQSTEER\_InterruptGroup0,  
    kIRQSTEER\_InterruptGroup1,  
    kIRQSTEER\_InterruptGroup2,  
    kIRQSTEER\_InterruptGroup3,  
    kIRQSTEER\_InterruptGroup4,  
    kIRQSTEER\_InterruptGroup5,  
    kIRQSTEER\_InterruptGroup6,  
    kIRQSTEER\_InterruptGroup7,  
    kIRQSTEER\_InterruptGroup8,  
    kIRQSTEER\_InterruptGroup9,  
    kIRQSTEER\_InterruptGroup10,  
    kIRQSTEER\_InterruptGroup11,  
    kIRQSTEER\_InterruptGroup12,  
    kIRQSTEER\_InterruptGroup13,  
    kIRQSTEER\_InterruptGroup14,  
    kIRQSTEER\_InterruptGroup15 }  
*IRQSTEER interrupt groups.*

- enum `irqsteer_int_master_t` {

## Overview

```
kIRQSTEER_InterruptMaster0,
kIRQSTEER_InterruptMaster1,
kIRQSTEER_InterruptMaster2,
kIRQSTEER_InterruptMaster3,
kIRQSTEER_InterruptMaster4,
kIRQSTEER_InterruptMaster5,
kIRQSTEER_InterruptMaster6,
kIRQSTEER_InterruptMaster7 }
```

*IRQSTEER master interrupts mapping.*

## Driver version

- #define **FSL\_IRQSTEER\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
*< Version 2.0.1.*

## Initialization and deinitialization

- void **IRQSTEER\_Init** (IRQSTEER\_Type \*base)  
*Initializes the IRQSTEER module.*
- void **IRQSTEER\_Deinit** (IRQSTEER\_Type \*base)  
*Deinitializes an IRQSTEER instance for operation.*

## Sources

- static void **IRQSTEER\_EnableInterrupt** (IRQSTEER\_Type \*base, IRQn\_Type irq)  
*Enables an interrupt source.*
- static void **IRQSTEER\_DisableInterrupt** (IRQSTEER\_Type \*base, IRQn\_Type irq)  
*Disables an interrupt source.*
- static void **IRQSTEER\_SetInterrupt** (IRQSTEER\_Type \*base, IRQn\_Type irq, bool set)  
*Sets/Forces an interrupt.*
- static void **IRQSTEER\_EnableMasterInterrupt** (IRQSTEER\_Type \*base, irqsteer\_int\_master\_t intMasterIndex)  
*Enables a master interrupt.*
- static void **IRQSTEER\_DisableMasterInterrupt** (IRQSTEER\_Type \*base, irqsteer\_int\_master\_t intMasterIndex)  
*Disables a master interrupt.*

## Status

- static bool **IRQSTEER\_IsInterruptSet** (IRQSTEER\_Type \*base, IRQn\_Type irq)  
*Checks the status of one specific IRQSTEER interrupt.*
- static bool **IRQSTEER\_IsMasterInterruptSet** (IRQSTEER\_Type \*base)  
*Checks the status of IRQSTEER master interrupt.*
- static uint32\_t **IRQSTEER\_GetGroupInterruptStatus** (IRQSTEER\_Type \*base, irqsteer\_int\_group\_t intGroupIndex)  
*Gets the status of IRQSTEER group interrupt.*
- IRQn\_Type **IRQSTEER\_GetMasterNextInterrupt** (IRQSTEER\_Type \*base, irqsteer\_int\_master\_t intMasterIndex)  
*Gets the next interrupt source (currently set) of one specific master.*

## 19.2 Macro Definition Documentation

**19.2.1 #define FSL\_IRQSTEER\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))**

**19.2.2 #define IRQSTEER\_INT\_SRC\_REG\_WIDTH 32U**

**19.2.3 #define IRQSTEER\_INT\_SRC\_REG\_INDEX( *irq* )**

**Value:**

```
((FSL_FEATURE_IRQSTEER_CHn_MASK_COUNT - 1U) - \
 ((irq - FSL_FEATURE_IRQSTEER IRQ_START_INDEX) / IRQSTEER_INT_SRC_REG_WIDTH))
```

**19.2.4 #define IRQSTEER\_INT\_SRC\_BIT\_OFFSET( *irq* ) ((*irq* - FSL\_FEATURE\_IRQSTEER IRQ\_START\_INDEX) % IRQSTEER\_INT\_SRC\_REG\_WIDTH)**

**19.2.5 #define IRQSTEER\_INT\_SRC\_NUM( *regIndex*, *bitOffset* ) (((FSL\_FEATURE\_IRQSTEER\_CHn\_MASK\_COUNT - 1U - (*regIndex*)) \* IRQSTEER\_INT\_SRC\_REG\_WIDTH) + (*bitOffset*))**

## 19.3 Enumeration Type Documentation

**19.3.1 enum irqsteer\_int\_group\_t**

Enumerator

|                                   |                                                 |
|-----------------------------------|-------------------------------------------------|
| <i>kIRQSTEER InterruptGroup0</i>  | Interrupt Group 0: interrupt source 31 - 0.     |
| <i>kIRQSTEER InterruptGroup1</i>  | Interrupt Group 1: interrupt source 63 - 32.    |
| <i>kIRQSTEER InterruptGroup2</i>  | Interrupt Group 2: interrupt source 95 - 64.    |
| <i>kIRQSTEER InterruptGroup3</i>  | Interrupt Group 3: interrupt source 127 - 96.   |
| <i>kIRQSTEER InterruptGroup4</i>  | Interrupt Group 4: interrupt source 159 - 128.  |
| <i>kIRQSTEER InterruptGroup5</i>  | Interrupt Group 5: interrupt source 191 - 160.  |
| <i>kIRQSTEER InterruptGroup6</i>  | Interrupt Group 6: interrupt source 223 - 192.  |
| <i>kIRQSTEER InterruptGroup7</i>  | Interrupt Group 7: interrupt source 255 - 224.  |
| <i>kIRQSTEER InterruptGroup8</i>  | Interrupt Group 8: interrupt source 287 - 256.  |
| <i>kIRQSTEER InterruptGroup9</i>  | Interrupt Group 9: interrupt source 319 - 288.  |
| <i>kIRQSTEER InterruptGroup10</i> | Interrupt Group 10: interrupt source 351 - 320. |
| <i>kIRQSTEER InterruptGroup11</i> | Interrupt Group 11: interrupt source 383 - 352. |
| <i>kIRQSTEER InterruptGroup12</i> | Interrupt Group 12: interrupt source 415 - 384. |
| <i>kIRQSTEER InterruptGroup13</i> | Interrupt Group 13: interrupt source 447 - 416. |
| <i>kIRQSTEER InterruptGroup14</i> | Interrupt Group 14: interrupt source 479 - 448. |
| <i>kIRQSTEER InterruptGroup15</i> | Interrupt Group 15: interrupt source 511 - 480. |

## Function Documentation

### 19.3.2 enum irqsteer\_int\_master\_t

Enumerator

|                                   |                                                 |
|-----------------------------------|-------------------------------------------------|
| <i>kIRQSTEER_InterruptMaster0</i> | Interrupt Master 0: interrupt source 63 - 0.    |
| <i>kIRQSTEER_InterruptMaster1</i> | Interrupt Master 1: interrupt source 127 - 64.  |
| <i>kIRQSTEER_InterruptMaster2</i> | Interrupt Master 2: interrupt source 191 - 128. |
| <i>kIRQSTEER_InterruptMaster3</i> | Interrupt Master 3: interrupt source 255 - 192. |
| <i>kIRQSTEER_InterruptMaster4</i> | Interrupt Master 4: interrupt source 319 - 256. |
| <i>kIRQSTEER_InterruptMaster5</i> | Interrupt Master 5: interrupt source 383 - 320. |
| <i>kIRQSTEER_InterruptMaster6</i> | Interrupt Master 6: interrupt source 447 - 384. |
| <i>kIRQSTEER_InterruptMaster7</i> | Interrupt Master 7: interrupt source 511 - 448. |

## 19.4 Function Documentation

### 19.4.1 void IRQSTEER\_Init ( IRQSTEER\_Type \* *base* )

This function enables the clock gate for the specified IRQSTEER.

Parameters

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

### 19.4.2 void IRQSTEER\_Deinit ( IRQSTEER\_Type \* *base* )

The clock gate for the specified IRQSTEER is disabled.

Parameters

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

### 19.4.3 static void IRQSTEER\_EnableInterrupt ( IRQSTEER\_Type \* *base*, IRQn\_Type *irq* ) [inline], [static]

Parameters

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

|            |                                                                   |
|------------|-------------------------------------------------------------------|
| <i>irq</i> | Interrupt to be routed. The interrupt must be an IRQSTEER source. |
|------------|-------------------------------------------------------------------|

#### 19.4.4 static void IRQSTEER\_DisableInterrupt ( IRQSTEER\_Type \* *base*, IRQn\_Type *irq* ) [inline], [static]

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | IRQSTEER peripheral base address.                                  |
| <i>intSrc</i> | Interrupt source number. The interrupt must be an IRQSTEER source. |

#### 19.4.5 static void IRQSTEER\_SetInterrupt ( IRQSTEER\_Type \* *base*, IRQn\_Type *irq*, bool *set* ) [inline], [static]

Parameters

|               |                                                                                                          |
|---------------|----------------------------------------------------------------------------------------------------------|
| <i>base</i>   | IRQSTEER peripheral base address.                                                                        |
| <i>intSrc</i> | Interrupt to be set/forced. The interrupt must be an IRQSTEER source.                                    |
| <i>set</i>    | Switcher of the interrupt set/force function. "true" means to set. "false" means not (normal operation). |

Note

This function is not affected by the function [IRQSTEER\\_DisableInterrupt](#) and [IRQSTEER\\_EnableInterrupt](#).

#### 19.4.6 static void IRQSTEER\_EnableMasterInterrupt ( IRQSTEER\_Type \* *base*, irqsteer\_int\_master\_t *intMasterIndex* ) [inline], [static]

By default, all the master interrupts are enabled.

Parameters

---

## Function Documentation

|                       |                                                                          |
|-----------------------|--------------------------------------------------------------------------|
| <i>base</i>           | IRQSTEER peripheral base address.                                        |
| <i>intMasterIndex</i> | Master index of interrupt sources to be routed. "irqsteer_int_master_t". |

For example, to enable the interrupt sources of master 1:

```
* IRQSTEER_EnableMasterInterrupt(IRQSTEER_M4_0,
* kIRQSTEER_InterruptMaster1);
*
```

### 19.4.7 static void IRQSTEER\_DisableMasterInterrupt ( IRQSTEER\_Type \* *base*, irqsteer\_int\_master\_t *intMasterIndex* ) [inline], [static]

Parameters

|                       |                                                                            |
|-----------------------|----------------------------------------------------------------------------|
| <i>base</i>           | IRQSTEER peripheral base address.                                          |
| <i>intMasterIndex</i> | Master index of interrupt sources to be disabled. "irqsteer_int_master_t". |

For example, to disable the interrupt sources of master 1:

```
* IRQSTEER_DisableMasterInterrupt(IRQSTEER_M4_0,
* kIRQSTEER_InterruptMaster1);
*
```

### 19.4.8 static bool IRQSTEER\_IsInterruptSet ( IRQSTEER\_Type \* *base*, IRQn\_Type *irq* ) [inline], [static]

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | IRQSTEER peripheral base address.                                                |
| <i>intSrc</i> | Interrupt source status to be checked. The interrupt must be an IRQSTEER source. |

Returns

The interrupt status. "true" means interrupt set. "false" means not.

For example, to check whether interrupt from output 0 of Display 1 is set:

```
* if (IRQSTEER_IsInterruptSet(IRQSTEER_DISPLAY1_INT_OUT0)
* {
* ...
* }
```

### 19.4.9 static bool IRQSTEER\_IsMasterInterruptSet ( IRQSTEER\_Type \* *base* ) [inline], [static]

The master interrupt status represents at least one interrupt is asserted or not among ALL

Parameters

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

Returns

The master interrupt status. "true" means at least one interrupt set. "false" means not.

Note

The master interrupt status is not affected by the function [IRQSTEER\\_DisableMasterInterrupt](#).

### 19.4.10 static uint32\_t IRQSTEER\_GetGroupInterruptStatus ( IRQSTEER\_Type \* *base*, irqsteer\_int\_group\_t *intGroupIndex* ) [inline], [static]

The group interrupt status represents all the interrupt status within the group specified. This API aims for facilitating the status return of one set of interrupts.

Parameters

|                      |                                             |
|----------------------|---------------------------------------------|
| <i>base</i>          | IRQSTEER peripheral base address.           |
| <i>intGroupIndex</i> | Index of the interrupt group status to get. |

Returns

The mask of the group interrupt status. Bit[n] set means the source with bit offset n in group int-GroupIndex of IRQSTEER is asserted.

### 19.4.11 IRQn\_Type IRQSTEER\_GetMasterNextInterrupt ( IRQSTEER\_Type \* *base*, irqsteer\_int\_master\_t *intMasterIndex* )

## Function Documentation

### Parameters

|                       |                                                             |
|-----------------------|-------------------------------------------------------------|
| <i>base</i>           | IRQSTEER peripheral base address.                           |
| <i>intMasterIndex</i> | Master index of interrupt sources. "irqsteer_int_master_t". |

### Returns

The current set next interrupt source number of one specific master. Return IRQSTEER\_INT\_Invalid if no interrupt set.

# Chapter 20

## ISI: Image Sensing Interface

### 20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Image Sensing Interface(ISI) of I.MX devices.

The ISI module supports:

- Scaling
- Color Space Conversion
- Alpha insertion
- Image flipping
- Image cropping

The ISI driver provides separate functions for these features so these features can be enabled according to the use case.

To use an ISI channel, the function [ISI\\_Init](#) should be called first to enable the clock and set ISI to a defined status. After initialization, use the [ISI\\_SetConfig](#) to set the basic configuration. The ISI can work with the basic configurations, to enable the additional features and call the feature configuration functions such as [ISI\\_SetCropConfig](#) to set the configuration for specific feature. When the configuration is finished, call [ISI\\_Start](#) to start the ISI to work.

### 20.2 Typical use case

#### 20.2.1 Output buffer

Every ISI channel has two output buffers, every buffer has three panels, used by Y, U, and V accordingly. When a frame transfer is finished, the next frame is saved to the other buffer automatically.

In this example, the output format is RGBA8888, so only the outputBufferAddrY is used. To show that how to update the output buffer address, this example uses 5 memory blocks as the output buffer. The output frame is saved to these 5 memory blocks one by one. This means the first frame is saved to outputBufs[0] by ISI output buffer 0, the second frame is saved to outputBufs[1] by the ISI output buffer 1, the third frame is saved to outputBufs[2] by the ISI output buffer 0, the forth frame is saved to outputBufs[3] by the ISI output buffer 1, and so on.

```
#define ISI_BASE ISI0
#define MEM_BLK_CNT 5
#define IMG_HEIGHT 320
#define IMG_WIDTH 480

/* Memory blocks used as output buffer. */
uint32_t outputBufs[MEM_BLK_CNT][IMG_HEIGHT][IMG_WIDTH];
/* Index of outputBufs to save ISI output frame next. This value could be 0 to (MEM_BLK_CNT - 1). */
uint8_t outputBufIdx;
/* ISI output buffer that currently used. */
uint8_t isiBufIdx;
```

## Typical use case

```
void ISI_Configure(ISI_Type * base)
{
 isi_config_t isiConfig;

 ISI_GetDefaultConfig(&isiConfig);
 isiConfig->inputHeight = IMG_HEIGHT;
 isiConfig->inputWidth = IMG_WIDTH;
 isiConfig->outputFormat = kISI_OutputRGBAB8888;

 ISI_SetConfig(base, &isiConfig);

 /* Because color space conversion is enabled by default, so disable it. */
 ISI_EnableColorSpaceConversion(base, false);

 /* Set the address for output buffer. */
 ISI_SetOutputBufferAddr(base, 0U, (uint32_t)(outputBufs[0]), 0U, 0U);
 ISI_SetOutputBufferAddr(base, 1U, (uint32_t)(outputBufs[1]), 0U, 0U);

 /* outputBufs[2] will be used to save output frame next. */
 outputBufIdx = 2U;

 /* At the begining, ISI uses the output buffer 0. */
 isiBufIdx = 0U;
}

void ISI_IRQHandler(void)
{
 if (kISI_FrameReceivedInterrupt &
 ISI_GetInterruptStatus(ISI_BASE))
 {
 ISI_ClearInterruptStatus(ISI_BASE,
 kISI_FrameReceivedInterrupt);

 /* Frame output completed, set the output buffer address. */
 ISI_SetOutputBufferAddr(base, isiBufIdx, (uint32_t)(outputBufs[outputBufIdx])
), 0U, 0U);

 /* There are 2 ISI output buffers, so the output buffer index is 0 -> 1 -> 0 -> 1 -> ... */
 isiBufIdx ^= 1;

 /* Update the buffer memory block index. */
 outputBufIdx++;
 if (outputBufIdx >= MEM_BLK_CNT)
 {
 outputBufIdx = 0U;
 }
 }
}

void main(void)
{
 ISI_Init(ISI_BASE);

 ISI_Configure(ISI_BASE);

 /* Enable the ISI interrupt in SOC level, NVIC and IRQSTEER. */
 /* Enable the frame complete interrupt. */
 ISI_EnableInterrupts(ISI_BASE,
 kISI_FrameReceivedInterrupt);

 /* Start working. */
 ISI_Start(ISI_BASE);

 while (1)
 {
 }
}
```

## 20.2.2 Output panic and overflow

ISI employs two 256-bytes ping pong buffers between ISI and output memory. There are three level overflow interrupt for the ping pong buffer:

1. Ping pong buffer nearly full. The nearly full threshold is set by `isi_threshold_t`. The alert interrupt such as `kISI_OverflowAlertYInterrupt` occurs if the threshold is reached. When this kind of interrupt occurs, the application should request higher AXI write channel priority to make sure the ping pong buffer is read out in time.
2. Ping buffer overflow less than 256 Bytes. In this case, the overflow interrupt such as `kISI_OverflowYInterrupt` occurs. The number of overflow bytes could be gotten by `ISI_GetOverflowBytes`. ISI can insert a blank pixel for each lost pixel.
3. Ping buffer overflow more than 256 Bytes. This is monitored by excess overflow interrupt such as `kISI_ExcessOverflowYInterrupt`. In this case, application should reset the ISI.

```
void ISI_IRQHandler(void)
{
 uint32_t interrupts = ISI_GetInterruptStatus(ISI_BASE);
 ISI_ClearInterruptStatus(ISI_BASE, interrupts);

 if (kISI_ExcessOverflowYInterrupt & interrupts)
 {
 // Reset the ISI;
 }

 if (kISI_OverflowYInterrupt & interrupts)
 {
 uint8_t overflowBytes = ISI_GetOverflowBytes(ISI_BASE);
 }

 if (kISI_OverflowAlertYInterrupt & interrupts)
 {
 // Request higher AXI write channel priority;
 }
}
```

## Data Structures

- struct `isi_config_t`  
*ISI basic configuration.* [More...](#)
- struct `isi_csc_config_t`  
*ISI color space conversion configurations.* [More...](#)
- struct `isi_crop_config_t`  
*ISI cropping configurations.* [More...](#)
- struct `isi_region_alpha_config_t`  
*ISI regional region alpha configurations.* [More...](#)
- struct `isi_input_mem_config_t`  
*ISI input memory configurations.* [More...](#)

## Typical use case

### Enumerations

- enum `_isi_interrupt` {  
    `kISI_MemReadCompletedInterrupt` = `ISI_CHNL_IER_MEM_RD_DONE_EN_MASK`,  
    `kISI_LineReceivedInterrupt` = `ISI_CHNL_IER_LINE_RCVD_EN_MASK`,  
    `kISI_FrameReceivedInterrupt` = `ISI_CHNL_IER_FRM_RCVD_EN_MASK`,  
    `kISI_AxiWriteErrorVInterrupt`,  
    `kISI_AxiWriteErrorUIInterrupt`,  
    `kISI_AxiWriteErrorYInterrupt`,  
    `kISI_AxiReadErrorHandler` = `ISI_CHNL_IER_AXI_RD_ERR_EN_MASK`,  
    `kISI_OverflowAlertVInterrupt`,  
    `kISI_ExcessOverflowVInterrupt`,  
    `kISI_OverflowVInterrupt` = `ISI_CHNL_IER_OFLW_V_BUF_EN_MASK`,  
    `kISI_OverflowAlertUIInterrupt`,  
    `kISI_ExcessOverflowUIInterrupt`,  
    `kISI_OverflowUIInterrupt` = `ISI_CHNL_IER_OFLW_U_BUF_EN_MASK`,  
    `kISI_OverflowAlertYInterrupt`,  
    `kISI_ExcessOverflowYInterrupt`,  
    `kISI_OverflowYInterrupt` = `ISI_CHNL_IER_OFLW_Y_BUF_EN_MASK` }  
    *ISI interrupts.*  
}
- enum `isi_output_format_t` {

kISI\_OutputRGBA8888 = 0U,  
kISI\_OutputABGR8888 = 1U,  
kISI\_OutputARGB8888 = 2U,  
kISI\_OutputRGBX8888 = 3U,  
kISI\_OutputXBGR8888 = 4U,  
kISI\_OutputXRGB8888 = 5U,  
kISI\_OutputRGB888 = 6U,  
kISI\_OutputBGR888 = 7U,  
kISI\_OutputA2BGR10 = 8U,  
kISI\_OutputA2RGB10 = 9U,  
kISI\_OutputRGB565 = 10U,  
kISI\_OutputRaw8 = 11U,  
kISI\_OutputRaw10 = 12U,  
kISI\_OutputRaw10P = 13U,  
kISI\_OutputRaw12P = 14U,  
kISI\_OutputRaw16P = 15U,  
kISI\_OutputYUV444\_1P8P = 16U,  
kISI\_OutputYUV444\_2P8P = 17U,  
kISI\_OutputYUV444\_3P8P = 18U,  
kISI\_OutputYUV444\_1P8 = 19U,  
kISI\_OutputYUV444\_1P10 = 20U,  
kISI\_OutputYUV444\_2P10 = 21U,  
kISI\_OutputYUV444\_3P10 = 22U,  
kISI\_OutputYUV444\_1P10P = 24U,  
kISI\_OutputYUV444\_2P10P = 25U,  
kISI\_OutputYUV444\_3P10P = 26U,  
kISI\_OutputYUV444\_1P12 = 28U,  
kISI\_OutputYUV444\_2P12 = 29U,  
kISI\_OutputYUV444\_3P12 = 30U,  
kISI\_OutputYUV422\_1P8P = 32U,  
kISI\_OutputYUV422\_2P8P = 33U,  
kISI\_OutputYUV422\_3P8P = 34U,  
kISI\_OutputYUV422\_1P10 = 36U,  
kISI\_OutputYUV422\_2P10 = 37U,  
kISI\_OutputYUV422\_3P10 = 38U,  
kISI\_OutputYUV422\_1P10P = 40U,  
kISI\_OutputYUV422\_2P10P = 41U,  
kISI\_OutputYUV422\_3P10P = 42U,  
kISI\_OutputYUV422\_1P12 = 44U,  
kISI\_OutputYUV422\_2P12 = 45U,  
kISI\_OutputYUV422\_3P12 = 46U,  
kISI\_OutputYUV420\_2P8P = 49U,  
kISI\_OutputYUV420\_3P8P = 50U,  
kISI\_OutputYUV420\_2P10 = 53U,  
kISI\_OutputYUV420\_3P10 = 54U,  
kISI\_OutputYUV420\_2P10P = 57U,  
kISI\_OutputYUV420\_3P10P = 58U  
kISI\_OutputYUV420\_2P12 = 61U,

## Typical use case

```
kISI_OutputYUV420_3P12 = 62U }

ISI output image format.
• enum isi_chain_mode_t {
 kISI_ChainDisable = 0U,
 kISI_ChainTwo = 1U }

ISI line buffer chain mode.
• enum isi_deint_mode_t {
 kISI_DeintDisable = 0U,
 kISI_DeintWeaveOddOnTop = 2U,
 kISI_DeintWeaveEvenOnTop = 3U,
 kISI_DeintBlendingOddFirst = 4U,
 kISI_DeintBlendingEvenFirst = 5U,
 kISI_DeintDoublingOdd = 6U,
 kISI_DeintDoublingEven = 7U }

ISI de-interlacing mode.
• enum isi_threshold_t {
 kISI_ThresholdDisable = 0U,
 kISI_Threshold25Percent = 1U,
 kISI_Threshold50Percent = 2U,
 kISI_Threshold75Percent = 3U }

ISI overflow panic alert threshold.
• enum isi_csc_mode_t {
 kISI_CscYUV2RGB,
 kISI_CscYCbCr2RGB,
 kISI_CscRGB2YUV,
 kISI_CscRGB2YCbCr }

ISI color space conversion mode.
• enum isi_flip_mode_t {
 kISI_FlipDisable = 0U,
 kISI_FlipHorizontal = ISI_CHNL_IMG_CTRL_HFLIP_EN_MASK,
 kISI_FlipVertical = ISI_CHNL_IMG_CTRL_VFLIP_EN_MASK,
 kISI_FlipBoth = ISI_CHNL_IMG_CTRL_VFLIP_EN_MASK | ISI_CHNL_IMG_CTRL_HFLIP_
 _EN_MASK }

ISI flipping mode.
• enum isi_input_mem_format_t {
```

```

kISI_InputMemBGR888 = 0U,
kISI_InputMemRGB888 = 1U,
kISI_InputMemXRGB8888 = 2U,
kISI_InputMemRGBX8888 = 3U,
kISI_InputMemXBGR8888 = 4U,
kISI_InputMemRGB565 = 5U,
kISI_InputMemA2BGR10 = 6U,
kISI_InputMemA2RGB10 = 7U,
kISI_InputMemYUV444_1P8P = 8U,
kISI_InputMemYUV444_1P10 = 9U,
kISI_InputMemYUV444_1P10P = 10U,
kISI_InputMemYUV444_1P12 = 11U,
kISI_InputMemYUV444_1P8 = 12U,
kISI_InputMemYUV422_1P8P = 13U,
kISI_InputMemYUV422_1P10 = 14U,
kISI_InputMemYUV422_1P12 = 15U }

```

*ISI image format of the input memory.*

## Functions

- static uint8\_t **ISI\_GetOverflowBytes** (ISI\_Type \*base)  
*Gets the number of valid pixel bytes lost due to overflow.*
- void **ISI\_SetConfig** (ISI\_Type \*base, const **isi\_config\_t** \*config)  
*Set the ISI channel basic configurations.*
- void **ISI\_GetDefaultConfig** (**isi\_config\_t** \*config)  
*Get the ISI channel default basic configurations.*

## Driver version

- #define **FSL\_ISI\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
*ISI driver version.*

## ISI initialization and de-initialization

- void **ISI\_Init** (ISI\_Type \*base)  
*Initializes the ISI peripheral.*
- void **ISI\_Deinit** (ISI\_Type \*base)  
*Deinitializes the ISI peripheral.*
- void **ISI\_Reset** (ISI\_Type \*base)  
*Reset the ISI peripheral.*

## ISI interrupts

- static uint32\_t **ISI\_EnableInterrupts** (ISI\_Type \*base, uint32\_t mask)  
*Enables ISI interrupts.*
- static uint32\_t **ISI\_DisableInterrupts** (ISI\_Type \*base, uint32\_t mask)  
*Disables ISI interrupts.*
- static uint32\_t **ISI\_GetInterruptStatus** (ISI\_Type \*base)

## Typical use case

- static void **ISI\_ClearInterruptStatus** (ISI\_Type \*base, uint32\_t mask)  
*Clear ISI interrupt pending flags.*

## ISI scaler

- void **ISI\_SetScalerConfig** (ISI\_Type \*base, uint16\_t inputWidth, uint16\_t inputHeight, uint16\_t outputWidth, uint16\_t outputHeight)  
*Set the ISI channel scaler configurations.*

## ISI color space conversion

- void **ISI\_SetColorSpaceConversionConfig** (ISI\_Type \*base, const **isi\_csc\_config\_t** \*config)  
*Set the ISI color space conversion configurations.*
- void **ISI\_ColorSpaceConversionGetDefaultConfig** (**isi\_csc\_config\_t** \*config)  
*Get the ISI color space conversion default configurations.*
- static void **ISI\_EnableColorSpaceConversion** (ISI\_Type \*base, bool enable)  
*Enable or disable the ISI color space conversion.*

## ISI cropping

- void **ISI\_SetCropConfig** (ISI\_Type \*base, const **isi\_crop\_config\_t** \*config)  
*Set the ISI cropping configurations.*
- void **ISI\_CropGetDefaultConfig** (**isi\_crop\_config\_t** \*config)  
*Get the ISI cropping default configurations.*
- static void **ISI\_EnableCrop** (ISI\_Type \*base, bool enable)  
*Enable or disable the ISI cropping.*

## ISI alpha

- static void **ISI\_SetGlobalAlpha** (ISI\_Type \*base, uint8\_t alpha)  
*Set the global alpha value.*
- static void **ISI\_EnableGlobalAlpha** (ISI\_Type \*base, bool enable)  
*Enable the global alpha insertion.*
- void **ISI\_SetRegionAlphaConfig** (ISI\_Type \*base, uint8\_t index, const **isi\_region\_alpha\_config\_t** \*config)  
*Set the alpha value for region of interest.*
- void **ISI\_RegionAlphaGetDefaultConfig** (**isi\_region\_alpha\_config\_t** \*config)  
*Get the regional alpha insertion default configurations.*
- static void **ISI\_EnableRegionAlpha** (ISI\_Type \*base, uint8\_t index, bool enable)  
*Enable or disable the alpha value insertion for region of interest.*

## ISI input memory.

- void **ISI\_SetInputMemConfig** (ISI\_Type \*base, const **isi\_input\_mem\_config\_t** \*config)  
*Set the input memory configuration.*
- void **ISI\_InputMemGetDefaultConfig** (**isi\_input\_mem\_config\_t** \*config)  
*Get the input memory default configurations.*
- static void **ISI\_SetInputMemAddr** (ISI\_Type \*base, uint32\_t addr)  
*Set the input memory address.*

- void **ISI\_TriggerInputMemRead** (ISI\_Type \*base)  
*Trigger the ISI pipeline to read the input memory.*

## ISI misc control.

- static void **ISI\_SetFlipMode** (ISI\_Type \*base, isi\_flip\_mode\_t mode)  
*Set the ISI channel flipping mode.*
- void **ISI\_SetOutputBufferAddr** (ISI\_Type \*base, uint8\_t index, uint32\_t addrY, uint32\_t addrU, uint32\_t addrV)  
*Set the ISI output buffer address.*
- static void **ISI\_Start** (ISI\_Type \*base)  
*Start the ISI channel.*
- static void **ISI\_Stop** (ISI\_Type \*base)  
*Stop the ISI channel.*

## 20.3 Data Structure Documentation

### 20.3.1 struct isi\_config\_t

#### Data Fields

- bool **isChannelBypassed**  
*Bypass the channel, if bypassed, the scaling and color space conversion could not work.*
- bool **isSourceMemory**  
*Whether the input source is memory or not.*
- bool **isYCbCr**  
*Whether the input source is YCbCr mode or not.*
- isi\_chain\_mode\_t **chainMode**  
*The line buffer chain mode.*
- isi\_deint\_mode\_t **deintMode**  
*The de-interlacing mode.*
- uint8\_t **blankPixel**  
*The pixel to insert into image when overflow occurs.*
- uint8\_t **sourcePort**  
*Input source port selection.*
- uint8\_t **mipiChannel**  
*MIPI virtual channel, ignored if input source is not MIPI CSI.*
- uint16\_t **inputHeight**  
*Input image height(lines).*
- uint16\_t **inputWidth**  
*Input image width(pixels).*
- isi\_output\_format\_t **outputFormat**  
*Output image format.*
- isi\_threshold\_t **thresholdY**  
*Panic alert threshold for RGB or Luma (Y) buffer.*
- isi\_threshold\_t **thresholdU**  
*Panic alert threshold for Chroma (U/Cb/UV/CbCr) buffer.*
- isi\_threshold\_t **thresholdV**  
*Panic alert threshold for Chroma (V/Cr) buffer.*

## Data Structure Documentation

### 20.3.1.0.0.55 Field Documentation

20.3.1.0.0.55.1 **bool isi\_config\_t::isChannelBypassed**

20.3.1.0.0.55.2 **bool isi\_config\_t::isSourceMemory**

20.3.1.0.0.55.3 **bool isi\_config\_t::isYCbCr**

20.3.1.0.0.55.4 **isi\_chain\_mode\_t isi\_config\_t::chainMode**

20.3.1.0.0.55.5 **isi\_deint\_mode\_t isi\_config\_t::deintMode**

20.3.1.0.0.55.6 **uint8\_t isi\_config\_t::blankPixel**

20.3.1.0.0.55.7 **uint8\_t isi\_config\_t::sourcePort**

20.3.1.0.0.55.8 **uint8\_t isi\_config\_t::mipiChannel**

20.3.1.0.0.55.9 **uint16\_t isi\_config\_t::inputHeight**

20.3.1.0.0.55.10 **uint16\_t isi\_config\_t::inputWidth**

20.3.1.0.0.55.11 **isi\_output\_format\_t isi\_config\_t::outputFormat**

20.3.1.0.0.55.12 **isi\_threshold\_t isi\_config\_t::thresholdY**

20.3.1.0.0.55.13 **isi\_threshold\_t isi\_config\_t::thresholdU**

20.3.1.0.0.55.14 **isi\_threshold\_t isi\_config\_t::thresholdV**

### 20.3.2 struct isi\_csc\_config\_t

(a) RGB to YUV (or YCbCr) conversion

- $Y = (A1 \times R) + (A2 \times G) + (A3 \times B) + D1$
- $U = (B1 \times R) + (B2 \times G) + (B3 \times B) + D2$
- $V = (C1 \times R) + (C2 \times G) + (C3 \times B) + D3$

(b) YUV (or YCbCr) to RGB conversion

- $R = (A1 \times (Y + D1)) + (A2 \times (U + D2)) + (A3 \times (V + D3))$
- $G = (B1 \times (Y + D1)) + (B2 \times (U + D2)) + (B3 \times (V + D3))$
- $B = (C1 \times (Y + D1)) + (C2 \times (U + D2)) + (C3 \times (V + D3))$

Overflow for the three channels are saturated at 0x255 and underflow is saturated at 0x00.

## Data Fields

- **isi\_csc\_mode\_t mode**  
*Conversion mode.*

- float [A1](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- float [A2](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- float [A3](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- float [B1](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- float [B2](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- float [B3](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- float [C1](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- float [C2](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- float [C3](#)  
*Must be in the range of [-3.99609375, 3.99609375].*
- int16\_t [D1](#)  
*Must be in the range of [-256, 255].*
- int16\_t [D2](#)  
*Must be in the range of [-256, 255].*
- int16\_t [D3](#)  
*Must be in the range of [-256, 255].*

## Data Structure Documentation

### 20.3.2.0.0.56 Field Documentation

20.3.2.0.0.56.1 `isi_csc_mode_t isi_csc_config_t::mode`

20.3.2.0.0.56.2 `float isi_csc_config_t::A1`

20.3.2.0.0.56.3 `float isi_csc_config_t::A2`

20.3.2.0.0.56.4 `float isi_csc_config_t::A3`

20.3.2.0.0.56.5 `float isi_csc_config_t::B1`

20.3.2.0.0.56.6 `float isi_csc_config_t::B2`

20.3.2.0.0.56.7 `float isi_csc_config_t::B3`

20.3.2.0.0.56.8 `float isi_csc_config_t::C1`

20.3.2.0.0.56.9 `float isi_csc_config_t::C2`

20.3.2.0.0.56.10 `float isi_csc_config_t::C3`

20.3.2.0.0.56.11 `int16_t isi_csc_config_t::D1`

20.3.2.0.0.56.12 `int16_t isi_csc_config_t::D2`

20.3.2.0.0.56.13 `int16_t isi_csc_config_t::D3`

### 20.3.3 `struct isi_crop_config_t`

#### Data Fields

- `uint16_t upperLeftX`  
*X of upper left corner.*
- `uint16_t upperLeftY`  
*Y of upper left corner.*
- `uint16_t lowerRightX`  
*X of lower right corner.*
- `uint16_t lowerRightY`  
*Y of lower right corner.*

### 20.3.3.0.0.57 Field Documentation

20.3.3.0.0.57.1 `uint16_t isi_crop_config_t::upperLeftX`

20.3.3.0.0.57.2 `uint16_t isi_crop_config_t::upperLeftY`

20.3.3.0.0.57.3 `uint16_t isi_crop_config_t::lowerRightX`

20.3.3.0.0.57.4 `uint16_t isi_crop_config_t::lowerRightY`

### 20.3.4 `struct isi_region_alpha_config_t`

#### Data Fields

- `uint16_t upperLeftX`  
*X of upper left corner.*
- `uint16_t upperLeftY`  
*Y of upper left corner.*
- `uint16_t lowerRightX`  
*X of lower right corner.*
- `uint16_t lowerRightY`  
*Y of lower right corner.*
- `uint8_t alpha`  
*Alpha value.*

### 20.3.4.0.0.58 Field Documentation

20.3.4.0.0.58.1 `uint16_t isi_region_alpha_config_t::upperLeftX`

20.3.4.0.0.58.2 `uint16_t isi_region_alpha_config_t::upperLeftY`

20.3.4.0.0.58.3 `uint16_t isi_region_alpha_config_t::lowerRightX`

20.3.4.0.0.58.4 `uint16_t isi_region_alpha_config_t::lowerRightY`

20.3.4.0.0.58.5 `uint8_t isi_region_alpha_config_t::alpha`

### 20.3.5 `struct isi_input_mem_config_t`

#### Data Fields

- `uint32_t adddr`  
*Address of the input memory.*
- `uint16_t linePitchBytes`  
*Line phtch in bytes.*
- `uint16_t framePitchBytes`  
*Frame phtch in bytes.*
- `isi_input_mem_format_t format`  
*Image format of the input memory.*

## Enumeration Type Documentation

### 20.3.5.0.0.59 Field Documentation

20.3.5.0.0.59.1 `uint32_t isi_input_mem_config_t::addr`

20.3.5.0.0.59.2 `uint16_t isi_input_mem_config_t::linePitchBytes`

20.3.5.0.0.59.3 `uint16_t isi_input_mem_config_t::framePitchBytes`

20.3.5.0.0.59.4 `isi_input_mem_format_t isi_input_mem_config_t::format`

## 20.4 Macro Definition Documentation

20.4.1 `#define FSL_ISI_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

Version 2.0.1.

## 20.5 Enumeration Type Documentation

### 20.5.1 enum \_isi\_interrupt

Enumerator

`kISI_MemReadCompletedInterrupt` Input memory read completed.

`kISI_LineReceivedInterrupt` Line received.

`kISI_FrameReceivedInterrupt` Frame received.

`kISI_AxiWriteErrorVInterrupt` AXI Bus write error when storing V data to memory.

`kISI_AxiWriteErrorUInterrupt` AXI Bus write error when storing U data to memory.

`kISI_AxiWriteErrorYInterrupt` AXI Bus write error when storing Y data to memory.

`kISI_AxiReadErrorInterrupt` AXI Bus error when reading the input memory.

`kISI_OverflowAlertVInterrupt` V output buffer overflow threshold accrossed.

`kISI_ExcessOverflowVInterrupt` V output buffer excess overflow interrupt.

`kISI_OverflowVInterrupt` V output buffer overflow interrupt.

`kISI_OverflowAlertUInterrupt` U output buffer overflow threshold accrossed.

`kISI_ExcessOverflowUInterrupt` U output buffer excess overflow interrupt.

`kISI_OverflowUIInterrupt` U output buffer overflow interrupt.

`kISI_OverflowAlertYInterrupt` V output buffer overflow threshold accrossed.

`kISI_ExcessOverflowYInterrupt` V output buffer excess overflow interrupt.

`kISI_OverflowYInterrupt` V output buffer overflow interrupt.

### 20.5.2 enum isi\_output\_format\_t

Enumerator

`kISI_OutputRGBA8888` RGBA8888.

`kISI_OutputABGR8888` ABGR8888.

`kISI_OutputARGB8888` ARGB8888.

***kISI\_OutputRGBX8888*** RGBX8888 unpacked and MSB aligned in 32-bit.

***kISI\_OutputXBGR8888*** XBGR8888 unpacked and LSB aligned in 32-bit.

***kISI\_OutputXRGB8888*** XRGB8888 unpacked and LSB aligned in 32-bit.

***kISI\_OutputRGB888*** RGB888 packed into 32-bit.

***kISI\_OutputBGR888*** BGR888 packed into 32-bit.

***kISI\_OutputA2BGR10*** BGR format with 2-bits alpha in MSB; 10-bits per color component.

***kISI\_OutputA2RGB10*** RGB format with 2-bits alpha in MSB; 10-bits per color component.

***kISI\_OutputRGB565*** RGB565 packed into 32-bit.

***kISI\_OutputRaw8*** 8-bit raw data packed into 32-bit.

***kISI\_OutputRaw10*** 10-bit raw data packed into 16-bit with 6 LSBs wasted.

***kISI\_OutputRaw10P*** 10-bit raw data packed into 32-bit.

***kISI\_OutputRaw12P*** 16-bit raw data packed into 16-bit with 4 LSBs wasted.

***kISI\_OutputRaw16P*** 16-bit raw data packed into 32-bit.

***kISI\_OutputYUV444\_1P8P*** 8-bits per color component; 1-plane, YUV interleaved packed bytes.

***kISI\_OutputYUV444\_2P8P*** 8-bits per color component; 2-plane, UV interleaved packed bytes.

***kISI\_OutputYUV444\_3P8P*** 8-bits per color component; 3-plane, non-interleaved packed bytes.

***kISI\_OutputYUV444\_1P8*** 8-bits per color component; 1-plane YUV interleaved unpacked bytes (8 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV444\_1P10*** 10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV444\_2P10*** 10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV444\_3P10*** 10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV444\_1P10P*** 10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV444\_2P10P*** 10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV444\_3P10P*** 10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV444\_1P12*** 12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV444\_2P12*** 12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV444\_3P12*** 12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV422\_1P8P*** 8-bits per color component; 1-plane, YUV interleaved packed bytes.

***kISI\_OutputYUV422\_2P8P*** 8-bits per color component; 2-plane, UV interleaved packed bytes.

***kISI\_OutputYUV422\_3P8P*** 8-bits per color component; 3-plane, non-interleaved packed bytes.

***kISI\_OutputYUV422\_1P10*** 10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV422\_2P10*** 10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV422\_3P10*** 10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

## Enumeration Type Documentation

***kISI\_OutputYUV422\_1P10P*** 10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV422\_2P10P*** 10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV422\_3P10P*** 10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV422\_1P12*** 12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV422\_2P12*** 12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV422\_3P12*** 12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV420\_2P8P*** 8-bits per color component; 2-plane, UV interleaved packed bytes.

***kISI\_OutputYUV420\_3P8P*** 8-bits per color component; 3-plane, non-interleaved packed bytes.

***kISI\_OutputYUV420\_2P10*** 10-bits per color component; 2-plane, UV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV420\_3P10*** 10-bits per color component; 3-plane, non-interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV420\_2P10P*** 10-bits per color component; 2-plane, UV interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV420\_3P10P*** 10-bits per color component; 3-plane, non-interleaved packed bytes (2 MSBs waste bits in 32-bit DWORD).

***kISI\_OutputYUV420\_2P12*** 12-bits per color component; 2-plane, UV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

***kISI\_OutputYUV420\_3P12*** 12-bits per color component; 3-plane, non-interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

### 20.5.3 enum isi\_chain\_mode\_t

Enumerator

***kISI\_ChainDisable*** No line buffers chained, for 2048 or less horizontal resolution.

***kISI\_ChainTwo*** Line buffers of channel n and n+1 chained, for 4096 horizontal resolution.

### 20.5.4 enum isi\_deint\_mode\_t

Enumerator

***kISI\_DeintDisable*** No de-interlacing.

***kISI\_DeintWeaveOddOnTop*** Weave de-interlacing (Odd, Even) method used.

***kISI\_DeintWeaveEvenOnTop*** Weave de-interlacing (Even, Odd) method used.

***kISI\_DeintBlendingOddFirst*** Blending or linear interpolation (Odd + Even).

***kISI\_DeintBlendingEvenFirst*** Blending or linear interpolation (Even + Odd).

***kISI\_DeintDoublingOdd*** Doubling odd frame and discard even frame.

***kISI\_DeintDoublingEven*** Doubling even frame and discard odd frame.

## 20.5.5 enum isi\_threshold\_t

Enumerator

***kISI\_ThresholdDisable*** No panic alert will be asserted.

***kISI\_Threshold25Percent*** Panic will assert when the buffers are 25% full.

***kISI\_Threshold50Percent*** Panic will assert when the buffers are 50% full.

***kISI\_Threshold75Percent*** Panic will assert when the buffers are 75% full.

## 20.5.6 enum isi\_csc\_mode\_t

Enumerator

***kISI\_CscYUV2RGB*** Convert YUV to RGB.

***kISI\_CscYCbCr2RGB*** Convert YCbCr to RGB.

***kISI\_CscRGB2YUV*** Convert RGB to YUV.

***kISI\_CscRGB2YCbCr*** Convert RGB to YCbCr.

## 20.5.7 enum isi\_flip\_mode\_t

Enumerator

***kISI\_FlipDisable*** Flip disabled.

***kISI\_FlipHorizontal*** Horizontal flip.

***kISI\_FlipVertical*** Vertical flip.

***kISI\_FlipBoth*** Flip both direction.

## 20.5.8 enum isi\_input\_mem\_format\_t

Enumerator

***kISI\_InputMemBGR888*** BGR format with 8-bits per color component, packed into 32-bit, 24 bits per pixel.

***kISI\_InputMemRGB888*** RGB format with 8-bits per color component, packed into 32-bit, 24 bits per pixel.

***kISI\_InputMemXRGB8888*** RGB format with 8-bits per color component, unpacked and LSB aligned in 32-bit, 32 bits per pixel.

## Function Documentation

***kISI\_InputMemRGBX8888*** RGB format with 8-bits per color component, unpacked and MSB aligned in 32-bit, 32 bits per pixel.

***kISI\_InputMemXBGR8888*** BGR format with 8-bits per color component, unpacked and LSB aligned in 32-bit, 32 bits per pixel.

***kISI\_InputMemRGB565*** RGB format with 5-bits of R, B; 6-bits of G (packed into 32-bit)

***kISI\_InputMemA2BGR10*** BGR format with 2-bits alpha in MSB; 10-bits per color component.

***kISI\_InputMemA2RGB10*** RGB format with 2-bits alpha in MSB; 10-bits per color component.

***kISI\_InputMemYUV444\_1P8P*** 8-bits per color component; 1-plane, YUV interleaved packed bytes.

***kISI\_InputMemYUV444\_1P10*** 10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_InputMemYUV444\_1P10P*** 10-bits per color component; 1-plane, YUV interleaved packed bytes (2 MSBs waste bits in 32-bit WORD).

***kISI\_InputMemYUV444\_1P12*** 12-bits per color component; 1-plane, YUV interleaved unpacked bytes (4 LSBs waste bits in 16-bit WORD).

***kISI\_InputMemYUV444\_1P8*** 8-bits per color component; 1-plane YUV interleaved unpacked bytes (8 MSBs waste bits in 32-bit DWORD).

***kISI\_InputMemYUV422\_1P8P*** 8-bits per color component; 1-plane YUV interleaved packed bytes.

***kISI\_InputMemYUV422\_1P10*** 10-bits per color component; 1-plane, YUV interleaved unpacked bytes (6 LSBs waste bits in 16-bit WORD).

***kISI\_InputMemYUV422\_1P12*** 12-bits per color component; 1-plane, YUV interleaved packed bytes (4 MSBs waste bits in 16-bit WORD).

## 20.6 Function Documentation

### 20.6.1 void ISI\_Init ( ISI\_Type \* *base* )

This function ungates the ISI clock, it should be called before any other ISI functions.

Parameters

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

### 20.6.2 void ISI\_Deinit ( ISI\_Type \* *base* )

This function gates the ISI clock.

Parameters

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

### 20.6.3 void ISI\_Reset ( ISI\_Type \* *base* )

This function resets the ISI channel processing pipeline similar to a hardware reset. The channel will need to be reconfigured after reset before it can be used.

Parameters

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

### 20.6.4 static uint32\_t ISI\_EnableInterrupts ( ISI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | ISI peripheral base address                                       |
| <i>mask</i> | Interrupt source, OR'ed value of <a href="#">_isi_interrupt</a> . |

Returns

OR'ed value of the enabled interrupts before calling this function.

### 20.6.5 static uint32\_t ISI\_DisableInterrupts ( ISI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | ISI peripheral base address                                       |
| <i>mask</i> | Interrupt source, OR'ed value of <a href="#">_isi_interrupt</a> . |

Returns

OR'ed value of the enabled interrupts before calling this function.

## Function Documentation

### 20.6.6 static uint32\_t ISI\_GetInterruptStatus ( ISI\_Type \* *base* ) [inline], [static]

All interrupt pending flags are returned, upper layer could compare with the OR'ed value of [\\_isi\\_interrupt](#). For example, to check whether memory read completed, use like this:

```
uint32_t mask = ISI_GetInterruptStatus(ISI);
if (mask & kISI_MemReadCompletedInterrupt)
{
 // memory read completed
}
```

Parameters

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

Returns

The OR'ed value of the pending interrupt flags. of [\\_isi\\_interrupt](#).

### 20.6.7 static void ISI\_ClearInterruptStatus ( ISI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function could clear one or more flags at one time, the flags to clear are passed in as an OR'ed value of [\\_isi\\_interrupt](#). For example, to clear both line received interrupt flag and frame received flag, use like this:

```
ISI_ClearInterruptStatus(ISI, kISI_LineReceivedInterrupt
| kISI_FrameReceivedInterrupt);
```

Parameters

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| <i>base</i> | ISI peripheral base address                                               |
| <i>mask</i> | The flags to clear, it is OR'ed value of <a href="#">_isi_interrupt</a> . |

### 20.6.8 static uint8\_t ISI\_GetOverflowBytes ( ISI\_Type \* *base* ) [inline], [static]

If multiple output buffers overflow, then this function only returns the status of the buffer with highest priority. The buffer priority is: Y output buffer > U output buffer > V output buffer.

## Parameters

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

## Returns

The number of valid pixel bytes lost due to overflow.

**20.6.9 void ISI\_SetConfig ( ISI\_Type \* *base*, const isi\_config\_t \* *config* )**

This function sets the basic configurations, generally the channel could be started to work after this function. To enable other features such as cropping, flipping, please call the functions accordingly.

## Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | ISI peripheral base address             |
| <i>config</i> | Pointer to the configuration structure. |

**20.6.10 void ISI\_GetDefaultConfig ( isi\_config\_t \* *config* )**

The default value is:

```
config->isChannelBypassed = false;
config->isSourceMemory = false;
config->isYCbCr = false;
config->chainMode = kISI_ChainDisable;
config->deintMode = kISI_DeintDisable;
config->blankPixel = 0xFFU;
config->sourcePort = 0U;
config->mipiChannel = 0U;
config->inputHeight = 1080U;
config->inputWidth = 1920U;
config->outputFormat = kISI_OutputRGBA8888;
config->outputLinePitchBytes = 0U;
config->thresholdY = kISI_ThresholdDisable;
config->thresholdU = kISI_ThresholdDisable;
config->thresholdV = kISI_ThresholdDisable;
```

## Parameters

## Function Documentation

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

### 20.6.11 void ISI\_SetScalerConfig ( ISI\_Type \* *base*, uint16\_t *inputWidth*, uint16\_t *inputHeight*, uint16\_t *outputWidth*, uint16\_t *outputHeight* )

This function sets the scaling configurations. If the ISI channel is bypassed, then the scaling feature could not be used.

ISI only supports down scaling but not up scaling.

Parameters

|                     |                             |
|---------------------|-----------------------------|
| <i>base</i>         | ISI peripheral base address |
| <i>inputWidth</i>   | Input image width.          |
| <i>inputHeight</i>  | Input image height.         |
| <i>outputWidth</i>  | Output image width.         |
| <i>outputHeight</i> | Output image height.        |

Note

Total bytes in one line after down scaling must be more than 256 bytes.

### 20.6.12 void ISI\_SetColorSpaceConversionConfig ( ISI\_Type \* *base*, const isi\_csc\_config\_t \* *config* )

This function sets the color space conversion configurations. After setting the configuration, use the function [ISI\\_EnableColorSpaceConversion](#) to enable this feature. If the ISI channel is bypassed, then the color space conversion feature could not be used.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | ISI peripheral base address             |
| <i>config</i> | Pointer to the configuration structure. |

### 20.6.13 void ISI\_ColorSpaceConversionGetDefaultConfig ( isi\_csc\_config\_t \* *config* )

The default value is:

```
config->mode = kISI_CscYUV2RGB;
config->A1 = 0.0;
config->A2 = 0.0;
config->A3 = 0.0;
config->B1 = 0.0;
config->B2 = 0.0;
config->B3 = 0.0;
config->C1 = 0.0;
config->C2 = 0.0;
config->C3 = 0.0;
config->D1 = 0;
config->D2 = 0;
config->D3 = 0;
```

## Parameters

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

### 20.6.14 static void ISI\_EnableColorSpaceConversion ( ISI\_Type \* *base*, bool *enable* ) [inline], [static]

If the ISI channel is bypassed, then the color space conversion feature could not be used even enable using this function.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | ISI peripheral base address       |
| <i>enable</i> | True to enable, false to disable. |

## Note

The CSC is enabled by default. Disable it if it is not required.

### 20.6.15 void ISI\_SetCropConfig ( ISI\_Type \* *base*, const isi\_crop\_config\_t \* *config* )

This function sets the cropping configurations. After setting the configuration, use the function [ISI\\_EnableCrop](#) to enable the feature. Cropping still works when the ISI channel is bypassed.

## Parameters

## Function Documentation

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | ISI peripheral base address             |
| <i>config</i> | Pointer to the configuration structure. |

### Note

The upper left corner and lower right corner should be configured base on the image resolution output from the scaler.

### **20.6.16 void ISI\_CropGetDefaultConfig ( *isi\_crop\_config\_t* \* *config* )**

The default value is:

```
config->upperLeftX = 0U;
config->upperLeftY = 0U;
config->lowerRightX = 0U;
config->lowerRightY = 0U;
```

### Parameters

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

### **20.6.17 static void ISI\_EnableCrop ( *ISI\_Type* \* *base*, *bool enable* ) [inline], [static]**

If the ISI channel is bypassed, the cropping still works.

### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | ISI peripheral base address       |
| <i>enable</i> | True to enable, false to disable. |

### **20.6.18 static void ISI\_SetGlobalAlpha ( *ISI\_Type* \* *base*, *uint8\_t alpha* ) [inline], [static]**

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | ISI peripheral base address |
| <i>alpha</i> | The global alpha value.     |

#### 20.6.19 static void ISI\_EnableGlobalAlpha ( ISI\_Type \* *base*, bool *enable* ) [inline], [static]

Alpha still works when channel bypassed.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | ISI peripheral base address       |
| <i>enable</i> | True to enable, false to disable. |

#### 20.6.20 void ISI\_SetRegionAlphaConfig ( ISI\_Type \* *base*, uint8\_t *index*, const isi\_region\_alpha\_config\_t \* *config* )

Set the alpha insertion configuration for specific region of interest. The function [ISI\\_EnableRegionAlpha](#) could be used to enable the alpha insertion. Alpha insertion still works when channel bypassed.

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | ISI peripheral base address                               |
| <i>index</i>  | Index of the region of interest, Could be 0, 1, 2, and 3. |
| <i>config</i> | Pointer to the configuration structure.                   |

Note

The upper left corner and lower right corner should be configured base on the image resolution output from the scaler.

#### 20.6.21 void ISI\_RegionAlphaGetDefaultConfig ( isi\_region\_alpha\_config\_t \* *config* )

The default configuration is:

```
config->upperLeftX = 0U;
config->upperLeftY = 0U;
config->lowerRightX = 0U;
```

## Function Documentation

```
config->lowerRightY = 0U;
config->alpha = 0U;
```

Parameters

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

### 20.6.22 void ISI\_EnableRegionAlpha ( ISI\_Type \* *base*, uint8\_t *index*, bool *enable* )

Alpha insertion still works when channel bypassed.

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | ISI peripheral base address                               |
| <i>index</i>  | Index of the region of interest, Could be 0, 1, 2, and 3. |
| <i>enable</i> | True to enable, false to disable.                         |

### 20.6.23 void ISI\_SetInputMemConfig ( ISI\_Type \* *base*, const isi\_input\_mem\_config\_t \* *config* )

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | ISI peripheral base address             |
| <i>config</i> | Pointer to the configuration structure. |

### 20.6.24 void ISI\_InputMemGetDefaultConfig ( isi\_input\_mem\_config\_t \* *config* )

The default configuration is:

```
config->addr = 0U;
config->linePitchBytes = 0U;
config->framePitchBytes = 0U;
config->format = kISI_InputMemBGR8P;
```

Parameters

---

## Function Documentation

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

### 20.6.25 static void ISI\_SetInputMemAddr ( ISI\_Type \* *base*, uint32\_t *addr* ) [inline], [static]

This function only sets the input memory address, it is used for fast run-time setting.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | ISI peripheral base address |
| <i>addr</i> | Input memory address.       |

### 20.6.26 void ISI\_TriggerInputMemRead ( ISI\_Type \* *base* )

Parameters

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

### 20.6.27 static void ISI\_SetFlipMode ( ISI\_Type \* *base*, isi\_flip\_mode\_t *mode* ) [inline], [static]

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | ISI peripheral base address |
| <i>mode</i> | Flipping mode.              |

### 20.6.28 void ISI\_SetOutputBufferAddr ( ISI\_Type \* *base*, uint8\_t *index*, uint32\_t *addrY*, uint32\_t *addrU*, uint32\_t *addrV* )

This function sets the output buffer address and trigger the ISI to shadow the address, it is used for fast run-time setting.

Parameters

|              |                                              |
|--------------|----------------------------------------------|
| <i>base</i>  | ISI peripheral base address                  |
| <i>index</i> | Index of output buffer, could be 0 and 1.    |
| <i>addrY</i> | RGB or Luma (Y) output buffer address.       |
| <i>addrU</i> | Chroma (U/Cb/UV/CbCr) output buffer address. |
| <i>addrV</i> | Chroma (V/Cr) output buffer address.         |

### 20.6.29 static void ISI\_Start ( ISI\_Type \* *base* ) [inline], [static]

Start the ISI channel to work, this function should be called after all channel configuration finished.

Parameters

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

### 20.6.30 static void ISI\_Stop ( ISI\_Type \* *base* ) [inline], [static]

Parameters

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

## Function Documentation

# Chapter 21

## LDB: LVDS Display Bridge

### 21.1 Overview

#### Modules

- LDB Driver

#### Files

- file `fsl_ldb.h`

#### Data Structures

- struct `ldb_channel_config_t`  
*LDB channel configuration.* [More...](#)

#### Enumerations

- enum `ldb_output_bus_t`  
*LDB output bus format.*
- enum `_ldb_input_flag` {  
  `kLDB_InputVsyncActiveLow` = 0U,  
  `kLDB_InputVsyncActiveHigh` = 1U << 0U,  
  `kLDB_InputHsyncActiveLow` = 0U,  
  `kLDB_InputHsyncActiveHigh` = 1U << 1U,  
  `kLDB_InputDataLatchOnFallingEdge` = 0U,  
  `kLDB_InputDataLatchOnRisingEdge` = 1U << 2U }  
*LDB input signal priority.*

#### Functions

- void `LDB_Init` (`LDB_Type` \*base)  
*Initializes the LDB module.*
- void `LDB_Deinit` (`LDB_Type` \*base)  
*De-initializes the LDB module.*
- status\_t `LDB_InitChannel` (`LDB_Type` \*base, `uint8_t` channel, const `ldb_channel_config_t` \*config)  
*Initializes the LDB channel.*
- void `LDB_DeinitChannel` (`LDB_Type` \*base, `uint8_t` channel)  
*De-initializes the LDB channel.*

#### Driver version

- #define `FSL_LDB_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
*LDB driver version 2.0.0.*

## Function Documentation

### 21.2 Data Structure Documentation

#### 21.2.1 struct ldb\_channel\_config\_t

##### Data Fields

- `ldb_output_bus_t outputBus`  
*Output bus format.*
- `uint32_t inputFlag`  
*Input flag, OR'ed value of \_ldb\_input\_flag.*
- `uint32_t pixelClock_Hz`  
*Pixel clock in Hz.*

##### 21.2.1.0.0.60 Field Documentation

###### 21.2.1.0.0.60.1 `ldb_output_bus_t ldb_channel_config_t::outputBus`

###### 21.2.1.0.0.60.2 `uint32_t ldb_channel_config_t::inputFlag`

###### 21.2.1.0.0.60.3 `uint32_t ldb_channel_config_t::pixelClock_Hz`

### 21.3 Macro Definition Documentation

#### 21.3.1 `#define FSL_LDB_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

### 21.4 Enumeration Type Documentation

#### 21.4.1 `enum ldb_output_bus_t`

#### 21.4.2 `enum _ldb_input_flag`

Enumerator

`kLDB_InputVsyncActiveLow` VSYNC active low.

`kLDB_InputVsyncActiveHigh` VSYNC active high.

`kLDB_InputHsyncActiveLow` HSYNC active low.

`kLDB_InputHsyncActiveHigh` HSYNC active high.

`kLDB_InputDataLatchOnFallingEdge` Latch data on falling clock edge.

`kLDB_InputDataLatchOnRisingEdge` Latch data on rising clock edge.

### 21.5 Function Documentation

#### 21.5.1 `void LDB_Init ( LDB_Type * base )`

Parameters

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

### 21.5.2 void LDB\_Deinit ( LDB\_Type \* *base* )

Parameters

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

### 21.5.3 status\_t LDB\_InitChannel ( LDB\_Type \* *base*, uint8\_t *channel*, const ldb\_channel\_config\_t \* *config* )

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | LDB peripheral base address.  |
| <i>channel</i> | Channel index.                |
| <i>config</i>  | Pointer to the configuration. |

Returns

Return kStatus\_Success if success.

### 21.5.4 void LDB\_DeinitChannel ( LDB\_Type \* *base*, uint8\_t *channel* )

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | LDB peripheral base address. |
| <i>channel</i> | Channel index.               |

### 21.6 LDB Driver

The MCUXpresso SDK provides a peripheral driver for the LVDS Display Bridge (LDB) module of MCUXpresso SDK devices.

SDK provides the APIs to initialize the LDB and configure the LDB channel. Refer the example for details.

# Chapter 22

## LPADC: 12-bit SAR Analog-to-Digital Converter Driver

### 22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit SAR Analog-to-Digital Converter (LP-ADC) module of MCUXpresso SDK devices.

### 22.2 Typical use case

#### 22.2.1 Polling Configuration

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

#### 22.2.2 Interrupt Configuration

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

### Files

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

### Data Structures

- struct [lpadc\\_config\\_t](#)  
*LPADC global configuration. [More...](#)*
- struct [lpadc\\_conv\\_command\\_config\\_t](#)  
*Define structure to keep the configuration for conversion command. [More...](#)*
- struct [lpadc\\_conv\\_trigger\\_config\\_t](#)  
*Define structure to keep the configuration for conversion trigger. [More...](#)*
- struct [lpadc\\_conv\\_result\\_t](#)  
*Define the structure to keep the conversion result. [More...](#)*

### Macros

- #define [LPADC\\_GET\\_ACTIVE\\_COMMAND\\_STATUS](#)(statusVal) ((statusVal & ADC\_STAT\_CMDACT\_MASK) >> ADC\_STAT\_CMDACT\_SHIFT)  
*Define the MACRO function to get command status from status value.*
- #define [LPADC\\_GET\\_ACTIVE\\_TRIGGER\\_STATUS](#)(statusVal) ((statusVal & ADC\_STAT\_TRGACT\_MASK) >> ADC\_STAT\_TRGACT\_SHIFT)  
*Define the MACRO function to get trigger status from status value.*

## Typical use case

### Enumerations

- enum `_lpadc_status_flags` {  
  `kLPADC_ResultFIFOOverflowFlag` = `ADC_STAT_FOF_MASK`,  
  `kLPADC_ResultFIFOReadyFlag` = `ADC_STAT_RDY_MASK` }  
    *Define hardware flags of the module.*
- enum `_lpadc_interrupt_enable` {  
  `kLPADC_ResultFIFOOverflowInterruptEnable` = `ADC_IE_FOFIE_MASK`,  
  `kLPADC_FIFOWatermarkInterruptEnable` = `ADC_IE_FWMIE_MASK` }  
    *Define interrupt switchers of the module.*
- enum `lpadc_sample_scale_mode_t` {  
  `kLPADC_SamplePartScale` = `0U`,  
  `kLPADC_SampleFullScale` = `1U` }  
    *Define enumeration of sample scale mode.*
- enum `lpadc_sample_channel_mode_t` {  
  `kLPADC_SampleChannelSingleEndSideA` = `0U`,  
  `kLPADC_SampleChannelSingleEndSideB` = `1U`,  
  `kLPADC_SampleChannelDiffBothSideAB` = `2U`,  
  `kLPADC_SampleChannelDiffBothSideBA` = `3U` }  
    *Define enumeration of channel sample mode.*
- enum `lpadc_hardware_average_mode_t` {  
  `kLPADC_HardwareAverageCount1` = `0U`,  
  `kLPADC_HardwareAverageCount2` = `1U`,  
  `kLPADC_HardwareAverageCount4` = `2U`,  
  `kLPADC_HardwareAverageCount8` = `3U`,  
  `kLPADC_HardwareAverageCount16` = `4U`,  
  `kLPADC_HardwareAverageCount32` = `5U`,  
  `kLPADC_HardwareAverageCount64` = `6U`,  
  `kLPADC_HardwareAverageCount128` = `7U` }  
    *Define enumeration of hardware average selection.*
- enum `lpadc_sample_time_mode_t` {  
  `kLPADC_SampleTimeADCK3` = `0U`,  
  `kLPADC_SampleTimeADCK5` = `1U`,  
  `kLPADC_SampleTimeADCK7` = `2U`,  
  `kLPADC_SampleTimeADCK11` = `3U`,  
  `kLPADC_SampleTimeADCK19` = `4U`,  
  `kLPADC_SampleTimeADCK35` = `5U`,  
  `kLPADC_SampleTimeADCK67` = `6U`,  
  `kLPADC_SampleTimeADCK131` = `7U` }  
    *Define enumeration of sample time selection.*
- enum `lpadc_hardware_compare_mode_t` {  
  `kLPADC_HardwareCompareDisabled` = `0U`,  
  `kLPADC_HardwareCompareStoreOnTrue` = `2U`,  
  `kLPADC_HardwareCompareRepeatUntilTrue` = `3U` }  
    *Define enumeration of hardware compare mode.*
- enum `lpadc_conversion_resolution_mode_t` {  
  `kLPADC_ConversionResolutionStandard` = `0U`,

- `kLPADC_ConversionResolutionHigh = 1U }`  
*Define enumeration of conversion resolution mode.*
- enum `lpadc_reference_voltage_source_t` {
 `kLPADC_ReferenceVoltageAlt1 = 0U,`  
`kLPADC_ReferenceVoltageAlt2 = 1U,`  
`kLPADC_ReferenceVoltageAlt3 = 2U }`  
*Define enumeration of reference voltage source.*
- enum `lpadc_power_level_mode_t` {
 `kLPADC_PowerLevelAlt1 = 0U,`  
`kLPADC_PowerLevelAlt2 = 1U,`  
`kLPADC_PowerLevelAlt3 = 2U,`  
`kLPADC_PowerLevelAlt4 = 3U }`  
*Define enumeration of power configuration.*
- enum `lpadc_trigger_priority_policy_t` {
 `kLPADC_TriggerPriorityPreemptImmediately = 0U,`  
`kLPADC_TriggerPriorityPreemptSoftly = 1U,`  
`kLPADC_TriggerPriorityPreemptSubsequently }`  
*Define enumeration of trigger priority policy.*

## Driver version

- #define `FSL_LPADC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 3))  
*LPADC driver version 2.0.3.*

## Initialization & de-initialization.

- void `LPADC_Init` (ADC\_Type \*base, const `lpadc_config_t` \*config)  
*Initializes the LPADC module.*
- void `LPADC_GetDefaultConfig` (`lpadc_config_t` \*config)  
*Gets an available pre-defined settings for initial configuration.*
- void `LPADC_Deinit` (ADC\_Type \*base)  
*De-initializes the LPADC module.*
- static void `LPADC_Enable` (ADC\_Type \*base, bool enable)  
*Switch on/off the LPADC module.*
- static void `LPADC_DoResetFIFO` (ADC\_Type \*base)  
*Do reset the conversion FIFO.*
- static void `LPADC_DoResetConfig` (ADC\_Type \*base)  
*Do reset the module's configuration.*

## Status

- static uint32\_t `LPADC_GetStatusFlags` (ADC\_Type \*base)  
*Get status flags.*
- static void `LPADC_ClearStatusFlags` (ADC\_Type \*base, uint32\_t mask)  
*Clear status flags.*

## Interrupts

- static void `LPADC_EnableInterrupts` (ADC\_Type \*base, uint32\_t mask)

## Data Structure Documentation

- static void [LPADC\\_DisableInterrupts](#) (ADC\_Type \*base, uint32\_t mask)  
*Disable interrupts.*

## DMA Control

- static void [LPADC\\_EnableFIFOWatermarkDMA](#) (ADC\_Type \*base, bool enable)  
*Switch on/off the DMA trigger for FIFO watermark event.*

## Trigger and conversion with FIFO.

- static uint32\_t [LPADC\\_GetConvResultCount](#) (ADC\_Type \*base)  
*Get the count of result kept in conversion FIFO.*
- bool [LPADC\\_GetConvResult](#) (ADC\_Type \*base, [lpadc\\_conv\\_result\\_t](#) \*result)  
*Get the result in conversion FIFO.*
- void [LPADC\\_SetConvTriggerConfig](#) (ADC\_Type \*base, uint32\_t triggerId, const [lpadc\\_conv\\_trigger\\_config\\_t](#) \*config)  
*Configure the conversion trigger source.*
- void [LPADC\\_GetDefaultConvTriggerConfig](#) ([lpadc\\_conv\\_trigger\\_config\\_t](#) \*config)  
*Gets an available pre-defined settings for trigger's configuration.*
- static void [LPADC\\_DoSoftwareTrigger](#) (ADC\_Type \*base, uint32\_t triggerIdMask)  
*Do software trigger to conversion command.*
- void [LPADC\\_SetConvCommandConfig](#) (ADC\_Type \*base, uint32\_t commandId, const [lpadc\\_conv\\_command\\_config\\_t](#) \*config)  
*Configure conversion command.*
- void [LPADC\\_GetDefaultConvCommandConfig](#) ([lpadc\\_conv\\_command\\_config\\_t](#) \*config)  
*Gets an available pre-defined settings for conversion command's configuration.*

## 22.3 Data Structure Documentation

### 22.3.1 struct [lpadc\\_config\\_t](#)

This structure would used to keep the settings for initialization.

## Data Fields

- bool [enableInDozeMode](#)  
*Control system transition to Stop and Wait power modes while ADC is converting.*
- bool [enableAnalogPreliminary](#)  
*ADC analog circuits are pre-enabled and ready to execute conversions without startup delays(at the cost of higher DC current consumption).*
- uint32\_t [powerUpDelay](#)  
*When the analog circuits are not pre-enabled, the ADC analog circuits are only powered while the ADC is active and there is a counted delay defined by this field after an initial trigger transitions the ADC from its Idle state to allow time for the analog circuits to stabilize.*
- [lpadc\\_reference\\_voltage\\_source\\_t](#) [referenceVoltageSource](#)  
*Selects the voltage reference high used for conversions.*
- [lpadc\\_power\\_level\\_mode\\_t](#) [powerLevelMode](#)

*Power Configuration Selection.*

- [lpadc\\_trigger\\_priority\\_policy\\_t triggerPriorityPolicy](#)  
*Control how higher priority triggers are handled, see to #lpadc\_trigger\_priority\_policy\_mode\_t.*
- bool [enableConvPause](#)  
*Enables the ADC pausing function.*
- uint32\_t [convPauseDelay](#)  
*Controls the duration of pausing during command execution sequencing.*
- uint32\_t [FIFOWatermark](#)  
*FIFOWatermark is a programmable threshold setting.*

### 22.3.1.0.0.61 Field Documentation

#### 22.3.1.0.0.61.1 bool lpadc\_config\_t::enableInDozeMode

When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

#### 22.3.1.0.0.61.2 bool lpadc\_config\_t::enableAnalogPreliminary

#### 22.3.1.0.0.61.3 uint32\_t lpadc\_config\_t::powerUpDelay

The startup delay count of (powerUpDelay \* 4) ADCK cycles must result in a longer delay than the analog startup time.

#### 22.3.1.0.0.61.4 lpadc\_reference\_voltage\_source\_t lpadc\_config\_t::referenceVoltageSource

#### 22.3.1.0.0.61.5 lpadc\_power\_level\_mode\_t lpadc\_config\_t::powerLevelMode

#### 22.3.1.0.0.61.6 lpadc\_trigger\_priority\_policy\_t lpadc\_config\_t::triggerPriorityPolicy

#### 22.3.1.0.0.61.7 bool lpadc\_config\_t::enableConvPause

When enabled, a programmable delay is inserted during command execution sequencing between LOOP iterations, between commands in a sequence, and between conversions when command is executing in "Compare Until True" configuration.

#### 22.3.1.0.0.61.8 uint32\_t lpadc\_config\_t::convPauseDelay

The pause delay is a count of (convPauseDelay\*4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

#### 22.3.1.0.0.61.9 uint32\_t lpadc\_config\_t::FIFOWatermark

When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

## Data Structure Documentation

### 22.3.2 struct lpadc\_conv\_command\_config\_t

#### Data Fields

- `lpadc_sample_scale_mode_t sampleScaleMode`  
*Sample scale mode.*
- `lpadc_sample_channel_mode_t sampleChannelMode`  
*Channel sample mode.*
- `uint32_t channelNumber`  
*Channel number, select the channel or channel pair.*
- `uint32_t chainedNextCommandNumber`  
*Selects the next command to be executed after this command completes.*
- `bool enableAutoChannelIncrement`  
*Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.*
- `uint32_t loopCount`  
*Selects how many times this command executes before finish and transition to the next command or Idle state.*
- `lpadc_hardware_average_mode_t hardwareAverageMode`  
*Hardware average selection.*
- `lpadc_sample_time_mode_t sampleTimeMode`  
*Sample time selection.*
- `lpadc_hardware_compare_mode_t hardwareCompareMode`  
*Hardware compare selection.*
- `uint32_t hardwareCompareValueHigh`  
*Compare Value High.*
- `uint32_t hardwareCompareValueLow`  
*Compare Value Low.*

#### 22.3.2.0.0.62 Field Documentation

##### 22.3.2.0.0.62.1 `lpadc_sample_scale_mode_t lpadc_conv_command_config_t::sampleScaleMode`

##### 22.3.2.0.0.62.2 `lpadc_sample_channel_mode_t lpadc_conv_command_config_t::sampleChannelMode`

##### 22.3.2.0.0.62.3 `uint32_t lpadc_conv_command_config_t::channelNumber`

##### 22.3.2.0.0.62.4 `uint32_t lpadc_conv_command_config_t::chainedNextCommandNumber`

1-15 is available, 0 is to terminate the chain after this command.

##### 22.3.2.0.0.62.5 `bool lpadc_conv_command_config_t::enableAutoChannelIncrement`

##### 22.3.2.0.0.62.6 `uint32_t lpadc_conv_command_config_t::loopCount`

Command executes LOOP+1 times. 0-15 is available.

**22.3.2.0.0.62.7 `lpadc_hardware_average_mode_t lpadc_conv_command_config_t::hardwareAverageMode`**

**22.3.2.0.0.62.8 `lpadc_sample_time_mode_t lpadc_conv_command_config_t::sampleTimeMode`**

**22.3.2.0.0.62.9 `lpadc_hardware_compare_mode_t lpadc_conv_command_config_t::hardwareCompareMode`**

**22.3.2.0.0.62.10 `uint32_t lpadc_conv_command_config_t::hardwareCompareValueHigh`**

The available value range is in 16-bit.

**22.3.2.0.0.62.11 `uint32_t lpadc_conv_command_config_t::hardwareCompareValueLow`**

The available value range is in 16-bit.

## 22.3.3 struct `lpadc_conv_trigger_config_t`

### Data Fields

- `uint32_t targetCommandId`  
*Select the command from command buffer to execute upon detect of the associated trigger event.*
- `uint32_t delayPower`  
*Select the trigger delay duration to wait at the start of servicing a trigger event.*
- `uint32_t priority`  
*Sets the priority of the associated trigger source.*
- `bool enableHardwareTrigger`  
*Enable hardware trigger source to initiate conversion on the rising edge of the input trigger source or not.*

### 22.3.3.0.0.63 Field Documentation

**22.3.3.0.0.63.1 `uint32_t lpadc_conv_trigger_config_t::targetCommandId`**

**22.3.3.0.0.63.2 `uint32_t lpadc_conv_trigger_config_t::delayPower`**

When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is  $2^{\text{delayPower}}$  ADCK cycles. The available value range is 4-bit.

**22.3.3.0.0.63.3 `uint32_t lpadc_conv_trigger_config_t::priority`**

If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

**22.3.3.0.0.63.4 `bool lpadc_conv_trigger_config_t::enableHardwareTrigger`**

The software trigger is always available.

## Enumeration Type Documentation

### 22.3.4 struct lpadc\_conv\_result\_t

#### Data Fields

- `uint32_t commandIdSource`  
*Indicate the command buffer being executed that generated this result.*
- `uint32_t loopCountIndex`  
*Indicate the loop count value during command execution that generated this result.*
- `uint32_t triggerIdSource`  
*Indicate the trigger source that initiated a conversion and generated this result.*
- `uint16_t convValue`  
*Data result.*

#### 22.3.4.0.0.64 Field Documentation

22.3.4.0.0.64.1 `uint32_t lpadc_conv_result_t::commandIdSource`

22.3.4.0.0.64.2 `uint32_t lpadc_conv_result_t::loopCountIndex`

22.3.4.0.0.64.3 `uint32_t lpadc_conv_result_t::triggerIdSource`

22.3.4.0.0.64.4 `uint16_t lpadc_conv_result_t::convValue`

#### 22.4 Macro Definition Documentation

22.4.1 `#define FSL_LPADC_DRIVER_VERSION(MAKE_VERSION(2, 0, 3))`

22.4.2 `#define LPADC_GET_ACTIVE_COMMAND_STATUS( statusVal ) ((statusVal & ADC_STAT_CMDACT_MASK) >> ADC_STAT_CMDACT_SHIFT)`

The statusVal is the return value from [LPADC\\_GetStatusFlags\(\)](#).

22.4.3 `#define LPADC_GET_ACTIVE_TRIGGER_STATUS( statusVal ) ((statusVal & ADC_STAT_TRGACT_MASK) >> ADC_STAT_TRGACT_SHIFT)`

The statusVal is the return value from [LPADC\\_GetStatusFlags\(\)](#).

#### 22.5 Enumeration Type Documentation

##### 22.5.1 enum \_lpadc\_status\_flags

Enumerator

`kLPADC_ResultFIFOOverflowFlag` Indicates that more data has been written to the Result FIFO than it can hold.

`kLPADC_ResultFIFOReadyFlag` Indicates when the number of valid datawords in the result FIFO is greater than the setting watermark level.

## 22.5.2 enum \_lpadc\_interrupt\_enable

Enumerator

***kLPADC\_ResultFIFOOverflowInterruptEnable*** Configures ADC to generate overflow interrupt requests when FOF flag is asserted.

***kLPADC\_FIFOWatermarkInterruptEnable*** Configures ADC to generate watermark interrupt requests when RDY flag is asserted.

## 22.5.3 enum lpadc\_sample\_scale\_mode\_t

The sample scale mode is used to reduce the selected ADC analog channel input voltage level by a factor. The maximum possible voltage on the ADC channel input should be considered when selecting a scale mode to ensure that the reducing factor always results voltage level at or below the VREFH reference. This reducing capability allows conversion of analog inputs higher than VREFH. A-side and B-side channel inputs are both scaled using the scale mode.

Enumerator

***kLPADC\_SamplePartScale*** Use divided input voltage signal. (Factor of 30/64).

***kLPADC\_SampleFullScale*** Full scale (Factor of 1).

## 22.5.4 enum lpadc\_sample\_channel\_mode\_t

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Enumerator

***kLPADC\_SampleChannelSingleEndSideA*** Single end mode, using side A.

***kLPADC\_SampleChannelSingleEndSideB*** Single end mode, using side B.

***kLPADC\_SampleChannelDiffBothSideAB*** Differential mode, using A as plus side and B as minus side.

***kLPADC\_SampleChannelDiffBothSideBA*** Differential mode, using B as plus side and A as minus side.

## 22.5.5 enum lpadc\_hardware\_average\_mode\_t

It Selects how many ADC conversions are averaged to create the ADC result. An internal storage buffer is used to capture temporary results while the averaging iterations are executed.

Enumerator

***kLPADC\_HardwareAverageCount1*** Single conversion.

## Enumeration Type Documentation

*kLPADC\_HardwareAverageCount2* 2 conversions averaged.  
*kLPADC\_HardwareAverageCount4* 4 conversions averaged.  
*kLPADC\_HardwareAverageCount8* 8 conversions averaged.  
*kLPADC\_HardwareAverageCount16* 16 conversions averaged.  
*kLPADC\_HardwareAverageCount32* 32 conversions averaged.  
*kLPADC\_HardwareAverageCount64* 64 conversions averaged.  
*kLPADC\_HardwareAverageCount128* 128 conversions averaged.

### 22.5.6 enum lpadc\_sample\_time\_mode\_t

The shortest sample time maximizes conversion speed for lower impedance inputs. Extending sample time allows higher impedance inputs to be accurately sampled. Longer sample times can also be used to lower overall power consumption when command looping and sequencing is configured and high conversion rates are not required.

Enumerator

*kLPADC\_SampleTimeADCK3* 3 ADCK cycles total sample time.  
*kLPADC\_SampleTimeADCK5* 5 ADCK cycles total sample time.  
*kLPADC\_SampleTimeADCK7* 7 ADCK cycles total sample time.  
*kLPADC\_SampleTimeADCK11* 11 ADCK cycles total sample time.  
*kLPADC\_SampleTimeADCK19* 19 ADCK cycles total sample time.  
*kLPADC\_SampleTimeADCK35* 35 ADCK cycles total sample time.  
*kLPADC\_SampleTimeADCK67* 69 ADCK cycles total sample time.  
*kLPADC\_SampleTimeADCK131* 131 ADCK cycles total sample time.

### 22.5.7 enum lpadc\_hardware\_compare\_mode\_t

After an ADC channel input is sampled and converted and any averaging iterations are performed, this mode setting guides operation of the automatic compare function to optionally only store when the compare operation is true. When compare is enabled, the conversion result is compared to the compare values.

Enumerator

*kLPADC\_HardwareCompareDisabled* Compare disabled.  
*kLPADC\_HardwareCompareStoreOnTrue* Compare enabled. Store on true.  
*kLPADC\_HardwareCompareRepeatUntilTrue* Compare enabled. Repeat channel acquisition until true.

### 22.5.8 enum lpadc\_conversion\_resolution\_mode\_t

Configure the resolution bit in specific conversion type. For detailed resolution accuracy, see to #\_lpadc\_sample\_channel\_mode

Enumerator

***kLPADC\_ConversionResolutionStandard*** Standard resolution. Single-ended 12-bit conversion, Differential 13-bit conversion with 2's complement output.

***kLPADC\_ConversionResolutionHigh*** High resolution. Single-ended 16-bit conversion; Differential 16-bit conversion with 2's complement output.

## 22.5.9 enum lpadc\_reference\_voltage\_source\_t

For detail information, need to check the SoC's specification.

Enumerator

***kLPADC\_ReferenceVoltageAlt1*** Option 1 setting.

***kLPADC\_ReferenceVoltageAlt2*** Option 2 setting.

***kLPADC\_ReferenceVoltageAlt3*** Option 3 setting.

## 22.5.10 enum lpadc\_power\_level\_mode\_t

Configures the ADC for power and performance. In the highest power setting the highest conversion rates will be possible. Refer to the device data sheet for power and performance capabilities for each setting.

Enumerator

***kLPADC\_PowerLevelAlt1*** Lowest power setting.

***kLPADC\_PowerLevelAlt2*** Next lowest power setting.

***kLPADC\_PowerLevelAlt3*** ...

***kLPADC\_PowerLevelAlt4*** Highest power setting.

## 22.5.11 enum lpadc\_trigger\_priority\_policy\_t

This selection controls how higher priority triggers are handled.

Enumerator

***kLPADC\_TriggerPriorityPreemptImmediately*** If a higher priority trigger is detected during command processing, the current conversion is aborted and the new command specified by the trigger is started.

***kLPADC\_TriggerPriorityPreemptSoftly*** If a higher priority trigger is received during command processing, the current conversion is completed (including averaging iterations and compare function if enabled) and stored to the result FIFO before the higher priority trigger/command is initiated.

***kLPADC\_TriggerPriorityPreemptSubsequently*** If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger.

## Function Documentation

### 22.6 Function Documentation

#### 22.6.1 void LPADC\_Init ( ADC\_Type \* *base*, const lpadc\_config\_t \* *config* )

Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | LPADC peripheral base address.                            |
| <i>config</i> | Pointer to configuration structure. See "lpadc_config_t". |

#### 22.6.2 void LPADC\_GetDefaultConfig ( lpadc\_config\_t \* *config* )

This function initializes the converter configuration structure with an available settings. The default values are:

```
* config->enableInDozeMode = true;
* config->enableAnalogPreliminary = false;
* config->powerUpDelay = 0x80;
* config->referenceVoltageSource = kLPADC_ReferenceVoltageAlt1;
* config->powerLevelMode = kLPADC_PowerLevelAlt1;
* config->triggerPriorityPolicy = kLPADC_TriggerPriorityPreemptImmediately
 ;
* config->enableConvPause = false;
* config->convPauseDelay = 0U;
* config->FIFOWatermark = 0U;
```

Parameters

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

#### 22.6.3 void LPADC\_Deinit ( ADC\_Type \* *base* )

Parameters

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

#### 22.6.4 static void LPADC\_Enable ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>enable</i> | switcher to the module.        |

## 22.6.5 static void LPADC\_DoResetFIFO ( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

## 22.6.6 static void LPADC\_DoResetConfig ( ADC\_Type \* *base* ) [inline], [static]

Reset all ADC internal logic and registers, except the Control Register (ADCx\_CTRL).

Parameters

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

## 22.6.7 static uint32\_t LPADC\_GetStatusFlags ( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

status flags' mask. See to [\\_lpadc\\_status\\_flags](#).

## 22.6.8 static void LPADC\_ClearStatusFlags ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Only the flags can be cleared by writing ADCx\_STATUS register would be cleared by this API.

## Function Documentation

Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                   |
| <i>mask</i> | Mask value for flags to be cleared. See to <a href="#">_lpadc_status_flags</a> . |

**22.6.9 static void LPADC\_EnableInterrupts ( ADC\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address. Mask value for interrupt events. See to <a href="#">_lpadc_interrupt_enable</a> . |
|-------------|------------------------------------------------------------------------------------------------------------------|

**22.6.10 static void LPADC\_DisableInterrupts ( ADC\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | LPADC peripheral base address.                                                    |
| <i>mask</i> | Mask value for interrupt events. See to <a href="#">_lpadc_interrupt_enable</a> . |

**22.6.11 static void LPADC\_EnableFIFOWatermarkDMA ( ADC\_Type \* *base*, bool  
*enable* ) [inline], [static]**

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPADC peripheral base address. |
| <i>enable</i> | Switcher to the event.         |

**22.6.12 static uint32\_t LPADC\_GetConvResultCount ( ADC\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

The count of result kept in conversion FIFO.

### 22.6.13 **bool LPADC\_GetConvResult ( ADC\_Type \* *base*, lpadc\_conv\_result\_t \* *result* )**

Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | LPADC peripheral base address.                                                     |
| <i>result</i> | Pointer to structure variable that keeps the conversion result in conversion FIFO. |

Returns

Status whether FIFO entry is valid.

### 22.6.14 **void LPADC\_SetConvTriggerConfig ( ADC\_Type \* *base*, uint32\_t *triggerId*, const lpadc\_conv\_trigger\_config\_t \* *config* )**

Each programmable trigger can launch the conversion command in command buffer.

Parameters

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| <i>base</i>      | LPADC peripheral base address.                                                           |
| <i>triggerId</i> | ID for each trigger. Typically, the available value range is from 0.                     |
| <i>config</i>    | Pointer to configuration structure. See to <a href="#">lpadc_conv_trigger_config_t</a> . |

### 22.6.15 **void LPADC\_GetDefaultConvTriggerConfig ( lpadc\_conv\_trigger\_config\_t \* *config* )**

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
* config->commandIdSource = 0U;
* config->loopCountIndex = 0U;
* config->triggerIdSource = 0U;
* config->enableHardwareTrigger = false;
*
```

## Function Documentation

Parameters

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

### 22.6.16 static void LPADC\_DoSoftwareTrigger ( ADC\_Type \* *base*, uint32\_t *triggerIdMask* ) [inline], [static]

Parameters

|                      |                                                                 |
|----------------------|-----------------------------------------------------------------|
| <i>base</i>          | LPADC peripheral base address.                                  |
| <i>triggerIdMask</i> | Mask value for software trigger indexes, which count from zero. |

### 22.6.17 void LPADC\_SetConvCommandConfig ( ADC\_Type \* *base*, uint32\_t *commandId*, const lpadc\_conv\_command\_config\_t \* *config* )

Parameters

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| <i>base</i>      | LPADC peripheral base address.                                                           |
| <i>commandId</i> | ID for command in command buffer. Typically, the available value range is 1 - 15.        |
| <i>config</i>    | Pointer to configuration structure. See to <a href="#">lpadc_conv_command_config_t</a> . |

### 22.6.18 void LPADC\_GetDefaultConvCommandConfig ( lpadc\_conv\_command\_config\_t \* *config* )

This function initializes the conversion command's configuration structure with an available settings. The default values are:

```
* config->sampleScaleMode = kLPADC_SampleFullScale;
* config->channelSampleMode = kLPADC_SampleChannelSingleEndSideA
 ;
* config->channelNumber = OU;
* config->chainedNextCmdNumber = OU;
* config->enableAutoChannelIncrement = false;
* config->loopCount = OU;
* config->hardwareAverageMode = kLPADC_HardwareAverageCount1;
* config->sampleTimeMode = kLPADC_SampleTimeADCK3;
* config->hardwareCompareMode = kLPADC_HardwareCompareDisabled;
* config->hardwareCompareValueHigh = OU;
* config->hardwareCompareValueLow = OU;
* config->conversionResoultuionMode = kLPADC_ConversionResolutionStandard
 ;
* config->enableWaitTrigger = false;
```

## Parameters

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

## Function Documentation

# Chapter 23

## LPI2C: Low Power I2C Driver

### 23.1 Overview

#### Modules

- LPI2C FreeRTOS Driver
- LPI2C Master DMA Driver
- LPI2C Master Driver
- LPI2C Slave Driver

#### Macros

- #define `LPI2C_WAIT_TIMEOUT` 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Timeout times for waiting flag.*

#### Enumerations

- enum `_lpi2c_status` {  
  `kStatus_LPI2C_Busy` = MAKE\_STATUS(kStatusGroup\_LPI2C, 0),  
  `kStatus_LPI2C_Idle` = MAKE\_STATUS(kStatusGroup\_LPI2C, 1),  
  `kStatus_LPI2C_Nak` = MAKE\_STATUS(kStatusGroup\_LPI2C, 2),  
  `kStatus_LPI2C_FifoError` = MAKE\_STATUS(kStatusGroup\_LPI2C, 3),  
  `kStatus_LPI2C_BitError` = MAKE\_STATUS(kStatusGroup\_LPI2C, 4),  
  `kStatus_LPI2C_ArbitrationLost` = MAKE\_STATUS(kStatusGroup\_LPI2C, 5),  
  `kStatus_LPI2C_PinLowTimeout`,  
  `kStatus_LPI2C_NoTransferInProgress`,  
  `kStatus_LPI2C_DmaRequestFail` = MAKE\_STATUS(kStatusGroup\_LPI2C, 8),  
  `kStatus_LPI2C_Timeout` = MAKE\_STATUS(kStatusGroup\_LPI2C, 9) }  
*LPI2C status return codes.*

#### Driver version

- #define `FSL_LPI2C_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 6))  
*LPI2C driver version 2.1.6.*

### 23.2 Macro Definition Documentation

#### 23.2.1 #define `FSL_LPI2C_DRIVER_VERSION` (MAKE\_VERSION(2, 1, 6))

#### 23.2.2 #define `LPI2C_WAIT_TIMEOUT` 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/

## Enumeration Type Documentation

### 23.3 Enumeration Type Documentation

#### 23.3.1 enum \_lpi2c\_status

Enumerator

*kStatus\_LPI2C\_Busy* The master is already performing a transfer.

*kStatus\_LPI2C\_Idle* The slave driver is idle.

*kStatus\_LPI2C\_Nak* The slave device sent a NAK in response to a byte.

*kStatus\_LPI2C\_FifoError* FIFO under run or overrun.

*kStatus\_LPI2C\_BitError* Transferred bit was not seen on the bus.

*kStatus\_LPI2C\_ArbitrationLost* Arbitration lost error.

*kStatus\_LPI2C\_PinLowTimeout* SCL or SDA were held low longer than the timeout.

*kStatus\_LPI2C\_NoTransferInProgress* Attempt to abort a transfer when one is not in progress.

*kStatus\_LPI2C\_DmaRequestFail* DMA request failed.

*kStatus\_LPI2C\_Timeout* Timeout polling status flags.

## 23.4 LPI2C Master Driver

### 23.4.1 Overview

#### Data Structures

- struct `lpi2c_master_config_t`  
*Structure with settings to initialize the LPI2C master module. [More...](#)*
- struct `lpi2c_data_match_config_t`  
*LPI2C master data match configuration structure. [More...](#)*
- struct `lpi2c_master_transfer_t`  
*Non-blocking transfer descriptor structure. [More...](#)*
- struct `lpi2c_master_handle_t`  
*Driver handle for master non-blocking APIs. [More...](#)*

#### Typedefs

- typedef void(\* `lpi2c_master_transfer_callback_t`)  
`(LPI2C_Type *base, lpi2c_master_handle_t *handle, status_t completionStatus, void *userData)`  
*Master completion callback function pointer type.*

#### Enumerations

- enum `_lpi2c_master_flags` {
   
`kLPI2C_MasterTxReadyFlag` = LPI2C\_MSR\_TDF\_MASK,  
`kLPI2C_MasterRxReadyFlag` = LPI2C\_MSR\_RDF\_MASK,  
`kLPI2C_MasterEndOfPacketFlag` = LPI2C\_MSR\_EPF\_MASK,  
`kLPI2C_MasterStopDetectFlag` = LPI2C\_MSR\_SDF\_MASK,  
`kLPI2C_MasterNackDetectFlag` = LPI2C\_MSR\_NDF\_MASK,  
`kLPI2C_MasterArbitrationLostFlag` = LPI2C\_MSR\_ALF\_MASK,  
`kLPI2C_MasterFifoErrFlag` = LPI2C\_MSR\_FEF\_MASK,  
`kLPI2C_MasterPinLowTimeoutFlag` = LPI2C\_MSR\_PLTF\_MASK,  
`kLPI2C_MasterDataMatchFlag` = LPI2C\_MSR\_DMF\_MASK,  
`kLPI2C_MasterBusyFlag` = LPI2C\_MSR\_MBF\_MASK,  
`kLPI2C_MasterBusBusyFlag` = LPI2C\_MSR\_BBF\_MASK }
   
*LPI2C master peripheral flags.*
- enum `lpi2c_direction_t` {
   
`kLPI2C_Write` = 0U,  
`kLPI2C_Read` = 1U }
   
*Direction of master and slave transfers.*
- enum `lpi2c_master_pin_config_t` {

## LPI2C Master Driver

```
kLPI2C_2PinOpenDrain = 0x0U,
kLPI2C_2PinOutputOnly = 0x1U,
kLPI2C_2PinPushPull = 0x2U,
kLPI2C_4PinPushPull = 0x3U,
kLPI2C_2PinOpenDrainWithSeparateSlave,
kLPI2C_2PinOutputOnlyWithSeparateSlave,
kLPI2C_2PinPushPullWithSeparateSlave,
kLPI2C_4PinPushPullWithInvertedOutput = 0x7U }
```

*LPI2C pin configuration.*

- enum `lpi2c_host_request_source_t` {  
  `kLPI2C_HostRequestExternalPin` = 0x0U,  
  `kLPI2C_HostRequestInputTrigger` = 0x1U }

*LPI2C master host request selection.*

- enum `lpi2c_host_request_polarity_t` {  
  `kLPI2C_HostRequestPinActiveLow` = 0x0U,  
  `kLPI2C_HostRequestPinActiveHigh` = 0x1U }

*LPI2C master host request pin polarity configuration.*

- enum `lpi2c_data_match_config_mode_t` {  
  `kLPI2C_MatchDisabled` = 0x0U,  
  `kLPI2C_1stWordEqualsM0OrM1` = 0x2U,  
  `kLPI2C_AnyWordEqualsM0OrM1` = 0x3U,  
  `kLPI2C_1stWordEqualsM0And2ndWordEqualsM1`,  
  `kLPI2C_AnyWordEqualsM0AndNextWordEqualsM1`,  
  `kLPI2C_1stWordAndM1EqualsM0AndM1`,  
  `kLPI2C_AnyWordAndM1EqualsM0AndM1` }

*LPI2C master data match configuration modes.*

- enum `_lpi2c_master_transfer_flags` {  
  `kLPI2C_TransferDefaultFlag` = 0x00U,  
  `kLPI2C_TransferNoStartFlag` = 0x01U,  
  `kLPI2C_TransferRepeatedStartFlag` = 0x02U,  
  `kLPI2C_TransferNoStopFlag` = 0x04U }

*Transfer option flags.*

## Initialization and deinitialization

- void `LPI2C_MasterGetDefaultConfig` (`lpi2c_master_config_t` \*`masterConfig`)  
*Provides a default configuration for the LPI2C master peripheral.*
- void `LPI2C_MasterInit` (`LPI2C_Type` \*`base`, const `lpi2c_master_config_t` \*`masterConfig`, `uint32_t` `sourceClock_Hz`)  
*Initializes the LPI2C master peripheral.*
- void `LPI2C_MasterDeinit` (`LPI2C_Type` \*`base`)  
*Deinitializes the LPI2C master peripheral.*
- void `LPI2C_MasterConfigureDataMatch` (`LPI2C_Type` \*`base`, const `lpi2c_data_match_config_t` \*`config`)  
*Configures LPI2C master data match feature.*
- `status_t` `LPI2C_MasterCheckAndClearError` (`LPI2C_Type` \*`base`, `uint32_t` `status`)

- status\_t **LPI2C\_CheckForBusyBus** (LPI2C\_Type \*base)
- static void **LPI2C\_MasterReset** (LPI2C\_Type \*base)  
*Performs a software reset.*
- static void **LPI2C\_MasterEnable** (LPI2C\_Type \*base, bool enable)  
*Enables or disables the LPI2C module as master.*

## Status

- static uint32\_t **LPI2C\_MasterGetStatusFlags** (LPI2C\_Type \*base)  
*Gets the LPI2C master status flags.*
- static void **LPI2C\_MasterClearStatusFlags** (LPI2C\_Type \*base, uint32\_t statusMask)  
*Clears the LPI2C master status flag state.*

## Interrupts

- static void **LPI2C\_MasterEnableInterrupts** (LPI2C\_Type \*base, uint32\_t interruptMask)  
*Enables the LPI2C master interrupt requests.*
- static void **LPI2C\_MasterDisableInterrupts** (LPI2C\_Type \*base, uint32\_t interruptMask)  
*Disables the LPI2C master interrupt requests.*
- static uint32\_t **LPI2C\_MasterGetEnabledInterrupts** (LPI2C\_Type \*base)  
*Returns the set of currently enabled LPI2C master interrupt requests.*

## DMA control

- static void **LPI2C\_MasterEnableDMA** (LPI2C\_Type \*base, bool enableTx, bool enableRx)  
*Enables or disables LPI2C master DMA requests.*
- static uint32\_t **LPI2C\_MasterGetTxFifoAddress** (LPI2C\_Type \*base)  
*Gets LPI2C master transmit data register address for DMA transfer.*
- static uint32\_t **LPI2C\_MasterGetRxFifoAddress** (LPI2C\_Type \*base)  
*Gets LPI2C master receive data register address for DMA transfer.*

## FIFO control

- static void **LPI2C\_MasterSetWatermarks** (LPI2C\_Type \*base, size\_t txWords, size\_t rxWords)  
*Sets the watermarks for LPI2C master FIFOs.*
- static void **LPI2C\_MasterGetFifoCounts** (LPI2C\_Type \*base, size\_t \*rxCount, size\_t \*txCount)  
*Gets the current number of words in the LPI2C master FIFOs.*

## Bus operations

- void **LPI2C\_MasterSetBaudRate** (LPI2C\_Type \*base, uint32\_t sourceClock\_Hz, uint32\_t baudRate\_Hz)  
*Sets the I2C bus frequency for master transactions.*
- static bool **LPI2C\_MasterGetBusIdleState** (LPI2C\_Type \*base)

## LPI2C Master Driver

*Returns whether the bus is idle.*

- status\_t [LPI2C\\_MasterStart](#) (LPI2C\_Type \*base, uint8\_t address, lpi2c\_direction\_t dir)  
*Sends a START signal and slave address on the I2C bus.*
- static status\_t [LPI2C\\_MasterRepeatedStart](#) (LPI2C\_Type \*base, uint8\_t address, lpi2c\_direction\_t dir)  
*Sends a repeated START signal and slave address on the I2C bus.*
- status\_t [LPI2C\\_MasterSend](#) (LPI2C\_Type \*base, void \*txBuff, size\_t txSize)  
*Performs a polling send transfer on the I2C bus.*
- status\_t [LPI2C\\_MasterReceive](#) (LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize)  
*Performs a polling receive transfer on the I2C bus.*
- status\_t [LPI2C\\_MasterStop](#) (LPI2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- status\_t [LPI2C\\_MasterTransferBlocking](#) (LPI2C\_Type \*base, lpi2c\_master\_transfer\_t \*transfer)  
*Performs a master polling transfer on the I2C bus.*

## Non-blocking

- void [LPI2C\\_MasterTransferCreateHandle](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle, lpi2c\_master\_transfer\_callback\_t callback, void \*userData)  
*Creates a new handle for the LPI2C master non-blocking APIs.*
- status\_t [LPI2C\\_MasterTransferNonBlocking](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle, lpi2c\_master\_transfer\_t \*transfer)  
*Performs a non-blocking transaction on the I2C bus.*
- status\_t [LPI2C\\_MasterTransferGetCount](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle, size\_t \*count)  
*Returns number of bytes transferred so far.*
- void [LPI2C\\_MasterTransferAbort](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle)  
*Terminates a non-blocking LPI2C master transmission early.*

## IRQ handler

- void [LPI2C\\_MasterTransferHandleIRQ](#) (LPI2C\_Type \*base, lpi2c\_master\_handle\_t \*handle)  
*Reusable routine to handle master interrupts.*

### 23.4.2 Data Structure Documentation

#### 23.4.2.1 struct lpi2c\_master\_config\_t

This structure holds configuration settings for the LPI2C peripheral. To initialize this structure to reasonable defaults, call the [LPI2C\\_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.*
  - bool `enableDoze`  
*Whether master is enabled in doze mode.*
  - bool `debugEnable`  
*Enable transfers to continue when halted in debug mode.*
  - bool `ignoreAck`  
*Whether to ignore ACK/NACK.*
  - `lpi2c_master_pin_config_t pinConfig`  
*The pin configuration option.*
  - `uint32_t baudRate_Hz`  
*Desired baud rate in Hertz.*
  - `uint32_t busIdleTimeout_ns`  
*Bus idle timeout in nanoseconds.*
  - `uint32_t pinLowTimeout_ns`  
*Pin low timeout in nanoseconds.*
  - `uint8_t sdaGlitchFilterWidth_ns`  
*Width in nanoseconds of glitch filter on SDA pin.*
  - `uint8_t sclGlitchFilterWidth_ns`  
*Width in nanoseconds of glitch filter on SCL pin.*
  - struct {
    - bool `enable`  
*Enable host request.*
    - `lpi2c_host_request_source_t source`  
*Host request source.*
    - `lpi2c_host_request_polarity_t polarity`  
*Host request pin polarity.*
} `hostRequest`
- Host request options.*

### 23.4.2.1.0.65 Field Documentation

**23.4.2.1.0.65.1 bool lpi2c\_master\_config\_t::enableMaster**

**23.4.2.1.0.65.2 bool lpi2c\_master\_config\_t::enableDoze**

**23.4.2.1.0.65.3 bool lpi2c\_master\_config\_t::debugEnable**

**23.4.2.1.0.65.4 bool lpi2c\_master\_config\_t::ignoreAck**

**23.4.2.1.0.65.5 lpi2c\_master\_pin\_config\_t lpi2c\_master\_config\_t::pinConfig**

**23.4.2.1.0.65.6 uint32\_t lpi2c\_master\_config\_t::baudRate\_Hz**

**23.4.2.1.0.65.7 uint32\_t lpi2c\_master\_config\_t::busIdleTimeout\_ns**

Set to 0 to disable.

## LPI2C Master Driver

### 23.4.2.1.0.65.8 uint32\_t lpi2c\_master\_config\_t::pinLowTimeout\_ns

Set to 0 to disable.

### 23.4.2.1.0.65.9 uint8\_t lpi2c\_master\_config\_t::sdaGlitchFilterWidth\_ns

Set to 0 to disable.

### 23.4.2.1.0.65.10 uint8\_t lpi2c\_master\_config\_t::sclGlitchFilterWidth\_ns

Set to 0 to disable.

### 23.4.2.1.0.65.11 bool lpi2c\_master\_config\_t::enable

### 23.4.2.1.0.65.12 lpi2c\_host\_request\_source\_t lpi2c\_master\_config\_t::source

### 23.4.2.1.0.65.13 lpi2c\_host\_request\_polarity\_t lpi2c\_master\_config\_t::polarity

### 23.4.2.1.0.65.14 struct { ... } lpi2c\_master\_config\_t::hostRequest

## 23.4.2.2 struct lpi2c\_data\_match\_config\_t

### Data Fields

- [lpi2c\\_data\\_match\\_config\\_mode\\_t](#) matchMode  
*Data match configuration setting.*
- bool rxDataMatchOnly  
*When set to true, received data is ignored until a successful match.*
- uint32\_t match0  
*Match value 0.*
- uint32\_t match1  
*Match value 1.*

### 23.4.2.2.0.66 Field Documentation

#### 23.4.2.2.0.66.1 lpi2c\_data\_match\_config\_mode\_t lpi2c\_data\_match\_config\_t::matchMode

#### 23.4.2.2.0.66.2 bool lpi2c\_data\_match\_config\_t::rxDataMatchOnly

#### 23.4.2.2.0.66.3 uint32\_t lpi2c\_data\_match\_config\_t::match0

#### 23.4.2.2.0.66.4 uint32\_t lpi2c\_data\_match\_config\_t::match1

## 23.4.2.3 struct \_lpi2c\_master\_transfer

This structure is used to pass transaction parameters to the [LPI2C\\_MasterTransferNonBlocking\(\)](#) API.

### Data Fields

- uint32\_t flags

- **uint16\_t slaveAddress**  
*The 7-bit slave address.*
- **lpi2c\_direction\_t direction**  
*Either `kLPI2C_Read` or `kLPI2C_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.*

### 23.4.2.3.0.67 Field Documentation

#### 23.4.2.3.0.67.1 uint32\_t lpi2c\_master\_transfer\_t::flags

See enumeration `_lpi2c_master_transfer_flags` for available options. Set to 0 or `kLPI2C_TransferDefaultFlag` for normal transfers.

#### 23.4.2.3.0.67.2 uint16\_t lpi2c\_master\_transfer\_t::slaveAddress

#### 23.4.2.3.0.67.3 lpi2c\_direction\_t lpi2c\_master\_transfer\_t::direction

#### 23.4.2.3.0.67.4 uint32\_t lpi2c\_master\_transfer\_t::subaddress

Transferred MSB first.

#### 23.4.2.3.0.67.5 size\_t lpi2c\_master\_transfer\_t::subaddressSize

Maximum size is 4 bytes.

#### 23.4.2.3.0.67.6 void\* lpi2c\_master\_transfer\_t::data

#### 23.4.2.3.0.67.7 size\_t lpi2c\_master\_transfer\_t::dataSize

### 23.4.2.4 struct \_lpi2c\_master\_handle

Note

The contents of this structure are private and subject to change.

### Data Fields

- **uint8\_t state**  
*Transfer state machine current state.*
- **uint16\_t remainingBytes**  
*Remaining byte count in current state.*
- **uint8\_t \* buf**

## LPI2C Master Driver

- `uint16_t commandBuffer[7]`  
*Buffer pointer for current state.*
- `lpi2c_master_transfer_t transfer`  
*LPI2C command sequence.*
- `lpi2c_master_transfer_callback_t completionCallback`  
*Copy of the current transfer info.*
- `void *userData`  
*Callback function pointer.*
- `Application data passed to callback.`

### 23.4.2.4.0.68 Field Documentation

**23.4.2.4.0.68.1 `uint8_t lpi2c_master_handle_t::state`**

**23.4.2.4.0.68.2 `uint16_t lpi2c_master_handle_t::remainingBytes`**

**23.4.2.4.0.68.3 `uint8_t* lpi2c_master_handle_t::buf`**

**23.4.2.4.0.68.4 `uint16_t lpi2c_master_handle_t::commandBuffer[7]`**

**23.4.2.4.0.68.5 `lpi2c_master_transfer_t lpi2c_master_handle_t::transfer`**

**23.4.2.4.0.68.6 `lpi2c_master_transfer_callback_t lpi2c_master_handle_t::completionCallback`**

**23.4.2.4.0.68.7 `void* lpi2c_master_handle_t::userData`**

### 23.4.3 Typedef Documentation

**23.4.3.1 `typedef void(* lpi2c_master_transfer_callback_t)(LPI2C_Type *base, lpi2c_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 [LPI2C\\_MasterTransferCreateHandle\(\)](#).

Parameters

|                               |                                                                                 |
|-------------------------------|---------------------------------------------------------------------------------|
| <code>base</code>             | The LPI2C peripheral base address.                                              |
| <code>completionStatus</code> | Either #kStatus_Success or an error code describing how the transfer completed. |
| <code>userData</code>         | Arbitrary pointer-sized value passed from the application.                      |

### 23.4.4 Enumeration Type Documentation

**23.4.4.1 `enum _lpi2c_master_flags`**

The following status register flags can be cleared:

- `kLPI2C_MasterEndOfPacketFlag`
- `kLPI2C_MasterStopDetectFlag`
- `kLPI2C_MasterNackDetectFlag`
- `kLPI2C_MasterArbitrationLostFlag`
- `kLPI2C_MasterFifoErrFlag`
- `kLPI2C_MasterPinLowTimeoutFlag`
- `kLPI2C_MasterDataMatchFlag`

All flags except `kLPI2C_MasterBusyFlag` and `kLPI2C_MasterBusBusyFlag` can be enabled as interrupts.

#### Note

These enums are meant to be OR'd together to form a bit mask.

#### Enumerator

- `kLPI2C_MasterTxReadyFlag` Transmit data flag.  
`kLPI2C_MasterRxReadyFlag` Receive data flag.  
`kLPI2C_MasterEndOfPacketFlag` End Packet flag.  
`kLPI2C_MasterStopDetectFlag` Stop detect flag.  
`kLPI2C_MasterNackDetectFlag` NACK detect flag.  
`kLPI2C_MasterArbitrationLostFlag` Arbitration lost flag.  
`kLPI2C_MasterFifoErrFlag` FIFO error flag.  
`kLPI2C_MasterPinLowTimeoutFlag` Pin low timeout flag.  
`kLPI2C_MasterDataMatchFlag` Data match flag.  
`kLPI2C_MasterBusyFlag` Master busy flag.  
`kLPI2C_MasterBusBusyFlag` Bus busy flag.

#### 23.4.4.2 enum lpi2c\_direction\_t

#### Enumerator

- `kLPI2C_Write` Master transmit.  
`kLPI2C_Read` Master receive.

#### 23.4.4.3 enum lpi2c\_master\_pin\_config\_t

#### Enumerator

- `kLPI2C_2PinOpenDrain` LPI2C Configured for 2-pin open drain mode.  
`kLPI2C_2PinOutputOnly` LPI2C Configured for 2-pin output only mode (ultra-fast mode)  
`kLPI2C_2PinPushPull` LPI2C Configured for 2-pin push-pull mode.  
`kLPI2C_4PinPushPull` LPI2C Configured for 4-pin push-pull mode.  
`kLPI2C_2PinOpenDrainWithSeparateSlave` LPI2C Configured for 2-pin open drain mode with separate LPI2C slave.

## LPI2C Master Driver

***kLPI2C\_2PinOutputOnlyWithSeparateSlave*** LPI2C Configured for 2-pin output only mode(ultra-fast mode) with separate LPI2C slave.

***kLPI2C\_2PinPushPullWithSeparateSlave*** LPI2C Configured for 2-pin push-pull mode with separate LPI2C slave.

***kLPI2C\_4PinPushPullWithInvertedOutput*** LPI2C Configured for 4-pin push-pull mode(inverted outputs)

### 23.4.4.4 enum lpi2c\_host\_request\_source\_t

Enumerator

***kLPI2C\_HostRequestExternalPin*** Select the LPI2C\_HREQ pin as the host request input.

***kLPI2C\_HostRequestInputTrigger*** Select the input trigger as the host request input.

### 23.4.4.5 enum lpi2c\_host\_request\_polarity\_t

Enumerator

***kLPI2C\_HostRequestPinActiveLow*** Configure the LPI2C\_HREQ pin active low.

***kLPI2C\_HostRequestPinActiveHigh*** Configure the LPI2C\_HREQ pin active high.

### 23.4.4.6 enum lpi2c\_data\_match\_config\_mode\_t

Enumerator

***kLPI2C\_MatchDisabled*** LPI2C Match Disabled.

***kLPI2C\_1stWordEqualsM0OrM1*** LPI2C Match Enabled and 1st data word equals MATCH0 OR MATCH1.

***kLPI2C\_AnyWordEqualsM0OrM1*** LPI2C Match Enabled and any data word equals MATCH0 OR MATCH1.

***kLPI2C\_1stWordEqualsM0And2ndWordEqualsM1*** LPI2C Match Enabled and 1st data word equals MATCH0, 2nd data equals MATCH1.

***kLPI2C\_AnyWordEqualsM0AndNextWordEqualsM1*** LPI2C Match Enabled and any data word equals MATCH0, next data equals MATCH1.

***kLPI2C\_1stWordAndM1EqualsM0AndM1*** LPI2C Match Enabled and 1st data word and MATCH0 equals MATCH0 and MATCH1.

***kLPI2C\_AnyWordAndM1EqualsM0AndM1*** LPI2C Match Enabled and any data word and MATCH0 equals MATCH0 and MATCH1.

#### 23.4.4.7 enum \_lpi2c\_master\_transfer\_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the `_lpi2c_master_transfer::flags` field.

Enumerator

- kLPI2C\_TransferDefaultFlag*** Transfer starts with a start signal, stops with a stop signal.
- kLPI2C\_TransferNoStartFlag*** Don't send a start condition, address, and sub address.
- kLPI2C\_TransferRepeatedStartFlag*** Send a repeated start condition.
- kLPI2C\_TransferNoStopFlag*** Don't send a stop condition.

### 23.4.5 Function Documentation

#### 23.4.5.1 void LPI2C\_MasterGetDefaultConfig ( `lpi2c_master_config_t * masterConfig` )

This function provides the following default configuration for the LPI2C master peripheral:

```
* masterConfig->enableMaster = true;
* masterConfig->debugEnable = false;
* masterConfig->ignoreAck = false;
* masterConfig->pinConfig = kLPI2C_2PinOpenDrain;
* masterConfig->baudRate_Hz = 100000U;
* masterConfig->busIdleTimeout_ns = 0;
* masterConfig->pinLowTimeout_ns = 0;
* masterConfig->sdaGlitchFilterWidth_ns = 0;
* masterConfig->sclGlitchFilterWidth_ns = 0;
* masterConfig->hostRequest.enable = false;
* masterConfig->hostRequest.source = kLPI2C_HostRequestExternalPin;
* masterConfig->hostRequest.polarity = kLPI2C_HostRequestPinActiveHigh;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with [LPI2C\\_MasterInit\(\)](#).

Parameters

|     |                           |                                                                                                            |
|-----|---------------------------|------------------------------------------------------------------------------------------------------------|
| out | <code>masterConfig</code> | User provided configuration structure for default values. Refer to <a href="#">lpi2c_master_config_t</a> . |
|-----|---------------------------|------------------------------------------------------------------------------------------------------------|

#### 23.4.5.2 void LPI2C\_MasterInit ( `LPI2C_Type * base, const lpi2c_master_config_t * masterConfig, uint32_t sourceClock_Hz` )

This function enables the peripheral clock and initializes the LPI2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

## LPI2C Master Driver

Parameters

|                       |                                                                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.                                                                                                         |
| <i>masterConfig</i>   | User provided peripheral configuration. Use <a href="#">LPI2C_MasterGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_Hz</i> | Frequency in Hertz of the LPI2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.            |

### 23.4.5.3 void LPI2C\_MasterDeinit ( LPI2C\_Type \* *base* )

This function disables the LPI2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

### 23.4.5.4 void LPI2C\_MasterConfigureDataMatch ( LPI2C\_Type \* *base*, const lpi2c\_data\_match\_config\_t \* *config* )

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.   |
| <i>config</i> | Settings for the data match feature. |

### 23.4.5.5 static void LPI2C\_MasterReset ( LPI2C\_Type \* *base* ) [inline], [static]

Restores the LPI2C master peripheral to reset conditions.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

### 23.4.5.6 static void LPI2C\_MasterEnable ( LPI2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                     |
| <i>enable</i> | Pass true to enable or false to disable the specified LPI2C as master. |

### 23.4.5.7 static uint32\_t LPI2C\_MasterGetStatusFlags ( LPI2C\_Type \* *base* ) [inline], [static]

A bit mask with the state of all LPI2C master 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 LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_lpi2c\\_master\\_flags](#)

### 23.4.5.8 static void LPI2C\_MasterClearStatusFlags ( LPI2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- [kLPI2C\\_MasterEndOfPacketFlag](#)
- [kLPI2C\\_MasterStopDetectFlag](#)
- [kLPI2C\\_MasterNackDetectFlag](#)
- [kLPI2C\\_MasterArbitrationLostFlag](#)
- [kLPI2C\\_MasterFifoErrFlag](#)
- [kLPI2C\\_MasterPinLowTimeoutFlag](#)
- [kLPI2C\\_MasterDataMatchFlag](#)

Attempts to clear other flags has no effect.

## LPI2C Master Driver

Parameters

|                   |                                                                                                                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The LPI2C peripheral base address.                                                                                                                                                                                                    |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_lpi2c_master_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">LPI2C_MasterGetStatusFlags()</a> . |

See Also

[\\_lpi2c\\_master\\_flags](#).

### 23.4.5.9 static void LPI2C\_MasterEnableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be enabled as interrupts.

Parameters

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_lpi2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

### 23.4.5.10 static void LPI2C\_MasterDisableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [kLPI2C\\_MasterBusyFlag](#) and [kLPI2C\\_MasterBusBusyFlag](#) can be disabled as interrupts.

Parameters

|                      |                                                                                                                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                     |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_lpi2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

### 23.4.5.11 static uint32\_t LPI2C\_MasterGetEnabledInterrupts ( LPI2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

A bitmask composed of `_lpi2c_master_flags` enumerators OR'd together to indicate the set of enabled interrupts.

#### 23.4.5.12 static void LPI2C\_MasterEnableDMA ( **LPI2C\_Type** \* *base*, bool *enableTx*, bool *enableRx* ) [inline], [static]

Parameters

|                 |                                                                                |
|-----------------|--------------------------------------------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.                                             |
| <i>enableTx</i> | Enable flag for transmit DMA request. Pass true for enable, false for disable. |
| <i>enableRx</i> | Enable flag for receive DMA request. Pass true for enable, false for disable.  |

#### 23.4.5.13 static uint32\_t LPI2C\_MasterGetTxFifoAddress ( **LPI2C\_Type** \* *base* ) [inline], [static]

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

The LPI2C Master Transmit Data Register address.

#### 23.4.5.14 static uint32\_t LPI2C\_MasterGetRxFifoAddress ( **LPI2C\_Type** \* *base* ) [inline], [static]

Parameters

## LPI2C Master Driver

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

The LPI2C Master Receive Data Register address.

### 23.4.5.15 static void LPI2C\_MasterSetWatermarks ( *LPI2C\_Type* \* *base*, *size\_t* *txWords*, *size\_t* *rxWords* ) [inline], [static]

Parameters

|                |                                                                                                                                                                                                                                                             |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                                                                                          |
| <i>txWords</i> | Transmit FIFO watermark value in words. The <a href="#">kLPI2C_MasterTxReadyFlag</a> flag is set whenever the number of words in the transmit FIFO is equal or less than <i>txWords</i> . Writing a value equal or greater than the FIFO size is truncated. |
| <i>rxWords</i> | Receive FIFO watermark value in words. The <a href="#">kLPI2C_MasterRxReadyFlag</a> flag is set whenever the number of words in the receive FIFO is greater than <i>rxWords</i> . Writing a value equal or greater than the FIFO size is truncated.         |

### 23.4.5.16 static void LPI2C\_MasterGetFifoCounts ( *LPI2C\_Type* \* *base*, *size\_t* \* *rxCount*, *size\_t* \* *txCount* ) [inline], [static]

Parameters

|     |                |                                                                                                                              |
|-----|----------------|------------------------------------------------------------------------------------------------------------------------------|
|     | <i>base</i>    | The LPI2C peripheral base address.                                                                                           |
| out | <i>txCount</i> | Pointer through which the current number of words 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 words in the receive FIFO is returned. Pass NULL if this value is not required.  |

### 23.4.5.17 void LPI2C\_MasterSetBaudRate ( *LPI2C\_Type* \* *base*, *uint32\_t* *sourceClock\_Hz*, *uint32\_t* *baudRate\_Hz* )

The LPI2C 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.

## Note

Please note that the second parameter is the clock frequency of LPI2C module, the third parameter means user configured bus baudrate, this implementation is different from other I2C drivers which use baudrate configuration as second parameter and source clock frequency as third parameter.

## LPI2C Master Driver

Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.         |
| <i>sourceClock_Hz</i> | LPI2C functional clock frequency in Hertz. |
| <i>baudRate_Hz</i>    | Requested bus frequency in Hertz.          |

**23.4.5.18 static bool LPI2C\_MasterGetBusIdleState ( LPI2C\_Type \* *base* ) [inline], [static]**

Requires the master mode to be enabled.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

**23.4.5.19 status\_t LPI2C\_MasterStart ( LPI2C\_Type \* *base*, uint8\_t *address*, lpi2c\_direction\_t *dir* )**

This function is used to initiate a new master mode transfer. First, the bus state is checked to ensure that another master is not occupying the bus. Then a START signal is transmitted, followed by the 7-bit address specified in the *address* parameter. Note that this function does not actually wait until the START and address are successfully sent on the bus before returning.

Parameters

|                |                                                                                                                                                                                     |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The LPI2C peripheral base address.                                                                                                                                                  |
| <i>address</i> | 7-bit slave device address, in bits [6:0].                                                                                                                                          |
| <i>dir</i>     | Master transfer direction, either <a href="#">kLPI2C_Read</a> or <a href="#">kLPI2C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                                 |                                                                           |
|---------------------------------|---------------------------------------------------------------------------|
| <code>#kStatus_Success</code>   | START signal and address were successfully enqueued in the transmit FIFO. |
| <code>kStatus_LPI2C_Busy</code> | Another master is currently utilizing the bus.                            |

### 23.4.5.20 `static status_t LPI2C_MasterRepeatedStart( LPI2C_Type * base, uint8_t address, lpi2c_direction_t dir ) [inline], [static]`

This function is used to send a Repeated START signal when a transfer is already in progress. Like [LPI2C\\_MasterStart\(\)](#), it also sends the specified 7-bit address.

Note

This function exists primarily to maintain compatible APIs between LPI2C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

|                      |                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>base</code>    | The LPI2C peripheral base address.                                                                                                                                                  |
| <code>address</code> | 7-bit slave device address, in bits [6:0].                                                                                                                                          |
| <code>dir</code>     | Master transfer direction, either <a href="#">kLPI2C_Read</a> or <a href="#">kLPI2C_Write</a> . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address. |

Return values

|                                 |                                                                                    |
|---------------------------------|------------------------------------------------------------------------------------|
| <code>#kStatus_Success</code>   | Repeated START signal and address were successfully enqueued in the transmit FIFO. |
| <code>kStatus_LPI2C_Busy</code> | Another master is currently utilizing the bus.                                     |

### 23.4.5.21 `status_t LPI2C_MasterSend( LPI2C_Type * base, void * txBuff, size_t txSize )`

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\\_LPI2C\\_Nak](#).

Parameters

## LPI2C Master Driver

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The LPI2C 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. |

Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>#kStatus_Success</i>              | Data was sent successfully.                        |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or over run.                        |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

### 23.4.5.22 **status\_t LPI2C\_MasterReceive ( LPI2C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize* )**

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The LPI2C 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. |

Return values

|                                      |                                                    |
|--------------------------------------|----------------------------------------------------|
| <i>#kStatus_Success</i>              | Data was received successfully.                    |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                         |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                            |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.  |

### 23.4.5.23 status\_t LPI2C\_MasterStop ( LPI2C\_Type \* *base* )

This function does not return until the STOP signal is seen on the bus, or an error occurs.

## LPI2C Master Driver

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Return values

|                                      |                                                                                  |
|--------------------------------------|----------------------------------------------------------------------------------|
| <i>#kStatus_Success</i>              | The STOP signal was successfully sent on the bus and the transaction terminated. |
| <i>kStatus_LPI2C_Busy</i>            | Another master is currently utilizing the bus.                                   |
| <i>kStatus_LPI2C_Nak</i>             | The slave device sent a NAK in response to a byte.                               |
| <i>kStatus_LPI2C_FifoError</i>       | FIFO under run or overrun.                                                       |
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                                                          |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout.                                |

### 23.4.5.24 **status\_t LPI2C\_MasterTransferBlocking ( LPI2C\_Type \* *base*, Ipi2c\_master\_transfer\_t \* *transfer* )**

Note

The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address. |
| <i>transfer</i> | Pointer to the transfer structure. |

Return values

|                                |                                                    |
|--------------------------------|----------------------------------------------------|
| <i>#kStatus_Success</i>        | Data was received successfully.                    |
| <i>kStatus_LPI2C_Busy</i>      | Another master is currently utilizing the bus.     |
| <i>kStatus_LPI2C_Nak</i>       | The slave device sent a NAK in response to a byte. |
| <i>kStatus_LPI2C_FifoError</i> | FIFO under run or overrun.                         |

|                                      |                                                   |
|--------------------------------------|---------------------------------------------------|
| <i>kStatus_LPI2C_ArbitrationLost</i> | Arbitration lost error.                           |
| <i>kStatus_LPI2C_PinLowTimeout</i>   | SCL or SDA were held low longer than the timeout. |

**23.4.5.25 void LPI2C\_MasterTransferCreateHandle ( *LPI2C\_Type* \* *base*,  
*Ipi2c\_master\_handle\_t* \* *handle*, *Ipi2c\_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 [LPI2C\\_MasterTransferAbort\(\)](#) API shall be called.

#### Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

#### Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The LPI2C peripheral base address.                           |
| out | <i>handle</i>   | Pointer to the LPI2C 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.      |

**23.4.5.26 *status\_t* LPI2C\_MasterTransferNonBlocking ( *LPI2C\_Type* \* *base*,  
*Ipi2c\_master\_handle\_t* \* *handle*, *Ipi2c\_master\_transfer\_t* \* *transfer* )**

#### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.         |
| <i>handle</i>   | Pointer to the LPI2C master driver handle. |
| <i>transfer</i> | The pointer to the transfer descriptor.    |

## LPI2C Master Driver

Return values

|                    |                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------|
| #kStatus_Success   | The transaction was started successfully.                                                                   |
| kStatus_LPI2C_Busy | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |

**23.4.5.27 status\_t LPI2C\_MasterTransferGetCount ( LPI2C\_Type \* *base*, Ipi2c\_master\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The LPI2C peripheral base address.                                  |
|     | <i>handle</i> | Pointer to the LPI2C master driver handle.                          |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

Return values

|                              |                                                                |
|------------------------------|----------------------------------------------------------------|
| #kStatus_Success             |                                                                |
| kStatus_NoTransferInProgress | There is not a non-blocking transaction currently in progress. |

**23.4.5.28 void LPI2C\_MasterTransferAbort ( LPI2C\_Type \* *base*, Ipi2c\_master\_handle\_t \* *handle* )**

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the LPI2C peripheral's IRQ priority.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

Return values

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| #kStatus_Success   | A transaction was successfully aborted.                        |
| kStatus_LPI2C_Idle | There is not a non-blocking transaction currently in progress. |

**23.4.5.29 void LPI2C\_MasterTransferHandleIRQ ( LPI2C\_Type \* *base*,  
Ipi2c\_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 LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

## LPI2C Slave Driver

### 23.5 LPI2C Slave Driver

#### 23.5.1 Overview

## Data Structures

- struct [lpi2c\\_slave\\_config\\_t](#)  
*Structure with settings to initialize the LPI2C slave module.* [More...](#)
- struct [lpi2c\\_slave\\_transfer\\_t](#)  
*LPI2C slave transfer structure.* [More...](#)
- struct [lpi2c\\_slave\\_handle\\_t](#)  
*LPI2C slave handle structure.* [More...](#)

## TypeDefs

- typedef void(\* [lpi2c\\_slave\\_transfer\\_callback\\_t](#))[\(LPI2C\\_Type \\*base, lpi2c\\_slave\\_transfer\\_t \\*transfer, void \\*userData\)](#)  
*Slave event callback function pointer type.*

## Enumerations

- enum [\\_lpi2c\\_slave\\_flags](#) {  
  kLPI2C\_SlaveTxReadyFlag = LPI2C\_SSR\_TDF\_MASK,  
  kLPI2C\_SlaveRxReadyFlag = LPI2C\_SSR\_RDF\_MASK,  
  kLPI2C\_SlaveAddressValidFlag = LPI2C\_SSR\_AVF\_MASK,  
  kLPI2C\_SlaveTransmitAckFlag = LPI2C\_SSR\_TAF\_MASK,  
  kLPI2C\_SlaveRepeatedStartDetectFlag = LPI2C\_SSR\_RSF\_MASK,  
  kLPI2C\_SlaveStopDetectFlag = LPI2C\_SSR\_SDF\_MASK,  
  kLPI2C\_SlaveBitErrFlag = LPI2C\_SSR\_BEF\_MASK,  
  kLPI2C\_SlaveFifoErrFlag = LPI2C\_SSR\_FEF\_MASK,  
  kLPI2C\_SlaveAddressMatch0Flag = LPI2C\_SSR\_AM0F\_MASK,  
  kLPI2C\_SlaveAddressMatch1Flag = LPI2C\_SSR\_AM1F\_MASK,  
  kLPI2C\_SlaveGeneralCallFlag = LPI2C\_SSR\_GCF\_MASK,  
  kLPI2C\_SlaveBusyFlag = LPI2C\_SSR\_SBF\_MASK,  
  kLPI2C\_SlaveBusBusyFlag = LPI2C\_SSR\_BBF\_MASK }  
*LPI2C slave peripheral flags.*
- enum [lpi2c\\_slave\\_address\\_match\\_t](#) {  
  kLPI2C\_MatchAddress0 = 0U,  
  kLPI2C\_MatchAddress0OrAddress1 = 2U,  
  kLPI2C\_MatchAddress0ThroughAddress1 = 6U }  
*LPI2C slave address match options.*
- enum [lpi2c\\_slave\\_transfer\\_event\\_t](#) {

```
kLPI2C_SlaveAddressMatchEvent = 0x01U,
kLPI2C_SlaveTransmitEvent = 0x02U,
kLPI2C_SlaveReceiveEvent = 0x04U,
kLPI2C_SlaveTransmitAckEvent = 0x08U,
kLPI2C_SlaveRepeatedStartEvent = 0x10U,
kLPI2C_SlaveCompletionEvent = 0x20U,
kLPI2C_SlaveAllEvents }
```

*Set of events sent to the callback for non blocking slave transfers.*

## Slave initialization and deinitialization

- void [LPI2C\\_SlaveGetDefaultConfig](#) ([lpi2c\\_slave\\_config\\_t](#) \*slaveConfig)  
*Provides a default configuration for the LPI2C slave peripheral.*
- void [LPI2C\\_SlaveInit](#) ([LPI2C\\_Type](#) \*base, const [lpi2c\\_slave\\_config\\_t](#) \*slaveConfig, [uint32\\_t](#) sourceClock\_Hz)  
*Initializes the LPI2C slave peripheral.*
- void [LPI2C\\_SlaveDeinit](#) ([LPI2C\\_Type](#) \*base)  
*Deinitializes the LPI2C slave peripheral.*
- static void [LPI2C\\_SlaveReset](#) ([LPI2C\\_Type](#) \*base)  
*Performs a software reset of the LPI2C slave peripheral.*
- static void [LPI2C\\_SlaveEnable](#) ([LPI2C\\_Type](#) \*base, bool enable)  
*Enables or disables the LPI2C module as slave.*

## Slave status

- static [uint32\\_t](#) [LPI2C\\_SlaveGetStatusFlags](#) ([LPI2C\\_Type](#) \*base)  
*Gets the LPI2C slave status flags.*
- static void [LPI2C\\_SlaveClearStatusFlags](#) ([LPI2C\\_Type](#) \*base, [uint32\\_t](#) statusMask)  
*Clears the LPI2C status flag state.*

## Slave interrupts

- static void [LPI2C\\_SlaveEnableInterrupts](#) ([LPI2C\\_Type](#) \*base, [uint32\\_t](#) interruptMask)  
*Enables the LPI2C slave interrupt requests.*
- static void [LPI2C\\_SlaveDisableInterrupts](#) ([LPI2C\\_Type](#) \*base, [uint32\\_t](#) interruptMask)  
*Disables the LPI2C slave interrupt requests.*
- static [uint32\\_t](#) [LPI2C\\_SlaveGetEnabledInterrupts](#) ([LPI2C\\_Type](#) \*base)  
*Returns the set of currently enabled LPI2C slave interrupt requests.*

## Slave DMA control

- static void [LPI2C\\_SlaveEnableDMA](#) ([LPI2C\\_Type](#) \*base, bool enableAddressValid, bool enableRx, bool enableTx)  
*Enables or disables the LPI2C slave peripheral DMA requests.*

## LPI2C Slave Driver

### Slave bus operations

- static bool [LPI2C\\_SlaveGetBusIdleState](#) (LPI2C\_Type \*base)  
*Returns whether the bus is idle.*
- static void [LPI2C\\_SlaveTransmitAck](#) (LPI2C\_Type \*base, bool ackOrNack)  
*Transmits either an ACK or NAK on the I2C bus in response to a byte from the master.*
- static uint32\_t [LPI2C\\_SlaveGetReceivedAddress](#) (LPI2C\_Type \*base)  
*Returns the slave address sent by the I2C master.*
- status\_t [LPI2C\\_SlaveSend](#) (LPI2C\_Type \*base, void \*txBuff, size\_t txSize, size\_t \*actualTxSize)  
*Performs a polling send transfer on the I2C bus.*
- status\_t [LPI2C\\_SlaveReceive](#) (LPI2C\_Type \*base, void \*rxBuff, size\_t rxSize, size\_t \*actualRxSize)  
*Performs a polling receive transfer on the I2C bus.*

### Slave non-blocking

- void [LPI2C\\_SlaveTransferCreateHandle](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, [lpi2c\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the LPI2C slave non-blocking APIs.*
- status\_t [LPI2C\\_SlaveTransferNonBlocking](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- status\_t [LPI2C\\_SlaveTransferGetCount](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the slave transfer status during a non-blocking transfer.*
- void [LPI2C\\_SlaveTransferAbort](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle)  
*Aborts the slave non-blocking transfers.*

### Slave IRQ handler

- void [LPI2C\\_SlaveTransferHandleIRQ](#) (LPI2C\_Type \*base, lpi2c\_slave\_handle\_t \*handle)  
*Reusable routine to handle slave interrupts.*

### 23.5.2 Data Structure Documentation

#### 23.5.2.1 struct lpi2c\_slave\_config\_t

This structure holds configuration settings for the LPI2C slave peripheral. To initialize this structure to reasonable defaults, call the [LPI2C\\_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

- bool `enableSlave`  
*Enable slave mode.*
- uint8\_t `address0`  
*Slave's 7-bit address.*
- uint8\_t `address1`  
*Alternate slave 7-bit address.*
- `lpi2c_slave_address_match_t addressMatchMode`  
*Address matching options.*
- bool `filterDozeEnable`  
*Enable digital glitch filter in doze mode.*
- bool `filterEnable`  
*Enable digital glitch filter.*
- bool `enableGeneralCall`  
*Enable general call address matching.*
- bool `ignoreAck`  
*Continue transfers after a NACK is detected.*
- bool `enableReceivedAddressRead`  
*Enable reading the address received address as the first byte of data.*
- uint32\_t `sdaGlitchFilterWidth_ns`  
*Width in nanoseconds of the digital filter on the SDA signal.*
- uint32\_t `sclGlitchFilterWidth_ns`  
*Width in nanoseconds of the digital filter on the SCL signal.*
- uint32\_t `dataValidDelay_ns`  
*Width in nanoseconds of the data valid delay.*
- uint32\_t `clockHoldTime_ns`  
*Width in nanoseconds of the clock hold time.*
- bool `enableAck`  
*Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted.*
- bool `enableTx`  
*Enables SCL clock stretching when the transmit data flag is set during a slave-transmit transfer.*
- bool `enableRx`  
*Enables SCL clock stretching when receive data flag is set during a slave-receive transfer.*
- bool `enableAddress`  
*Enables SCL clock stretching when the address valid flag is asserted.*

## LPI2C Slave Driver

### 23.5.2.1.0.69 Field Documentation

23.5.2.1.0.69.1 `bool lpi2c_slave_config_t::enableSlave`

23.5.2.1.0.69.2 `uint8_t lpi2c_slave_config_t::address0`

23.5.2.1.0.69.3 `uint8_t lpi2c_slave_config_t::address1`

23.5.2.1.0.69.4 `lpi2c_slave_address_match_t lpi2c_slave_config_t::addressMatchMode`

23.5.2.1.0.69.5 `bool lpi2c_slave_config_t::filterDozeEnable`

23.5.2.1.0.69.6 `bool lpi2c_slave_config_t::filterEnable`

23.5.2.1.0.69.7 `bool lpi2c_slave_config_t::enableGeneralCall`

23.5.2.1.0.69.8 `bool lpi2c_slave_config_t::enableAck`

Clock stretching occurs when transmitting the 9th bit. When enableAckSCLStall is enabled, there is no need to set either enableRxDataSCLStall or enableAddressSCLStall.

23.5.2.1.0.69.9 `bool lpi2c_slave_config_t::enableTx`

23.5.2.1.0.69.10 `bool lpi2c_slave_config_t::enableRx`

23.5.2.1.0.69.11 `bool lpi2c_slave_config_t::enableAddress`

23.5.2.1.0.69.12 `bool lpi2c_slave_config_t::ignoreAck`

23.5.2.1.0.69.13 `bool lpi2c_slave_config_t::enableReceivedAddressRead`

23.5.2.1.0.69.14 `uint32_t lpi2c_slave_config_t::sdaGlitchFilterWidth_ns`

23.5.2.1.0.69.15 `uint32_t lpi2c_slave_config_t::sclGlitchFilterWidth_ns`

23.5.2.1.0.69.16 `uint32_t lpi2c_slave_config_t::dataValidDelay_ns`

23.5.2.1.0.69.17 `uint32_t lpi2c_slave_config_t::clockHoldTime_ns`

### 23.5.2.2 `struct lpi2c_slave_transfer_t`

#### Data Fields

- `lpi2c_slave_transfer_event_t event`  
*Reason the callback is being invoked.*
- `uint8_t receivedAddress`  
*Matching address send by master.*
- `uint8_t * data`  
*Transfer buffer.*
- `size_t dataSize`  
*Transfer size.*

- **status\_t completionStatus**  
*Success or error code describing how the transfer completed.*
- **size\_t transferredCount**  
*Number of bytes actually transferred since start or last repeated start.*

### 23.5.2.2.0.70 Field Documentation

**23.5.2.2.0.70.1 lpi2c\_slave\_transfer\_event\_t lpi2c\_slave\_transfer\_t::event**

**23.5.2.2.0.70.2 uint8\_t lpi2c\_slave\_transfer\_t::receivedAddress**

**23.5.2.2.0.70.3 status\_t lpi2c\_slave\_transfer\_t::completionStatus**

Only applies for [kLPI2C\\_SlaveCompletionEvent](#).

**23.5.2.2.0.70.4 size\_t lpi2c\_slave\_transfer\_t::transferredCount**

### 23.5.2.3 struct \_lpi2c\_slave\_handle

Note

The contents of this structure are private and subject to change.

### Data Fields

- **lpi2c\_slave\_transfer\_t transfer**  
*LPI2C slave transfer copy.*
- **bool isBusy**  
*Whether transfer is busy.*
- **bool wasTransmit**  
*Whether the last transfer was a transmit.*
- **uint32\_t eventMask**  
*Mask of enabled events.*
- **uint32\_t transferredCount**  
*Count of bytes transferred.*
- **lpi2c\_slave\_transfer\_callback\_t callback**  
*Callback function called at transfer event.*
- **void \* userData**  
*Callback parameter passed to callback.*

## LPI2C Slave Driver

### 23.5.2.3.0.71 Field Documentation

23.5.2.3.0.71.1 `lpi2c_slave_transfer_t lpi2c_slave_handle_t::transfer`

23.5.2.3.0.71.2 `bool lpi2c_slave_handle_t::isBusy`

23.5.2.3.0.71.3 `bool lpi2c_slave_handle_t::wasTransmit`

23.5.2.3.0.71.4 `uint32_t lpi2c_slave_handle_t::eventMask`

23.5.2.3.0.71.5 `uint32_t lpi2c_slave_handle_t::transferredCount`

23.5.2.3.0.71.6 `lpi2c_slave_transfer_callback_t lpi2c_slave_handle_t::callback`

23.5.2.3.0.71.7 `void* lpi2c_slave_handle_t::userData`

### 23.5.3 Typedef Documentation

23.5.3.1 `typedef void(* lpi2c_slave_transfer_callback_t)(LPI2C_Type *base,  
lpi2c_slave_transfer_t *transfer, void *userData)`

This callback is used only for the slave non-blocking transfer API. To install a callback, use the LPI2C\_SlaveSetCallback() function after you have created a handle.

## Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address for the LPI2C 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.                           |

**23.5.4 Enumeration Type Documentation****23.5.4.1 enum \_lpi2c\_slave\_flags**

The following status register flags can be cleared:

- [kLPI2C\\_SlaveRepeatedStartDetectFlag](#)
- [kLPI2C\\_SlaveStopDetectFlag](#)
- [kLPI2C\\_SlaveBitErrFlag](#)
- [kLPI2C\\_SlaveFifoErrFlag](#)

All flags except [kLPI2C\\_SlaveBusyFlag](#) and [kLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

## Note

These enumerations are meant to be OR'd together to form a bit mask.

## Enumerator

- kLPI2C\_SlaveTxReadyFlag* Transmit data flag.  
*kLPI2C\_SlaveRxReadyFlag* Receive data flag.  
*kLPI2C\_SlaveAddressValidFlag* Address valid flag.  
*kLPI2C\_SlaveTransmitAckFlag* Transmit ACK flag.  
*kLPI2C\_SlaveRepeatedStartDetectFlag* Repeated start detect flag.  
*kLPI2C\_SlaveStopDetectFlag* Stop detect flag.  
*kLPI2C\_SlaveBitErrFlag* Bit error flag.  
*kLPI2C\_SlaveFifoErrFlag* FIFO error flag.  
*kLPI2C\_SlaveAddressMatch0Flag* Address match 0 flag.  
*kLPI2C\_SlaveAddressMatch1Flag* Address match 1 flag.  
*kLPI2C\_SlaveGeneralCallFlag* General call flag.  
*kLPI2C\_SlaveBusyFlag* Master busy flag.  
*kLPI2C\_SlaveBusBusyFlag* Bus busy flag.

**23.5.4.2 enum lpi2c\_slave\_address\_match\_t**

## Enumerator

- kLPI2C\_MatchAddress0* Match only address 0.

## LPI2C Slave Driver

***kLPI2C\_MatchAddress0OrAddress1*** Match either address 0 or address 1.

***kLPI2C\_MatchAddress0ThroughAddress1*** Match a range of slave addresses from address 0 through address 1.

### 23.5.4.3 enum lpi2c\_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 [LPI2C\\_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

***kLPI2C\_SlaveAddressMatchEvent*** Received the slave address after a start or repeated start.

***kLPI2C\_SlaveTransmitEvent*** Callback is requested to provide data to transmit (slave-transmitter role).

***kLPI2C\_SlaveReceiveEvent*** Callback is requested to provide a buffer in which to place received data (slave-receiver role).

***kLPI2C\_SlaveTransmitAckEvent*** Callback needs to either transmit an ACK or NACK.

***kLPI2C\_SlaveRepeatedStartEvent*** A repeated start was detected.

***kLPI2C\_SlaveCompletionEvent*** A stop was detected, completing the transfer.

***kLPI2C\_SlaveAllEvents*** Bit mask of all available events.

## 23.5.5 Function Documentation

### 23.5.5.1 void LPI2C\_SlaveGetDefaultConfig ( *Ipi2c\_slave\_config\_t \* slaveConfig* )

This function provides the following default configuration for the LPI2C slave peripheral:

```
* slaveConfig->enableSlave = true;
* slaveConfig->address0 = 0U;
* slaveConfig->address1 = 0U;
* slaveConfig->addressMatchMode = kLPI2C_MatchAddress0;
* slaveConfig->filterDozeEnable = true;
* slaveConfig->filterEnable = true;
* slaveConfig->enableGeneralCall = false;
* slaveConfig->sclStall.enableAck = false;
* slaveConfig->sclStall.enableTx = true;
* slaveConfig->sclStall.enableRx = true;
* slaveConfig->sclStall.enableAddress = true;
* slaveConfig->ignoreAck = false;
* slaveConfig->enableReceivedAddressRead = false;
* slaveConfig->sdaGlitchFilterWidth_ns = 0;
* slaveConfig->sclGlitchFilterWidth_ns = 0;
* slaveConfig->dataValidDelay_ns = 0;
* slaveConfig->clockHoldTime_ns = 0;
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with [LPI2C\\_SlaveInit\(\)](#). Be sure to override at least the *address0* member of the configuration structure with the desired slave address.

Parameters

|            |                    |                                                                                                                      |
|------------|--------------------|----------------------------------------------------------------------------------------------------------------------|
| <i>out</i> | <i>slaveConfig</i> | User provided configuration structure that is set to default values. Refer to <a href="#">lpi2c_slave_config_t</a> . |
|------------|--------------------|----------------------------------------------------------------------------------------------------------------------|

### **23.5.5.2 void LPI2C\_SlaveInit ( LPI2C\_Type \* *base*, const lpi2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *sourceClock\_Hz* )**

This function enables the peripheral clock and initializes the LPI2C slave peripheral as described by the user provided configuration.

Parameters

|                       |                                                                                                                                           |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | The LPI2C peripheral base address.                                                                                                        |
| <i>slaveConfig</i>    | User provided peripheral configuration. Use <a href="#">LPI2C_SlaveGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>sourceClock_Hz</i> | Frequency in Hertz of the LPI2C functional clock. Used to calculate the filter widths, data valid delay, and clock hold time.             |

### **23.5.5.3 void LPI2C\_SlaveDeinit ( LPI2C\_Type \* *base* )**

This function disables the LPI2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

### **23.5.5.4 static void LPI2C\_SlaveReset ( LPI2C\_Type \* *base* ) [inline], [static]**

Parameters

## LPI2C Slave Driver

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

**23.5.5.5 static void LPI2C\_SlaveEnable ( LPI2C\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                    |
| <i>enable</i> | Pass true to enable or false to disable the specified LPI2C as slave. |

**23.5.5.6 static uint32\_t LPI2C\_SlaveGetStatusFlags ( LPI2C\_Type \* *base* ) [inline], [static]**

A bit mask with the state of all LPI2C slave 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 LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

See Also

[\\_lpi2c\\_slave\\_flags](#)

**23.5.5.7 static void LPI2C\_SlaveClearStatusFlags ( LPI2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]**

The following status register flags can be cleared:

- [kLPI2C\\_SlaveRepeatedStartDetectFlag](#)
- [kLPI2C\\_SlaveStopDetectFlag](#)
- [kLPI2C\\_SlaveBitErrFlag](#)
- [kLPI2C\\_SlaveFifoErrFlag](#)

Attempts to clear other flags has no effect.

Parameters

|                   |                                                                                                                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The LPI2C peripheral base address.                                                                                                                                                                                                  |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_lpi2c_slave_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">LPI2C_SlaveGetStatusFlags()</a> . |

See Also

[\\_lpi2c\\_slave\\_flags](#).

#### 23.5.5.8 static void LPI2C\_SlaveEnableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [KLPI2C\\_SlaveBusyFlag](#) and [KLPI2C\\_SlaveBusBusyFlag](#) can be enabled as interrupts.

Parameters

|                      |                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                   |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_lpi2c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 23.5.5.9 static void LPI2C\_SlaveDisableInterrupts ( LPI2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

All flags except [KLPI2C\\_SlaveBusyFlag](#) and [KLPI2C\\_SlaveBusBusyFlag](#) can be disabled as interrupts.

Parameters

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The LPI2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_lpi2c_slave_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 23.5.5.10 static uint32\_t LPI2C\_SlaveGetEnabledInterrupts ( LPI2C\_Type \* *base* ) [inline], [static]

## LPI2C Slave Driver

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

A bitmask composed of [\\_lpi2c\\_slave\\_flags](#) enumerators OR'd together to indicate the set of enabled interrupts.

### 23.5.5.11 static void LPI2C\_SlaveEnableDMA ( LPI2C\_Type \* *base*, bool *enableAddressValid*, bool *enableRx*, bool *enableTx* ) [inline], [static]

Parameters

|                           |                                                                                                                                                                    |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | The LPI2C peripheral base address.                                                                                                                                 |
| <i>enableAddressValid</i> | Enable flag for the address valid DMA request. Pass true for enable, false for disable. The address valid DMA request is shared with the receive data DMA request. |
| <i>enableRx</i>           | Enable flag for the receive data DMA request. Pass true for enable, false for disable.                                                                             |
| <i>enableTx</i>           | Enable flag for the transmit data DMA request. Pass true for enable, false for disable.                                                                            |

### 23.5.5.12 static bool LPI2C\_SlaveGetBusIdleState ( LPI2C\_Type \* *base* ) [inline], [static]

Requires the slave mode to be enabled.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

### 23.5.5.13 static void LPI2C\_SlaveTransmitAck ( LPI2C\_Type \* *base*, bool *ackOrNack* ) [inline], [static]

Use this function to send an ACK or NAK when the [KLPI2C\\_SlaveTransmitAckFlag](#) is asserted. This only happens if you enable the sclStall.enableAck field of the [lpi2c\\_slave\\_config\\_t](#) configuration structure used to initialize the slave peripheral.

Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | The LPI2C peripheral base address.       |
| <i>ackOrNack</i> | Pass true for an ACK or false for a NAK. |

### 23.5.5.14 static uint32\_t LPI2C\_SlaveGetReceivedAddress ( LPI2C\_Type \* *base* ) [inline], [static]

This function should only be called if the [kLPI2C\\_SlaveAddressValidFlag](#) is asserted.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | The LPI2C peripheral base address. |
|-------------|------------------------------------|

Returns

The 8-bit address matched by the LPI2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

### 23.5.5.15 status\_t LPI2C\_SlaveSend ( LPI2C\_Type \* *base*, void \* *txBuff*, size\_t *txSize*, size\_t \* *actualTxSize* )

Parameters

|     |                     |                                                    |
|-----|---------------------|----------------------------------------------------|
|     | <i>base</i>         | The LPI2C 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. |
| out | <i>actualTxSize</i> |                                                    |

Returns

Error or success status returned by API.

### 23.5.5.16 status\_t LPI2C\_SlaveReceive ( LPI2C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize*, size\_t \* *actualRxSize* )

## LPI2C Slave Driver

Parameters

|     |                     |                                                    |
|-----|---------------------|----------------------------------------------------|
|     | <i>base</i>         | The LPI2C 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. |
| out | <i>actualRxSize</i> |                                                    |

Returns

Error or success status returned by API.

**23.5.5.17 void LPI2C\_SlaveTransferCreateHandle ( LPI2C\_Type \* *base*, Ipi2c\_slave\_handle\_t \* *handle*, Ipi2c\_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 [LPI2C\\_SlaveTransferAbort\(\)](#) API shall be called.

Note

The function also enables the NVIC IRQ for the input LPI2C. Need to notice that on some SoCs the LPI2C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The LPI2C peripheral base address.                           |
| out | <i>handle</i>   | Pointer to the LPI2C 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.      |

**23.5.5.18 status\_t LPI2C\_SlaveTransferNonBlocking ( LPI2C\_Type \* *base*, Ipi2c\_slave\_handle\_t \* *handle*, uint32\_t *eventMask* )**

Call this API after calling [I2C\\_SlaveInit\(\)](#) and [LPI2C\\_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 [LPI2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [Ipi2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The

`KLPI2C_SlaveTransmitEvent` and `KLPI2C_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 `KLPI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

Parameters

|                  |                                                                                                                                                                                                                                                                                                  |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The LPI2C peripheral base address.                                                                                                                                                                                                                                                               |
| <i>handle</i>    | Pointer to <code>#lpi2c_slave_handle_t</code> structure which stores the transfer state.                                                                                                                                                                                                         |
| <i>eventMask</i> | Bit mask formed by OR'ing together <code>lpi2c_slave_transfer_event_t</code> 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 <code>KLPI2C_SlaveAllEvents</code> to enable all events. |

Return values

|                                 |                                                           |
|---------------------------------|-----------------------------------------------------------|
| <code>#kStatus_Success</code>   | Slave transfers were successfully started.                |
| <code>kStatus_LPI2C_Busy</code> | Slave transfers have already been started on this handle. |

### 23.5.5.19 `status_t LPI2C_SlaveTransferGetCount ( LPI2C_Type * base, Ipi2c_slave_handle_t * handle, size_t * count )`

Parameters

|            |               |                                                                                                       |
|------------|---------------|-------------------------------------------------------------------------------------------------------|
|            | <i>base</i>   | The LPI2C peripheral base address.                                                                    |
|            | <i>handle</i> | Pointer to <code>i2c_slave_handle_t</code> structure.                                                 |
| <i>out</i> | <i>count</i>  | Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required. |

Return values

|                                             |  |
|---------------------------------------------|--|
| <code>#kStatus_Success</code>               |  |
| <code>#kStatus_NoTransferIn-Progress</code> |  |

### 23.5.5.20 `void LPI2C_SlaveTransferAbort ( LPI2C_Type * base, Ipi2c_slave_handle_t * handle )`

## LPI2C Slave Driver

### Note

This API could be called at any time to stop slave for handling the bus events.

### Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.                                          |
| <i>handle</i> | Pointer to #lpi2c_slave_handle_t structure which stores the transfer state. |

### Return values

|                           |  |
|---------------------------|--|
| #kStatus_Success          |  |
| <i>kStatus_LPI2C_Idle</i> |  |

### 23.5.5.21 void LPI2C\_SlaveTransferHandleIRQ ( LPI2C\_Type \* *base*, lpi2c\_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 LPI2C peripheral base address.                                          |
| <i>handle</i> | Pointer to #lpi2c_slave_handle_t structure which stores the transfer state. |

## 23.6 LPI2C Master DMA Driver

### 23.6.1 Overview

#### Data Structures

- struct [lpi2c\\_master\\_edma\\_handle\\_t](#)  
*Driver handle for master DMA APIs.* [More...](#)

#### Typedefs

- [typedef void\(\\* lpi2c\\_master\\_transfer\\_callback\\_t\)\(LPI2C\\_Type \\*base, lpi2c\\_master\\_edma\\_handle\\_t \\*handle, status\\_t completionStatus, void \\*userData\)](#)  
*Master DMA completion callback function pointer type.*

#### Master DMA

- [void LPI2C\\_MasterCreateEDMAHandle \(LPI2C\\_Type \\*base, lpi2c\\_master\\_edma\\_handle\\_t \\*handle, edma\\_handle\\_t \\*rxDmaHandle, edma\\_handle\\_t \\*txDmaHandle, lpi2c\\_master\\_edma\\_transfer\\_callback\\_t callback, void \\*userData\)](#)  
*Create a new handle for the LPI2C master DMA APIs.*
- [status\\_t LPI2C\\_MasterTransferEDMA \(LPI2C\\_Type \\*base, lpi2c\\_master\\_edma\\_handle\\_t \\*handle, lpi2c\\_master\\_transfer\\_t \\*transfer\)](#)  
*Performs a non-blocking DMA-based transaction on the I2C bus.*
- [status\\_t LPI2C\\_MasterTransferGetCountEDMA \(LPI2C\\_Type \\*base, lpi2c\\_master\\_edma\\_handle\\_t \\*handle, size\\_t \\*count\)](#)  
*Returns number of bytes transferred so far.*
- [status\\_t LPI2C\\_MasterTransferAbortEDMA \(LPI2C\\_Type \\*base, lpi2c\\_master\\_edma\\_handle\\_t \\*handle\)](#)  
*Terminates a non-blocking LPI2C master transmission early.*

### 23.6.2 Data Structure Documentation

#### 23.6.2.1 struct \_lpi2c\_master\_edma\_handle

Note

The contents of this structure are private and subject to change.

#### Data Fields

- [LPI2C\\_Type \\* base](#)  
*LPI2C base pointer.*
- [bool isBusy](#)

## LPI2C Master DMA Driver

- *Transfer state machine current state.*  
• `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `uint16_t commandBuffer[7]`  
*LPI2C command sequence.*
- `lpi2c_master_transfer_t transfer`  
*Copy of the current transfer info.*
- `lpi2c_master_edma_transfer_callback_t completionCallback`  
*Callback function pointer.*
- `void * userData`  
*Application data passed to callback.*
- `edma_handle_t * rx`  
*Handle for receive DMA channel.*
- `edma_handle_t * tx`  
*Handle for transmit DMA channel.*
- `edma_tcd_t tcds[2]`  
*Software TCD.*

### 23.6.2.1.0.72 Field Documentation

**23.6.2.1.0.72.1 LPI2C\_Type\* lpi2c\_master\_edma\_handle\_t::base**

**23.6.2.1.0.72.2 bool lpi2c\_master\_edma\_handle\_t::isBusy**

**23.6.2.1.0.72.3 uint8\_t lpi2c\_master\_edma\_handle\_t::nbytes**

**23.6.2.1.0.72.4 uint16\_t lpi2c\_master\_edma\_handle\_t::commandBuffer[7]**

**23.6.2.1.0.72.5 lpi2c\_master\_transfer\_t lpi2c\_master\_edma\_handle\_t::transfer**

**23.6.2.1.0.72.6 lpi2c\_master\_edma\_transfer\_callback\_t lpi2c\_master\_edma\_handle\_t-  
::completionCallback**

**23.6.2.1.0.72.7 void\* lpi2c\_master\_edma\_handle\_t::userData**

**23.6.2.1.0.72.8 edma\_handle\_t\* lpi2c\_master\_edma\_handle\_t::rx**

**23.6.2.1.0.72.9 edma\_handle\_t\* lpi2c\_master\_edma\_handle\_t::tx**

**23.6.2.1.0.72.10 edma\_tcd\_t lpi2c\_master\_edma\_handle\_t::tcds[2]**

Two are allocated to provide enough room to align to 32-bytes.

### 23.6.3 Typedef Documentation

23.6.3.1 **typedef void(\* Ipi2c\_master\_edma\_transfer\_callback\_t)(LPI2C\_Type \*base,  
Ipi2c\_master\_edma\_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 [LPI2C\\_MasterCreateEDMAHandle\(\)](#).

## LPI2C Master DMA Driver

Parameters

|                          |                                                                                 |
|--------------------------|---------------------------------------------------------------------------------|
| <i>base</i>              | The LPI2C peripheral base address.                                              |
| <i>handle</i>            | Handle associated with the completed transfer.                                  |
| <i>completion-Status</i> | Either #kStatus_Success or an error code describing how the transfer completed. |
| <i>userData</i>          | Arbitrary pointer-sized value passed from the application.                      |

### 23.6.4 Function Documentation

**23.6.4.1 void LPI2C\_MasterCreateEDMAHandle ( LPI2C\_Type \* *base*, Ipi2c\_master\_edma\_handle\_t \* *handle*, edma\_handle\_t \* *rxDmaHandle*, edma\_handle\_t \* *txDmaHandle*, Ipi2c\_master\_edma\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the DMA APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [LPI2C\\_MasterTransferAbortEDMA\(\)](#) API shall be called.

For devices where the LPI2C send and receive DMA requests are OR'd together, the *txDmaHandle* parameter is ignored and may be set to NULL.

Parameters

|     |                    |                                                                                           |
|-----|--------------------|-------------------------------------------------------------------------------------------|
|     | <i>base</i>        | The LPI2C peripheral base address.                                                        |
| out | <i>handle</i>      | Pointer to the LPI2C master driver handle.                                                |
|     | <i>rxDmaHandle</i> | Handle for the eDMA receive channel. Created by the user prior to calling this function.  |
|     | <i>txDmaHandle</i> | Handle for the eDMA transmit channel. Created by the user prior to calling this function. |
|     | <i>callback</i>    | User provided pointer to the asynchronous callback function.                              |
|     | <i>userData</i>    | User provided pointer to the application callback data.                                   |

**23.6.4.2 status\_t LPI2C\_MasterTransferEDMA ( LPI2C\_Type \* *base*, Ipi2c\_master\_edma\_handle\_t \* *handle*, Ipi2c\_master\_transfer\_t \* *transfer* )**

The callback specified when the *handle* was created is invoked when the transaction has completed.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | The LPI2C peripheral base address.         |
| <i>handle</i>   | Pointer to the LPI2C master driver handle. |
| <i>transfer</i> | The pointer to the transfer descriptor.    |

Return values

|                                 |                                                                                                          |
|---------------------------------|----------------------------------------------------------------------------------------------------------|
| <code>#kStatus_Success</code>   | The transaction was started successfully.                                                                |
| <code>kStatus_LPI2C_Busy</code> | Either another master is currently utilizing the bus, or another DMA transaction is already in progress. |

#### 23.6.4.3 `status_t LPI2C_MasterTransferGetCountEDMA ( LPI2C_Type * base, lpi2c_master_edma_handle_t * handle, size_t * count )`

Parameters

|            |               |                                                                     |
|------------|---------------|---------------------------------------------------------------------|
|            | <i>base</i>   | The LPI2C peripheral base address.                                  |
|            | <i>handle</i> | Pointer to the LPI2C master driver handle.                          |
| <i>out</i> | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

Return values

|                                            |                                                       |
|--------------------------------------------|-------------------------------------------------------|
| <code>#kStatus_Success</code>              |                                                       |
| <code>#kStatus_NoTransferInProgress</code> | There is not a DMA transaction currently in progress. |

#### 23.6.4.4 `status_t LPI2C_MasterTransferAbortEDMA ( LPI2C_Type * base, lpi2c_master_edma_handle_t * handle )`

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the eDMA peripheral's IRQ priority.

## LPI2C Master DMA Driver

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | The LPI2C peripheral base address.         |
| <i>handle</i> | Pointer to the LPI2C master driver handle. |

Return values

|                           |                                                       |
|---------------------------|-------------------------------------------------------|
| # <i>kStatus_Success</i>  | A transaction was successfully aborted.               |
| <i>kStatus_LPI2C_Idle</i> | There is not a DMA transaction currently in progress. |

## 23.7 LPI2C FreeRTOS Driver

### 23.7.1 Overview

#### Driver version

- #define **FSL\_LPI2C\_FREERTOS\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 5))  
*LPI2C freeRTOS driver version 2.1.5.*

#### LPI2C RTOS Operation

- status\_t **LPI2C\_RTOS\_Init** (lpi2c\_rtos\_handle\_t \*handle, LPI2C\_Type \*base, const lpi2c\_master\_config\_t \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes LPI2C.*
- status\_t **LPI2C\_RTOS\_Deinit** (lpi2c\_rtos\_handle\_t \*handle)  
*Deinitializes the LPI2C.*
- status\_t **LPI2C\_RTOS\_Transfer** (lpi2c\_rtos\_handle\_t \*handle, lpi2c\_master\_transfer\_t \*transfer)  
*Performs I2C transfer.*

### 23.7.2 Macro Definition Documentation

#### 23.7.2.1 #define FSL\_LPI2C\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 5))

### 23.7.3 Function Documentation

#### 23.7.3.1 status\_t LPI2C\_RTOS\_Init ( lpi2c\_rtos\_handle\_t \* handle, LPI2C\_Type \* base, const lpi2c\_master\_config\_t \* masterConfig, uint32\_t srcClock\_Hz )

This function initializes the LPI2C module and related RTOS context.

Parameters

|                     |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <i>handle</i>       | The RTOS LPI2C handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the LPI2C instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up LPI2C in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the LPI2C module.                              |

Returns

status of the operation.

### 23.7.3.2 status\_t LPI2C\_RTOS\_Deinit ( *Ipi2c\_rtos\_handle\_t \* handle* )

This function deinitializes the LPI2C module and related RTOS context.

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPI2C handle. |
|---------------|------------------------|

### 23.7.3.3 **status\_t LPI2C\_RTOS\_Transfer ( *Ipi2c\_rtos\_handle\_t \* handle,* *Ipi2c\_master\_transfer\_t \* transfer* )**

This function performs an I2C transfer using LPI2C module according to data given in the transfer structure.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS LPI2C handle.                        |
| <i>transfer</i> | Structure specifying the transfer parameters. |

Returns

status of the operation.



# Chapter 24

## LPIT: Low-Power Interrupt Timer

### 24.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Interrupt Timer (LPIT) of MCUXpresso SDK devices.

### 24.2 Function groups

The LPIT driver supports operating the module as a time counter.

#### 24.2.1 Initialization and deinitialization

The function [LPIT\\_Init\(\)](#) initializes the LPIT with specified configurations. The function [LPIT\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the LPIT operation in doze mode and debug mode.

The function [LPIT\\_SetupChannel\(\)](#) configures the operation of each LPIT channel.

The function [LPIT\\_Deinit\(\)](#) disables the LPIT module and disables the module clock.

#### 24.2.2 Timer period Operations

The function [LPITR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers begin counting down from the value set by this function until it reaches 0.

The function [LPIT\\_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. User can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds

#### 24.2.3 Start and Stop timer operations

The function [LPIT\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value set earlier via the [LPIT\\_SetPeriod\(\)](#) function and starts counting down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function [LPIT\\_StopTimer\(\)](#) stops the timer counting.

## Typical use case

### 24.2.4 Status

Provides functions to get and clear the LPIT status.

### 24.2.5 Interrupt

Provides functions to enable/disable LPIT interrupts and get current enabled interrupts.

## 24.3 Typical use case

### 24.3.1 LPIT tick example

Updates the LPIT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/lpit

## Data Structures

- struct [lpit\\_chnl\\_params\\_t](#)  
*Structure to configure the channel timer. [More...](#)*
- struct [lpit\\_config\\_t](#)  
*LPIT configuration structure. [More...](#)*

## Functions

- static void [LPIT\\_Reset](#) (LPIT\_Type \*base)  
*Performs a software reset on the LPIT module.*

## Driver version

- enum [lpit\\_chnl\\_t](#) {  
  kLPIT\_Chnl\_0 = 0U,  
  kLPIT\_Chnl\_1,  
  kLPIT\_Chnl\_2,  
  kLPIT\_Chnl\_3 }  
*List of LPIT channels.*
- enum [lpit\\_timer\\_modes\\_t](#) {  
  kLPIT\_PeriodicCounter = 0U,  
  kLPIT\_DualPeriodicCounter,  
  kLPIT\_TriggerAccumulator,  
  kLPIT\_InputCapture }  
*Mode options available for the LPIT timer.*
- enum [lpit\\_trigger\\_select\\_t](#) {

```
kLPIT_Trigger_TimerChn0 = 0U,
kLPIT_Trigger_TimerChn1,
kLPIT_Trigger_TimerChn2,
kLPIT_Trigger_TimerChn3,
kLPIT_Trigger_TimerChn4,
kLPIT_Trigger_TimerChn5,
kLPIT_Trigger_TimerChn6,
kLPIT_Trigger_TimerChn7,
kLPIT_Trigger_TimerChn8,
kLPIT_Trigger_TimerChn9,
kLPIT_Trigger_TimerChn10,
kLPIT_Trigger_TimerChn11,
kLPIT_Trigger_TimerChn12,
kLPIT_Trigger_TimerChn13,
kLPIT_Trigger_TimerChn14,
kLPIT_Trigger_TimerChn15 }
```

*Trigger options available.*

- enum `lpit_trigger_source_t` {
   
kLPIT\_TriggerSource\_External = 0U,
   
kLPIT\_TriggerSource\_Internal }

*Trigger source options available.*

- enum `lpit_interrupt_enable_t` {
   
kLPIT\_Channel0TimerInterruptEnable = (1U << 0),
   
kLPIT\_Channel1TimerInterruptEnable = (1U << 1),
   
kLPIT\_Channel2TimerInterruptEnable = (1U << 2),
   
kLPIT\_Channel3TimerInterruptEnable = (1U << 3) }

*List of LPIT interrupts.*

- enum `lpit_status_flags_t` {
   
kLPIT\_Channel0TimerFlag = (1U << 0),
   
kLPIT\_Channel1TimerFlag = (1U << 1),
   
kLPIT\_Channel2TimerFlag = (1U << 2),
   
kLPIT\_Channel3TimerFlag = (1U << 3) }

*List of LPIT status flags.*

- #define `FSL_LPIT_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 0))
   
*Version 2.0.0.*

## Initialization and deinitialization

- void `LPIT_Init` (LPIT\_Type \*base, const `lpit_config_t` \*config)
   
*Ungates the LPIT clock and configures the peripheral for a basic operation.*
- void `LPIT_Deinit` (LPIT\_Type \*base)
   
*Disables the module and gates the LPIT clock.*
- void `LPIT_GetDefaultConfig` (`lpit_config_t` \*config)
   
*Fills in the LPIT configuration structure with default settings.*
- status\_t `LPIT_SetupChannel` (LPIT\_Type \*base, `lpit_chnl_t` channel, const `lpit_chnl_params_t` \*chnlSetup)
   
*Sets up an LPIT channel based on the user's preference.*

## Data Structure Documentation

### Interrupt Interface

- static void [LPIT\\_EnableInterrupts](#) (LPIT\_Type \*base, uint32\_t mask)  
*Enables the selected PIT interrupts.*
- static void [LPIT\\_DisableInterrupts](#) (LPIT\_Type \*base, uint32\_t mask)  
*Disables the selected PIT interrupts.*
- static uint32\_t [LPIT\\_GetEnabledInterrupts](#) (LPIT\_Type \*base)  
*Gets the enabled LPIT interrupts.*

### Status Interface

- static uint32\_t [LPIT\\_GetStatusFlags](#) (LPIT\_Type \*base)  
*Gets the LPIT status flags.*
- static void [LPIT\\_ClearStatusFlags](#) (LPIT\_Type \*base, uint32\_t mask)  
*Clears the LPIT status flags.*

### Read and Write the timer period

- static void [LPIT\\_SetTimerPeriod](#) (LPIT\_Type \*base, lpit\_chnl\_t channel, uint32\_t ticks)  
*Sets the timer period in units of count.*
- static uint32\_t [LPIT\\_GetCurrentTimerCount](#) (LPIT\_Type \*base, lpit\_chnl\_t channel)  
*Reads the current timer counting value.*

### Timer Start and Stop

- static void [LPIT\\_StartTimer](#) (LPIT\_Type \*base, lpit\_chnl\_t channel)  
*Starts the timer counting.*
- static void [LPIT\\_StopTimer](#) (LPIT\_Type \*base, lpit\_chnl\_t channel)  
*Stops the timer counting.*

## 24.4 Data Structure Documentation

### 24.4.1 struct lpit\_chnl\_params\_t

#### Data Fields

- bool [chainChannel](#)  
*true: Timer chained to previous timer; false: Timer not chained*
- [lpit\\_timer\\_modes\\_t timerMode](#)  
*Timers mode of operation.*
- [lpit\\_trigger\\_select\\_t triggerSelect](#)  
*Trigger selection for the timer.*
- [lpit\\_trigger\\_source\\_t triggerSource](#)  
*Decides if we use external or internal trigger.*
- bool [enableReloadOnTrigger](#)  
*true: Timer reloads when a trigger is detected; false: No effect*
- bool [enableStopOnTimeout](#)  
*true: Timer will stop after timeout; false: does not stop after timeout*
- bool [enableStartOnTrigger](#)  
*true: Timer starts when a trigger is detected; false: decrement immediately*

**24.4.1.0.0.73 Field Documentation****24.4.1.0.0.73.1 lpit\_timer\_modes\_t lpit\_chnl\_params\_t::timerMode****24.4.1.0.0.73.2 lpit\_trigger\_source\_t lpit\_chnl\_params\_t::triggerSource****24.4.2 struct lpit\_config\_t**

This structure holds the configuration settings for the LPIT peripheral. To initialize this structure to reasonable defaults, call the [LPIT\\_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 [enableRunInDebug](#)  
*true: Timers run in debug mode; false: Timers stop in debug mode*
- bool [enableRunInDoze](#)  
*true: Timers run in doze mode; false: Timers stop in doze mode*

**24.5 Enumeration Type Documentation****24.5.1 enum lpit\_chnl\_t**

Note

Actual number of available channels is SoC-dependent

Enumerator

**kLPIT\_Chnl\_0** LPIT channel number 0.

**kLPIT\_Chnl\_1** LPIT channel number 1.

**kLPIT\_Chnl\_2** LPIT channel number 2.

**kLPIT\_Chnl\_3** LPIT channel number 3.

**24.5.2 enum lpit\_timer\_modes\_t**

Enumerator

**kLPIT\_PeriodicCounter** Use the all 32-bits, counter loads and decrements to zero.

**kLPIT\_DualPeriodicCounter** Counter loads, lower 16-bits decrement to zero, then upper 16-bits decrement.

**kLPIT\_TriggerAccumulator** Counter loads on first trigger and decrements on each trigger.

**kLPIT\_InputCapture** Counter loads with 0xFFFFFFFF, decrements to zero. It stores the inverse of the current value when a input trigger is detected

## Enumeration Type Documentation

### 24.5.3 enum lpit\_trigger\_select\_t

This is used for both internal and external trigger sources. The actual trigger options available is SoC-specific, user should refer to the reference manual.

Enumerator

|                                 |                                             |
|---------------------------------|---------------------------------------------|
| <i>kLPIT_Trigger_TimerChn0</i>  | Channel 0 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn1</i>  | Channel 1 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn2</i>  | Channel 2 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn3</i>  | Channel 3 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn4</i>  | Channel 4 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn5</i>  | Channel 5 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn6</i>  | Channel 6 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn7</i>  | Channel 7 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn8</i>  | Channel 8 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn9</i>  | Channel 9 is selected as a trigger source.  |
| <i>kLPIT_Trigger_TimerChn10</i> | Channel 10 is selected as a trigger source. |
| <i>kLPIT_Trigger_TimerChn11</i> | Channel 11 is selected as a trigger source. |
| <i>kLPIT_Trigger_TimerChn12</i> | Channel 12 is selected as a trigger source. |
| <i>kLPIT_Trigger_TimerChn13</i> | Channel 13 is selected as a trigger source. |
| <i>kLPIT_Trigger_TimerChn14</i> | Channel 14 is selected as a trigger source. |
| <i>kLPIT_Trigger_TimerChn15</i> | Channel 15 is selected as a trigger source. |

### 24.5.4 enum lpit\_trigger\_source\_t

Enumerator

|                                     |                             |
|-------------------------------------|-----------------------------|
| <i>kLPIT_TriggerSource_External</i> | Use external trigger input. |
| <i>kLPIT_TriggerSource_Internal</i> | Use internal trigger.       |

### 24.5.5 enum lpit\_interrupt\_enable\_t

Note

Number of timer channels are SoC-specific. See the SoC Reference Manual.

Enumerator

|                                           |                            |
|-------------------------------------------|----------------------------|
| <i>kLPIT_Channel0TimerInterruptEnable</i> | Channel 0 Timer interrupt. |
| <i>kLPIT_Channel1TimerInterruptEnable</i> | Channel 1 Timer interrupt. |
| <i>kLPIT_Channel2TimerInterruptEnable</i> | Channel 2 Timer interrupt. |
| <i>kLPIT_Channel3TimerInterruptEnable</i> | Channel 3 Timer interrupt. |

## 24.5.6 enum lpit\_status\_flags\_t

Note

Number of timer channels are SoC-specific. See the SoC Reference Manual.

Enumerator

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kLPIT_Channel0TimerFlag</i> | Channel 0 Timer interrupt flag. |
| <i>kLPIT_Channel1TimerFlag</i> | Channel 1 Timer interrupt flag. |
| <i>kLPIT_Channel2TimerFlag</i> | Channel 2 Timer interrupt flag. |
| <i>kLPIT_Channel3TimerFlag</i> | Channel 3 Timer interrupt flag. |

## 24.6 Function Documentation

### 24.6.1 void LPIT\_Init ( LPIT\_Type \* *base*, const lpit\_config\_t \* *config* )

This function issues a software reset to reset all channels and registers except the Module Control register.

Note

This API should be called at the beginning of the application using the LPIT driver.

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | LPIT peripheral base address.                |
| <i>config</i> | Pointer to the user configuration structure. |

### 24.6.2 void LPIT\_Deinit ( LPIT\_Type \* *base* )

Parameters

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

### 24.6.3 void LPIT\_GetDefaultConfig ( lpit\_config\_t \* *config* )

The default values are:

```
* config->enableRunInDebug = false;
* config->enableRunInDoze = false;
*
```

## Function Documentation

Parameters

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

### 24.6.4 **status\_t LPIT\_SetupChannel ( LPIT\_Type \* *base*, Ipit\_chnl\_t *channel*, const Ipit\_chnl\_params\_t \* *chnlSetup* )**

This function sets up the operation mode to one of the options available in the enumeration [Ipit\\_timer\\_modes\\_t](#). It sets the trigger source as either internal or external, trigger selection and the timers behaviour when a timeout occurs. It also chains the timer if a prior timer if requested by the user.

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>base</i>      | LPIT peripheral base address.     |
| <i>channel</i>   | Channel that is being configured. |
| <i>chnlSetup</i> | Configuration parameters.         |

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

Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPIT peripheral base address.                                                                                        |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">Ipit_interrupt_enable_t</a> |

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

Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPIT peripheral base address.                                                                                        |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">Ipit_interrupt_enable_t</a> |

24.6.7 **static uint32\_t LPIT\_GetEnabledInterrupts ( LPIT\_Type \* *base* )**  
[**inline**], [**static**]

## Function Documentation

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [lpit\\_interrupt\\_enable\\_t](#)

### 24.6.8 static uint32\_t LPIT\_GetStatusFlags ( LPIT\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [lpit\\_status\\_flags\\_t](#)

### 24.6.9 static void LPIT\_ClearStatusFlags ( LPIT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPIT peripheral base address.                                                                                     |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">lpit_status_flags_t</a> |

### 24.6.10 static void LPIT\_SetTimerPeriod ( LPIT\_Type \* *base*, lpit\_chnl\_t *channel*, uint32\_t *ticks* ) [inline], [static]

Timers begin counting down from the value set by this function until it reaches 0, at which point it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

Note

User can call the utility macros provided in `fsl_common.h` to convert to ticks.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | LPIT peripheral base address.   |
| <i>channel</i> | Timer channel number.           |
| <i>ticks</i>   | Timer period in units of ticks. |

#### 24.6.11 static uint32\_t LPIT\_GetCurrentTimerCount ( LPIT\_Type \* *base*, lpit\_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 microseconds or milliseconds.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | LPIT peripheral base address. |
| <i>channel</i> | Timer channel number.         |

Returns

Current timer counting value in ticks.

#### 24.6.12 static void LPIT\_StartTimer ( LPIT\_Type \* *base*, lpit\_chnl\_t *channel* ) [inline], [static]

After calling this function, timers load the period value and count down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | LPIT peripheral base address. |
| <i>channel</i> | Timer channel number.         |

#### 24.6.13 static void LPIT\_StopTimer ( LPIT\_Type \* *base*, lpit\_chnl\_t *channel* ) [inline], [static]

## Function Documentation

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | LPIT peripheral base address. |
| <i>channel</i> | Timer channel number.         |

### 24.6.14 static void LPIT\_Reset( LPIT\_Type \* *base* ) [inline], [static]

This resets all channels and registers except the Module Control Register.

Parameters

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

# Chapter 25

## LPSPI: Low Power Serial Peripheral Interface

### 25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power Serial Peripheral Interface (LPSPI) module of MCUXpresso SDK devices.

### Modules

- [LPSPI FreeRTOS Driver](#)
- [LPSPI Peripheral driver](#)
- [LPSPI eDMA Driver](#)

### 25.2 LPSPI Peripheral driver

#### 25.2.1 Overview

This section describes the programming interface of the LPSPI Peripheral driver. The LPSPI driver configures LPSPI module, provides the functional and transactional interfaces to build the LPSPI application.

#### 25.2.2 Function groups

##### 25.2.2.1 LPSPI Initialization and De-initialization

This function group initializes the default configuration structure for master and slave, initializes the LPSPI master with a master configuration, initializes the LPSPI slave with a slave configuration, and de-initializes the LPSPI module.

##### 25.2.2.2 LPSPI Basic Operation

This function group enables/disables the LPSPI module both interrupt and DMA, gets the data register address for the DMA transfer, sets master and slave, starts and stops the transfer, and so on.

##### 25.2.2.3 LPSPI Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

##### 25.2.2.4 LPSPI Status Operation

This function group gets/clears the LPSPI status.

##### 25.2.2.5 LPSPI Block Transfer Operation

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

#### 25.2.3 Typical use case

##### 25.2.3.1 Master Operation

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

### 25.2.3.2 Slave Operation

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

## Data Structures

- struct `lpspi_master_config_t`  
*LPSPI master configuration structure. [More...](#)*
- struct `lpspi_slave_config_t`  
*LPSPI slave configuration structure. [More...](#)*
- struct `lpspi_transfer_t`  
*LPSPI master/slave transfer structure. [More...](#)*
- struct `lpspi_master_handle_t`  
*LPSPI master transfer handle structure used for transactional API. [More...](#)*
- struct `lpspi_slave_handle_t`  
*LPSPI slave transfer handle structure used for transactional API. [More...](#)*

## Macros

- #define `LPSPI_DUMMY_DATA` (0x00U)  
*LPSPI dummy data if no Tx data.*
- #define `LPSPI_MASTER_PCS_SHIFT` (4U)  
*LPSPI master PCS shift macro , internal used.*
- #define `LPSPI_MASTER_PCS_MASK` (0xF0U)  
*LPSPI master PCS shift macro , internal used.*
- #define `LPSPI_SLAVE_PCS_SHIFT` (4U)  
*LPSPI slave PCS shift macro , internal used.*
- #define `LPSPI_SLAVE_PCS_MASK` (0xF0U)  
*LPSPI slave PCS shift macro , internal used.*

## Typedefs

- typedef void(\* `lpspi_master_transfer_callback_t` )(LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, status\_t status, void \*userData)  
*Master completion callback function pointer type.*
- typedef void(\* `lpspi_slave_transfer_callback_t` )(LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, status\_t status, void \*userData)  
*Slave completion callback function pointer type.*

## Enumerations

- enum `_lpspi_status` {
   
`kStatus_LPSPI_Busy` = MAKE\_STATUS(kStatusGroup\_LPSPI, 0),
   
`kStatus_LPSPI_Error` = MAKE\_STATUS(kStatusGroup\_LPSPI, 1),
   
`kStatus_LPSPI_Idle` = MAKE\_STATUS(kStatusGroup\_LPSPI, 2),
 }

## LPSPI Peripheral driver

- ```
kStatus_LPSPI_OutOfRange = MAKE_STATUS(kStatusGroup_LPSPI, 3) }

Status for the LPSPI driver.
• enum _lpspi_flags {
    kLPSPI_TxDataRequestFlag = LPSPI_SR_TDF_MASK,
    kLPSPI_RxDataReadyFlag = LPSPI_SR_RDF_MASK,
    kLPSPI_WordCompleteFlag = LPSPI_SR_WCF_MASK,
    kLPSPI_FrameCompleteFlag = LPSPI_SR_FCF_MASK,
    kLPSPI_TransferCompleteFlag = LPSPI_SR_TCF_MASK,
    kLPSPI_TransmitErrorFlag = LPSPI_SR_TEF_MASK,
    kLPSPI_ReceiveErrorFlag = LPSPI_SR_REF_MASK,
    kLPSPI_DataMatchFlag = LPSPI_SR_DMF_MASK,
    kLPSPI_ModuleBusyFlag = LPSPI_SR_MBF_MASK,
    kLPSPI_AllStatusFlag }

LPSPI status flags in SPIx_SR register.
• enum _lpspi_interrupt_enable {
    kLPSPI_TxInterruptEnable = LPSPI_IER_TDIE_MASK,
    kLPSPI_RxInterruptEnable = LPSPI_IER_RDIE_MASK,
    kLPSPI_WordCompleteInterruptEnable = LPSPI_IER_WCIE_MASK,
    kLPSPI_FrameCompleteInterruptEnable = LPSPI_IER_FCIE_MASK,
    kLPSPI_TransferCompleteInterruptEnable = LPSPI_IER_TCIE_MASK,
    kLPSPI_TransmitErrorInterruptEnable = LPSPI_IER_TEIE_MASK,
    kLPSPI_ReceiveErrorInterruptEnable = LPSPI_IER_REIE_MASK,
    kLPSPI_DataMatchInterruptEnable = LPSPI_IER_DMIE_MASK,
    kLPSPI_AllInterruptEnable }

LPSPI interrupt source.
• enum _lpspi_dma_enable {
    kLPSPI_TxDmaEnable = LPSPI_DER_TDDE_MASK,
    kLPSPI_RxDmaEnable = LPSPI_DER_RDDE_MASK }

LPSPI DMA source.
• enum lpspi_master_slave_mode_t {
    kLPSPI_Master = 1U,
    kLPSPI_Slave = 0U }

LPSPI master or slave mode configuration.
• enum lpspi_which_pcs_t {
    kLPSPI_Pcs0 = 0U,
    kLPSPI_Pcs1 = 1U,
    kLPSPI_Pcs2 = 2U,
    kLPSPI_Pcs3 = 3U }

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure).
• enum lpspi_pcs_polarity_config_t {
    kLPSPI_PcsActiveHigh = 1U,
    kLPSPI_PcsActiveLow = 0U }

LPSPI Peripheral Chip Select (PCS) Polarity configuration.
• enum _lpspi_pcs_polarity {
```

```

kLPSPI_Pcs0ActiveLow = 1U << 0,
kLPSPI_Pcs1ActiveLow = 1U << 1,
kLPSPI_Pcs2ActiveLow = 1U << 2,
kLPSPI_Pcs3ActiveLow = 1U << 3,
kLPSPI_PcsAllActiveLow = 0xFU }

```

LPSPI Peripheral Chip Select (PCS) Polarity.

- enum `lpspi_clock_polarity_t` {
 `kLPSPI_ClockPolarityActiveHigh` = 0U,
 `kLPSPI_ClockPolarityActiveLow` = 1U }

LPSPI clock polarity configuration.

- enum `lpspi_clock_phase_t` {
 `kLPSPI_ClockPhaseFirstEdge` = 0U,
 `kLPSPI_ClockPhaseSecondEdge` = 1U }

LPSPI clock phase configuration.

- enum `lpspi_shift_direction_t` {
 `kLPSPI_MsbFirst` = 0U,
 `kLPSPI_LsbFirst` = 1U }

LPSPI data shifter direction options.

- enum `lpspi_host_request_select_t` {
 `kLPSPI_HostReqExtPin` = 0U,
 `kLPSPI_HostReqInternalTrigger` = 1U }

LPSPI Host Request select configuration.

- enum `lpspi_match_config_t` {
 `kLPSI_MatchDisabled` = 0x0U,
 `kLPSI_1stWordEqualsM0orM1` = 0x2U,
 `kLPSI_AnyWordEqualsM0orM1` = 0x3U,
 `kLPSI_1stWordEqualsM0and2ndWordEqualsM1` = 0x4U,
 `kLPSI_AnyWordEqualsM0andNxtWordEqualsM1` = 0x5U,
 `kLPSI_1stWordAndM1EqualsM0andM1` = 0x6U,
 `kLPSI_AnyWordAndM1EqualsM0andM1` = 0x7U }

LPSPI Match configuration options.

- enum `lpspi_pin_config_t` {
 `kLPSPI_SdiInSdoOut` = 0U,
 `kLPSPI_SdiInSdiOut` = 1U,
 `kLPSPI_SdoInSdoOut` = 2U,
 `kLPSPI_SdoInSdiOut` = 3U }

LPSPI pin (SDO and SDI) configuration.

- enum `lpspi_data_out_config_t` {
 `kLpspiDataOutRetained` = 0U,
 `kLpspiDataOutTristate` = 1U }

LPSPI data output configuration.

- enum `lpspi_transfer_width_t` {
 `kLPSPI_SingleBitXfer` = 0U,
 `kLPSPI_TwoBitXfer` = 1U,
 `kLPSPI_FourBitXfer` = 2U }

LPSPI transfer width configuration.

- enum `lpspi_delay_type_t` {

LPSPI Peripheral driver

- ```
kLPSPI_PcsToSck = 1U,
kLPSPI_LastSckToPcs,
kLPSPI_BetweenTransfer }
```
- LPSPI delay type selection.*
- enum `_lpspi_transfer_config_flag_for_master` {  
    kLPSPI\_MasterPcs0 = 0U << LPSPI\_MASTER\_PCS\_SHIFT,  
    kLPSPI\_MasterPcs1 = 1U << LPSPI\_MASTER\_PCS\_SHIFT,  
    kLPSPI\_MasterPcs2 = 2U << LPSPI\_MASTER\_PCS\_SHIFT,  
    kLPSPI\_MasterPcs3 = 3U << LPSPI\_MASTER\_PCS\_SHIFT,  
    kLPSPI\_MasterPcsContinuous = 1U << 20,  
    kLPSPI\_MasterByteSwap }
- Use this enumeration for LPSPI master transfer configFlags.*
- enum `_lpspi_transfer_config_flag_for_slave` {  
    kLPSPI\_SlavePcs0 = 0U << LPSPI\_SLAVE\_PCS\_SHIFT,  
    kLPSPI\_SlavePcs1 = 1U << LPSPI\_SLAVE\_PCS\_SHIFT,  
    kLPSPI\_SlavePcs2 = 2U << LPSPI\_SLAVE\_PCS\_SHIFT,  
    kLPSPI\_SlavePcs3 = 3U << LPSPI\_SLAVE\_PCS\_SHIFT,  
    kLPSPI\_SlaveByteSwap }
- Use this enumeration for LPSPI slave transfer configFlags.*
- enum `_lpspi_transfer_state` {  
    kLPSPI\_Idle = 0x0U,  
    kLPSPI\_Busy,  
    kLPSPI\_Error }
- LPSPI transfer state, which is used for LPSPI transactional API state machine.*

## Variables

- volatile uint8\_t `g_lpspiDummyData` []  
*Global variable for dummy data value setting.*

## Driver version

- #define `FSL_LPSPI_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 3))  
*LPSPI driver version 2.0.3.*

## Initialization and deinitialization

- void `LPSPI_MasterInit` (LPSPI\_Type \*base, const `lpspi_master_config_t` \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the LPSPI master.*
- void `LPSPI_MasterGetDefaultConfig` (`lpspi_master_config_t` \*masterConfig)  
*Sets the `lpspi_master_config_t` structure to default values.*
- void `LPSPI_SlaveInit` (LPSPI\_Type \*base, const `lpspi_slave_config_t` \*slaveConfig)  
*LPSPI slave configuration.*

- void [LPSPI\\_SlaveGetDefaultConfig](#) ([lpspi\\_slave\\_config\\_t](#) \*slaveConfig)  
*Sets the [lpspi\\_slave\\_config\\_t](#) structure to default values.*
- void [LPSPI\\_Deinit](#) ([LPSPI\\_Type](#) \*base)  
*De-initializes the LPSPI peripheral.*
- void [LPSPI\\_Reset](#) ([LPSPI\\_Type](#) \*base)  
*Restores the LPSPI peripheral to reset state.*
- static void [LPSPI\\_Enable](#) ([LPSPI\\_Type](#) \*base, bool enable)  
*Enables the LPSPI peripheral and sets the MCR MDIS to 0.*

## Status

- static uint32\_t [LPSPI\\_GetStatusFlags](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI status flag state.*
- static uint32\_t [LPSPI\\_GetTxFifoSize](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Tx FIFO size.*
- static uint32\_t [LPSPI\\_GetRxFifoSize](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Rx FIFO size.*
- static uint32\_t [LPSPI\\_GetTxFifoCount](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Tx FIFO count.*
- static uint32\_t [LPSPI\\_GetRxFifoCount](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Rx FIFO count.*
- static void [LPSPI\\_ClearStatusFlags](#) ([LPSPI\\_Type](#) \*base, uint32\_t statusFlags)  
*Clears the LPSPI status flag.*

## Interrupts

- static void [LPSPI\\_EnableInterrupts](#) ([LPSPI\\_Type](#) \*base, uint32\_t mask)  
*Enables the LPSPI interrupts.*
- static void [LPSPI\\_DisableInterrupts](#) ([LPSPI\\_Type](#) \*base, uint32\_t mask)  
*Disables the LPSPI interrupts.*

## DMA Control

- static void [LPSPI\\_EnableDMA](#) ([LPSPI\\_Type](#) \*base, uint32\_t mask)  
*Enables the LPSPI DMA request.*
- static void [LPSPI\\_DisableDMA](#) ([LPSPI\\_Type](#) \*base, uint32\_t mask)  
*Disables the LPSPI DMA request.*
- static uint32\_t [LPSPI\\_GetTxRegisterAddress](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Transmit Data Register address for a DMA operation.*
- static uint32\_t [LPSPI\\_GetRxRegisterAddress](#) ([LPSPI\\_Type](#) \*base)  
*Gets the LPSPI Receive Data Register address for a DMA operation.*

## Bus Operations

- bool [LPSPI\\_CheckTransferArgument](#) ([lpspi\\_transfer\\_t](#) \*transfer, uint32\_t bitsPerFrame, uint32\_t bytesPerFrame)

## LPSPI Peripheral driver

*Check the argument for transfer.*

- static void **LPSPI\_SetMasterSlaveMode** (LPSPI\_Type \*base, lpspi\_master\_slave\_mode\_t mode)  
*Configures the LPSPI for either master or slave.*
- static bool **LPSPI\_IsMaster** (LPSPI\_Type \*base)  
*Returns whether the LPSPI module is in master mode.*
- static void **LPSPI\_FlushFifo** (LPSPI\_Type \*base, bool flushTxFifo, bool flushRxFifo)  
*Flushes the LPSPI FIFOs.*
- static void **LPSPI\_SetFifoWatermarks** (LPSPI\_Type \*base, uint32\_t txWater, uint32\_t rxWater)  
*Sets the transmit and receive FIFO watermark values.*
- static void **LPSPI\_SetAllPcsPolarity** (LPSPI\_Type \*base, uint32\_t mask)  
*Configures all LPSPI peripheral chip select polarities simultaneously.*
- static void **LPSPI\_SetFrameSize** (LPSPI\_Type \*base, uint32\_t frameSize)  
*Configures the frame size.*
- uint32\_t **LPSPI\_MasterSetBaudRate** (LPSPI\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz, uint32\_t \*tcrPrescaleValue)  
*Sets the LPSPI baud rate in bits per second.*
- void **LPSPI\_MasterSetDelayScaler** (LPSPI\_Type \*base, uint32\_t scaler, lpspi\_delay\_type\_t whichDelay)  
*Manually configures a specific LPSPI delay parameter (module must be disabled to change the delay values).*
- uint32\_t **LPSPI\_MasterSetDelayTimes** (LPSPI\_Type \*base, uint32\_t delayTimeInNanoSec, lpspi\_delay\_type\_t whichDelay, uint32\_t srcClock\_Hz)  
*Calculates the delay based on the desired delay input in nanoseconds (module must be disabled to change the delay values).*
- static void **LPSPI\_WriteData** (LPSPI\_Type \*base, uint32\_t data)  
*Writes data into the transmit data buffer.*
- static uint32\_t **LPSPI\_ReadData** (LPSPI\_Type \*base)  
*Reads data from the data buffer.*
- void **LPSPI\_SetDummyData** (LPSPI\_Type \*base, uint8\_t dummyData)  
*Set up the dummy data.*

## Transactional

- void **LPSPI\_MasterTransferCreateHandle** (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, lpspi\_master\_transfer\_callback\_t callback, void \*userData)  
*Initializes the LPSPI master handle.*
- status\_t **LPSPI\_MasterTransferBlocking** (LPSPI\_Type \*base, lpspi\_transfer\_t \*transfer)  
*LPSPI master transfer data using a polling method.*
- status\_t **LPSPI\_MasterTransferNonBlocking** (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, lpspi\_transfer\_t \*transfer)  
*LPSPI master transfer data using an interrupt method.*
- status\_t **LPSPI\_MasterTransferGetCount** (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the master transfer remaining bytes.*
- void **LPSPI\_MasterTransferAbort** (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle)  
*LPSPI master abort transfer which uses an interrupt method.*
- void **LPSPI\_MasterTransferHandleIRQ** (LPSPI\_Type \*base, lpspi\_master\_handle\_t \*handle)  
*LPSPI Master IRQ handler function.*
- void **LPSPI\_SlaveTransferCreateHandle** (LPSPI\_Type \*base, lpspi\_slave\_handle\_t \*handle, lpspi-

- `_slave_transfer_callback_t callback, void *userData)`  
*Initializes the LPSPI slave handle.*
- `status_t LPSPI_SlaveTransferNonBlocking (LPSPI_Type *base, lpspi_slave_handle_t *handle, lpspi_transfer_t *transfer)`  
*LPSPI slave transfer data using an interrupt method.*
- `status_t LPSPI_SlaveTransferGetCount (LPSPI_Type *base, lpspi_slave_handle_t *handle, size_t *count)`  
*Gets the slave transfer remaining bytes.*
- `void LPSPI_SlaveTransferAbort (LPSPI_Type *base, lpspi_slave_handle_t *handle)`  
*LPSPI slave aborts a transfer which uses an interrupt method.*
- `void LPSPI_SlaveTransferHandleIRQ (LPSPI_Type *base, lpspi_slave_handle_t *handle)`  
*LPSPI Slave IRQ handler function.*

## 25.2.4 Data Structure Documentation

### 25.2.4.1 struct lpspi\_master\_config\_t

#### Data Fields

- `uint32_t baudRate`  
*Baud Rate for LPSPI.*
- `uint32_t bitsPerFrame`  
*Bits per frame, minimum 8, maximum 4096.*
- `lpspi_clock_polarity_t cpol`  
*Clock polarity.*
- `lpspi_clock_phase_t cpha`  
*Clock phase.*
- `lpspi_shift_direction_t direction`  
*MSB or LSB data shift direction.*
- `uint32_t pcsToSckDelayInNanoSec`  
*PCS to SCK delay time in nanoseconds, setting to 0 sets the minimum delay.*
- `uint32_t lastSckToPcsDelayInNanoSec`  
*Last SCK to PCS delay time in nanoseconds, setting to 0 sets the minimum delay.*
- `uint32_t betweenTransferDelayInNanoSec`  
*After the SCK delay time with nanoseconds, setting to 0 sets the minimum delay.*
- `lpspi_which_pcs_t whichPcs`  
*Desired Peripheral Chip Select (PCS).*
- `lpspi_pcs_polarity_config_t pcsActiveHighOrLow`  
*Desired PCS active high or low.*
- `lpspi_pin_config_t pinCfg`  
*Configures which pins are used for input and output data during single bit transfers.*
- `lpspi_data_out_config_t dataOutConfig`  
*Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).*

## LPSPI Peripheral driver

### 25.2.4.1.0.74 Field Documentation

25.2.4.1.0.74.1 `uint32_t lpspi_master_config_t::baudRate`

25.2.4.1.0.74.2 `uint32_t lpspi_master_config_t::bitsPerFrame`

25.2.4.1.0.74.3 `lpspi_clock_polarity_t lpspi_master_config_t::cpol`

25.2.4.1.0.74.4 `lpspi_clock_phase_t lpspi_master_config_t::cpha`

25.2.4.1.0.74.5 `lpspi_shift_direction_t lpspi_master_config_t::direction`

25.2.4.1.0.74.6 `uint32_t lpspi_master_config_t::pcsToSckDelayInNanoSec`

It sets the boundary value if out of range.

25.2.4.1.0.74.7 `uint32_t lpspi_master_config_t::lastSckToPcsDelayInNanoSec`

It sets the boundary value if out of range.

25.2.4.1.0.74.8 `uint32_t lpspi_master_config_t::betweenTransferDelayInNanoSec`

It sets the boundary value if out of range.

25.2.4.1.0.74.9 `lpspi_which_pcs_t lpspi_master_config_t::whichPcs`

25.2.4.1.0.74.10 `lpspi_pin_config_t lpspi_master_config_t::pinCfg`

25.2.4.1.0.74.11 `lpspi_data_out_config_t lpspi_master_config_t::dataOutConfig`

### 25.2.4.2 `struct lpspi_slave_config_t`

#### Data Fields

- `uint32_t bitsPerFrame`  
*Bits per frame, minimum 8, maximum 4096.*
- `lpspi_clock_polarity_t cpol`  
*Clock polarity.*
- `lpspi_clock_phase_t cpha`  
*Clock phase.*
- `lpspi_shift_direction_t direction`  
*MSB or LSB data shift direction.*
- `lpspi_which_pcs_t whichPcs`  
*Desired Peripheral Chip Select (pcs)*
- `lpspi_pcs_polarity_config_t pcsActiveHighOrLow`  
*Desired PCS active high or low.*
- `lpspi_pin_config_t pinCfg`  
*Configures which pins are used for input and output data during single bit transfers.*
- `lpspi_data_out_config_t dataOutConfig`  
*Configures if the output data is tristated between accesses (LPSPI\_PCS is negated).*

#### 25.2.4.2.0.75 Field Documentation

25.2.4.2.0.75.1 `uint32_t lpspi_slave_config_t::bitsPerFrame`

25.2.4.2.0.75.2 `lpspi_clock_polarity_t lpspi_slave_config_t::cpol`

25.2.4.2.0.75.3 `lpspi_clock_phase_t lpspi_slave_config_t::cpha`

25.2.4.2.0.75.4 `lpspi_shift_direction_t lpspi_slave_config_t::direction`

25.2.4.2.0.75.5 `lpspi_pin_config_t lpspi_slave_config_t::pinCfg`

25.2.4.2.0.75.6 `lpspi_data_out_config_t lpspi_slave_config_t::dataOutConfig`

#### 25.2.4.3 `struct lpspi_transfer_t`

##### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `volatile size_t dataSize`  
*Transfer bytes.*
- `uint32_t configFlags`  
*Transfer transfer configuration flags.*

#### 25.2.4.3.0.76 Field Documentation

25.2.4.3.0.76.1 `uint8_t* lpspi_transfer_t::txData`

25.2.4.3.0.76.2 `uint8_t* lpspi_transfer_t::rxData`

25.2.4.3.0.76.3 `volatile size_t lpspi_transfer_t::dataSize`

25.2.4.3.0.76.4 `uint32_t lpspi_transfer_t::configFlags`

Set from `_lpspi_transfer_config_flag_for_master` if the transfer is used for master or `_lpspi_transfer_config_flag_for_slave` enumeration if the transfer is used for slave.

#### 25.2.4.4 `struct _lpspi_master_handle`

Forward declaration of the `_lpspi_master_handle` typedefs.

##### Data Fields

- `volatile bool isPcsContinuous`  
*Is PCS continuous in transfer.*
- `volatile bool writeTcrInIsr`

## LPSPI Peripheral driver

*A flag that whether should write TCR in ISR.*

- volatile bool **isByteSwap**

*A flag that whether should byte swap.*

- volatile uint8\_t **fifoSize**

*FIFO dataSize.*

- volatile uint8\_t **rxWatermark**

*Rx watermark.*

- volatile uint8\_t **bytesEachWrite**

*Bytes for each write TDR.*

- volatile uint8\_t **bytesEachRead**

*Bytes for each read RDR.*

- uint8\_t \*volatile **txData**

*Send buffer.*

- uint8\_t \*volatile **rxData**

*Receive buffer.*

- volatile size\_t **txRemainingByteCount**

*Number of bytes remaining to send.*

- volatile size\_t **rxRemainingByteCount**

*Number of bytes remaining to receive.*

- volatile uint32\_t **writeRegRemainingTimes**

*Write TDR register remaining times.*

- volatile uint32\_t **readRegRemainingTimes**

*Read RDR register remaining times.*

- uint32\_t **totalByteCount**

*Number of transfer bytes.*

- uint32\_t **txBuffIfNull**

*Used if the txData is NULL.*

- volatile uint8\_t **state**

*LPSPI transfer state , \_lpspi\_transfer\_state.*

- **lpspi\_master\_transfer\_callback\_t callback**

*Completion callback.*

- void \* **userData**

*Callback user data.*

#### 25.2.4.4.0.77 Field Documentation

- 25.2.4.4.0.77.1 `volatile bool lpspi_master_handle_t::isPcsContinuous`
- 25.2.4.4.0.77.2 `volatile bool lpspi_master_handle_t::writeTcrInlsr`
- 25.2.4.4.0.77.3 `volatile bool lpspi_master_handle_t::isByteSwap`
- 25.2.4.4.0.77.4 `volatile uint8_t lpspi_master_handle_t::fifoSize`
- 25.2.4.4.0.77.5 `volatile uint8_t lpspi_master_handle_t::rxWatermark`
- 25.2.4.4.0.77.6 `volatile uint8_t lpspi_master_handle_t::bytesEachWrite`
- 25.2.4.4.0.77.7 `volatile uint8_t lpspi_master_handle_t::bytesEachRead`
- 25.2.4.4.0.77.8 `uint8_t* volatile lpspi_master_handle_t::txData`
- 25.2.4.4.0.77.9 `uint8_t* volatile lpspi_master_handle_t::rxData`
- 25.2.4.4.0.77.10 `volatile size_t lpspi_master_handle_t::txRemainingByteCount`
- 25.2.4.4.0.77.11 `volatile size_t lpspi_master_handle_t::rxRemainingByteCount`
- 25.2.4.4.0.77.12 `volatile uint32_t lpspi_master_handle_t::writeRegRemainingTimes`
- 25.2.4.4.0.77.13 `volatile uint32_t lpspi_master_handle_t::readRegRemainingTimes`
- 25.2.4.4.0.77.14 `uint32_t lpspi_master_handle_t::txBuffIfNull`
- 25.2.4.4.0.77.15 `volatile uint8_t lpspi_master_handle_t::state`
- 25.2.4.4.0.77.16 `lpspi_master_transfer_callback_t lpspi_master_handle_t::callback`
- 25.2.4.4.0.77.17 `void* lpspi_master_handle_t::userData`

#### 25.2.4.5 struct \_lpspi\_slave\_handle

Forward declaration of the `_lpspi_slave_handle` typedefs.

#### Data Fields

- `volatile bool isByteSwap`  
*A flag that whether should byte swap.*
- `volatile uint8_t fifoSize`  
*FIFO dataSize.*
- `volatile uint8_t rxWatermark`  
*Rx watermark.*
- `volatile uint8_t bytesEachWrite`  
*Bytes for each write TDR.*

## LPSPI Peripheral driver

- volatile uint8\_t **bytesEachRead**  
*Bytes for each read RDR.*
- uint8\_t \*volatile **txData**  
*Send buffer.*
- uint8\_t \*volatile **rxData**  
*Receive buffer.*
- volatile size\_t **txRemainingByteCount**  
*Number of bytes remaining to send.*
- volatile size\_t **rxRemainingByteCount**  
*Number of bytes remaining to receive.*
- volatile uint32\_t **writeRegRemainingTimes**  
*Write TDR register remaining times.*
- volatile uint32\_t **readRegRemainingTimes**  
*Read RDR register remaining times.*
- uint32\_t **totalByteCount**  
*Number of transfer bytes.*
- volatile uint8\_t **state**  
*LPSPI transfer state , \_lpspi\_transfer\_state.*
- volatile uint32\_t **errorCount**  
*Error count for slave transfer.*
- **lpspi\_slave\_transfer\_callback\_t callback**  
*Completion callback.*
- void \* **userData**  
*Callback user data.*

#### 25.2.4.5.0.78 Field Documentation

- 25.2.4.5.0.78.1 `volatile bool lpspi_slave_handle_t::isByteSwap`
- 25.2.4.5.0.78.2 `volatile uint8_t lpspi_slave_handle_t::fifoSize`
- 25.2.4.5.0.78.3 `volatile uint8_t lpspi_slave_handle_t::rxWatermark`
- 25.2.4.5.0.78.4 `volatile uint8_t lpspi_slave_handle_t::bytesEachWrite`
- 25.2.4.5.0.78.5 `volatile uint8_t lpspi_slave_handle_t::bytesEachRead`
- 25.2.4.5.0.78.6 `uint8_t* volatile lpspi_slave_handle_t::txData`
- 25.2.4.5.0.78.7 `uint8_t* volatile lpspi_slave_handle_t::rxData`
- 25.2.4.5.0.78.8 `volatile size_t lpspi_slave_handle_t::txRemainingByteCount`
- 25.2.4.5.0.78.9 `volatile size_t lpspi_slave_handle_t::rxRemainingByteCount`
- 25.2.4.5.0.78.10 `volatile uint32_t lpspi_slave_handle_t::writeRegRemainingTimes`
- 25.2.4.5.0.78.11 `volatile uint32_t lpspi_slave_handle_t::readRegRemainingTimes`
- 25.2.4.5.0.78.12 `volatile uint8_t lpspi_slave_handle_t::state`
- 25.2.4.5.0.78.13 `volatile uint32_t lpspi_slave_handle_t::errorCount`
- 25.2.4.5.0.78.14 `lpspi_slave_transfer_callback_t lpspi_slave_handle_t::callback`
- 25.2.4.5.0.78.15 `void* lpspi_slave_handle_t::userData`

#### 25.2.5 Macro Definition Documentation

25.2.5.1 `#define FSL_LPSPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))`

25.2.5.2 `#define LPSPI_DUMMY_DATA (0x00U)`

Dummy data used for tx if there is not txData.

## LPSPI Peripheral driver

25.2.5.3 `#define LPSPI_MASTER_PCS_SHIFT (4U)`

25.2.5.4 `#define LPSPI_MASTER_PCS_MASK (0xF0U)`

25.2.5.5 `#define LPSPI_SLAVE_PCS_SHIFT (4U)`

25.2.5.6 `#define LPSPI_SLAVE_PCS_MASK (0xF0U)`

## 25.2.6 Typedef Documentation

25.2.6.1 `typedef void(* lpspi_master_transfer_callback_t)(LPSPI_Type *base,  
lpspi_master_handle_t *handle, status_t status, void *userData)`

Parameters

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                           |
| <i>handle</i>   | Pointer to the handle for the LPSPI master.                         |
| <i>status</i>   | Success or error code describing whether the transfer is completed. |
| <i>userData</i> | Arbitrary pointer-dataSized value passed from the application.      |

### 25.2.6.2 `typedef void(* lpspi_slave_transfer_callback_t)(LPSPI_Type *base, lpspi_slave_handle_t *handle, status_t status, void *userData)`

Parameters

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                           |
| <i>handle</i>   | Pointer to the handle for the LPSPI slave.                          |
| <i>status</i>   | Success or error code describing whether the transfer is completed. |
| <i>userData</i> | Arbitrary pointer-dataSized value passed from the application.      |

## 25.2.7 Enumeration Type Documentation

### 25.2.7.1 enum \_lpspi\_status

Enumerator

- kStatus\_LPSPI\_Busy* LPSPI transfer is busy.
- kStatus\_LPSPI\_Error* LPSPI driver error.
- kStatus\_LPSPI\_Idle* LPSPI is idle.
- kStatus\_LPSPI\_OutOfRange* LPSPI transfer out Of range.

### 25.2.7.2 enum \_lpspi\_flags

Enumerator

- kLPSPI\_TxDataRequestFlag* Transmit data flag.
- kLPSPI\_RxDataReadyFlag* Receive data flag.
- kLPSPI\_WordCompleteFlag* Word Complete flag.
- kLPSPI\_FrameCompleteFlag* Frame Complete flag.
- kLPSPI\_TransferCompleteFlag* Transfer Complete flag.
- kLPSPI\_TransmitErrorFlag* Transmit Error flag (FIFO underrun)
- kLPSPI\_ReceiveErrorFlag* Receive Error flag (FIFO overrun)
- kLPSPI\_DataMatchFlag* Data Match flag.

## LPSPI Peripheral driver

*kLPSPI\_ModuleBusyFlag* Module Busy flag.

*kLPSPI\_AllStatusFlag* Used for clearing all w1c status flags.

### 25.2.7.3 enum \_lpspi\_interrupt\_enable

Enumerator

*kLPSPI\_TxInterruptEnable* Transmit data interrupt enable.

*kLPSPI\_RxInterruptEnable* Receive data interrupt enable.

*kLPSPI\_WordCompleteInterruptEnable* Word complete interrupt enable.

*kLPSPI\_FrameCompleteInterruptEnable* Frame complete interrupt enable.

*kLPSPI\_TransferCompleteInterruptEnable* Transfer complete interrupt enable.

*kLPSPI\_TransmitErrorInterruptEnable* Transmit error interrupt enable(FIFO underrun)

*kLPSPI\_ReceiveErrorInterruptEnable* Receive Error interrupt enable (FIFO overrun)

*kLPSPI\_DataMatchInterruptEnable* Data Match interrupt enable.

*kLPSPI\_AllInterruptEnable* All above interrupts enable.

### 25.2.7.4 enum \_lpspi\_dma\_enable

Enumerator

*kLPSPI\_TxDmaEnable* Transmit data DMA enable.

*kLPSPI\_RxDmaEnable* Receive data DMA enable.

### 25.2.7.5 enum lpspi\_master\_slave\_mode\_t

Enumerator

*kLPSPI\_Master* LPSPI peripheral operates in master mode.

*kLPSPI\_Slave* LPSPI peripheral operates in slave mode.

### 25.2.7.6 enum lpspi\_which\_pcs\_t

Enumerator

*kLPSPI\_Pcs0* PCS[0].

*kLPSPI\_Pcs1* PCS[1].

*kLPSPI\_Pcs2* PCS[2].

*kLPSPI\_Pcs3* PCS[3].

### 25.2.7.7 enum lpspi\_pcs\_polarity\_config\_t

Enumerator

*kLPSPI\_PcsActiveHigh* PCS Active High (idles low)

*kLPSPI\_PcsActiveLow* PCS Active Low (idles high)

### 25.2.7.8 enum \_lpspi\_pcs\_polarity

Enumerator

*kLPSPI\_Pcs0ActiveLow* Pcs0 Active Low (idles high).

*kLPSPI\_Pcs1ActiveLow* Pcs1 Active Low (idles high).

*kLPSPI\_Pcs2ActiveLow* Pcs2 Active Low (idles high).

*kLPSPI\_Pcs3ActiveLow* Pcs3 Active Low (idles high).

*kLPSPI\_PcsAllActiveLow* Pcs0 to Pcs5 Active Low (idles high).

### 25.2.7.9 enum lpspi\_clock\_polarity\_t

Enumerator

*kLPSPI\_ClockPolarityActiveHigh* CPOL=0. Active-high LPSPI clock (idles low)

*kLPSPI\_ClockPolarityActiveLow* CPOL=1. Active-low LPSPI clock (idles high)

### 25.2.7.10 enum lpspi\_clock\_phase\_t

Enumerator

*kLPSPI\_ClockPhaseFirstEdge* CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.

*kLPSPI\_ClockPhaseSecondEdge* CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

### 25.2.7.11 enum lpspi\_shift\_direction\_t

Enumerator

*kLPSPI\_MsbFirst* Data transfers start with most significant bit.

*kLPSPI\_LsbFirst* Data transfers start with least significant bit.

## LPSPI Peripheral driver

### 25.2.7.12 enum lpspi\_host\_request\_select\_t

Enumerator

*kLPSPI\_HostReqExtPin* Host Request is an ext pin.

*kLPSPI\_HostReqInternalTrigger* Host Request is an internal trigger.

### 25.2.7.13 enum lpspi\_match\_config\_t

Enumerator

*kLPSI\_MatchDisabled* LPSPI Match Disabled.

*kLPSI\_1stWordEqualsM0orM1* LPSPI Match Enabled.

*kLPSI\_AnyWordEqualsM0orM1* LPSPI Match Enabled.

*kLPSI\_1stWordEqualsM0and2ndWordEqualsM1* LPSPI Match Enabled.

*kLPSI\_AnyWordEqualsM0andNxtWordEqualsM1* LPSPI Match Enabled.

*kLPSI\_1stWordAndM1EqualsM0andM1* LPSPI Match Enabled.

*kLPSI\_AnyWordAndM1EqualsM0andM1* LPSPI Match Enabled.

### 25.2.7.14 enum lpspi\_pin\_config\_t

Enumerator

*kLPSPI\_SdiInSdoOut* LPSPI SDI input, SDO output.

*kLPSPI\_SdiInSdiOut* LPSPI SDI input, SDI output.

*kLPSPI\_SdoInSdoOut* LPSPI SDO input, SDO output.

*kLPSPI\_SdoInSdiOut* LPSPI SDO input, SDI output.

### 25.2.7.15 enum lpspi\_data\_out\_config\_t

Enumerator

*kLpspiDataOutRetained* Data out retains last value when chip select is de-asserted.

*kLpspiDataOutTristate* Data out is tristated when chip select is de-asserted.

### 25.2.7.16 enum lpspi\_transfer\_width\_t

Enumerator

*kLPSPI\_SingleBitXfer* 1-bit shift at a time, data out on SDO, in on SDI (normal mode)

*kLPSPI\_TwoBitXfer* 2-bits shift out on SDO/SDI and in on SDO/SDI

*kLPSPI\_FourBitXfer* 4-bits shift out on SDO/SDI/PCS[3:2] and in on SDO/SDI/PCS[3:2]

### 25.2.7.17 enum lpspi\_delay\_type\_t

Enumerator

*kLPSPI\_PcsToSck* PCS-to-SCK delay.

*kLPSPI\_LastSckToPcs* Last SCK edge to PCS delay.

*kLPSPI\_BetweenTransfer* Delay between transfers.

### 25.2.7.18 enum \_lpspi\_transfer\_config\_flag\_for\_master

Enumerator

*kLPSPI\_MasterPcs0* LPSPI master transfer use PCS0 signal.

*kLPSPI\_MasterPcs1* LPSPI master transfer use PCS1 signal.

*kLPSPI\_MasterPcs2* LPSPI master transfer use PCS2 signal.

*kLPSPI\_MasterPcs3* LPSPI master transfer use PCS3 signal.

*kLPSPI\_MasterPcsContinuous* Is PCS signal continuous.

*kLPSPI\_MasterByteSwap* Is master swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi\_shift\_direction\_t to MSB).

1. If you set bitPerFrame = 8 , no matter the kLPSPI\_MasterByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
2. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI\_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI\_MasterByteSwap flag.
3. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI\_MasterByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI\_MasterByteSwap flag.

### 25.2.7.19 enum \_lpspi\_transfer\_config\_flag\_for\_slave

Enumerator

*kLPSPI\_SlavePcs0* LPSPI slave transfer use PCS0 signal.

*kLPSPI\_SlavePcs1* LPSPI slave transfer use PCS1 signal.

*kLPSPI\_SlavePcs2* LPSPI slave transfer use PCS2 signal.

*kLPSPI\_SlavePcs3* LPSPI slave transfer use PCS3 signal.

*kLPSPI\_SlaveByteSwap* Is slave swap the byte. For example, when want to send data 1 2 3 4 5 6 7 8 (suppose you set lpspi\_shift\_direction\_t to MSB).

1. If you set bitPerFrame = 8 , no matter the kLPSPI\_SlaveByteSwap flag is used or not, the waveform is 1 2 3 4 5 6 7 8.
2. If you set bitPerFrame = 16 : (1) the waveform is 2 1 4 3 6 5 8 7 if you do not use the kLPSPI\_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI\_SlaveByteSwap flag.

## LPSPI Peripheral driver

3. If you set bitPerFrame = 32 : (1) the waveform is 4 3 2 1 8 7 6 5 if you do not use the kLPSPI\_SlaveByteSwap flag. (2) the waveform is 1 2 3 4 5 6 7 8 if you use the kLPSPI\_SlaveByteSwap flag.

### 25.2.7.20 enum \_lpspi\_transfer\_state

Enumerator

*kLPSPI\_Idle* Nothing in the transmitter/receiver.

*kLPSPI\_Busy* Transfer queue is not finished.

*kLPSPI\_Error* Transfer error.

## 25.2.8 Function Documentation

### 25.2.8.1 void LPSPI\_MasterInit ( *LPSPI\_Type* \* *base*, *const lpspi\_master\_config\_t* \* *masterConfig*, *uint32\_t srcClock\_Hz* )

Parameters

|                     |                                                              |
|---------------------|--------------------------------------------------------------|
| <i>base</i>         | LPSPI peripheral address.                                    |
| <i>masterConfig</i> | Pointer to structure <a href="#">lpspi_master_config_t</a> . |
| <i>srcClock_Hz</i>  | Module source input clock in Hertz                           |

### 25.2.8.2 void LPSPI\_MasterGetDefaultConfig ( *lpspi\_master\_config\_t* \* *masterConfig* )

This API initializes the configuration structure for [LPSPI\\_MasterInit\(\)](#). The initialized structure can remain unchanged in [LPSPI\\_MasterInit\(\)](#), or can be modified before calling the [LPSPI\\_MasterInit\(\)](#). Example:

```
* lpspi_master_config_t masterConfig;
* LPSPI_MasterGetDefaultConfig(&masterConfig);
*
```

Parameters

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| <i>masterConfig</i> | pointer to <a href="#">lpspi_master_config_t</a> structure |
|---------------------|------------------------------------------------------------|

### 25.2.8.3 void LPSPI\_SlaveInit ( *LPSPI\_Type* \* *base*, *const lpspi\_slave\_config\_t* \* *slaveConfig* )

Parameters

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <i>base</i>        | LPSPI peripheral address.                                     |
| <i>slaveConfig</i> | Pointer to a structure <a href="#">lpspi_slave_config_t</a> . |

#### 25.2.8.4 void LPSPI\_SlaveGetDefaultConfig ( [lpspi\\_slave\\_config\\_t](#) \* *slaveConfig* )

This API initializes the configuration structure for [LPSPI\\_SlaveInit\(\)](#). The initialized structure can remain unchanged in [LPSPI\\_SlaveInit\(\)](#) or can be modified before calling the [LPSPI\\_SlaveInit\(\)](#). Example:

```
* lpspi_slave_config_t slaveConfig;
* LPSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>slaveConfig</i> | pointer to <a href="#">lpspi_slave_config_t</a> structure. |
|--------------------|------------------------------------------------------------|

#### 25.2.8.5 void LPSPI\_Deinit ( [LPSPI\\_Type](#) \* *base* )

Call this API to disable the LPSPI clock.

Parameters

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

#### 25.2.8.6 void LPSPI\_Reset ( [LPSPI\\_Type](#) \* *base* )

Note that this function sets all registers to reset state. As a result, the LPSPI module can't work after calling this API.

Parameters

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

#### 25.2.8.7 static void LPSPI\_Enable ( [LPSPI\\_Type](#) \* *base*, bool *enable* ) [inline], [static]

## LPSPI Peripheral driver

Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                            |
| <i>enable</i> | Pass true to enable module, false to disable module. |

**25.2.8.8 static uint32\_t LPSPI\_GetStatusFlags ( LPSPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The LPSPI status(in SR register).

**25.2.8.9 static uint32\_t LPSPI\_GetTxFifoSize ( LPSPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The LPSPI Tx FIFO size.

**25.2.8.10 static uint32\_t LPSPI\_GetRxFifoSize ( LPSPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The LPSPI Rx FIFO size.

**25.2.8.11 static uint32\_t LPSPI\_GetTxFifoCount ( LPSPI\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The number of words in the transmit FIFO.

#### 25.2.8.12 static uint32\_t LPSPI\_GetRxFifoCount ( LPSPI\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The number of words in the receive FIFO.

#### 25.2.8.13 static void LPSPI\_ClearStatusFlags ( LPSPI\_Type \* *base*, uint32\_t *statusFlags* ) [inline], [static]

This function clears the desired status bit by using a write-1-to-clear. The user passes in the base and the desired status flag bit to clear. The list of status flags is defined in the \_lpspi\_flags. Example usage:

```
* LPSPI_ClearStatusFlags(base, kLPSPI_TxDataRequestFlag|
 kLPSPI_RxDataReadyFlag);
*
```

Parameters

|                    |                                              |
|--------------------|----------------------------------------------|
| <i>base</i>        | LPSPI peripheral address.                    |
| <i>statusFlags</i> | The status flag used from type _lpspi_flags. |

< The status flags are cleared by writing 1 (w1c).

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

This function configures the various interrupt masks of the LPSPI. The parameters are base and an interrupt mask. Note that, for Tx fill and Rx FIFO drain requests, enabling the interrupt request disables the DMA request.

```
* LPSPI_EnableInterrupts(base, kLPSPI_TxInterruptEnable |
 kLPSPI_RxInterruptEnable);
*
```

## LPSPI Peripheral driver

Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                 |
| <i>mask</i> | The interrupt mask; Use the enum _lpspi_interrupt_enable. |

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

```
* LPSPI_DisableInterrupts(base, kLPSPI_TxInterruptEnable |
 kLPSPI_RxInterruptEnable);
*
```

Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                 |
| <i>mask</i> | The interrupt mask; Use the enum _lpspi_interrupt_enable. |

### 25.2.8.16 static void LPSPI\_EnableDMA ( LPSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
* LPSPI_EnableDMA(base, kLPSPI_TxDmaEnable |
 kLPSPI_RxDmaEnable);
*
```

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                           |
| <i>mask</i> | The interrupt mask; Use the enum _lpspi_dma_enable. |

### 25.2.8.17 static void LPSPI\_DisableDMA ( LPSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function configures the Rx and Tx DMA mask of the LPSPI. The parameters are base and a DMA mask.

```
* SPI_DisableDMA(base, kLPSPI_TxDmaEnable |
 kLPSPI_RxDmaEnable);
*
```

Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                           |
| <i>mask</i> | The interrupt mask; Use the enum _lpspi_dma_enable. |

#### 25.2.8.18 static uint32\_t LPSPI\_GetTxRegisterAddress ( LPSPI\_Type \* *base* ) [inline], [static]

This function gets the LPSPI Transmit Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

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

Returns

The LPSPI Transmit Data Register address.

#### 25.2.8.19 static uint32\_t LPSPI\_GetRxRegisterAddress ( LPSPI\_Type \* *base* ) [inline], [static]

This function gets the LPSPI Receive Data Register address because this value is needed for the DMA operation. This function can be used for either master or slave mode.

Parameters

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

Returns

The LPSPI Receive Data Register address.

#### 25.2.8.20 bool LPSPI\_CheckTransferArgument ( lpspi\_transfer\_t \* *transfer*, uint32\_t *bitsPerFrame*, uint32\_t *bytesPerFrame* )

## LPSPI Peripheral driver

Parameters

|                     |                                 |
|---------------------|---------------------------------|
| <i>transfer</i>     | the transfer struct to be used. |
| <i>bitPerFrame</i>  | The bit size of one frame.      |
| <i>bytePerFrame</i> | The byte size of one frame.     |

Returns

Return true for right and false for wrong.

### 25.2.8.21 static void LPSPI\_SetMasterSlaveMode ( LPSPI\_Type \* *base*, lpspi\_master\_slave\_mode\_t *mode* ) [inline], [static]

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                         |
| <i>mode</i> | Mode setting (master or slave) of type lpspi_master_slave_mode_t. |

### 25.2.8.22 static bool LPSPI\_IsMaster ( LPSPI\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

### 25.2.8.23 static void LPSPI\_FlushFifo ( LPSPI\_Type \* *base*, bool *flushTxFifo*, bool *flushRxFifo* ) [inline], [static]

Parameters

|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <i>base</i>        | LPSPI peripheral address.                                          |
| <i>flushTxFifo</i> | Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO. |
| <i>flushRxFifo</i> | Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO. |

#### 25.2.8.24 static void LPSPI\_SetFifoWatermarks ( LPSPI\_Type \* *base*, uint32\_t *txWater*, uint32\_t *rxWater* ) [inline], [static]

This function allows the user to set the receive and transmit FIFO watermarks. The function does not compare the watermark settings to the FIFO size. The FIFO watermark should not be equal to or greater than the FIFO size. It is up to the higher level driver to make this check.

Parameters

|                |                                                                                                |
|----------------|------------------------------------------------------------------------------------------------|
| <i>base</i>    | LPSPI peripheral address.                                                                      |
| <i>txWater</i> | The TX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated. |
| <i>rxWater</i> | The RX FIFO watermark value. Writing a value equal or greater than the FIFO size is truncated. |

#### 25.2.8.25 static void LPSPI\_SetAllPcsPolarity ( LPSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Note that the CFGR1 should only be written when the LPSPI is disabled (LPSPIx\_CR\_MEN = 0).

This is an example: PCS0 and PCS1 set to active low and other PCSs set to active high. Note that the number of PCS is device-specific.

```
* LPSPI_SetAllPcsPolarity(base, kLPSPI_Pcs0ActiveLow |
 kLPSPI_Pcs1ActiveLow);
*
```

Parameters

|             |                                                          |
|-------------|----------------------------------------------------------|
| <i>base</i> | LPSPI peripheral address.                                |
| <i>mask</i> | The PCS polarity mask; Use the enum _lpspi_pcs_polarity. |

#### 25.2.8.26 static void LPSPI\_SetFrameSize ( LPSPI\_Type \* *base*, uint32\_t *frameSize* ) [inline], [static]

The minimum frame size is 8-bits and the maximum frame size is 4096-bits. If the frame size is less than or equal to 32-bits, the word size and frame size are identical. If the frame size is greater than 32-bits, the

## LPSPI Peripheral driver

word size is 32-bits for each word except the last (the last word contains the remainder bits if the frame size is not divisible by 32). The minimum word size is 2-bits. A frame size of 33-bits (or similar) is not supported.

Note 1: The transmit command register should be initialized before enabling the LPSPI in slave mode, although the command register does not update until after the LPSPI is enabled. After it is enabled, the transmit command register should only be changed if the LPSPI is idle.

Note 2: The transmit and command FIFO is a combined FIFO that includes both transmit data and command words. That means the TCR register should be written to when the Tx FIFO is not full.

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>base</i>      | LPSPI peripheral address.         |
| <i>frameSize</i> | The frame size in number of bits. |

### 25.2.8.27 `uint32_t LPSPI_MasterSetBaudRate ( LPSPI_Type * base, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t * tcrPrescaleValue )`

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate and returns the calculated baud rate in bits-per-second. It requires the caller to provide the frequency of the module source clock (in Hertz). Note that the baud rate does not go into effect until the Transmit Control Register (TCR) is programmed with the prescale value. Hence, this function returns the prescale tcrPrescaleValue parameter for later programming in the TCR. The higher level peripheral driver should alert the user of an out of range baud rate input.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

Parameters

|                          |                                                   |
|--------------------------|---------------------------------------------------|
| <i>base</i>              | LPSPI peripheral address.                         |
| <i>baudRate_Bps</i>      | The desired baud rate in bits per second.         |
| <i>srcClock_Hz</i>       | Module source input clock in Hertz.               |
| <i>tcrPrescale-Value</i> | The TCR prescale value needed to program the TCR. |

## Returns

The actual calculated baud rate. This function may also return a "0" if the LPSPI is not configured for master mode or if the LPSPI module is not disabled.

### **25.2.8.28 void LPSPI\_MasterSetDelayScaler ( *LPSPI\_Type* \* *base*, *uint32\_t* *scaler*, *lpspi\_delay\_type\_t* *whichDelay* )**

This function configures the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type *lpspi\_delay\_type\_t*.

The user passes the desired delay along with the delay value. This allows the user to directly set the delay values if they have pre-calculated them or if they simply wish to manually increment the value.

Note that the LPSPI module must first be disabled before configuring this. Note that the LPSPI module must be configured for master mode before configuring this.

## Parameters

|                   |                                                                             |
|-------------------|-----------------------------------------------------------------------------|
| <i>base</i>       | LPSPI peripheral address.                                                   |
| <i>scaler</i>     | The 8-bit delay value 0x00 to 0xFF (255).                                   |
| <i>whichDelay</i> | The desired delay to configure, must be of type <i>lpspi_delay_type_t</i> . |

### **25.2.8.29 *uint32\_t* LPSPI\_MasterSetDelayTimes ( *LPSPI\_Type* \* *base*, *uint32\_t* *delayTimeInNanoSec*, *lpspi\_delay\_type\_t* *whichDelay*, *uint32\_t* *srcClock\_Hz* )**

This function calculates the values for the following: SCK to PCS delay, or PCS to SCK delay, or The configurations must occur between the transfer delay.

The delay names are available in type *lpspi\_delay\_type\_t*.

The user passes the desired delay and the desired delay value in nano-seconds. The function calculates the value needed for the desired delay parameter and returns the actual calculated delay because an exact delay match may not be possible. In this case, the closest match is calculated without going below the desired delay value input. It is possible to input a very large delay value that exceeds the capability of the part, in which case the maximum supported delay is returned. It is up to the higher level peripheral driver to alert the user of an out of range delay input.

Note that the LPSPI module must be configured for master mode before configuring this. And note that the *delayTime* = *LPSPI\_clockSource* / (*PRESCALE* \* *Delay\_scaler*).

## LPSPI Peripheral driver

Parameters

|                            |                                                                               |
|----------------------------|-------------------------------------------------------------------------------|
| <i>base</i>                | LPSPI peripheral address.                                                     |
| <i>delayTimeIn-NanoSec</i> | The desired delay value in nano-seconds.                                      |
| <i>whichDelay</i>          | The desired delay to configuration, which must be of type lpspi_delay_type_t. |
| <i>srcClock_Hz</i>         | Module source input clock in Hertz.                                           |

Returns

actual Calculated delay value in nano-seconds.

### 25.2.8.30 static void LPSPI\_WriteData ( LPSPI\_Type \* *base*, uint32\_t *data* ) [inline], [static]

This function writes data passed in by the user to the Transmit Data Register (TDR). The user can pass up to 32-bits of data to load into the TDR. If the frame size exceeds 32-bits, the user has to manage sending the data one 32-bit word at a time. Any writes to the TDR result in an immediate push to the transmit FIFO. This function can be used for either master or slave modes.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | LPSPI peripheral address. |
| <i>data</i> | The data word to be sent. |

### 25.2.8.31 static uint32\_t LPSPI\_ReadData ( LPSPI\_Type \* *base* ) [inline], [static]

This function reads the data from the Receive Data Register (RDR). This function can be used for either master or slave mode.

Parameters

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

Returns

The data read from the data buffer.

### 25.2.8.32 void LPSPI\_SetDummyData ( LPSPI\_Type \* *base*, uint8\_t *dummyData* )

Parameters

|                  |                                                                                                                                                                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | LPSPI peripheral address.                                                                                                                                                                                                                       |
| <i>dummyData</i> | Data to be transferred when tx buffer is NULL. Note: This API has no effect when LPSPI in slave interrupt mode, because driver will set the TXMSK bit to 1 if txData is NULL, no data is loaded from transmit FIFO and output pin is tristated. |

**25.2.8.33 void LPSPI\_MasterTransferCreateHandle ( **LPSPI\_Type** \* *base*,  
**lpspi\_master\_handle\_t** \* *handle*, **lpspi\_master\_transfer\_callback\_t** *callback*,  
**void** \* *userData* )**

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                              |
| <i>handle</i>   | LPSPI handle pointer to <b>lpspi_master_handle_t</b> . |
| <i>callback</i> | DSPI callback.                                         |
| <i>userData</i> | callback function parameter.                           |

**25.2.8.34 status\_t LPSPI\_MasterTransferBlocking ( **LPSPI\_Type** \* *base*, **lpspi\_transfer\_t** \* *transfer* )**

This function transfers data using a polling method. This is a blocking function, which does not return until all transfers have been completed.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                     |
| <i>transfer</i> | pointer to <b>lpspi_transfer_t</b> structure. |

Returns

status of **status\_t**.

## LPSPI Peripheral driver

### 25.2.8.35 status\_t LPSPI\_MasterTransferNonBlocking ( LPSPI\_Type \* *base*,                   lpspi\_master\_handle\_t \* *handle*, lpspi\_transfer\_t \* *transfer* )

This function transfers data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not integer multiples of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                                   |
| <i>handle</i>   | pointer to lpspi_master_handle_t structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure.                      |

Returns

status of status\_t.

### 25.2.8.36 status\_t LPSPI\_MasterTransferGetCount ( LPSPI\_Type \* *base*,                   lpspi\_master\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the master transfer remaining bytes.

Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                   |
| <i>handle</i> | pointer to lpspi_master_handle_t structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.         |

Returns

status of status\_t.

### 25.2.8.37 void LPSPI\_MasterTransferAbort ( LPSPI\_Type \* *base*, lpspi\_master\_handle\_t                   \* *handle* )

This function aborts a transfer which uses an interrupt method.

Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                   |
| <i>handle</i> | pointer to lpspi_master_handle_t structure which stores the transfer state. |

#### 25.2.8.38 void LPSPI\_MasterTransferHandleIRQ ( LPSPI\_Type \* *base*, lpspi\_master\_handle\_t \* *handle* )

This function processes the LPSPI transmit and receive IRQ.

Parameters

|               |                                                                             |
|---------------|-----------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                   |
| <i>handle</i> | pointer to lpspi_master_handle_t structure which stores the transfer state. |

#### 25.2.8.39 void LPSPI\_SlaveTransferCreateHandle ( LPSPI\_Type \* *base*, lpspi\_slave\_handle\_t \* *handle*, lpspi\_slave\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the LPSPI handle, which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                     |
| <i>handle</i>   | LPSPI handle pointer to lpspi_slave_handle_t. |
| <i>callback</i> | DSPI callback.                                |
| <i>userData</i> | callback function parameter.                  |

#### 25.2.8.40 status\_t LPSPI\_SlaveTransferNonBlocking ( LPSPI\_Type \* *base*, lpspi\_slave\_handle\_t \* *handle*, lpspi\_transfer\_t \* *transfer* )

This function transfer data using an interrupt method. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be integer multiples of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

## LPSPI Peripheral driver

Parameters

|                 |                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral address.                                                               |
| <i>handle</i>   | pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>lpspi_transfer_t</code> structure.                                     |

Returns

status of `status_t`.

### 25.2.8.41 `status_t LPSPI_SlaveTransferGetCount ( LPSPI_Type * base, lpspi_slave_handle_t * handle, size_t * count )`

This function gets the slave transfer remaining bytes.

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                               |
| <i>handle</i> | pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                     |

Returns

status of `status_t`.

### 25.2.8.42 `void LPSPI_SlaveTransferAbort ( LPSPI_Type * base, lpspi_slave_handle_t * handle )`

This function aborts a transfer which uses an interrupt method.

Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                               |
| <i>handle</i> | pointer to <code>lpspi_slave_handle_t</code> structure which stores the transfer state. |

### 25.2.8.43 `void LPSPI_SlaveTransferHandleIRQ ( LPSPI_Type * base, lpspi_slave_handle_t * handle )`

This function processes the LPSPI transmit and receives an IRQ.

Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral address.                                                  |
| <i>handle</i> | pointer to lpspi_slave_handle_t structure which stores the transfer state. |

## 25.2.9 Variable Documentation

### 25.2.9.1 volatile uint8\_t g\_lpspiDummyData[]

### 25.3 LPSPI eDMA Driver

#### 25.3.1 Overview

## Data Structures

- struct [lpspi\\_master\\_edma\\_handle\\_t](#)  
*LPSPI master eDMA transfer handle structure used for transactional API.* [More...](#)
- struct [lpspi\\_slave\\_edma\\_handle\\_t](#)  
*LPSPI slave eDMA transfer handle structure used for transactional API.* [More...](#)

## Typedefs

- typedef void(\* [lpspi\\_master\\_edma\\_transfer\\_callback\\_t](#))(LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*Completion callback function pointer type.*
- typedef void(\* [lpspi\\_slave\\_edma\\_transfer\\_callback\\_t](#))(LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*Completion callback function pointer type.*

## Functions

- void [LPSPI\\_MasterTransferCreateHandleEDMA](#) (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, [lpspi\\_master\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*edmaRxRegToRxDataHandle, [edma\\_handle\\_t](#) \*edmaTxDataToTxRegHandle)  
*Initializes the LPSPI master eDMA handle.*
- status\_t [LPSPI\\_MasterTransferEDMA](#) (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, [lpspi\\_transfer\\_t](#) \*transfer)  
*LPSPI master transfer data using eDMA.*
- void [LPSPI\\_MasterTransferAbortEDMA](#) (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle)  
*LPSPI master aborts a transfer which is using eDMA.*
- status\_t [LPSPI\\_MasterTransferGetCountEDMA](#) (LPSPI\_Type \*base, lpspi\_master\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets the master eDMA transfer remaining bytes.*
- void [LPSPI\\_SlaveTransferCreateHandleEDMA](#) (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, [lpspi\\_slave\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*edmaRxRegToRxDataHandle, [edma\\_handle\\_t](#) \*edmaTxDataToTxRegHandle)  
*Initializes the LPSPI slave eDMA handle.*
- status\_t [LPSPI\\_SlaveTransferEDMA](#) (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, [lpspi\\_transfer\\_t](#) \*transfer)  
*LPSPI slave transfers data using eDMA.*
- void [LPSPI\\_SlaveTransferAbortEDMA](#) (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle)  
*LPSPI slave aborts a transfer which is using eDMA.*
- status\_t [LPSPI\\_SlaveTransferGetCountEDMA](#) (LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, size\_t \*count)

*Gets the slave eDMA transfer remaining bytes.*

## Driver version

- #define **FSL\_LPSPI\_EDMA\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 2))  
*LPSPI EDMA driver version 2.0.2.*

### 25.3.2 Data Structure Documentation

#### 25.3.2.1 struct \_lpspi\_master\_edma\_handle

Forward declaration of the **\_lpspi\_master\_edma\_handle** typedefs.

#### Data Fields

- volatile bool **isPcsContinuous**  
*Is PCS continuous in transfer.*
- volatile bool **isByteSwap**  
*A flag that whether should byte swap.*
- volatile uint8\_t **fifoSize**  
*FIFO dataSize.*
- volatile uint8\_t **rxWatermark**  
*Rx watermark.*
- volatile uint8\_t **bytesEachWrite**  
*Bytes for each write TDR.*
- volatile uint8\_t **bytesEachRead**  
*Bytes for each read RDR.*
- volatile uint8\_t **bytesLastRead**  
*Bytes for last read RDR.*
- volatile uint8\_t **isThereExtraRxBytes**  
*Is there extra RX byte.*
- uint8\_t \*volatile **txData**  
*Send buffer.*
- uint8\_t \*volatile **rxData**  
*Receive buffer.*
- volatile size\_t **txRemainingByteCount**  
*Number of bytes remaining to send.*
- volatile size\_t **rxRemainingByteCount**  
*Number of bytes remaining to receive.*
- volatile uint32\_t **writeRegRemainingTimes**  
*Write TDR register remaining times.*
- volatile uint32\_t **readRegRemainingTimes**  
*Read RDR register remaining times.*
- uint32\_t **totalByteCount**  
*Number of transfer bytes.*
- uint32\_t **txBuffIfNull**  
*Used if there is not txData for DMA purpose.*

## LPSPI eDMA Driver

- `uint32_t rxBuffIfNull`  
*Used if there is not rxData for DMA purpose.*
- `uint32_t transmitCommand`  
*Used to write TCR for DMA purpose.*
- `volatile uint8_t state`  
*LPSPI transfer state , \_lpspi\_transfer\_state.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `lpspi_master_edma_transfer_callback_t callback`  
*Completion callback.*
- `void *userData`  
*Callback user data.*
- `edma_handle_t *edmaRxRegToRxDataHandle`  
*edma\_handle\_t handle point used for RxReg to RxData buff*
- `edma_handle_t *edmaTxDataToTxRegHandle`  
*edma\_handle\_t handle point used for TxData to TxReg buff*
- `edma_tcd_t lpspiSoftwareTCD [3]`  
*SoftwareTCD, internal used.*

### 25.3.2.1.0.79 Field Documentation

- 25.3.2.1.0.79.1 `volatile bool lpspi_master_edma_handle_t::isPcsContinuous`
- 25.3.2.1.0.79.2 `volatile bool lpspi_master_edma_handle_t::isByteSwap`
- 25.3.2.1.0.79.3 `volatile uint8_t lpspi_master_edma_handle_t::fifoSize`
- 25.3.2.1.0.79.4 `volatile uint8_t lpspi_master_edma_handle_t::rxWatermark`
- 25.3.2.1.0.79.5 `volatile uint8_t lpspi_master_edma_handle_t::bytesEachWrite`
- 25.3.2.1.0.79.6 `volatile uint8_t lpspi_master_edma_handle_t::bytesEachRead`
- 25.3.2.1.0.79.7 `volatile uint8_t lpspi_master_edma_handle_t::bytesLastRead`
- 25.3.2.1.0.79.8 `volatile uint8_t lpspi_master_edma_handle_t::isThereExtraRxBytes`
- 25.3.2.1.0.79.9 `uint8_t* volatile lpspi_master_edma_handle_t::txData`
- 25.3.2.1.0.79.10 `uint8_t* volatile lpspi_master_edma_handle_t::rxData`
- 25.3.2.1.0.79.11 `volatile size_t lpspi_master_edma_handle_t::txRemainingByteCount`
- 25.3.2.1.0.79.12 `volatile size_t lpspi_master_edma_handle_t::rxRemainingByteCount`
- 25.3.2.1.0.79.13 `volatile uint32_t lpspi_master_edma_handle_t::writeRegRemainingTimes`
- 25.3.2.1.0.79.14 `volatile uint32_t lpspi_master_edma_handle_t::readRegRemainingTimes`
- 25.3.2.1.0.79.15 `uint32_t lpspi_master_edma_handle_t::txBuffIfNull`
- 25.3.2.1.0.79.16 `uint32_t lpspi_master_edma_handle_t::rxBuffIfNull`
- 25.3.2.1.0.79.17 `uint32_t lpspi_master_edma_handle_t::transmitCommand`
- 25.3.2.1.0.79.18 `volatile uint8_t lpspi_master_edma_handle_t::state`
- 25.3.2.1.0.79.19 `uint8_t lpspi_master_edma_handle_t::nbytes`
- 25.3.2.1.0.79.20 `lpspi_master_edma_transfer_callback_t lpspi_master_edma_handle_t::callback`
- 25.3.2.1.0.79.21 `void* lpspi_master_edma_handle_t::userData`

### 25.3.2.2 `struct _lpspi_slave_edma_handle`

Forward declaration of the `_lpspi_slave_edma_handle` typedefs.

### Data Fields

- volatile bool **isByteSwap**  
*A flag that whether should byte swap.*
- volatile uint8\_t **fifoSize**  
*FIFO dataSize.*
- volatile uint8\_t **rxWatermark**  
*Rx watermark.*
- volatile uint8\_t **bytesEachWrite**  
*Bytes for each write TDR.*
- volatile uint8\_t **bytesEachRead**  
*Bytes for each read RDR.*
- volatile uint8\_t **bytesLastRead**  
*Bytes for last read RDR.*
- volatile uint8\_t **isThereExtraRxBytes**  
*Is there extra RX byte.*
- uint8\_t **nbytes**  
*eDMA minor byte transfer count initially configured.*
- uint8\_t \*volatile **txData**  
*Send buffer.*
- uint8\_t \*volatile **rxData**  
*Receive buffer.*
- volatile size\_t **txRemainingByteCount**  
*Number of bytes remaining to send.*
- volatile size\_t **rxRemainingByteCount**  
*Number of bytes remaining to receive.*
- volatile uint32\_t **writeRegRemainingTimes**  
*Write TDR register remaining times.*
- volatile uint32\_t **readRegRemainingTimes**  
*Read RDR register remaining times.*
- uint32\_t **totalByteCount**  
*Number of transfer bytes.*
- uint32\_t **txBuffIfNull**  
*Used if there is not txData for DMA purpose.*
- uint32\_t **rxBuffIfNull**  
*Used if there is not rxData for DMA purpose.*
- volatile uint8\_t **state**  
*LPSPI transfer state.*
- uint32\_t **errorCount**  
*Error count for slave transfer.*
- **lpspi\_slave\_edma\_transfer\_callback\_t callback**  
*Completion callback.*
- void \* **userData**  
*Callback user data.*
- edma\_handle\_t \* **edmaRxRegToRxDataHandle**  
*edma\_handle\_t handle point used for RxReg to RxData buff*
- edma\_handle\_t \* **edmaTxDataToTxRegHandle**  
*edma\_handle\_t handle point used for TxData to TxReg*
- edma\_tcd\_t **lpspiSoftwareTCD** [2]  
*SoftwareTCD, internal used.*



## LPSPI eDMA Driver

### 25.3.2.2.0.80 Field Documentation

- 25.3.2.2.0.80.1 `volatile bool lpspi_slave_edma_handle_t::isByteSwap`
- 25.3.2.2.0.80.2 `volatile uint8_t lpspi_slave_edma_handle_t::fifoSize`
- 25.3.2.2.0.80.3 `volatile uint8_t lpspi_slave_edma_handle_t::rxWatermark`
- 25.3.2.2.0.80.4 `volatile uint8_t lpspi_slave_edma_handle_t::bytesEachWrite`
- 25.3.2.2.0.80.5 `volatile uint8_t lpspi_slave_edma_handle_t::bytesEachRead`
- 25.3.2.2.0.80.6 `volatile uint8_t lpspi_slave_edma_handle_t::bytesLastRead`
- 25.3.2.2.0.80.7 `volatile uint8_t lpspi_slave_edma_handle_t::isThereExtraRxBytes`
- 25.3.2.2.0.80.8 `uint8_t lpspi_slave_edma_handle_t::nbytes`
- 25.3.2.2.0.80.9 `uint8_t* volatile lpspi_slave_edma_handle_t::txData`
- 25.3.2.2.0.80.10 `uint8_t* volatile lpspi_slave_edma_handle_t::rxData`
- 25.3.2.2.0.80.11 `volatile size_t lpspi_slave_edma_handle_t::txRemainingByteCount`
- 25.3.2.2.0.80.12 `volatile size_t lpspi_slave_edma_handle_t::rxRemainingByteCount`
- 25.3.2.2.0.80.13 `volatile uint32_t lpspi_slave_edma_handle_t::writeRegRemainingTimes`
- 25.3.2.2.0.80.14 `volatile uint32_t lpspi_slave_edma_handle_t::readRegRemainingTimes`
- 25.3.2.2.0.80.15 `uint32_t lpspi_slave_edma_handle_t::txBuffIfNull`
- 25.3.2.2.0.80.16 `uint32_t lpspi_slave_edma_handle_t::rxBuffIfNull`
- 25.3.2.2.0.80.17 `volatile uint8_t lpspi_slave_edma_handle_t::state`
- 25.3.2.2.0.80.18 `uint32_t lpspi_slave_edma_handle_t::errorCount`
- 25.3.2.2.0.80.19 `lpspi_slave_edma_transfer_callback_t lpspi_slave_edma_handle_t::callback`
- 25.3.2.2.0.80.20 `void* lpspi_slave_edma_handle_t::userData`

### 25.3.3 Macro Definition Documentation

- 25.3.3.1 `#define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

### 25.3.4 Typedef Documentation

- 25.3.4.1 `typedef void(* lpspi_master_edma_transfer_callback_t)(LPSPI_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)`

Parameters

|                 |                                                                  |
|-----------------|------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                   |
| <i>handle</i>   | Pointer to the handle for the LPSPI master.                      |
| <i>status</i>   | Success or error code describing whether the transfer completed. |
| <i>userData</i> | Arbitrary pointer-dataSized value passed from the application.   |

#### 25.3.4.2 **typedef void(\* lpspi\_slave\_edma\_transfer\_callback\_t)(LPSPI\_Type \*base, lpspi\_slave\_edma\_handle\_t \*handle, status\_t status, void \*userData)**

Parameters

|                 |                                                                  |
|-----------------|------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                   |
| <i>handle</i>   | Pointer to the handle for the LPSPI slave.                       |
| <i>status</i>   | Success or error code describing whether the transfer completed. |
| <i>userData</i> | Arbitrary pointer-dataSized value passed from the application.   |

### 25.3.5 Function Documentation

#### 25.3.5.1 **void LPSPI\_MasterTransferCreateHandleEDMA ( LPSPI\_Type \* *base*, lpspi\_master\_edma\_handle\_t \* *handle*, lpspi\_master\_edma\_transfer\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *edmaRxRegToRxDataHandle*, edma\_handle\_t \* *edmaTxDataToTxRegHandle* )**

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that the LPSPI eDMA has a separated (Rx and Rx as two sources) or shared (Rx and Tx are the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaIntermediaryTo-TxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle.

Parameters

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

## LPSPI eDMA Driver

|                                  |                                                                   |
|----------------------------------|-------------------------------------------------------------------|
| <i>handle</i>                    | LPSPI handle pointer to <code>lpspi_master_edma_handle_t</code> . |
| <i>callback</i>                  | LPSPI callback.                                                   |
| <i>userData</i>                  | callback function parameter.                                      |
| <i>edmaRxRegTo-RxDataHandle</i>  | edmaRxRegToRxDataHandle pointer to <code>edma_handle_t</code> .   |
| <i>edmaTxData-ToTxReg-Handle</i> | edmaTxDataToTxRegHandle pointer to <code>edma_handle_t</code> .   |

### 25.3.5.2 `status_t LPSPI_MasterTransferEDMA ( LPSPI_Type * base, lpspi_master_edma_handle_t * handle, lpspi_transfer_t * transfer )`

This function transfers data using eDMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                                                |
| <i>handle</i>   | pointer to <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <code>lpspi_transfer_t</code> structure.                                           |

Returns

status of `status_t`.

### 25.3.5.3 `void LPSPI_MasterTransferAbortEDMA ( LPSPI_Type * base, lpspi_master_edma_handle_t * handle )`

This function aborts a transfer which is using eDMA.

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                                   |
| <i>handle</i> | pointer to lpspi_master_edma_handle_t structure which stores the transfer state. |

#### 25.3.5.4 **status\_t LPSPI\_MasterTransferGetCountEDMA ( LPSPI\_Type \* *base*, lpspi\_master\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the master eDMA transfer remaining bytes.

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                                   |
| <i>handle</i> | pointer to lpspi_master_edma_handle_t structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the EDMA transaction.                      |

Returns

status of status\_t.

#### 25.3.5.5 **void LPSPI\_SlaveTransferCreateHandleEDMA ( LPSPI\_Type \* *base*, lpspi\_slave\_edma\_handle\_t \* *handle*, lpspi\_slave\_edma\_transfer\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *edmaRxRegToRxDataHandle*, edma\_handle\_t \* *edmaTxDataToTxRegHandle* )**

This function initializes the LPSPI eDMA handle which can be used for other LPSPI transactional APIs. Usually, for a specified LPSPI instance, call this API once to get the initialized handle.

Note that LPSPI eDMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx as the same source) DMA request source.

(1) For a separated DMA request source, enable and set the Rx DMAMUX source for edmaRxRegToRxDataHandle and Tx DMAMUX source for edmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for edmaRxRegToRxDataHandle .

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                     |
| <i>handle</i> | LPSPI handle pointer to lpspi_slave_edma_handle_t. |

## LPSPI eDMA Driver

|                                  |                                                                    |
|----------------------------------|--------------------------------------------------------------------|
| <i>callback</i>                  | LPSPI callback.                                                    |
| <i>userData</i>                  | callback function parameter.                                       |
| <i>edmaRxRegTo-RxDataHandle</i>  | edmaRxRegToRxDataHandle pointer to <a href="#">edma_handle_t</a> . |
| <i>edmaTxData-ToTxReg-Handle</i> | edmaTxDataToTxRegHandle pointer to <a href="#">edma_handle_t</a> . |

### 25.3.5.6 **status\_t LPSPI\_SlaveTransferEDMA ( LPSPI\_Type \* *base*, lpspi\_slave\_edma\_handle\_t \* *handle*, lpspi\_transfer\_t \* *transfer* )**

This function transfers data using eDMA. This is a non-blocking function, which return right away. When all data is transferred, the callback function is called.

Note: The transfer data size should be an integer multiple of bytesPerFrame if bytesPerFrame is less than or equal to 4. For bytesPerFrame greater than 4: The transfer data size should be equal to bytesPerFrame if the bytesPerFrame is not an integer multiple of 4. Otherwise, the transfer data size can be an integer multiple of bytesPerFrame.

Parameters

|                 |                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------|
| <i>base</i>     | LPSPI peripheral base address.                                                                  |
| <i>handle</i>   | pointer to <a href="#">lpspi_slave_edma_handle_t</a> structure which stores the transfer state. |
| <i>transfer</i> | pointer to <a href="#">lpspi_transfer_t</a> structure.                                          |

Returns

status of status\_t.

### 25.3.5.7 **void LPSPI\_SlaveTransferAbortEDMA ( LPSPI\_Type \* *base*, lpspi\_slave\_edma\_handle\_t \* *handle* )**

This function aborts a transfer which is using eDMA.

Parameters

|               |                                                                                 |
|---------------|---------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                                  |
| <i>handle</i> | pointer to lpspi_slave_edma_handle_t structure which stores the transfer state. |

### 25.3.5.8 **status\_t LPSPI\_SlaveTransferGetCountEDMA ( LPSPI\_Type \* *base*, lpspi\_slave\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

This function gets the slave eDMA transfer remaining bytes.

Parameters

|               |                                                                                 |
|---------------|---------------------------------------------------------------------------------|
| <i>base</i>   | LPSPI peripheral base address.                                                  |
| <i>handle</i> | pointer to lpspi_slave_edma_handle_t structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the eDMA transaction.                     |

Returns

status of status\_t.

### 25.4 LPSPI FreeRTOS Driver

#### 25.4.1 Overview

##### Driver version

- #define `FSL_LPSPI_FREERTOS_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))  
*LPSPI freertos driver version 2.0.2.*

##### LPSPI RTOS Operation

- status\_t `LPSPI_RTOS_Init` (lpspi\_rtos\_handle\_t \*handle, LPSPI\_Type \*base, const lpspi\_master\_config\_t \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes LPSPI.*
- status\_t `LPSPI_RTOS_Deinit` (lpspi\_rtos\_handle\_t \*handle)  
*Deinitializes the LPSPI.*
- status\_t `LPSPI_RTOS_Transfer` (lpspi\_rtos\_handle\_t \*handle, lpspi\_transfer\_t \*transfer)  
*Performs SPI transfer.*

#### 25.4.2 Macro Definition Documentation

##### 25.4.2.1 #define `FSL_LPSPI_FREERTOS_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))

#### 25.4.3 Function Documentation

##### 25.4.3.1 status\_t `LPSPI_RTOS_Init` ( lpspi\_rtos\_handle\_t \* *handle*, LPSPI\_Type \* *base*, const lpspi\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )

This function initializes the LPSPI module and related RTOS context.

Parameters

|                     |                                                                            |
|---------------------|----------------------------------------------------------------------------|
| <i>handle</i>       | The RTOS LPSPI handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the LPSPI instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up LPSPI in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the LPSPI module.                              |

Returns

status of the operation.

### 25.4.3.2 status\_t LPSPI\_RTOS\_Deinit ( *lpspi\_rtos\_handle\_t \* handle* )

This function deinitializes the LPSPI module and related RTOS context.

## LPSPI FreeRTOS Driver

Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPSPI handle. |
|---------------|------------------------|

### 25.4.3.3 **status\_t LPSPI\_RTOS\_Transfer ( *lpspi\_rtos\_handle\_t \* handle*, *lpspi\_transfer\_t \* transfer* )**

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS LPSPI handle.                        |
| <i>transfer</i> | Structure specifying the transfer parameters. |

Returns

status of the operation.

# Chapter 26

## LPUART: Low Power UART Driver

### 26.1 Overview

#### Modules

- LPUART DMA Driver
- LPUART Driver
- LPUART FreeRTOS Driver
- LPUART eDMA Driver

### 26.2 LPUART Driver

#### 26.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

#### 26.2.2 Typical use case

##### 26.2.2.1 LPUART Operation

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

## Data Structures

- struct [lpuart\\_config\\_t](#)  
*LPUART configuration structure.* [More...](#)
- struct [lpuart\\_transfer\\_t](#)  
*LPUART transfer structure.* [More...](#)
- struct [lpuart\\_handle\\_t](#)  
*LPUART handle structure.* [More...](#)

## Typedefs

- typedef void(\* [lpuart\\_transfer\\_callback\\_t](#) )(LPUART\_Type \*base, lpuart\_handle\_t \*handle, status\_t status, void \*userData)  
*LPUART transfer callback function.*

## Enumerations

- enum `_lpuart_status` {
   
kStatus\_LPUART\_TxBusy = MAKE\_STATUS(kStatusGroup\_LPUART, 0),
   
kStatus\_LPUART\_RxBusy = MAKE\_STATUS(kStatusGroup\_LPUART, 1),
   
kStatus\_LPUART\_TxIdle = MAKE\_STATUS(kStatusGroup\_LPUART, 2),
   
kStatus\_LPUART\_RxIdle = MAKE\_STATUS(kStatusGroup\_LPUART, 3),
   
kStatus\_LPUART\_TxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_LPUART, 4),
   
kStatus\_LPUART\_RxWatermarkTooLarge = MAKE\_STATUS(kStatusGroup\_LPUART, 5),
   
kStatus\_LPUART\_FlagCannotClearManually = MAKE\_STATUS(kStatusGroup\_LPUART, 6),
   
kStatus\_LPUART\_Error = MAKE\_STATUS(kStatusGroup\_LPUART, 7),
   
kStatus\_LPUART\_RxRingBufferOverrun,
   
kStatus\_LPUART\_RxHardwareOverrun = MAKE\_STATUS(kStatusGroup\_LPUART, 9),
   
kStatus\_LPUART\_NoiseError = MAKE\_STATUS(kStatusGroup\_LPUART, 10),
   
kStatus\_LPUART\_FramingError = MAKE\_STATUS(kStatusGroup\_LPUART, 11),
   
kStatus\_LPUART\_ParityError = MAKE\_STATUS(kStatusGroup\_LPUART, 12),
   
kStatus\_LPUART\_BaudrateNotSupport,
   
kStatus\_LPUART\_IdleLineDetected = MAKE\_STATUS(kStatusGroup\_LPUART, 14) }

*Error codes for the LPUART driver.*

- enum `lpuart_parity_mode_t` {
   
kLPUART\_ParityDisabled = 0x0U,
   
kLPUART\_ParityEven = 0x2U,
   
kLPUART\_ParityOdd = 0x3U }
- enum `lpuart_data_bits_t` {
   
kLPUART\_EightDataBits = 0x0U,
   
kLPUART\_SevenDataBits = 0x1U }
- enum `lpuart_stop_bit_count_t` {
   
kLPUART\_OneStopBit = 0U,
   
kLPUART\_TwoStopBit = 1U }
- enum `lpuart_transmit_cts_source_t` {
   
kLPUART\_CtsSourcePin = 0U,
   
kLPUART\_CtsSourceMatchResult = 1U }
- enum `lpuart_transmit_cts_config_t` {
   
kLPUART\_CtsSampleAtStart = 0U,
   
kLPUART\_CtsSampleAtIdle = 1U }
- enum `lpuart_idle_type_select_t` {
   
kLPUART\_IdleTypeStartBit = 0U,
   
kLPUART\_IdleTypeStopBit = 1U }
- enum `lpuart_idle_config_t` {

## LPUART Driver

```
kLPUART_IdleCharacter1 = 0U,
kLPUART_IdleCharacter2 = 1U,
kLPUART_IdleCharacter4 = 2U,
kLPUART_IdleCharacter8 = 3U,
kLPUART_IdleCharacter16 = 4U,
kLPUART_IdleCharacter32 = 5U,
kLPUART_IdleCharacter64 = 6U,
kLPUART_IdleCharacter128 = 7U }
```

*LPUART idle detected configuration.*

- enum \_lpuart\_interrupt\_enable {  
kLPUART\_LinBreakInterruptEnable = (LPUART\_BAUD\_LBKDIIE\_MASK >> 8),  
kLPUART\_RxActiveEdgeInterruptEnable = (LPUART\_BAUD\_RXEDGIE\_MASK >> 8),  
kLPUART\_TxDataRegEmptyInterruptEnable = (LPUART\_CTRL\_TIE\_MASK),  
kLPUART\_TransmissionCompleteInterruptEnable = (LPUART\_CTRL\_TCIE\_MASK),  
kLPUART\_RxDataRegFullInterruptEnable = (LPUART\_CTRL\_RIE\_MASK),  
kLPUART\_IdleLineInterruptEnable = (LPUART\_CTRL\_ILIE\_MASK),  
kLPUART\_RxOverrunInterruptEnable = (LPUART\_CTRL\_ORIE\_MASK),  
kLPUART\_NoiseErrorInterruptEnable = (LPUART\_CTRL\_NEIE\_MASK),  
kLPUART\_FramingErrorInterruptEnable = (LPUART\_CTRL\_FEIE\_MASK),  
kLPUART\_ParityErrorInterruptEnable = (LPUART\_CTRL\_PEIE\_MASK),  
kLPUART\_TxFifoOverflowInterruptEnable = (LPUART\_FIFO\_TXOFE\_MASK >> 8),  
kLPUART\_RxFifoUnderflowInterruptEnable = (LPUART\_FIFO\_RXUFE\_MASK >> 8) }

*LPUART interrupt configuration structure, default settings all disabled.*

- enum \_lpuart\_flags {  
kLPUART\_TxDataRegEmptyFlag,  
kLPUART\_TransmissionCompleteFlag,  
kLPUART\_RxDataRegFullFlag,  
kLPUART\_IdleLineFlag = (LPUART\_STAT\_IDLE\_MASK),  
kLPUART\_RxOverrunFlag = (LPUART\_STAT\_OR\_MASK),  
kLPUART\_NoiseErrorFlag = (LPUART\_STAT\_NF\_MASK),  
kLPUART\_FramingErrorFlag,  
kLPUART\_ParityErrorFlag = (LPUART\_STAT\_PF\_MASK),  
kLPUART\_LinBreakFlag = (int)(LPUART\_STAT\_LBKDIF\_MASK),  
kLPUART\_RxActiveEdgeFlag,  
kLPUART\_RxActiveFlag,  
kLPUART\_DataMatch1Flag = LPUART\_STAT\_MA1F\_MASK,  
kLPUART\_DataMatch2Flag = LPUART\_STAT\_MA2F\_MASK,  
kLPUART\_NoiseErrorInRxDataRegFlag,  
kLPUART\_ParityErrorInRxDataRegFlag,  
kLPUART\_TxFifoEmptyFlag = (LPUART\_FIFO\_TXEMPT\_MASK >> 16),  
kLPUART\_RxFifoEmptyFlag = (LPUART\_FIFO\_RXEMPT\_MASK >> 16),  
kLPUART\_TxFifoOverflowFlag,  
kLPUART\_RxFifoUnderflowFlag }

*LPUART status flags.*

## Driver version

- #define **FSL\_LPUART\_DRIVER\_VERSION** (MAKE\_VERSION(2, 2, 7))  
*LPUART driver version 2.2.7.*

## Software Reset

- static void **LPUART\_SoftwareReset** (LPUART\_Type \*base)  
*Resets the LPUART using software.*

## Initialization and deinitialization

- status\_t **LPUART\_Init** (LPUART\_Type \*base, const **lpuart\_config\_t** \*config, uint32\_t srcClock\_Hz)  
*Initializes an LPUART instance with the user configuration structure and the peripheral clock.*
- void **LPUART\_Deinit** (LPUART\_Type \*base)  
*Deinitializes a LPUART instance.*
- void **LPUART\_GetDefaultConfig** (**lpuart\_config\_t** \*config)  
*Gets the default configuration structure.*
- status\_t **LPUART\_SetBaudRate** (LPUART\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the LPUART instance baudrate.*

## Status

- uint32\_t **LPUART\_GetStatusFlags** (LPUART\_Type \*base)  
*Gets LPUART status flags.*
- status\_t **LPUART\_ClearStatusFlags** (LPUART\_Type \*base, uint32\_t mask)  
*Clears status flags with a provided mask.*

## Interrupts

- void **LPUART\_EnableInterrupts** (LPUART\_Type \*base, uint32\_t mask)  
*Enables LPUART interrupts according to a provided mask.*
- void **LPUART\_DisableInterrupts** (LPUART\_Type \*base, uint32\_t mask)  
*Disables LPUART interrupts according to a provided mask.*
- uint32\_t **LPUART\_GetEnabledInterrupts** (LPUART\_Type \*base)  
*Gets enabled LPUART interrupts.*
- static uint32\_t **LPUART\_GetDataRegisterAddress** (LPUART\_Type \*base)  
*Gets the LPUART data register address.*
- static void **LPUART\_EnableTxDMA** (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART transmitter DMA request.*
- static void **LPUART\_EnableRxDMA** (LPUART\_Type \*base, bool enable)  
*Enables or disables the LPUART receiver DMA.*

### Bus Operations

- `uint32_t LPUART_GetInstance (LPUART_Type *base)`  
*Get the LPUART instance from peripheral base address.*
- `static void LPUART_EnableTx (LPUART_Type *base, bool enable)`  
*Enables or disables the LPUART transmitter.*
- `static void LPUART_EnableRx (LPUART_Type *base, bool enable)`  
*Enables or disables the LPUART receiver.*
- `static void LPUART_WriteByte (LPUART_Type *base, uint8_t data)`  
*Writes to the transmitter register.*
- `static uint8_t LPUART_ReadByte (LPUART_Type *base)`  
*Reads the receiver register.*
- `void LPUART_WriteBlocking (LPUART_Type *base, const uint8_t *data, size_t length)`  
*Writes to the transmitter register using a blocking method.*
- `status_t LPUART_ReadBlocking (LPUART_Type *base, uint8_t *data, size_t length)`  
*Reads the receiver data register using a blocking method.*

### Transactional

- `void LPUART_TransferCreateHandle (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_callback_t callback, void *userData)`  
*Initializes the LPUART handle.*
- `status_t LPUART_TransferSendNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer)`  
*Transmits a buffer of data using the interrupt method.*
- `void LPUART_TransferStartRingBuffer (LPUART_Type *base, lpuart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)`  
*Sets up the RX ring buffer.*
- `void LPUART_TransferStopRingBuffer (LPUART_Type *base, lpuart_handle_t *handle)`  
*Aborts the background transfer and uninstalls the ring buffer.*
- `size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type *base, lpuart_handle_t *handle)`  
*Get the length of received data in RX ring buffer.*
- `void LPUART_TransferAbortSend (LPUART_Type *base, lpuart_handle_t *handle)`  
*Aborts the interrupt-driven data transmit.*
- `status_t LPUART_TransferGetSendCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`  
*Gets the number of bytes that have been written to the LPUART transmitter register.*
- `status_t LPUART_TransferReceiveNonBlocking (LPUART_Type *base, lpuart_handle_t *handle, lpuart_transfer_t *xfer, size_t *receivedBytes)`  
*Receives a buffer of data using the interrupt method.*
- `void LPUART_TransferAbortReceive (LPUART_Type *base, lpuart_handle_t *handle)`  
*Aborts the interrupt-driven data receiving.*
- `status_t LPUART_TransferGetReceiveCount (LPUART_Type *base, lpuart_handle_t *handle, uint32_t *count)`  
*Gets the number of bytes that have been received.*
- `void LPUART_TransferHandleIRQ (LPUART_Type *base, lpuart_handle_t *handle)`  
*LPUART IRQ handle function.*
- `void LPUART_TransferHandleErrorIRQ (LPUART_Type *base, lpuart_handle_t *handle)`

*LPUART Error IRQ handle function.*

## 26.2.3 Data Structure Documentation

### 26.2.3.1 struct lpuart\_config\_t

#### Data Fields

- `uint32_t baudRate_Bps`  
*LPUART baud rate.*
- `lpuart_parity_mode_t parityMode`  
*Parity mode, disabled (default), even, odd.*
- `lpuart_data_bits_t dataBitsCount`  
*Data bits count, eight (default), seven.*
- `bool isMsb`  
*Data bits order, LSB (default), MSB.*
- `lpuart_stop_bit_count_t stopBitCount`  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- `uint8_t txFifoWatermark`  
*TX FIFO watermark.*
- `uint8_t rxFifoWatermark`  
*RX FIFO watermark.*
- `bool enableRxRTS`  
*RX RTS enable.*
- `bool enableTxCTS`  
*TX CTS enable.*
- `lpuart_transmit_cts_source_t txCtsSource`  
*TX CTS source.*
- `lpuart_transmit_cts_config_t txCtsConfig`  
*TX CTS configure.*
- `lpuart_idle_type_select_t rxIdleType`  
*RX IDLE type.*
- `lpuart_idle_config_t rxIdleConfig`  
*RX IDLE configuration.*
- `bool enableTx`  
*Enable TX.*
- `bool enableRx`  
*Enable RX.*

## LPUART Driver

### 26.2.3.1.0.81 Field Documentation

#### 26.2.3.1.0.81.1 `lpuart_idle_type_select_t lpuart_config_t::rxIdleType`

#### 26.2.3.1.0.81.2 `lpuart_idle_config_t lpuart_config_t::rxIdleConfig`

### 26.2.3.2 `struct lpuart_transfer_t`

#### Data Fields

- `uint8_t * data`  
*The buffer of data to be transfer.*
- `size_t dataSize`  
*The byte count to be transfer.*

### 26.2.3.2.0.82 Field Documentation

#### 26.2.3.2.0.82.1 `uint8_t* lpuart_transfer_t::data`

#### 26.2.3.2.0.82.2 `size_t lpuart_transfer_t::dataSize`

### 26.2.3.3 `struct _lpuart_handle`

#### Data Fields

- `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.*
- `lpuart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*LPUART callback function parameter.*
- `volatile uint8_t txState`  
*TX transfer state.*

- volatile uint8\_t **rxState**  
*RX transfer state.*
- bool **isSevenDataBits**  
*Seven data bits flag.*

## LPUART Driver

### 26.2.3.3.0.83 Field Documentation

- 26.2.3.3.0.83.1 `uint8_t* volatile Ipuart_handle_t::txData`
- 26.2.3.3.0.83.2 `volatile size_t Ipuart_handle_t::txDataSize`
- 26.2.3.3.0.83.3 `size_t Ipuart_handle_t::txDataSizeAll`
- 26.2.3.3.0.83.4 `uint8_t* volatile Ipuart_handle_t::rxData`
- 26.2.3.3.0.83.5 `volatile size_t Ipuart_handle_t::rxDataSize`
- 26.2.3.3.0.83.6 `size_t Ipuart_handle_t::rxDataSizeAll`
- 26.2.3.3.0.83.7 `uint8_t* Ipuart_handle_t::rxRingBuffer`
- 26.2.3.3.0.83.8 `size_t Ipuart_handle_t::rxRingBufferSize`
- 26.2.3.3.0.83.9 `volatile uint16_t Ipuart_handle_t::rxRingBufferHead`
- 26.2.3.3.0.83.10 `volatile uint16_t Ipuart_handle_t::rxRingBufferTail`
- 26.2.3.3.0.83.11 `Ipuart_transfer_callback_t Ipuart_handle_t::callback`
- 26.2.3.3.0.83.12 `void* Ipuart_handle_t::userData`
- 26.2.3.3.0.83.13 `volatile uint8_t Ipuart_handle_t::txState`
- 26.2.3.3.0.83.14 `volatile uint8_t Ipuart_handle_t::rxState`
- 26.2.3.3.0.83.15 `bool Ipuart_handle_t::isSevenDataBits`

### 26.2.4 Macro Definition Documentation

- 26.2.4.1 `#define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 2, 7))`

### 26.2.5 Typedef Documentation

- 26.2.5.1 `typedef void(* Ipuart_transfer_callback_t)(LPUART_Type *base, Ipuart_handle_t *handle, status_t status, void *userData)`

### 26.2.6 Enumeration Type Documentation

#### 26.2.6.1 enum \_Ipuart\_status

Enumerator

*kStatus\_LPUART\_TxBusy* TX busy.

*kStatus\_LPUART\_RxBusy* RX busy.  
*kStatus\_LPUART\_TxIdle* LPUART transmitter is idle.  
*kStatus\_LPUART\_RxIdle* LPUART receiver is idle.  
*kStatus\_LPUART\_TxWatermarkTooLarge* TX FIFO watermark too large.  
*kStatus\_LPUART\_RxWatermarkTooLarge* RX FIFO watermark too large.  
*kStatus\_LPUART\_FlagCannotClearManually* Some flag can't manually clear.  
*kStatus\_LPUART\_Error* Error happens on LPUART.  
*kStatus\_LPUART\_RxRingBufferOverrun* LPUART RX software ring buffer overrun.  
*kStatus\_LPUART\_RxHardwareOverrun* LPUART RX receiver overrun.  
*kStatus\_LPUART\_NoiseError* LPUART noise error.  
*kStatus\_LPUART\_FramingError* LPUART framing error.  
*kStatus\_LPUART\_ParityError* LPUART parity error.  
*kStatus\_LPUART\_BaudrateNotSupport* Baudrate is not support in current clock source.  
*kStatus\_LPUART\_IdleLineDetected* IDLE flag.

### 26.2.6.2 enum lpuart\_parity\_mode\_t

Enumerator

*kLPUART\_ParityDisabled* Parity disabled.  
*kLPUART\_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.  
*kLPUART\_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

### 26.2.6.3 enum lpuart\_data\_bits\_t

Enumerator

*kLPUART\_EightDataBits* Eight data bit.  
*kLPUART\_SevenDataBits* Seven data bit.

### 26.2.6.4 enum lpuart\_stop\_bit\_count\_t

Enumerator

*kLPUART\_OneStopBit* One stop bit.  
*kLPUART\_TwoStopBit* Two stop bits.

### 26.2.6.5 enum lpuart\_transmit\_cts\_source\_t

Enumerator

*kLPUART\_CtsSourcePin* CTS resource is the LPUART\_CTS pin.  
*kLPUART\_CtsSourceMatchResult* CTS resource is the match result.

### 26.2.6.6 enum lpuart\_transmit\_cts\_config\_t

Enumerator

*kLPUART\_CtsSampleAtStart* CTS input is sampled at the start of each character.

*kLPUART\_CtsSampleAtIdle* CTS input is sampled when the transmitter is idle.

### 26.2.6.7 enum lpuart\_idle\_type\_select\_t

Enumerator

*kLPUART\_IdleTypeStartBit* Start counting after a valid start bit.

*kLPUART\_IdleTypeStopBit* Start counting after a stop bit.

### 26.2.6.8 enum lpuart\_idle\_config\_t

This structure defines the number of idle characters that must be received before the IDLE flag is set.

Enumerator

*kLPUART\_IdleCharacter1* the number of idle characters.

*kLPUART\_IdleCharacter2* the number of idle characters.

*kLPUART\_IdleCharacter4* the number of idle characters.

*kLPUART\_IdleCharacter8* the number of idle characters.

*kLPUART\_IdleCharacter16* the number of idle characters.

*kLPUART\_IdleCharacter32* the number of idle characters.

*kLPUART\_IdleCharacter64* the number of idle characters.

*kLPUART\_IdleCharacter128* the number of idle characters.

### 26.2.6.9 enum \_lpuart\_interrupt\_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

*kLPUART\_LinBreakInterruptEnable* LIN break detect.

*kLPUART\_RxActiveEdgeInterruptEnable* Receive Active Edge.

*kLPUART\_TxDataRegEmptyInterruptEnable* Transmit data register empty.

*kLPUART\_TransmissionCompleteInterruptEnable* Transmission complete.

*kLPUART\_RxDataRegFullInterruptEnable* Receiver data register full.

*kLPUART\_IdleLineInterruptEnable* Idle line.

*kLPUART\_RxOverrunInterruptEnable* Receiver Overrun.

*kLPUART\_NoiseErrorInterruptEnable* Noise error flag.

***kLPUART\_FramingErrorInterruptEnable*** Framing error flag.

***kLPUART\_ParityErrorInterruptEnable*** Parity error flag.

***kLPUART\_TxFifoOverflowInterruptEnable*** Transmit FIFO Overflow.

***kLPUART\_RxFifoUnderflowInterruptEnable*** Receive FIFO Underflow.

### 26.2.6.10 enum \_lpuart\_flags

This provides constants for the LPUART status flags for use in the LPUART functions.

Enumerator

***kLPUART\_TxDataRegEmptyFlag*** Transmit data register empty flag, sets when transmit buffer is empty.

***kLPUART\_TransmissionCompleteFlag*** Transmission complete flag, sets when transmission activity complete.

***kLPUART\_RxDataRegFullFlag*** Receive data register full flag, sets when the receive data buffer is full.

***kLPUART\_IdleLineFlag*** Idle line detect flag, sets when idle line detected.

***kLPUART\_RxOverrunFlag*** Receive Overrun, sets when new data is received before data is read from receive register.

***kLPUART\_NoiseErrorFlag*** Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets

***kLPUART\_FramingErrorFlag*** Frame error flag, sets if logic 0 was detected where stop bit expected.

***kLPUART\_ParityErrorFlag*** If parity enabled, sets upon parity error detection.

***kLPUART\_LinBreakFlag*** LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.

***kLPUART\_RxActiveEdgeFlag*** Receive pin active edge interrupt flag, sets when active edge detected.

***kLPUART\_RxActiveFlag*** Receiver Active Flag (RAF), sets at beginning of valid start bit.

***kLPUART\_DataMatch1Flag*** The next character to be read from LPUART\_DATA matches MA1.

***kLPUART\_DataMatch2Flag*** The next character to be read from LPUART\_DATA matches MA2.

***kLPUART\_NoiseErrorInRxDataRegFlag*** NOISY bit, sets if noise detected in current data word.

***kLPUART\_ParityErrorInRxDataRegFlag*** PARITY bit, sets if noise detected in current data word.

***kLPUART\_TxFifoEmptyFlag*** TXEMPT bit, sets if transmit buffer is empty.

***kLPUART\_RxFifoEmptyFlag*** RXEMPT bit, sets if receive buffer is empty.

***kLPUART\_TxFifoOverflowFlag*** TXOF bit, sets if transmit buffer overflow occurred.

***kLPUART\_RxFifoUnderflowFlag*** RXUF bit, sets if receive buffer underflow occurred.

### 26.2.7 Function Documentation

**26.2.7.1 static void LPUART\_SoftwareReset ( LPUART\_Type \* *base* ) [inline],  
[static]**

This function resets all internal logic and registers except the Global Register. Remains set until cleared by software.

## Parameters

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

### 26.2.7.2 **status\_t LPUART\_Init( LPUART\_Type \* *base*, const lpuart\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )**

This function configures the LPUART module with user-defined settings. Call the [LPUART\\_GetDefaultConfig\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
* lpuart_config_t lpuartConfig;
* lpuartConfig.baudRate_Bps = 115200U;
* lpuartConfig.parityMode = kLPUART_ParityDisabled;
* lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
* lpuartConfig.isMsb = false;
* lpuartConfig.stopBitCount = kLPUART_OneStopBit;
* lpuartConfig.txFifoWatermark = 0;
* lpuartConfig.rxFifoWatermark = 1;
* LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

## Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>base</i>        | LPUART peripheral base address.                    |
| <i>config</i>      | Pointer to a user-defined configuration structure. |
| <i>srcClock_Hz</i> | LPUART clock source frequency in HZ.               |

## Return values

|                                          |                                                  |
|------------------------------------------|--------------------------------------------------|
| <i>kStatus_LPUART_BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                   | LPUART initialize succeed                        |

### 26.2.7.3 **void LPUART\_Deinit( LPUART\_Type \* *base* )**

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

## Parameters

## LPUART Driver

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

### 26.2.7.4 void LPUART\_GetDefaultConfig ( *Ipuart\_config\_t* \* *config* )

This function initializes the LPUART configuration structure to a default value. The default values are:  
: IpuartConfig->baudRate\_Bps = 115200U; IpuartConfig->parityMode = kLPUART\_ParityDisabled;  
IpuartConfig->dataBitsCount = kLPUART\_EightDataBits; IpuartConfig->isMsb = false; IpuartConfig->stopBitCount = kLPUART\_OneStopBit; IpuartConfig->txFifoWatermark = 0; IpuartConfig->rxFifoWatermark = 1; IpuartConfig->rxIdleType = kLPUART\_IdleTypeStartBit; IpuartConfig->rxIdleConfig = kLPUART\_IdleCharacter1; IpuartConfig->enableTx = false; IpuartConfig->enableRx = false;

Parameters

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

### 26.2.7.5 status\_t LPUART\_SetBaudRate ( *LPUART\_Type* \* *base*, *uint32\_t* *baudRate\_Bps*, *uint32\_t* *srcClock\_Hz* )

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART\_Init.

```
* LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

|                     |                                      |
|---------------------|--------------------------------------|
| <i>base</i>         | LPUART peripheral base address.      |
| <i>baudRate_Bps</i> | LPUART baudrate to be set.           |
| <i>srcClock_Hz</i>  | LPUART clock source frequency in HZ. |

Return values

|                                          |                                                        |
|------------------------------------------|--------------------------------------------------------|
| <i>kStatus_LPUART_BaudrateNotSupport</i> | Baudrate is not supported in the current clock source. |
| <i>kStatus_Success</i>                   | Set baudrate succeeded.                                |

### 26.2.7.6 *uint32\_t* LPUART\_GetStatusFlags ( *LPUART\_Type* \* *base* )

This function gets all LPUART status flags. The flags are returned as the logical OR value of the enumerators [\\_lpuart\\_flags](#). To check for a specific status, compare the return value with enumerators in the [\\_lpuart\\_flags](#). For example, to check whether the TX is empty:

```

* if (kLPUART_TxDataRegEmptyFlag &
* LPUART_GetStatusFlags(LPUART1))
* {
* ...
* }
*

```

## Parameters

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

## Returns

LPUART status flags which are ORed by the enumerators in the \_lpuart\_flags.

**26.2.7.7 `status_t LPUART_ClearStatusFlags ( LPUART_Type * base, uint32_t mask )`**

This function clears LPUART status flags with a provided mask. Automatically cleared flags can't be cleared by this function. Flags that can only be cleared or set by hardware are: kLPUART\_TxDataRegEmptyFlag, kLPUART\_TransmissionCompleteFlag, kLPUART\_RxDataRegFullFlag, kLPUART\_RxActiveFlag, kLPUART\_NoiseErrorInRxDataRegFlag, kLPUART\_ParityErrorInRxDataRegFlag, kLPUART\_TxFifoEmptyFlag, kLPUART\_RxFifoEmptyFlag. Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.

## Parameters

|             |                                                                                                                                        |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                                                                        |
| <i>mask</i> | the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask. |

## Returns

0 succeed, others failed.

## Return values

|                                               |                                                                                         |
|-----------------------------------------------|-----------------------------------------------------------------------------------------|
| <i>kStatus_LPUART_FlagCannotClearManually</i> | The flag can't be cleared by this function but it is cleared automatically by hardware. |
|-----------------------------------------------|-----------------------------------------------------------------------------------------|

## LPUART Driver

|                        |                                 |
|------------------------|---------------------------------|
| <i>kStatus_Success</i> | Status in the mask are cleared. |
|------------------------|---------------------------------|

### 26.2.7.8 void LPUART\_EnableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )

This function enables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See the [\\_lpuart\\_interrupt\\_enable](#). This examples shows how to enable TX empty interrupt and RX full interrupt:

```
* LPUART_EnableInterrupts(LPUART1,
 kLPUART_TxDataRegEmptyInterruptEnable |
 kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                  |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_uart_interrupt_enable</a> . |

### 26.2.7.9 void LPUART\_DisableInterrupts ( LPUART\_Type \* *base*, uint32\_t *mask* )

This function disables the LPUART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [\\_lpuart\\_interrupt\\_enable](#). This example shows how to disable the TX empty interrupt and RX full interrupt:

```
* LPUART_DisableInterrupts(LPUART1,
 kLPUART_TxDataRegEmptyInterruptEnable |
 kLPUART_RxDataRegFullInterruptEnable);
*
```

Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | LPUART peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_lpuart_interrupt_enable</a> . |

### 26.2.7.10 uint32\_t LPUART\_GetEnabledInterrupts ( LPUART\_Type \* *base* )

This function gets the enabled LPUART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [\\_lpuart\\_interrupt\\_enable](#). To check a specific interrupt enable status, compare the return value with enumerators in [\\_lpuart\\_interrupt\\_enable](#). For example, to check whether the TX empty interrupt is enabled:

```

* uint32_t enabledInterrupts = LPUART_GetEnabledInterrupts(LPUART1);
*
* if (kLPUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)
* {
* ...
* }
*

```

Parameters

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

Returns

LPUART interrupt flags which are logical OR of the enumerators in [\\_lpuart\\_interrupt\\_enable](#).

#### 26.2.7.11 static uint32\_t LPUART\_GetDataRegisterAddress ( LPUART\_Type \* *base* ) [inline], [static]

This function returns the LPUART data register address, which is mainly used by the DMA/eDMA.

Parameters

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

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

#### 26.2.7.12 static void LPUART\_EnableTxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the transmit data register empty flag, STAT[TDRE], to generate DMA requests.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

#### 26.2.7.13 static void LPUART\_EnableRxDMA ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the receiver data register full flag, STAT[RDRF], to generate DMA requests.

## LPUART Driver

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

### 26.2.7.14 `uint32_t LPUART_GetInstance ( LPUART_Type * base )`

Parameters

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

Returns

LPUART instance.

### 26.2.7.15 `static void LPUART_EnableTx ( LPUART_Type * base, bool enable ) [inline], [static]`

This function enables or disables the LPUART transmitter.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

### 26.2.7.16 `static void LPUART_EnableRx ( LPUART_Type * base, bool enable ) [inline], [static]`

This function enables or disables the LPUART receiver.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPUART peripheral base address.   |
| <i>enable</i> | True to enable, false to disable. |

### 26.2.7.17 `static void LPUART_WriteByte ( LPUART_Type * base, uint8_t data ) [inline], [static]`

This function writes data to the transmitter register directly. The upper layer must ensure that the TX register is empty or that the TX FIFO has room before calling this function.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | LPUART peripheral base address. |
| <i>data</i> | Data write to the TX register.  |

#### 26.2.7.18 static uint8\_t LPUART\_ReadByte ( LPUART\_Type \* *base* ) [inline], [static]

This function reads data from the receiver register directly. The upper layer must ensure that the receiver register is full or that the RX FIFO has data before calling this function.

Parameters

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

Returns

Data read from data register.

#### 26.2.7.19 void LPUART\_WriteBlocking ( LPUART\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )

This function polls the transmitter register, waits for the register to be empty or for TX FIFO to have room, and writes data to the transmitter buffer.

Note

This function does not check whether all data has been sent out to the bus. Before disabling the transmitter, check the kLPUART\_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | LPUART peripheral base address.     |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

#### 26.2.7.20 status\_t LPUART\_ReadBlocking ( LPUART\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )

This function polls the receiver register, waits for the receiver register full or receiver FIFO has data, and reads data from the TX register.

## LPUART Driver

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | LPUART 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_LPUART_Rx-HardwareOverrun</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_LPUART_Noise-Error</i>        | Noise error happened while receiving data.      |
| <i>kStatus_LPUART_-FramingError</i>      | Framing error happened while receiving data.    |
| <i>kStatus_LPUART_Parity-Error</i>       | Parity error happened while receiving data.     |
| <i>kStatus_Success</i>                   | Successfully received all data.                 |

### 26.2.7.21 void LPUART\_TransferCreateHandle ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, Ipuart\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the LPUART handle, which can be used for other LPUART transactional APIs. Usually, for a specified LPUART instance, call this API once to get the initialized handle.

The LPUART driver supports the "background" receiving, which means that user can set up an RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [LPUART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as *ringBuffer*.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>base</i>     | LPUART peripheral base address. |
| <i>handle</i>   | LPUART handle pointer.          |
| <i>callback</i> | Callback function.              |
| <i>userData</i> | User data.                      |

### 26.2.7.22 status\_t LPUART\_TransferSendNonBlocking ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, Ipuart\_transfer\_t \* *xfer* )

This function send data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data written to the transmitter register. When all data is written to the TX register in the ISR, the LPUART driver calls the callback function and passes the [kStatus\\_LPUART\\_TxIdle](#) as status parameter.

#### Note

The [kStatus\\_LPUART\\_TxIdle](#) is passed to the upper layer when all data are written to the TX register. However, there is no check to ensure that all the data sent out. Before disabling the T-X, check the [kLPUART\\_TransmissionCompleteFlag](#) to ensure that the transmit is finished.

#### Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                    |
| <i>handle</i> | LPUART handle pointer.                                             |
| <i>xfer</i>   | LPUART transfer structure, see <a href="#">Ipuart_transfer_t</a> . |

#### Return values

|                                |                                                                                    |
|--------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data transmission.                                          |
| <i>kStatus_LPUART_TxBusy</i>   | Previous transmission still not finished, data not all written to the TX register. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                                                  |

### 26.2.7.23 void LPUART\_TransferStartRingBuffer ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the [UART\\_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 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.

## LPUART Driver

Parameters

|                       |                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------|
| <i>base</i>           | LPUART peripheral base address.                                                              |
| <i>handle</i>         | LPUART handle pointer.                                                                       |
| <i>ringBuffer</i>     | Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.                                                                     |

**26.2.7.24 void LPUART\_TransferStopRingBuffer ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )**

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

**26.2.7.25 size\_t LPUART\_TransferGetRxRingBufferLength ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )**

Parameters

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

Returns

Length of received data in RX ring buffer.

**26.2.7.26 void LPUART\_TransferAbortSend ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )**

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are not sent out.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

#### 26.2.7.27 **status\_t LPUART\_TransferGetSendCount ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been written to LPUART TX register by an interrupt method.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART 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> ; |

#### 26.2.7.28 **status\_t LPUART\_TransferReceiveNonBlocking ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, Ipuart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )**

This function receives data using an interrupt method. This is a non-blocking function which returns without waiting to ensure that all data are 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 for read, the receive request is saved by the LPUART driver. When the new data arrives, the receive request is serviced first. When all data is received, the LPUART driver notifies the upper layer through a callback function and passes a status parameter *kStatus\_UART\_RxIdle*. For example, the upper layer needs 10 bytes but there are only 5 bytes in ring buffer. The 5 bytes are copied to *xfer->data*, which returns with the parameter *receivedBytes* set to 5. For the remaining 5 bytes, the newly arrived data is saved from *xfer->data[5]*. When 5 bytes are received, the LPUART 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 *xfer->data*. When all data is received, the upper layer is notified.

## LPUART Driver

Parameters

|                      |                                                  |
|----------------------|--------------------------------------------------|
| <i>base</i>          | LPUART peripheral base address.                  |
| <i>handle</i>        | LPUART handle pointer.                           |
| <i>xfer</i>          | LPUART transfer structure, see #uart_transfer_t. |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.    |

Return values

|                                |                                                          |
|--------------------------------|----------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully queue the transfer into the transmit queue. |
| <i>kStatus_LPUART_Rx-Busy</i>  | Previous receive request is not finished.                |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                        |

### 26.2.7.29 void LPUART\_TransferAbortReceive ( LPUART\_Type \* *base*, Ipuart\_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>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 26.2.7.30 status\_t LPUART\_TransferGetReceiveCount ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART 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 count; |

### 26.2.7.31 void LPUART\_TransferHandleIRQ ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function handles the LPUART transmit and receive IRQ request.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

### 26.2.7.32 void LPUART\_TransferHandleErrorIRQ ( LPUART\_Type \* *base*, Ipuart\_handle\_t \* *handle* )

This function handles the LPUART error IRQ request.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |

## 26.3 LPUART DMA Driver

## 26.4 LPUART eDMA Driver

### 26.4.1 Overview

#### Data Structures

- struct [lpuart\\_edma\\_handle\\_t](#)  
*LPUART eDMA handle.* [More...](#)

#### TypeDefs

- [typedef void\(\\* lpuart\\_edma\\_transfer\\_callback\\_t \)](#)(LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*LPUART transfer callback function.*

#### Driver version

- #define [FSL\\_LPUART\\_EDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 2, 7))  
*LPUART EDMA driver version 2.2.7.*

#### eDMA transactional

- void [LPUART\\_TransferCreateHandleEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, lpuart\_edma\_transfer\_callback\_t callback, void \*userData, [edma\\_handle\\_t](#) \*txEdmaHandle, [edma\\_handle\\_t](#) \*rxEdmaHandle)  
*Initializes the LPUART handle which is used in transactional functions.*
- status\_t [LPUART\\_SendEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Sends data using eDMA.*
- status\_t [LPUART\\_ReceiveEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, [lpuart\\_transfer\\_t](#) \*xfer)  
*Receives data using eDMA.*
- void [LPUART\\_TransferAbortSendEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the sent data using eDMA.*
- void [LPUART\\_TransferAbortReceiveEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle)  
*Aborts the received data using eDMA.*
- status\_t [LPUART\\_TransferGetSendCountEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes written to the LPUART TX register.*
- status\_t [LPUART\\_TransferGetReceiveCountEDMA](#) (LPUART\_Type \*base, lpuart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of received bytes.*

### 26.4.2 Data Structure Documentation

#### 26.4.2.1 struct \_lpuart\_edma\_handle

##### Data Fields

- **lpuart\_edma\_transfer\_callback\_t callback**  
*Callback function.*
- **void \*userData**  
*LPUART callback function parameter.*
- **size\_t rxDataSizeAll**  
*Size of the data to receive.*
- **size\_t txDataSizeAll**  
*Size of the data to send out.*
- **edma\_handle\_t \*txEdmaHandle**  
*The eDMA TX channel used.*
- **edma\_handle\_t \*rxEdmaHandle**  
*The eDMA RX channel used.*
- **uint8\_t nbytes**  
*eDMA minor byte transfer count initially configured.*
- **volatile uint8\_t txState**  
*TX transfer state.*
- **volatile uint8\_t rxState**  
*RX transfer state.*

#### 26.4.2.1.0.84 Field Documentation

26.4.2.1.0.84.1 `lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback`

26.4.2.1.0.84.2 `void* lpuart_edma_handle_t::userData`

26.4.2.1.0.84.3 `size_t lpuart_edma_handle_t::rxDataSizeAll`

26.4.2.1.0.84.4 `size_t lpuart_edma_handle_t::txDataSizeAll`

26.4.2.1.0.84.5 `edma_handle_t* lpuart_edma_handle_t::txEdmaHandle`

26.4.2.1.0.84.6 `edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle`

26.4.2.1.0.84.7 `uint8_t lpuart_edma_handle_t::nbytes`

26.4.2.1.0.84.8 `volatile uint8_t lpuart_edma_handle_t::txState`

#### 26.4.3 Macro Definition Documentation

26.4.3.1 `#define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 7))`

#### 26.4.4 Typedef Documentation

26.4.4.1 `typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base,  
lpuart_edma_handle_t *handle, status_t status, void *userData)`

#### 26.4.5 Function Documentation

26.4.5.1 `void LPUART_TransferCreateHandleEDMA ( LPUART_Type * base,  
lpuart_edma_handle_t * handle, lpuart_edma_transfer_callback_t callback, void  
* userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`

## LPUART eDMA Driver

Parameters

|                     |                                                |
|---------------------|------------------------------------------------|
| <i>base</i>         | LPUART peripheral base address.                |
| <i>handle</i>       | Pointer to lpuart_edma_handle_t structure.     |
| <i>callback</i>     | Callback function.                             |
| <i>userData</i>     | User data.                                     |
| <i>txEdmaHandle</i> | User requested DMA handle for TX DMA transfer. |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer. |

### 26.4.5.2 status\_t LPUART\_SendEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer* )

This function sends data using eDMA. 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>   | LPUART peripheral base address.                                         |
| <i>handle</i> | LPUART handle pointer.                                                  |
| <i>xfer</i>   | LPUART eDMA transfer structure. See <a href="#">lpuart_transfer_t</a> . |

Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_LPUART_TxBusy</i>   | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

### 26.4.5.3 status\_t LPUART\_ReceiveEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, lpuart\_transfer\_t \* *xfer* )

This function receives data using eDMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.                                         |
| <i>handle</i> | Pointer to lpuart_edma_handle_t structure.                              |
| <i>xfer</i>   | LPUART eDMA transfer structure, see <a href="#">lpuart_transfer_t</a> . |

Return values

|                                |                            |
|--------------------------------|----------------------------|
| <i>kStatus_Success</i>         | if succeed, others fail.   |
| <i>kStatus_LPUART_Rx-Busy</i>  | Previous transfer ongoing. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.          |

#### 26.4.5.4 void LPUART\_TransferAbortSendEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle* )

This function aborts the sent data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.            |
| <i>handle</i> | Pointer to lpuart_edma_handle_t structure. |

#### 26.4.5.5 void LPUART\_TransferAbortReceiveEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle* )

This function aborts the received data using eDMA.

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | LPUART peripheral base address.            |
| <i>handle</i> | Pointer to lpuart_edma_handle_t structure. |

#### 26.4.5.6 status\_t LPUART\_TransferGetSendCountEDMA ( LPUART\_Type \* *base*, lpuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes written to the LPUART TX register by DMA.

## LPUART eDMA Driver

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART handle pointer.          |
| <i>count</i>  | Send bytes count.               |

Return values

|                                      |                                                       |
|--------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferIn-Progress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>       | Parameter is invalid.                                 |
| <i>kStatus_Success</i>               | Get successfully through the parameter <i>count</i> ; |

### 26.4.5.7 **status\_t LPUART\_TransferGetReceiveCountEDMA ( LPUART\_Type \* *base*, Ipuart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of received bytes.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | LPUART peripheral base address. |
| <i>handle</i> | LPUART 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 <i>count</i> ; |

## 26.5 LPUART FreeRTOS Driver

### 26.5.1 Overview

#### Data Structures

- struct `lpuart_rtos_config_t`  
*LPUART RTOS configuration structure.* [More...](#)

#### Driver version

- #define `FSL_LPUART_FREERTOS_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 6))  
*LPUART freertos driver version 2.2.6.*

#### LPUART RTOS Operation

- int `LPUART_RTOS_Init` (lpuart\_rtos\_handle\_t \*handle, lpuart\_handle\_t \*t\_handle, const `lpuart_rtos_config_t` \*cfg)  
*Initializes an LPUART instance for operation in RTOS.*
- int `LPUART_RTOS_Deinit` (lpuart\_rtos\_handle\_t \*handle)  
*Deinitializes an LPUART instance for operation.*

#### LPUART transactional Operation

- int `LPUART_RTOS_Send` (lpuart\_rtos\_handle\_t \*handle, const uint8\_t \*buffer, uint32\_t length)  
*Sends data in the background.*
- int `LPUART_RTOS_Receive` (lpuart\_rtos\_handle\_t \*handle, uint8\_t \*buffer, uint32\_t length, size\_t \*received)  
*Receives data.*

### 26.5.2 Data Structure Documentation

#### 26.5.2.1 struct `lpuart_rtos_config_t`

##### Data Fields

- `LPUART_Type * base`  
*UART base address.*
- `uint32_t srclk`  
*UART source clock in Hz.*
- `uint32_t baudrate`  
*Desired communication speed.*
- `lpuart_parity_mode_t parity`  
*Parity setting.*

## LPUART FreeRTOS Driver

- `lpuart_stop_bit_count_t stopbits`  
*Number of stop bits to use.*
- `uint8_t * buffer`  
*Buffer for background reception.*
- `uint32_t buffer_size`  
*Size of buffer for background reception.*

### 26.5.3 Macro Definition Documentation

#### 26.5.3.1 `#define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 2, 6))`

### 26.5.4 Function Documentation

#### 26.5.4.1 `int LPUART_RTOS_Init ( lpuart_rtos_handle_t * handle, lpuart_handle_t * t_handle, const lpuart_rtos_config_t * cfg )`

Parameters

|                       |                                                                                      |
|-----------------------|--------------------------------------------------------------------------------------|
| <code>handle</code>   | The RTOS LPUART handle, the pointer to an allocated space for RTOS context.          |
| <code>t_handle</code> | The pointer to an allocated space to store the transactional layer internal state.   |
| <code>cfg</code>      | The pointer to the parameters required to configure the LPUART after initialization. |

Returns

0 succeed, others failed

#### 26.5.4.2 `int LPUART_RTOS_Deinit ( lpuart_rtos_handle_t * handle )`

This function deinitializes the LPUART module, sets all register value to the reset value, and releases the resources.

Parameters

|                     |                         |
|---------------------|-------------------------|
| <code>handle</code> | The RTOS LPUART handle. |
|---------------------|-------------------------|

#### 26.5.4.3 `int LPUART_RTOS_Send ( lpuart_rtos_handle_t * handle, const uint8_t * buffer, uint32_t length )`

This function sends data. It is an synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS LPUART handle.        |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 26.5.4.4 int LPUART\_RTOs\_Receive ( *Ipuart\_rtos\_handle\_t \* handle*, *uint8\_t \* buffer*, *uint32\_t length*, *size\_t \* received* )

This function receives data from LPUART. It is an synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS LPUART handle.                                                          |
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.                                                  |
| <i>received</i> | The pointer to a variable of size_t where the number of received data is filled. |



# **Chapter 27**

## **Memory-Mapped Cryptographic Acceleration Unit (mmCAU)**

### **27.1 Overview**

The mmCAU software library uses the mmCAU co-processor that is connected to the ARM Cortex-M4-/M0+ Private Peripheral Bus (PPB). In this chapter, CAU refers to both CAU and mmCAU unless explicitly noted.

### **27.2 Purpose**

The following chapter describes how to use the mmCAU software library in any application to integrate a cryptographic algorithm or hashing function supported by the software library. NXP products supported by the software library are MCU/MPUs. Check the specific Freescale product for CAU availability.

### **27.3 Library Features**

The library is as compact and generic as possible to simplify the integration with existing cryptographic software. The library has a standard header file with ANSI C prototypes for all functions: "cau\_api.h". This software library is thread safe only if CAU registers are saved on a context switch. The mmCAU software library is also compatible to ARM C compiler conventions (EABI). All pointers passed to mmCAU API functions (input and output data blocks, keys, key schedules, and so on) are aligned to 0-modulo-4 addresses.

For applications that don't need to deal with the aligned addresses, a simple wrapper layer is provided. The wrapper layer consists of the "fsl\_mmcau.h" header file and "fsl\_mmcau.c" source code file. The only function of the wrapper layer is that it supports unaligned addresses

. The CAU library supports the following encryption/decryption algorithms and hashing functions:

- AES128
- AES192
- AES256
- DES
- MD5
- SHA1
- SHA256

**Note:** 3DES crypto algorithms are supported by calling the corresponding DES crypto function three times. Hardware support for SHA256 is only present in the CAU version 2. See the appropriate MC-U/MPU reference manual for details about availability. Additionally, the [cau\\_sha256\\_initialize\\_output\(\)](#) function checks the hardware revision and returns a (-1) value if the CAU lacks SHA256 support.

### **27.4 CAU and mmCAU software library overview**

Table 1 shows the crypto algorithms and hashing functions included in the software library:

## mmCAU software library usage

|                          |                            |                              |
|--------------------------|----------------------------|------------------------------|
| <b>Crypto Algorithms</b> | AES128<br>AES192<br>AES256 | cau_aes_set_key              |
|                          |                            | cau_aes_encrypt              |
|                          |                            | cau_aes_decrypt              |
|                          | DES/3DES                   | cau_des_chk_parity           |
|                          |                            | cau_des_encrypt              |
|                          |                            | cau_des_decrypt              |
| <b>Hashing Functions</b> | MD5                        | cau_md5_initialize_output    |
|                          |                            | cau_md5_hash_n               |
|                          |                            | cau_md5_update               |
|                          |                            | cau_md5_hash                 |
|                          | SHA1                       | cau_sha1_initialize_output   |
|                          |                            | cau_sha1_hash_n              |
|                          |                            | cau_sha1_update              |
|                          |                            | cau_sha1_hash                |
|                          | SHA256                     | cau_sha256_initialize_output |
|                          |                            | cau_sha256_hash_n            |
|                          |                            | cau_sha256_update            |
|                          |                            | cau_sha256_hash              |

**Table 1: Library Overview**

## 27.5 mmCAU software library usage

The software library contains the following files:

| File        | Description               |
|-------------|---------------------------|
| cau_api.h   | CAU and mmCAU header file |
| lib_mmcau.a | mmCAU library             |

**Table 2: File Description**

The header file and lib\_mmcau.a must always be included in the project.

## Functions

- void `cau_aes_set_key` (const unsigned char \*key, const int key\_size, unsigned char \*key\_sch)  
*AES: Performs an AES key expansion.*
- void `cau_aes_encrypt` (const unsigned char \*in, const unsigned char \*key\_sch, const int nr, unsigned char \*out)

- void **cau\_aes\_decrypt** (const unsigned char \*in, const unsigned char \*key\_sch, const int nr, unsigned char \*out)
 

*AES: Encrypts a single 16 byte block.*
- int **cau\_des\_chk\_parity** (const unsigned char \*key)
 

*DES: Checks key parity.*
- void **cau\_des\_encrypt** (const unsigned char \*in, const unsigned char \*key, unsigned char \*out)
 

*DES: Encrypts a single 8-byte block.*
- void **cau\_des\_decrypt** (const unsigned char \*in, const unsigned char \*key, unsigned char \*out)
 

*DES: Decrypts a single 8-byte block.*
- void **cau\_md5\_initialize\_output** (const unsigned char \*md5\_state)
 

*MD5: Initializes the MD5 state variables.*
- void **cau\_md5\_hash\_n** (const unsigned char \*msg\_data, const int num\_blks, unsigned char \*md5\_state)
 

*MD5: Updates MD5 state variables with n message blocks.*
- void **cau\_md5\_update** (const unsigned char \*msg\_data, const int num\_blks, unsigned char \*md5\_state)
 

*MD5: Updates MD5 state variables.*
- void **cau\_md5\_hash** (const unsigned char \*msg\_data, unsigned char \*md5\_state)
 

*MD5: Updates MD5 state variables with one message block.*
- void **cau\_sha1\_initialize\_output** (const unsigned int \*sha1\_state)
 

*SHA1: Initializes the SHA1 state variables.*
- void **cau\_sha1\_hash\_n** (const unsigned char \*msg\_data, const int num\_blks, unsigned int \*sha1\_state)
 

*SHA1: Updates SHA1 state variables with n message blocks.*
- void **cau\_sha1\_update** (const unsigned char \*msg\_data, const int num\_blks, unsigned int \*sha1\_state)
 

*SHA1: Updates SHA1 state variables.*
- void **cau\_sha1\_hash** (const unsigned char \*msg\_data, unsigned int \*sha1\_state)
 

*SHA1: Updates SHA1 state variables with one message block.*
- int **cau\_sha256\_initialize\_output** (const unsigned int \*output)
 

*SHA256: Initializes the SHA256 state variables.*
- void **cau\_sha256\_hash\_n** (const unsigned char \*input, const int num\_blks, unsigned int \*output)
 

*SHA256: Updates SHA256 state variables with n message blocks.*
- void **cau\_sha256\_update** (const unsigned char \*input, const int num\_blks, unsigned int \*output)
 

*SHA256: Updates SHA256 state variables.*
- void **cau\_sha256\_hash** (const unsigned char \*input, unsigned int \*output)
 

*SHA256: Updates SHA256 state variables with one message block.*
- status\_t **MMCAU\_AES\_SetKey** (const uint8\_t \*key, const size\_t keySize, uint8\_t \*keySch)
 

*AES: Performs an AES key expansion.*
- status\_t **MMCAU\_AES\_EncryptEcb** (const uint8\_t \*in, const uint8\_t \*keySch, uint32\_t aesRounds, uint8\_t \*out)
 

*AES: Encrypts a single 16 byte block.*
- status\_t **MMCAU\_AES\_DecryptEcb** (const uint8\_t \*in, const uint8\_t \*keySch, uint32\_t aesRounds, uint8\_t \*out)
 

*AES: Decrypts a single 16-byte block.*
- status\_t **MMCAU\_DES\_ChkParity** (const uint8\_t \*key)
 

*DES: Checks the key parity.*
- status\_t **MMCAU\_DES\_EncryptEcb** (const uint8\_t \*in, const uint8\_t \*key, uint8\_t \*out)
 

*DES: Encrypts a single 8-byte block.*

## Function Documentation

- status\_t [MMCAU\\_DES\\_DecryptEcb](#) (const uint8\_t \*in, const uint8\_t \*key, uint8\_t \*out)  
*DES: Decrypts a single 8-byte block.*
- status\_t [MMCAU\\_MD5\\_InitializeOutput](#) (uint32\_t \*md5State)  
*MD5: Initializes the MD5 state variables.*
- status\_t [MMCAU\\_MD5\\_HashN](#) (const uint8\_t \*msgData, uint32\_t numBlocks, uint32\_t \*md5State)  
*MD5: Updates the MD5 state variables with n message blocks.*
- status\_t [MMCAU\\_MD5\\_Update](#) (const uint8\_t \*msgData, uint32\_t numBlocks, uint32\_t \*md5State)  
*MD5: Updates the MD5 state variables.*
- status\_t [MMCAU\\_SHA1\\_InitializeOutput](#) (uint32\_t \*sha1State)  
*SHA1: Initializes the SHA1 state variables.*
- status\_t [MMCAU\\_SHA1\\_HashN](#) (const uint8\_t \*msgData, uint32\_t numBlocks, uint32\_t \*sha1State)  
*SHA1: Updates the SHA1 state variables with n message blocks.*
- status\_t [MMCAU\\_SHA1\\_Update](#) (const uint8\_t \*msgData, uint32\_t numBlocks, uint32\_t \*sha1State)  
*SHA1: Updates the SHA1 state variables.*
- status\_t [MMCAU\\_SHA256\\_InitializeOutput](#) (uint32\_t \*sha256State)  
*SHA256: Initializes the SHA256 state variables.*
- status\_t [MMCAU\\_SHA256\\_HashN](#) (const uint8\_t \*input, uint32\_t numBlocks, uint32\_t \*sha256State)  
*SHA256: Updates the SHA256 state variables with n message blocks.*
- status\_t [MMCAU\\_SHA256\\_Update](#) (const uint8\_t \*input, uint32\_t numBlocks, uint32\_t \*sha256State)  
*SHA256: Updates SHA256 state variables.*

## 27.6 Function Documentation

### 27.6.1 void cau\_aes\_set\_key ( const unsigned char \* key, const int key\_size, unsigned char \* key\_sch )

This function performs an AES key expansion

Parameters

|     |                 |                                                       |
|-----|-----------------|-------------------------------------------------------|
|     | <i>key</i>      | Pointer to input key (128, 192, 256 bits in length).  |
|     | <i>key_size</i> | Key size in bits (128, 192, 256)                      |
| out | <i>key_sch</i>  | Pointer to key schedule output (44, 52, 60 longwords) |

Note

All pointers must have word (4 bytes) alignment

Table below shows the requirements for the [cau\\_aes\\_set\\_key\(\)](#) function when using AES128, AES192 or AES256.

| [in] Key Size (bits) | [out] Key Schedule Size (32 bit data values) |
|----------------------|----------------------------------------------|
| 128                  | 44                                           |
| 192                  | 52                                           |
| 256                  | 60                                           |

### 27.6.2 void cau\_aes\_encrypt( const unsigned char \* *in*, const unsigned char \* *key\_sch*, const int *nr*, unsigned char \* *out* )

This function encrypts a single 16-byte block for AES128, AES192 and AES256

Parameters

|     |                |                                                     |
|-----|----------------|-----------------------------------------------------|
|     | <i>in</i>      | Pointer to 16-byte block of input plaintext         |
|     | <i>key_sch</i> | Pointer to key schedule (44, 52, 60 longwords)      |
|     | <i>nr</i>      | Number of AES rounds (10, 12, 14 = f(key_schedule)) |
| out | <i>out</i>     | Pointer to 16-byte block of output ciphertext       |

Note

All pointers must have word (4 bytes) alignment

Input and output blocks may overlap.

Table below shows the requirements for the [cau\\_aes\\_encrypt\(\)](#)/[cau\\_aes\\_decrypt\(\)](#) function when using AES128, AES192 or AES256.

| Block Cipher | [in] Key Schedule Size (longwords) | [in] Number of AES rounds |
|--------------|------------------------------------|---------------------------|
| AES128       | 44                                 | 10                        |
| AES192       | 52                                 | 12                        |
| AES256       | 60                                 | 14                        |

### 27.6.3 void cau\_aes\_decrypt( const unsigned char \* *in*, const unsigned char \* *key\_sch*, const int *nr*, unsigned char \* *out* )

This function decrypts a single 16-byte block for AES128, AES192 and AES256

## Function Documentation

Parameters

|     |                |                                                     |
|-----|----------------|-----------------------------------------------------|
|     | <i>in</i>      | Pointer to 16-byte block of input ciphertext        |
|     | <i>key_sch</i> | Pointer to key schedule (44, 52, 60 longwords)      |
|     | <i>nr</i>      | Number of AES rounds (10, 12, 14 = f(key_schedule)) |
| out | <i>out</i>     | Pointer to 16-byte block of output plaintext        |

Note

All pointers must have word (4 bytes) alignment

Input and output blocks may overlap.

Table below shows the requirements for the [cau\\_aes\\_encrypt\(\)](#)/[cau\\_aes\\_decrypt\(\)](#) function when using AES128, AES192 or AES256.

|  | <b>Block Cipher</b> | <b>[in] Key Schedule Size (longwords)</b> | <b>[in] Number of AES rounds</b> |
|--|---------------------|-------------------------------------------|----------------------------------|
|  | AES128              | 44                                        | 10                               |
|  | AES192              | 52                                        | 12                               |
|  | AES256              | 60                                        | 14                               |

### 27.6.4 int cau\_des\_chk\_parity ( const unsigned char \* key )

This function checks the parity of a DES key

Parameters

|            |                                                                      |
|------------|----------------------------------------------------------------------|
| <i>key</i> | 64-bit DES key with parity bits. Must have word (4 bytes) alignment. |
|------------|----------------------------------------------------------------------|

Returns

0 no error

-1 parity error

### 27.6.5 void cau\_des\_encrypt ( const unsigned char \* *in*, const unsigned char \* *key*, unsigned char \* *out* )

This function encrypts a single 8-byte block with DES algorithm.

Parameters

|     |            |                                              |
|-----|------------|----------------------------------------------|
|     | <i>in</i>  | Pointer to 8-byte block of input plaintext   |
|     | <i>key</i> | Pointer to 64-bit DES key with parity bits   |
| out | <i>out</i> | Pointer to 8-byte block of output ciphertext |

Note

All pointers must have word (4 bytes) alignment

Input and output blocks may overlap.

#### **27.6.6 void cau\_des\_decrypt ( const unsigned char \* *in*, const unsigned char \* *key*, unsigned char \* *out* )**

This function decrypts a single 8-byte block with DES algorithm.

Parameters

|     |            |                                             |
|-----|------------|---------------------------------------------|
|     | <i>in</i>  | Pointer to 8-byte block of input ciphertext |
|     | <i>key</i> | Pointer to 64-bit DES key with parity bits  |
| out | <i>out</i> | Pointer to 8-byte block of output plaintext |

Note

All pointers must have word (4 bytes) alignment

Input and output blocks may overlap.

#### **27.6.7 void cau\_md5\_initialize\_output ( const unsigned char \* *md5\_state* )**

This function initializes the MD5 state variables. The output can be used as input to [cau\\_md5\\_hash\(\)](#) and [cau\\_md5\\_hash\\_n\(\)](#).

Parameters

|     |                  |                                                          |
|-----|------------------|----------------------------------------------------------|
| out | <i>md5_state</i> | Pointer to 128-bit block of md5 state variables: a,b,c,d |
|-----|------------------|----------------------------------------------------------|

Note

All pointers must have word (4 bytes) alignment

## Function Documentation

**27.6.8 void cau\_md5\_hash\_n ( const unsigned char \* *msg\_data*, const int *num\_blk*s, unsigned char \* *md5\_state* )**

This function updates MD5 state variables for one or more input message blocks

## Parameters

|         |                  |                                                          |
|---------|------------------|----------------------------------------------------------|
|         | <i>msg_data</i>  | Pointer to start of input message data                   |
|         | <i>num_blk</i> s | Number of 512-bit blocks to process                      |
| in, out | <i>md5_state</i> | Pointer to 128-bit block of MD5 state variables: a,b,c,d |

## Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_md5\\_initialize\\_output\(\)](#) function must be called when starting a new hash. Useful when handling non-contiguous input message blocks.

### 27.6.9 void cau\_md5\_update ( const unsigned char \* *msg\_data*, const int *num\_blk*s, unsigned char \* *md5\_state* )

This function updates MD5 state variables for one or more input message blocks. It starts a new hash as it internally calls [cau\\_md5\\_initialize\\_output\(\)](#) first.

## Parameters

|     |                  |                                                          |
|-----|------------------|----------------------------------------------------------|
|     | <i>msg_data</i>  | Pointer to start of input message data                   |
|     | <i>num_blk</i> s | Number of 512-bit blocks to process                      |
| out | <i>md5_state</i> | Pointer to 128-bit block of MD5 state variables: a,b,c,d |

## Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_md5\\_initialize\\_output\(\)](#) function is not required to be called as it is called internally to start a new hash. All input message blocks must be contiguous.

### 27.6.10 void cau\_md5\_hash ( const unsigned char \* *msg\_data*, unsigned char \* *md5\_state* )

This function updates MD5 state variables for one input message block

## Function Documentation

Parameters

|        |                  |                                                          |
|--------|------------------|----------------------------------------------------------|
|        | <i>msg_data</i>  | Pointer to start of 512-bits of input message data       |
| in,out | <i>md5_state</i> | Pointer to 128-bit block of MD5 state variables: a,b,c,d |

Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_md5\\_initialize\\_output\(\)](#) function must be called when starting a new hash.

### 27.6.11 void cau\_sha1\_initialize\_output ( const unsigned int \* *sha1\_state* )

This function initializes the SHA1 state variables. The output can be used as input to [cau\\_sha1\\_hash\(\)](#) and [cau\\_sha1\\_hash\\_n\(\)](#).

Parameters

|     |                   |                                                             |
|-----|-------------------|-------------------------------------------------------------|
| out | <i>sha1_state</i> | Pointer to 160-bit block of SHA1 state variables: a,b,c,d,e |
|-----|-------------------|-------------------------------------------------------------|

Note

All pointers must have word (4 bytes) alignment

### 27.6.12 void cau\_sha1\_hash\_n ( const unsigned char \* *msg\_data*, const int *num\_blk*s, unsigned int \* *sha1\_state* )

This function updates SHA1 state variables for one or more input message blocks

Parameters

|        |                   |                                                             |
|--------|-------------------|-------------------------------------------------------------|
|        | <i>msg_data</i>   | Pointer to start of input message data                      |
|        | <i>num_blk</i> s  | Number of 512-bit blocks to process                         |
| in,out | <i>sha1_state</i> | Pointer to 160-bit block of SHA1 state variables: a,b,c,d,e |

Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_sha1\\_initialize\\_output\(\)](#) function must be called when starting a new hash. Useful when handling non-contiguous input message blocks.

**27.6.13 void cau\_sha1\_update ( const unsigned char \* *msg\_data*, const int *num\_blk*s, unsigned int \* *sha1\_state* )**

This function updates SHA1 state variables for one or more input message blocks. It starts a new hash as it internally calls [cau\\_sha1\\_initialize\\_output\(\)](#) first.

## Function Documentation

Parameters

|     |                   |                                                             |
|-----|-------------------|-------------------------------------------------------------|
|     | <i>msg_data</i>   | Pointer to start of input message data                      |
|     | <i>num_blk</i> s  | Number of 512-bit blocks to process                         |
| out | <i>sha1_state</i> | Pointer to 160-bit block of SHA1 state variables: a,b,c,d,e |

Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_sha1\\_initialize\\_output\(\)](#) function is not required to be called as it is called internally to start a new hash. All input message blocks must be contiguous.

### **27.6.14 void cau\_sha1\_hash ( const unsigned char \* *msg\_data*, unsigned int \* *sha1\_state* )**

This function updates SHA1 state variables for one input message block

Parameters

|         |                   |                                                             |
|---------|-------------------|-------------------------------------------------------------|
|         | <i>msg_data</i>   | Pointer to start of 512-bits of input message data          |
| in, out | <i>sha1_state</i> | Pointer to 160-bit block of SHA1 state variables: a,b,c,d,e |

Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_sha1\\_initialize\\_output\(\)](#) function must be called when starting a new hash.

### **27.6.15 int cau\_sha256\_initialize\_output ( const unsigned int \* *output* )**

This function initializes the SHA256 state variables. The output can be used as input to [cau\\_sha256\\_hash\(\)](#) and [cau\\_sha256\\_hash\\_n\(\)](#).

Parameters

|     |               |                                                                  |
|-----|---------------|------------------------------------------------------------------|
| out | <i>output</i> | Pointer to 256-bit block of SHA2 state variables a,b,c,d,e,f,g,h |
|-----|---------------|------------------------------------------------------------------|

## Note

All pointers must have word (4 bytes) alignment

## Returns

- 0 No error. CAU hardware support for SHA256 is present.
- 1 Error. CAU hardware support for SHA256 is not present.

### **27.6.16 void cau\_sha256\_hash\_n ( const unsigned char \* *input*, const int *num\_blk*s, unsigned int \* *output* )**

This function updates SHA256 state variables for one or more input message blocks

## Parameters

|         |                  |                                                                   |
|---------|------------------|-------------------------------------------------------------------|
|         | <i>input</i>     | Pointer to start of input message data                            |
|         | <i>num_blk</i> s | Number of 512-bit blocks to process                               |
| in, out | <i>output</i>    | Pointer to 256-bit block of SHA2 state variables: a,b,c,d,e,f,g,h |

## Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_sha256\\_initialize\\_output\(\)](#) function must be called when starting a new hash. Useful when handling non-contiguous input message blocks.

### **27.6.17 void cau\_sha256\_update ( const unsigned char \* *input*, const int *num\_blk*s, unsigned int \* *output* )**

This function updates SHA256 state variables for one or more input message blocks. It starts a new hash as it internally calls [cau\\_sha256\\_initialize\\_output\(\)](#) first.

## Function Documentation

Parameters

|     |                |                                                                   |
|-----|----------------|-------------------------------------------------------------------|
|     | <i>input</i>   | Pointer to start of input message data                            |
|     | <i>num_blk</i> | Number of 512-bit blocks to process                               |
| out | <i>output</i>  | Pointer to 256-bit block of SHA2 state variables: a,b,c,d,e,f,g,h |

Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_sha256\\_initialize\\_output\(\)](#) function is not required to be called as it is called internally to start a new hash. All input message blocks must be contiguous.

### 27.6.18 void cau\_sha256\_hash ( const unsigned char \* *input*, unsigned int \* *output* )

This function updates SHA256 state variables for one input message block

Parameters

|         |               |                                                                   |
|---------|---------------|-------------------------------------------------------------------|
|         | <i>input</i>  | Pointer to start of 512-bits of input message data                |
| in, out | <i>output</i> | Pointer to 256-bit block of SHA2 state variables: a,b,c,d,e,f,g,h |

Note

All pointers must have word (4 bytes) alignment

Input message and digest output blocks must not overlap. The [cau\\_sha256\\_initialize\\_output\(\)](#) function must be called when starting a new hash.

### 27.6.19 status\_t MMCAU\_AES\_SetKey ( const uint8\_t \* *key*, const size\_t *keySize*, uint8\_t \* *keySch* )

This function performs an AES key expansion.

Parameters

|     |                |                                                       |
|-----|----------------|-------------------------------------------------------|
|     | <i>key</i>     | Pointer to input key (128, 192, 256 bits in length).  |
|     | <i>keySize</i> | Key size in bytes (16, 24, 32)                        |
| out | <i>keySch</i>  | Pointer to key schedule output (44, 52, 60 longwords) |

## Note

Table below shows the requirements for the [MMCAU\\_AES\\_SetKey\(\)](#) function when using AES128, AES192, or AES256.

|  | <b>[in] Key Size (bits)</b> | <b>[out] Key Schedule Size (32 bit data values)</b> |
|--|-----------------------------|-----------------------------------------------------|
|  | 128                         | 44                                                  |
|  | 192                         | 52                                                  |
|  | 256                         | 60                                                  |

## Returns

Status of the operation. (kStatus\_Success, kStatus\_InvalidArgument, kStatus\_Fail)

### 27.6.20 **status\_t MMCAU\_AES\_EncryptEcb ( const uint8\_t \* *in*, const uint8\_t \* *keySch*, uint32\_t *aesRounds*, uint8\_t \* *out* )**

This function encrypts a single 16-byte block for AES128, AES192, and AES256.

## Parameters

|     |                  |                                                      |
|-----|------------------|------------------------------------------------------|
|     | <i>in</i>        | Pointer to 16-byte block of input plaintext.         |
|     | <i>keySch</i>    | Pointer to key schedule (44, 52, 60 longwords).      |
|     | <i>aesRounds</i> | Number of AES rounds (10, 12, 14 = f(key_schedule)). |
| out | <i>out</i>       | Pointer to 16-byte block of output ciphertext.       |

## Note

Input and output blocks may overlap.

Table below shows the requirements for the [MMCAU\\_AES\\_EncryptEcb\(\)](#)/[MMCAU\\_AES\\_DecryptEcb\(\)](#) function when using AES128, AES192 or AES256.

| <b>Block Cipher</b> | <b>[in] Key Schedule Size (longwords)</b> | <b>[in] Number of AES rounds</b> |
|---------------------|-------------------------------------------|----------------------------------|
| AES128              | 44                                        | 10                               |
| AES192              | 52                                        | 12                               |
| AES256              | 60                                        | 14                               |

## Returns

Status of the operation. (kStatus\_Success, kStatus\_InvalidArgument, kStatus\_Fail)

## Function Documentation

**27.6.21 status\_t MMCAU\_AES\_DecryptEcb ( const uint8\_t \* *in*, const uint8\_t \* *keySch*, uint32\_t *aesRounds*, uint8\_t \* *out* )**

This function decrypts a single 16-byte block for AES128, AES192, and AES256.

Parameters

|            |                  |                                                      |
|------------|------------------|------------------------------------------------------|
|            | <i>in</i>        | Pointer to 16-byte block of input ciphertext.        |
|            | <i>keySch</i>    | Pointer to key schedule (44, 52, 60 longwords).      |
|            | <i>aesRounds</i> | Number of AES rounds (10, 12, 14 = f(key_schedule)). |
| <i>out</i> | <i>out</i>       | Pointer to 16-byte block of output plaintext.        |

Note

Input and output blocks may overlap.

Table below shows the requirements for the [cau\\_aes\\_encrypt\(\)](#)/[cau\\_aes\\_decrypt\(\)](#). function when using AES128, AES192 or AES256.

| Block Cipher | [in] Key Schedule Size (longwords) | [in] Number of AES rounds |
|--------------|------------------------------------|---------------------------|
| AES128       | 44                                 | 10                        |
| AES192       | 52                                 | 12                        |
| AES256       | 60                                 | 14                        |

Returns

Status of the operation. (kStatus\_Success, kStatus\_InvalidArgument, kStatus\_Fail)

### 27.6.22 status\_t MMCAU\_DES\_ChkParity ( const uint8\_t \* key )

This function checks the parity of a DES key.

Parameters

|            |                                  |
|------------|----------------------------------|
| <i>key</i> | 64-bit DES key with parity bits. |
|------------|----------------------------------|

Returns

kStatus\_Success No error.

kStatus\_Fail Parity error.

kStatus\_InvalidArgument Key argument is NULL.

### 27.6.23 status\_t MMCAU\_DES\_EncryptEcb ( const uint8\_t \* in, const uint8\_t \* key, uint8\_t \* out )

This function encrypts a single 8-byte block with the DES algorithm.

## Function Documentation

### Parameters

|     |            |                                               |
|-----|------------|-----------------------------------------------|
|     | <i>in</i>  | Pointer to 8-byte block of input plaintext.   |
|     | <i>key</i> | Pointer to 64-bit DES key with parity bits.   |
| out | <i>out</i> | Pointer to 8-byte block of output ciphertext. |

### Note

Input and output blocks may overlap.

### Returns

Status of the operation. (kStatus\_Success, kStatus\_InvalidArgument, kStatus\_Fail)

### **27.6.24 status\_t MMCAU\_DES\_DecryptEcb ( const uint8\_t \* *in*, const uint8\_t \* *key*, uint8\_t \* *out* )**

This function decrypts a single 8-byte block with the DES algorithm.

### Parameters

|     |            |                                              |
|-----|------------|----------------------------------------------|
|     | <i>in</i>  | Pointer to 8-byte block of input ciphertext. |
|     | <i>key</i> | Pointer to 64-bit DES key with parity bits.  |
| out | <i>out</i> | Pointer to 8-byte block of output plaintext. |

### Note

Input and output blocks may overlap.

### Returns

Status of the operation. (kStatus\_Success, kStatus\_InvalidArgument, kStatus\_Fail)

### **27.6.25 status\_t MMCAU\_MD5\_InitializeOutput ( uint32\_t \* *md5State* )**

This function initializes the MD5 state variables. The output can be used as input to [MMCAU\\_MD5\\_HashN\(\)](#).

Parameters

|     |                 |                                                          |
|-----|-----------------|----------------------------------------------------------|
| out | <i>md5State</i> | Pointer to 128-bit block of md5 state variables: a,b,c,d |
|-----|-----------------|----------------------------------------------------------|

### 27.6.26 status\_t MMCAU\_MD5\_HashN ( *const uint8\_t \* msgData, uint32\_t numBlocks, uint32\_t \* md5State* )

This function updates the MD5 state variables for one or more input message blocks.

Parameters

|        |                  |                                                              |
|--------|------------------|--------------------------------------------------------------|
|        | <i>msgData</i>   | Pointer to start of input message data.                      |
|        | <i>numBlocks</i> | Number of 512-bit blocks to process.                         |
| in,out | <i>md5State</i>  | Pointer to 128-bit block of MD5 state variables: a, b, c, d. |

Note

Input message and digest output blocks must not overlap. The [MMCAU\\_MD5\\_InitializeOutput\(\)](#) function must be called when starting a new hash. Useful when handling non-contiguous input message blocks.

### 27.6.27 status\_t MMCAU\_MD5\_Update ( *const uint8\_t \* msgData, uint32\_t numBlocks, uint32\_t \* md5State* )

This function updates the MD5 state variables for one or more input message blocks. It starts a new hash as it internally calls [MMCAU\\_MD5\\_InitializeOutput\(\)](#) first.

Parameters

|     |                  |                                                              |
|-----|------------------|--------------------------------------------------------------|
|     | <i>msgData</i>   | Pointer to start of input message data.                      |
|     | <i>numBlocks</i> | Number of 512-bit blocks to process.                         |
| out | <i>md5State</i>  | Pointer to 128-bit block of MD5 state variables: a, b, c, d. |

Note

Input message and digest output blocks must not overlap. The [MMCAU\\_MD5\\_InitializeOutput\(\)](#) function is not required to be called as it is called internally to start a new hash. All input message blocks must be contiguous.

## Function Documentation

### 27.6.28 **status\_t MMCAU\_SHA1\_InitializeOutput( uint32\_t \* sha1State )**

This function initializes the SHA1 state variables. The output can be used as input to [MMCAU\\_SHA1\\_HashN\(\)](#).

Parameters

|                  |                        |                                                                  |
|------------------|------------------------|------------------------------------------------------------------|
| <code>out</code> | <code>sha1State</code> | Pointer to 160-bit block of SHA1 state variables: a, b, c, d, e. |
|------------------|------------------------|------------------------------------------------------------------|

### 27.6.29 `status_t MMCAU_SHA1_HashN( const uint8_t * msgData, uint32_t numBlocks, uint32_t * sha1State )`

This function updates the SHA1 state variables for one or more input message blocks.

Parameters

|                      |                        |                                                                  |
|----------------------|------------------------|------------------------------------------------------------------|
|                      | <code>msgData</code>   | Pointer to start of input message data.                          |
|                      | <code>numBlocks</code> | Number of 512-bit blocks to process.                             |
| <code>in, out</code> | <code>sha1State</code> | Pointer to 160-bit block of SHA1 state variables: a, b, c, d, e. |

Note

Input message and digest output blocks must not overlap. The [MMCAU\\_SHA1\\_InitializeOutput\(\)](#) function must be called when starting a new hash. Useful when handling non-contiguous input message blocks.

### 27.6.30 `status_t MMCAU_SHA1_Update( const uint8_t * msgData, uint32_t numBlocks, uint32_t * sha1State )`

This function updates the SHA1 state variables for one or more input message blocks. It starts a new hash as it internally calls [MMCAU\\_SHA1\\_InitializeOutput\(\)](#) first.

Parameters

|                  |                        |                                                                  |
|------------------|------------------------|------------------------------------------------------------------|
|                  | <code>msgData</code>   | Pointer to start of input message data.                          |
|                  | <code>numBlocks</code> | Number of 512-bit blocks to process.                             |
| <code>out</code> | <code>sha1State</code> | Pointer to 160-bit block of SHA1 state variables: a, b, c, d, e. |

Note

Input message and digest output blocks must not overlap. The [MMCAU\\_SHA1\\_InitializeOutput\(\)](#) function is not required to be called as it is called internally to start a new hash. All input message blocks must be contiguous.

## Function Documentation

### 27.6.31 **status\_t MMCAU\_SHA256\_InitializeOutput ( uint32\_t \* sha256State )**

This function initializes the SHA256 state variables. The output can be used as input to [MMCAU\\_SHA256\\_HashN\(\)](#).

Parameters

|                  |                          |                                                                          |
|------------------|--------------------------|--------------------------------------------------------------------------|
| <code>out</code> | <code>sha256State</code> | Pointer to 256-bit block of SHA2 state variables a, b, c, d, e, f, g, h. |
|------------------|--------------------------|--------------------------------------------------------------------------|

Returns

`kStatus_Success` No error. CAU hardware support for SHA256 is present.

`kStatus_Fail` Error. CAU hardware support for SHA256 is not present.

`kStatus_InvalidArgument` Error. `sha256State` is NULL.

### 27.6.32 `status_t MMCAU_SHA256_HashN ( const uint8_t * input, uint32_t numBlocks, uint32_t * sha256State )`

This function updates SHA256 state variables for one or more input message blocks.

Parameters

|                      |                          |                                                                           |
|----------------------|--------------------------|---------------------------------------------------------------------------|
|                      | <code>input</code>       | Pointer to start of input message data.                                   |
|                      | <code>numBlocks</code>   | Number of 512-bit blocks to process.                                      |
| <code>in, out</code> | <code>sha256State</code> | Pointer to 256-bit block of SHA2 state variables: a, b, c, d, e, f, g, h. |

Note

Input message and digest output blocks must not overlap. The [MMCAU\\_SHA256\\_InitializeOutput\(\)](#) function must be called when starting a new hash. Useful when handling non-contiguous input message blocks.

### 27.6.33 `status_t MMCAU_SHA256_Update ( const uint8_t * input, uint32_t numBlocks, uint32_t * sha256State )`

This function updates the SHA256 state variables for one or more input message blocks. It starts a new hash as it internally calls [cau\\_sha256\\_initialize\\_output\(\)](#) first.

Parameters

|                  |                          |                                                                           |
|------------------|--------------------------|---------------------------------------------------------------------------|
|                  | <code>input</code>       | Pointer to start of input message data.                                   |
|                  | <code>numBlocks</code>   | Number of 512-bit blocks to process.                                      |
| <code>out</code> | <code>sha256State</code> | Pointer to 256-bit block of SHA2 state variables: a, b, c, d, e, f, g, h. |

## Function Documentation

### Note

Input message and digest output blocks must not overlap. The [MMCAU\\_SHA256\\_InitializeOutput\(\)](#) function is not required to be called, as it is called internally to start a new hash. All input message blocks must be contiguous.

# Chapter 28

## MIPI CSI2 RX: MIPI CSI2 RX Driver

### 28.1 Overview

The MCUXpresso SDK provides a peripheral driver for the MIPI CSI-2 RX.

### Data Structures

- struct `csi2rx_config_t`  
*CSI2RX configuration.* [More...](#)

### Enumerations

- enum `_csi2rx_data_lane` {  
  `kCSI2RX_DataLane0` = (1U << 0U),  
  `kCSI2RX_DataLane1` = (1U << 1U),  
  `kCSI2RX_DataLane2` = (1U << 2U),  
  `kCSI2RX_DataLane3` = (1U << 3U) }  
*CSI2RX data lanes.*
- enum `_csi2rx_payload` {

## Overview

```
kCSI2RX_PayloadGroup0Null = (1U << 0U),
kCSI2RX_PayloadGroup0Blank = (1U << 1U),
kCSI2RX_PayloadGroup0Embedded = (1U << 2U),
kCSI2RX_PayloadGroup0YUV420_8Bit = (1U << 10U),
kCSI2RX_PayloadGroup0YUV422_8Bit = (1U << 14U),
kCSI2RX_PayloadGroup0YUV422_10Bit = (1U << 15U),
kCSI2RX_PayloadGroup0RGB444 = (1U << 16U),
kCSI2RX_PayloadGroup0RGB555 = (1U << 17U),
kCSI2RX_PayloadGroup0RGB565 = (1U << 18U),
kCSI2RX_PayloadGroup0RGB666 = (1U << 19U),
kCSI2RX_PayloadGroup0RGB888 = (1U << 20U),
kCSI2RX_PayloadGroup0Raw6 = (1U << 24U),
kCSI2RX_PayloadGroup0Raw7 = (1U << 25U),
kCSI2RX_PayloadGroup0Raw8 = (1U << 26U),
kCSI2RX_PayloadGroup0Raw10 = (1U << 27U),
kCSI2RX_PayloadGroup0Raw12 = (1U << 28U),
kCSI2RX_PayloadGroup0Raw14 = (1U << 29U),
kCSI2RX_PayloadGroup1UserDefined1 = (1U << 0U),
kCSI2RX_PayloadGroup1UserDefined2 = (1U << 1U),
kCSI2RX_PayloadGroup1UserDefined3 = (1U << 2U),
kCSI2RX_PayloadGroup1UserDefined4 = (1U << 3U),
kCSI2RX_PayloadGroup1UserDefined5 = (1U << 4U),
kCSI2RX_PayloadGroup1UserDefined6 = (1U << 5U),
kCSI2RX_PayloadGroup1UserDefined7 = (1U << 6U),
kCSI2RX_PayloadGroup1UserDefined8 = (1U << 7U) }
```

*CSI2RX payload type.*

- enum `_csi2rx_bit_error` {  
  `kCSI2RX_BitErrorEccTwoBit` = (1U << 0U),  
  `kCSI2RX_BitErrorEccOneBit` = (1U << 1U) }

*MIPI CSI2RX bit errors.*

- enum `csi2rx_ppi_error_t` {  
  `kCSI2RX_PpiErrorSotHs`,  
  `kCSI2RX_PpiErrorSotSyncHs`,  
  `kCSI2RX_PpiErrorEsc`,  
  `kCSI2RX_PpiErrorSyncEsc`,  
  `kCSI2RX_PpiErrorControl` }

*MIPI CSI2RX PPI error types.*

- enum `_csi2rx_interrupt`  
*MIPI CSI2RX interrupt.*
- enum `_csi2rx_ulps_status` {

```

kCSI2RX_ClockLaneUlps = (1U << 0U),
kCSI2RX_DataLane0Ulps = (1U << 1U),
kCSI2RX_DataLane1Ulps = (1U << 2U),
kCSI2RX_DataLane2Ulps = (1U << 3U),
kCSI2RX_DataLane3Ulps = (1U << 4U),
kCSI2RX_ClockLaneMark = (1U << 5U),
kCSI2RX_DataLane0Mark = (1U << 6U),
kCSI2RX_DataLane1Mark = (1U << 7U),
kCSI2RX_DataLane2Mark = (1U << 8U),
kCSI2RX_DataLane3Mark = (1U << 9U) }

```

*MIPI CSI2RX D-PHY ULPS state.*

## Functions

- void [CSI2RX\\_Init](#) (MIPI\_CSI2RX\_Type \*base, const [csi2rx\\_config\\_t](#) \*config)  
*Enables and configures the CSI2RX peripheral module.*
- void [CSI2RX\\_Deinit](#) (MIPI\_CSI2RX\_Type \*base)  
*Disables the CSI2RX peripheral module.*
- static uint32\_t [CSI2RX\\_GetBitError](#) (MIPI\_CSI2RX\_Type \*base)  
*Gets the MIPI CSI2RX bit error status.*
- static uint32\_t [CSI2RX\\_GetEccBitErrorPosition](#) (uint32\_t bitError)  
*Get ECC one bit error bit position.*
- static uint32\_t [CSI2RX\\_GetUlpsStatus](#) (MIPI\_CSI2RX\_Type \*base)  
*Gets the MIPI CSI2RX D-PHY ULPS status.*
- static uint32\_t [CSI2RX\\_GetPpiErrorDataLanes](#) (MIPI\_CSI2RX\_Type \*base, [csi2rx\\_ppi\\_error\\_t](#) errorType)  
*Gets the MIPI CSI2RX D-PHY PPI error lanes.*
- static void [CSI2RX\\_EnableInterrupts](#) (MIPI\_CSI2RX\_Type \*base, uint32\_t mask)  
*Enable the MIPI CSI2RX interrupts.*
- static void [CSI2RX\\_DisableInterrupts](#) (MIPI\_CSI2RX\_Type \*base, uint32\_t mask)  
*Disable the MIPI CSI2RX interrupts.*
- static uint32\_t [CSI2RX\\_GetInterruptStatus](#) (MIPI\_CSI2RX\_Type \*base)  
*Get the MIPI CSI2RX interrupt status.*

## Driver version

- #define [FSL\\_CSI2RX\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*CSI2RX driver version 2.0.0.*

## 28.2 Data Structure Documentation

### 28.2.1 struct [csi2rx\\_config\\_t](#)

#### Data Fields

- uint8\_t [laneNum](#)  
*Number of active lanes used for receiving data.*
- uint8\_t [tHsSettle\\_EscClk](#)  
*Number of rx\_clk\_esc clock periods for T\_HS\_SETTLE.*

## Enumeration Type Documentation

### 28.2.1.0.0.85 Field Documentation

28.2.1.0.0.85.1 `uint8_t csi2rx_config_t::laneNum`

28.2.1.0.0.85.2 `uint8_t csi2rx_config_t::tHsSettle_EscClk`

The T\_HS\_SETTLE should be in the range of 85ns + 6UI to 145ns + 10UI.

## 28.3 Macro Definition Documentation

28.3.1 `#define FSL_CSI2RX_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

## 28.4 Enumeration Type Documentation

### 28.4.1 enum \_csi2rx\_data\_lane

Enumerator

*kCSI2RX\_DataLane0* Data lane 0.  
*kCSI2RX\_DataLane1* Data lane 1.  
*kCSI2RX\_DataLane2* Data lane 2.  
*kCSI2RX\_DataLane3* Data lane 3.

### 28.4.2 enum \_csi2rx\_payload

Enumerator

*kCSI2RX\_PayloadGroup0Null* NULL.  
*kCSI2RX\_PayloadGroup0Blank* Blank.  
*kCSI2RX\_PayloadGroup0Embedded* Embedded.  
*kCSI2RX\_PayloadGroup0YUV420\_8Bit* Legacy YUV420 8 bit.  
*kCSI2RX\_PayloadGroup0YUV422\_8Bit* YUV422 8 bit.  
*kCSI2RX\_PayloadGroup0YUV422\_10Bit* YUV422 10 bit.  
*kCSI2RX\_PayloadGroup0RGB444* RGB444.  
*kCSI2RX\_PayloadGroup0RGB555* RGB555.  
*kCSI2RX\_PayloadGroup0RGB565* RGB565.  
*kCSI2RX\_PayloadGroup0RGB666* RGB666.  
*kCSI2RX\_PayloadGroup0RGB888* RGB888.  
*kCSI2RX\_PayloadGroup0Raw6* Raw 6.  
*kCSI2RX\_PayloadGroup0Raw7* Raw 7.  
*kCSI2RX\_PayloadGroup0Raw8* Raw 8.  
*kCSI2RX\_PayloadGroup0Raw10* Raw 10.  
*kCSI2RX\_PayloadGroup0Raw12* Raw 12.  
*kCSI2RX\_PayloadGroup0Raw14* Raw 14.  
*kCSI2RX\_PayloadGroup1UserDefined1* User defined 8-bit data type 1, 0x30.

|                                          |                                       |
|------------------------------------------|---------------------------------------|
| <i>kCSI2RX_PayloadGroup1UserDefined2</i> | User defined 8-bit data type 2, 0x31. |
| <i>kCSI2RX_PayloadGroup1UserDefined3</i> | User defined 8-bit data type 3, 0x32. |
| <i>kCSI2RX_PayloadGroup1UserDefined4</i> | User defined 8-bit data type 4, 0x33. |
| <i>kCSI2RX_PayloadGroup1UserDefined5</i> | User defined 8-bit data type 5, 0x34. |
| <i>kCSI2RX_PayloadGroup1UserDefined6</i> | User defined 8-bit data type 6, 0x35. |
| <i>kCSI2RX_PayloadGroup1UserDefined7</i> | User defined 8-bit data type 7, 0x36. |
| <i>kCSI2RX_PayloadGroup1UserDefined8</i> | User defined 8-bit data type 8, 0x37. |

#### 28.4.3 enum \_csi2rx\_bit\_error

Enumerator

*kCSI2RX\_BitErrorEccTwoBit* ECC two bit error has occurred.

*kCSI2RX\_BitErrorEccOneBit* ECC one bit error has occurred.

#### 28.4.4 enum csi2rx\_ppi\_error\_t

Enumerator

*kCSI2RX\_PpiErrorSotHs* CSI2RX DPHY PPI error ErrSotHS.

*kCSI2RX\_PpiErrorSotSyncHs* CSI2RX DPHY PPI error ErrSotSync\_HS.

*kCSI2RX\_PpiErrorEsc* CSI2RX DPHY PPI error ErrEsc.

*kCSI2RX\_PpiErrorSyncEsc* CSI2RX DPHY PPI error ErrSyncEsc.

*kCSI2RX\_PpiErrorControl* CSI2RX DPHY PPI error ErrControl.

#### 28.4.5 enum \_csi2rx\_interrupt

#### 28.4.6 enum \_csi2rx\_ulps\_status

Enumerator

*kCSI2RX\_ClockLaneUlps* Clock lane is in ULPS state.

*kCSI2RX\_DataLane0Ulps* Data lane 0 is in ULPS state.

*kCSI2RX\_DataLane1Ulps* Data lane 1 is in ULPS state.

*kCSI2RX\_DataLane2Ulps* Data lane 2 is in ULPS state.

*kCSI2RX\_DataLane3Ulps* Data lane 3 is in ULPS state.

*kCSI2RX\_ClockLaneMark* Clock lane is in mark state.

*kCSI2RX\_DataLane0Mark* Data lane 0 is in mark state.

*kCSI2RX\_DataLane1Mark* Data lane 1 is in mark state.

*kCSI2RX\_DataLane2Mark* Data lane 2 is in mark state.

*kCSI2RX\_DataLane3Mark* Data lane 3 is in mark state.

## Function Documentation

### 28.5 Function Documentation

28.5.1 `void CSI2RX_Init ( MIPI_CSI2RX_Type * base, const csi2rx_config_t * config )`

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | CSI2RX peripheral address.             |
| <i>config</i> | CSI2RX module configuration structure. |

**28.5.2 void CSI2RX\_Deinit ( MIPI\_CSI2RX\_Type \* *base* )**

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | CSI2RX peripheral address. |
|-------------|----------------------------|

**28.5.3 static uint32\_t CSI2RX\_GetBitError ( MIPI\_CSI2RX\_Type \* *base* )  
[inline], [static]**

This function gets the RX bit error status, the return value could be compared with `_csi2rx_bit_error`. If one bit ECC error detected, the return value could be passed to the function `CSI2RX_GetEccBitErrorPosition` to get the position of the ECC error bit.

Example:

```
uint32_t bitError;
uint32_t bitErrorPosition;

// Get the bit errors.
bitError = CSI2RX_GetBitError(MIPI_CSI2RX);

if (kCSI2RX_BitErrorEccTwoBit & bitError)
{
 // Two bits error;
}
else if (kCSI2RX_BitErrorEccOneBit & bitError)
{
 // One bits error;
 bitErrorPosition = CSI2RX_GetEccBitErrorPosition(bitError);
}
```

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | CSI2RX peripheral address. |
|-------------|----------------------------|

## Returns

The RX bit error status.

## Function Documentation

**28.5.4 static uint32\_t CSI2RX\_GetEccBitErrorPosition ( uint32\_t *bitError* )  
[inline], [static]**

If [CSI2RX\\_GetBitError](#) detects ECC one bit error, this function could extract the error bit position from the return value of [CSI2RX\\_GetBitError](#).

## Parameters

|                 |                                                                |
|-----------------|----------------------------------------------------------------|
| <i>bitError</i> | The bit error returned by <a href="#">CSI2RX_GetBitError</a> . |
|-----------------|----------------------------------------------------------------|

## Returns

The position of error bit.

### 28.5.5 static uint32\_t CSI2RX\_GetUlpsStatus ( MIPI\_CSI2RX\_Type \* *base* ) [inline], [static]

## Example:

```
// Check whether data lane 0 is in ULPS status.
uint32_t status = CSI2RX_GetUlpsStatus(MIPI_CSI2RX);

if (kCSI2RX_DataLane0Ulps & status)
{
 // Data lane 0 is in ULPS status.
}
```

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | CSI2RX peripheral address. |
|-------------|----------------------------|

## Returns

The MIPI CSI2RX D-PHY ULPS status, it is OR'ed value or [\\_csi2rx\\_ulps\\_status](#).

### 28.5.6 static uint32\_t CSI2RX\_GetPpiErrorDataLanes ( MIPI\_CSI2RX\_Type \* *base*, csi2rx\_ppi\_error\_t *errorType* ) [inline], [static]

This function checks the PPI error occurred on which data lanes, the returned value is OR'ed value of [csi2rx\\_ppi\\_error\\_t](#). For example, if the ErrSotHS is detected, to check the ErrSotHS occurred on which data lanes, use like this:

```
uint32_t errorDataLanes = CSI2RX_GetPpiErrorDataLanes(MIPI_CSI2RX,
 kCSI2RX_PpiErrorSotHs);

if (kCSI2RX_DataLane0 & errorDataLanes)
{
 // ErrSotHS occurred on data lane 0.
}

if (kCSI2RX_DataLane1 & errorDataLanes)
{
 // ErrSotHS occurred on data lane 1.
}
```

## Function Documentation

Parameters

|                  |                              |
|------------------|------------------------------|
| <i>base</i>      | CSI2RX peripheral address.   |
| <i>errorType</i> | What kind of error to check. |

Returns

The data lane mask that error *errorType* occurred.

### 28.5.7 static void CSI2RX\_EnableInterrupts ( MIPI\_CSI2RX\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function enables the MIPI CSI2RX interrupts. The interrupts to enable are passed in as an OR'ed value of [\\_csi2rx\\_interrupt](#). For example, to enable one bit and two bit ECC error interrupts, use like this:

```
CSI2RX_EnableInterrupts(MIPI_CSI2RX, kCSI2RX_InterruptEccOneBitError |
kCSI2RX_InterruptEccTwoBitError);
```

Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>base</i> | CSI2RX peripheral address.                         |
| <i>mask</i> | OR'ed value of <a href="#">_csi2rx_interrupt</a> . |

### 28.5.8 static void CSI2RX\_DisableInterrupts ( MIPI\_CSI2RX\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function disables the MIPI CSI2RX interrupts. The interrupts to disable are passed in as an OR'ed value of [\\_csi2rx\\_interrupt](#). For example, to disable one bit and two bit ECC error interrupts, use like this:

```
CSI2RX_DisableInterrupts(MIPI_CSI2RX, kCSI2RX_InterruptEccOneBitError |
kCSI2RX_InterruptEccTwoBitError);
```

Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | CSI2RX peripheral address. |
|-------------|----------------------------|

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>mask</i> | OR'ed value of <a href="#">_csi2rx_interrupt</a> . |
|-------------|----------------------------------------------------|

### 28.5.9 static uint32\_t CSI2RX\_GetInterruptStatus ( MIPI\_CSI2RX\_Type \* *base* ) [inline], [static]

This function returns the MIPI CSI2RX interrupts status as an OR'ed value of [\\_csi2rx\\_interrupt](#).

Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | CSI2RX peripheral address. |
|-------------|----------------------------|

Returns

OR'ed value of [\\_csi2rx\\_interrupt](#).

## Function Documentation

# Chapter 29

## MIPI DSI Driver

### 29.1 Overview

The MCUXpresso SDK provides a peripheral driver for the MIPI DSI

The MIPI DSI driver supports both video mode and command mode. For both modes, first call [DSI\\_Init](#) and [DSI\\_InitDphy](#) to initialize the module and enable the D-PHY. The DSI driver provides function [DSI\\_GetDphyDefaultConfig](#) to help with the D-PHY timing parameter calculation. With the input txHsBitClk frequency and txEscClk frequency, the function can generate the timing parameters based on the D-PHY specification. The user can use the parameter directly, or change them according to the special device.

For the command mode, DSI driver provides polling method and interrupt method for the data transfer. At the same time, there are also small functional APIs so that user can construct them for their special purpose.

When the peripheral is configured through command mode, the video mode can be started by [DSI\\_SetDpiConfig](#).

### Data Structures

- struct [dsi\\_config\\_t](#)  
*MIPI DSI controller configuration. [More...](#)*
- struct [dsi\\_dpi\\_config\\_t](#)  
*MIPI DSI controller DPI interface configuration. [More...](#)*
- struct [dsi\\_dphy\\_config\\_t](#)  
*MIPI DSI D-PHY configuration. [More...](#)*
- struct [dsi\\_transfer\\_t](#)  
*Structure for the data transfer. [More...](#)*
- struct [dsi\\_handle\\_t](#)  
*MIPI DSI transfer handle structure. [More...](#)*

### Typedefs

- typedef void(\* [dsi\\_callback\\_t](#))[\(MIPI\\_DSI\\_HOST\\_Type \\*base, dsi\\_handle\\_t \\*handle, status\\_t status, void \\*userData\)](#)  
*MIPI DSI callback for finished transfer.*

### Enumerations

- enum [\\_dsi\\_status](#) {  
    [kStatus\\_DSI\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_MIPI\_DSI, 0),  
    [kStatus\\_DSI\\_RxDataError](#) = MAKE\_STATUS(kStatusGroup\_MIPI\_DSI, 1),  
    [kStatus\\_DSI\\_ErrorReportReceived](#) = MAKE\_STATUS(kStatusGroup\_MIPI\_DSI, 2),  
    [kStatus\\_DSI\\_NotSupported](#) = MAKE\_STATUS(kStatusGroup\_MIPI\_DSI, 3) }

## Overview

*Error codes for the MIPI DSI driver.*

- enum `dsi_dpi_color_coding_t` {  
    kDSI\_Dpi16BitConfig1 = 0U,  
    kDSI\_Dpi16BitConfig2 = 1U,  
    kDSI\_Dpi16BitConfig3 = 2U,  
    kDSI\_Dpi18BitConfig1 = 3U,  
    kDSI\_Dpi18BitConfig2 = 4U,  
    kDSI\_Dpi24Bit = 5U }

*MIPI DPI interface color coding.*

- enum `dsi_dpi_pixel_packet_t` {  
    kDSI\_PixelPacket16Bit = 0U,  
    kDSI\_PixelPacket18Bit = 1U,  
    kDSI\_PixelPacket18BitLoosely = 2U,  
    kDSI\_PixelPacket24Bit = 3U }

*MIPI DSI pixel packet type send through DPI interface.*

- enum `_dsi_dpi_polarity_flag` {  
    kDSI\_DpiVsyncActiveLow = 0U,  
    kDSI\_DpiHsyncActiveLow = 0U,  
    kDSI\_DpiVsyncActiveHigh = (1U << 0U),  
    kDSI\_DpiHsyncActiveHigh = (1U << 1U) }

*DPI signal polarity.*

- enum `dsi_dpi_video_mode_t` {  
    kDSI\_DpiNonBurstWithSyncPulse = 0U,  
    kDSI\_DpiNonBurstWithSyncEvent = 1U,  
    kDSI\_DpiBurst = 2U }

*DPI video mode.*

- enum `dsi_dpi_bllp_mode_t` {  
    kDSI\_DpiBllpLowPower,  
    kDSI\_DpiBllpBlanking,  
    kDSI\_DpiBllpNull }

*Behavior in BLLP (Blanking or Low-Power Interval).*

- enum `_dsi_apb_status` {  
    kDSI\_ApbNotIdle = (1U << 0U),  
    kDSI\_ApbTxDone = (1U << 1U),  
    kDSI\_ApbRxControl = (1U << 2U),  
    kDSI\_ApbTxOverflow = (1U << 3U),  
    kDSI\_ApbTxUnderflow = (1U << 4U),  
    kDSI\_ApbRxOverflow = (1U << 5U),  
    kDSI\_ApbRxUnderflow = (1U << 6U),  
    kDSI\_ApbRxHeaderReceived = (1U << 7U),  
    kDSI\_ApbRxPacketReceived = (1U << 8U) }

*Status of APB to packet interface.*

- enum `_dsi_rx_error_status` {

```

kDSI_RxErrorEccOneBit = (1U << 0U),
kDSI_RxErrorEccMultiBit = (1U << 1U),
kDSI_RxErrorCrc = (1U << 7U),
kDSI_RxErrorHtxTo = (1U << 8U),
kDSI_RxErrorLrxTo = (1U << 9U),
kDSI_RxErrorBtaTo = (1U << 10U) }
```

*Host receive error status.*

- enum \_dsi\_host\_status {
   
kDSI\_HostSoTError = (1U << 0U),
 kDSI\_HostSoTSyncError = (1U << 1U),
 kDSI\_HostEoTSyncError = (1U << 2U),
 kDSI\_HostEscEntryCmdError = (1U << 3U),
 kDSI\_HostLpTxSyncError = (1U << 4U),
 kDSI\_HostPeriphToError = (1U << 5U),
 kDSI\_HostFalseControlError = (1U << 6U),
 kDSI\_HostContentionDetected = (1U << 7U),
 kDSI\_HostEccErrorOneBit = (1U << 8U),
 kDSI\_HostEccErrorMultiBit = (1U << 9U),
 kDSI\_HostChecksumError = (1U << 10U),
 kDSI\_HostInvalidDataType = (1U << 11U),
 kDSI\_HostInvalidVcId = (1U << 12U),
 kDSI\_HostInvalidTxLength = (1U << 13U),
 kDSI\_HostProtocolViolation = (1U << 15U),
 kDSI\_HostResetTriggerReceived = (1U << 16U),
 kDSI\_HostTearTriggerReceived = (1U << 17U),
 kDSI\_HostAckTriggerReceived = (1U << 18U) }

*DSI host controller status (status\_out)*

- enum \_dsi\_interrupt {

## Overview

```
kDSI_InterruptGroup1ApbNotIdle = (1U << 0U),
kDSI_InterruptGroup1ApbTxDone = (1U << 1U),
kDSI_InterruptGroup1ApbRxControl = (1U << 2U),
kDSI_InterruptGroup1ApbTxOverflow = (1U << 3U),
kDSI_InterruptGroup1ApbTxUnderflow = (1U << 4U),
kDSI_InterruptGroup1ApbRxOverflow = (1U << 5U),
kDSI_InterruptGroup1ApbRxUnderflow = (1U << 6U),
kDSI_InterruptGroup1ApbRxHeaderReceived = (1U << 7U),
kDSI_InterruptGroup1ApbRxPacketReceived = (1U << 8U),
kDSI_InterruptGroup1SoTError = (1U << 9U),
kDSI_InterruptGroup1SoTSyncError = (1U << 10U),
kDSI_InterruptGroup1EoTSyncError = (1U << 11U),
kDSI_InterruptGroup1EscEntryCmdError = (1U << 12U),
kDSI_InterruptGroup1LpTxSyncError = (1U << 13U),
kDSI_InterruptGroup1PeriphToError = (1U << 14U),
kDSI_InterruptGroup1FalseControlError = (1U << 15U),
kDSI_InterruptGroup1ContentionDetected = (1U << 16U),
kDSI_InterruptGroup1EccErrorOneBit = (1U << 17U),
kDSI_InterruptGroup1EccErrorMultiBit = (1U << 18U),
kDSI_InterruptGroup1ChecksumError = (1U << 19U),
kDSI_InterruptGroup1InvalidDataType = (1U << 20U),
kDSI_InterruptGroup1InvalidVcId = (1U << 21U),
kDSI_InterruptGroup1InvalidTxLength = (1U << 22U),
kDSI_InterruptGroup1ProtocalViolation = (1U << 24U),
kDSI_InterruptGroup1ResetTriggerReceived = (1U << 25U),
kDSI_InterruptGroup1TearTriggerReceived = (1U << 26U),
kDSI_InterruptGroup1AckTriggerReceived = (1U << 27U),
kDSI_InterruptGroup1BtaTo = (1U << 29U),
kDSI_InterruptGroup1LrxTo = (1U << 30U),
kDSI_InterruptGroup1HtxTo = (1U << 31U),
kDSI_InterruptGroup2EccOneBit = (1U << 0U),
kDSI_InterruptGroup2EccMultiBit = (1U << 1U),
kDSI_InterruptGroup2CrcError = (1U << 2U) }
```

*DSI interrupt.*

- enum `dsi_tx_data_type_t` {

```

kDSI_TxDataVsyncStart = 0x01U,
kDSI_TxDataVsyncEnd = 0x11U,
kDSI_TxDataHsyncStart = 0x21U,
kDSI_TxDataHsyncEnd = 0x31U,
kDSI_TxDataEoTp = 0x08U,
kDSI_TxDataCmOff = 0x02U,
kDSI_TxDataCmOn = 0x12U,
kDSI_TxDataShutDownPeriph = 0x22U,
kDSI_TxDataTurnOnPeriph = 0x32U,
kDSI_TxDataGenShortWrNoParam = 0x03U,
kDSI_TxDataGenShortWrOneParam = 0x13U,
kDSI_TxDataGenShortWrTwoParam = 0x23U,
kDSI_TxDataGenShortRdNoParam = 0x04U,
kDSI_TxDataGenShortRdOneParam = 0x14U,
kDSI_TxDataGenShortRdTwoParam = 0x24U,
kDSI_TxDataDcsShortWrNoParam = 0x05U,
kDSI_TxDataDcsShortWrOneParam = 0x15U,
kDSI_TxDataDcsShortRdNoParam = 0x06U,
kDSI_TxDataSetMaxReturnPktSize = 0x37U,
kDSI_TxDataNull = 0x09U,
kDSI_TxDataBlanking = 0x19U,
kDSI_TxDataGenLongWr = 0x29U,
kDSI_TxDataDcsLongWr = 0x39U,
kDSI_TxDataLooselyPackedPixel20BitYCbCr = 0x0CU,
kDSI_TxDataPackedPixel24BitYCbCr = 0x1CU,
kDSI_TxDataPackedPixel16BitYCbCr = 0x2CU,
kDSI_TxDataPackedPixel30BitRGB = 0x0DU,
kDSI_TxDataPackedPixel36BitRGB = 0x1DU,
kDSI_TxDataPackedPixel12BitYCrCb = 0x3DU,
kDSI_TxDataPackedPixel16BitRGB = 0x0EU,
kDSI_TxDataPackedPixel18BitRGB = 0x1EU,
kDSI_TxDataLooselyPackedPixel18BitRGB = 0x2EU,
kDSI_TxDataPackedPixel24BitRGB = 0x3EU }

```

*DSI TX data type.*

- enum `dsi_rx_data_type_t` {
 

```

kDSI_RxDataAckAndErrorReport = 0x02U,
kDSI_RxDataEoTp = 0x08U,
kDSI_RxDataGenShortRdResponseOneByte = 0x11U,
kDSI_RxDataGenShortRdResponseTwoByte = 0x12U,
kDSI_RxDataGenLongRdResponse = 0x1AU,
kDSI_RxDataDcsLongRdResponse = 0x1CU,
kDSI_RxDataDcsShortRdResponseOneByte = 0x21U,
kDSI_RxDataDcsShortRdResponseTwoByte = 0x22U }

```

*DSI RX data type.*

- enum `_dsi_transfer_flags` {

## Overview

```
kDSI_TransferUseHighSpeed = (1U << 0U),
kDSI_TransferPerformBTA = (1U << 1U) }
DSI transfer control flags.
```

## Driver version

- #define **FSL\_MIPI\_DSI\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

## MIPI\_DSI host initialization.

- void **DSI\_Init** (MIPI\_DSI\_HOST\_Type \*base, const **dsi\_config\_t** \*config)  
*Initializes an MIPI DSI host with the user configuration.*
- void **DSI\_Deinit** (MIPI\_DSI\_HOST\_Type \*base)  
*Deinitializes an MIPI DSI host.*
- void **DSI\_GetDefaultConfig** (**dsi\_config\_t** \*config)  
*Get the default configuration to initialize the MIPI DSI host.*

## DPI interface

- void **DSI\_SetDpiConfig** (MIPI\_DSI\_HOST\_Type \*base, const **dpi\_config\_t** \*config, uint8\_t numLanes, uint32\_t dpiPixelClkFreq\_Hz, uint32\_t dsiHsBitClkFreq\_Hz)  
*Configure the DPI interface core.*

## D-PHY configuration.

- uint32\_t **DSI\_InitDphy** (MIPI\_DSI\_HOST\_Type \*base, const **dphy\_config\_t** \*config, uint32\_t refClkFreq\_Hz)  
*Initializes the D-PHY.*
- void **DSI\_DeinitDphy** (MIPI\_DSI\_HOST\_Type \*base)  
*Deinitializes the D-PHY.*
- void **DSI\_GetDphyDefaultConfig** (**dphy\_config\_t** \*config, uint32\_t txHsBitClk\_Hz, uint32\_t txEscClk\_Hz)  
*Get the default D-PHY configuration.*

## Interrupts

- static void **DSI\_EnableInterrupts** (MIPI\_DSI\_HOST\_Type \*base, uint32\_t intGroup1, uint32\_t intGroup2)  
*Enable the interrupts.*
- static void **DSI\_DisableInterrupts** (MIPI\_DSI\_HOST\_Type \*base, uint32\_t intGroup1, uint32\_t intGroup2)  
*Disable the interrupts.*
- static void **DSI\_GetAndClearInterruptStatus** (MIPI\_DSI\_HOST\_Type \*base, uint32\_t \*intGroup1, uint32\_t \*intGroup2)  
*Get and clear the interrupt status.*

## MIPI DSI APB

- void [DSI\\_SetApbPacketControl](#) (MIPI\_DSI\_HOST\_Type \*base, uint16\_t wordCount, uint8\_t virtualChannel, [dsi\\_tx\\_data\\_type\\_t](#) dataType, uint8\_t flags)
 

*Configure the APB packet to send.*
- void [DSI\\_WriteApbTxPayload](#) (MIPI\_DSI\_HOST\_Type \*base, const uint8\_t \*payload, uint16\_t payloadSize)
 

*Fill the long APB packet payload.*
- void [DSI\\_ReadApbRxPayload](#) (MIPI\_DSI\_HOST\_Type \*base, uint8\_t \*payload, uint16\_t payloadSize)
 

*Read the long APB packet payload.*
- static void [DSI\\_SendApbPacket](#) (MIPI\_DSI\_HOST\_Type \*base)
 

*Trigger the controller to send out APB packet.*
- static uint32\_t [DSI\\_GetApbStatus](#) (MIPI\_DSI\_HOST\_Type \*base)
 

*Get the APB status.*
- static uint32\_t [DSI\\_GetRxErrorStatus](#) (MIPI\_DSI\_HOST\_Type \*base)
 

*Get the error status during data transfer.*
- static uint8\_t [DSI\\_GetEccRxErrorPosition](#) (uint32\_t rxErrorStatus)
 

*Get the one-bit RX ECC error position.*
- static uint32\_t [DSI\\_GetAndClearHostStatus](#) (MIPI\_DSI\_HOST\_Type \*base)
 

*Get and clear the DSI host status.*
- static uint32\_t [DSI\\_GetRxPacketHeader](#) (MIPI\_DSI\_HOST\_Type \*base)
 

*Get the RX packet header.*
- static [dsi\\_rx\\_data\\_type\\_t](#) [DSI\\_GetRxPacketType](#) (uint32\_t rxPktHeader)
 

*Extract the RX packet type from the packet header.*
- static uint16\_t [DSI\\_GetRxPacketWordCount](#) (uint32\_t rxPktHeader)
 

*Extract the RX packet word count from the packet header.*
- static uint8\_t [DSI\\_GetRxPacketVirtualChannel](#) (uint32\_t rxPktHeader)
 

*Extract the RX packet virtual channel from the packet header.*
- status\_t [DSI\\_TransferBlocking](#) (MIPI\_DSI\_HOST\_Type \*base, [dsi\\_transfer\\_t](#) \*xfer)
 

*APB data transfer using blocking method.*

## Transactional

- status\_t [DSI\\_TransferCreateHandle](#) (MIPI\_DSI\_HOST\_Type \*base, [dsi\\_handle\\_t](#) \*handle, [dsi\\_callback\\_t](#) callback, void \*userData)
 

*Create the MIPI DSI handle.*
- status\_t [DSI\\_TransferNonBlocking](#) (MIPI\_DSI\_HOST\_Type \*base, [dsi\\_handle\\_t](#) \*handle, [dsi\\_transfer\\_t](#) \*xfer)
 

*APB data transfer using interrupt method.*
- void [DSI\\_TransferAbort](#) (MIPI\_DSI\_HOST\_Type \*base, [dsi\\_handle\\_t](#) \*handle)
 

*Abort current APB data transfer.*
- void [DSI\\_TransferHandleIRQ](#) (MIPI\_DSI\_HOST\_Type \*base, [dsi\\_handle\\_t](#) \*handle)
 

*Interrupt handler for the DSI.*

## Data Structure Documentation

### 29.2 Data Structure Documentation

#### 29.2.1 struct dsi\_config\_t

##### Data Fields

- `uint8_t numLanes`  
*Number of lanes.*
- `bool enableNonContinuousHsClk`  
*In enabled, the high speed clock will enter low power mode between transmissions.*
- `bool enableTxUlps`  
*Enable the TX ULPS.*
- `bool autoInsertEoTp`  
*Insert an EoTp short package when switching from HS to LP.*
- `uint8_t numExtraEoTp`  
*How many extra EoTp to send after the end of a packet.*
- `uint32_t htxTo_ByteClk`  
*HS TX timeout count (HTX\_TO) in byte clock.*
- `uint32_t lrxHostTo_ByteClk`  
*LP RX host timeout count (LRX-H\_TO) in byte clock.*
- `uint32_t btaTo_ByteClk`  
*Bus turn around timeout count (TA\_TO) in byte clock.*

##### 29.2.1.0.0.86 Field Documentation

###### 29.2.1.0.0.86.1 `uint8_t dsi_config_t::numLanes`

###### 29.2.1.0.0.86.2 `bool dsi_config_t::enableNonContinuousHsClk`

###### 29.2.1.0.0.86.3 `bool dsi_config_t::enableTxUlps`

###### 29.2.1.0.0.86.4 `bool dsi_config_t::autoInsertEoTp`

###### 29.2.1.0.0.86.5 `uint8_t dsi_config_t::numExtraEoTp`

###### 29.2.1.0.0.86.6 `uint32_t dsi_config_t::htxTo_ByteClk`

###### 29.2.1.0.0.86.7 `uint32_t dsi_config_t::lrxHostTo_ByteClk`

###### 29.2.1.0.0.86.8 `uint32_t dsi_config_t::btaTo_ByteClk`

#### 29.2.2 struct dsi\_dpi\_config\_t

##### Data Fields

- `uint16_t pixelPayloadSize`  
*Maximum number of pixels that should be sent as one DSI packet.*
- `dsi_dpi_color_coding_t dpiColorCoding`  
*DPI color coding.*
- `dsi_dpi_pixel_packet_t pixelPacket`

- *Pixel packet format.*
- `dsi_dpi_video_mode_t` `videoMode`
  - Video mode.*
- `dsi_dpi_bllp_mode_t` `bllpMode`
  - Behavior in BLLP.*
- `uint8_t` `polarityFlags`
  - OR'ed value of \_dsi\_dpi\_polarity\_flag controls signal polarity.*
- `uint16_t` `hfp`
  - Horizontal front porch, in dpi pixel clock.*
- `uint16_t` `hbp`
  - Horizontal back porch, in dpi pixel clock.*
- `uint16_t` `hsw`
  - Horizontal sync width, in dpi pixel clock.*
- `uint8_t` `vfp`
  - Number of lines in vertical front porch.*
- `uint8_t` `vbp`
  - Number of lines in vertical back porch.*
- `uint16_t` `panelHeight`
  - Line number in vertical active area.*
- `uint8_t` `virtualChannel`
  - Virtual channel.*

## 29.2.2.0.0.87 Field Documentation

### 29.2.2.0.0.87.1 `uint16_t dsi_dpi_config_t::pixelPayloadSize`

Recommended that the line size (in pixels) is evenly divisible by this parameter.

## Data Structure Documentation

- 29.2.2.0.0.87.2 `dsi_dpi_color_coding_t dsi_dpi_config_t::dpiColorCoding`
- 29.2.2.0.0.87.3 `dsi_dpi_pixel_packet_t dsi_dpi_config_t::pixelPacket`
- 29.2.2.0.0.87.4 `dsi_dpi_video_mode_t dsi_dpi_config_t::videoMode`
- 29.2.2.0.0.87.5 `dsi_dpi_bllp_mode_t dsi_dpi_config_t::bllpMode`
- 29.2.2.0.0.87.6 `uint8_t dsi_dpi_config_t::polarityFlags`
- 29.2.2.0.0.87.7 `uint16_t dsi_dpi_config_t::hfp`
- 29.2.2.0.0.87.8 `uint16_t dsi_dpi_config_t::hbp`
- 29.2.2.0.0.87.9 `uint16_t dsi_dpi_config_t::hsw`
- 29.2.2.0.0.87.10 `uint8_t dsi_dpi_config_t::vfp`
- 29.2.2.0.0.87.11 `uint8_t dsi_dpi_config_t::vbp`
- 29.2.2.0.0.87.12 `uint16_t dsi_dpi_config_t::panelHeight`
- 29.2.2.0.0.87.13 `uint8_t dsi_dpi_config_t::virtualChannel`

### 29.2.3 struct `dsi_dphy_config_t`

#### Data Fields

- `uint32_t txHsBitClk_Hz`  
*The generated HS TX bit clock in Hz.*
- `uint8_t tClkPre_ByteClk`  
*TCLK-PRE in byte clock.*
- `uint8_t tClkPost_ByteClk`  
*TCLK-POST in byte clock.*
- `uint8_t tHsExit_ByteClk`  
*THS-EXIT in byte clock.*
- `uint32_t tWakeup_EscClk`  
*Number of clk\_esc clock periods to keep a clock or data lane in Mark-1 state after exiting ULPS.*
- `uint8_t tHsPrepare_HalfEscClk`  
*THS-PREPARE in clk\_esc/2.*
- `uint8_t tClkPrepare_HalfEscClk`  
*TCLK-PREPARE in clk\_esc/2.*
- `uint8_t tHsZero_ByteClk`  
*THS-ZERO in clk\_byte.*
- `uint8_t tClkZero_ByteClk`  
*TCLK-ZERO in clk\_byte.*
- `uint8_t tHsTrail_ByteClk`  
*THS-TRAIL in clk\_byte.*
- `uint8_t tClkTrail_ByteClk`  
*TCLK-TRAIL in clk\_byte.*

### 29.2.3.0.0.88 Field Documentation

**29.2.3.0.0.88.1 uint32\_t dsi\_dphy\_config\_t::txHsBitClk\_Hz**

**29.2.3.0.0.88.2 uint8\_t dsi\_dphy\_config\_t::tClkPre\_ByteClk**

Set how long the controller will wait after enabling clock lane for HS before enabling data lanes for HS.

**29.2.3.0.0.88.3 uint8\_t dsi\_dphy\_config\_t::tClkPost\_ByteClk**

Set how long the controller will wait before putting clock lane into LP mode after data lanes detected in stop state.

**29.2.3.0.0.88.4 uint8\_t dsi\_dphy\_config\_t::tHsExit\_ByteClk**

Set how long the controller will wait after the clock lane has been put into LP mode before enabling clock lane for HS again.

**29.2.3.0.0.88.5 uint32\_t dsi\_dphy\_config\_t::tWakeup\_EscClk**

**29.2.3.0.0.88.6 uint8\_t dsi\_dphy\_config\_t::tHsPrepare\_HalfEscClk**

Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3, 4, 5.

**29.2.3.0.0.88.7 uint8\_t dsi\_dphy\_config\_t::tClkPrepare\_HalfEscClk**

Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3.

**29.2.3.0.0.88.8 uint8\_t dsi\_dphy\_config\_t::tHsZero\_ByteClk**

Set how long that controller drives data lane HS-0 state before transmit the Sync sequence. Available values are 6, 7, ..., 37.

**29.2.3.0.0.88.9 uint8\_t dsi\_dphy\_config\_t::tClkZero\_ByteClk**

Set how long that controller drives clock lane HS-0 state before transmit the Sync sequence. Available values are 3, 4, ..., 66.

**29.2.3.0.0.88.10 uint8\_t dsi\_dphy\_config\_t::tHsTrail\_ByteClk**

Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

**29.2.3.0.0.88.11 uint8\_t dsi\_dphy\_config\_t::tClkTrail\_ByteClk**

Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

## Data Structure Documentation

### 29.2.4 struct dsi\_transfer\_t

#### Data Fields

- uint8\_t `virtualChannel`  
*Virtual channel.*
- `dsi_tx_data_type_t txDataType`  
*TX data type.*
- uint8\_t `flags`  
*Flags to control the transfer, see `_dsi_transfer_flags`.*
- const uint8\_t \* `txData`  
*The TX data buffer.*
- uint8\_t \* `rxData`  
*The RX data buffer.*
- uint16\_t `txDataSize`  
*Size of the TX data.*
- uint16\_t `rxDataSize`  
*Size of the RX data.*

#### 29.2.4.0.0.89 Field Documentation

##### 29.2.4.0.0.89.1 uint8\_t `dsi_transfer_t::virtualChannel`

##### 29.2.4.0.0.89.2 `dsi_tx_data_type_t dsi_transfer_t::txDataType`

##### 29.2.4.0.0.89.3 uint8\_t `dsi_transfer_t::flags`

##### 29.2.4.0.0.89.4 const uint8\_t\* `dsi_transfer_t::txData`

##### 29.2.4.0.0.89.5 uint8\_t\* `dsi_transfer_t::rxData`

##### 29.2.4.0.0.89.6 uint16\_t `dsi_transfer_t::txDataSize`

##### 29.2.4.0.0.89.7 uint16\_t `dsi_transfer_t::rxDataSize`

### 29.2.5 struct \_dsi\_handle

MIPI DSI transfer handle.

#### Data Fields

- volatile bool `isBusy`  
*MIPI DSI is busy with APB data transfer.*
- `dsi_transfer_t xfer`  
*Transfer information.*
- `dsi_callback_t callback`  
*DSI callback.*
- void \* `userData`

*Callback parameter.*

### 29.2.5.0.0.90 Field Documentation

#### 29.2.5.0.0.90.1 volatile bool dsi\_handle\_t::isBusy

#### 29.2.5.0.0.90.2 dsi\_transfer\_t dsi\_handle\_t::xfer

## 29.3 Typedef Documentation

### 29.3.1 **typedef void(\* dsi\_callback\_t)(MIPI\_DSI\_HOST\_Type \*base, dsi\_handle\_t \*handle, status\_t status, void \*userData)**

When transfer finished, one of these status values will be passed to the user:

- kStatus\_Success Data transfer finished with no error.
- kStatus\_Timeout Transfer failed because of timeout.
- **kStatus\_DSI\_RxDataError** RX data error, user could use [DSI\\_GetRxErrorStatus](#) to check the error details.
- **kStatus\_DSI\_ErrorReportReceived** Error Report packet received, user could use [DSI\\_GetAndClearHostStatus](#) to check the error report status.
- kStatus\_Fail Transfer failed for other reasons.

## 29.4 Enumeration Type Documentation

### 29.4.1 enum \_dsi\_status

Enumerator

***kStatus\_DSI\_Busy*** DSI is busy.

***kStatus\_DSI\_RxDataError*** Read data error.

***kStatus\_DSI\_ErrorReportReceived*** Error report package received.

***kStatus\_DSI\_NotSupported*** The transfer type not supported.

### 29.4.2 enum dsi\_dpi\_color\_coding\_t

Enumerator

***kDSI\_Dpi16BitConfig1*** 16-bit configuration 1. RGB565: XXXXXXXX\_RRRRGGG\_GGGBB-BBB.

***kDSI\_Dpi16BitConfig2*** 16-bit configuration 2. RGB565: XXXRRRRR\_XXGGGGGG\_XXXBB-BBB.

***kDSI\_Dpi16BitConfig3*** 16-bit configuration 3. RGB565: XXRRRRRX\_XXGGGGGG\_XXBBB-BBX.

***kDSI\_Dpi18BitConfig1*** 18-bit configuration 1. RGB666: XXXXXRR\_RRRRGGGG\_GGBBB-BBB.

## Enumeration Type Documentation

*kDSI\_Dpi18BitConfig2* 18-bit configuration 2. RGB666: XXRRRRRR\_XXGGGGGG\_XXBBB-BBB.  
*kDSI\_Dpi24Bit* 24-bit.

### 29.4.3 enum dsi\_dpi\_pixel\_packet\_t

Enumerator

*kDSI\_PixelPacket16Bit* 16 bit RGB565.  
*kDSI\_PixelPacket18Bit* 18 bit RGB666 packed.  
*kDSI\_PixelPacket18BitLoosely* 18 bit RGB666 loosely packed into three bytes.  
*kDSI\_PixelPacket24Bit* 24 bit RGB888, each pixel uses three bytes.

### 29.4.4 enum \_dsi\_dpi\_polarity\_flag

Enumerator

*kDSI\_DpiVsyncActiveLow* VSYNC active low.  
*kDSI\_DpiHsyncActiveLow* HSYNC active low.  
*kDSI\_DpiVsyncActiveHigh* VSYNC active high.  
*kDSI\_DpiHsyncActiveHigh* HSYNC active high.

### 29.4.5 enum dsi\_dpi\_video\_mode\_t

Enumerator

*kDSI\_DpiNonBurstWithSyncPulse* Non-Burst mode with Sync Pulses.  
*kDSI\_DpiNonBurstWithSyncEvent* Non-Burst mode with Sync Events.  
*kDSI\_DpiBurst* Burst mode.

### 29.4.6 enum dsi\_dpi\_bllp\_mode\_t

Enumerator

*kDSI\_DpiBllpLowPower* LP mode used in BLLP periods.  
*kDSI\_DpiBllpBlanking* Blanking packets used in BLLP periods.  
*kDSI\_DpiBllpNull* Null packets used in BLLP periods.

### 29.4.7 enum \_dsi\_apb\_status

Enumerator

*kDSI\_ApbNotIdle* State machine not idle.  
*kDSI\_ApbTxDone* Tx packet done.  
*kDSI\_ApbRxControl* DPHY direction 0 - tx had control, 1 - rx has control.  
*kDSI\_ApbTxOverflow* TX fifo overflow.  
*kDSI\_ApbTxUnderflow* TX fifo underflow.  
*kDSI\_ApbRxOverflow* RX fifo overflow.  
*kDSI\_ApbRxUnderflow* RX fifo underflow.  
*kDSI\_ApbRxHeaderReceived* RX packet header has been received.  
*kDSI\_ApbRxPacketReceived* All RX packet payload data has been received.

### 29.4.8 enum \_dsi\_rx\_error\_status

Enumerator

*kDSI\_RxErrorEccOneBit* ECC single bit error detected.  
*kDSI\_RxErrorEccMultiBit* ECC multi bit error detected.  
*kDSI\_RxErrorCrc* CRC error detected.  
*kDSI\_RxErrorHtxTo* High Speed forward TX timeout detected.  
*kDSI\_RxErrorLrxTo* Reverse Low power data receive timeout detected.  
*kDSI\_RxErrorBtaTo* BTA timeout detected.

### 29.4.9 enum \_dsi\_host\_status

Enumerator

*kDSI\_HostSoTError* SoT error from peripheral error report.  
*kDSI\_HostSoTSyncError* SoT Sync error from peripheral error report.  
*kDSI\_HostEoTSyncError* EoT Sync error from peripheral error report.  
*kDSI\_HostEscEntryCmdError* Escape Mode Entry Command Error from peripheral error report.  
*kDSI\_HostLpTxSyncError* Low-power transmit Sync Error from peripheral error report.  
*kDSI\_HostPeriphToError* Peripheral timeout error from peripheral error report.  
*kDSI\_HostFalseControlError* False control error from peripheral error report.  
*kDSI\_HostContentionDetected* Contention detected from peripheral error report.  
*kDSI\_HostEccErrorOneBit* Single bit ECC error (corrected) from peripheral error report.  
*kDSI\_HostEccErrorMultiBit* Multi bit ECC error (not corrected) from peripheral error report.  
*kDSI\_HostChecksumError* Checksum error from peripheral error report.  
*kDSI\_HostInvalidDataType* DSI data type not recognized.  
*kDSI\_HostInvalidVcId* DSI VC ID invalid.  
*kDSI\_HostInvalidTxLength* Invalid transmission length.

## Enumeration Type Documentation

*kDSI\_HostProtocolViolation* DSI protocol violation.

*kDSI\_HostResetTriggerReceived* Reset trigger received.

*kDSI\_HostTearTriggerReceived* Tear effect trigger receive.

*kDSI\_HostAckTriggerReceived* Acknowledge trigger message received.

### 29.4.10 enum \_dsi\_interrupt

Enumerator

*kDSI\_InterruptGroup1ApbNotIdle* State machine not idle.

*kDSI\_InterruptGroup1ApbTxDone* Tx packet done.

*kDSI\_InterruptGroup1ApbRxControl* DPHY direction 0 - tx control, 1 - rx control.

*kDSI\_InterruptGroup1ApbTxOverflow* TX fifo overflow.

*kDSI\_InterruptGroup1ApbTxUnderflow* TX fifo underflow.

*kDSI\_InterruptGroup1ApbRxOverflow* RX fifo overflow.

*kDSI\_InterruptGroup1ApbRxUnderflow* RX fifo underflow.

*kDSI\_InterruptGroup1ApbRxHeaderReceived* RX packet header has been received.

*kDSI\_InterruptGroup1ApbRxPacketReceived* All RX packet payload data has been received.

*kDSI\_InterruptGroup1SoTError* SoT error from peripheral error report.

*kDSI\_InterruptGroup1SoTSyncError* SoT Sync error from peripheral error report.

*kDSI\_InterruptGroup1EoTSyncError* EoT Sync error from peripheral error report.

*kDSI\_InterruptGroup1EscEntryCmdError* Escape Mode Entry Command Error from peripheral error report.

*kDSI\_InterruptGroup1LpTxSyncError* Low-power transmit Sync Error from peripheral error report.

*kDSI\_InterruptGroup1PeriphToError* Peripheral timeout error from peripheral error report.

*kDSI\_InterruptGroup1FalseControlError* False control error from peripheral error report.

*kDSI\_InterruptGroup1ContentionDetected* Contention detected from peripheral error report.

*kDSI\_InterruptGroup1EccErrorOneBit* Single bit ECC error (corrected) from peripheral error report.

*kDSI\_InterruptGroup1EccErrorMultiBit* Multi bit ECC error (not corrected) from peripheral error report.

*kDSI\_InterruptGroup1ChecksumError* Checksum error from peripheral error report.

*kDSI\_InterruptGroup1InvalidDataType* DSI data type not recognized.

*kDSI\_InterruptGroup1InvalidVcId* DSI VC ID invalid.

*kDSI\_InterruptGroup1InvalidTxLength* Invalid transmission length.

*kDSI\_InterruptGroup1ProtocolViolation* DSI protocol violation.

*kDSI\_InterruptGroup1ResetTriggerReceived* Reset trigger received.

*kDSI\_InterruptGroup1TearTriggerReceived* Tear effect trigger receive.

*kDSI\_InterruptGroup1AckTriggerReceived* Acknowledge trigger message received.

*kDSI\_InterruptGroup1BtaTo* Host BTA timeout.

*kDSI\_InterruptGroup1LrxTo* Low power RX timeout.

*kDSI\_InterruptGroup1HtxTo* High speed TX timeout.

*kDSI\_InterruptGroup2EccOneBit* Sinle bit ECC error.

***kDSI\_InterruptGroup2EccMultiBit*** Multi bit ECC error.

***kDSI\_InterruptGroup2CrcError*** CRC error.

#### 29.4.11 enum dsi\_tx\_data\_type\_t

Enumerator

***kDSI\_TxDataVsyncStart*** V Sync start.

***kDSI\_TxDataVsyncEnd*** V Sync end.

***kDSI\_TxDataHsyncStart*** H Sync start.

***kDSI\_TxDataHsyncEnd*** H Sync end.

***kDSI\_TxDataEoTp*** End of transmission packet.

***kDSI\_TxDataCmOff*** Color mode off.

***kDSI\_TxDataCmOn*** Color mode on.

***kDSI\_TxDataShutDownPeriph*** Shut down peripheral.

***kDSI\_TxDataTurnOnPeriph*** Turn on peripheral.

***kDSI\_TxDataGenShortWrNoParam*** Generic Short WRITE, no parameters.

***kDSI\_TxDataGenShortWrOneParam*** Generic Short WRITE, one parameter.

***kDSI\_TxDataGenShortWrTwoParam*** Generic Short WRITE, two parameter.

***kDSI\_TxDataGenShortRdNoParam*** Generic Short READ, no parameters.

***kDSI\_TxDataGenShortRdOneParam*** Generic Short READ, one parameter.

***kDSI\_TxDataGenShortRdTwoParam*** Generic Short READ, two parameter.

***kDSI\_TxDataDcsShortWrNoParam*** DCS Short WRITE, no parameters.

***kDSI\_TxDataDcsShortWrOneParam*** DCS Short WRITE, one parameter.

***kDSI\_TxDataDcsShortRdNoParam*** DCS Short READ, no parameters.

***kDSI\_TxDataSetMaxReturnPktSize*** Set the Maximum Return Packet Size.

***kDSI\_TxDataNull*** Null Packet, no data.

***kDSI\_TxDataBlanking*** Blanking Packet, no data.

***kDSI\_TxDataGenLongWr*** Generic long write.

***kDSI\_TxDataDcsLongWr*** DCS Long Write/write\_LUT Command Packet.

***kDSI\_TxDataLooselyPackedPixel20BitYCbCr*** Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format.

***kDSI\_TxDataPackedPixel24BitYCbCr*** Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format.

***kDSI\_TxDataPackedPixel16BitYCbCr*** Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format.

***kDSI\_TxDataPackedPixel30BitRGB*** Packed Pixel Stream, 30-bit RGB, 10-10-10 Format.

***kDSI\_TxDataPackedPixel36BitRGB*** Packed Pixel Stream, 36-bit RGB, 12-12-12 Format.

***kDSI\_TxDataPackedPixel12BitYCrCb*** Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format.

***kDSI\_TxDataPackedPixel16BitRGB*** Packed Pixel Stream, 16-bit RGB, 5-6-5 Format.

***kDSI\_TxDataPackedPixel18BitRGB*** Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

***kDSI\_TxDataLooselyPackedPixel18BitRGB*** Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.

***kDSI\_TxDataPackedPixel24BitRGB*** Packed Pixel Stream, 24-bit RGB, 8-8-8 Format.

## Function Documentation

### 29.4.12 enum dsi\_rx\_data\_type\_t

Enumerator

*kDSI\_RxDataAckAndErrorReport* Acknowledge and Error Report.

*kDSI\_RxDataEoTp* End of Transmission packet.

*kDSI\_RxDataGenShortRdResponseOneByte* Generic Short READ Response, 1 byte returned.

*kDSI\_RxDataGenShortRdResponseTwoByte* Generic Short READ Response, 2 byte returned.

*kDSI\_RxDataGenLongRdResponse* Generic Long READ Response.

*kDSI\_RxDataDcsLongRdResponse* DCS Long READ Response.

*kDSI\_RxDataDcsShortRdResponseOneByte* DCS Short READ Response, 1 byte returned.

*kDSI\_RxDataDcsShortRdResponseTwoByte* DCS Short READ Response, 2 byte returned.

### 29.4.13 enum \_dsi\_transfer\_flags

Enumerator

*kDSI\_TransferUseHighSpeed* Use high speed mode or not.

*kDSI\_TransferPerformBTA* Perform BTA or not.

## 29.5 Function Documentation

### 29.5.1 void DSI\_Init ( MIPI\_DSI\_HOST\_Type \* *base*, const dsi\_config\_t \* *config* )

This function initializes the MIPI DSI host with the configuration, it should be called first before other MIPI DSI driver functions.

Parameters

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

### 29.5.2 void DSI\_Deinit ( MIPI\_DSI\_HOST\_Type \* *base* )

This function should be called after all bother MIPI DSI driver functions.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | MIPI DSI host peripheral base address. |
|-------------|----------------------------------------|

### 29.5.3 void DSI\_GetDefaultConfig ( *dsi\_config\_t* \* *config* )

The default value is:

```
config->numLanes = 4;
config->enableNonContinuousHsClk = false;
config->enableTxUlps = false;
config->autoInsertEoTp = true;
config->numExtraEoTp = 0;
config->htxTo_ByteClk = 0;
config->lrxHostTo_ByteClk = 0;
config->btaTo_ByteClk = 0;
```

Parameters

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

### 29.5.4 void DSI\_SetDpiConfig ( *MIPI\_DSI\_HOST\_Type* \* *base*, *const dsi\_dpi\_config\_t* \* *config*, *uint8\_t numLanes*, *uint32\_t dpiPixelClkFreq\_Hz*, *uint32\_t dsiHsBitClkFreq\_Hz* )

This function sets the DPI interface configuration, it should be used in video mode.

Parameters

|                            |                                                                                    |
|----------------------------|------------------------------------------------------------------------------------|
| <i>base</i>                | MIPI DSI host peripheral base address.                                             |
| <i>config</i>              | Pointer to the DPI interface configuration.                                        |
| <i>numLanes</i>            | Lane number, should be same with the setting in <i>dsi_dpi_config_t</i> .          |
| <i>dpiPixelClk-Freq_Hz</i> | The DPI pixel clock frequency in Hz.                                               |
| <i>dsiHsBitClk-Freq_Hz</i> | The DSI high speed bit clock frequency in Hz. It is the same with DPHY PLL output. |

### 29.5.5 *uint32\_t DSI\_InitDphy ( *MIPI\_DSI\_HOST\_Type* \* *base*, *const dsi\_dphy\_config\_t* \* *config*, *uint32\_t refClkFreq\_Hz* )*

This function configures the D-PHY timing and setups the D-PHY PLL based on user configuration. The configuration structure could be got by the function [DSI\\_GetDphyDefaultConfig](#).

## Function Documentation

Parameters

|                      |                                        |
|----------------------|----------------------------------------|
| <i>base</i>          | MIPI DSI host peripheral base address. |
| <i>config</i>        | Pointer to the D-PHY configuration.    |
| <i>refClkFreq_Hz</i> | The REFCLK frequency in Hz.            |

Returns

The actual D-PHY PLL output frequency. If could not configure the PLL to the target frequency, the return value is 0.

### 29.5.6 void DSI\_DeinitDphy ( MIPI\_DSI\_HOST\_Type \* *base* )

Power down the D-PHY PLL and shut down D-PHY.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | MIPI DSI host peripheral base address. |
|-------------|----------------------------------------|

### 29.5.7 void DSI\_GetDphyDefaultConfig ( dsi\_dphy\_config\_t \* *config*, uint32\_t *txHsBitClk\_Hz*, uint32\_t *txEscClk\_Hz* )

Gets the default D-PHY configuration, the timing parameters are set according to D-PHY specification. User could use the configuration directly, or change some parameters according to the special device.

Parameters

|                      |                                     |
|----------------------|-------------------------------------|
| <i>config</i>        | Pointer to the D-PHY configuration. |
| <i>txHsBitClk_Hz</i> | High speed bit clock in Hz.         |
| <i>txEscClk_Hz</i>   | Esc clock in Hz.                    |

### 29.5.8 static void DSI\_EnableInterrupts ( MIPI\_DSI\_HOST\_Type \* *base*, uint32\_t *intGroup1*, uint32\_t *intGroup2* ) [inline], [static]

The interrupts to enable are passed in as OR'ed mask value of [\\_dsi\\_interrupt](#).

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | MIPI DSI host peripheral base address. |
| <i>intGroup1</i> | Interrupts to enable in group 1.       |
| <i>intGroup2</i> | Interrupts to enable in group 2.       |

### 29.5.9 static void DSI\_DisableInterrupts ( MIPI\_DSI\_HOST\_Type \* *base*, uint32\_t *intGroup1*, uint32\_t *intGroup2* ) [inline], [static]

The interrupts to disable are passed in as OR'ed mask value of [\\_dsi\\_interrupt](#).

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | MIPI DSI host peripheral base address. |
| <i>intGroup1</i> | Interrupts to disable in group 1.      |
| <i>intGroup2</i> | Interrupts to disable in group 2.      |

### 29.5.10 static void DSI\_GetAndClearInterruptStatus ( MIPI\_DSI\_HOST\_Type \* *base*, uint32\_t \* *intGroup1*, uint32\_t \* *intGroup2* ) [inline], [static]

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | MIPI DSI host peripheral base address. |
| <i>intGroup1</i> | Group 1 interrupt status.              |
| <i>intGroup2</i> | Group 2 interrupt status.              |

### 29.5.11 void DSI\_SetApbPacketControl ( MIPI\_DSI\_HOST\_Type \* *base*, uint16\_t *wordCount*, uint8\_t *virtualChannel*, dsi\_tx\_data\_type\_t *dataType*, uint8\_t *flags* )

This function configures the next APB packet transfer. After configuration, the packet transfer could be started with function [DSI\\_SendApbPacket](#). If the packet is long packet, Use [DSI\\_WriteApbTxPayload](#) to fill the payload before start transfer.

## Function Documentation

Parameters

|                       |                                                                                                         |
|-----------------------|---------------------------------------------------------------------------------------------------------|
| <i>base</i>           | MIPI DSI host peripheral base address.                                                                  |
| <i>wordCount</i>      | For long packet, this is the byte count of the payload. For short packet, this is (data1 << 8)   data0. |
| <i>virtualChannel</i> | Virtual channel.                                                                                        |
| <i>dataType</i>       | The packet data type, (DI).                                                                             |
| <i>flags</i>          | The transfer control flags, see <a href="#">_dsi_transfer_flags</a> .                                   |

**29.5.12 void DSI\_WriteApbTxPayload ( MIPI\_DSI\_HOST\_Type \* *base*, const uint8\_t \* *payload*, uint16\_t *payloadSize* )**

Write the long packet payload to TX FIFO.

Parameters

|                    |                                        |
|--------------------|----------------------------------------|
| <i>base</i>        | MIPI DSI host peripheral base address. |
| <i>payload</i>     | Pointer to the payload.                |
| <i>payloadSize</i> | Payload size in byte.                  |

**29.5.13 void DSI\_ReadApbRxPayload ( MIPI\_DSI\_HOST\_Type \* *base*, uint8\_t \* *payload*, uint16\_t *payloadSize* )**

Read the long packet payload from RX FIFO. This function reads directly but does not check the RX FIFO status. Upper layer should make sure there are available data.

Parameters

|                    |                                        |
|--------------------|----------------------------------------|
| <i>base</i>        | MIPI DSI host peripheral base address. |
| <i>payload</i>     | Pointer to the payload.                |
| <i>payloadSize</i> | Payload size in byte.                  |

**29.5.14 static void DSI\_SendApbPacket ( MIPI\_DSI\_HOST\_Type \* *base* )  
[inline], [static]**

Send the packet set by [DSI\\_SetApbPacketControl](#).

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | MIPI DSI host peripheral base address. |
|-------------|----------------------------------------|

### 29.5.15 static uint32\_t DSI\_GetApbStatus ( MIPI\_DSI\_HOST\_Type \* *base* ) [inline], [static]

The return value is OR'ed value of [\\_dsi\\_apb\\_status](#).

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | MIPI DSI host peripheral base address. |
|-------------|----------------------------------------|

Returns

The APB status.

### 29.5.16 static uint32\_t DSI\_GetRxErrorStatus ( MIPI\_DSI\_HOST\_Type \* *base* ) [inline], [static]

The return value is OR'ed value of [\\_dsi\\_rx\\_error\\_status](#).

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | MIPI DSI host peripheral base address. |
|-------------|----------------------------------------|

Returns

The error status.

### 29.5.17 static uint8\_t DSI\_GetEccRxErrorPosition ( uint32\_t *rxErrorStatus* ) [inline], [static]

When one-bit ECC RX error detected using [DSI\\_GetRxErrorStatus](#), this function could be used to get the error bit position.

```
uint8_t eccErrorPos;
uint32_t rxErrorStatus = DSI_GetRxErrorStatus(MIPI_DSI);
if (kDSI_RxErrorEccOneBit & rxErrorStatus)
{
 eccErrorPos = DSI_GetEccRxErrorPosition(rxErrorStatus);
}
```

## Function Documentation

Parameters

|                      |                                                                     |
|----------------------|---------------------------------------------------------------------|
| <i>rxErrorStatus</i> | The error status returned by <a href="#">DSI_GetRxErrorStatus</a> . |
|----------------------|---------------------------------------------------------------------|

Returns

The 1-bit ECC error position.

### 29.5.18 static uint32\_t DSI\_GetAndClearHostStatus ( MIPI\_DSI\_HOST\_Type \* *base* ) [inline], [static]

The host status are returned as mask value of [\\_dsi\\_host\\_status](#).

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | MIPI DSI host peripheral base address. |
|-------------|----------------------------------------|

Returns

The DSI host status.

### 29.5.19 static uint32\_t DSI\_GetRxPacketHeader ( MIPI\_DSI\_HOST\_Type \* *base* ) [inline], [static]

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | MIPI DSI host peripheral base address. |
|-------------|----------------------------------------|

Returns

The RX packet header.

### 29.5.20 static dsi\_rx\_data\_type\_t DSI\_GetRxPacketType ( uint32\_t *rxPktHeader* ) [inline], [static]

Extract the RX packet type from the packet header get by [DSI\\_GetRxPacketHeader](#).

Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>rxPktHeader</i> | The RX packet header get by <a href="#">DSI_GetRxPacketHeader</a> . |
|--------------------|---------------------------------------------------------------------|

Returns

The RX packet type.

### 29.5.21 static uint16\_t DSI\_GetRxPacketWordCount ( uint32\_t *rxPktHeader* ) [inline], [static]

Extract the RX packet word count from the packet header get by [DSI\\_GetRxPacketHeader](#).

Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>rxPktHeader</i> | The RX packet header get by <a href="#">DSI_GetRxPacketHeader</a> . |
|--------------------|---------------------------------------------------------------------|

Returns

For long packet, return the payload word count (byte). For short packet, return the (data0 << 8) | data1.

### 29.5.22 static uint8\_t DSI\_GetRxPacketVirtualChannel ( uint32\_t *rxPktHeader* ) [inline], [static]

Extract the RX packet virtual channel from the packet header get by [DSI\\_GetRxPacketHeader](#).

Parameters

|                    |                                                                     |
|--------------------|---------------------------------------------------------------------|
| <i>rxPktHeader</i> | The RX packet header get by <a href="#">DSI_GetRxPacketHeader</a> . |
|--------------------|---------------------------------------------------------------------|

Returns

The virtual channel.

### 29.5.23 status\_t DSI\_TransferBlocking ( MIPI\_DSI\_HOST\_Type \* *base*, dsi\_transfer\_t \* *xfer* )

Perform APB data transfer using blocking method. This function waits until all data send or received, or timeout happens.

## Function Documentation

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | MIPI DSI host peripheral base address. |
| <i>xfer</i> | Pointer to the transfer structure.     |

Return values

|                                         |                                                                                                                          |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>                  | Data transfer finished with no error.                                                                                    |
| <i>kStatus_Timeout</i>                  | Transfer failed because of timeout.                                                                                      |
| <i>kStatus_DSI_RxData-Error</i>         | RX data error, user could use <a href="#">DSI_GetRxErrorStatus</a> to check the error details.                           |
| <i>kStatus_DSI_Error-ReportReceived</i> | Error Report packet received, user could use <a href="#">DSI_GetAndClearHostStatus</a> to check the error report status. |
| <i>kStatus_DSI_Not-Supported</i>        | Transfer format not supported.                                                                                           |
| <i>kStatus_DSI_Fail</i>                 | Transfer failed for other reasons.                                                                                       |

### 29.5.24 **status\_t DSI\_TransferCreateHandle ( MIPI\_DSI\_HOST\_Type \* *base*, dsi\_handle\_t \* *handle*, dsi\_callback\_t *callback*, void \* *userData* )**

This function initializes the MIPI DSI handle which can be used for other transactional APIs.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | MIPI DSI host peripheral base address. |
| <i>handle</i>   | Handle pointer.                        |
| <i>callback</i> | Callback function.                     |
| <i>userData</i> | User data.                             |

### 29.5.25 **status\_t DSI\_TransferNonBlocking ( MIPI\_DSI\_HOST\_Type \* *base*, dsi\_handle\_t \* *handle*, dsi\_transfer\_t \* *xfer* )**

Perform APB data transfer using interrupt method, when transfer finished, upper layer could be informed through callback function.

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | MIPI DSI host peripheral base address.                             |
| <i>handle</i> | pointer to dsi_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the transfer structure.                                 |

Return values

|                                 |                                                                      |
|---------------------------------|----------------------------------------------------------------------|
| <i>kStatus_Success</i>          | Data transfer started successfully.                                  |
| <i>kStatus_DSI_Busy</i>         | Failed to start transfer because DSI is busy with previous transfer. |
| <i>kStatus_DSI_NotSupported</i> | Transfer format not supported.                                       |

### 29.5.26 void DSI\_TransferAbort ( MIPI\_DSI\_HOST\_Type \* *base*, dsi\_handle\_t \* *handle* )

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | MIPI DSI host peripheral base address.                             |
| <i>handle</i> | pointer to dsi_handle_t structure which stores the transfer state. |

### 29.5.27 void DSI\_TransferHandleIRQ ( MIPI\_DSI\_HOST\_Type \* *base*, dsi\_handle\_t \* *handle* )

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | MIPI DSI host peripheral base address.                             |
| <i>handle</i> | pointer to dsi_handle_t structure which stores the transfer state. |

## Function Documentation

# Chapter 30

## MU: Messaging Unit

### 30.1 Overview

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

### 30.2 Function description

The MU driver provides these functions:

- Functions to initialize the MU module.
- Functions to send and receive messages.
- Functions for MU flags for both MU sides.
- Functions for status flags and interrupts.
- Other miscellaneous functions.

#### 30.2.1 MU initialization

The function [MU\\_Init\(\)](#) initializes the MU module and enables the MU clock. It should be called before any other MU functions.

The function [MU\\_Deinit\(\)](#) deinitializes the MU module and disables the MU clock. No MU functions can be called after this function.

#### 30.2.2 MU message

The MU message must be sent when the transmit register is empty. The MU driver provides blocking API and non-blocking API to send message.

The [MU\\_SendMsgNonBlocking\(\)](#) function writes a message to the MU transmit register without checking the transmit register status. The upper layer should check that the transmit register is empty before calling this function. This function can be used in the ISR for better performance.

The [MU\\_SendMsg\(\)](#) function is a blocking function. It waits until the transmit register is empty and sends the message.

Correspondingly, there are blocking and non-blocking APIs for receiving a message. The [MU\\_ReadMsgNonBlocking\(\)](#) function is a non-blocking API. The [MU\\_ReadMsg\(\)](#) function is the blocking API.

## Function description

### 30.2.3 MU flags

The MU driver provides 3-bit general purpose flags. When the flags are set on one side, they are reflected on the other side.

The MU flags must be set when the previous flags have been updated to the other side. The MU driver provides a non-blocking function and a blocking function. The blocking function [MU\\_SetFlags\(\)](#) waits until previous flags have been updated to the other side and then sets flags. The non-blocking function sets the flags directly. Ensure that the kMU\_FlagsUpdatingFlag is not pending before calling this function.

The function [MU\\_GetFlags\(\)](#) gets the MU flags on the current side.

### 30.2.4 Status and interrupt

The function [MU\\_GetStatusFlags\(\)](#) returns all MU status flags. Use the `_mu_status_flags` to check for specific flags, for example, to check RX0 and RX1 register full, use the following code:

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/mu. The receive full flags are cleared automatically after messages are read out. The transmit empty flags are cleared automatically after new messages are written to the transmit register. The general purpose interrupt flags must be cleared manually using the function [MU\\_ClearStatusFlags\(\)](#).

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/mu. To enable or disable a specific interrupt, use [MU\\_EnableInterrupts\(\)](#) and [MU\\_DisableInterrupts\(\)](#) functions. The interrupts to enable or disable should be passed in as a bit mask of the `_mu_interrupt_enable`.

The [MU\\_TriggerInterrupts\(\)](#) function triggers general purpose interrupts and NMI to the other core. The interrupts to trigger are passed in as a bit mask of the `_mu_interrupt_trigger`. If previously triggered interrupts have not been processed by the other side, this function returns an error.

### 30.2.5 MU misc functions

The [MU\\_BootCoreB\(\)](#) and [MU\\_HoldCoreBReset\(\)](#) functions should only be used from A side. They are used to boot the core B or to hold core B in reset.

The [MU\\_ResetBothSides\(\)](#) function resets MU at both A and B sides. However, only the A side can call this function.

If a core enters stop mode, the platform clock of this core is disabled by default. The function [MU\\_SetClockOnOtherCoreEnable\(\)](#) forces the other core's platform clock to remain enabled even after that core has entered a stop mode. In this case, the other core's platform clock keeps running until the current core enters stop mode too.

Function [MU\\_GetOtherCorePowerMode\(\)](#) gets the power mode of the other core.

## Enumerations

- enum `_mu_status_flags` {
   
`kMU_Tx0EmptyFlag` = (1U << (MU\_SR\_TEEn\_SHIFT + 3U)),
   
`kMU_Tx1EmptyFlag` = (1U << (MU\_SR\_TEEn\_SHIFT + 2U)),
   
`kMU_Tx2EmptyFlag` = (1U << (MU\_SR\_TEEn\_SHIFT + 1U)),
   
`kMU_Tx3EmptyFlag` = (1U << (MU\_SR\_TEEn\_SHIFT + 0U)),
   
`kMU_Rx0FullFlag` = (1U << (MU\_SR\_RFn\_SHIFT + 3U)),
   
`kMU_Rx1FullFlag` = (1U << (MU\_SR\_RFn\_SHIFT + 2U)),
   
`kMU_Rx2FullFlag` = (1U << (MU\_SR\_RFn\_SHIFT + 1U)),
   
`kMU_Rx3FullFlag` = (1U << (MU\_SR\_RFn\_SHIFT + 0U)),
   
`kMU_GenInt0Flag` = (1U << (MU\_SR\_GIPn\_SHIFT + 3U)),
   
`kMU_GenInt1Flag` = (1U << (MU\_SR\_GIPn\_SHIFT + 2U)),
   
`kMU_GenInt2Flag` = (1U << (MU\_SR\_GIPn\_SHIFT + 1U)),
   
`kMU_GenInt3Flag` = (1U << (MU\_SR\_GIPn\_SHIFT + 0U)),
   
`kMU_EventPendingFlag` = MU\_SR\_EP\_MASK,
   
`kMU_FlagsUpdatingFlag` = MU\_SR\_FUP\_MASK }

*MU status flags.*

- enum `_mu_interrupt_enable` {
   
`kMU_Tx0EmptyInterruptEnable` = (1U << (MU\_CR\_TIEEn\_SHIFT + 3U)),
   
`kMU_Tx1EmptyInterruptEnable` = (1U << (MU\_CR\_TIEEn\_SHIFT + 2U)),
   
`kMU_Tx2EmptyInterruptEnable` = (1U << (MU\_CR\_TIEEn\_SHIFT + 1U)),
   
`kMU_Tx3EmptyInterruptEnable` = (1U << (MU\_CR\_TIEEn\_SHIFT + 0U)),
   
`kMU_Rx0FullInterruptEnable` = (1U << (MU\_CR\_RIEEn\_SHIFT + 3U)),
   
`kMU_Rx1FullInterruptEnable` = (1U << (MU\_CR\_RIEEn\_SHIFT + 2U)),
   
`kMU_Rx2FullInterruptEnable` = (1U << (MU\_CR\_RIEEn\_SHIFT + 1U)),
   
`kMU_Rx3FullInterruptEnable` = (1U << (MU\_CR\_RIEEn\_SHIFT + 0U)),
   
`kMU_GenInt0InterruptEnable` = (1U << (MU\_CR\_GIEEn\_SHIFT + 3U)),
   
`kMU_GenInt1InterruptEnable` = (1U << (MU\_CR\_GIEEn\_SHIFT + 2U)),
   
`kMU_GenInt2InterruptEnable` = (1U << (MU\_CR\_GIEEn\_SHIFT + 1U)),
   
`kMU_GenInt3InterruptEnable` = (1U << (MU\_CR\_GIEEn\_SHIFT + 0U)) }

*MU interrupt source to enable.*

- enum `_mu_interrupt_trigger` {
   
`kMU_NmiInterruptTrigger` = MU\_CR\_NMI\_MASK,
   
`kMU_GenInt0InterruptTrigger` = (1U << (MU\_CR\_GIRn\_SHIFT + 3U)),
   
`kMU_GenInt1InterruptTrigger` = (1U << (MU\_CR\_GIRn\_SHIFT + 2U)),
   
`kMU_GenInt2InterruptTrigger` = (1U << (MU\_CR\_GIRn\_SHIFT + 1U)),
   
`kMU_GenInt3InterruptTrigger` = (1U << (MU\_CR\_GIRn\_SHIFT + 0U)) }

*MU interrupt that could be triggered to the other core.*

## Driver version

- #define `FSL_MU_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 2))
   
*MU driver version 2.0.2.*

## Function description

### MU initialization.

- void **MU\_Init** (MU\_Type \*base)  
*Initializes the MU module.*
- void **MU\_Deinit** (MU\_Type \*base)  
*De-initializes the MU module.*

### MU Message

- static void **MU\_SendMsgNonBlocking** (MU\_Type \*base, uint32\_t regIndex, uint32\_t msg)  
*Writes a message to the TX register.*
- void **MU\_SendMsg** (MU\_Type \*base, uint32\_t regIndex, uint32\_t msg)  
*Blocks to send a message.*
- static uint32\_t **MU\_ReceiveMsgNonBlocking** (MU\_Type \*base, uint32\_t regIndex)  
*Reads a message from the RX register.*
- uint32\_t **MU\_ReceiveMsg** (MU\_Type \*base, uint32\_t regIndex)  
*Blocks to receive a message.*

### MU Flags

- static void **MU\_SetFlagsNonBlocking** (MU\_Type \*base, uint32\_t flags)  
*Sets the 3-bit MU flags reflect on the other MU side.*
- void **MU\_SetFlags** (MU\_Type \*base, uint32\_t flags)  
*Blocks setting the 3-bit MU flags reflect on the other MU side.*
- static uint32\_t **MU\_GetFlags** (MU\_Type \*base)  
*Gets the current value of the 3-bit MU flags set by the other side.*

### Status and Interrupt.

- static uint32\_t **MU\_GetStatusFlags** (MU\_Type \*base)  
*Gets the MU status flags.*
- static void **MU\_ClearStatusFlags** (MU\_Type \*base, uint32\_t mask)  
*Clears the specific MU status flags.*
- static void **MU\_EnableInterrupts** (MU\_Type \*base, uint32\_t mask)  
*Enables the specific MU interrupts.*
- static void **MU\_DisableInterrupts** (MU\_Type \*base, uint32\_t mask)  
*Disables the specific MU interrupts.*
- status\_t **MU\_TriggerInterrupts** (MU\_Type \*base, uint32\_t mask)  
*Triggers interrupts to the other core.*
- static void **MU\_ClearNmi** (MU\_Type \*base)  
*Clear non-maskable interrupt (NMI) sent by the other core.*

### MU misc functions

- void **MU\_BootCoreB** (MU\_Type \*base, mu\_core\_boot\_mode\_t mode)  
*Boots the core at B side.*
- static void **MU\_HoldCoreBReset** (MU\_Type \*base)  
*Holds the core reset of B side.*
- void **MU\_BootOtherCore** (MU\_Type \*base, mu\_core\_boot\_mode\_t mode)  
*Boots the other core.*
- static void **MU\_HoldOtherCoreReset** (MU\_Type \*base)

- static void [MU\\_ResetBothSides](#) (MU\_Type \*base)
 

*Holds the other core reset.*

*Resets the MU for both A side and B side.*
- void [MU\\_HardwareResetOtherCore](#) (MU\_Type \*base, bool waitReset, bool holdReset, mu\_core\_boot\_mode\_t bootMode)
 

*Hardware reset the other core.*
- static void [MU\\_SetClockOnOtherCoreEnable](#) (MU\_Type \*base, bool enable)
 

*Enables or disables the clock on the other core.*
- static mu\_power\_mode\_t [MU\\_GetOtherCorePowerMode](#) (MU\_Type \*base)
 

*Gets the power mode of the other core.*

### 30.3 Macro Definition Documentation

#### 30.3.1 #define FSL\_MU\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

### 30.4 Enumeration Type Documentation

#### 30.4.1 enum \_mu\_status\_flags

Enumerator

- kMU\_Tx0EmptyFlag* TX0 empty.
- kMU\_Tx1EmptyFlag* TX1 empty.
- kMU\_Tx2EmptyFlag* TX2 empty.
- kMU\_Tx3EmptyFlag* TX3 empty.
- kMU\_Rx0FullFlag* RX0 full.
- kMU\_Rx1FullFlag* RX1 full.
- kMU\_Rx2FullFlag* RX2 full.
- kMU\_Rx3FullFlag* RX3 full.
- kMU\_GenInt0Flag* General purpose interrupt 0 pending.
- kMU\_GenInt1Flag* General purpose interrupt 0 pending.
- kMU\_GenInt2Flag* General purpose interrupt 0 pending.
- kMU\_GenInt3Flag* General purpose interrupt 0 pending.
- kMU\_EventPendingFlag* MU event pending.
- kMU\_FlagsUpdatingFlag* MU flags update is on-going.

#### 30.4.2 enum \_mu\_interrupt\_enable

Enumerator

- kMU\_Tx0EmptyInterruptEnable* TX0 empty.
- kMU\_Tx1EmptyInterruptEnable* TX1 empty.
- kMU\_Tx2EmptyInterruptEnable* TX2 empty.
- kMU\_Tx3EmptyInterruptEnable* TX3 empty.
- kMU\_Rx0FullInterruptEnable* RX0 full.
- kMU\_Rx1FullInterruptEnable* RX1 full.

## Function Documentation

*kMU\_Rx2FullInterruptEnable* RX2 full.  
*kMU\_Rx3FullInterruptEnable* RX3 full.  
*kMU\_GenInt0InterruptEnable* General purpose interrupt 0.  
*kMU\_GenInt1InterruptEnable* General purpose interrupt 1.  
*kMU\_GenInt2InterruptEnable* General purpose interrupt 2.  
*kMU\_GenInt3InterruptEnable* General purpose interrupt 3.

### 30.4.3 enum \_mu\_interrupt\_trigger

Enumerator

*kMU\_NmiInterruptTrigger* NMI interrupt.  
*kMU\_GenInt0InterruptTrigger* General purpose interrupt 0.  
*kMU\_GenInt1InterruptTrigger* General purpose interrupt 1.  
*kMU\_GenInt2InterruptTrigger* General purpose interrupt 2.  
*kMU\_GenInt3InterruptTrigger* General purpose interrupt 3.

## 30.5 Function Documentation

### 30.5.1 void MU\_Init( MU\_Type \* *base* )

This function enables the MU clock only.

Parameters

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

### 30.5.2 void MU\_Deinit( MU\_Type \* *base* )

This function disables the MU clock only.

Parameters

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

### 30.5.3 static void MU\_SendMsgNonBlocking( MU\_Type \* *base*, uint32\_t *regIndex*, uint32\_t *msg* ) [inline], [static]

This function writes a message to the specific TX register. It does not check whether the TX register is empty or not. The upper layer should make sure the TX register is empty before calling this function. This function can be used in ISR for better performance.

```
* while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { } // Wait for
 TX0 register empty.
* MU_SendMsgNonBlocking(base, 0U, MSG_VAL); // Write message to the TX0 register.
*
```

## Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>base</i>     | MU peripheral base address. |
| <i>regIndex</i> | TX register index.          |
| <i>msg</i>      | Message to send.            |

**30.5.4 void MU\_SendMsg ( MU\_Type \* *base*, uint32\_t *regIndex*, uint32\_t *msg* )**

This function waits until the TX register is empty and sends the message.

## Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>base</i>     | MU peripheral base address. |
| <i>regIndex</i> | TX register index.          |
| <i>msg</i>      | Message to send.            |

**30.5.5 static uint32\_t MU\_ReceiveMsgNonBlocking ( MU\_Type \* *base*, uint32\_t *regIndex* ) [inline], [static]**

This function reads a message from the specific RX register. It does not check whether the RX register is full or not. The upper layer should make sure the RX register is full before calling this function. This function can be used in ISR for better performance.

```
* uint32_t msg;
* while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
* {
* } // Wait for the RX0 register full.
*
* msg = MU_ReceiveMsgNonBlocking(base, 0U); // Read message from RX0 register.
*
```

## Parameters

## Function Documentation

|                 |                             |
|-----------------|-----------------------------|
| <i>base</i>     | MU peripheral base address. |
| <i>regIndex</i> | TX register index.          |

Returns

The received message.

### 30.5.6 `uint32_t MU_ReceiveMsg ( MU_Type * base, uint32_t regIndex )`

This function waits until the RX register is full and receives the message.

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>base</i>     | MU peripheral base address. |
| <i>regIndex</i> | RX register index.          |

Returns

The received message.

### 30.5.7 `static void MU_SetFlagsNonBlocking ( MU_Type * base, uint32_t flags ) [inline], [static]`

This function sets the 3-bit MU flags directly. Every time the 3-bit MU flags are changed, the status flag kMU\_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU\_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. The upper layer should make sure the status flag kMU\_FlagsUpdatingFlag is cleared before calling this function.

```
* while (kMU_FlagsUpdatingFlag & MU_GetStatusFlags(base))
* {
* } // Wait for previous MU flags updating.
*
* MU_SetFlagsNonBlocking(base, 0U); // Set the mU flags.
*
```

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | MU peripheral base address. |
| <i>flags</i> | The 3-bit MU flags to set.  |

### 30.5.8 void MU\_SetFlags ( MU\_Type \* *base*, uint32\_t *flags* )

This function blocks setting the 3-bit MU flags. Every time the 3-bit MU flags are changed, the status flag kMU\_FlagsUpdatingFlag asserts indicating the 3-bit MU flags are updating to the other side. After the 3-bit MU flags are updated, the status flag kMU\_FlagsUpdatingFlag is cleared by hardware. During the flags updating period, the flags cannot be changed. This function waits for the MU status flag kMU\_FlagsUpdatingFlag cleared and sets the 3-bit MU flags.

Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | MU peripheral base address. |
| <i>flags</i> | The 3-bit MU flags to set.  |

### 30.5.9 static uint32\_t MU\_GetFlags ( MU\_Type \* *base* ) [inline], [static]

This function gets the current 3-bit MU flags on the current side.

Parameters

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

Returns

*flags* Current value of the 3-bit flags.

### 30.5.10 static uint32\_t MU\_GetStatusFlags ( MU\_Type \* *base* ) [inline], [static]

This function returns the bit mask of the MU status flags. See \_mu\_status\_flags.

```
* uint32_t flags;
* flags = MU_GetStatusFlags(base); // Get all status flags.
* if (kMU_Tx0EmptyFlag & flags)
* {
* // The TX0 register is empty. Message can be sent.
* MU_SendMsgNonBlocking(base, 0U, MSG0_VAL);
* }
* if (kMU_Tx1EmptyFlag & flags)
* {
```

## Function Documentation

```
* // The TX1 register is empty. Message can be sent.
* MU_SendMsgNonBlocking(base, 1U, MSG1_VAL);
* }
*
```

### Parameters

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

### Returns

Bit mask of the MU status flags, see `_mu_status_flags`.

### 30.5.11 static void MU\_ClearStatusFlags ( MU\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function clears the specific MU status flags. The flags to clear should be passed in as bit mask. See `_mu_status_flags`.

```
* //Clear general interrupt 0 and general interrupt 1 pending flags.
* MU_ClearStatusFlags(base, kMU_GenInt0Flag |
* kMU_GenInt1Flag);
*
```

### Parameters

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | MU peripheral base address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>mask</i> | Bit mask of the MU status flags. See <code>_mu_status_flags</code> . The following flags are cleared by hardware, this function could not clear them. <ul style="list-style-type: none"><li>• kMU_Tx0EmptyFlag</li><li>• kMU_Tx1EmptyFlag</li><li>• kMU_Tx2EmptyFlag</li><li>• kMU_Tx3EmptyFlag</li><li>• kMU_Rx0FullFlag</li><li>• kMU_Rx1FullFlag</li><li>• kMU_Rx2FullFlag</li><li>• kMU_Rx3FullFlag</li><li>• kMU_EventPendingFlag</li><li>• kMU_FlagsUpdatingFlag</li><li>• kMU_OtherSideInResetFlag</li></ul> |

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

This function enables the specific MU interrupts. The interrupts to enable should be passed in as bit mask. See [\\_mu\\_interrupt\\_enable](#).

```
* // Enable general interrupt 0 and TX0 empty interrupt.
* MU_EnableInterrupts(base, kMU_GenInt0InterruptEnable |
 kMU_Tx0EmptyInterruptEnable);
*
```

Parameters

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| <i>base</i> | MU peripheral base address.                                               |
| <i>mask</i> | Bit mask of the MU interrupts. See <a href="#">_mu_interrupt_enable</a> . |

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

This function disables the specific MU interrupts. The interrupts to disable should be passed in as bit mask. See [\\_mu\\_interrupt\\_enable](#).

```
* // Disable general interrupt 0 and TX0 empty interrupt.
* MU_DisableInterrupts(base, kMU_GenInt0InterruptEnable |
 kMU_Tx0EmptyInterruptEnable);
*
```

Parameters

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| <i>base</i> | MU peripheral base address.                                               |
| <i>mask</i> | Bit mask of the MU interrupts. See <a href="#">_mu_interrupt_enable</a> . |

### 30.5.14 status\_t MU\_TriggerInterrupts ( MU\_Type \* *base*, uint32\_t *mask* )

This function triggers the specific interrupts to the other core. The interrupts to trigger are passed in as bit mask. See [\\_mu\\_interrupt\\_trigger](#). The MU should not trigger an interrupt to the other core when the previous interrupt has not been processed by the other core. This function checks whether the previous interrupts have been processed. If not, it returns an error.

```
* if (kStatus_Success != MU_TriggerInterrupts(base,
 kMU_GenInt0InterruptTrigger |
 kMU_GenInt2InterruptTrigger))
{
 // Previous general purpose interrupt 0 or general purpose interrupt 2
 // has not been processed by the other core.
}
```

## Function Documentation

Parameters

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>base</i> | MU peripheral base address.                                                     |
| <i>mask</i> | Bit mask of the interrupts to trigger. See <code>_mu_interrupt_trigger</code> . |

Return values

|                        |                                              |
|------------------------|----------------------------------------------|
| <i>kStatus_Success</i> | Interrupts have been triggered successfully. |
| <i>kStatus_Fail</i>    | Previous interrupts have not been accepted.  |

### 30.5.15 static void MU\_ClearNmi ( MU\_Type \* *base* ) [inline], [static]

This function clears non-maskable interrupt (NMI) sent by the other core.

Parameters

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

### 30.5.16 void MU\_BootCoreB ( MU\_Type \* *base*, mu\_core\_boot\_mode\_t *mode* )

This function sets the B side core's boot configuration and releases the core from reset.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | MU peripheral base address. |
| <i>mode</i> | Core B boot mode.           |

Note

Only MU side A can use this function.

### 30.5.17 static void MU\_HoldCoreBReset ( MU\_Type \* *base* ) [inline], [static]

This function causes the core of B side to be held in reset following any reset event.

Parameters

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

Note

Only A side could call this function.

### 30.5.18 void MU\_BootOtherCore ( MU\_Type \* *base*, mu\_core\_boot\_mode\_t *mode* )

This function boots the other core with a boot configuration.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | MU peripheral base address. |
| <i>mode</i> | The other core boot mode.   |

### 30.5.19 static void MU\_HoldOtherCoreReset ( MU\_Type \* *base* ) [inline], [static]

This function causes the other core to be held in reset following any reset event.

Parameters

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

### 30.5.20 static void MU\_ResetBothSides ( MU\_Type \* *base* ) [inline], [static]

This function resets the MU for both A side and B side. Before reset, it is recommended to interrupt processor B, because this function may affect the ongoing processor B programs.

Parameters

## Function Documentation

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

### Note

For some platforms, only MU side A could use this function, check reference manual for details.

### 30.5.21 void MU\_HardwareResetOtherCore ( MU\_Type \* *base*, bool *waitReset*, bool *holdReset*, mu\_core\_boot\_mode\_t *bootMode* )

This function resets the other core, the other core could mask the hardware reset by calling MU\_MaskHardwareReset. The hardware reset mask feature is only available for some platforms. This function could be used together with MU\_BootOtherCore to control the other core reset workflow.

Example 1: Reset the other core, and no hold reset

```
* MU_HardwareResetOtherCore(MU_A, true, false, bootMode);
*
```

In this example, the core at MU side B will reset with the specified boot mode.

Example 2: Reset the other core and hold it, then boot the other core later.

```
* // Here the other core enters reset, and the reset is hold
* MU_HardwareResetOtherCore(MU_A, true, true, modeDontCare);
* // Current core boot the other core when necessary.
* MU_BootOtherCore(MU_A, bootMode);
*
```

### Parameters

|                  |                                                                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | MU peripheral base address.                                                                                                                                                                                                                                   |
| <i>waitReset</i> | Wait the other core enters reset. <ul style="list-style-type: none"><li>• true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked.</li><li>• false: Don't wait the reset.</li></ul> |

|                  |                                                                                                                                                                                                                                                                                                         |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>holdReset</i> | Hold the other core reset or not. <ul style="list-style-type: none"> <li>• true: Hold the other core in reset, this function returns directly when the other core enters reset.</li> <li>• false: Don't hold the other core in reset, this function waits until the other core out of reset.</li> </ul> |
| <i>bootMode</i>  | Boot mode of the other core, if <i>holdReset</i> is true, this parameter is useless.                                                                                                                                                                                                                    |

### 30.5.22 static void MU\_SetClockOnOtherCoreEnable ( MU\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the platform clock on the other core when that core enters a stop mode. If disabled, the platform clock for the other core is disabled when it enters stop mode. If enabled, the platform clock keeps running on the other core in stop mode, until this core also enters stop mode.

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | MU peripheral base address.                    |
| <i>enable</i> | Enable or disable the clock on the other core. |

### 30.5.23 static mu\_power\_mode\_t MU\_GetOtherCorePowerMode ( MU\_Type \* *base* ) [inline], [static]

This function gets the power mode of the other core.

Parameters

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

Returns

Power mode of the other core.

## Function Documentation

# Chapter 31

## Notification Framework

### 31.1 Overview

This section describes the programming interface of the Notifier driver.

### 31.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

These are the steps for the configuration transition.

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.  
The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending a "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system switches to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application.

```
#include "fsl_notifier.h"

// Definition of the Power Manager callback.
status_t callback0(notifier_notification_block_t *notify, void *data)
{
 status_t ret = kStatus_Success;

 ...
 ...

 return ret;
}
// Definition of the Power Manager user function.
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *userData)
```

## Notifier Overview

```
...
...
...
}
...
...
...
...
...
...
// Main function.
int main(void)
{
 // Define a notifier handle.
 notifier_handle_t powerModeHandle;

 // Callback configuration.
 user_callback_data_t callbackData0;

 notifier_callback_config_t callbackCfg0 = {callback0,
 kNOTIFIER_CallbackBeforeAfter,
 (void *)&callbackData0};

 notifier_callback_config_t callbacks[] = {callbackCfg0};

 // Power mode configurations.
 power_user_config_t vlprConfig;
 power_user_config_t stopConfig;

 notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

 // Definition of a transition to and out the power modes.
 vlprConfig.mode = kAPP_PowerModeVlpr;
 vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

 stopConfig = vlprConfig;
 stopConfig.mode = kAPP_PowerModeStop;

 // Create Notifier handle.
 NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
 APP_PowerModeSwitch, NULL);
 ...
 ...
 // Power mode switch.
 NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
 kNOTIFIER_PolicyAgreement);
}
```

## Data Structures

- struct **notifier\_notification\_block\_t**  
*notification block passed to the registered callback function.* [More...](#)
- struct **notifier\_callback\_config\_t**  
*Callback configuration structure.* [More...](#)
- struct **notifier\_handle\_t**  
*Notifier handle structure.* [More...](#)

## Typedefs

- **typedef void notifier\_user\_config\_t**  
*Notifier user configuration type.*
- **typedef status\_t(\* notifier\_user\_function\_t )(notifier\_user\_config\_t \*targetConfig, void \*userData)**  
*Notifier user function prototype Use this function to execute specific operations in configuration switch.*

- `typedef status_t(* notifier_callback_t )(notifier_notification_block_t *notify, void *data)`  
*Callback prototype.*

## Enumerations

- `enum _notifier_status {`  
 `kStatus_NOTIFIER_ErrorNotificationBefore,`  
 `kStatus_NOTIFIER_ErrorNotificationAfter }`  
*Notifier error codes.*
- `enum notifier_policy_t {`  
 `kNOTIFIER_PolicyAgreement,`  
 `kNOTIFIER_PolicyForcible }`  
*Notifier policies.*
- `enum notifier_notification_type_t {`  
 `kNOTIFIER_NotifyRecover = 0x00U,`  
 `kNOTIFIER_NotifyBefore = 0x01U,`  
 `kNOTIFIER_NotifyAfter = 0x02U }`  
*Notification type.*
- `enum notifier_callback_type_t {`  
 `kNOTIFIER_CallbackBefore = 0x01U,`  
 `kNOTIFIER_CallbackAfter = 0x02U,`  
 `kNOTIFIER_CallbackBeforeAfter = 0x03U }`  
*The callback type, which indicates kinds of notification the callback handles.*

## Functions

- `status_t NOTIFIER_CreateHandle (notifier_handle_t *notifierHandle, notifier_user_config_t **configs, uint8_t configsNumber, notifier_callback_config_t *callbacks, uint8_t callbacksNumber, notifier_user_function_t userFunction, void *userData)`  
*Creates a Notifier handle.*
- `status_t NOTIFIER_SwitchConfig (notifier_handle_t *notifierHandle, uint8_t configIndex, notifier_policy_t policy)`  
*Switches the configuration according to a pre-defined structure.*
- `uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t *notifierHandle)`  
*This function returns the last failed notification callback.*

### 31.3 Data Structure Documentation

#### 31.3.1 struct notifier\_notification\_block\_t

##### Data Fields

- `notifier_user_config_t * targetConfig`  
*Pointer to target configuration.*
- `notifier_policy_t policy`  
*Configure transition policy.*
- `notifier_notification_type_t notifyType`  
*Configure notification type.*

## Data Structure Documentation

### 31.3.1.0.0.91 Field Documentation

31.3.1.0.0.91.1 `notifier_user_config_t* notifier_notification_block_t::targetConfig`

31.3.1.0.0.91.2 `notifier_policy_t notifier_notification_block_t::policy`

31.3.1.0.0.91.3 `notifier_notification_type_t notifier_notification_block_t::notifyType`

### 31.3.2 `struct notifier_callback_config_t`

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. callback - pointer to the callback function callbackType - specifies when the callback is called callbackData - pointer to the data passed to the callback.

## Data Fields

- `notifier_callback_t callback`  
*Pointer to the callback function.*
- `notifier_callback_type_t callbackType`  
*Callback type.*
- `void * callbackData`  
*Pointer to the data passed to the callback.*

### 31.3.2.0.0.92 Field Documentation

31.3.2.0.0.92.1 `notifier_callback_t notifier_callback_config_t::callback`

31.3.2.0.0.92.2 `notifier_callback_type_t notifier_callback_config_t::callbackType`

31.3.2.0.0.92.3 `void* notifier_callback_config_t::callbackData`

### 31.3.3 `struct notifier_handle_t`

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. `NOTIFIER_CreateHandle()` must be called to initialize this handle.

## Data Fields

- `notifier_user_config_t ** configsTable`  
*Pointer to configure table.*
- `uint8_t configsNumber`  
*Number of configurations.*
- `notifier_callback_config_t * callbacksTable`  
*Pointer to callback table.*

- `uint8_t callbacksNumber`  
*Maximum number of callback configurations.*
- `uint8_t errorCallbackIndex`  
*Index of callback returns error.*
- `uint8_t currentConfigIndex`  
*Index of current configuration.*
- `notifier_user_function_t userFunction`  
*User function.*
- `void *userData`  
*User data passed to user function.*

### 31.3.3.0.0.93 Field Documentation

**31.3.3.0.0.93.1 `notifier_user_config_t** notifier_handle_t::configsTable`**

**31.3.3.0.0.93.2 `uint8_t notifier_handle_t::configsNumber`**

**31.3.3.0.0.93.3 `notifier_callback_config_t* notifier_handle_t::callbacksTable`**

**31.3.3.0.0.93.4 `uint8_t notifier_handle_t::callbacksNumber`**

**31.3.3.0.0.93.5 `uint8_t notifier_handle_t::errorCallbackIndex`**

**31.3.3.0.0.93.6 `uint8_t notifier_handle_t::currentConfigIndex`**

**31.3.3.0.0.93.7 `notifier_user_function_t notifier_handle_t::userFunction`**

**31.3.3.0.0.93.8 `void* notifier_handle_t::userData`**

## 31.4 Typedef Documentation

### 31.4.1 **typedef void notifier\_user\_config\_t**

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

### 31.4.2 **typedef status\_t(\* notifier\_user\_function\_t)(notifier\_user\_config\_t \*targetConfig, void \*userData)**

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, [NOTIFIER\\_SwitchConfig\(\)](#) exits.

Parameters

---

## Enumeration Type Documentation

|                     |                                                        |
|---------------------|--------------------------------------------------------|
| <i>targetConfig</i> | target Configuration.                                  |
| <i>userData</i>     | Refers to other specific data passed to user function. |

Returns

An error code or kStatus\_Success.

### 31.4.3 **typedef status\_t(\* notifier\_callback\_t)(notifier\_notification\_block\_t \*notify, void \*data)**

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the [notifier\\_callback\\_config\\_t](#) callback configuration structure. Depending on callback type, function of this prototype is called (see [NOTIFIER\\_SwitchConfig\(\)](#)) before configuration switch, after it or in both use cases to notify about the switch progress (see [notifier\\_callback\\_type\\_t](#)). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see [notifier\\_notification\\_block\\_t](#)) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see [notifier\\_policy\\_t](#)), the callback may deny the execution of the user function by returning an error code different than kStatus\_Success (see [NOTIFIER\\_SwitchConfig\(\)](#)).

Parameters

|               |                                                                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>notify</i> | Notification block.                                                                                                                                        |
| <i>data</i>   | Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information. |

Returns

An error code or kStatus\_Success.

## 31.5 Enumeration Type Documentation

### 31.5.1 enum \_notifier\_status

Used as return value of Notifier functions.

Enumerator

***kStatus\_NOTIFIER\_ErrorNotificationBefore*** An error occurs during send "BEFORE" notification.

***kStatus\_NOTIFIER\_ErrorNotificationAfter*** An error occurs during send "AFTER" notification.

### 31.5.2 enum notifier\_policy\_t

Defines whether the user function execution is forced or not. For `kNOTIFIER_PolicyForcible`, the user function is executed regardless of the callback results, while `kNOTIFIER_PolicyAgreement` policy is used to exit `NOTIFIER_SwitchConfig()` when any of the callbacks returns error code. See also `NOTIFIER_SwitchConfig()` description.

Enumerator

***kNOTIFIER\_PolicyAgreement*** `NOTIFIER_SwitchConfig()` method is exited when any of the callbacks returns error code.

***kNOTIFIER\_PolicyForcible*** The user function is executed regardless of the results.

### 31.5.3 enum notifier\_notification\_type\_t

Used to notify registered callbacks

Enumerator

***kNOTIFIER\_NotifyRecover*** Notify IP to recover to previous work state.

***kNOTIFIER\_NotifyBefore*** Notify IP that configuration setting is going to change.

***kNOTIFIER\_NotifyAfter*** Notify IP that configuration setting has been changed.

### 31.5.4 enum notifier\_callback\_type\_t

Used in the callback configuration structure (`notifier_callback_config_t`) to specify when the registered callback is called during configuration switch initiated by the `NOTIFIER_SwitchConfig()`. Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect `NOTIFIER_SwitchConfig()` execution. See the `NOTIFIER_SwitchConfig()` and `notifier_policy_t` documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

***kNOTIFIER\_CallbackBefore*** Callback handles BEFORE notification.

***kNOTIFIER\_CallbackAfter*** Callback handles AFTER notification.

***kNOTIFIER\_CallbackBeforeAfter*** Callback handles BEFORE and AFTER notification.

## Function Documentation

### 31.6 Function Documentation

31.6.1 `status_t NOTIFIER_CreateHandle ( notifier_handle_t * notifierHandle,  
notifier_user_config_t ** configs, uint8_t configsNumber, notifier_callback-  
_config_t * callbacks, uint8_t callbacksNumber, notifier_user_function_t  
userFunction, void * userData )`

## Parameters

|                         |                                                                                                                                         |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>notifierHandle</i>   | A pointer to the notifier handle.                                                                                                       |
| <i>configs</i>          | A pointer to an array with references to all configurations which is handled by the Notifier.                                           |
| <i>configsNumber</i>    | Number of configurations. Size of the configuration array.                                                                              |
| <i>callbacks</i>        | A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value. |
| <i>callbacks-Number</i> | Number of registered callbacks. Size of the callbacks array.                                                                            |
| <i>userFunction</i>     | User function.                                                                                                                          |
| <i>userData</i>         | User data passed to user function.                                                                                                      |

## Returns

An error Code or kStatus\_Success.

### 31.6.2 **status\_t NOTIFIER\_SwitchConfig ( notifier\_handle\_t \* *notifierHandle*, uint8\_t *configIndex*, notifier\_policy\_t *policy* )**

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER\_PolicyForcible) or exited (kNOTIFIER\_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER\_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when NOTIFIER\_SwitchConfig() exits.

## Parameters

## Function Documentation

|                       |                                                                            |
|-----------------------|----------------------------------------------------------------------------|
| <i>notifierHandle</i> | pointer to notifier handle                                                 |
| <i>configIndex</i>    | Index of the target configuration.                                         |
| <i>policy</i>         | Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible. |

Returns

An error code or kStatus\_Success.

### 31.6.3 **uint8\_t NOTIFIER\_GetErrorCallbackIndex ( notifier\_handle\_t \*notifierHandle )**

This function returns an index of the last callback that failed during the configuration switch while the last [NOTIFIER\\_SwitchConfig\(\)](#) was called. If the last [NOTIFIER\\_SwitchConfig\(\)](#) call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

|                       |                                |
|-----------------------|--------------------------------|
| <i>notifierHandle</i> | Pointer to the notifier handle |
|-----------------------|--------------------------------|

Returns

Callback Index of the last failed callback or value equal to callbacks count.

# Chapter 32

## RGPIO: Rapid General-Purpose Input/Output Driver

### 32.1 Overview

#### Modules

- FGPIO Driver
- RGPIO Driver

#### Data Structures

- struct `rgpio_pin_config_t`  
*The RGPIO pin configuration structure.* [More...](#)

#### Enumerations

- enum `rgpio_pin_direction_t` {  
  `kRGPIO_DigitalInput` = 0U,  
  `kRGPIO_DigitalOutput` = 1U }  
*RGPIO direction definition.*

#### Driver version

- #define `FSL_RGPIO_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 1))  
*RGPIO driver version 2.0.1.*

### 32.2 Data Structure Documentation

#### 32.2.1 struct `rgpio_pin_config_t`

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the `outputConfig` unused. Note that in some use cases, the corresponding port property should be configured in advance with the `PORT_SetPinConfig()`.

#### Data Fields

- `rgpio_pin_direction_t pinDirection`  
*RGPIO direction, input or output.*
- `uint8_t outputLogic`  
*Set a default output logic, which has no use in input.*

## Enumeration Type Documentation

### 32.3 Macro Definition Documentation

32.3.1 `#define FSL_RGPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

### 32.4 Enumeration Type Documentation

#### 32.4.1 `enum rgpio_pin_direction_t`

Enumerator

*kRGPIO\_DigitalInput* Set current pin as digital input.

*kRGPIO\_DigitalOutput* Set current pin as digital output.

## 32.5 RGPIO Driver

### 32.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Rapid General-Purpose Input/Output (RGPIO) module of MCUXpresso SDK devices.

### 32.5.2 Typical use case

#### 32.5.2.1 Output Operation

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

#### 32.5.2.2 Input Operation

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

## GPIO Configuration

- void [RGPIO\\_PinInit](#) (RGPIO\_Type \*base, uint32\_t pin, const rgpio\_pin\_config\_t \*config)  
*Initializes a RGPIO pin used by the board.*
- uint32\_t [RGPIO\\_GetInstance](#) (RGPIO\_Type \*base)  
*Gets the RGPIO instance according to the RGPIO base.*

## GPIO Output Operations

- static void [RGPIO\\_PinWrite](#) (RGPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the multiple RGPIO pins to the logic 1 or 0.*
- static void [RGPIO\\_WritePinOutput](#) (RGPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the multiple RGPIO pins to the logic 1 or 0.*
- static void [RGPIO\\_PortSet](#) (RGPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple RGPIO pins to the logic 1.*
- static void [RGPIO\\_SetPinsOutput](#) (RGPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple RGPIO pins to the logic 1.*
- static void [RGPIO\\_PortClear](#) (RGPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple RGPIO pins to the logic 0.*
- static void [RGPIO\\_ClearPinsOutput](#) (RGPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple RGPIO pins to the logic 0.*
- static void [RGPIO\\_PortToggle](#) (RGPIO\_Type \*base, uint32\_t mask)  
*Reverses the current output logic of the multiple RGPIO pins.*
- static void [RGPIO\\_TogglePinsOutput](#) (RGPIO\_Type \*base, uint32\_t mask)  
*Reverses the current output logic of the multiple RGPIO pins.*

### RGPIO Input Operations

- static uint32\_t [RGPIO\\_PinRead](#) (RGPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the RGPIO port.*
- static uint32\_t [RGPIO\\_ReadPinInput](#) (RGPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the RGPIO port.*

### 32.5.3 Function Documentation

#### 32.5.3.1 void [RGPIO\\_PinInit](#) ( RGPIO\_Type \* *base*, uint32\_t *pin*, const rgpio\_pin\_config\_t \* *config* )

To initialize the RGPIO, define a pin configuration, as either input or output, in the user file. Then, call the [RGPIO\\_PinInit\(\)](#) function.

This is an example to define an input pin or an output pin configuration.

```
* // Define a digital input pin configuration,
* rgpio_pin_config_t config =
* {
* kRGPIO_DigitalInput,
* 0,
* }
* //Define a digital output pin configuration,
* rgpio_pin_config_t config =
* {
* kRGPIO_DigitalOutput,
* 0,
* }
```

Parameters

|               |                                                                    |
|---------------|--------------------------------------------------------------------|
| <i>base</i>   | RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.) |
| <i>pin</i>    | RGPIO port pin number                                              |
| <i>config</i> | RGPIO pin configuration pointer                                    |

#### 32.5.3.2 uint32\_t [RGPIO\\_GetInstance](#) ( RGPIO\_Type \* *base* )

Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>base</i> | RGPIO peripheral base pointer(PTA, PTB, PTC, etc.) |
|-------------|----------------------------------------------------|

Return values

|              |          |
|--------------|----------|
| <i>RGPIO</i> | instance |
|--------------|----------|

### 32.5.3.3 static void RGPIO\_PinWrite ( RGPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]

Parameters

|               |                                                                                                                                                                                         |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)                                                                                                                      |
| <i>pin</i>    | RGPIO pin number                                                                                                                                                                        |
| <i>output</i> | RGPIO 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> |

### 32.5.3.4 static void RGPIO\_WritePinOutput ( RGPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]

### 32.5.3.5 static void RGPIO\_PortSet ( RGPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.) |
| <i>mask</i> | RGPIO pin number macro                                             |

### 32.5.3.6 static void RGPIO\_SetPinsOutput ( RGPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### 32.5.3.7 static void RGPIO\_PortClear ( RGPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## GPIO Driver

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**32.5.3.8 static void GPIO\_Clear PinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**32.5.3.9 static void GPIO\_PortToggle ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro                                          |

**32.5.3.10 static void GPIO\_Toggle PinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

**32.5.3.11 static uint32\_t GPIO\_PinRead ( GPIO\_Type \* *base*, uint32\_t *pin* )  
[inline], [static]**

Parameters

|             |                                                                |
|-------------|----------------------------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.) |
| <i>pin</i>  | GPIO pin number                                                |

Return values

|              |                                                                                                                                                                       |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>RGPIO</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> |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**32.5.3.12 static uint32\_t RGPIO\_ReadPinInput ( RGPIO\_Type \* *base*, uint32\_t *pin* )  
[inline], [static]**

### 32.6 FGPIO Driver

This chapter describes the programming interface of the FGPIO driver. The FGPIO driver configures the FGPIO module and provides a functional interface to build the RGPIO application.

Note

FGPIO (Fast GPIO) is only available in a few MCUs. FGPIO and RGPIO share the same peripheral but use different registers. FGPIO is closer to the core than the regular RGPIO and it's faster to read and write.

#### 32.6.1 Typical use case

##### 32.6.1.1 Output Operation

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

##### 32.6.1.2 Input Operation

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

# Chapter 33

## Shell

### 33.1 Overview

This part describes the programming interface of the Shell middleware. Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

### 33.2 Function groups

#### 33.2.1 Initialization

To initialize the Shell middleware, call the `SHELL_Init()` function with these parameters. This function automatically enables the middleware.

```
void SHELL_Init(p_shell_context_t context, send_data_cb_t send_cb, recv_data_cb_t recv_cb, 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(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
```

#### 33.2.2 Advanced Feature

- Support to get a character from standard input devices.

```
static uint8_t GetChar(p_shell_context_t context);
```

| Commands   | Description                                      |
|------------|--------------------------------------------------|
| Help       | Lists all commands which are supported by Shell. |
| Exit       | Exits the Shell program.                         |
| strCompare | Compares the two input strings.                  |

| Input character | Description                                         |
|-----------------|-----------------------------------------------------|
| A               | Gets the latest command in the history.             |
| B               | Gets the first command in the history.              |
| C               | Replaces one character at the right of the pointer. |

## Function groups

| Input character | Description                                        |
|-----------------|----------------------------------------------------|
| D               | Replaces one character at the left of the pointer. |
|                 | Run AutoComplete function                          |
|                 | Run cmdProcess function                            |
|                 | Clears a command.                                  |

### 33.2.3 Shell Operation

```
SHELL_Init(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
SHELL_Main(&user_context);
```

## 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](#) (520U)  
*The handle size of the shell module.*
- #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)` }

## 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 *command)`
  
*Registers the shell command.*
- `shell_status_t SHELL_UnregisterCommand (shell_command_t *command)`
  
*Unregisters the shell command.*
- `shell_status_t SHELL_Write (shell_handle_t shellHandle, 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.*
- `void SHELL_Task (shell_handle_t shellHandle)`
  
*The task function for Shell.*

## 33.3 Data Structure Documentation

### 33.3.1 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*

#### 33.3.1.0.0.94 Field Documentation

##### 33.3.1.0.0.94.1 const char\* shell\_command\_t::pcCommand

For example "help". It must be all lower case.

## Macro Definition Documentation

### 33.3.1.0.0.94.2 `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".

### 33.3.1.0.0.94.3 `const cmd_function_t shell_command_t::pFuncCallBack`

### 33.3.1.0.0.94.4 `uint8_t shell_command_t::cExpectedNumberOfParameters`

## 33.4 Macro Definition Documentation

### 33.4.1 `#define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE`

### 33.4.2 `#define SHELL_AUTO_COMPLETE (1U)`

### 33.4.3 `#define SHELL_BUFFER_SIZE (64U)`

### 33.4.4 `#define SHELL_MAX_ARGS (8U)`

### 33.4.5 `#define SHELL_HISTORY_COUNT (3U)`

### 33.4.6 `#define SHELL_HANDLE_SIZE (520U)`

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

### 33.4.7 `#define SHELL_COMMAND_DEFINE( command, descriptor, callback, paramInt )`

#### Value:

```
\n shell_command_t g_shellCommand##command = {\n (#command), (descriptor), (callback), (paramCount), {0},\n } \n
```

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);\n* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));\n*
```

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.                                                                              |

### 33.4.8 #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". |
|----------------|------------------------------------------------------------------------------------------------------------------------------|

## 33.5 Typedef Documentation

### 33.5.1 **typedef shell\_status\_t(\* cmd\_function\_t)(shell\_handle\_t shellHandle, int32\_t argc, char \*\*argv)**

## 33.6 Enumeration Type Documentation

### 33.6.1 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.

## 33.7 Function Documentation

### 33.7.1 **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.

## Function Documentation

```
* static uint8_t s_shellHandleBuffer[SHELL_HANDLE_SIZE];
* static shell_handle_t s_shellHandle = &s_shellHandleBuffer[0];
* SHELL_Init(s_shellHandle, 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. |
| <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.                     |

### 33.7.2 **shell\_status\_t SHELL\_RegisterCommand ( shell\_handle\_t *shellHandle*, shell\_command\_t \* *command* )**

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>command</i>     | The command element.             |

### Return values

|                              |                                    |
|------------------------------|------------------------------------|
| <i>kStatus_SHELL_Success</i> | Successfully register the command. |
| <i>kStatus_SHELL_Error</i>   | An error occurred.                 |

### 33.7.3 shell\_status\_t SHELL\_UnregisterCommand ( shell\_command\_t \* *command* )

This function is used to unregister the shell command.

Parameters

|                |                      |
|----------------|----------------------|
| <i>command</i> | The command element. |
|----------------|----------------------|

Return values

|                              |                                      |
|------------------------------|--------------------------------------|
| <i>kStatus_SHELL_Success</i> | Successfully unregister the command. |
|------------------------------|--------------------------------------|

### 33.7.4 shell\_status\_t SHELL\_Write ( shell\_handle\_t *shellHandle*, 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.      |

### 33.7.5 int SHELL\_Printf ( shell\_handle\_t *shellHandle*, const char \* *formatString*, ... )

Call this function to write a formatted output to the shell output stream.

## Function Documentation

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.

### 33.7.6 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. |
|--------------------|----------------------------------|

# Chapter 34

## SEMA42: Hardware Semaphores Driver

### 34.1 Overview

The MCUXpresso SDK provides a driver for the SEMA42 module of MCUXpresso SDK devices.

The SEMA42 driver is used for multicore platforms. Before using the SEMA42, call the [SEMA42\\_Init\(\)](#) function to initialize the module. Note that this function only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either the [SEMA42\\_ResetGate\(\)](#) or [SEMA42\\_ResetAllGates\(\)](#) functions. The function [SEMA42\\_Deinit\(\)](#) deinitializes the SEMA42.

The SEMA42 provides two functions to lock the SEMA42 gate. The function [SEMA42\\_TryLock\(\)](#) tries to lock the gate. If the gate has been locked by another processor, this function returns an error immediately. The function [SEMA42\\_Lock\(\)](#) is a blocking method, which waits until the gate is free and locks it.

The [SEMA42\\_Unlock\(\)](#) unlocks the SEMA42 gate. The gate can only be unlocked by the processor which locked it. If the gate is not locked by the current processor, this function takes no effect. The function [SEMA42\\_GetGateStatus\(\)](#) returns a status whether the gate is unlocked and which processor locks the gate.

The SEMA42 gate can be reset to unlock forcefully. The function [SEMA42\\_ResetGate\(\)](#) resets a specific gate. The function [SEMA42\\_ResetAllGates\(\)](#) resets all gates.

### 34.2 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/sema42

### Macros

- #define [SEMA42\\_GATE\\_NUM\\_RESET\\_ALL](#) (64U)  
*The number to reset all SEMA42 gates.*
- #define [SEMA42\\_GATEn\(base, n\)](#) (\*(&((base)->GATE3) + ((n) ^ 3U)))  
*SEMA42 gate n register address.*

### Enumerations

- enum [\\_sema42\\_status](#) {  
    kStatus\_SEMA42\_Busy = MAKE\_STATUS(kStatusGroup\_SEMA42, 0),  
    kStatus\_SEMA42\_Reseting = MAKE\_STATUS(kStatusGroup\_SEMA42, 1) }  
*SEMA42 status return codes.*
- enum [sema42\\_gate\\_status\\_t](#) {

## Typical use case

```
kSEMA42_Unlocked = 0U,
kSEMA42_LockedByProc0 = 1U,
kSEMA42_LockedByProc1 = 2U,
kSEMA42_LockedByProc2 = 3U,
kSEMA42_LockedByProc3 = 4U,
kSEMA42_LockedByProc4 = 5U,
kSEMA42_LockedByProc5 = 6U,
kSEMA42_LockedByProc6 = 7U,
kSEMA42_LockedByProc7 = 8U,
kSEMA42_LockedByProc8 = 9U,
kSEMA42_LockedByProc9 = 10U,
kSEMA42_LockedByProc10 = 11U,
kSEMA42_LockedByProc11 = 12U,
kSEMA42_LockedByProc12 = 13U,
kSEMA42_LockedByProc13 = 14U,
kSEMA42_LockedByProc14 = 15U }
```

*SEMA42 gate lock status.*

## Functions

- void [SEMA42\\_Init](#) (SEMA42\_Type \*base)  
*Initializes the SEMA42 module.*
- void [SEMA42\\_Deinit](#) (SEMA42\_Type \*base)  
*De-initializes the SEMA42 module.*
- status\_t [SEMA42\\_TryLock](#) (SEMA42\_Type \*base, uint8\_t gateNum, uint8\_t procNum)  
*Tries to lock the SEMA42 gate.*
- void [SEMA42\\_Lock](#) (SEMA42\_Type \*base, uint8\_t gateNum, uint8\_t procNum)  
*Locks the SEMA42 gate.*
- static void [SEMA42\\_Unlock](#) (SEMA42\_Type \*base, uint8\_t gateNum)  
*Unlocks the SEMA42 gate.*
- static [sema42\\_gate\\_status\\_t](#) [SEMA42\\_GetGateStatus](#) (SEMA42\_Type \*base, uint8\_t gateNum)  
*Gets the status of the SEMA42 gate.*
- status\_t [SEMA42\\_ResetGate](#) (SEMA42\_Type \*base, uint8\_t gateNum)  
*Resets the SEMA42 gate to an unlocked status.*
- static status\_t [SEMA42\\_ResetAllGates](#) (SEMA42\_Type \*base)  
*Resets all SEMA42 gates to an unlocked status.*

## Driver version

- #define [FSL\\_SEMA42\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 0))  
*SEMA42 driver version.*

### 34.3 Macro Definition Documentation

#### 34.3.1 #define SEMA42\_GATE\_NUM\_RESET\_ALL (64U)

#### 34.3.2 #define SEMA42\_GATEn( *base*, *n* ) (\*(&((*base*)->GATE3) + ((*n*) ^ 3U)))

The SEMA42 gates are sorted in the order 3, 2, 1, 0, 7, 6, 5, 4, ... not in the order 0, 1, 2, 3, 4, 5, 6, 7, ... The macro SEMA42\_GATEn gets the SEMA42 gate based on the gate index.

The input gate index is XOR'ed with 3U:  $0 \wedge 3 = 3$   $1 \wedge 3 = 2$   $2 \wedge 3 = 1$   $3 \wedge 3 = 0$   $4 \wedge 3 = 7$   $5 \wedge 3 = 6$   $6 \wedge 3 = 5$   $7 \wedge 3 = 4$  ...

### 34.4 Enumeration Type Documentation

#### 34.4.1 enum \_sema42\_status

Enumerator

*kStatus\_SEMA42\_Busy* SEMA42 gate has been locked by other processor.

*kStatus\_SEMA42\_Reseting* SEMA42 gate reseting is ongoing.

#### 34.4.2 enum sema42\_gate\_status\_t

Enumerator

*kSEMA42\_Unlocked* The gate is unlocked.

*kSEMA42\_LockedByProc0* The gate is locked by processor 0.

*kSEMA42\_LockedByProc1* The gate is locked by processor 1.

*kSEMA42\_LockedByProc2* The gate is locked by processor 2.

*kSEMA42\_LockedByProc3* The gate is locked by processor 3.

*kSEMA42\_LockedByProc4* The gate is locked by processor 4.

*kSEMA42\_LockedByProc5* The gate is locked by processor 5.

*kSEMA42\_LockedByProc6* The gate is locked by processor 6.

*kSEMA42\_LockedByProc7* The gate is locked by processor 7.

*kSEMA42\_LockedByProc8* The gate is locked by processor 8.

*kSEMA42\_LockedByProc9* The gate is locked by processor 9.

*kSEMA42\_LockedByProc10* The gate is locked by processor 10.

*kSEMA42\_LockedByProc11* The gate is locked by processor 11.

*kSEMA42\_LockedByProc12* The gate is locked by processor 12.

*kSEMA42\_LockedByProc13* The gate is locked by processor 13.

*kSEMA42\_LockedByProc14* The gate is locked by processor 14.

## Function Documentation

### 34.5 Function Documentation

#### 34.5.1 void SEMA42\_Init ( SEMA42\_Type \* *base* )

This function initializes the SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA42\_ResetGate or SEMA42\_ResetAllGates function.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | SEMA42 peripheral base address. |
|-------------|---------------------------------|

#### 34.5.2 void SEMA42\_Deinit ( SEMA42\_Type \* *base* )

This function de-initializes the SEMA42 module. It only disables the clock.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | SEMA42 peripheral base address. |
|-------------|---------------------------------|

#### 34.5.3 status\_t SEMA42\_TryLock ( SEMA42\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *procNum* )

This function tries to lock the specific SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number to lock.            |
| <i>procNum</i> | Current processor number.       |

Return values

|                            |                                                   |
|----------------------------|---------------------------------------------------|
| <i>kStatus_Success</i>     | Lock the sema42 gate successfully.                |
| <i>kStatus_SEMA42_Busy</i> | Sema42 gate has been locked by another processor. |

#### 34.5.4 void SEMA42\_Lock ( SEMA42\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *procNum* )

This function locks the specific SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

## Function Documentation

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number to lock.            |
| <i>procNum</i> | Current processor number.       |

### 34.5.5 static void SEMA42\_Unlock ( SEMA42\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function unlocks the specific SEMA42 gate. It only writes unlock value to the SEMA42 gate register. However, it does not check whether the SEMA42 gate is locked by the current processor or not. As a result, if the SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number to unlock.          |

### 34.5.6 static sema42\_gate\_status\_t SEMA42\_GetGateStatus ( SEMA42\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function checks the lock status of a specific SEMA42 gate.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number.                    |

Returns

status Current status.

### 34.5.7 status\_t SEMA42\_ResetGate ( SEMA42\_Type \* *base*, uint8\_t *gateNum* )

This function resets a SEMA42 gate to an unlocked status.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number.                    |

Return values

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_Success</i>         | SEMA42 gate is reset successfully.   |
| <i>kStatus_SEMA42_Reseting</i> | Some other reset process is ongoing. |

### 34.5.8 static status\_t SEMA42\_ResetAllGates ( SEMA42\_Type \* *base* ) [inline], [static]

This function resets all SEMA42 gate to an unlocked status.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | SEMA42 peripheral base address. |
|-------------|---------------------------------|

Return values

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_Success</i>         | SEMA42 is reset successfully.        |
| <i>kStatus_SEMA42_Reseting</i> | Some other reset process is ongoing. |

## Function Documentation

# Chapter 35

## TPM: Timer PWM Module

### 35.1 Overview

The MCUXpresso SDK provides a driver for the Timer PWM Module (TPM) of MCUXpresso SDK devices.

The TPM driver supports the generation of PWM signals, input capture, and output compare modes. On some SoCs, the driver supports the generation of combined PWM signals, dual-edge capture, and quadrature decoder modes. The driver also supports configuring each of the TPM fault inputs. The fault input is available only on some SoCs.

The function [TPM\\_Init\(\)](#) initializes the TPM with a specified configurations. The function [TPM\\_GetDefaultConfig\(\)](#) gets the default configurations. On some SoCs, the initialization function issues a software reset to reset the TPM internal logic. The initialization function configures the TPM's behavior when it receives a trigger input and its operation in doze and debug modes.

The function [TPM\\_Deinit\(\)](#) disables the TPM counter and turns off the module clock.

The function [TPM\\_SetupPwm\(\)](#) sets up TPM channels for the PWM output. The function can set up the PWM signal properties for multiple channels. Each channel has its own [tpm\\_chnl\\_pwm\\_signal\\_param\\_t](#) structure that is used to specify the output signals duty cycle and level-mode. However, the same PWM period and PWM mode is applied to all channels requesting a PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 where 0=inactive signal (0% duty cycle) and 100=always active signal (100% duty cycle). When generating a combined PWM signal, the channel number passed refers to a channel pair number, for example 0 refers to channel 0 and 1, 1 refers to channels 2 and 3.

The function [TPM\\_UpdatePwmDutycycle\(\)](#) updates the PWM signal duty cycle of a particular TPM channel.

The function [TPM\\_UpdateChnlEdgeLevelSelect\(\)](#) updates the level select bits of a particular TPM channel. This can be used to disable the PWM output when making changes to the PWM signal.

The function [TPM\\_SetupInputCapture\(\)](#) sets up a TPM channel for input capture. The user can specify the capture edge.

The function [TPM\\_SetupDualEdgeCapture\(\)](#) can be used to measure the pulse width of a signal. This is available only for certain SoCs. A channel pair is used during the capture with the input signal coming through a channel that can be configured. The user can specify the capture edge for each channel and any filter value to be used when processing the input signal.

The function [TPM\\_SetupOutputCompare\(\)](#) sets up a TPM channel for output comparison. The user can specify the channel output on a successful comparison and a comparison value.

The function [TPM\\_SetupQuadDecode\(\)](#) sets up TPM channels 0 and 1 for quad decode, which is available only for certain SoCs. The user can specify the quad decode mode, polarity, and filter properties for each

## Typical use case

input signal.

The function TPM\_SetupFault() sets up the properties for each fault, which is available only for certain SoCs. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

Provides functions to get and clear the TPM status.

Provides functions to enable/disable TPM interrupts and get current enabled interrupts.

## 35.2 Typical use case

### 35.2.1 PWM output

Output the PWM signal on 2 TPM channels with different duty cycles. Periodically update the PWM signal duty cycle. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/tpm

## Data Structures

- struct [tpm\\_chnl\\_pwm\\_signal\\_param\\_t](#)  
*Options to configure a TPM channel's PWM signal. [More...](#)*
- struct [tpm\\_dual\\_edge\\_capture\\_param\\_t](#)  
*TPM dual edge capture parameters. [More...](#)*
- struct [tpm\\_phase\\_params\\_t](#)  
*TPM quadrature decode phase parameters. [More...](#)*
- struct [tpm\\_config\\_t](#)  
*TPM config structure. [More...](#)*

## Enumerations

- enum [tpm\\_chnl\\_t](#) {  
  kTPM\_Chnl\_0 = 0U,  
  kTPM\_Chnl\_1,  
  kTPM\_Chnl\_2,  
  kTPM\_Chnl\_3,  
  kTPM\_Chnl\_4,  
  kTPM\_Chnl\_5,  
  kTPM\_Chnl\_6,  
  kTPM\_Chnl\_7 }  
*List of TPM channels.*
- enum [tpm\\_pwm\\_mode\\_t](#) {  
  kTPM\_EdgeAlignedPwm = 0U,  
  kTPM\_CenterAlignedPwm,  
  kTPM\_CombinedPwm }  
*TPM PWM operation modes.*
- enum [tpm\\_pwm\\_level\\_select\\_t](#) {  
  kTPM\_NoPwmSignal = 0U,  
  kTPM\_LowTrue,

- kTPM\_HighTrue }
- TPM PWM output pulse mode: high-true, low-true or no output.*
- enum `tpm_trigger_select_t`
  - Trigger options available.*
- enum `tpm_trigger_source_t` {
  - `kTPM_TriggerSource_External` = 0U,
  - `kTPM_TriggerSource_Internal` }
  - Trigger source options available.*
- enum `tpm_output_compare_mode_t` {
  - `kTPM_NoOutputSignal` = (1U << TPM\_CnSC\_MSA\_SHIFT),
  - `kTPM_ToggleOnMatch` = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (1U << TPM\_CnSC\_ELSA\_S-HIFT)),
  - `kTPM_ClearOnMatch` = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (2U << TPM\_CnSC\_ELSA\_S-HIFT)),
  - `kTPM_SetOnMatch` = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (3U << TPM\_CnSC\_ELSA\_SHIFT)),
  - `kTPM_HighPulseOutput` = ((3U << TPM\_CnSC\_MSA\_SHIFT) | (1U << TPM\_CnSC\_ELSA\_S-HIFT)),
  - `kTPM_LowPulseOutput` = ((3U << TPM\_CnSC\_MSA\_SHIFT) | (2U << TPM\_CnSC\_ELSA\_S-HIFT)) }

*TPM output compare modes.*
- enum `tpm_input_capture_edge_t` {
  - `kTPM_RisingEdge` = (1U << TPM\_CnSC\_ELSA\_SHIFT),
  - `kTPM_FallingEdge` = (2U << TPM\_CnSC\_ELSA\_SHIFT),
  - `kTPM_RiseAndFallEdge` = (3U << TPM\_CnSC\_ELSA\_SHIFT) }

*TPM input capture edge.*
- enum `tpm_quad_decode_mode_t` {
  - `kTPM_QuadPhaseEncode` = 0U,
  - `kTPM_QuadCountAndDir` }

*TPM quadrature decode modes.*
- enum `tpm_phase_polarity_t` {
  - `kTPM_QuadPhaseNormal` = 0U,
  - `kTPM_QuadPhaseInvert` }

*TPM quadrature phase polarities.*
- enum `tpm_clock_source_t` {
  - `kTPM_SystemClock` = 1U,
  - `kTPM_ExternalClock` }

*TPM clock source selection.*
- enum `tpm_clock_prescale_t` {
  - `kTPM_Prescale_Divide_1` = 0U,
  - `kTPM_Prescale_Divide_2`,
  - `kTPM_Prescale_Divide_4`,
  - `kTPM_Prescale_Divide_8`,
  - `kTPM_Prescale_Divide_16`,
  - `kTPM_Prescale_Divide_32`,
  - `kTPM_Prescale_Divide_64`,
  - `kTPM_Prescale_Divide_128` }

## Typical use case

- *TPM prescale value selection for the clock source.*  
• enum `tpm_interrupt_enable_t` {  
  `kTPM_Chnl0InterruptEnable` = (1U << 0),  
  `kTPM_Chnl1InterruptEnable` = (1U << 1),  
  `kTPM_Chnl2InterruptEnable` = (1U << 2),  
  `kTPM_Chnl3InterruptEnable` = (1U << 3),  
  `kTPM_Chnl4InterruptEnable` = (1U << 4),  
  `kTPM_Chnl5InterruptEnable` = (1U << 5),  
  `kTPM_Chnl6InterruptEnable` = (1U << 6),  
  `kTPM_Chnl7InterruptEnable` = (1U << 7),  
  `kTPM_TimeOverflowInterruptEnable` = (1U << 8) }  
*List of TPM interrupts.*  
• enum `tpm_status_flags_t` {  
  `kTPM_Chnl0Flag` = (1U << 0),  
  `kTPM_Chnl1Flag` = (1U << 1),  
  `kTPM_Chnl2Flag` = (1U << 2),  
  `kTPM_Chnl3Flag` = (1U << 3),  
  `kTPM_Chnl4Flag` = (1U << 4),  
  `kTPM_Chnl5Flag` = (1U << 5),  
  `kTPM_Chnl6Flag` = (1U << 6),  
  `kTPM_Chnl7Flag` = (1U << 7),  
  `kTPM_TimeOverflowFlag` = (1U << 8) }  
*List of TPM flags.*

## Functions

- static void `TPM_Reset` (TPM\_Type \*base)  
*Performs a software reset on the TPM module.*

## Driver version

- #define `FSL TPM_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 3))  
*Version 2.0.3.*

## Initialization and deinitialization

- void `TPM_Init` (TPM\_Type \*base, const `tpm_config_t` \*config)  
*Ungates the TPM clock and configures the peripheral for basic operation.*
- void `TPM_Deinit` (TPM\_Type \*base)  
*Stops the counter and gates the TPM clock.*
- void `TPM_GetDefaultConfig` (`tpm_config_t` \*config)  
*Fill in the TPM config struct with the default settings.*

## Channel mode operations

- status\_t `TPM_SetupPwm` (TPM\_Type \*base, const `tpm_chnl_pwm_signal_param_t` \*chnlParams, uint8\_t numOfChnls, `tpm_pwm_mode_t` mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)  
*Configures the PWM signal parameters.*

- void [TPM\\_UpdatePwmDutycycle](#) (TPM\_Type \*base, [tpm\\_chnl\\_t](#) chnlNumber, [tpm\\_pwm\\_mode\\_t](#) currentPwmMode, uint8\_t dutyCyclePercent)  
*Update the duty cycle of an active PWM signal.*
- void [TPM\\_UpdateChnlEdgeLevelSelect](#) (TPM\_Type \*base, [tpm\\_chnl\\_t](#) chnlNumber, uint8\_t level)  
*Update the edge level selection for a channel.*
- void [TPM\\_SetupInputCapture](#) (TPM\_Type \*base, [tpm\\_chnl\\_t](#) chnlNumber, [tpm\\_input\\_capture\\_edge\\_t](#) captureMode)  
*Enables capturing an input signal on the channel using the function parameters.*
- void [TPM\\_SetupOutputCompare](#) (TPM\_Type \*base, [tpm\\_chnl\\_t](#) chnlNumber, [tpm\\_output\\_compare\\_mode\\_t](#) compareMode, uint32\_t compareValue)  
*Configures the TPM to generate timed pulses.*
- void [TPM\\_SetupDualEdgeCapture](#) (TPM\_Type \*base, [tpm\\_chnl\\_t](#) chnlPairNumber, const [tpm\\_dual\\_edge\\_capture\\_param\\_t](#) \*edgeParam, uint32\_t filterValue)  
*Configures the dual edge capture mode of the TPM.*
- void [TPM\\_SetupQuadDecode](#) (TPM\_Type \*base, const [tpm\\_phase\\_params\\_t](#) \*phaseAParams, const [tpm\\_phase\\_params\\_t](#) \*phaseBParams, [tpm\\_quad\\_decode\\_mode\\_t](#) quadMode)  
*Configures the parameters and activates the quadrature decode mode.*

## Interrupt Interface

- void [TPM\\_EnableInterrupts](#) (TPM\_Type \*base, uint32\_t mask)  
*Enables the selected TPM interrupts.*
- void [TPM\\_DisableInterrupts](#) (TPM\_Type \*base, uint32\_t mask)  
*Disables the selected TPM interrupts.*
- uint32\_t [TPM\\_GetEnabledInterrupts](#) (TPM\_Type \*base)  
*Gets the enabled TPM interrupts.*

## Status Interface

- static uint32\_t [TPM\\_GetStatusFlags](#) (TPM\_Type \*base)  
*Gets the TPM status flags.*
- static void [TPM\\_ClearStatusFlags](#) (TPM\_Type \*base, uint32\_t mask)  
*Clears the TPM status flags.*

## Read and write the timer period

- static void [TPM\\_SetTimerPeriod](#) (TPM\_Type \*base, uint32\_t ticks)  
*Sets the timer period in units of ticks.*
- static uint32\_t [TPM\\_GetCurrentTimerCount](#) (TPM\_Type \*base)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [TPM\\_StartTimer](#) (TPM\_Type \*base, [tpm\\_clock\\_source\\_t](#) clockSource)  
*Starts the TPM counter.*
- static void [TPM\\_StopTimer](#) (TPM\_Type \*base)  
*Stops the TPM counter.*

## Data Structure Documentation

### 35.3 Data Structure Documentation

#### 35.3.1 struct tpm\_chnl\_pwm\_signal\_param\_t

##### Data Fields

- `tpm_chnl_t chnlNumber`  
*TPM channel to configure.*
- `tpm_pwm_level_select_t level`  
*PWM output active level select.*
- `uint8_t dutyCyclePercent`  
*PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...*
- `uint8_t firstEdgeDelayPercent`  
*Used only in combined PWM mode to generate asymmetrical PWM.*

##### 35.3.1.0.0.95 Field Documentation

###### 35.3.1.0.0.95.1 `tpm_chnl_t tpm_chnl_pwm_signal_param_t::chnlNumber`

In combined mode (available in some SoC's, this represents the channel pair number

###### 35.3.1.0.0.95.2 `uint8_t tpm_chnl_pwm_signal_param_t::dutyCyclePercent`

100=always active signal (100% duty cycle)

###### 35.3.1.0.0.95.3 `uint8_t tpm_chnl_pwm_signal_param_t::firstEdgeDelayPercent`

Specifies the delay to the first edge in a PWM period. If unsure, leave as 0; Should be specified as percentage of the PWM period

### 35.3.2 struct tpm\_dual\_edge\_capture\_param\_t

##### Note

This mode is available only on some SoC's.

##### Data Fields

- `bool enableSwap`  
*true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored*
- `tpm_input_capture_edge_t currChanEdgeMode`  
*Input capture edge select for channel n.*
- `tpm_input_capture_edge_t nextChanEdgeMode`  
*Input capture edge select for channel n+1.*

### 35.3.3 struct tpm\_phase\_params\_t

#### Data Fields

- `uint32_t phaseFilterVal`  
*Filter value, filter is disabled when the value is zero.*
- `tpm_phase_polarity_t phasePolarity`  
*Phase polarity.*

### 35.3.4 struct tpm\_config\_t

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the `TPM_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

- `tpm_clock_prescale_t prescale`  
*Select TPM clock prescale value.*
- `bool useGlobalTimeBase`  
*true: Use of an external global time base is enabled; false: disabled*
- `tpm_trigger_select_t triggerSelect`  
*Input trigger to use for controlling the counter operation.*
- `tpm_trigger_source_t triggerSource`  
*Decides if we use external or internal trigger.*
- `bool enableDoze`  
*true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode*
- `bool enableDebugMode`  
*true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode*
- `bool enableReloadOnTrigger`  
*true: TPM counter is reloaded on trigger; false: TPM counter not reloaded*
- `bool enableStopOnOverflow`  
*true: TPM counter stops after overflow; false: TPM counter continues running after overflow*
- `bool enableStartOnTrigger`  
*true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately*
- `bool enablePauseOnTrigger`  
*true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running*

#### 35.3.4.0.0.96 Field Documentation

##### 35.3.4.0.0.96.1 `tpm_trigger_source_t tpm_config_t::triggerSource`

### 35.4 Enumeration Type Documentation

#### 35.4.1 enum tpm\_chnl\_t

## Enumeration Type Documentation

Note

Actual number of available channels is SoC dependent

Enumerator

- kTPM\_Chnl\_0* TPM channel number 0.
- kTPM\_Chnl\_1* TPM channel number 1.
- kTPM\_Chnl\_2* TPM channel number 2.
- kTPM\_Chnl\_3* TPM channel number 3.
- kTPM\_Chnl\_4* TPM channel number 4.
- kTPM\_Chnl\_5* TPM channel number 5.
- kTPM\_Chnl\_6* TPM channel number 6.
- kTPM\_Chnl\_7* TPM channel number 7.

### 35.4.2 enum tpm\_pwm\_mode\_t

Enumerator

- kTPM\_EdgeAlignedPwm* Edge aligned PWM.
- kTPM\_CenterAlignedPwm* Center aligned PWM.
- kTPM\_CombinedPwm* Combined PWM.

### 35.4.3 enum tpm\_pwm\_level\_select\_t

Enumerator

- kTPM\_NoPwmSignal* No PWM output on pin.
- kTPM\_LowTrue* Low true pulses.
- kTPM\_HighTrue* High true pulses.

### 35.4.4 enum tpm\_trigger\_select\_t

This is used for both internal & external trigger sources (external option available in certain SoC's)

Note

The actual trigger options available is SoC-specific.

### 35.4.5 enum tpm\_trigger\_source\_t

Note

This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal trigger.

Enumerator

*kTPM\_TriggerSource\_External* Use external trigger input.

*kTPM\_TriggerSource\_Internal* Use internal trigger.

### 35.4.6 enum tpm\_output\_compare\_mode\_t

Enumerator

*kTPM\_NoOutputSignal* No channel output when counter reaches CnV.

*kTPM\_ToggleOnMatch* Toggle output.

*kTPM\_ClearOnMatch* Clear output.

*kTPM\_SetOnMatch* Set output.

*kTPM\_HighPulseOutput* Pulse output high.

*kTPM\_LowPulseOutput* Pulse output low.

### 35.4.7 enum tpm\_input\_capture\_edge\_t

Enumerator

*kTPM\_RisingEdge* Capture on rising edge only.

*kTPM\_FallingEdge* Capture on falling edge only.

*kTPM\_RiseAndFallEdge* Capture on rising or falling edge.

### 35.4.8 enum tpm\_quad\_decode\_mode\_t

Note

This mode is available only on some SoC's.

Enumerator

*kTPM\_QuadPhaseEncode* Phase A and Phase B encoding mode.

*kTPM\_QuadCountAndDir* Count and direction encoding mode.

## Enumeration Type Documentation

### 35.4.9 enum tpm\_phase\_polarity\_t

Enumerator

*kTPM\_QuadPhaseNormal* Phase input signal is not inverted.

*kTPM\_QuadPhaseInvert* Phase input signal is inverted.

### 35.4.10 enum tpm\_clock\_source\_t

Enumerator

*kTPM\_SystemClock* System clock.

*kTPM\_ExternalClock* External clock.

### 35.4.11 enum tpm\_clock\_prescale\_t

Enumerator

*kTPM\_Prescale\_Divide\_1* Divide by 1.

*kTPM\_Prescale\_Divide\_2* Divide by 2.

*kTPM\_Prescale\_Divide\_4* Divide by 4.

*kTPM\_Prescale\_Divide\_8* Divide by 8.

*kTPM\_Prescale\_Divide\_16* Divide by 16.

*kTPM\_Prescale\_Divide\_32* Divide by 32.

*kTPM\_Prescale\_Divide\_64* Divide by 64.

*kTPM\_Prescale\_Divide\_128* Divide by 128.

### 35.4.12 enum tpm\_interrupt\_enable\_t

Enumerator

*kTPM\_Chnl0InterruptEnable* Channel 0 interrupt.

*kTPM\_Chnl1InterruptEnable* Channel 1 interrupt.

*kTPM\_Chnl2InterruptEnable* Channel 2 interrupt.

*kTPM\_Chnl3InterruptEnable* Channel 3 interrupt.

*kTPM\_Chnl4InterruptEnable* Channel 4 interrupt.

*kTPM\_Chnl5InterruptEnable* Channel 5 interrupt.

*kTPM\_Chnl6InterruptEnable* Channel 6 interrupt.

*kTPM\_Chnl7InterruptEnable* Channel 7 interrupt.

*kTPM\_TimeOverflowInterruptEnable* Time overflow interrupt.

### 35.4.13 enum tpm\_status\_flags\_t

Enumerator

|                              |                     |
|------------------------------|---------------------|
| <i>kTPM_Chnl0Flag</i>        | Channel 0 flag.     |
| <i>kTPM_Chnl1Flag</i>        | Channel 1 flag.     |
| <i>kTPM_Chnl2Flag</i>        | Channel 2 flag.     |
| <i>kTPM_Chnl3Flag</i>        | Channel 3 flag.     |
| <i>kTPM_Chnl4Flag</i>        | Channel 4 flag.     |
| <i>kTPM_Chnl5Flag</i>        | Channel 5 flag.     |
| <i>kTPM_Chnl6Flag</i>        | Channel 6 flag.     |
| <i>kTPM_Chnl7Flag</i>        | Channel 7 flag.     |
| <i>kTPM_TimeOverflowFlag</i> | Time overflow flag. |

## 35.5 Function Documentation

### 35.5.1 void TPM\_Init ( TPM\_Type \* *base*, const tpm\_config\_t \* *config* )

Note

This API should be called at the beginning of the application using the TPM driver.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | TPM peripheral base address             |
| <i>config</i> | Pointer to user's TPM config structure. |

### 35.5.2 void TPM\_Deinit ( TPM\_Type \* *base* )

Parameters

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

### 35.5.3 void TPM\_GetDefaultConfig ( tpm\_config\_t \* *config* )

The default values are:

```
* config->prescale = kTPM_Prescale_Divide_1;
* config->useGlobalTimeBase = false;
* config->dozeEnable = false;
* config->dbgMode = false;
* config->enableReloadOnTrigger = false;
* config->enableStopOnOverflow = false;
* config->enableStartOnTrigger = false;
```

## Function Documentation

```
*#if FSL_FEATURE TPM HAS_PAUSE_COUNTER_ON_TRIGGER
* config->enablePauseOnTrigger = false;
#endif
* config->triggerSelect = kTPM_Trigger_Select_0;
*#if FSL_FEATURE TPM HAS_EXTERNAL_TRIGGER_SELECTION
* config->triggerSource = kTPM_TriggerSource_External;
#endif
*
```

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>config</i> | Pointer to user's TPM config structure. |
|---------------|-----------------------------------------|

### 35.5.4 **status\_t TPM\_SetupPwm ( TPM\_Type \* *base*, const tpm\_chnl\_pwm\_signal\_param\_t \* *chnlParams*, uint8\_t *numOfChnls*, tpm\_pwm\_mode\_t *mode*, uint32\_t *pwmFreq\_Hz*, uint32\_t *srcClock\_Hz* )**

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

### Parameters

|                    |                                                                                     |
|--------------------|-------------------------------------------------------------------------------------|
| <i>base</i>        | TPM peripheral base address                                                         |
| <i>chnlParams</i>  | Array of PWM channel parameters to configure the channel(s)                         |
| <i>numOfChnls</i>  | Number of channels to configure, this should be the size of the array passed in     |
| <i>mode</i>        | PWM operation mode, options available in enumeration <a href="#">tpm_pwm_mode_t</a> |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz                                                          |
| <i>srcClock_Hz</i> | TPM counter clock in Hz                                                             |

### Returns

kStatus\_Success if the PWM setup was successful, kStatus\_Error on failure

### 35.5.5 **void TPM\_UpdatePwmDutycycle ( TPM\_Type \* *base*, tpm\_chnl\_t *chnlNumber*, tpm\_pwm\_mode\_t *currentPwmMode*, uint8\_t *dutyCyclePercent* )**

Parameters

|                          |                                                                                                                               |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | TPM peripheral base address                                                                                                   |
| <i>chnlNumber</i>        | The channel number. In combined mode, this represents the channel pair number                                                 |
| <i>currentPwm-Mode</i>   | The current PWM mode set during PWM setup                                                                                     |
| <i>dutyCycle-Percent</i> | New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |

### 35.5.6 void TPM\_UpdateChnlEdgeLevelSelect ( **TPM\_Type** \* *base*, **tpm\_chnl\_t** *chnlNumber*, **uint8\_t** *level* )

Parameters

|                   |                                                                                                                                                       |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | TPM peripheral base address                                                                                                                           |
| <i>chnlNumber</i> | The channel number                                                                                                                                    |
| <i>level</i>      | The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field. |

### 35.5.7 void TPM\_SetupInputCapture ( **TPM\_Type** \* *base*, **tpm\_chnl\_t** *chnlNumber*, **tpm\_input\_capture\_edge\_t** *captureMode* )

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>base</i>        | TPM peripheral base address     |
| <i>chnlNumber</i>  | The channel number              |
| <i>captureMode</i> | Specifies which edge to capture |

### 35.5.8 void TPM\_SetupOutputCompare ( **TPM\_Type** \* *base*, **tpm\_chnl\_t** *chnlNumber*, **tpm\_output\_compare\_mode\_t** *compareMode*, **uint32\_t** *compareValue* )

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

## Function Documentation

Parameters

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| <i>base</i>         | TPM peripheral base address                                            |
| <i>chnlNumber</i>   | The channel number                                                     |
| <i>compareMode</i>  | Action to take on the channel output when the compare condition is met |
| <i>compareValue</i> | Value to be programmed in the CnV register.                            |

**35.5.9 void TPM\_SetupDualEdgeCapture ( TPM\_Type \* *base*, tpm\_chnl\_t *chnlPairNumber*, const tpm\_dual\_edge\_capture\_param\_t \* *edgeParam*, uint32\_t *filterValue* )**

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

|                        |                                                     |
|------------------------|-----------------------------------------------------|
| <i>base</i>            | TPM peripheral base address                         |
| <i>chnlPair-Number</i> | The TPM channel pair number; options are 0, 1, 2, 3 |
| <i>edgeParam</i>       | Sets up the dual edge capture function              |
| <i>filterValue</i>     | Filter value, specify 0 to disable filter.          |

**35.5.10 void TPM\_SetupQuadDecode ( TPM\_Type \* *base*, const tpm\_phase\_params\_t \* *phaseAParams*, const tpm\_phase\_params\_t \* *phaseBParams*, tpm\_quad\_decode\_mode\_t *quadMode* )**

Parameters

|                     |                                                       |
|---------------------|-------------------------------------------------------|
| <i>base</i>         | TPM peripheral base address                           |
| <i>phaseAParams</i> | Phase A configuration parameters                      |
| <i>phaseBParams</i> | Phase B configuration parameters                      |
| <i>quadMode</i>     | Selects encoding mode used in quadrature decoder mode |

**35.5.11 void TPM\_EnableInterrupts ( TPM\_Type \* *base*, uint32\_t *mask* )**

Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | TPM peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">tpm_interrupt_enable_t</a> |

### 35.5.12 void TPM\_DisableInterrupts ( TPM\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | TPM peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">tpm_interrupt_enable_t</a> |

### 35.5.13 uint32\_t TPM\_GetEnabledInterrupts ( TPM\_Type \* *base* )

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [tpm\\_interrupt\\_enable\\_t](#)

### 35.5.14 static uint32\_t TPM\_GetStatusFlags ( TPM\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [tpm\\_status\\_flags\\_t](#)

### 35.5.15 static void TPM\_ClearStatusFlags ( TPM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Function Documentation

Parameters

|             |                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | TPM peripheral base address                                                                                      |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">tpm_status_flags_t</a> |

### 35.5.16 static void TPM\_SetTimerPeriod ( **TPM\_Type** \* *base*, **uint32\_t** *ticks* ) [**inline**], [**static**]

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note

1. This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.
2. Call the utility macros provided in the `fsl_common.h` to convert usec or msec to ticks.

Parameters

|              |                                                                            |
|--------------|----------------------------------------------------------------------------|
| <i>base</i>  | TPM peripheral base address                                                |
| <i>ticks</i> | A timer period in units of ticks, which should be equal or greater than 1. |

### 35.5.17 static uint32\_t TPM\_GetCurrentTimerCount ( **TPM\_Type** \* *base* ) [**inline**], [**static**]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the `fsl_common.h` to convert ticks to usec or msec.

Parameters

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

Returns

The current counter value in ticks

35.5.18 **static void TPM\_StartTimer ( TPM\_Type \* *base*, tpm\_clock\_source\_t *clockSource* ) [inline], [static]**

## Function Documentation

Parameters

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>base</i>        | TPM peripheral base address                                               |
| <i>clockSource</i> | TPM clock source; once clock source is set the counter will start running |

### 35.5.19 static void TPM\_StopTimer ( TPM\_Type \* *base* ) [inline], [static]

Parameters

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

### 35.5.20 static void TPM\_Reset ( TPM\_Type \* *base* ) [inline], [static]

Reset all internal logic and registers, except the Global Register. Remains set until cleared by software.

Note

TPM software reset is available on certain SoC's only

Parameters

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

# Chapter 36

## TSTMR: Timestamp Timer Driver

### 36.1 Overview

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

### Functions

- static uint64\_t **TSTMR\_ReadTimeStamp** (TSTMR\_Type \*base)  
*Reads the time stamp.*
- static void **TSTMR\_DelayUs** (TSTMR\_Type \*base, uint32\_t delayInUs)  
*Delays for a specified number of microseconds.*

### Driver version

- #define **FSL\_TSTMR\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 0))  
*Version 2.0.0.*

### 36.2 Function Documentation

#### 36.2.1 static uint64\_t TSTMR\_ReadTimeStamp ( **TSTMR\_Type \* base** ) [**inline**], [**static**]

This function reads the low and high registers and returns the 56-bit free running counter value. This can be read by software at any time to determine the software ticks.

Parameters

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

Returns

The 56-bit time stamp value.

#### 36.2.2 static void TSTMR\_DelayUs ( **TSTMR\_Type \* base**, **uint32\_t delayInUs** ) [**inline**], [**static**]

This function repeatedly reads the timestamp register and waits for the user-specified delay value.

## Function Documentation

### Parameters

|                  |                                |
|------------------|--------------------------------|
| <i>base</i>      | TSTMR peripheral base address. |
| <i>delayInUs</i> | Delay value in microseconds.   |

# Chapter 37

## WDOG32: 32-bit Watchdog Timer

### 37.1 Overview

The MCUXpresso SDK provides a peripheral driver for the WDOG32 module of MCUXpresso SDK devices.

### 37.2 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/wdog32

## Data Structures

- struct `wdog32_work_mode_t`  
*Defines WDOG32 work mode. [More...](#)*
- struct `wdog32_config_t`  
*Describes WDOG32 configuration structure. [More...](#)*

## Enumerations

- enum `wdog32_clock_source_t`{  
  kWDOG32\_ClockSource0 = 0U,  
  kWDOG32\_ClockSource1 = 1U,  
  kWDOG32\_ClockSource2 = 2U,  
  kWDOG32\_ClockSource3 = 3U }  
*Describes WDOG32 clock source.*
- enum `wdog32_clock_prescaler_t`{  
  kWDOG32\_ClockPrescalerDivide1 = 0x0U,  
  kWDOG32\_ClockPrescalerDivide256 = 0x1U }  
*Describes the selection of the clock prescaler.*
- enum `wdog32_test_mode_t`{  
  kWDOG32\_TestModeDisabled = 0U,  
  kWDOG32\_UserModeEnabled = 1U,  
  kWDOG32\_LowByteTest = 2U,  
  kWDOG32\_HighByteTest = 3U }  
*Describes WDOG32 test mode.*
- enum `_wdog32_interrupt_enable_t` { kWDOG32\_InterruptEnable = WDOG\_CS\_INT\_MASK }  
*WDOG32 interrupt configuration structure.*
- enum `_wdog32_status_flags_t`{  
  kWDOG32\_RunningFlag = WDOG\_CS\_EN\_MASK,  
  kWDOG32\_InterruptFlag = WDOG\_CS\_FLG\_MASK }  
*WDOG32 status flags.*

## Typical use case

### Unlock sequence

- #define **WDOG\_FIRST\_WORD\_OF\_UNLOCK** (WDOG\_UPDATE\_KEY & 0xFFFFU)  
*First word of unlock sequence.*
- #define **WDOG\_SECOND\_WORD\_OF\_UNLOCK** ((WDOG\_UPDATE\_KEY >> 16U) & 0xFF-FFU)  
*Second word of unlock sequence.*

### Refresh sequence

- #define **WDOG\_FIRST\_WORD\_OF\_REFRESH** (WDOG\_REFRESH\_KEY & 0xFFFFU)  
*First word of refresh sequence.*
- #define **WDOG\_SECOND\_WORD\_OF\_REFRESH** ((WDOG\_REFRESH\_KEY >> 16U) & 0xF-FFFU)  
*Second word of refresh sequence.*

### Driver version

- #define **FSL\_WDOG32\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
*WDOG32 driver version 2.0.1.*

## WDOG32 Initialization and De-initialization

- void **WDOG32\_GetDefaultConfig** (wdog32\_config\_t \*config)  
*Initializes the WDOG32 configuration structure.*
- **AT\_QUICKACCESS\_SECTION\_CODE** (void WDOG32\_Init(WDOG\_Type \*base, const wdog32\_config\_t \*config))  
*Initializes the WDOG32 module.*
- void **WDOG32\_Deinit** (WDOG\_Type \*base)  
*De-initializes the WDOG32 module.*

## WDOG32 functional Operation

- static void **WDOG32\_Enable** (WDOG\_Type \*base)  
*Enables the WDOG32 module.*
- static void **WDOG32\_Disable** (WDOG\_Type \*base)  
*Disables the WDOG32 module.*
- static void **WDOG32\_EnableInterrupts** (WDOG\_Type \*base, uint32\_t mask)  
*Enables the WDOG32 interrupt.*
- static void **WDOG32\_DisableInterrupts** (WDOG\_Type \*base, uint32\_t mask)  
*Disables the WDOG32 interrupt.*
- static uint32\_t **WDOG32\_GetStatusFlags** (WDOG\_Type \*base)  
*Gets the WDOG32 all status flags.*
- **AT\_QUICKACCESS\_SECTION\_CODE** (void WDOG32\_ClearStatusFlags(WDOG\_Type \*base, uint32\_t mask))  
*Clears the WDOG32 flag.*
- static void **WDOG32\_SetTimeoutValue** (WDOG\_Type \*base, uint16\_t timeoutCount)  
*Sets the WDOG32 timeout value.*
- static void **WDOG32\_SetWindowValue** (WDOG\_Type \*base, uint16\_t windowValue)  
*Sets the WDOG32 window value.*

- static void **WDOG32\_Unlock** (WDOG\_Type \*base)  
*Unlocks the WDOG32 register written.*
- static void **WDOG32\_Refresh** (WDOG\_Type \*base)  
*Refreshes the WDOG32 timer.*
- static uint16\_t **WDOG32\_GetCounterValue** (WDOG\_Type \*base)  
*Gets the WDOG32 counter value.*

## 37.3 Data Structure Documentation

### 37.3.1 struct wdog32\_work\_mode\_t

#### Data Fields

- bool **enableWait**  
*Enables or disables WDOG32 in wait mode.*
- bool **enableStop**  
*Enables or disables WDOG32 in stop mode.*
- bool **enableDebug**  
*Enables or disables WDOG32 in debug mode.*

### 37.3.2 struct wdog32\_config\_t

#### Data Fields

- bool **enableWdog32**  
*Enables or disables WDOG32.*
- **wdog32\_clock\_source\_t clockSource**  
*Clock source select.*
- **wdog32\_clock\_prescaler\_t prescaler**  
*Clock prescaler value.*
- **wdog32\_work\_mode\_t workMode**  
*Configures WDOG32 work mode in debug stop and wait mode.*
- **wdog32\_test\_mode\_t testMode**  
*Configures WDOG32 test mode.*
- bool **enableUpdate**  
*Update write-once register enable.*
- bool **enableInterrupt**  
*Enables or disables WDOG32 interrupt.*
- bool **enableWindowMode**  
*Enables or disables WDOG32 window mode.*
- uint16\_t **windowValue**  
*Window value.*
- uint16\_t **timeoutValue**  
*Timeout value.*

## Enumeration Type Documentation

### 37.4 Macro Definition Documentation

37.4.1 `#define FSL_WDOG32_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

### 37.5 Enumeration Type Documentation

#### 37.5.1 enum wdog32\_clock\_source\_t

Enumerator

*kWDOG32\_ClockSource0* Clock source 0.

*kWDOG32\_ClockSource1* Clock source 1.

*kWDOG32\_ClockSource2* Clock source 2.

*kWDOG32\_ClockSource3* Clock source 3.

#### 37.5.2 enum wdog32\_clock\_prescaler\_t

Enumerator

*kWDOG32\_ClockPrescalerDivide1* Divided by 1.

*kWDOG32\_ClockPrescalerDivide256* Divided by 256.

#### 37.5.3 enum wdog32\_test\_mode\_t

Enumerator

*kWDOG32\_TestModeDisabled* Test Mode disabled.

*kWDOG32\_UserModeEnabled* User Mode enabled.

*kWDOG32\_LowByteTest* Test Mode enabled, only low byte is used.

*kWDOG32\_HighByteTest* Test Mode enabled, only high byte is used.

#### 37.5.4 enum \_wdog32\_interrupt\_enable\_t

This structure contains the settings for all of the WDOG32 interrupt configurations.

Enumerator

*kWDOG32\_InterruptEnable* Interrupt is generated before forcing a reset.

### 37.5.5 enum \_wdog32\_status\_flags\_t

This structure contains the WDOG32 status flags for use in the WDOG32 functions.

Enumerator

- kWDOG32\_RunningFlag* Running flag, set when WDOG32 is enabled.
- kWDOG32\_InterruptFlag* Interrupt flag, set when interrupt occurs.

## 37.6 Function Documentation

### 37.6.1 void WDOG32\_GetDefaultConfig ( *wdog32\_config\_t \* config* )

This function initializes the WDOG32 configuration structure to default values. The default values are:

```
* wdog32Config->enableWdog32 = true;
* wdog32Config->clockSource = kWDOG32_ClockSource1;
* wdog32Config->prescaler = kWDOG32_ClockPrescalerDivide1;
* wdog32Config->workMode.enableWait = true;
* wdog32Config->workMode.enableStop = false;
* wdog32Config->workMode.enableDebug = false;
* wdog32Config->testMode = kWDOG32_TestModeDisabled;
* wdog32Config->enableUpdate = true;
* wdog32Config->enableInterrupt = false;
* wdog32Config->enableWindowMode = false;
* wdog32Config->>windowValue = 0U;
* wdog32Config->timeoutValue = 0xFFFFU;
*
```

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>config</i> | Pointer to the WDOG32 configuration structure. |
|---------------|------------------------------------------------|

See Also

[wdog32\\_config\\_t](#)

### 37.6.2 AT\_QUICKACCESS\_SECTION\_CODE ( **void WDOG32\_InitWDOG\_Type** *\*base, const wdog32\_config\_t \*config* )

This function initializes the WDOG32. To reconfigure the WDOG32 without forcing a reset first, enableUpdate must be set to true in the configuration.

Example:

```
* wdog32_config_t config;
* WDOG32_GetDefaultConfig(&config);
* config.timeoutValue = 0x7ffU;
* config.enableUpdate = true;
* WDOG32_Init(wdog_base,&config);
*
```

## Function Documentation

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | WDOG32 peripheral base address.  |
| <i>config</i> | The configuration of the WDOG32. |

### 37.6.3 void WDOG32\_Deinit ( WDOG\_Type \* *base* )

This function shuts down the WDOG32. Ensure that the WDOG\_CS.UPDATE is 1, which means that the register update is enabled.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | WDOG32 peripheral base address. |
|-------------|---------------------------------|

### 37.6.4 static void WDOG32\_Enable ( WDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_CS register to enable the WDOG32. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | WDOG32 peripheral base address. |
|-------------|---------------------------------|

### 37.6.5 static void WDOG32\_Disable ( WDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_CS register to disable the WDOG32. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | WDOG32 peripheral base address |
|-------------|--------------------------------|

### 37.6.6 static void WDOG32\_EnableInterrupts ( **WDOG\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

This function writes a value into the WDOG\_CS register to enable the WDOG32 interrupt. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|             |                                                                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | WDOG32 peripheral base address.                                                                                                                                            |
| <i>mask</i> | The interrupts to enable. The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kWDOG32_InterruptEnable</li></ul> |

### 37.6.7 static void WDOG32\_DisableInterrupts ( **WDOG\_Type** \* *base*, **uint32\_t** *mask* ) [inline], [static]

This function writes a value into the WDOG\_CS register to disable the WDOG32 interrupt. The WDOG\_CS register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|             |                                                                                                                                                                              |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | WDOG32 peripheral base address.                                                                                                                                              |
| <i>mask</i> | The interrupts to disabled. The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kWDOG32_InterruptEnable</li></ul> |

### 37.6.8 static **uint32\_t** WDOG32\_GetStatusFlags ( **WDOG\_Type** \* *base* ) [inline], [static]

This function gets all status flags.

Example to get the running flag:

```
* uint32_t status;
* status = WDOG32_GetStatusFlags(wdog_base) &
* kWDOG32_RunningFlag;
```

## Function Documentation

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | WDOG32 peripheral base address |
|-------------|--------------------------------|

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

[\\_wdog32\\_status\\_flags\\_t](#)

- true: related status flag has been set.
- false: related status flag is not set.

### 37.6.9 AT\_QUICKACCESS\_SECTION\_CODE ( void WDOG32\_ClearStatusFlagsW- DOG\_Type \**base*, uint32\_t *mask* )

This function clears the WDOG32 status flag.

Example to clear an interrupt flag:

```
* WDOG32_ClearStatusFlags (wdog_base, kWDOG32_InterruptFlag);
*
```

Parameters

|             |                                                                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | WDOG32 peripheral base address.                                                                                                                                  |
| <i>mask</i> | The status flags to clear. The parameter can be any combination of the following values: <ul style="list-style-type: none"><li>• kWDOG32_InterruptFlag</li></ul> |

### 37.6.10 static void WDOG32\_SetTimeoutValue ( WDOG\_Type \* *base*, uint16\_t *timeoutCount* ) [inline], [static]

This function writes a timeout value into the WDOG\_TOVAL register. The WDOG\_TOVAL register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|                     |                                                    |
|---------------------|----------------------------------------------------|
| <i>base</i>         | WDOG32 peripheral base address                     |
| <i>timeoutCount</i> | WDOG32 timeout value, count of WDOG32 clock ticks. |

### 37.6.11 static void WDOG32\_SetWindowValue ( WDOG\_Type \* *base*, uint16\_t *windowValue* ) [inline], [static]

This function writes a window value into the WDOG\_WIN register. The WDOG\_WIN register is a write-once register. Ensure that the WCT window is still open and this register has not been written in this WCT while the function is called.

Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>base</i>        | WDOG32 peripheral base address. |
| <i>windowValue</i> | WDOG32 window value.            |

### 37.6.12 static void WDOG32\_Unlock ( WDOG\_Type \* *base* ) [inline], [static]

This function unlocks the WDOG32 register written.

Before starting the unlock sequence and following the configuration, disable the global interrupts. Otherwise, an interrupt could effectively invalidate the unlock sequence and the WCT may expire. After the configuration finishes, re-enable the global interrupts.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | WDOG32 peripheral base address |
|-------------|--------------------------------|

### 37.6.13 static void WDOG32\_Refresh ( WDOG\_Type \* *base* ) [inline], [static]

This function feeds the WDOG32. This function should be called before the Watchdog timer is in timeout. Otherwise, a reset is asserted.

## Function Documentation

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | WDOG32 peripheral base address |
|-------------|--------------------------------|

**37.6.14 static uint16\_t WDOG32\_GetCounterValue ( WDOG\_Type \* *base* )  
[inline], [static]**

This function gets the WDOG32 counter value.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | WDOG32 peripheral base address. |
|-------------|---------------------------------|

Returns

Current WDOG32 counter value.

# Chapter 38

## Serial Manager

### 38.1 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, usbcdc and swo.

### Modules

- Serial Port SWO
- Serial Port USB
- Serial Port Uart

### Data Structures

- struct `serial_manager_config_t`  
*serial manager config structure. [More...](#)*
- struct `serial_manager_callback_message_t`  
*Callback message structure. [More...](#)*

### Macros

- #define `SERIAL_PORT_TYPE_UART` (1U)  
*Enable or disable uart 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_MANAGER_WRITE_HANDLE_SIZE` (4U)  
*Set serial manager write handle size.*
- #define `SERIAL_MANAGER_HANDLE_SIZE` (`SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U`)  
*`SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated size.`*

### Typedefs

- typedef void(\* `serial_manager_callback_t` )(void \*callbackParam, `serial_manager_callback_message_t` \*message, `serial_manager_status_t` status)  
*callback function*

## Overview

### Enumerations

- enum `serial_port_type_t` {  
    `kSerialPort_Uart` = 1U,  
    `kSerialPort_UsbCdc`,  
    `kSerialPort_Swo` }  
    *serial port 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` }  
    *serial manager error code*

### Functions

- `serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, serial_manager_config_t *config)`  
    *Initializes a serial manager module with the serial manager handle and the user configuration structure.*
- `serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)`  
    *De-initializes the serial manager module instance.*
- `serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_write_handle_t writeHandle)`  
    *Opens a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)`  
    *Closes a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_read_handle_t readHandle)`  
    *Opens a reading handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t readHandle)`  
    *Closes a reading for the serial manager module.*
- `serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t writeHandle, uint8_t *buffer, uint32_t length)`  
    *Transmits data with the blocking mode.*
- `serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length)`  
    *Reads data with the blocking mode.*
- `serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)`  
    *Prepares to enter low power consumption.*
- `serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t serialHandle)`  
    *Restores from low power consumption.*

## 38.2 Data Structure Documentation

### 38.2.1 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.*
- `void * portConfig`  
*Serial port configuration.*

#### 38.2.1.0.0.97 Field Documentation

##### 38.2.1.0.0.97.1 `uint8_t* serial_manager_config_t::ringBuffer`

Besides, the memory space cannot be free during the lifetime of the serial manager module.

## 38.2.2 struct serial\_manager\_callback\_message\_t

#### Data Fields

- `uint8_t * buffer`  
*Transferred buffer.*
- `uint32_t length`  
*Transferred data length.*

## 38.3 Enumeration Type Documentation

### 38.3.1 enum serial\_port\_type\_t

Enumerator

- `kSerialPort_Uart` Serial port UART.
- `kSerialPort_UsbCdc` Serial port USB CDC.
- `kSerialPort_Swo` Serial port SWO.

### 38.3.2 enum serial\_manager\_status\_t

Enumerator

- `kStatus_SerialManager_Success` Success.
- `kStatus_SerialManager_Error` Failed.

## Function Documentation

*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.

## 38.4 Function Documentation

### 38.4.1 `serial_manager_status_t SerialManager_Init ( serial_handle_t serialHandle, serial_manager_config_t * config )`

This function configures the serial manager module with user-defined settings. The user can configure the configuration structure. The parameter `serialHandle` is a pointer to point to a memory space of size `SERIAL_MANAGER_HANDLE_SIZE` allocated by the caller. The serial manager module supports two types of serial port, uart (includes UART, USART, LPSCI, LPUART, etc) and USB CDC. Please refer to `serial_port_type_t` for serial port setting. These two 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 uint8_t s_serialHandleBuffer[SERIAL_MANAGER_HANDLE_SIZE];
* static serial_handle_t s_serialHandle = &s_serialHandleBuffer[0];
* 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;
* config.portConfig = &uartConfig;
* SerialManager_Init(s_serialHandle, &config);
*
```

For USB CDC,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static uint8_t s_serialHandleBuffer[SERIAL_MANAGER_HANDLE_SIZE];
* static serial_handle_t s_serialHandle = &s_serialHandleBuffer[0];
* 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(s_serialHandle, &config);
*
```

Parameters

|                     |                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | Pointer to point to a memory space of size <a href="#">SERIAL_MANAGER_HANDLE_SIZE</a> allocated by the caller. |
| <i>config</i>       | Pointer to user-defined configuration structure.                                                               |

Return values

|                                      |                                                   |
|--------------------------------------|---------------------------------------------------|
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                                |
| <i>kStatus_SerialManager_Success</i> | The serial manager module initialization succeed. |

### 38.4.2 **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. |

### 38.4.3 **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.

## Function Documentation

### Parameters

|                     |                                                   |
|---------------------|---------------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer.         |
| <i>writeHandle</i>  | The serial manager module writing handle pointer. |

### Return values

|                                             |                                |
|---------------------------------------------|--------------------------------|
| <i>kStatus_SerialManager_Error</i>          | An error occurred.             |
| <i>kStatus_SerialManager_HandleConflict</i> | The writing handle was opened. |
| <i>kStatus_SerialManager_Success</i>        | The writing handle is opened.  |

Example below shows how to use this API to write data. For task 1,

```
* static uint8_t s_serialWriteHandleBuffer1[SERIAL_MANAGER_WRITE_HANDLE_SIZE
];
* static serial_write_handle_t s_serialWriteHandle1 = &s_serialWriteHandleBuffer1[0];
* static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
* SerialManager_OpenWriteHandle(serialHandle, s_serialWriteHandle1);
* SerialManager_InstallTxCallback(s_serialWriteHandle1, Task1_SerialManagerTxCallback,
 s_serialWriteHandle1);
* SerialManager_WriteNonBlocking(s_serialWriteHandle1, s_nonBlockingWelcome1, sizeof(
 s_nonBlockingWelcome1) - 1);
*
```

For task 2,

```
* static uint8_t s_serialWriteHandleBuffer2[SERIAL_MANAGER_WRITE_HANDLE_SIZE
];
* static serial_write_handle_t s_serialWriteHandle2 = &s_serialWriteHandleBuffer2[0];
* static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
* SerialManager_OpenWriteHandle(serialHandle, s_serialWriteHandle2);
* SerialManager_InstallTxCallback(s_serialWriteHandle2, Task2_SerialManagerTxCallback,
 s_serialWriteHandle2);
* SerialManager_WriteNonBlocking(s_serialWriteHandle2, s_nonBlockingWelcome2, sizeof(
 s_nonBlockingWelcome2) - 1);
*
```

### 38.4.4 **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. |
|---------------------------------------|-------------------------------|

**38.4.5 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.         |
| <i>readHandle</i>   | The serial manager module reading handle pointer. |

## 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 uint8_t s_serialReadHandleBuffer[SERIAL_MANAGER_READ_HANDLE_SIZE];
* static serial_read_handle_t s_serialReadHandle = &s_serialReadHandleBuffer[0];
* SerialManager_OpenReadHandle(serialHandle, s_serialReadHandle);
* static uint8_t s_nonBlockingBuffer[64];
* SerialManager_InstallRxCallback(s_serialReadHandle, APP_SerialManagerRxCallback, s_serialReadHandle);
* SerialManager_ReadNonBlocking(s_serialReadHandle, s_nonBlockingBuffer, sizeof(s_nonBlockingBuffer));
*
```

**38.4.6 serial\_manager\_status\_t SerialManager\_CloseReadHandle ( serial\_read\_handle\_t readHandle )**

This function Closes a reading for the serial manager module.

## Function Documentation

Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>readHandle</i> | The serial manager module reading handle pointer. |
|-------------------|---------------------------------------------------|

Return values

|                                       |                               |
|---------------------------------------|-------------------------------|
| <i>kStatus_SerialManager_-Success</i> | The reading handle is closed. |
|---------------------------------------|-------------------------------|

### 38.4.7 **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_-<br/>Error</i> | An error occurred. |
|------------------------------------------|--------------------|

### 38.4.8 **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_-<br/>Success</i> | Successfully received all data.                                      |
| <i>kStatus_SerialManager_-<br/>Busy</i>    | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_-<br/>Error</i>   | An error occurred.                                                   |

### 38.4.9 **serial\_manager\_status\_t SerialManager\_EnterLowpower ( serial\_handle\_t *serialHandle* )**

This function is used to prepare to enter low power consumption.

## Function Documentation

Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

Return values

|                                       |                       |
|---------------------------------------|-----------------------|
| <i>kStatus_SerialManager_-Success</i> | Successful operation. |
|---------------------------------------|-----------------------|

### **38.4.10 `serial_manager_status_t SerialManager_ExitLowpower ( serial_handle_t serialHandle )`**

This function is used to restore from low power consumption.

Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

Return values

|                                       |                       |
|---------------------------------------|-----------------------|
| <i>kStatus_SerialManager_-Success</i> | Successful operation. |
|---------------------------------------|-----------------------|

## 38.5 Serial Port Uart

### 38.5.1 Overview

#### Data Structures

- struct `serial_port_uart_config_t`  
*serial port uart config struct More...*

#### Macros

- #define `SERIAL_PORT_UART_HANDLE_SIZE` (4U)  
*serial port uart handle size*

#### Enumerations

- enum `serial_port_uart_parity_mode_t` {
   
`kSerialManager_UartParityDisabled` = 0x0U,  
`kSerialManager_UartParityEven` = 0x1U,  
`kSerialManager_UartParityOdd` = 0x2U }  
*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*

### 38.5.2 Data Structure Documentation

#### 38.5.2.1 struct `serial_port_uart_config_t`

##### Data Fields

- `uint32_t clockRate`  
*clock rate*
- `uint32_t baudRate`  
*baud rate*
- `serial_port_uart_parity_mode_t parityMode`  
*Parity mode, disabled (default), even, odd.*
- `serial_port_uart_stop_bit_count_t stopBitCount`  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- `uint8_t instance`  
*Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.*
- `uint8_t enableRx`  
*Enable RX.*
- `uint8_t enableTx`

## Serial Port Uart

*Enable TX.*

### 38.5.2.1.0.98 Field Documentation

#### 38.5.2.1.0.98.1 `uint8_t serial_port_uart_config_t::instance`

### 38.5.3 Enumeration Type Documentation

#### 38.5.3.1 `enum serial_port_uart_parity_mode_t`

Enumerator

*kSerialManager\_UartParityDisabled* Parity disabled.

*kSerialManager\_UartParityEven* Parity even enabled.

*kSerialManager\_UartParityOdd* Parity odd enabled.

#### 38.5.3.2 `enum serial_port_uart_stop_bit_count_t`

Enumerator

*kSerialManager\_UartOneStopBit* One stop bit.

*kSerialManager\_UartTwoStopBit* Two stop bits.

## 38.6 Serial Port USB

### 38.6.1 Overview

#### Modules

- [USB Device Configuration](#)

#### Data Structures

- struct [serial\\_port\\_usb\\_cdc\\_config\\_t](#)  
*serial port usb config struct* [More...](#)

#### Macros

- #define [SERIAL\\_PORT\\_USB\\_CDC\\_HANDLE\\_SIZE](#) (72)  
*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.*

### 38.6.2 Data Structure Documentation

#### 38.6.2.1 struct serial\_port\_usb\_cdc\_config\_t

##### Data Fields

- `serial_port_usb_cdc_controller_index_t controllerIndex`  
*controller index*

### 38.6.3 Enumeration Type Documentation

#### 38.6.3.1 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.

## 38.6.4 USB Device Configuration

### 38.6.4.1 Overview

#### Macros

- #define **USB\_DEVICE\_CONFIG\_SELF\_POWER** (1U)  
*Whether device is self power.*
- #define **USB\_DEVICE\_CONFIG\_ENDPOINTS** (4U)  
*How many endpoints are supported in the stack.*
- #define **USB\_DEVICE\_CONFIG\_USE\_TASK** (0U)  
*Whether the device task is enabled.*
- #define **USB\_DEVICE\_CONFIG\_MAX\_MESSAGES** (8U)  
*How many the notification message are supported when the device task is enabled.*
- #define **USB\_DEVICE\_CONFIG\_USB20\_TEST\_MODE** (0U)  
*Whether test mode enabled.*
- #define **USB\_DEVICE\_CONFIG\_CV\_TEST** (0U)  
*Whether device CV test is enabled.*
- #define **USB\_DEVICE\_CONFIG\_COMPLIANCE\_TEST** (0U)  
*Whether device compliance test is enabled.*
- #define **USB\_DEVICE\_CONFIG\_KEEP\_ALIVE\_MODE** (0U)  
*Whether the keep alive feature enabled.*
- #define **USB\_DEVICE\_CONFIG\_BUFFER\_PROPERTY\_CACHEABLE** (0U)  
*Whether the transfer buffer is cache-enabled or not.*
- #define **USB\_DEVICE\_CONFIG\_LOW\_POWER\_MODE** (0U)  
*Whether the low power mode is enabled or not.*
- #define **USB\_DEVICE\_CONFIG\_REMOTE\_WAKEUP** (0U)  
*The device remote wakeup is unsupported.*
- #define **USB\_DEVICE\_CONFIG\_DETACH\_ENABLE** (0U)  
*Whether the device detached feature is enabled or not.*
- #define **USB\_DEVICE\_CONFIG\_ERROR\_HANDLING** (0U)  
*Whether handle the USB bus error.*
- #define **USB\_DEVICE\_CHARGER\_DETECT\_ENABLE** (0U)  
*Whether the device charger detect feature is enabled or not.*

#### class instance define

- #define **USB\_DEVICE\_CONFIG\_HID** (0U)  
*HID instance count.*
- #define **USB\_DEVICE\_CONFIG\_CDC\_ACM** (1U)  
*CDC ACM instance count.*
- #define **USB\_DEVICE\_CONFIG\_MSC** (0U)  
*MSC instance count.*
- #define **USB\_DEVICE\_CONFIG\_AUDIO** (0U)  
*Audio instance count.*
- #define **USB\_DEVICE\_CONFIG\_PHDC** (0U)  
*PHDC instance count.*
- #define **USB\_DEVICE\_CONFIG\_VIDEO** (0U)  
*Video instance count.*
- #define **USB\_DEVICE\_CONFIG\_CCID** (0U)

## Serial Port USB

- `#define USB_DEVICE_CONFIG_PRINTER (0U)`  
*Printer instance count.*
- `#define USB_DEVICE_CONFIG_DFU (0U)`  
*DFU instance count.*

### 38.6.4.2 Macro Definition Documentation

#### 38.6.4.2.1 `#define USB_DEVICE_CONFIG_SELF_POWER (1U)`

1U supported, 0U not supported

#### 38.6.4.2.2 `#define USB_DEVICE_CONFIG_ENDPOINTS (4U)`

#### 38.6.4.2.3 `#define USB_DEVICE_CONFIG_USE_TASK (0U)`

#### 38.6.4.2.4 `#define USB_DEVICE_CONFIG_MAX_MESSAGES (8U)`

#### 38.6.4.2.5 `#define USB_DEVICE_CONFIG_USB20_TEST_MODE (0U)`

#### 38.6.4.2.6 `#define USB_DEVICE_CONFIG_CV_TEST (0U)`

#### 38.6.4.2.7 `#define USB_DEVICE_CONFIG_COMPLIANCE_TEST (0U)`

If the macro is enabled, the test mode and CV test macros will be set.

#### 38.6.4.2.8 `#define USB_DEVICE_CONFIG_KEEP_ALIVE_MODE (0U)`

#### 38.6.4.2.9 `#define USB_DEVICE_CONFIG_BUFFER_PROPERTY_CACHEABLE (0U)`

#### 38.6.4.2.10 `#define USB_DEVICE_CONFIG_LOW_POWER_MODE (0U)`

#### 38.6.4.2.11 `#define USB_DEVICE_CONFIG_REMOTE_WAKEUP (0U)`

#### 38.6.4.2.12 `#define USB_DEVICE_CONFIG_DETACH_ENABLE (0U)`

#### 38.6.4.2.13 `#define USB_DEVICE_CONFIG_ERROR_HANDLING (0U)`

#### 38.6.4.2.14 `#define USB_DEVICE_CHARGER_DETECT_ENABLE (0U)`

## 38.7 Serial Port SWO

### 38.7.1 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*

### 38.7.2 Data Structure Documentation

#### 38.7.2.1 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.*

### 38.7.3 Enumeration Type Documentation

#### 38.7.3.1 enum `serial_port_swo_protocol_t`

Enumerator

`kSerialManager_SwoProtocolManchester` SWO Manchester protocol.  
`kSerialManager_SwoProtocolNrz` SWO UART/NRZ protocol.



# Chapter 39

## GenericList

### 39.1 Overview

#### Data Structures

- struct `list_handle_t`  
*The list structure.* [More...](#)
- struct `list_element_handle_t`  
*The list element.* [More...](#)

#### Enumerations

- enum `list_status_t` {  
  `kLIST_Ok` = `kStatus_Success`,  
  `kLIST_Full` = `MAKE_STATUS(kStatusGroup_LIST, 1)`,  
  `kLIST_Empty` = `MAKE_STATUS(kStatusGroup_LIST, 2)`,  
  `kLIST_OrphanElement` = `MAKE_STATUS(kStatusGroup_LIST, 3)` }

#### Functions

- void `LIST_Init` (`list_handle_t` list, `uint32_t` max)
- `list_handle_t LIST_GetList` (`list_element_handle_t` element)  
*Gets the list that contains the given element.*
- `list_status_t LIST_AddHead` (`list_handle_t` list, `list_element_handle_t` element)  
*Links element to the head of the list.*
- `list_status_t LIST_AddTail` (`list_handle_t` list, `list_element_handle_t` element)  
*Links element to the tail of the list.*
- `list_element_handle_t LIST_RemoveHead` (`list_handle_t` list)  
*Unlinks element from the head of the list.*
- `list_element_handle_t LIST_GetHead` (`list_handle_t` list)  
*Gets head element handle.*
- `list_element_handle_t LIST_GetNext` (`list_element_handle_t` element)  
*Gets next element handle for given element handle.*
- `list_element_handle_t LIST_GetPrev` (`list_element_handle_t` element)  
*Gets previous element handle for given element handle.*
- `list_status_t LIST_RemoveElement` (`list_element_handle_t` element)  
*Unlinks an element from its list.*
- `list_status_t LIST_AddPrevElement` (`list_element_handle_t` element, `list_element_handle_t` newElement)  
*Links an element in the previous position relative to a given member of a list.*
- `uint32_t LIST.GetSize` (`list_handle_t` list)  
*Gets the current size of a list.*
- `uint32_t LIST_GetAvailableSize` (`list_handle_t` list)

## Enumeration Type Documentation

*Gets the number of free places in the list.*

## 39.2 Data Structure Documentation

### 39.2.1 struct list\_t

#### Data Fields

- struct list\_element\_tag \* **head**  
*list head*
- struct list\_element\_tag \* **tail**  
*list tail*
- uint16\_t **size**  
*list size*
- uint16\_t **max**  
*list max number of elements*

### 39.2.2 struct list\_element\_t

#### Data Fields

- struct list\_element\_tag \* **next**  
*next list element*
- struct list\_element\_tag \* **prev**  
*previous list element*
- struct list\_tag \* **list**  
*pointer to the list*

## 39.3 Enumeration Type Documentation

### 39.3.1 enum list\_status\_t

Include

Public macro definitions

Public type definitions

The list status

Enumerator

- kLIST\_Ok** Success.  
**kLIST\_Full** FULL.  
**kLIST\_Empty** Empty.  
**kLIST\_OrphanElement** Orphan Element.

## 39.4 Function Documentation

### 39.4.1 void LIST\_Init ( *list\_handle\_t list*, *uint32\_t max* )

Public prototypes

Initialize the list.

This function initialize the list.

Parameters

|             |                                                        |
|-------------|--------------------------------------------------------|
| <i>list</i> | - List handle to initialize.                           |
| <i>max</i>  | - Maximum number of elements in list. 0 for unlimited. |

### 39.4.2 *list\_handle\_t* LIST\_GetList ( *list\_element\_handle\_t element* )

Parameters

|                |                          |
|----------------|--------------------------|
| <i>element</i> | - Handle of the element. |
|----------------|--------------------------|

Return values

|             |                                                                        |
|-------------|------------------------------------------------------------------------|
| <i>NULL</i> | if element is orphan, Handle of the list the element is inserted into. |
|-------------|------------------------------------------------------------------------|

### 39.4.3 *list\_status\_t* LIST\_AddHead ( *list\_handle\_t list*, *list\_element\_handle\_t element* )

Parameters

|                |                          |
|----------------|--------------------------|
| <i>list</i>    | - Handle of the list.    |
| <i>element</i> | - Handle of the element. |

Return values

|                   |                                                               |
|-------------------|---------------------------------------------------------------|
| <i>kLIST_Full</i> | if list is full, <i>kLIST_Ok</i> if insertion was successful. |
|-------------------|---------------------------------------------------------------|

### 39.4.4 *list\_status\_t* LIST\_AddTail ( *list\_handle\_t list*, *list\_element\_handle\_t element* )

## Function Documentation

Parameters

|                |                          |
|----------------|--------------------------|
| <i>list</i>    | - Handle of the list.    |
| <i>element</i> | - Handle of the element. |

Return values

|                   |                                                        |
|-------------------|--------------------------------------------------------|
| <i>kLIST_Full</i> | if list is full, kLIST_Ok if insertion was successful. |
|-------------------|--------------------------------------------------------|

### 39.4.5 **list\_element\_handle\_t LIST\_RemoveHead ( list\_handle\_t *list* )**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>list</i> | - Handle of the list. |
|-------------|-----------------------|

Return values

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>NULL</i> | if list is empty, handle of removed element(pointer) if removal was successful. |
|-------------|---------------------------------------------------------------------------------|

### 39.4.6 **list\_element\_handle\_t LIST\_GetHead ( list\_handle\_t *list* )**

Parameters

|             |                       |
|-------------|-----------------------|
| <i>list</i> | - Handle of the list. |
|-------------|-----------------------|

Return values

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>NULL</i> | if list is empty, handle of removed element(pointer) if removal was successful. |
|-------------|---------------------------------------------------------------------------------|

### 39.4.7 **list\_element\_handle\_t LIST\_GetNext ( list\_element\_handle\_t *element* )**

Parameters

|                |                          |
|----------------|--------------------------|
| <i>element</i> | - Handle of the element. |
|----------------|--------------------------|

Return values

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>NULL</i> | if list is empty, handle of removed element(pointer) if removal was successful. |
|-------------|---------------------------------------------------------------------------------|

### 39.4.8 **list\_element\_handle\_t LIST\_GetPrev ( list\_element\_handle\_t *element* )**

Parameters

|                |                          |
|----------------|--------------------------|
| <i>element</i> | - Handle of the element. |
|----------------|--------------------------|

Return values

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>NULL</i> | if list is empty, handle of removed element(pointer) if removal was successful. |
|-------------|---------------------------------------------------------------------------------|

### 39.4.9 **list\_status\_t LIST\_RemoveElement ( list\_element\_handle\_t *element* )**

Parameters

|                |                          |
|----------------|--------------------------|
| <i>element</i> | - Handle of the element. |
|----------------|--------------------------|

Return values

|                           |                                     |
|---------------------------|-------------------------------------|
| <i>kLIST_OphanElement</i> | if element is not part of any list. |
| <i>kLIST_Ok</i>           | if removal was successful.          |

### 39.4.10 **list\_status\_t LIST\_AddPrevElement ( list\_element\_handle\_t *element*, list\_element\_handle\_t *newElement* )**

Parameters

## Function Documentation

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <i>element</i>    | - Handle of the element.                         |
| <i>newElement</i> | - New element to insert before the given member. |

Return values

|                           |                                     |
|---------------------------|-------------------------------------|
| <i>kLIST_OphanElement</i> | if element is not part of any list. |
| <i>kLIST_Ok</i>           | if removal was successful.          |

### 39.4.11 uint32\_t LIST\_GetSize ( list\_handle\_t *list* )

Parameters

|             |                       |
|-------------|-----------------------|
| <i>list</i> | - Handle of the list. |
|-------------|-----------------------|

Return values

|                |                   |
|----------------|-------------------|
| <i>Current</i> | size of the list. |
|----------------|-------------------|

### 39.4.12 uint32\_t LIST\_GetAvailableSize ( list\_handle\_t *list* )

Parameters

|             |                       |
|-------------|-----------------------|
| <i>list</i> | - Handle of the list. |
|-------------|-----------------------|

Return values

|                  |                   |
|------------------|-------------------|
| <i>Available</i> | size of the list. |
|------------------|-------------------|

# Chapter 40

## UART\_Adapter

### 40.1 Overview

#### Data Structures

- struct `hal_uart_config_t`  
*uart configuration structure.* [More...](#)
- struct `hal_uart_transfer_t`  
*uart transfer structure.* [More...](#)

#### Macros

- `#define UART_ADAPTER_NON_BLOCKING_MODE (0U)`  
*Enable or disable Uart adapter non-blocking mode (1 - enable, 0 - disable)*
- `#define HAL_UART_TRANSFER_MODE (0U)`  
*Whether enable transactional function of the uart.*

#### Typedefs

- `typedef void(* hal_uart_transfer_callback_t )(hal_uart_handle_t handle, hal_uart_status_t status, void *callbackParam)`  
*uart transfer callback function.*

#### Enumerations

- enum `hal_uart_status_t` {  
  `kStatus_HAL_UartSuccess` = `kStatus_Success`,  
  `kStatus_HAL_UartTxBusy` = `MAKE_STATUS(kStatusGroup_HAL_UART, 1)`,  
  `kStatus_HAL_UartRxBusy` = `MAKE_STATUS(kStatusGroup_HAL_UART, 2)`,  
  `kStatus_HAL_UartTxIdle` = `MAKE_STATUS(kStatusGroup_HAL_UART, 3)`,  
  `kStatus_HAL_UartRxIdle` = `MAKE_STATUS(kStatusGroup_HAL_UART, 4)`,  
  `kStatus_HAL_UartBaudrateNotSupport`,  
  `kStatus_HAL_UartProtocolError`,  
  `kStatus_HAL_UartError` = `MAKE_STATUS(kStatusGroup_HAL_UART, 7)` }  
    *uart status*
- enum `hal_uart_parity_mode_t` {  
  `kHAL_UartParityDisabled` = `0x0U`,  
  `kHAL_UartParityEven` = `0x1U`,  
  `kHAL_UartParityOdd` = `0x2U` }  
    *uart parity mode.*
- enum `hal_uart_stop_bit_count_t` {  
  `kHAL_UartOneStopBit` = `0U`,  
  `kHAL_UartTwoStopBit` = `1U` }  
    *uart stop bit count.*

## Overview

## Functions

- void [HAL\\_UartIsrFunction](#) (hal\_uart\_handle\_t handle)  
*uart IRQ handle function.*

## Initialization and deinitialization

- [hal\\_uart\\_status\\_t HAL\\_UartInit](#) (hal\_uart\_handle\_t handle, [hal\\_uart\\_config\\_t](#) \*config)  
*Initializes a uart instance with the uart handle and the user configuration structure.*
- [hal\\_uart\\_status\\_t HAL\\_UartDeinit](#) (hal\_uart\_handle\_t handle)  
*Deinitializes a uart instance.*

## Blocking bus Operations

- [hal\\_uart\\_status\\_t HAL\\_UartReceiveBlocking](#) (hal\_uart\_handle\_t handle, uint8\_t \*data, size\_t length)  
*Reads RX data register using a blocking method.*
- [hal\\_uart\\_status\\_t HAL\\_UartSendBlocking](#) (hal\_uart\_handle\_t handle, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*

## Functional API with non-blocking mode.

### Note

The functional API and the transactional API cannot be used at the same time. The macro [HAL\\_UART\\_TRANSFER\\_MODE](#) is used to set which one will be used. If [HAL\\_UART\\_TRANSFER\\_MODE](#) is zero, the functional API with non-blocking mode will be used. Otherwise, transactional API will be used.

- [hal\\_uart\\_status\\_t HAL\\_UartInstallCallback](#) (hal\_uart\_handle\_t handle, [hal\\_uart\\_transfer\\_callback\\_t](#) callback, void \*callbackParam)  
*Installs a callback and callback parameter.*
- [hal\\_uart\\_status\\_t HAL\\_UartReceiveNonBlocking](#) (hal\_uart\_handle\_t handle, uint8\_t \*data, size\_t length)  
*Receives a buffer of data using an interrupt method.*
- [hal\\_uart\\_status\\_t HAL\\_UartSendNonBlocking](#) (hal\_uart\_handle\_t handle, const uint8\_t \*data, size\_t length)  
*Transmits a buffer of data using the interrupt method.*
- [hal\\_uart\\_status\\_t HAL\\_UartGetReceiveCount](#) (hal\_uart\_handle\_t handle, uint32\_t \*count)  
*Gets the number of bytes that have been received.*
- [hal\\_uart\\_status\\_t HAL\\_UartGetSendCount](#) (hal\_uart\_handle\_t handle, uint32\_t \*count)  
*Gets the number of bytes written to the uart TX register.*
- [hal\\_uart\\_status\\_t HAL\\_UartAbortReceive](#) (hal\_uart\_handle\_t handle)  
*Aborts the interrupt-driven data receiving.*
- [hal\\_uart\\_status\\_t HAL\\_UartAbortSend](#) (hal\_uart\_handle\_t handle)  
*Aborts the interrupt-driven data sending.*

## 40.2 Data Structure Documentation

### 40.2.1 struct hal\_uart\_config\_t

#### Data Fields

- `uint32_t srcClock_Hz`  
*Source clock.*
- `uint32_t baudRate_Bps`  
*Baud rate.*
- `hal_uart_parity_mode_t parityMode`  
*Parity mode, disabled (default), even, odd.*
- `hal_uart_stop_bit_count_t stopBitCount`  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- `uint8_t enableRx`  
*Enable RX.*
- `uint8_t enableTx`  
*Enable TX.*
- `uint8_t instance`  
*Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.*

#### 40.2.1.0.0.1 Field Documentation

##### 40.2.1.0.0.1.1 uint8\_t hal\_uart\_config\_t::instance

Invalid instance value will cause initialization failure.

## 40.2.2 struct hal\_uart\_transfer\_t

#### Data Fields

- `uint8_t * data`  
*The buffer of data to be transfer.*
- `size_t dataSize`  
*The byte count to be transfer.*

#### 40.2.2.0.0.2 Field Documentation

##### 40.2.2.0.0.2.1 uint8\_t\* hal\_uart\_transfer\_t::data

##### 40.2.2.0.0.2.2 size\_t hal\_uart\_transfer\_t::dataSize

## 40.3 Macro Definition Documentation

### 40.3.1 #define HAL\_UART\_TRANSFER\_MODE (0U)

(0 - disable, 1 - enable)

## Function Documentation

### 40.4 Typedef Documentation

40.4.1 `typedef void(* hal_uart_transfer_callback_t)(hal_uart_handle_t handle, hal_uart_status_t status, void *callbackParam)`

### 40.5 Enumeration Type Documentation

#### 40.5.1 `enum hal_uart_status_t`

Enumerator

*kStatus\_HAL\_UartSuccess* Successfully.  
*kStatus\_HAL\_UartTxBusy* TX busy.  
*kStatus\_HAL\_UartRxBusy* RX busy.  
*kStatus\_HAL\_UartTxIdle* HAL uart transmitter is idle.  
*kStatus\_HAL\_UartRxIdle* HAL uart receiver is idle.  
*kStatus\_HAL\_UartBaudrateNotSupport* Baudrate is not support in current clock source.  
*kStatus\_HAL\_UartProtocolError* Error occurs for Noise, Framing, Parity, etc. For transactional transfer, The up layer needs to abort the transfer and then starts again  
*kStatus\_HAL\_UartError* Error occurs on HAL uart.

#### 40.5.2 `enum hal_uart_parity_mode_t`

Enumerator

*kHAL\_UartParityDisabled* Parity disabled.  
*kHAL\_UartParityEven* Parity even enabled.  
*kHAL\_UartParityOdd* Parity odd enabled.

#### 40.5.3 `enum hal_uart_stop_bit_count_t`

Enumerator

*kHAL\_UartOneStopBit* One stop bit.  
*kHAL\_UartTwoStopBit* Two stop bits.

### 40.6 Function Documentation

40.6.1 `hal_uart_status_t HAL_UartInit ( hal_uart_handle_t handle, hal_uart_config_t * config )`

This function configures the uart module with user-defined settings. The user can configure the configuration structure. The parameter handle is a pointer to point to a memory space of size #HAL\_UART\_HANDLE\_SIZE allocated by the caller. Example below shows how to use this API to configure the uart.

```

* uint8_t g_UartHandleBuffer[HAL_UART_HANDLE_SIZE];
* hal_uart_handle_t g_UartHandle = &g_UartHandleBuffer[0];
* hal_uart_config_t config;
* config.srcClock_Hz = 48000000;
* config.baudRate_Bps = 115200U;
* config.parityMode = kHAL_UartParityDisabled;
* config.stopBitCount = kHAL_UartOneStopBit;
* config.enableRx = 1;
* config.enableTx = 1;
* config.instance = 0;
* HAL_UartInit(g_UartHandle, &config);
*

```

## Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>handle</i> | Pointer to point to a memory space of size #HAL_UART_HANDLE_SIZE allocated by the caller. |
| <i>config</i> | Pointer to user-defined configuration structure.                                          |

## Return values

|                                            |                                                  |
|--------------------------------------------|--------------------------------------------------|
| <i>kStatus_HAL_Uart-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_HAL_Uart-Success</i>            | uart initialization succeed                      |

**40.6.2 hal\_uart\_status\_t HAL\_UartDeinit ( hal\_uart\_handle\_t *handle* )**

This function waits for TX complete, disables TX and RX, and disables the uart clock.

## Parameters

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

## Return values

|                                 |                                |
|---------------------------------|--------------------------------|
| <i>kStatus_HAL_Uart-Success</i> | uart de-initialization succeed |
|---------------------------------|--------------------------------|

**40.6.3 hal\_uart\_status\_t HAL\_UartReceiveBlocking ( hal\_uart\_handle\_t *handle*, 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 reads data from the RX register.

## Function Documentation

### Note

The function [HAL\\_UartReceiveBlocking](#) and the function `#HAL_UartTransferReceiveNonBlocking` cannot be used at the same time. And, the function `#HAL_UartTransferAbortReceive` cannot be used to abort the transmission of this function.

### Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>handle</i> | uart handle pointer.                                    |
| <i>data</i>   | Start address of the buffer to store the received data. |
| <i>length</i> | Size of the buffer.                                     |

### Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_HAL_UartError</i>        | An error occurred while receiving data.       |
| <i>kStatus_HAL_UartParity-Error</i> | A parity error occurred while receiving data. |
| <i>kStatus_HAL_Uart-Success</i>     | Successfully received all data.               |

### 40.6.4 `hal_uart_status_t HAL_UartSendBlocking ( hal_uart_handle_t handle, 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.

### Note

The function [HAL\\_UartSendBlocking](#) and the function `#HAL_UartTransferSendNonBlocking` cannot be used at the same time. And, the function `#HAL_UartTransferAbortSend` cannot be used to abort the transmission of this function.

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>handle</i> | uart handle pointer.                |
| <i>data</i>   | Start address of the data to write. |

|               |                            |
|---------------|----------------------------|
| <i>length</i> | Size of the data to write. |
|---------------|----------------------------|

Return values

|                                 |                             |
|---------------------------------|-----------------------------|
| <i>kStatus_HAL_Uart-Success</i> | Successfully sent all data. |
|---------------------------------|-----------------------------|

#### 40.6.5 **hal\_uart\_status\_t HAL\_UartInstallCallback ( hal\_uart\_handle\_t handle, hal\_uart\_transfer\_callback\_t callback, void \* callbackParam )**

This function is used to install the callback and callback parameter for uart module. When non-blocking sending or receiving finished, the adapter 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>handle</i>        | uart handle pointer.                    |
| <i>callback</i>      | The callback function.                  |
| <i>callbackParam</i> | The parameter of the callback function. |

Return values

|                                 |                                    |
|---------------------------------|------------------------------------|
| <i>kStatus_HAL_Uart-Success</i> | Successfully install the callback. |
|---------------------------------|------------------------------------|

#### 40.6.6 **hal\_uart\_status\_t HAL\_UartReceiveNonBlocking ( hal\_uart\_handle\_t handle, uint8\_t \* data, size\_t length )**

This function receives data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be received. The receive request is saved by the uart adapter. When the new data arrives, the receive request is serviced first. When all data is received, the uart adapter notifies the upper layer through a callback function and passes the status parameter kStatus\_UART\_RxIdle.

Note

The function [HAL\\_UartReceiveBlocking](#) and the function [HAL\\_UartReceiveNonBlocking](#) cannot be used at the same time.

## Function Documentation

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>handle</i> | uart handle pointer.                |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

Return values

|                                 |                                                      |
|---------------------------------|------------------------------------------------------|
| <i>kStatus_HAL_Uart-Success</i> | Successfully queue the transfer into transmit queue. |
| <i>kStatus_HAL_UartRx-Busy</i>  | Previous receive request is not finished.            |
| <i>kStatus_HAL_UartError</i>    | An error occurred.                                   |

### 40.6.7 **hal\_uart\_status\_t HAL\_UartSendNonBlocking ( hal\_uart\_handle\_t *handle*, const uint8\_t \* *data*, size\_t *length* )**

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 ISR, the uart driver calls the callback function and passes the kStatus\_UART\_TxIdle as status parameter.

Note

The function [HAL\\_UartSendBlocking](#) and the function [HAL\\_UartSendNonBlocking](#) cannot be used at the same time.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>handle</i> | uart handle pointer.                |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

Return values

|                                 |                                           |
|---------------------------------|-------------------------------------------|
| <i>kStatus_HAL_Uart-Success</i> | Successfully start the data transmission. |
|---------------------------------|-------------------------------------------|

|                                |                                                                                    |
|--------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_HAL_UartTx-Busy</i> | Previous transmission still not finished; data not all written to TX register yet. |
| <i>kStatus_HAL_UartError</i>   | An error occurred.                                                                 |

#### 40.6.8 **hal\_uart\_status\_t HAL\_UartGetReceiveCount ( hal\_uart\_handle\_t handle, uint32\_t \* count )**

This function gets the number of bytes that have been received.

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | uart handle pointer. |
| <i>count</i>  | Receive bytes count. |

Return values

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <i>kStatus_HAL_UartError</i> | An error occurred.                                    |
| <i>kStatus_Success</i>       | Get successfully through the parameter <i>count</i> . |

#### 40.6.9 **hal\_uart\_status\_t HAL\_UartGetSendCount ( hal\_uart\_handle\_t handle, uint32\_t \* count )**

This function gets the number of bytes written to the uart TX register by using the interrupt method.

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | uart handle pointer. |
| <i>count</i>  | Send bytes count.    |

Return values

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <i>kStatus_HAL_UartError</i> | An error occurred.                                    |
| <i>kStatus_Success</i>       | Get successfully through the parameter <i>count</i> . |

#### 40.6.10 **hal\_uart\_status\_t HAL\_UartAbortReceive ( hal\_uart\_handle\_t handle )**

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes are not received yet.

## Function Documentation

### Note

The function [HAL\\_UartAbortReceive](#) cannot be used to abort the transmission of the function [HAL\\_UartReceiveBlocking](#).

### Parameters

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

### Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Get successfully abort the receiving. |
|------------------------|---------------------------------------|

## 40.6.11 **hal\_uart\_status\_t HAL\_UartAbortSend ( hal\_uart\_handle\_t handle )**

This function aborts the interrupt-driven data sending. The user can get the remainBytes to find out how many bytes are not sent out.

### Note

The function [HAL\\_UartAbortSend](#) cannot be used to abort the transmission of the function [HAL\\_UartSendBlocking](#).

### Parameters

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

### Return values

|                        |                                     |
|------------------------|-------------------------------------|
| <i>kStatus_Success</i> | Get successfully abort the sending. |
|------------------------|-------------------------------------|

## 40.6.12 **void HAL\_UartIsrFunction ( hal\_uart\_handle\_t handle )**

This function handles the uart transmit and receive IRQ request.

### Parameters

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

## Function Documentation

**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.

© 2019 NXP B.V.

**arm**