
MCUXpresso SDK API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKAPIRM
Rev. 0
Jul 2020



Contents

Chapter [Introduction](#)

Chapter [Trademarks](#)

Chapter [Architectural Overview](#)

Chapter [Driver errors status](#)

Chapter [Deprecated List](#)

Chapter [Clock Driver](#)

6.1	Overview	13
6.2	Macro Definition Documentation	15
6.2.1	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	15
6.2.2	FSL_CLOCK_DRIVER_VERSION	15
6.2.3	MU_CLOCKS	15
6.2.4	GPIO_CLOCKS	16
6.2.5	RGPIO_CLOCKS	16
6.2.6	FTM_CLOCKS	16
6.2.7	GPT_CLOCKS	17
6.2.8	FLEXCAN_CLOCKS	17
6.2.9	FLEXSPI_CLOCKS	17
6.2.10	LPUART_CLOCKS	17
6.2.11	LPADC_CLOCKS	17
6.2.12	INTMUX_CLOCKS	18
6.2.13	SAI_CLOCKS	18
6.2.14	SEMA42_CLOCKS	18
6.2.15	TPM_CLOCKS	18
6.2.16	LPIT_CLOCKS	18
6.2.17	LPI2C_CLOCKS	19
6.2.18	LPSPI_CLOCKS	19
6.2.19	EDMA_CLOCKS	19
6.2.20	ESAI_CLOCKS	19
6.2.21	ISI_CLOCKS	20
6.2.22	MIPI_CSI2RX_CLOCKS	20

Contents

Section Number	Title	Page Number
6.2.23	MIPI_DSI_HOST_CLOCKS	20
6.2.24	ENET_CLOCKS	20
6.2.25	EMVSIM_CLOCKS	20
6.2.26	DPU_CLOCKS	21
6.2.27	LDB_CLOCKS	21
6.2.28	LPCG_TUPLE	21
6.2.29	LPCG_TUPLE_REG_BASE	21
6.2.30	LPCG_TUPLE_RSRC	21
6.2.31	NV	21
6.3	Enumeration Type Documentation	21
6.3.1	clock_ip_src_t	21
6.3.2	clock_name_t	22
6.3.3	clock_ip_name_t	22
6.4	Function Documentation	22
6.4.1	CLOCK_Init	22
6.4.2	CLOCK_EnableClockExt	22
6.4.3	CLOCK_EnableClock	22
6.4.4	CLOCK_DisableClock	23
6.4.5	CLOCK_SetIpFreq	23
6.4.6	CLOCK_GetIpFreq	23
6.4.7	CLOCK_GetFreq	24
6.4.8	CLOCK_GetCoreSysClkFreq	24
6.4.9	CLOCK_ConfigLPCG	24
6.4.10	CLOCK_SetLpcgGate	24
Chapter	SCU EVENT Driver	
7.1	Overview	27
7.2	Data Structure Documentation	29
7.2.1	struct sc_event_list_t	29
7.2.2	struct _sc_event_handler	29
7.3	Macro Definition Documentation	30
7.3.1	FSL_SC_EVENT_DRIVER_VERSION	30
7.3.2	SC_EVENT_HANDLER_MEM_POOL_SIZE	30
7.3.3	SC_EVENT_USE_SYSCTR	30
7.4	Typedef Documentation	30
7.4.1	sc_event_callback_t	30
7.4.2	sc_event_sema4_wait_t	30
7.4.3	sc_event_sema4_post_t	30
7.5	Enumeration Type Documentation	30

Contents

Section Number	Title	Page Number
7.5.1	sc_event_t	30
7.6	Function Documentation	31
7.6.1	SCEvent_Init	31
7.6.2	SCEvent_Deinit	32
7.6.3	SCEvent_RegisterEventHandler	32
7.6.4	SCEvent_UnregisterEventHandler	32
7.6.5	SCEvent_Config	33
7.6.6	SCEvent_Process	34
7.6.7	SCEvent_WaitEvent	34
Chapter	ASMC: Auxiliary System Mode Controller Driver	
8.1	Overview	37
8.2	Typical use case	37
8.2.1	Enter wait or stop modes	37
8.3	Macro Definition Documentation	39
8.3.1	FSL_ASMC_DRIVER_VERSION	39
8.4	Enumeration Type Documentation	39
8.4.1	_asmc_system_reset_status_flags	39
8.4.2	asmc_power_mode_protection_t	40
8.4.3	asmc_power_state_t	40
8.4.4	asmc_run_mode_t	40
8.4.5	asmc_stop_mode_t	40
8.4.6	asmc_partial_stop_option_t	41
8.5	Function Documentation	41
8.5.1	ASMC_GetSystemResetStatusFlags	41
8.5.2	ASMC_SetPowerModeProtection	41
8.5.3	ASMC_GetPowerModeState	42
8.5.4	ASMC_PreEnterStopModes	42
8.5.5	ASMC_PostExitStopModes	42
8.5.6	ASMC_PreEnterWaitModes	42
8.5.7	ASMC_PostExitWaitModes	42
8.5.8	ASMC_SetPowerModeRun	43
8.5.9	ASMC_SetPowerModeWait	44
8.5.10	ASMC_SetPowerModeStop	44
8.5.11	ASMC_SetPowerModeVlpr	44
8.5.12	ASMC_SetPowerModeVlps	45
8.5.13	ASMC_SetPowerModeVlps	45
8.5.14	ASMC_SetPowerModeLls	45
8.5.15	ASMC_SetPowerModeVlls	45

Contents

Section Number	Title	Page Number
Chapter	CACHE: LMEM CACHE Memory Controller	
9.1	Function groups	47
9.1.1	L1 CACHE Operation	47
Chapter	Common Driver	
10.1	Overview	49
10.2	Macro Definition Documentation	53
10.2.1	MAKE_STATUS	53
10.2.2	MAKE_VERSION	53
10.2.3	FSL_COMMON_DRIVER_VERSION	53
10.2.4	DEBUG_CONSOLE_DEVICE_TYPE_NONE	53
10.2.5	DEBUG_CONSOLE_DEVICE_TYPE_UART	53
10.2.6	DEBUG_CONSOLE_DEVICE_TYPE_LPUART	53
10.2.7	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	53
10.2.8	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	53
10.2.9	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	53
10.2.10	DEBUG_CONSOLE_DEVICE_TYPE_IUART	53
10.2.11	DEBUG_CONSOLE_DEVICE_TYPE_VUSART	53
10.2.12	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	53
10.2.13	DEBUG_CONSOLE_DEVICE_TYPE_SWO	53
10.2.14	ARRAY_SIZE	53
10.3	Typedef Documentation	53
10.3.1	status_t	53
10.4	Enumeration Type Documentation	53
10.4.1	_status_groups	53
10.4.2	anonymous enum	56
10.5	Function Documentation	56
10.5.1	EnableIRQ	56
10.5.2	DisableIRQ	57
10.5.3	DisableGlobalIRQ	57
10.5.4	EnableGlobalIRQ	57
10.5.5	SDK_Malloc	58
10.5.6	SDK_Free	58
10.5.7	SDK_DelayAtLeastUs	58
Chapter	EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver	
11.1	Overview	59
11.2	Typical use case	59

Contents

Section Number	Title	Page Number
11.2.1	EDMA Operation	59
11.3	Data Structure Documentation	64
11.3.1	struct edma_config_t	64
11.3.2	struct edma_transfer_config_t	65
11.3.3	struct edma_channel_Preemption_config_t	66
11.3.4	struct edma_minor_offset_config_t	66
11.3.5	struct edma_tcd_t	67
11.3.6	struct edma_handle_t	68
11.4	Macro Definition Documentation	69
11.4.1	FSL_EDMA_DRIVER_VERSION	69
11.5	Typedef Documentation	69
11.5.1	edma_callback	69
11.6	Enumeration Type Documentation	69
11.6.1	edma_transfer_size_t	69
11.6.2	edma_modulo_t	70
11.6.3	edma_bandwidth_t	70
11.6.4	edma_channel_link_type_t	71
11.6.5	anonymous enum	71
11.6.6	anonymous enum	71
11.6.7	anonymous enum	71
11.6.8	edma_interrupt_enable_t	72
11.6.9	edma_transfer_type_t	72
11.6.10	anonymous enum	72
11.7	Function Documentation	72
11.7.1	EDMA_Init	72
11.7.2	EDMA_Deinit	72
11.7.3	EDMA_InstallTCD	73
11.7.4	EDMA_GetDefaultConfig	73
11.7.5	EDMA_EnableAllChannelLink	73
11.7.6	EDMA_ResetChannel	74
11.7.7	EDMA_SetTransferConfig	74
11.7.8	EDMA_SetMinorOffsetConfig	75
11.7.9	EDMA_SetChannelArbitrationGroup	75
11.7.10	EDMA_SetChannelPreemptionConfig	76
11.7.11	EDMA_GetChannelSystemBusInformation	77
11.7.12	EDMA_SetChannelLink	77
11.7.13	EDMA_SetBandWidth	78
11.7.14	EDMA_SetModulo	78
11.7.15	EDMA_EnableAsyncRequest	78
11.7.16	EDMA_EnableAutoStopRequest	79

Contents

Section Number	Title	Page Number
11.7.17	EDMA_EnableChannelInterrupts	79
11.7.18	EDMA_DisableChannelInterrupts	79
11.7.19	EDMA_TcdReset	80
11.7.20	EDMA_TcdSetTransferConfig	81
11.7.21	EDMA_TcdSetMinorOffsetConfig	81
11.7.22	EDMA_TcdSetChannelLink	82
11.7.23	EDMA_TcdSetBandWidth	82
11.7.24	EDMA_TcdSetModulo	83
11.7.25	EDMA_TcdEnableAutoStopRequest	83
11.7.26	EDMA_TcdEnableInterrupts	83
11.7.27	EDMA_TcdDisableInterrupts	83
11.7.28	EDMA_EnableChannelRequest	84
11.7.29	EDMA_DisableChannelRequest	84
11.7.30	EDMA_TriggerChannelStart	84
11.7.31	EDMA_GetRemainingMajorLoopCount	85
11.7.32	EDMA_GetErrorStatusFlags	85
11.7.33	EDMA_GetChannelStatusFlags	86
11.7.34	EDMA_ClearChannelStatusFlags	86
11.7.35	EDMA_CreateHandle	86
11.7.36	EDMA_InstallTCDDMemory	87
11.7.37	EDMA_SetCallback	87
11.7.38	EDMA_PrepareTransferConfig	87
11.7.39	EDMA_PrepareTransfer	88
11.7.40	EDMA_SubmitTransfer	89
11.7.41	EDMA_StartTransfer	89
11.7.42	EDMA_StopTransfer	89
11.7.43	EDMA_AbortTransfer	90
11.7.44	EDMA_GetUnusedTCDNumber	90
11.7.45	EDMA_GetNextTCDAddress	90
11.7.46	EDMA_HandleIRQ	91

Chapter DPR: Display Prefetch Resolve

12.1	Overview	93
12.2	Data Structure Documentation	94
12.2.1	struct dpr_buffer_config_t	94
12.3	Macro Definition Documentation	94
12.3.1	FSL_DPR_DRIVER_VERSION	94
12.4	Enumeration Type Documentation	94
12.4.1	dpr_data_type_t	94
12.5	Function Documentation	94

Contents

Section Number	Title	Page Number
12.5.1	DPR_Init	94
12.5.2	DPR_Deinit	95
12.5.3	DPR_SetBufferConfig	95
12.5.4	DPR_BufferGetDefaultConfig	95
12.5.5	DPR_Start	95
12.5.6	DPR_StartRepeat	96
12.5.7	DPR_Stop	96
12.5.8	DPR_SetBufferAddr	96
Chapter	Display Processing Unit (DPU)	
13.1	Overview	97
13.2	Program model	97
13.3	Path configuration	101
13.4	Data Structure Documentation	119
13.4.1	struct dpu_fetch_unit_config_t	119
13.4.2	struct dpu_coordinates_config_t	119
13.4.3	struct dpu_warp_config_t	120
13.4.4	struct dpu_src_buffer_config_t	121
13.4.5	struct dpu_clip_window_config_t	122
13.4.6	struct dpu_dst_buffer_config_t	123
13.4.7	struct dpu_layer_blend_config_t	124
13.4.8	struct dpu_blit_blend_config_t	125
13.4.9	struct dpu_rop_config_t	126
13.4.10	struct dpu_const_frame_config_t	126
13.4.11	struct dpu_display_timing_config_t	127
13.4.12	struct dpu_display_config_t	128
13.4.13	struct dpu_scaler_config_t	129
13.4.14	struct dpu_signature_config_t	130
13.4.15	struct dpu_signature_window_config_t	131
13.5	Macro Definition Documentation	131
13.5.1	FSL_DPU_DRIVER_VERSION	131
13.5.2	DPU_PALETTE_ENTRY_NUM	131
13.5.3	DPU_FETCH_UNIT_BURST_LENGTH	131
13.5.4	DPU_FETCH_UNIT_BURST_SIZE	131
13.5.5	DPU_MAKE_SRC_REG1	132
13.5.6	DPU_MAKE_SRC_REG2	132
13.5.7	DPU_MAKE_SRC_REG3	132
13.5.8	DPU_MAKE_CONST_COLOR	132
13.5.9	DPU_FRAME_GEN_INT_DISABLE	132
13.5.10	DPU_FRAME_GEN_INT_PER_LINE	132

Contents

Section Number	Title	Page Number
13.5.11	DPU_FRAME_GEN_INT_PER_FRAME	132
13.6	Enumeration Type Documentation	132
13.6.1	dpu_unit_t	132
13.6.2	_dpu_interrupt	132
13.6.3	_dpu_unit_source	134
13.6.4	dpu_pixel_format_t	134
13.6.5	dpu_warp_coordinate_mode_t	135
13.6.6	dpu_clip_color_mode_t	135
13.6.7	dpu_alpha_mask_mode_t	135
13.6.8	dpu_blend_mode_t	135
13.6.9	dpu_blit_blend_func_t	136
13.6.10	dpu_blit_blend_mode_t	136
13.6.11	dpu_blit_blend_neutral_border_mode_t	137
13.6.12	_dpu_rop_flags	137
13.6.13	_dpu_display_timing_flags	137
13.6.14	dpu_display_mode_t	138
13.6.15	_dpu_signature_window_flags	138
13.6.16	_dpu_signature_status	138
13.7	Function Documentation	138
13.7.1	DPU_Init	138
13.7.2	DPU_Deinit	139
13.7.3	DPU_PreparePathConfig	139
13.7.4	DPU_EnableInterrupts	139
13.7.5	DPU_DisableInterrupts	140
13.7.6	DPU_GetInterruptsPendingFlags	140
13.7.7	DPU_ClearInterruptsPendingFlags	141
13.7.8	DPU_SetInterruptsPendingFlags	141
13.7.9	DPU_MaskUserInterrupts	142
13.7.10	DPU_EnableUserInterrupts	142
13.7.11	DPU_DisableUserInterrupts	142
13.7.12	DPU_GetUserInterruptsPendingFlags	143
13.7.13	DPU_ClearUserInterruptsPendingFlags	143
13.7.14	DPU_SetUserInterruptsPendingFlags	143
13.7.15	DPU_EnableShadowLoad	144
13.7.16	DPU_InitPipeline	144
13.7.17	DPU_DeinitPipeline	144
13.7.18	DPU_TriggerPipelineShadowLoad	145
13.7.19	DPU_TriggerPipelineCompleteInterrupt	145
13.7.20	DPU_SetUnitSrc	145
13.7.21	DPU_FetchUnitGetDefaultConfig	146
13.7.22	DPU_InitFetchUnit	146
13.7.23	DPU_SetColorPaletteIndexWidth	147
13.7.24	DPU_UpdateColorPalette	147

Contents

Section Number	Title	Page Number
13.7.25	DPU_EnableColorPalette	148
13.7.26	DPU_CorrordinatesGetDefaultConfig	148
13.7.27	DPU_InitWarpCoordinates	148
13.7.28	DPU_FetcUnitGetDefaultWarpConfig	149
13.7.29	DPU_InitFetchUnitWarp	149
13.7.30	DPU_SrcBufferGetDefaultConfig	150
13.7.31	DPU_SetFetchUnitSrcBufferConfig	150
13.7.32	DPU_SetFetchUnitSrcBufferAddr	151
13.7.33	DPU_SetFetchUnitFrameSize	151
13.7.34	DPU_SetFetchUnitOffset	151
13.7.35	DPU_EnableFetchUnitSrcBuffer	152
13.7.36	DPU_ClipWindowGetDefaultConfig	152
13.7.37	DPU_SetFetchUnitClipWindowConfig	152
13.7.38	DPU_EnableFetchUnitClipWindow	153
13.7.39	DPU_SetFetchUnitClipColor	153
13.7.40	DPU_InitExtDst	153
13.7.41	DPU_InitStore	154
13.7.42	DPU_SetStoreDstBufferConfig	155
13.7.43	DPU_DstBufferGetDefaultConfig	156
13.7.44	DPU_SetStoreDstBufferAddr	156
13.7.45	DPU_SetStoreOffset	156
13.7.46	DPU_StartStore	157
13.7.47	DPU_InitRop	158
13.7.48	DPU_RopGetDefaultConfig	158
13.7.49	DPU_SetRopConfig	159
13.7.50	DPU_EnableRop	159
13.7.51	DPU_InitBlitBlend	159
13.7.52	DPU_BlitBlendGetDefaultConfig	160
13.7.53	DPU_SetBlitBlendConfig	160
13.7.54	DPU_EnableBlitBlend	161
13.7.55	DPU_LayerBlendGetDefaultConfig	162
13.7.56	DPU_InitLayerBlend	162
13.7.57	DPU_SetLayerBlendConfig	163
13.7.58	DPU_EnableLayerBlend	163
13.7.59	DPU_InitConstFrame	164
13.7.60	DPU_ConstFrameGetDefaultConfig	164
13.7.61	DPU_SetConstFrameConfig	164
13.7.62	DPU_InitScaler	164
13.7.63	DPU_ScalerGetDefaultConfig	165
13.7.64	DPU_SetScalerConfig	165
13.7.65	DPU_DisplayTimingGetDefaultConfig	166
13.7.66	DPU_InitDisplayTiming	166
13.7.67	DPU_DisplayGetDefaultConfig	166
13.7.68	DPU_SetDisplayConfig	167
13.7.69	DPU_StartDisplay	167

Contents

Section Number	Title	Page Number
13.7.70	DPU_StopDisplay	167
13.7.71	DPU_SetFrameGenInterruptConfig	167
13.7.72	DPU_TriggerDisplayShadowLoad	168
13.7.73	DPU_SignatureGetDefaultConfig	168
13.7.74	DPU_InitSignature	168
13.7.75	DPU_SignatureWindowGetDefaultConfig	169
13.7.76	DPU_SetSignatureWindowConfig	169
13.7.77	DPU_EnableSignatureWindowCompute	169
13.7.78	DPU_EnableSignatureWindowCheck	170
13.7.79	DPU_GetSignatureWindowCrc	170
13.7.80	DPU_SetSignatureWindowRefCrc	170
13.7.81	DPU_GetSignatureStatus	171
13.7.82	DPU_TriggerSignatureShadowLoad	172
Chapter	DPU IRQSTEER: Interrupt Request Steering Driver	
14.1	Overview	173
14.2	Function Documentation	173
14.2.1	DPU_IRQSTEER_EnableInterrupt	173
14.2.2	DPU_IRQSTEER_DisableInterrupt	173
14.2.3	DPU_IRQSTEER_IsInterruptSet	174
Chapter	ENET: Ethernet MAC Driver	
15.1	Overview	175
15.2	Operations of Ethernet MAC Driver	175
15.2.1	MII interface Operation	175
15.2.2	MAC address filter	175
15.2.3	Other Baisc control Operations	175
15.2.4	Transactional Operation	175
15.2.5	PTP IEEE 1588 Feature Operation	176
15.3	Typical use case	176
15.3.1	ENET Initialization, receive, and transmit operations	176
15.4	Data Structure Documentation	185
15.4.1	struct enet_rx_bd_struct_t	185
15.4.2	struct enet_tx_bd_struct_t	185
15.4.3	struct enet_data_error_stats_t	185
15.4.4	struct enet_frame_info_t	186
15.4.5	struct enet_tx_dirty_ring_t	186
15.4.6	struct enet_buffer_config_t	187
15.4.7	struct enet_intcoalesce_config_t	188

Contents

Section Number	Title	Page Number
15.4.8	struct enet_avb_config_t	189
15.4.9	struct enet_config_t	189
15.4.10	struct enet_tx_bd_ring_t	191
15.4.11	struct enet_rx_bd_ring_t	192
15.4.12	struct _enet_handle	192
15.5	Macro Definition Documentation	193
15.5.1	FSL_ENET_DRIVER_VERSION	193
15.5.2	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	195
15.5.3	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	195
15.5.4	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	195
15.5.5	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	195
15.5.6	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	195
15.5.7	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	195
15.5.8	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	195
15.5.9	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	195
15.5.10	ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK	195
15.5.11	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	195
15.5.12	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	195
15.5.13	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	195
15.5.14	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	195
15.5.15	ENET_BUFFDESCRIPTOR_TX_READY_MASK	195
15.5.16	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK	195
15.5.17	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	195
15.5.18	ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK	195
15.5.19	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	195
15.5.20	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	195
15.5.21	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	195
15.5.22	ENET_FRAME_MAX_FRAMELEN	196
15.5.23	ENET_FIFO_MIN_RX_FULL	196
15.5.24	ENET_RX_MIN_BUFFERSIZE	196
15.5.25	ENET_PHY_MAXADDRESS	196
15.5.26	ENET_TX_INTERRUPT	196
15.5.27	ENET_RX_INTERRUPT	196
15.5.28	ENET_TS_INTERRUPT	196
15.5.29	ENET_ERR_INTERRUPT	196
15.5.30	ENET_TX_LAST_BD_FLAG	197
15.5.31	ENET_TX_TIMESTAMP_FLAG	197
15.6	Typedef Documentation	197
15.6.1	enet_callback_t	197
15.6.2	enet_isr_ring_t	197
15.7	Enumeration Type Documentation	197
15.7.1	anonymous enum	197

Contents

Section Number	Title	Page Number
15.7.2	enet_mii_mode_t	197
15.7.3	enet_mii_speed_t	198
15.7.4	enet_mii_duplex_t	198
15.7.5	enet_mii_write_t	198
15.7.6	enet_mii_read_t	198
15.7.7	enet_mii_extend_opcode	198
15.7.8	enet_special_control_flag_t	199
15.7.9	enet_interrupt_enable_t	199
15.7.10	enet_event_t	200
15.7.11	enet_idle_slope_t	200
15.7.12	enet_tx_accelerator_t	201
15.7.13	enet_rx_accelerator_t	201
15.8	Function Documentation	201
15.8.1	ENET_GetInstance	201
15.8.2	ENET_GetDefaultConfig	201
15.8.3	ENET_Up	202
15.8.4	ENET_Init	203
15.8.5	ENET_Down	204
15.8.6	ENET_Deinit	204
15.8.7	ENET_Reset	204
15.8.8	ENET_SetMII	204
15.8.9	ENET_SetSMI	205
15.8.10	ENET_GetSMI	205
15.8.11	ENET_ReadSMIData	205
15.8.12	ENET_StartSMIRead	206
15.8.13	ENET_StartSMIWrite	206
15.8.14	ENET_StartExtC45SMIRead	206
15.8.15	ENET_StartExtC45SMIWrite	207
15.8.16	ENET_StartExtC45SMIWriteReg	207
15.8.17	ENET_StartExtC45SMIWriteData	208
15.8.18	ENET_StartExtC45SMIReadData	208
15.8.19	ENET_SetRGMIIClockDelay	209
15.8.20	ENET_SetMacAddr	209
15.8.21	ENET_GetMacAddr	209
15.8.22	ENET_AddMulticastGroup	209
15.8.23	ENET_LeaveMulticastGroup	210
15.8.24	ENET_ActiveRead	210
15.8.25	ENET_EnableSleepMode	210
15.8.26	ENET_GetAccelFunction	211
15.8.27	ENET_EnableInterrupts	212
15.8.28	ENET_DisableInterrupts	212
15.8.29	ENET_GetInterruptStatus	213
15.8.30	ENET_ClearInterruptStatus	213
15.8.31	ENET_SetRxISRHandler	214

Contents

Section Number	Title	Page Number
15.8.32	ENET_SetTxISRHandler	214
15.8.33	ENET_SetErrISRHandler	214
15.8.34	ENET_SetCallback	214
15.8.35	ENET_GetRxErrBeforeReadFrame	215
15.8.36	ENET_GetRxFrameSize	215
15.8.37	ENET_ReadFrame	216
15.8.38	ENET_SendFrame	217
15.8.39	ENET_SetTxReclaim	218
15.8.40	ENET_GetRxBuffer	218
15.8.41	ENET_ReleaseRxBuffer	219
15.8.42	ENET_SendFrameZeroCopy	220
15.8.43	ENET_SetTxBuffer	221
15.8.44	ENET_TransmitIRQHandler	221
15.8.45	ENET_ReceiveIRQHandler	222
15.8.46	ENET_CommonFrame1IRQHandler	223
15.8.47	ENET_CommonFrame2IRQHandler	223
15.8.48	ENET_ErrorIRQHandler	223
15.8.49	ENET_CommonFrame0IRQHandler	223

Chapter ESAI: Enhanced Serial Audio Interface

16.1	Overview	225
16.2	Typical use case	225
16.2.1	ESAI Send/Receive using an interrupt method	225
16.2.2	ESAI Send/receive using a DMA method	225
16.3	Data Structure Documentation	232
16.3.1	struct esai_customer_protocol_t	232
16.3.2	struct esai_config_t	232
16.3.3	struct esai_format_t	233
16.3.4	struct esai_transfer_t	233
16.3.5	struct _esai_handle	234
16.4	Macro Definition Documentation	234
16.4.1	ESAI_XFER_QUEUE_SIZE	234
16.5	Enumeration Type Documentation	234
16.5.1	anonymous enum	234
16.5.2	esai_mode_t	235
16.5.3	esai_protocol_t	235
16.5.4	esai_master_slave_t	235
16.5.5	esai_sync_mode_t	235
16.5.6	esai_clock_polarity_t	236
16.5.7	esai_shift_direction_t	236

Contents

Section Number	Title	Page Number
16.5.8	esai_clock_direction_t	236
16.5.9	anonymous enum	236
16.5.10	anonymous enum	236
16.5.11	anonymous enum	237
16.5.12	esai_sample_rate_t	237
16.5.13	esai_word_width_t	237
16.5.14	esai_slot_format_t	238
16.6	Function Documentation	238
16.6.1	ESAI_Init	238
16.6.2	ESAI_GetDefaultConfig	239
16.6.3	ESAI_Deinit	239
16.6.4	ESAI_Enable	239
16.6.5	ESAI_Reset	239
16.6.6	ESAI_TxReset	240
16.6.7	ESAI_RxReset	240
16.6.8	ESAI_TxResetFIFO	240
16.6.9	ESAI_RxResetFIFO	240
16.6.10	ESAI_TxEnable	241
16.6.11	ESAI_RxEnable	241
16.6.12	ESAI_TxEnableFIFO	241
16.6.13	ESAI_RxEnableFIFO	241
16.6.14	ESAI_TxSetSlotMask	242
16.6.15	EASI_RxSetSlotMask	242
16.6.16	ESAI_AnalysisSlot	242
16.6.17	ESAI_GetInstance	242
16.6.18	ESAI_GetStatusFlag	243
16.6.19	ESAI_GetSAIStatusFlag	243
16.6.20	ESAI_GetTxFIFOStatus	243
16.6.21	ESAI_GetRxFIFOStatus	243
16.6.22	ESAI_TxEnableInterrupts	244
16.6.23	ESAI_RxEnableInterrupts	244
16.6.24	ESAI_TxDisableInterrupts	244
16.6.25	ESAI_RxDisableInterrupts	245
16.6.26	ESAI_TxGetDataRegisterAddress	245
16.6.27	ESAI_RxGetDataRegisterAddress	245
16.6.28	ESAI_TxSetFormat	246
16.6.29	ESAI_RxSetFormat	247
16.6.30	ESAI_WriteBlocking	247
16.6.31	ESAI_WriteData	248
16.6.32	ESAI_ReadBlocking	248
16.6.33	ESAI_ReadData	248
16.6.34	ESAI_TransferTxCreateHandle	249
16.6.35	ESAI_TransferRxCreateHandle	250
16.6.36	ESAI_TransferTxSetFormat	250

Contents

Section Number	Title	Page Number
16.6.37	ESAI_TransferRxSetFormat	251
16.6.38	ESAI_TransferSendNonBlocking	251
16.6.39	ESAI_TransferReceiveNonBlocking	252
16.6.40	ESAI_TransferGetSendCount	252
16.6.41	ESAI_TransferGetReceiveCount	253
16.6.42	ESAI_TransferAbortSend	253
16.6.43	ESAI_TransferAbortReceive	253
16.6.44	ESAI_TransferTxHandleIRQ	254
16.6.45	ESAI_TransferRxHandleIRQ	254
16.7	ESAI eDMA Driver	255
16.7.1	Overview	255
16.7.2	Data Structure Documentation	256
16.7.3	Function Documentation	257
Chapter	FlexCAN: Flex Controller Area Network Driver	
17.1	Overview	263
17.2	FlexCAN Driver	264
17.2.1	Overview	264
17.2.2	Typical use case	264
17.2.3	Data Structure Documentation	274
17.2.4	Macro Definition Documentation	279
17.2.5	Typedef Documentation	284
17.2.6	Enumeration Type Documentation	284
17.2.7	Function Documentation	288
17.3	FlexCAN eDMA Driver	313
17.3.1	Overview	313
17.3.2	Data Structure Documentation	314
17.3.3	Macro Definition Documentation	314
17.3.4	Typedef Documentation	314
17.3.5	Function Documentation	314
Chapter	FLEXSPI: Flexible Serial Peripheral Interface Driver	
18.1	Overview	319
18.2	Data Structure Documentation	324
18.2.1	struct flexspi_config_t	324
18.2.2	struct flexspi_device_config_t	327
18.2.3	struct flexspi_transfer_t	328
18.2.4	struct _flexspi_handle	329

Contents

Section Number	Title	Page Number
18.3	Macro Definition Documentation	330
18.3.1	FSL_FLEXSPI_DRIVER_VERSION	330
18.3.2	FLEXSPI_LUT_SEQ	330
18.4	Typedef Documentation	330
18.4.1	flexspi_transfer_callback_t	330
18.5	Enumeration Type Documentation	330
18.5.1	anonymous enum	330
18.5.2	anonymous enum	330
18.5.3	flexspi_pad_t	331
18.5.4	flexspi_flags_t	331
18.5.5	flexspi_read_sample_clock_t	332
18.5.6	flexspi_cs_interval_cycle_unit_t	332
18.5.7	flexspi_ahb_write_wait_unit_t	333
18.5.8	flexspi_ip_error_code_t	333
18.5.9	flexspi_ahb_error_code_t	333
18.5.10	flexspi_port_t	334
18.5.11	flexspi_arb_command_source_t	334
18.5.12	flexspi_command_type_t	334
18.6	Function Documentation	334
18.6.1	FLEXSPI_Init	334
18.6.2	FLEXSPI_GetDefaultConfig	334
18.6.3	FLEXSPI_Deinit	334
18.6.4	FLEXSPI_SetFlashConfig	335
18.6.5	FLEXSPI_SoftwareReset	335
18.6.6	FLEXSPI_Enable	335
18.6.7	FLEXSPI_EnableInterrupts	335
18.6.8	FLEXSPI_DisableInterrupts	336
18.6.9	FLEXSPI_EnableTxDMA	336
18.6.10	FLEXSPI_EnableRxDMA	336
18.6.11	FLEXSPI_GetTxFifoAddress	336
18.6.12	FLEXSPI_GetRxFifoAddress	337
18.6.13	FLEXSPI_ResetFifos	337
18.6.14	FLEXSPI_GetFifoCounts	337
18.6.15	FLEXSPI_GetInterruptStatusFlags	338
18.6.16	FLEXSPI_ClearInterruptStatusFlags	338
18.6.17	FLEXSPI_GetDataLearningPhase	338
18.6.18	FLEXSPI_GetArbitratorCommandSource	338
18.6.19	FLEXSPI_GetIPCommandErrorCode	339
18.6.20	FLEXSPI_GetAHBCommandErrorCode	339
18.6.21	FLEXSPI_GetBusIdleStatus	339
18.6.22	FLEXSPI_UpdateRxSampleClock	340
18.6.23	FLEXSPI_EnableIPParallelMode	340

Contents

Section Number	Title	Page Number
18.6.24	FLEXSPI_EnableAHBParallelMode	340
18.6.25	FLEXSPI_UpdateLUT	340
18.6.26	FLEXSPI_WriteData	341
18.6.27	FLEXSPI_ReadData	341
18.6.28	FLEXSPI_WriteBlocking	341
18.6.29	FLEXSPI_ReadBlocking	342
18.6.30	FLEXSPI_TransferBlocking	343
18.6.31	FLEXSPI_TransferCreateHandle	344
18.6.32	FLEXSPI_TransferNonBlocking	344
18.6.33	FLEXSPI_TransferGetCount	345
18.6.34	FLEXSPI_TransferAbort	345
18.6.35	FLEXSPI_TransferHandleIRQ	346
Chapter	FTM: FlexTimer Driver	
19.1	Overview	347
19.2	Function groups	347
19.2.1	Initialization and deinitialization	347
19.2.2	PWM Operations	347
19.2.3	Input capture operations	347
19.2.4	Output compare operations	348
19.2.5	Quad decode	348
19.2.6	Fault operation	348
19.3	Register Update	348
19.4	Typical use case	349
19.4.1	PWM output	349
19.5	Data Structure Documentation	355
19.5.1	struct ftm_chnl_pwm_signal_param_t	355
19.5.2	struct ftm_chnl_pwm_config_param_t	356
19.5.3	struct ftm_dual_edge_capture_param_t	356
19.5.4	struct ftm_phase_params_t	357
19.5.5	struct ftm_fault_param_t	357
19.5.6	struct ftm_config_t	357
19.6	Macro Definition Documentation	358
19.6.1	FSL_FTM_DRIVER_VERSION	358
19.7	Enumeration Type Documentation	358
19.7.1	ftm_chnl_t	358
19.7.2	ftm_fault_input_t	359
19.7.3	ftm_pwm_mode_t	359
19.7.4	ftm_pwm_level_select_t	359

Contents

Section Number	Title	Page Number
19.7.5	ftm_output_compare_mode_t	359
19.7.6	ftm_input_capture_edge_t	360
19.7.7	ftm_dual_edge_capture_mode_t	360
19.7.8	ftm_quad_decode_mode_t	360
19.7.9	ftm_phase_polarity_t	360
19.7.10	ftm_deadtime_prescale_t	360
19.7.11	ftm_clock_source_t	361
19.7.12	ftm_clock_prescale_t	361
19.7.13	ftm_bdm_mode_t	361
19.7.14	ftm_fault_mode_t	361
19.7.15	ftm_external_trigger_t	362
19.7.16	ftm_pwm_sync_method_t	362
19.7.17	ftm_reload_point_t	362
19.7.18	ftm_interrupt_enable_t	363
19.7.19	ftm_status_flags_t	363
19.7.20	anonymous enum	364
19.8	Function Documentation	364
19.8.1	FTM_Init	364
19.8.2	FTM_Deinit	364
19.8.3	FTM_GetDefaultConfig	365
19.8.4	FTM_SetupPwm	365
19.8.5	FTM_UpdatePwmDutycycle	366
19.8.6	FTM_UpdateChnlEdgeLevelSelect	366
19.8.7	FTM_SetupPwmMode	366
19.8.8	FTM_SetupInputCapture	367
19.8.9	FTM_SetupOutputCompare	367
19.8.10	FTM_SetupDualEdgeCapture	368
19.8.11	FTM_SetupFault	368
19.8.12	FTM_EnableInterrupts	368
19.8.13	FTM_DisableInterrupts	369
19.8.14	FTM_GetEnabledInterrupts	369
19.8.15	FTM_GetStatusFlags	369
19.8.16	FTM_ClearStatusFlags	369
19.8.17	FTM_SetTimerPeriod	370
19.8.18	FTM_GetCurrentTimerCount	370
19.8.19	FTM_StartTimer	371
19.8.20	FTM_StopTimer	372
19.8.21	FTM_SetSoftwareCtrlEnable	372
19.8.22	FTM_SetSoftwareCtrlVal	372
19.8.23	FTM_SetGlobalTimeBaseOutputEnable	372
19.8.24	FTM_SetOutputMask	373
19.8.25	FTM_SetPwmOutputEnable	373
19.8.26	FTM_SetFaultControlEnable	373
19.8.27	FTM_SetDeadTimeEnable	374

Contents

Section Number	Title	Page Number
19.8.28	FTM_SetComplementaryEnable	375
19.8.29	FTM_SetInvertEnable	375
19.8.30	FTM_SetupQuadDecode	376
19.8.31	FTM_GetQuadDecoderFlags	376
19.8.32	FTM_SetQuadDecoderModuloValue	376
19.8.33	FTM_GetQuadDecoderCounterValue	376
19.8.34	FTM_ClearQuadDecoderCounterValue	377
19.8.35	FTM_SetSoftwareTrigger	377
19.8.36	FTM_SetWriteProtection	377
19.8.37	FTM_EnableDmaTransfer	377
Chapter	GPT: General Purpose Timer	
20.1	Overview	379
20.2	Function groups	379
20.2.1	Initialization and deinitialization	379
20.3	Typical use case	379
20.3.1	GPT interrupt example	379
20.4	Data Structure Documentation	382
20.4.1	struct gpt_config_t	382
20.5	Enumeration Type Documentation	383
20.5.1	gpt_clock_source_t	383
20.5.2	gpt_input_capture_channel_t	383
20.5.3	gpt_input_operation_mode_t	384
20.5.4	gpt_output_compare_channel_t	384
20.5.5	gpt_output_operation_mode_t	384
20.5.6	gpt_interrupt_enable_t	384
20.5.7	gpt_status_flag_t	385
20.6	Function Documentation	385
20.6.1	GPT_Init	385
20.6.2	GPT_Deinit	385
20.6.3	GPT_GetDefaultConfig	385
20.6.4	GPT_SoftwareReset	386
20.6.5	GPT_SetClockSource	386
20.6.6	GPT_GetClockSource	386
20.6.7	GPT_SetClockDivider	386
20.6.8	GPT_GetClockDivider	387
20.6.9	GPT_SetOscClockDivider	387
20.6.10	GPT_GetOscClockDivider	387
20.6.11	GPT_StartTimer	387

Contents

Section Number	Title	Page Number
20.6.12	GPT_StopTimer	388
20.6.13	GPT_GetCurrentTimerCount	388
20.6.14	GPT_SetInputOperationMode	388
20.6.15	GPT_GetInputOperationMode	388
20.6.16	GPT_GetInputCaptureValue	389
20.6.17	GPT_SetOutputOperationMode	389
20.6.18	GPT_GetOutputOperationMode	390
20.6.19	GPT_SetOutputCompareValue	390
20.6.20	GPT_GetOutputCompareValue	390
20.6.21	GPT_ForceOutput	391
20.6.22	GPT_EnableInterrupts	391
20.6.23	GPT_DisableInterrupts	391
20.6.24	GPT_GetEnabledInterrupts	391
20.6.25	GPT_GetStatusFlags	392
20.6.26	GPT_ClearStatusFlags	392
Chapter	GPIO: General-Purpose Input/Output Driver	
21.1	Overview	393
21.2	Typical use case	393
21.2.1	Input Operation	393
21.3	Data Structure Documentation	395
21.3.1	struct gpio_pin_config_t	395
21.4	Macro Definition Documentation	395
21.4.1	FSL_GPIO_DRIVER_VERSION	395
21.5	Enumeration Type Documentation	395
21.5.1	gpio_pin_direction_t	395
21.5.2	gpio_interrupt_mode_t	395
21.6	Function Documentation	396
21.6.1	GPIO_PinInit	396
21.6.2	GPIO_PinWrite	397
21.6.3	GPIO_WritePinOutput	397
21.6.4	GPIO_PortSet	397
21.6.5	GPIO_SetPinsOutput	397
21.6.6	GPIO_PortClear	398
21.6.7	GPIO_ClearPinsOutput	399
21.6.8	GPIO_PortToggle	399
21.6.9	GPIO_PinRead	399
21.6.10	GPIO_ReadPinInput	399
21.6.11	GPIO_PinReadPadStatus	400

Contents

Section Number	Title	Page Number
21.6.12	GPIO_ReadPadStatus	401
21.6.13	GPIO_PinSetInterruptConfig	401
21.6.14	GPIO_SetPinInterruptConfig	401
21.6.15	GPIO_PortEnableInterrupts	401
21.6.16	GPIO_EnableInterrupts	402
21.6.17	GPIO_PortDisableInterrupts	402
21.6.18	GPIO_DisableInterrupts	402
21.6.19	GPIO_PortGetInterruptFlags	402
21.6.20	GPIO_GetPinsInterruptFlags	403
21.6.21	GPIO_PortClearInterruptFlags	403
21.6.22	GPIO_ClearPinsInterruptFlags	403

Chapter INTMUX: Interrupt Multiplexer Driver

22.1	Overview	405
22.2	Typical use case	405
22.2.1	Channel Configure	405
22.3	Macro Definition Documentation	406
22.3.1	FSL_INTMUX_DRIVER_VERSION	406
22.4	Enumeration Type Documentation	406
22.4.1	intmux_channel_logic_mode_t	406
22.5	Function Documentation	406
22.5.1	INTMUX_Init	406
22.5.2	INTMUX_Deinit	406
22.5.3	INTMUX_ResetChannel	406
22.5.4	INTMUX_SetChannelMode	407
22.5.5	INTMUX_EnableInterrupt	407
22.5.6	INTMUX_DisableInterrupt	407
22.5.7	INTMUX_GetChannelPendingSources	408

Chapter IRQSTEER: Interrupt Request Steering Driver

23.1	Overview	409
23.2	Macro Definition Documentation	411
23.2.1	FSL_IRQSTEER_DRIVER_VERSION	411
23.2.2	IRQSTEER_INT_SRC_REG_WIDTH	411
23.2.3	IRQSTEER_INT_SRC_REG_INDEX	411
23.2.4	IRQSTEER_INT_SRC_BIT_OFFSET	411
23.2.5	IRQSTEER_INT_SRC_NUM	411
23.3	Enumeration Type Documentation	411

Contents

Section Number	Title	Page Number
23.3.1	irqsteer_int_group_t	411
23.3.2	irqsteer_int_master_t	412
23.4	Function Documentation	412
23.4.1	IRQSTEER_Init	412
23.4.2	IRQSTEER_Deinit	412
23.4.3	IRQSTEER_EnableInterrupt	412
23.4.4	IRQSTEER_DisableInterrupt	413
23.4.5	IRQSTEER_SetInterrupt	413
23.4.6	IRQSTEER_EnableMasterInterrupt	413
23.4.7	IRQSTEER_DisableMasterInterrupt	414
23.4.8	IRQSTEER_IsInterruptSet	414
23.4.9	IRQSTEER_IsMasterInterruptSet	415
23.4.10	IRQSTEER_GetGroupInterruptStatus	415
23.4.11	IRQSTEER_GetMasterNextInterrupt	415
Chapter	ISI: Image Sensing Interface	
24.1	Overview	417
24.2	Typical use case	417
24.2.1	Output buffer	417
24.2.2	Output panic and overflow	419
24.3	Data Structure Documentation	425
24.3.1	struct isi_config_t	425
24.3.2	struct isi_csc_config_t	426
24.3.3	struct isi_crop_config_t	428
24.3.4	struct isi_region_alpha_config_t	429
24.3.5	struct isi_input_mem_config_t	429
24.4	Macro Definition Documentation	430
24.4.1	FSL_ISI_DRIVER_VERSION	430
24.5	Enumeration Type Documentation	430
24.5.1	_isi_interrupt	430
24.5.2	isi_output_format_t	430
24.5.3	isi_chain_mode_t	432
24.5.4	isi_deint_mode_t	432
24.5.5	isi_threshold_t	433
24.5.6	isi_csc_mode_t	433
24.5.7	isi_flip_mode_t	433
24.5.8	isi_input_mem_format_t	433
24.6	Function Documentation	434
24.6.1	ISI_Init	434

Contents

Section Number	Title	Page Number
24.6.2	ISI_Deinit	434
24.6.3	ISI_Reset	435
24.6.4	ISI_EnableInterrupts	435
24.6.5	ISI_DisableInterrupts	435
24.6.6	ISI_GetInterruptStatus	436
24.6.7	ISI_ClearInterruptStatus	436
24.6.8	ISI_GetOverflowBytes	436
24.6.9	ISI_SetConfig	437
24.6.10	ISI_GetDefaultConfig	437
24.6.11	ISI_SetScalerConfig	438
24.6.12	ISI_SetColorSpaceConversionConfig	438
24.6.13	ISI_ColorSpaceConversionGetDefaultConfig	438
24.6.14	ISI_EnableColorSpaceConversion	439
24.6.15	ISI_SetCropConfig	439
24.6.16	ISI_CropGetDefaultConfig	440
24.6.17	ISI_EnableCrop	440
24.6.18	ISI_SetGlobalAlpha	440
24.6.19	ISI_EnableGlobalAlpha	441
24.6.20	ISI_SetRegionAlphaConfig	441
24.6.21	ISI_RegionAlphaGetDefaultConfig	441
24.6.22	ISI_EnableRegionAlpha	443
24.6.23	ISI_SetInputMemConfig	443
24.6.24	ISI_InputMemGetDefaultConfig	443
24.6.25	ISI_SetInputMemAddr	444
24.6.26	ISI_TriggerInputMemRead	444
24.6.27	ISI_SetFlipMode	444
24.6.28	ISI_SetOutputBufferAddr	444
24.6.29	ISI_Start	445
24.6.30	ISI_Stop	445

Chapter LDB: LVDS Display Bridge

25.1	Overview	447
25.2	Data Structure Documentation	448
25.2.1	struct ldb_channel_config_t	448
25.3	Macro Definition Documentation	448
25.3.1	FSL_LDB_DRIVER_VERSION	448
25.4	Enumeration Type Documentation	448
25.4.1	ldb_output_bus_t	448
25.4.2	_ldb_input_flag	448
25.5	Function Documentation	448

Contents

Section Number	Title	Page Number
25.5.1	LDB_Init	448
25.5.2	LDB_Deinit	449
25.5.3	LDB_InitChannel	449
25.5.4	LDB_DeinitChannel	449
Chapter	LPADC: 12-bit SAR Analog-to-Digital Converter Driver	
26.1	Overview	451
26.2	Typical use case	451
26.2.1	Polling Configuration	451
26.2.2	Interrupt Configuration	451
26.3	Data Structure Documentation	454
26.3.1	struct lpadc_config_t	454
26.3.2	struct lpadc_conv_command_config_t	455
26.3.3	struct lpadc_conv_trigger_config_t	457
26.3.4	struct lpadc_conv_result_t	458
26.4	Macro Definition Documentation	458
26.4.1	FSL_LPADC_DRIVER_VERSION	458
26.4.2	LPADC_GET_ACTIVE_COMMAND_STATUS	458
26.4.3	LPADC_GET_ACTIVE_TRIGGER_STATUE	458
26.5	Enumeration Type Documentation	458
26.5.1	_lpadc_status_flags	458
26.5.2	_lpadc_interrupt_enable	459
26.5.3	lpadc_sample_scale_mode_t	459
26.5.4	lpadc_sample_channel_mode_t	459
26.5.5	lpadc_hardware_average_mode_t	459
26.5.6	lpadc_sample_time_mode_t	460
26.5.7	lpadc_hardware_compare_mode_t	460
26.5.8	lpadc_reference_voltage_source_t	460
26.5.9	lpadc_power_level_mode_t	461
26.5.10	lpadc_trigger_priority_policy_t	461
26.6	Function Documentation	461
26.6.1	LPADC_Init	461
26.6.2	LPADC_GetDefaultConfig	462
26.6.3	LPADC_Deinit	462
26.6.4	LPADC_Enable	462
26.6.5	LPADC_DoResetFIFO	463
26.6.6	LPADC_DoResetConfig	463
26.6.7	LPADC_GetStatusFlags	463
26.6.8	LPADC_ClearStatusFlags	463

Contents

Section Number	Title	Page Number
26.6.9	LPADC_EnableInterrupts	464
26.6.10	LPADC_DisableInterrupts	464
26.6.11	LPADC_EnableFIFOWatermarkDMA	464
26.6.12	LPADC_GetConvResultCount	464
26.6.13	LPADC_GetConvResult	465
26.6.14	LPADC_SetConvTriggerConfig	465
26.6.15	LPADC_GetDefaultConvTriggerConfig	465
26.6.16	LPADC_DoSoftwareTrigger	466
26.6.17	LPADC_SetConvCommandConfig	466
26.6.18	LPADC_GetDefaultConvCommandConfig	466
Chapter	LPI2C: Low Power Inter-Integrated Circuit Driver	
27.1	Overview	469
27.2	Macro Definition Documentation	470
27.2.1	FSL_LPI2C_DRIVER_VERSION	470
27.2.2	I2C_RETRY_TIMES	470
27.3	Enumeration Type Documentation	470
27.3.1	anonymous enum	470
27.4	LPI2C Master Driver	471
27.4.1	Overview	471
27.4.2	Data Structure Documentation	474
27.4.3	Typedef Documentation	478
27.4.4	Enumeration Type Documentation	478
27.4.5	Function Documentation	481
27.5	LPI2C Slave Driver	495
27.5.1	Overview	495
27.5.2	Data Structure Documentation	497
27.5.3	Typedef Documentation	501
27.5.4	Enumeration Type Documentation	502
27.5.5	Function Documentation	503
27.6	LPI2C Master DMA Driver	512
27.6.1	Overview	512
27.6.2	Data Structure Documentation	512
27.6.3	Typedef Documentation	514
27.6.4	Function Documentation	515
27.7	LPI2C FreeRTOS Driver	518
27.7.1	Overview	518
27.7.2	Macro Definition Documentation	518
27.7.3	Function Documentation	518

Contents

Section Number	Title	Page Number
27.8	LPI2C CMSIS Driver	521
27.8.1	LPI2C CMSIS Driver	521
Chapter	LPIT: Low-Power Interrupt Timer	
28.1	Overview	523
28.2	Function groups	523
28.2.1	Initialization and deinitialization	523
28.2.2	Timer period Operations	523
28.2.3	Start and Stop timer operations	523
28.2.4	Status	524
28.2.5	Interrupt	524
28.3	Typical use case	524
28.3.1	LPIT tick example	524
28.4	Data Structure Documentation	526
28.4.1	struct lpit_chnl_params_t	526
28.4.2	struct lpit_config_t	527
28.5	Enumeration Type Documentation	527
28.5.1	lpit_chnl_t	527
28.5.2	lpit_timer_modes_t	527
28.5.3	lpit_trigger_select_t	528
28.5.4	lpit_trigger_source_t	528
28.5.5	lpit_interrupt_enable_t	528
28.5.6	lpit_status_flags_t	529
28.6	Function Documentation	529
28.6.1	LPIT_Init	529
28.6.2	LPIT_Deinit	529
28.6.3	LPIT_GetDefaultConfig	529
28.6.4	LPIT_SetupChannel	530
28.6.5	LPIT_EnableInterrupts	530
28.6.6	LPIT_DisableInterrupts	530
28.6.7	LPIT_GetEnabledInterrupts	531
28.6.8	LPIT_GetStatusFlags	532
28.6.9	LPIT_ClearStatusFlags	532
28.6.10	LPIT_SetTimerPeriod	532
28.6.11	LPIT_GetCurrentTimerCount	533
28.6.12	LPIT_StartTimer	533
28.6.13	LPIT_StopTimer	533
28.6.14	LPIT_Reset	534

Contents

Section Number	Title	Page Number
Chapter	LPSPI: Low Power Serial Peripheral Interface	
29.1	Overview	535
29.2	LPSPI Peripheral driver	536
29.2.1	Overview	536
29.2.2	Function groups	536
29.2.3	Typical use case	536
29.2.4	Data Structure Documentation	543
29.2.5	Macro Definition Documentation	549
29.2.6	Typedef Documentation	550
29.2.7	Enumeration Type Documentation	551
29.2.8	Function Documentation	556
29.2.9	Variable Documentation	572
29.3	LPSPI eDMA Driver	573
29.3.1	Overview	573
29.3.2	Data Structure Documentation	574
29.3.3	Macro Definition Documentation	579
29.3.4	Typedef Documentation	579
29.3.5	Function Documentation	580
29.4	LPSPI FreeRTOS Driver	585
29.4.1	Overview	585
29.4.2	Macro Definition Documentation	585
29.4.3	Function Documentation	585
29.5	LPSPI CMSIS Driver	588
29.5.1	Function groups	588
29.5.2	Typical use case	589
Chapter	LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver	
30.1	Overview	591
30.2	LPUART Driver	592
30.2.1	Overview	592
30.2.2	Typical use case	592
30.2.3	Data Structure Documentation	597
30.2.4	Macro Definition Documentation	601
30.2.5	Typedef Documentation	601
30.2.6	Enumeration Type Documentation	601
30.2.7	Function Documentation	605
30.3	LPUART eDMA Driver	619
30.3.1	Overview	619

Contents

Section Number	Title	Page Number
30.3.2	Data Structure Documentation	620
30.3.3	Macro Definition Documentation	621
30.3.4	Typedef Documentation	621
30.3.5	Function Documentation	621
30.4	LPUART FreeRTOS Driver	625
30.4.1	Overview	625
30.4.2	Data Structure Documentation	625
30.4.3	Macro Definition Documentation	626
30.4.4	Function Documentation	626
30.5	LPUART CMSIS Driver	628
30.5.1	Function groups	628
Chapter	MIPI CSI2 RX: MIPI CSI2 RX Driver	
31.1	Overview	631
31.2	Data Structure Documentation	633
31.2.1	struct csi2rx_config_t	633
31.3	Macro Definition Documentation	634
31.3.1	FSL_CSI2RX_DRIVER_VERSION	634
31.4	Enumeration Type Documentation	634
31.4.1	_csi2rx_data_lane	634
31.4.2	_csi2rx_payload	634
31.4.3	_csi2rx_bit_error	635
31.4.4	csi2rx_ppi_error_t	635
31.4.5	_csi2rx_interrupt	635
31.4.6	_csi2rx_ulps_status	635
31.5	Function Documentation	636
31.5.1	CSI2RX_Init	636
31.5.2	CSI2RX_Deinit	637
31.5.3	CSI2RX_GetBitError	637
31.5.4	CSI2RX_GetEccBitErrorPosition	638
31.5.5	CSI2RX_GetUlpsStatus	639
31.5.6	CSI2RX_GetPpiErrorDataLanes	639
31.5.7	CSI2RX_EnableInterrupts	640
31.5.8	CSI2RX_DisableInterrupts	640
31.5.9	CSI2RX_GetInterruptStatus	641
Chapter	MIPI DSI: MIPI DSI Host Controller	
32.1	Overview	643

Contents

Section Number	Title	Page Number
32.2	MIPI DSI Driver	644
32.2.1	Overview	644
32.2.2	Command mode data transfer	644
32.2.3	Data Structure Documentation	652
32.2.4	Typedef Documentation	657
32.2.5	Enumeration Type Documentation	658
32.2.6	Function Documentation	663
Chapter	MU: Messaging Unit	
33.1	Overview	675
33.2	Function description	675
33.2.1	MU initialization	675
33.2.2	MU message	675
33.2.3	MU flags	676
33.2.4	Status and interrupt	676
33.2.5	MU misc functions	676
33.3	Macro Definition Documentation	679
33.3.1	FSL_MU_DRIVER_VERSION	679
33.4	Enumeration Type Documentation	679
33.4.1	_mu_status_flags	679
33.4.2	_mu_interrupt_enable	679
33.4.3	_mu_interrupt_trigger	680
33.5	Function Documentation	680
33.5.1	MU_Init	680
33.5.2	MU_Deinit	680
33.5.3	MU_SendMsgNonBlocking	680
33.5.4	MU_SendMsg	681
33.5.5	MU_ReceiveMsgNonBlocking	681
33.5.6	MU_ReceiveMsg	682
33.5.7	MU_SetFlagsNonBlocking	682
33.5.8	MU_SetFlags	683
33.5.9	MU_GetFlags	683
33.5.10	MU_GetStatusFlags	683
33.5.11	MU_GetInterruptsPending	684
33.5.12	MU_ClearStatusFlags	684
33.5.13	MU_EnableInterrupts	685
33.5.14	MU_DisableInterrupts	685
33.5.15	MU_TriggerInterrupts	686
33.5.16	MU_ClearNmi	686
33.5.17	MU_BootCoreB	686

Contents

Section Number	Title	Page Number
33.5.18	MU_HoldCoreBReset	687
33.5.19	MU_BootOtherCore	687
33.5.20	MU_HoldOtherCoreReset	687
33.5.21	MU_ResetBothSides	688
33.5.22	MU_HardwareResetOtherCore	688
33.5.23	MU_SetClockOnOtherCoreEnable	689
33.5.24	MU_GetOtherCorePowerMode	689

Chapter PRG: Prefetch Resolve Gasket

34.1	Overview	691
34.2	Data Structure Documentation	692
34.2.1	struct prg_buffer_config_t	692
34.3	Macro Definition Documentation	692
34.3.1	FSL_PRG_DRIVER_VERSION	692
34.4	Enumeration Type Documentation	692
34.4.1	prg_data_type_t	692
34.5	Function Documentation	692
34.5.1	PRG_Init	692
34.5.2	PRG_Deinit	693
34.5.3	PRG_Enable	693
34.5.4	PRG_EnableShadowLoad	693
34.5.5	PRG_UpdateRegister	693
34.5.6	PRG_SetBufferConfig	695
34.5.7	PRG_BufferGetDefaultConfig	695
34.5.8	PRG_SetBufferAddr	695

Chapter RGPIO: Rapid General-Purpose Input/Output Driver

35.1	Overview	697
35.2	Data Structure Documentation	697
35.2.1	struct rgpio_pin_config_t	697
35.3	Macro Definition Documentation	698
35.3.1	FSL_RGPIO_DRIVER_VERSION	698
35.4	Enumeration Type Documentation	698
35.4.1	rgpio_pin_direction_t	698
35.5	RGPIO Driver	699
35.5.1	Overview	699

Contents

Section Number	Title	Page Number
35.5.2	Typical use case	699
35.5.3	Function Documentation	700
35.6	FGPIO Driver	705
35.6.1	Typical use case	705
Chapter	SAI: Serial Audio Interface	
36.1	Overview	707
36.2	Typical configurations	707
36.3	Typical use case	708
36.3.1	SAI Send/receive using an interrupt method	708
36.3.2	SAI Send/receive using a DMA method	708
36.4	SAI Driver	709
36.4.1	Overview	709
36.4.2	Data Structure Documentation	717
36.4.3	Macro Definition Documentation	721
36.4.4	Enumeration Type Documentation	721
36.4.5	Function Documentation	725
36.5	SAI EDMA Driver	754
36.5.1	Overview	754
36.5.2	Data Structure Documentation	755
36.5.3	Function Documentation	756
Chapter	SEMA42: Hardware Semaphores Driver	
37.1	Overview	765
37.2	Typical use case	765
37.3	Macro Definition Documentation	767
37.3.1	SEMA42_GATE_NUM_RESET_ALL	767
37.3.2	SEMA42_GATEn	767
37.4	Enumeration Type Documentation	767
37.4.1	anonymous enum	767
37.4.2	sema42_gate_status_t	767
37.5	Function Documentation	768
37.5.1	SEMA42_Init	768
37.5.2	SEMA42_Deinit	768
37.5.3	SEMA42_TryLock	768

Contents

Section Number	Title	Page Number
37.5.4	SEMA42_Lock	769
37.5.5	SEMA42_Unlock	770
37.5.6	SEMA42_GetGateStatus	770
37.5.7	SEMA42_ResetGate	770
37.5.8	SEMA42_ResetAllGates	771
Chapter	TPM: Timer PWM Module	
38.1	Overview	773
38.2	Introduction of TPM	773
38.2.1	Initialization and deinitialization	773
38.2.2	PWM Operations	773
38.2.3	Input capture operations	774
38.2.4	Output compare operations	774
38.2.5	Quad decode	774
38.2.6	Fault operation	774
38.2.7	Status	774
38.2.8	Interrupt	774
38.3	Typical use case	775
38.3.1	PWM output	775
38.4	Data Structure Documentation	779
38.4.1	struct tpm_chnl_pwm_signal_param_t	779
38.4.2	struct tpm_dual_edge_capture_param_t	779
38.4.3	struct tpm_phase_params_t	780
38.4.4	struct tpm_config_t	780
38.5	Enumeration Type Documentation	780
38.5.1	tpm_chnl_t	780
38.5.2	tpm_pwm_mode_t	781
38.5.3	tpm_pwm_level_select_t	781
38.5.4	tpm_trigger_select_t	781
38.5.5	tpm_trigger_source_t	782
38.5.6	tpm_output_compare_mode_t	782
38.5.7	tpm_input_capture_edge_t	782
38.5.8	tpm_quad_decode_mode_t	782
38.5.9	tpm_phase_polarity_t	783
38.5.10	tpm_clock_source_t	783
38.5.11	tpm_clock_prescale_t	783
38.5.12	tpm_interrupt_enable_t	783
38.5.13	tpm_status_flags_t	784
38.6	Function Documentation	784

Contents

Section Number	Title	Page Number
38.6.1	TPM_Init	784
38.6.2	TPM_Deinit	784
38.6.3	TPM_GetDefaultConfig	784
38.6.4	TPM_SetupPwm	785
38.6.5	TPM_UpdatePwmDutycycle	785
38.6.6	TPM_UpdateChnlEdgeLevelSelect	786
38.6.7	TPM_SetupInputCapture	786
38.6.8	TPM_SetupOutputCompare	786
38.6.9	TPM_SetupDualEdgeCapture	787
38.6.10	TPM_SetupQuadDecode	787
38.6.11	TPM_EnableInterrupts	787
38.6.12	TPM_DisableInterrupts	788
38.6.13	TPM_GetEnabledInterrupts	788
38.6.14	TPM_GetStatusFlags	788
38.6.15	TPM_ClearStatusFlags	788
38.6.16	TPM_SetTimerPeriod	789
38.6.17	TPM_GetCurrentTimerCount	789
38.6.18	TPM_StartTimer	790
38.6.19	TPM_StopTimer	791
38.6.20	TPM_Reset	791

Chapter TSTMR: Timestamp Timer Driver

39.1	Overview	793
39.2	Function Documentation	793
39.2.1	TSTMR_ReadTimeStamp	793
39.2.2	TSTMR_DelayUs	793

Chapter WDOG32: 32-bit Watchdog Timer

40.1	Overview	795
40.2	Typical use case	795
40.3	Data Structure Documentation	797
40.3.1	struct wdog32_work_mode_t	797
40.3.2	struct wdog32_config_t	797
40.4	Macro Definition Documentation	797
40.4.1	FSL_WDOG32_DRIVER_VERSION	797
40.5	Enumeration Type Documentation	798
40.5.1	wdog32_clock_source_t	798
40.5.2	wdog32_clock_prescaler_t	798

Contents

Section Number	Title	Page Number
40.5.3	wdog32_test_mode_t	798
40.5.4	_wdog32_interrupt_enable_t	798
40.5.5	_wdog32_status_flags_t	798
40.6	Function Documentation	799
40.6.1	WDOG32_GetDefaultConfig	799
40.6.2	WDOG32_Init	799
40.6.3	WDOG32_Deinit	800
40.6.4	WDOG32_Enable	800
40.6.5	WDOG32_Disable	800
40.6.6	WDOG32_EnableInterrupts	800
40.6.7	WDOG32_DisableInterrupts	802
40.6.8	WDOG32_GetStatusFlags	802
40.6.9	WDOG32_ClearStatusFlags	803
40.6.10	WDOG32_SetTimeoutValue	803
40.6.11	WDOG32_SetWindowValue	804
40.6.12	WDOG32_Unlock	804
40.6.13	WDOG32_Refresh	804
40.6.14	WDOG32_GetCounterValue	805
Chapter	Debug Console	
41.1	Overview	807
41.2	Function groups	807
41.2.1	Initialization	807
41.2.2	Advanced Feature	808
41.3	Typical use case	812
41.4	Macro Definition Documentation	814
41.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	814
41.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK	814
41.4.3	DEBUGCONSOLE_DISABLE	814
41.4.4	SDK_DEBUGCONSOLE	814
41.4.5	PRINTF	814
41.5	Function Documentation	815
41.5.1	DbgConsole_Init	815
41.5.2	DbgConsole_Deinit	815
41.5.3	DbgConsole_Printf	815
41.5.4	DbgConsole_Putchar	816
41.5.5	DbgConsole_Scanf	816
41.5.6	DbgConsole_Getchar	817
41.5.7	DbgConsole_BlockingPrintf	817

Contents

Section Number	Title	Page Number
41.5.8	DbgConsole_Flush	817
41.5.9	StrFormatPrintf	818
41.5.10	StrFormatScanf	818
41.6	Semihosting	819
41.6.1	Guide Semihosting for IAR	819
41.6.2	Guide Semihosting for Keil μ Vision	819
41.6.3	Guide Semihosting for MCUXpresso IDE	820
41.6.4	Guide Semihosting for ARMGCC	820
41.7	SWO	823
41.7.1	Guide SWO for SDK	823
41.7.2	Guide SWO for Keil μ Vision	824
41.7.3	Guide SWO for MCUXpresso IDE	825
41.7.4	Guide SWO for ARMGCC	825
Chapter	Notification Framework	
42.1	Overview	827
42.2	Notifier Overview	827
42.3	Data Structure Documentation	829
42.3.1	struct notifier_notification_block_t	829
42.3.2	struct notifier_callback_config_t	830
42.3.3	struct notifier_handle_t	830
42.4	Typedef Documentation	831
42.4.1	notifier_user_config_t	831
42.4.2	notifier_user_function_t	831
42.4.3	notifier_callback_t	832
42.5	Enumeration Type Documentation	832
42.5.1	_notifier_status	832
42.5.2	notifier_policy_t	833
42.5.3	notifier_notification_type_t	833
42.5.4	notifier_callback_type_t	833
42.6	Function Documentation	834
42.6.1	NOTIFIER_CreateHandle	834
42.6.2	NOTIFIER_SwitchConfig	835
42.6.3	NOTIFIER_GetErrorCallbackIndex	836
Chapter	Shell	
43.1	Overview	837

Contents

Section Number	Title	Page Number
43.2	Function groups	837
43.2.1	Initialization	837
43.2.2	Advanced Feature	837
43.2.3	Shell Operation	837
43.3	Data Structure Documentation	839
43.3.1	struct shell_command_t	839
43.4	Macro Definition Documentation	840
43.4.1	SHELL_NON_BLOCKING_MODE	840
43.4.2	SHELL_AUTO_COMPLETE	840
43.4.3	SHELL_BUFFER_SIZE	840
43.4.4	SHELL_MAX_ARGS	840
43.4.5	SHELL_HISTORY_COUNT	840
43.4.6	SHELL_HANDLE_SIZE	840
43.4.7	SHELL_USE_COMMON_TASK	840
43.4.8	SHELL_TASK_PRIORITY	840
43.4.9	SHELL_TASK_STACK_SIZE	840
43.4.10	SHELL_HANDLE_DEFINE	840
43.4.11	SHELL_COMMAND_DEFINE	841
43.4.12	SHELL_COMMAND	841
43.5	Typedef Documentation	842
43.5.1	cmd_function_t	842
43.6	Enumeration Type Documentation	842
43.6.1	shell_status_t	842
43.7	Function Documentation	842
43.7.1	SHELL_Init	842
43.7.2	SHELL_RegisterCommand	843
43.7.3	SHELL_UnregisterCommand	843
43.7.4	SHELL_Write	844
43.7.5	SHELL_Printf	844
43.7.6	SHELL_Task	845
Chapter	CODEC codec Driver	
44.1	Overview	847
44.2	codec common Driver	848
44.2.1	Overview	848
44.2.2	Data Structure Documentation	852
44.2.3	Macro Definition Documentation	853
44.2.4	Enumeration Type Documentation	853
44.2.5	Function Documentation	858

Contents

Section Number	Title	Page Number
44.3	cs42888 Driver	862
44.3.1	Overview	862
44.3.2	Data Structure Documentation	864
44.3.3	Macro Definition Documentation	865
44.3.4	Enumeration Type Documentation	865
44.3.5	Function Documentation	866
44.3.6	cs42888 adapter	872
44.4	da7212 Driver	878
44.4.1	Overview	878
44.4.2	Data Structure Documentation	881
44.4.3	Macro Definition Documentation	882
44.4.4	Enumeration Type Documentation	882
44.4.5	Function Documentation	884
44.4.6	da7212 adapter	889
44.5	sgtl5000 Driver	895
44.5.1	Overview	895
44.5.2	Data Structure Documentation	897
44.5.3	Macro Definition Documentation	899
44.5.4	Enumeration Type Documentation	899
44.5.5	Function Documentation	900
44.5.6	sgtl5000 adapter	907
44.6	wm8960 Driver	913
44.6.1	Overview	913
44.6.2	Data Structure Documentation	916
44.6.3	Macro Definition Documentation	918
44.6.4	Enumeration Type Documentation	918
44.6.5	Function Documentation	920
44.6.6	wm8960 adapter	926
44.7	wm8904 Driver	932
44.7.1	Overview	932
44.7.2	Data Structure Documentation	936
44.7.3	Macro Definition Documentation	937
44.7.4	Enumeration Type Documentation	937
44.7.5	Function Documentation	940
44.7.6	wm8904 adapter	949
Chapter	Serial_Manager	
45.1	Overview	955
45.2	Data Structure Documentation	957

Contents

Section Number	Title	Page Number
45.2.1	struct serial_manager_config_t	957
45.2.2	struct serial_manager_callback_message_t	957
45.3	Macro Definition Documentation	958
45.3.1	SERIAL_MANAGER_HANDLE_SIZE	958
45.3.2	SERIAL_MANAGER_HANDLE_DEFINE	958
45.3.3	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	958
45.3.4	SERIAL_MANAGER_READ_HANDLE_DEFINE	959
45.3.5	SERIAL_MANAGER_USE_COMMON_TASK	959
45.3.6	SERIAL_MANAGER_TASK_PRIORITY	959
45.3.7	SERIAL_MANAGER_TASK_STACK_SIZE	959
45.4	Enumeration Type Documentation	959
45.4.1	serial_port_type_t	959
45.4.2	serial_manager_status_t	960
45.5	Function Documentation	960
45.5.1	SerialManager_Init	960
45.5.2	SerialManager_Deinit	961
45.5.3	SerialManager_OpenWriteHandle	962
45.5.4	SerialManager_CloseWriteHandle	963
45.5.5	SerialManager_OpenReadHandle	963
45.5.6	SerialManager_CloseReadHandle	964
45.5.7	SerialManager_WriteBlocking	964
45.5.8	SerialManager_ReadBlocking	965
45.5.9	SerialManager_EnterLowpower	966
45.5.10	SerialManager_ExitLowpower	966
45.6	Serial Port Uart	968
45.6.1	Overview	968
45.6.2	Data Structure Documentation	968
45.6.3	Enumeration Type Documentation	969
45.7	Serial Port SWO	970
45.7.1	Overview	970
45.7.2	Data Structure Documentation	970
45.7.3	Enumeration Type Documentation	970
Chapter	Cache	
46.1	Overview	971
46.2	Macro Definition Documentation	972
46.2.1	FSL_CACHE_DRIVER_VERSION	972
46.2.2	L1CODEBUSCACHE_LINESIZE_BYTE	972
46.2.3	L1SYSTEMBUSCACHE_LINESIZE_BYTE	972

Contents

Section Number	Title	Page Number
46.3	Function Documentation	973
46.3.1	L1CACHE_InvalidateCodeCacheByRange	973
46.3.2	L1CACHE_CleanCodeCacheByRange	974
46.3.3	L1CACHE_CleanInvalidateCodeCacheByRange	974
46.3.4	L1CACHE_EnableCodeCacheWriteBuffer	975
46.3.5	L1CACHE_InvalidateSystemCacheByRange	976
46.3.6	L1CACHE_CleanSystemCacheByRange	976
46.3.7	L1CACHE_CleanInvalidateSystemCacheByRange	976
46.3.8	L1CACHE_EnableSystemCacheWriteBuffer	977
46.3.9	L1CACHE_InvalidateICacheByRange	977
46.3.10	L1CACHE_InvalidateDCacheByRange	977
46.3.11	L1CACHE_CleanDCacheByRange	978
46.3.12	L1CACHE_CleanInvalidateDCacheByRange	978
46.3.13	ICACHE_InvalidateByRange	978
46.3.14	DCACHE_InvalidateByRange	979
46.3.15	DCACHE_CleanByRange	979
46.3.16	DCACHE_CleanInvalidateByRange	980
Chapter	Data Structure Documentation	
47.0.17	codec_i2c_config_t Struct Reference	983

Chapter 1

Introduction

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

Trademarks

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

How to Reach Us:

Home Page: nxp.com

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: <http://www.nxp.com/SalesTermsandConditions>.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, 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, Arm7TDMI, 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 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

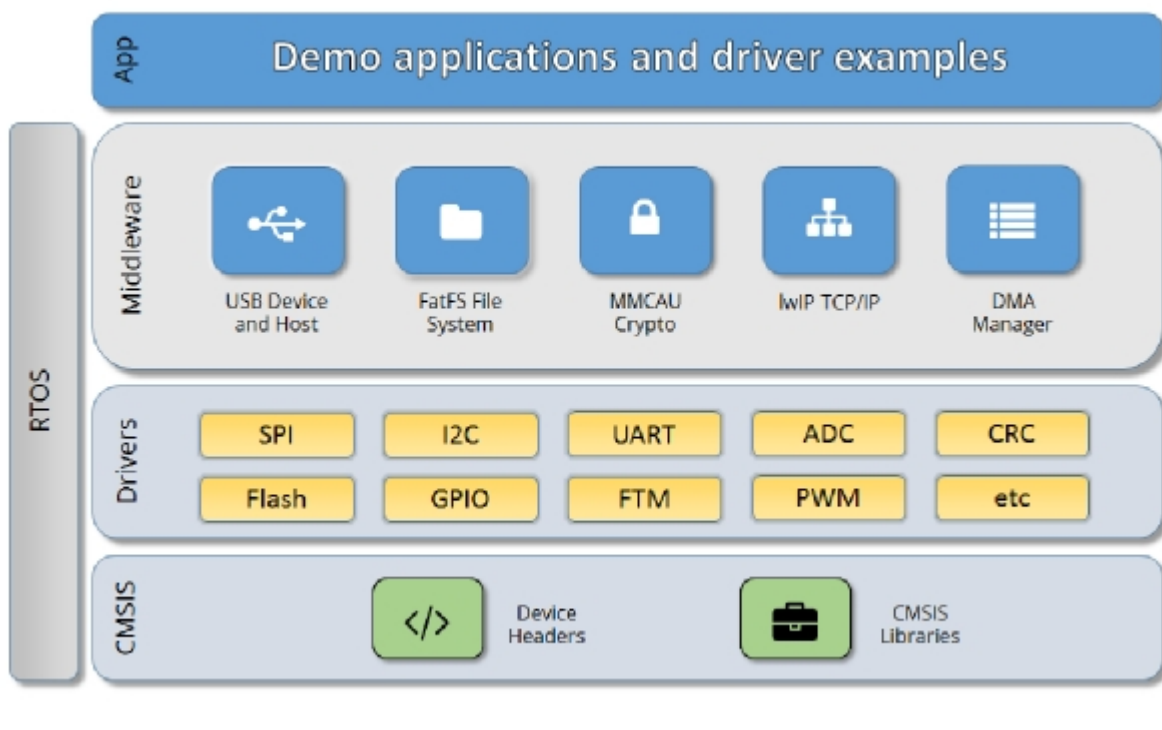


Figure 1: MCUXpresso SDK Block Diagram

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/.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 reimplement 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

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_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_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_RxRingBufferOverflow` = 1308
- `kStatus_LPUART_RxHardwareOverflow` = 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_SAI_TxBusy` = 1900
- `kStatus_SAI_RxBusy` = 1901
- `kStatus_SAI_TxError` = 1902
- `kStatus_SAI_RxError` = 1903
- `kStatus_SAI_QueueFull` = 1904
- `kStatus_SAI_TxIdle` = 1905
- `kStatus_SAI_RxIdle` = 1906
- `kStatus_SEMA42_Busy` = 1600
- `kStatus_SEMA42_Reseting` = 1601
- `kStatus_NOTIFIER_ErrorNotificationBefore` = 9800
- `kStatus_NOTIFIER_ErrorNotificationAfter` = 9801

Chapter 5

Deprecated List

Global [CS42888_SetFuncMode](#) ([cs42888_handle_t](#) *handle, [cs42888_func_mode](#) mode)

api, Do not use it anymore. It has been superceded by [CS42888_SelectFunctionalMode](#).

Global [ENET_StartExtC45SMIRRead](#) ([ENET_Type](#) *base, [uint32_t](#) phyAddr, [uint32_t](#) phyReg)

Do not use this function. It has been superceded by [ENET_StartExtC45SMIWriteReg](#) and [ENET_StartExtC45SMIRReadData](#).

Global [ENET_StartExtC45SMIWrite](#) ([ENET_Type](#) *base, [uint32_t](#) phyAddr, [uint32_t](#) phyReg, [uint32_t](#) data)

Do not use this function. It has been superceded by [ENET_StartExtC45SMIWriteReg](#) and [ENET_StartExtC45SMIWriteData](#).

Global [flexcan_clock_source_t](#)

Do not use the [kFLEXCAN_ClkSrcOs](#). It has been superceded [kFLEXCAN_ClkSrc0](#)

Do not use the [kFLEXCAN_ClkSrcPeri](#). It has been superceded [kFLEXCAN_ClkSrc1](#)

Global [GPIO_ClearPinsOutput](#) ([GPIO_Type](#) *base, [uint32_t](#) mask)

Do not use this function. It has been superceded by [GPIO_PortClear](#).

Global [GPIO_DisableInterrupts](#) ([GPIO_Type](#) *base, [uint32_t](#) mask)

Do not use this function. It has been superceded by [GPIO_PortDisableInterrupts](#).

Global [GPIO_ReadPadStatus](#) ([GPIO_Type](#) *base, [uint32_t](#) pin)

Do not use this function. It has been superceded by [GPIO_PinReadPadStatus](#).

Global [GPIO_ReadPinInput](#) ([GPIO_Type](#) *base, [uint32_t](#) pin)

Do not use this function. It has been superceded by [GPIO_PinRead](#).

Global [GPIO_SetPinInterruptConfig](#) ([GPIO_Type](#) *base, [uint32_t](#) pin, [gpio_interrupt_mode_t](#) pin-InterruptMode)

Do not use this function. It has been superceded by [GPIO_PinSetInterruptConfig](#).

Global [GPIO_SetPinsOutput](#) ([GPIO_Type](#) *base, [uint32_t](#) mask)

Do not use this function. It has been superceded by [GPIO_PortSet](#).

Global [GPIO_WritePinOutput](#) ([GPIO_Type](#) *base, [uint32_t](#) pin, [uint8_t](#) output)

Do not use this function. It has been superceded by [GPIO_PinWrite](#).

Global [RGPIO_ClearPinsOutput](#) ([RGPIO_Type](#) *base, [uint32_t](#) mask)

Do not use this function. It has been superceded by [RGPIO_PortClear](#).

Global [RGPIO_ReadPinInput](#) (RGPIO_Type *base, uint32_t pin)

Do not use this function. It has been superceded by [RGPIO_PinRead](#).

Global [RGPIO_SetPinsOutput](#) (RGPIO_Type *base, uint32_t mask)

Do not use this function. It has been superceded by [RGPIO_PortSet](#).

Global [RGPIO_TogglePinsOutput](#) (RGPIO_Type *base, uint32_t mask)

Do not use this function. It has been superceded by [RGPIO_PortToggle](#).

Global [RGPIO_WritePinOutput](#) (RGPIO_Type *base, uint32_t pin, uint8_t output)

Do not use this function. It has been superceded by [RGPIO_PinWrite](#).

Global [SAI_RxGetDefaultConfig](#) (sai_config_t *config)

Do not use this function. It has been superceded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeftJustifiedConfig](#), [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

Global [SAI_RxInit](#) (I2S_Type *base, const sai_config_t *config)

Do not use this function. It has been superceded by [SAI_Init](#)

Global [SAI_RxSetFormat](#) (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Do not use this function. It has been superceded by [SAI_RxSetConfig](#)

Global [SAI_TransferRxSetFormat](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Do not use this function. It has been superceded by [SAI_TransferRxSetConfig](#)

Global [SAI_TransferTxSetFormat](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Do not use this function. It has been superceded by [SAI_TransferTxSetConfig](#)

Global [SAI_TxGetDefaultConfig](#) (sai_config_t *config)

Do not use this function. It has been superceded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeftJustifiedConfig](#), [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

Global [SAI_TxInit](#) (I2S_Type *base, const sai_config_t *config)

Do not use this function. It has been superceded by [SAI_Init](#)

Global [SAI_TxSetFormat](#) (I2S_Type *base, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)

Do not use this function. It has been superceded by [SAI_TxSetConfig](#)

Global [WM8904_SetMasterSlave](#) (wm8904_handle_t *handle, bool master)

DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY [WM8904_SeMasterClock](#)

Chapter 6 Clock Driver

Overview

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

The clock driver supports:

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

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](#)

Overview

- *Clock ip name array for LPSPI.*
• #define **EDMA_CLOCKS**
- *Clock ip name array for EDMA.*
• #define **ESAI_CLOCKS**
- *Clock ip name array for ESAI.*
• #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**(rsrc, base) (((uint32_t) (((base) >> 12U) << 10U) | (rsrc)))
LPCG TUPLE macors to map corresponding ip clock name, SCFW API 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 difinition 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, bool swGate, bool hwGate)
- *Config the LPCG cell for specific IP.*
void **CLOCK_SetLpcgGate** (volatile uint32_t *regBase, bool swGate, bool hwGate, uint32_t bitsMask)
- *Set LPCG gate for specific LPCG.*

Driver version

- #define **FSL_CLOCK_DRIVER_VERSION** (MAKE_VERSION(2, 4, 0))
CLOCK driver version 2.4.0.

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, 4, 0))

6.2.3 #define MU_CLOCKS

Value:

```
{
    \
    kCLOCK_M4_0_Mu0B, kCLOCK_M4_0_Mu0B, kCLOCK_M4_0_Mu0B, kCLOCK_M4_0_Mu0B, kCLOCK_M4_0_Mu0A0,
    kCLOCK_M4_0_Mu0A1, \
}
```

Macro Definition Documentation

```
kCLOCK_M4_0_Mu0A2, kCLOCK_M4_0_Mu0A3, kCLOCK_M4_0_Mu1A, kCLOCK_M4_1_Mu0B, kCLOCK_M4_1_Mu0B,
\
kCLOCK_M4_1_Mu0B, kCLOCK_M4_1_Mu0B, kCLOCK_M4_1_Mu0A0, kCLOCK_M4_1_Mu0A1, kCLOCK_M4_1_Mu0A2,
\
kCLOCK_M4_1_Mu0A3, kCLOCK_M4_1_Mu1A, kCLOCK_LSIO_Mu0A, kCLOCK_LSIO_Mu1A, kCLOCK_LSIO_Mu2A,
\
kCLOCK_LSIO_Mu3A, kCLOCK_LSIO_Mu4A, kCLOCK_LSIO_Mu5A, kCLOCK_LSIO_Mu6A, kCLOCK_LSIO_Mu7A,
\
kCLOCK_LSIO_Mu8A, kCLOCK_LSIO_Mu9A, kCLOCK_LSIO_Mu10A, kCLOCK_LSIO_Mu11A, kCLOCK_LSIO_Mu12A,
\
kCLOCK_LSIO_Mu13A, kCLOCK_LSIO_Mu5B, kCLOCK_LSIO_Mu6B, kCLOCK_LSIO_Mu7B, kCLOCK_LSIO_Mu8B,
\
kCLOCK_LSIO_Mu9B, kCLOCK_LSIO_Mu10B, kCLOCK_LSIO_Mu11B, kCLOCK_LSIO_Mu12B, kCLOCK_LSIO_Mu13B,
}
```

6.2.4 #define GPIO_CLOCKS

Value:

```
{
\
kCLOCK_IpInvalid, kCLOCK_IpInvalid, kCLOCK_IpInvalid, kCLOCK_IpInvalid, kCLOCK_IpInvalid,
kCLOCK_HSIO_Gpio, \
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_LSIO_Gpt0, kCLOCK_LSIO_Gpt1, kCLOCK_LSIO_Gpt2, kCLOCK_LSIO_Gpt3, \
    kCLOCK_LSIO_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:

```
{
    kCLOCK_M4_0_Intmux, kCLOCK_M4_1_Intmux, kCLOCK_IpInvalid, \
}
```

6.2.13 #define SAI_CLOCKS

Value:

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

6.2.14 #define SEMA42_CLOCKS

Value:

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

6.2.15 #define TPM_CLOCKS

Value:

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

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, \
}
```

6.2.19 #define EDMA_CLOCKS

Value:

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

6.2.20 #define ESAI_CLOCKS

Value:

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

Macro Definition Documentation

6.2.21 #define ISI_CLOCKS

Value:

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

6.2.22 #define MIPI_CSI2RX_CLOCKS

Value:

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

6.2.23 #define MIPI_DSI_HOST_CLOCKS

Value:

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

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.

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)&0x3FF-U))

6.2.31 #define NV (0U)

Enumeration Type Documentation

6.3.1 enum clock_ip_src_t

Enumerator

kCLOCK_IpSrcNone Clock is off.
kCLOCK_IpSrcDummy Clock option 1.

Function Documentation

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.

Function Documentation

6.4.1 void CLOCK_Init (sc_ipc_t ipc)

Parameters

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

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.

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

Function Documentation

<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>name</i>	Clock names defined in <code>clock_name_t</code>
-------------	--

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*, bool *swGate*, bool *hwGate*)

Parameters

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

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

Parameters

<i>regBase</i>	LPCG register base address.
<i>swGate</i>	Software clock gating. false: clock is gated; true: clock is enabled
<i>hwGate</i>	Hardware auto gating. false: disable the HW clock gate control; true: 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

SCU EVENT Driver

Overview

This driver is used for SCU firmware event monitoring and management.

Data Structures

- struct [sc_event_list_t](#)
SC Event list fields. [More...](#)
- struct [sc_event_handler_t](#)
SC Event handler structure. [More...](#)

Macros

- #define [SC_EVENT_HANDLER_MEM_POOL_SIZE](#) (200U)
The SC Event handler memory pool size in bytes.
- #define [SC_EVENT_USE_SYSCTR](#) (0)
The SYSCTR SC Event maybe used as wakeup source in low power applications.
- #define [SC_EVENT_LIST_OBJ](#)(type, field, list) (type)((uint32_t)(list) - (uint32_t)((&((type)0)->field))
Get SC Event list object structure pointer.

Typedefs

- typedef void(* [sc_event_callback_t](#))(uint32_t status, void *userData)
SC Event callback function.
- typedef [status_t](#)(* [sc_event_sema4_wait_t](#))(void *sem, uint32_t timeout)
SC Event Semaphore Wait function.
- typedef void(* [sc_event_sema4_post_t](#))(void *sem)
SC Event Semaphore Post function.

Enumerations

- enum [sc_event_t](#) {
 [kSCEvent_TempHigh](#) = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_TEMP, SC_IRQ_TEMP-

Overview

```
_HIGH),
kSCEvent_TempCpu0High,
kSCEvent_TempCpu1High,
kSCEvent_TempGpu0High,
kSCEvent_TempGpu1High,
kSCEvent_TempDrc0High,
kSCEvent_TempDrc1High,
kSCEvent_TempVpuHigh,
kSCEvent_TempPmic0High,
kSCEvent_TempPmic1High,
kSCEvent_TempLow = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_TEMP, SC_IRQ_TEMP-
_LOW),
kSCEvent_TempCpu0Low,
kSCEvent_TempCpu1Low,
kSCEvent_TempGpu0Low,
kSCEvent_TempGpu1Low,
kSCEvent_TempDrc0Low,
kSCEvent_TempDrc1Low,
kSCEvent_TempVpuLow = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_TEMP, SC_IRQ_TE-
MP_VPU_LOW),
kSCEvent_TempPmic0Low,
kSCEvent_TempPmic1Low,
kSCEvent_TempPmic2High,
kSCEvent_TempPmic2Low,
kSCEvent_Wdog = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_WDOG, SC_IRQ_WDOG),
kSCEvent_Rtc = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_RTC, SC_IRQ_RTC),
kSCEvent_Button = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_WAKE, SC_IRQ_BUTTO-
N),
kSCEvent_Pad = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_WAKE, SC_IRQ_PAD),
kSCEvent_User1 = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_WAKE, SC_IRQ_USR1),
kSCEvent_User2 = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_WAKE, SC_IRQ_USR2),
kSCEvent_BcPad,
kSCEvent_Rebooted,
kSCEvent_Reboot = SC_EVENT_TYPE_TUPLE(SC_IRQ_GROUP_REBOOT, SC_EVENT_IR-
Q_DUMMY) }
    SC Event type.
```

Functions

- void [SCEvent_Init](#) (uint8_t priority)
Initialize SC Event module.
- void [SCEvent_Deinit](#) (void)
Deinitialize SC Event.
- sc_event_handler_t [SCEvent_RegisterEventHandler](#) (sc_event_t event, sc_event_callback_t call-
back, void *userData)
Register SC Event handler.

- void [SCEvent_UnregisterEventHandler](#) (sc_event_handler_t handler)
Unregister SC Event handler.
- [status_t SCEvent_Config](#) (sc_event_t event, bool enable, uint32_t pt)
Configure SC Event.
- void [SCEvent_Process](#) (void)
SCEvent Process function.
- [status_t SCEvent_WaitEvent](#) (sc_event_sema4_wait_t wait, sc_event_sema4_post_t post, void *sem, uint32_t timeout)
Set Semaphore and Semaphore wait/post function in RTOS environment.

Driver version

- #define [FSL_SC_EVENT_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 0, 1))
Group interrupt driver version for SDK.

Data Structure Documentation

7.2.1 struct sc_event_list_t

Data Fields

- struct _sc_event_list * [prev](#)
previous list node
- struct _sc_event_list * [next](#)
next list node

7.2.2 struct _sc_event_handler

SC Event handler is a pointer to the SC Event handler instance.

Data Fields

- [sc_event_t](#) event
Event type.
- [sc_event_callback_t](#) callback
Event callback.
- void * [data](#)
User data.

7.2.2.0.0.1 Field Documentation

7.2.2.0.0.1.1 [sc_event_t](#) [sc_event_handler_t::event](#)

7.2.2.0.0.1.2 [sc_event_callback_t](#) [sc_event_handler_t::callback](#)

7.2.2.0.0.1.3 [void*](#) [sc_event_handler_t::data](#)

Enumeration Type Documentation

Macro Definition Documentation

7.3.1 `#define FSL_SC_EVENT_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

Version 2.0.1.

7.3.2 `#define SC_EVENT_HANDLER_MEM_POOL_SIZE (200U)`

7.3.3 `#define SC_EVENT_USE_SYSCTR (0)`

Such as in LPM module(lpm.c), the SYSCTR is used as wakeup source in tickless idle.

Typedef Documentation

7.4.1 `typedef void(* sc_event_callback_t)(uint32_t status, void *userData)`

The SC Event callback returns a status from the underlying layer. The status reflects the SC IRQ status got by `sc_irq_status` SCFW API. It's the bitmask of enabled IRQ in a group. Such as for `kSCEvent_Reboot/kSCEvent_Rebooted` event, if more than one partition reboot event is enabled, this status flag can be used to identify which partition the event comes from. Refer to SCFW API `sc_irq_status` for more details.

7.4.2 `typedef status_t(* sc_event_sema4_wait_t)(void *sem, uint32_t timeout)`

The Semaphore Wait/Post function is used in RTOS environment to pass Semaphore Wait/Post function into SC Event driver. Unused in baremetal environment. The task can wait the event by calling `SCEvent_WaitEvent` function.

7.4.3 `typedef void(* sc_event_sema4_post_t)(void *sem)`

The Semaphore Wait/Post function is used in RTOS environment to pass Semaphore Wait/Post function into SC Event driver. Unused in baremetal environment. The post function is called to give semaphore if event triggered in IRQ.

Enumeration Type Documentation

7.5.1 `enum sc_event_t`

Enumerator

kSCEvent_TempHigh Temp alarm interrupt.

kSCEvent_TempCpu0High CPU0 temp alarm interrupt.

kSCEvent_TempCpu1High CPU1 temp alarm interrupt.

kSCEvent_TempGpu0High GPU0 temp alarm interrupt.
kSCEvent_TempGpu1High GPU1 temp alarm interrupt.
kSCEvent_TempDrc0High DRC0 temp alarm interrupt.
kSCEvent_TempDrc1High DRC1 temp alarm interrupt.
kSCEvent_TempVpuHigh VPU temp alarm interrupt.
kSCEvent_TempPmic0High PMIC0 temp alarm interrupt.
kSCEvent_TempPmic1High PMIC1 temp alarm interrupt.
kSCEvent_TempLow Temp alarm interrupt.
kSCEvent_TempCpu0Low CPU0 temp alarm interrupt.
kSCEvent_TempCpu1Low CPU1 temp alarm interrupt.
kSCEvent_TempGpu0Low GPU0 temp alarm interrupt.
kSCEvent_TempGpu1Low GPU1 temp alarm interrupt.
kSCEvent_TempDrc0Low DRC0 temp alarm interrupt.
kSCEvent_TempDrc1Low DRC1 temp alarm interrupt.
kSCEvent_TempVpuLow VPU temp alarm interrupt.
kSCEvent_TempPmic0Low PMIC0 temp alarm interrupt.
kSCEvent_TempPmic1Low PMIC1 temp alarm interrupt.
kSCEvent_TempPmic2High PMIC2 temp alarm interrupt.
kSCEvent_TempPmic2Low PMIC2 temp alarm interrupt.
kSCEvent_Wdog Watchdog interrupt.
kSCEvent_Rtc RTC interrupt.
kSCEvent_Button Button interrupt.
kSCEvent_Pad Pad wakeup.
kSCEvent_User1 User defined 1.
kSCEvent_User2 User defined 2.
kSCEvent_BcPad Pad wakeup (broadcast to all partitions)
kSCEvent_Rebooted Partition reboot complete.
kSCEvent_Reboot Partition reboot starting.

Function Documentation

7.6.1 void SCEvent_Init (uint8_t *priority*)

This function enable the IPC MU interrupt, setup to listen the event comes from SCU/SCFW.

Parameters

<i>priority</i>	The SC IRQ interrupt priority. Once any SC IRQ enabled, the SCU will broadcast MU interrupt through the IPC MU to M4 cores.
-----------------	---

Function Documentation

Return values

<i>None.</i>	
--------------	--

7.6.2 void SCEvent_Deinit (void)

This function disable the IPC MU interrupt.

7.6.3 sc_event_handler_t SCEvent_RegisterEventHandler (sc_event_t event, sc_event_callback_t callback, void * userData)

This function register the handler for the specified event. When the event is enabled by SCEvent_Config and happens, the callback will be called from SCEvent_process. More than one callback can be added for an event. But the total registered events are limited to SC_EVENT_HANDLER_MEM_POOL_SIZE/(sizeof(struct _sc_event_handler)).

Parameters

<i>event</i>	The event to be registered with.
<i>callback</i>	The callback function. This function will be called from SCEvent_process.
<i>userData</i>	User defined callback function parameter.

Return values

<i>SC</i>	Event handler.
-----------	----------------

7.6.4 void SCEvent_UnregisterEventHandler (sc_event_handler_t handler)

This function unregister and delete the given event handler.

Parameters

<i>handler</i>	The SC Event handler to unregister.
----------------	-------------------------------------

Return values

<i>none.</i>	
--------------	--

7.6.5 **status_t SCEvent_Config (sc_event_t event, bool enable, uint32_t pt)**

This function enables/disables response for the event comes from SCU/SCFW.

Parameters

<i>event</i>	The event to be configured.
<i>enable</i>	Enable the event or not.
<i>pt</i>	Which partition the event comes from. Only applicable for partition reboot event: kSCEvent_Rebooted, kSCEvent_Reboot.

Return values

<i>kStatus_Success</i>	- Configure SC Event Successfully.
<i>kStatus_Fail</i>	- Failed to configure SC Event.

7.6.6 **void SCEvent_Process (void)**

This function is used to check SC event status and handle all the registered event handlers. The SCFW API call is a blocking call and used to check SC IRQ status, so this function should be executed in task context to make sure other event(interrupt) handled in time. In RTOS environment, application creates a task and wait for some events came using SCEvent_WaitEvent to run this function to handle the events. In baremetal environment, application need periodically call this function to make sure the pending events handled.

7.6.7 **status_t SCEvent_WaitEvent (sc_event_sema4_wait_t wait, sc_event_sema4_post_t post, void * sem, uint32_t timeout)**

The Semaphore Wait/Post function is used in RTOS environment to pass Semaphore Wait/Post function into SC Event driver. Unused in baremetal environment. The SCEvent_process will wait the Semaphore registered by SCEvent_SetWaitPostFunc until the event comes, and give the Semaphore, then it will handle the event and call the event handler's callback.

Function Documentation

Parameters

<i>wait</i>	Semaphore wait function.
<i>post</i>	Semaphore post function.
<i>sem</i>	Semaphore.
<i>timeout</i>	timeout.

Return values

<i>kStatus_Success</i>	if succeed, kStatus_Timeout if timeout.
------------------------	---

Chapter 8

ASMC: Auxiliary System Mode Controller Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the Auxiliary System Mode Controller (ASMC) module of MCUXpresso SDK devices. The ASMC module is responsible for sequencing Core 1 into and out of all low-power Stop and Run modes. The ASMC (Auxiliary System Mode Control) provides status and control registers for Core 1.

API functions are provided for configuring the system working in a dedicated power mode. For different power modes, the function `ASMC_SetPowerModexxx` accepts different parameters. System power mode state transitions are not available for between power modes. For details about available transitions, see the Power mode transitions section in the SoC reference manual.

Typical use case

8.2.1 Enter wait or stop modes

The ASMC driver provides APIs to set the MCU to different wait modes and stop modes. At the same time, there are pre function and post function for the modes setting. The pre function and post function are used to:

1. Disable/Enable the interrupt through PRIMASK. In practise, there is such a scenario: the application sets the wakeup interrupt and calls SMC function [ASMC_SetPowerModeStop](#) to set MCU to STOP mode, but the wakeup interrupt happens so quickly that the ISR completes before the function [ASMC_SetPowerModeStop](#). As a result, the MCU enters STOP mode and is never waken up by the interrupt. In this case, the application could first disable the interrupt through PRIMASK, then set the wakeup interrupt and enter STOP mode. After wakeup, the first thing you need to do is enable the interrupt through PRIMASK. The MCU can still wakeup when you disable interrupt through PRIMASK. The pre and post functions handle the PRIMASK inside.

```
ASMC_PreEnterStopModes();  
  
/* Enable the wakeup interrupt here. */  
  
ASMC_SetPowerModeStop(ASMC, kASMC_PartialStop);  
  
ASMC_PostExitStopModes();
```

Typical use case

Enumerations

- enum `_asmc_system_reset_status_flags` {
 `kASMC_LowLeakageWakeupResetFlag` = `ASMC_SRS_WAKEUP_MASK`,
 `kASMC_WatchdogResetFlag` = `ASMC_SRS_WDOG1_MASK`,
 `kASMC_ChipResetNotPORFlag` = `ASMC_SRS_RES_MASK`,
 `kASMC_PowerOnResetFlag` = `ASMC_SRS_POR_MASK`,
 `kASMC_Core1LockupResetFlag` = `ASMC_SRS_LOCKUP_MASK`,
 `kASMC_SoftwareResetFlag` = `ASMC_SRS_SW_MASK`,
 `kASMC_StopModeAcknowledgeErrorResetFlag` }

System Reset Status.

- enum `asmc_power_mode_protection_t` {
 `kASMC_AllowPowerModeVlls` = `ASMC_PMPROT_AVLLS_MASK`,
 `kASMC_AllowPowerModeLls` = `ASMC_PMPROT_ALLS_MASK`,
 `kASMC_AllowPowerModeVlp` = `ASMC_PMPROT_AVLP_MASK`,
 `kASMC_AllowPowerModeAll` }

Power Modes Protection.

- enum `asmc_power_state_t` {
 `kASMC_PowerStateRun` = `0x01U << 0U`,
 `kASMC_PowerStateStop` = `0x01U << 1U`,
 `kASMC_PowerStateVlpr` = `0x01U << 2U`,
 `kASMC_PowerStateVlpw` = `0x01U << 3U`,
 `kASMC_PowerStateVlps` = `0x01U << 4U`,
 `kASMC_PowerStateLls` = `0x01U << 5U`,
 `kASMC_PowerStateVlls` = `0x01U << 6U` }

Power Modes in PMSTAT.

- enum `asmc_run_mode_t` {
 `kASMC_RunNormal` = `0U`,
 `kASMC_RunVlpr` = `2U` }

Run mode definition.

- enum `asmc_stop_mode_t` {
 `kASMC_StopNormal` = `0U`,
 `kASMC_StopVlps` = `2U`,
 `kASMC_StopLls` = `3U`,
 `kASMC_StopVlls` = `4U` }

Stop mode definition.

- enum `asmc_partial_stop_option_t` {
 `kASMC_PartialStop` = `0U`,
 `kASMC_PartialStop1` = `1U`,
 `kASMC_PartialStop2` = `2U` }

Partial STOP option.

Driver version

- #define `FSL_ASMC_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)
 ASMC driver version 2.0.1.

Auxiliary system mode controller APIs

- static uint32_t [ASMC_GetSystemResetStatusFlags](#) (ASMC_Type *base)
Gets ASMC system reset status flags.
- static void [ASMC_SetPowerModeProtection](#) (ASMC_Type *base, uint8_t allowedModes)
Configures all power mode protection settings.
- static asmc_power_state_t [ASMC_GetPowerModeState](#) (ASMC_Type *base)
Gets the current power mode status.
- static void [ASMC_PreEnterStopModes](#) (void)
Prepare to enter stop modes.
- static void [ASMC_PostExitStopModes](#) (void)
Recovering after wake up from stop modes.
- static void [ASMC_PreEnterWaitModes](#) (void)
Prepare to enter wait modes.
- static void [ASMC_PostExitWaitModes](#) (void)
Recovering after wake up from stop modes.
- status_t [ASMC_SetPowerModeRun](#) (ASMC_Type *base)
Configure the system to RUN power mode.
- status_t [ASMC_SetPowerModeWait](#) (ASMC_Type *base)
Configure the system to WAIT power mode.
- status_t [ASMC_SetPowerModeStop](#) (ASMC_Type *base, asmc_partial_stop_option_t option)
Configure the system to Stop power mode.
- status_t [ASMC_SetPowerModeVlpr](#) (ASMC_Type *base)
Configure the system to VLPR power mode.
- status_t [ASMC_SetPowerModeVlpw](#) (ASMC_Type *base)
Configure the system to VLPW power mode.
- status_t [ASMC_SetPowerModeVlps](#) (ASMC_Type *base)
Configure the system to VLPS power mode.
- status_t [ASMC_SetPowerModeLls](#) (ASMC_Type *base)
Configure the system to LLS power mode.
- status_t [ASMC_SetPowerModeVlls](#) (ASMC_Type *base)
Configure the system to VLLS power mode.

Macro Definition Documentation

8.3.1 #define FSL_ASMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Enumeration Type Documentation

8.4.1 enum _asmc_system_reset_status_flags

Enumerator

- kASMC_LowLeakageWakeupResetFlag*** Reset caused by LLWU module wakeup source.
- kASMC_WatchdogResetFlag*** Reset caused by watchdog timeout.
- kASMC_ChipResetNotPORFlag*** Chip Reset caused by a source other than POR occurred.
- kASMC_PowerOnResetFlag*** Reset caused by POR.
- kASMC_Core1LockupResetFlag*** Reset caused by core LOCKUP event.
- kASMC_SoftwareResetFlag*** Reset caused by software setting of SYSRESETREQ bit.

Enumeration Type Documentation

kASMC_StopModeAcknowledgeErrorResetFlag Reset caused by peripheral failure to acknowledge attempt to enter stop mode.

8.4.2 enum asmc_power_mode_protection_t

Enumerator

kASMC_AllowPowerModeVlls Allow Very-Low-Leakage Stop Mode.
kASMC_AllowPowerModeLls Allow Low-Leakage Stop Mode.
kASMC_AllowPowerModeVlp Allow Very-Low-Power Mode.
kASMC_AllowPowerModeAll Allow all power mode.

8.4.3 enum asmc_power_state_t

Enumerator

kASMC_PowerStateRun 0000_0001 - Current power mode is RUN
kASMC_PowerStateStop 0000_0010 - Current power mode is STOP
kASMC_PowerStateVlpr 0000_0100 - Current power mode is VLPR
kASMC_PowerStateVlpw 0000_1000 - Current power mode is VLPW
kASMC_PowerStateVlps 0001_0000 - Current power mode is VLPS
kASMC_PowerStateLls 0010_0000 - Current power mode is LLS
kASMC_PowerStateVlls 0100_0000 - Current power mode is VLLS

8.4.4 enum asmc_run_mode_t

Enumerator

kASMC_RunNormal normal RUN mode.
kASMC_RunVlpr Very-Low-Power RUN mode.

8.4.5 enum asmc_stop_mode_t

Enumerator

kASMC_StopNormal Normal STOP mode.
kASMC_StopVlps Very-Low-Power STOP mode.
kASMC_StopLls Low-Leakage Stop mode.
kASMC_StopVlls Very-Low-Leakage Stop mode.

8.4.6 enum asmc_partial_stop_option_t

Enumerator

kASMC_PartialStop STOP - Normal Stop mode.

kASMC_PartialStop1 Partial Stop with both system and bus clocks disabled.

kASMC_PartialStop2 Partial Stop with system clock disabled and bus clock enabled.

Function Documentation

8.5.1 static uint32_t ASMC_GetSystemResetStatusFlags (ASMC_Type * *base*) [inline], [static]

This function gets all ASMC system reset status flags. The flag indicates the source of the most recent MCU reset. The reset state of these bits depends on what caused the MCU to reset. The flags are returned as the logical OR value of the enumerators [_asmc_system_reset_status_flags](#). To check for a specific status, compare the return value with enumerators in the [_asmc_system_reset_status_flags](#). For example, to check whether the reset is caused by POR:

```
*  if (kASMC_PowerOnResetFlag &
*    ASMC_GetSystemResetStatusFlags(ASMC))
*  {
*    ...
*  }
*
```

Parameters

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

Returns

ASMC system reset status flags. See "[_asmc_system_reset_status_flags](#)".

8.5.2 static void ASMC_SetPowerModeProtection (ASMC_Type * *base*, uint8_t *allowedModes*) [inline], [static]

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the [smc_power_mode_protection_t](#). This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map, for example, to allow LLS and VLLS, use [ASMC_SetPowerModeProtection\(kASMC_AllowPowerModeVlls | kASMC_AllowPowerModeVlps\)](#). To allow all modes, use [ASMC_SetPowerModeProtection\(kASMC_AllowPowerModeAll\)](#).

Function Documentation

Parameters

<i>base</i>	ASMC peripheral base address.
<i>allowedModes</i>	Bitmap of the allowed power modes. See "asmc_power_mode_protection_t".

8.5.3 static asmc_power_state_t ASMC_GetPowerModeState (ASMC_Type * *base*) [inline], [static]

This function returns the current power mode stat. Once application switches the power mode, it should always check the stat to check whether it runs into the specified mode or not. An application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the `asmc_power_state_t` for information about the power stat.

Parameters

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

Returns

Current power mode status.

8.5.4 static void ASMC_PreEnterStopModes (void) [inline], [static]

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

8.5.5 static void ASMC_PostExitStopModes (void) [inline], [static]

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used together with [ASMC_PreEnterStopModes](#).

8.5.6 static void ASMC_PreEnterWaitModes (void) [inline], [static]

This function should be called before entering WAIT/VLPW modes..

8.5.7 static void ASMC_PostExitWaitModes (void) [inline], [static]

This function should be called after wake up from WAIT/VLPW modes. It is used together with [ASMC_PreEnterWaitModes](#).

8.5.8 `status_t ASMC_SetPowerModeRun (ASMC_Type * base)`

Function Documentation

Parameters

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

Returns

ASMC configuration error code.

8.5.9 `status_t ASMC_SetPowerModeWait (ASMC_Type * base)`

Parameters

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

Returns

ASMC configuration error code.

8.5.10 `status_t ASMC_SetPowerModeStop (ASMC_Type * base, asmc_partial_stop_option_t option)`

Parameters

<i>base</i>	ASMC peripheral base address.
<i>option</i>	Partial Stop mode option.

Returns

ASMC configuration error code.

8.5.11 `status_t ASMC_SetPowerModeVlpr (ASMC_Type * base)`

Parameters

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

Returns

ASMC configuration error code.

8.5.12 **status_t** **ASMC_SetPowerModeVlpw** (**ASMC_Type** * *base*)

Parameters

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

Returns

ASMC configuration error code.

8.5.13 **status_t** **ASMC_SetPowerModeVlps** (**ASMC_Type** * *base*)

Parameters

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

Returns

ASMC configuration error code.

8.5.14 **status_t** **ASMC_SetPowerModeLls** (**ASMC_Type** * *base*)

Parameters

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

Returns

ASMC configuration error code.

8.5.15 **status_t** **ASMC_SetPowerModeVlls** (**ASMC_Type** * *base*)

Function Documentation

Parameters

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

Returns

ASMC configuration error code.

Chapter 9

CACHE: LMEM CACHE Memory Controller

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.

Function groups

9.1.1 L1 CACHE Operation

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.



Function groups

Chapter 10 Common Driver

Overview

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

Macros

- #define **MAKE_STATUS**(group, code) (((group)*100) + (code))
Construct a status code value from a group and code number.
- #define **MAKE_VERSION**(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
Construct the version number for drivers.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_NONE** 0U
No debug console.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_UART** 1U
Debug console based on UART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_LPUART** 2U
Debug console based on LPUART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_LPSCI** 3U
Debug console based on LPSCI.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_USBCDC** 4U
Debug console based on USBCDC.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM** 5U
Debug console based on FLEXCOMM.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_IUART** 6U
Debug console based on i.MX UART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_VUSART** 7U
Debug console based on LPC_VUSART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART** 8U
Debug console based on LPC_USART.
- #define **DEBUG_CONSOLE_DEVICE_TYPE_SWO** 9U
Debug console based on SWO.
- #define **ARRAY_SIZE**(x) (sizeof(x) / sizeof((x)[0]))
Computes the number of elements in an array.

Typedefs

- typedef int32_t **status_t**
Type used for all status and error return values.

Enumerations

- enum _status_groups {
 - kStatusGroup_Generic = 0,
 - kStatusGroup_FLASH = 1,
 - kStatusGroup_LPSPI = 4,
 - kStatusGroup_FLEXIO_SPI = 5,
 - kStatusGroup_DSPI = 6,
 - kStatusGroup_FLEXIO_UART = 7,
 - kStatusGroup_FLEXIO_I2C = 8,
 - kStatusGroup_LPI2C = 9,
 - kStatusGroup_UART = 10,
 - kStatusGroup_I2C = 11,
 - kStatusGroup_LPSCI = 12,
 - kStatusGroup_LPUART = 13,
 - kStatusGroup_SPI = 14,
 - kStatusGroup_XRDC = 15,
 - kStatusGroup_SEMA42 = 16,
 - kStatusGroup_SDHC = 17,
 - kStatusGroup_SDMMC = 18,
 - kStatusGroup_SAI = 19,
 - kStatusGroup_MCG = 20,
 - kStatusGroup_SCG = 21,
 - kStatusGroup_SDSPI = 22,
 - kStatusGroup_FLEXIO_I2S = 23,
 - kStatusGroup_FLEXIO_MCULCD = 24,
 - kStatusGroup_FLASHIAP = 25,
 - kStatusGroup_FLEXCOMM_I2C = 26,
 - kStatusGroup_I2S = 27,
 - kStatusGroup_IUART = 28,
 - kStatusGroup_CSI = 29,
 - kStatusGroup_MIPI_DSI = 30,
 - kStatusGroup_SDRAMC = 35,
 - kStatusGroup_POWER = 39,
 - kStatusGroup_ENET = 40,
 - kStatusGroup_PHY = 41,
 - kStatusGroup_TRGMUX = 42,
 - kStatusGroup_SMARTCARD = 43,
 - kStatusGroup_LMEM = 44,
 - kStatusGroup_QSPI = 45,
 - kStatusGroup_DMA = 50,
 - kStatusGroup_EDMA = 51,
 - kStatusGroup_DMAMGR = 52,
 - kStatusGroup_FLEXCAN = 53,
 - kStatusGroup_LTC = 54,
 - kStatusGroup_FLEXIO_CAMERA = 55,
 - kStatusGroup_LPC_SPI = 56,
 - kStatusGroup_LPC_USMCA = 57,
 - kStatusGroup_DMIC = 58,
 - kStatusGroup_SDIF = 59,

```
kStatusGroup_SDIO SLV = 151 }
```

Status group numbers.

- enum {
 kStatus_Success = MAKE_STATUS(kStatusGroup_Generic, 0),
 kStatus_Fail = MAKE_STATUS(kStatusGroup_Generic, 1),
 kStatus_ReadOnly = MAKE_STATUS(kStatusGroup_Generic, 2),
 kStatus_OutOfRange = MAKE_STATUS(kStatusGroup_Generic, 3),
 kStatus_InvalidArgument = MAKE_STATUS(kStatusGroup_Generic, 4),
 kStatus_Timeout = MAKE_STATUS(kStatusGroup_Generic, 5),
 kStatus_NoTransferInProgress = MAKE_STATUS(kStatusGroup_Generic, 6) }

Generic status return codes.

Functions

- static [status_t EnableIRQ](#) (IRQn_Type interrupt)
 Enable specific interrupt.
- static [status_t DisableIRQ](#) (IRQn_Type interrupt)
 Disable specific interrupt.
- static [uint32_t DisableGlobalIRQ](#) (void)
 Disable the global IRQ.
- static void [EnableGlobalIRQ](#) (uint32_t primask)
 Enable the global IRQ.
- void * [SDK_Malloc](#) (size_t size, size_t alignbytes)
 Allocate memory with given alignment and aligned size.
- void [SDK_Free](#) (void *ptr)
 Free memory.
- void [SDK_DelayAtLeastUs](#) (uint32_t delay_us, uint32_t coreClock_Hz)
 Delay at least for some time.

Driver version

- #define [FSL_COMMON_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 2, 4))
 common driver version 2.2.4.

Min/max macros

- #define [MIN](#)(a, b) (((a) < (b)) ? (a) : (b))
- #define [MAX](#)(a, b) (((a) > (b)) ? (a) : (b))

UINT16_MAX/UINT32_MAX value

- #define [UINT16_MAX](#) ((uint16_t)-1)
- #define [UINT32_MAX](#) ((uint32_t)-1)

Timer utilities

- #define [USEC_TO_COUNT](#)(us, clockFreqInHz) (uint64_t)(((uint64_t)(us) * (clockFreqInHz)) / 1000000U)
 Macro to convert a microsecond period to raw count value.

Overview

- #define **COUNT_TO_USEC**(count, clockFreqInHz) (uint64_t)((uint64_t)(count) * 1000000U / (clockFreqInHz))
Macro to convert a raw count value to microsecond.
- #define **MSEC_TO_COUNT**(ms, clockFreqInHz) (uint64_t)((uint64_t)(ms) * (clockFreqInHz) / 1000U)
Macro to convert a millisecond period to raw count value.
- #define **COUNT_TO_MSEC**(count, clockFreqInHz) (uint64_t)((uint64_t)(count) * 1000U / (clockFreqInHz))
Macro to convert a raw count value to millisecond.

Alignment variable definition macros

- #define **SDK_ALIGN**(var, alignbytes) var
- #define **SDK_L1DCACHE_ALIGN**(var) var
- #define **SDK_SIZEALIGN**(var, alignbytes) ((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(~(unsigned int)((alignbytes)-1U)))
Macro to change a value to a given size aligned value.

Non-cacheable region definition macros

- #define **AT_NONCACHEABLE_SECTION**(var) var
- #define **AT_NONCACHEABLE_SECTION_ALIGN**(var, alignbytes) var
- #define **AT_NONCACHEABLE_SECTION_INIT**(var) var
- #define **AT_NONCACHEABLE_SECTION_ALIGN_INIT**(var, alignbytes) var

Suppress fallthrough warning macro

- #define **SUPPRESS_FALL_THROUGH_WARNING**()

Macro Definition Documentation

10.2.1 **#define** MAKE_STATUS(*group*, *code*) (((group)*100) + (code)))

10.2.2 **#define** MAKE_VERSION(*major*, *minor*, *bugfix*) (((major) << 16) | ((minor) << 8) | (bugfix))

10.2.3 **#define** FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 2, 4))

10.2.4 **#define** DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

10.2.5 **#define** DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

10.2.6 **#define** DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

10.2.7 **#define** DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

10.2.8 **#define** DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

10.2.9 **#define** DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

10.2.10 **#define** DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

10.2.11 **#define** DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

10.2.12 **#define** DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

10.2.13 **#define** DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

10.2.14 **#define** ARRAY_SIZE(*x*) (sizeof(x) / sizeof((x)[0]))

Typedef Documentation

10.3.1 **typedef** int32_t status_t

Enumeration Type Documentation

10.4.1 **enum** _status_groups

Enumerator

kStatusGroup_Generic Group number for generic status codes.

kStatusGroup_FLASH Group number for FLASH status codes.

kStatusGroup_LPSPI Group number for LPSPI status codes.

kStatusGroup_FLEXIO_SPI Group number for FLEXIO SPI status codes.

kStatusGroup_DSPI Group number for DSPI status codes.

kStatusGroup_FLEXIO_UART Group number for FLEXIO UART status codes.

kStatusGroup_FLEXIO_I2C Group number for FLEXIO I2C status codes.

kStatusGroup_LPI2C Group number for LPI2C status codes.

kStatusGroup_UART Group number for UART status codes.

kStatusGroup_I2C Group number for I2C status codes.

kStatusGroup_LPSCI Group number for LPSCI status codes.

kStatusGroup_LPUART Group number for LPUART status codes.

kStatusGroup_SPI Group number for SPI status code.

kStatusGroup_XRDC Group number for XRDC status code.

kStatusGroup_SEMA42 Group number for SEMA42 status code.

kStatusGroup_SDHC Group number for SDHC status code.

kStatusGroup_SDMMC Group number for SDMMC status code.

kStatusGroup_SAI Group number for SAI status code.

kStatusGroup_MCG Group number for MCG status codes.

kStatusGroup_SCG Group number for SCG status codes.

kStatusGroup_SDSPI Group number for SDSPI status codes.

kStatusGroup_FLEXIO_I2S Group number for FLEXIO I2S status codes.

kStatusGroup_FLEXIO_MCULCD Group number for FLEXIO LCD status codes.

kStatusGroup_FLASHIAP Group number for FLASHIAP status codes.

kStatusGroup_FLEXCOMM_I2C Group number for FLEXCOMM I2C status codes.

kStatusGroup_I2S Group number for I2S status codes.

kStatusGroup_IUART Group number for IUART status codes.

kStatusGroup_CSI Group number for CSI status codes.

kStatusGroup_MIPI_DSI Group number for MIPI DSI status codes.

kStatusGroup_SDRAMC Group number for SDRAMC status codes.

kStatusGroup_POWER Group number for POWER status codes.

kStatusGroup_ENET Group number for ENET status codes.

kStatusGroup_PHY Group number for PHY status codes.

kStatusGroup_TRGMUX Group number for TRGMUX status codes.

kStatusGroup_SMARTCARD Group number for SMARTCARD status codes.

kStatusGroup_LMEM Group number for LMEM status codes.

kStatusGroup_QSPI Group number for QSPI status codes.

kStatusGroup_DMA Group number for DMA status codes.

kStatusGroup_EDMA Group number for EDMA status codes.

kStatusGroup_DMAMGR Group number for DMAMGR status codes.

kStatusGroup_FLEXCAN Group number for FlexCAN status codes.

kStatusGroup_LTC Group number for LTC status codes.

kStatusGroup_FLEXIO_CAMERA Group number for FLEXIO CAMERA status codes.

kStatusGroup_LPC_SPI Group number for LPC_SPI status codes.

kStatusGroup_LPC_USART Group number for LPC_USART status codes.

kStatusGroup_DMIC Group number for DMIC status codes.

kStatusGroup_SDIF Group number for SDIF status codes.
kStatusGroup_SPIFI Group number for SPIFI status codes.
kStatusGroup_OTP Group number for OTP status codes.
kStatusGroup_MCAN Group number for MCAN status codes.
kStatusGroup_CAAM Group number for CAAM status codes.
kStatusGroup_ECSPi Group number for ECSPi status codes.
kStatusGroup_USDHC Group number for USDHC status codes.
kStatusGroup_LPC_I2C Group number for LPC_I2C status codes.
kStatusGroup_DCP Group number for DCP status codes.
kStatusGroup_MSCAN Group number for MSCAN status codes.
kStatusGroup_ESAI Group number for ESAI status codes.
kStatusGroup_FLEXSPI Group number for FLEXSPI status codes.
kStatusGroup_MMDC Group number for MMDC status codes.
kStatusGroup_PDM Group number for MIC status codes.
kStatusGroup_SDMA Group number for SDMA status codes.
kStatusGroup_ICS Group number for ICS status codes.
kStatusGroup_SPDIF Group number for SPDIF status codes.
kStatusGroup_LPC_MINISPI Group number for LPC_MINISPI status codes.
kStatusGroup_HASHCRYPT Group number for Hashcrypt status codes.
kStatusGroup_LPC_SPI_SSP Group number for LPC_SPI_SSP status codes.
kStatusGroup_I3C Group number for I3C status codes.
kStatusGroup_LPC_I2C_1 Group number for LPC_I2C_1 status codes.
kStatusGroup_NOTIFIER Group number for NOTIFIER status codes.
kStatusGroup_DebugConsole Group number for debug console status codes.
kStatusGroup_SEMC Group number for SEMC status codes.
kStatusGroup_ApplicationRangeStart Starting number for application groups.
kStatusGroup_IAP Group number for IAP status codes.
kStatusGroup_SFA Group number for SFA status codes.
kStatusGroup_SPC Group number for SPC status codes.
kStatusGroup_PUF Group number for PUF status codes.
kStatusGroup_HAL_GPIO Group number for HAL GPIO status codes.
kStatusGroup_HAL_UART Group number for HAL UART status codes.
kStatusGroup_HAL_TIMER Group number for HAL TIMER status codes.
kStatusGroup_HAL_SPI Group number for HAL SPI status codes.
kStatusGroup_HAL_I2C Group number for HAL I2C status codes.
kStatusGroup_HAL_FLASH Group number for HAL FLASH status codes.
kStatusGroup_HAL_PWM Group number for HAL PWM status codes.
kStatusGroup_HAL_RNG Group number for HAL RNG status codes.
kStatusGroup_TIMERMANAGER Group number for TiMER MANAGER status codes.
kStatusGroup_SERIALMANAGER Group number for SERIAL MANAGER status codes.
kStatusGroup_LED Group number for LED status codes.
kStatusGroup_BUTTON Group number for BUTTON status codes.
kStatusGroup_EXTERN_EEPROM Group number for EXTERN EEPROM status codes.
kStatusGroup_SHELL Group number for SHELL status codes.
kStatusGroup_MEM_MANAGER Group number for MEM MANAGER status codes.

Function Documentation

kStatusGroup_LIST Group number for List status codes.
kStatusGroup_OSA Group number for OSA status codes.
kStatusGroup_COMMON_TASK Group number for Common task status codes.
kStatusGroup_MSG Group number for messaging status codes.
kStatusGroup_SDK_OCOTP Group number for OCOTP status codes.
kStatusGroup_SDK_FLEXSPINOR Group number for FLEXSPINOR status codes.
kStatusGroup_CODEEC Group number for codec status codes.
kStatusGroup_ASRC Group number for codec status ASRC.
kStatusGroup_OTFAD Group number for codec status codes.
kStatusGroup_SDIOSLV Group number for SDIOSLV status codes.

10.4.2 anonymous enum

Enumerator

kStatus_Success Generic status for Success.
kStatus_Fail Generic status for Fail.
kStatus_ReadOnly Generic status for read only failure.
kStatus_OutOfRange Generic status for out of range access.
kStatus_InvalidArgument Generic status for invalid argument check.
kStatus_Timeout Generic status for timeout.
kStatus_NoTransferInProgress Generic status for no transfer in progress.

Function Documentation

10.5.1 static status_t EnableIRQ (IRQn_Type *interrupt*) [inline], [static]

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

<i>interrupt</i>	The IRQ number.
------------------	-----------------

Return values

<i>kStatus_Success</i>	Interrupt enabled successfully
<i>kStatus_Fail</i>	Failed to enable the interrupt

10.5.2 static status_t DisableIRQ (IRQn_Type *interrupt*) [inline], [static]

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS.

Parameters

<i>interrupt</i>	The IRQ number.
------------------	-----------------

Return values

<i>kStatus_Success</i>	Interrupt disabled successfully
<i>kStatus_Fail</i>	Failed to disable the interrupt

10.5.3 static uint32_t DisableGlobalIRQ (void) [inline], [static]

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the [EnableGlobalIRQ\(\)](#).

Returns

Current primask value.

10.5.4 static void EnableGlobalIRQ (uint32_t *primask*) [inline], [static]

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask. User is required to use the [EnableGlobalIRQ\(\)](#) and [DisableGlobalIRQ\(\)](#) in pair.

Function Documentation

Parameters

<i>primask</i>	value of primask register to be restored. The primask value is supposed to be provided by the DisableGlobalIRQ() .
----------------	--

10.5.5 void* SDK_Malloc (size_t size, size_t alignbytes)

This is provided to support the dynamically allocated memory used in cache-able region.

Parameters

<i>size</i>	The length required to malloc.
<i>alignbytes</i>	The alignment size.

Return values

<i>The</i>	allocated memory.
------------	-------------------

10.5.6 void SDK_Free (void * ptr)

Parameters

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

10.5.7 void SDK_DelayAtLeastUs (uint32_t delay_us, uint32_t coreClock_Hz)

Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

<i>delay_us</i>	Delay time in unit of microsecond.
<i>coreClock_Hz</i>	Core clock frequency with Hz.

Chapter 11

EDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

Overview

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

Typical use case

11.2.1 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_memorymap_callback](#))(uint32_t addr)
Memory map function callback for DMA.

Enumerations

- enum [edma_transfer_size_t](#) {
 [kEDMA_TransferSize1Bytes](#) = 0x0U,
 [kEDMA_TransferSize2Bytes](#) = 0x1U,
 [kEDMA_TransferSize4Bytes](#) = 0x2U,
 [kEDMA_TransferSize8Bytes](#) = 0x3U,
 [kEDMA_TransferSize16Bytes](#) = 0x4U,
 [kEDMA_TransferSize32Bytes](#) = 0x5U,
}

Typical use case

```
kEDMA_TransferSize64Bytes = 0x6U }
```

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 {
 - kEDMA_DoneFlag = 0x1U,
 - kEDMA_ErrorFlag = 0x2U,
 - kEDMA_InterruptFlag = 0x4U }*eDMA channel status flags, _edma_channel_status_flags*
 - enum {
 - 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, _edma_error_status_flags*
 - enum {
 - kEDMA_AttributeOutput = DMA_CH_SBR_ATTR_MASK,
 - kEDMA_PrivilegedAccessLevel = DMA_CH_SBR_PAL_MASK,
 - kEDMA_MasterId }*eDMA channel system bus information, _edma_channel_sys_bus_info*
 - 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 {
 - kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),
 - kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }*eDMA transfer status, _edma_transfer_status*

Driver version

- #define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))
eDMA driver version

eDMA initialization and de-initialization

- void EDMA_Init (DMA_Type *base, const edma_config_t *config)

Typical use case

- Initializes the eDMA peripheral.*
- void [EDMA_Deinit](#) (DMA_Type *base)
Deinitializes the eDMA peripheral.
- void [EDMA_InstallTCD](#) (DMA_Type *base, uint32_t channel, [edma_tcd_t](#) *tcd)
Push content of TCD structure into hardware TCD register.
- 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_InstallTCDMemory` (`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.

Data Structure Documentation

- void [EDMA_PrepareTransferConfig](#) ([edma_transfer_config_t](#) *config, void *srcAddr, uint32_t srcWidth, int16_t srcOffset, void *destAddr, uint32_t destWidth, int16_t destOffset, uint32_t bytesEachRequest, uint32_t transferBytes)
Prepares the eDMA transfer structure configurations.
- void [EDMA_PrepareTransfer](#) ([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.
- 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.

Data Structure Documentation

11.3.1 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.

11.3.1.0.0.2 Field Documentation

11.3.1.0.0.2.1 bool edma_config_t::enableMasterIdReplication

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

11.3.1.0.0.2.2 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.

11.3.1.0.0.2.3 bool edma_config_t::enableDebugMode

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

11.3.1.0.0.2.4 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.

11.3.2 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

11.3.2.0.0.3 Field Documentation

11.3.2.0.0.3.1 `uint32_t edma_transfer_config_t::srcAddr`

11.3.2.0.0.3.2 `uint32_t edma_transfer_config_t::destAddr`

11.3.2.0.0.3.3 `edma_transfer_size_t edma_transfer_config_t::srcTransferSize`

11.3.2.0.0.3.4 `edma_transfer_size_t edma_transfer_config_t::destTransferSize`

11.3.2.0.0.3.5 `int16_t edma_transfer_config_t::srcOffset`

11.3.2.0.0.3.6 `int16_t edma_transfer_config_t::destOffset`

11.3.2.0.0.3.7 `uint32_t edma_transfer_config_t::majorLoopCounts`

11.3.3 `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.

11.3.4 `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.

11.3.4.0.0.4 Field Documentation

11.3.4.0.0.4.1 `bool edma_minor_offset_config_t::enableSrcMinorOffset`

11.3.4.0.0.4.2 `bool edma_minor_offset_config_t::enableDestMinorOffset`

11.3.4.0.0.4.3 `uint32_t edma_minor_offset_config_t::minorOffset`

11.3.5 `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

11.3.5.0.0.5 Field Documentation

11.3.5.0.0.5.1 `__IO uint16_t edma_tcd_t::CITER`

11.3.5.0.0.5.2 `__IO uint16_t edma_tcd_t::BITER`

11.3.6 `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_memorymap_callback memoryCallback`
Callback function for memory map convert in complex system.

11.3.6.0.0.6 Field Documentation**11.3.6.0.0.6.1** `edma_callback edma_handle_t::callback`**11.3.6.0.0.6.2** `void* edma_handle_t::userData`**11.3.6.0.0.6.3** `DMA_Type* edma_handle_t::base`**11.3.6.0.0.6.4** `edma_tcd_t* edma_handle_t::tcdPool`**11.3.6.0.0.6.5** `uint8_t edma_handle_t::channel`**11.3.6.0.0.6.6** `volatile int8_t edma_handle_t::header`**11.3.6.0.0.6.7** `volatile int8_t edma_handle_t::tail`**11.3.6.0.0.6.8** `volatile int8_t edma_handle_t::tcdUsed`**11.3.6.0.0.6.9** `volatile int8_t edma_handle_t::tcdSize`**11.3.6.0.0.6.10** `uint8_t edma_handle_t::flags`**Macro Definition Documentation****11.4.1** `#define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))`

Version 2.2.2.

Typedef Documentation**11.5.1** `typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)`**Enumeration Type Documentation****11.6.1** `enum edma_transfer_size_t`

Enumerator

kEDMA_TransferSize1Bytes Source/Destination data transfer size is 1 byte every time.
kEDMA_TransferSize2Bytes Source/Destination data transfer size is 2 bytes every time.
kEDMA_TransferSize4Bytes Source/Destination data transfer size is 4 bytes every time.
kEDMA_TransferSize8Bytes Source/Destination data transfer size is 8 bytes every time.
kEDMA_TransferSize16Bytes Source/Destination data transfer size is 16 bytes every time.
kEDMA_TransferSize32Bytes Source/Destination data transfer size is 32 bytes every time.
kEDMA_TransferSize64Bytes Source/Destination data transfer size is 64 bytes every time.

11.6.2 enum edma_modulo_t

Enumerator

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

11.6.3 enum edma_bandwidth_t

Enumerator

kEDMA_BandwidthStallNone No eDMA engine stalls.
kEDMA_BandwidthStall4Cycle eDMA engine stalls for 4 cycles after each read/write.
kEDMA_BandwidthStall8Cycle eDMA engine stalls for 8 cycles after each read/write.

11.6.4 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.

11.6.5 anonymous enum

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

11.6.6 anonymous enum

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.

11.6.7 anonymous enum

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

11.6.8 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.

11.6.9 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.

11.6.10 anonymous enum

Enumerator

kStatus_EDMA_QueueFull TCD queue is full.
kStatus_EDMA_Busy Channel is busy and can't handle the transfer request.

Function Documentation

11.7.1 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.

11.7.2 void EDMA_Deinit (DMA_Type * *base*)

This function gates the eDMA clock.

Parameters

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

11.7.3 void EDMA_InstallTCD (DMA_Type * *base*, uint32_t *channel*, edma_tcd_t * *tcd*)

Parameters

<i>base</i>	EDMA peripheral base address.
<i>channel</i>	EDMA channel number.
<i>tcd</i>	Point to TCD structure.

11.7.4 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.
---------------	--

11.7.5 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);
*
```

Function Documentation

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.

11.7.6 void EDMA_ResetChannel (DMA_Type * *base*, uint32_t *channel*)

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

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.

11.7.7 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, &stcd);
*
```

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.

11.7.8 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.

11.7.9 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.

11.7.10 `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.

11.7.11 static uint32_t EDMA_GetChannelSystemBusInformation (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

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.

11.7.12 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"> • <code>kEDMA_LinkNone</code> • <code>kEDMA_MinorLink</code> • <code>kEDMA_MajorLink</code>

Function Documentation

<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.

11.7.13 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">• kEDMABandwidthStallNone• kEDMABandwidthStall4Cycle• kEDMABandwidthStall8Cycle

11.7.14 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.

11.7.15 `static void EDMA_EnableAsyncRequest (DMA_Type * base, uint32_t channel, bool enable) [inline], [static]`

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).

11.7.16 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.

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).

11.7.17 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.

11.7.18 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 <code>edma_interrupt_enable_t</code> type.

11.7.19 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.

11.7.20 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.

Function Documentation

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.

11.7.21 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.

Parameters

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

11.7.22 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">• kEDMA_LinkNone• kEDMA_MinorLink• kEDMA_MajorLink
<i>linkedChannel</i>	The linked channel number.

11.7.23 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"> • kEDMABandwidthStallNone • kEDMABandwidthStall4Cycle • kEDMABandwidthStall8Cycle

11.7.24 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.

11.7.25 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).

11.7.26 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 <code>edma_interrupt_enable_t</code> type.

11.7.27 void EDMA_TcdDisableInterrupts (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 <code>edma_interrupt_enable_t</code> type.

11.7.28 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.

11.7.29 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.

Function Documentation

11.7.30 `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.

11.7.31 **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)

11.7.32 **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.

11.7.33 `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.

11.7.34 `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 <code>_edma_channel_status_flags</code> type.

11.7.35 `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.

11.7.36 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.

11.7.37 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.

11.7.38 void EDMA_PrepareTransferConfig (edma_transfer_config_t * *config*, void * *srcAddr*, uint32_t *srcWidth*, int16_t *srcOffset*, void * *destAddr*, uint32_t *destWidth*, int16_t *destOffset*, uint32_t *bytesEachRequest*, uint32_t *transferBytes*)

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

Function Documentation

Parameters

<i>config</i>	The user configuration structure of type edma_transfer_config_t .
<i>srcAddr</i>	eDMA transfer source address.
<i>srcWidth</i>	eDMA transfer source address width(bytes).
<i>srcOffset</i>	eDMA transfer source address offset
<i>destAddr</i>	eDMA transfer destination address.
<i>destWidth</i>	eDMA transfer destination address width(bytes).
<i>destOffset</i>	eDMA transfer destination address offset
<i>bytesEach-Request</i>	eDMA transfer bytes per channel request.
<i>transferBytes</i>	eDMA transfer bytes to be transferred.

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).

11.7.39 void EDMA_PrepareTransfer (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.

Parameters

<i>config</i>	The user configuration structure of type edma_transfer_config_t .
<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>bytesEach-Request</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).

11.7.40 **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.

11.7.41 **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.

Parameters

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

11.7.42 **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.

Function Documentation

Parameters

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

11.7.43 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.
---------------	---------------------

11.7.44 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.

11.7.45 static uint32_t EDMA_GetNextTCDAddress (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.

11.7.46 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.

Function Documentation

Parameters

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

Chapter 12

DPR: Display Prefetch Resolve

Overview

The MCUXpresso SDK provides a peripheral driver for the DPR module of MCUXpresso SDK devices. The DPR works with Prefetch Resolve Gasket (PRG) to prefetch the frame buffer data for display controller.

Data Structures

- struct `dpr_buffer_config_t`
Frame buffer configuration. [More...](#)

Macros

- #define `FSL_DPR_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)
Driver version.

Enumerations

- enum `dpr_data_type_t` {
 `kDPR_DataType16Bpp` = 1U,
 `kDPR_DataType32Bpp` = 2U }
Data type of the frame buffer.

Functions

- void `DPR_Init` (DPR_Type *base)
Enables and configures the DPR peripheral module.
- void `DPR_Deinit` (DPR_Type *base)
Disables the DPR peripheral module.
- void `DPR_SetBufferConfig` (DPR_Type *base, const `dpr_buffer_config_t` *config)
Set the input frame buffer configuration.
- void `DPR_BufferGetDefaultConfig` (`dpr_buffer_config_t` *config)
Get the input frame buffer default configuration.
- static void `DPR_Start` (DPR_Type *base)
Starts the DPR.
- static void `DPR_StartRepeat` (DPR_Type *base)
Starts the DPR to run repeatedly.
- static void `DPR_Stop` (DPR_Type *base)
Stops the DPR.
- static void `DPR_SetBufferAddr` (DPR_Type *base, uint32_t addr)
Set the frame buffer address.

Function Documentation

Data Structure Documentation

12.2.1 struct dpr_buffer_config_t

Data Fields

- uint16_t [width](#)
Frame buffer width, how many pixels per line.
- uint16_t [height](#)
Frame buffer height.
- uint16_t [strideBytes](#)
Stride in bytes.
- [dpr_data_type_t](#) [dataType](#)
Data type.

12.2.1.0.0.7 Field Documentation

12.2.1.0.0.7.1 uint16_t dpr_buffer_config_t::width

12.2.1.0.0.7.2 uint16_t dpr_buffer_config_t::height

12.2.1.0.0.7.3 uint16_t dpr_buffer_config_t::strideBytes

12.2.1.0.0.7.4 dpr_data_type_t dpr_buffer_config_t::dataType

Macro Definition Documentation

12.3.1 #define FSL_DPR_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Enumeration Type Documentation

12.4.1 enum dpr_data_type_t

Enumerator

kDPR_DataType16Bpp 16 bits per pixel.

kDPR_DataType32Bpp 32 bits per pixel.

Function Documentation

12.5.1 void DPR_Init (DPR_Type * *base*)

Parameters

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

12.5.2 void DPR_Deinit (DPR_Type * *base*)

Parameters

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

12.5.3 void DPR_SetBufferConfig (DPR_Type * *base*, const dpr_buffer_config_t * *config*)

Parameters

<i>base</i>	DPR peripheral address.
<i>config</i>	Pointer to the configuration.

12.5.4 void DPR_BufferGetDefaultConfig (dpr_buffer_config_t * *config*)

The default configuration is

```
config->width = 1080U;
config->height = 1920U;
config->strideBytes = 4U * 1080U;
config->dataType = kDPR_DataType32Bpp;
```

Parameters

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

12.5.5 static void DPR_Start (DPR_Type * *base*) [inline], [static]

This function triggers the DPR to load the new configuration and start processing the next frame. It should be called before display started.

Function Documentation

Parameters

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

12.5.6 static void DPR_StartRepeat (DPR_Type * *base*) [inline], [static]

This function should be called after display started. The display signal triggers the new configuration loading repeatedly.

Parameters

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

12.5.7 static void DPR_Stop (DPR_Type * *base*) [inline], [static]

Parameters

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

12.5.8 static void DPR_SetBufferAddr (DPR_Type * *base*, uint32_t *addr*) [inline], [static]

Parameters

<i>base</i>	DPR peripheral address.
<i>addr</i>	Frame buffer address.

Chapter 13

Display Processing Unit (DPU)

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.

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

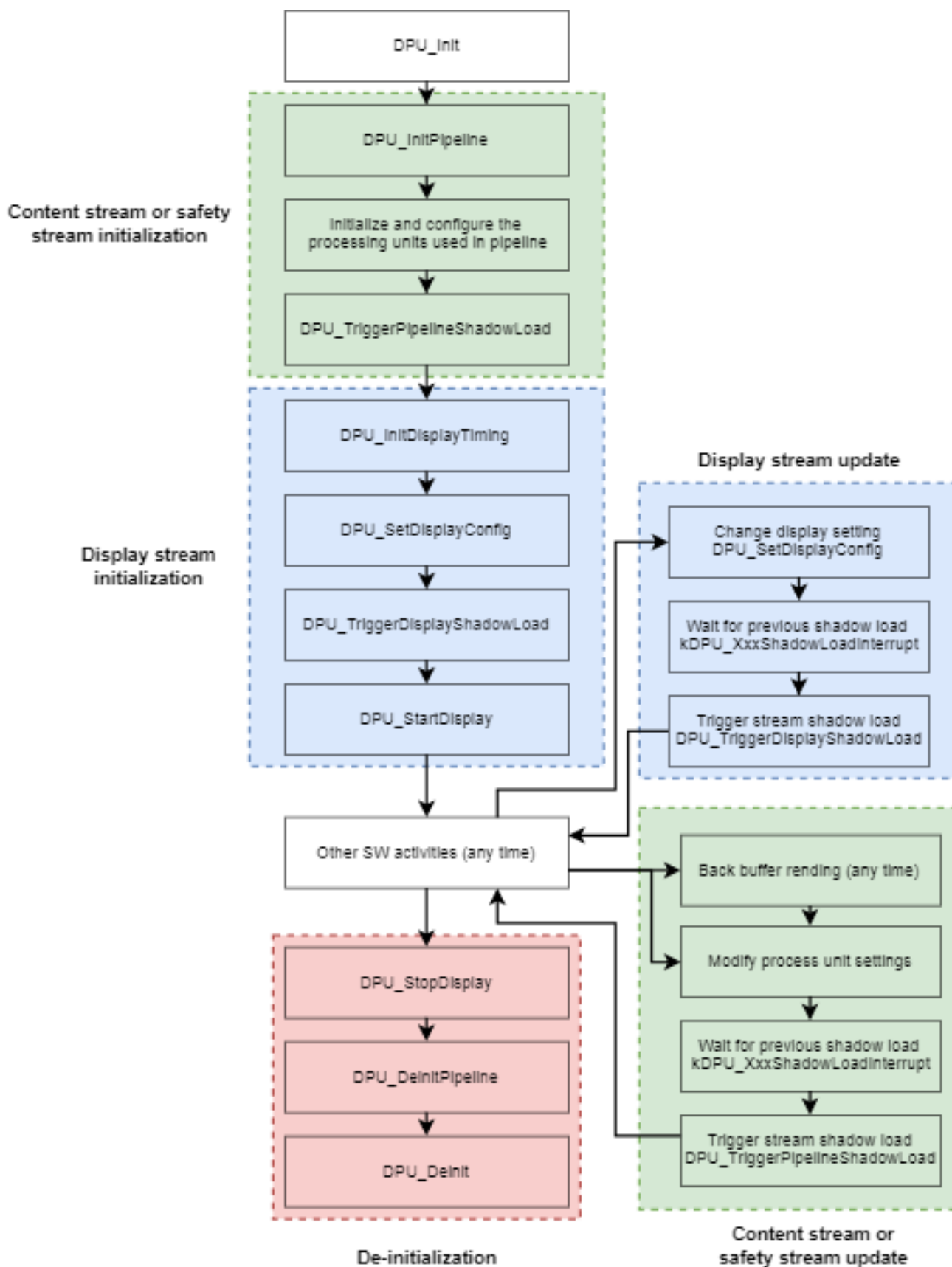


Figure 13.2.1: DPU display workflow
MCUXpresso SDK API Reference Manual

For the blit 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 blit engine units.
2. Trigger the blit 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:

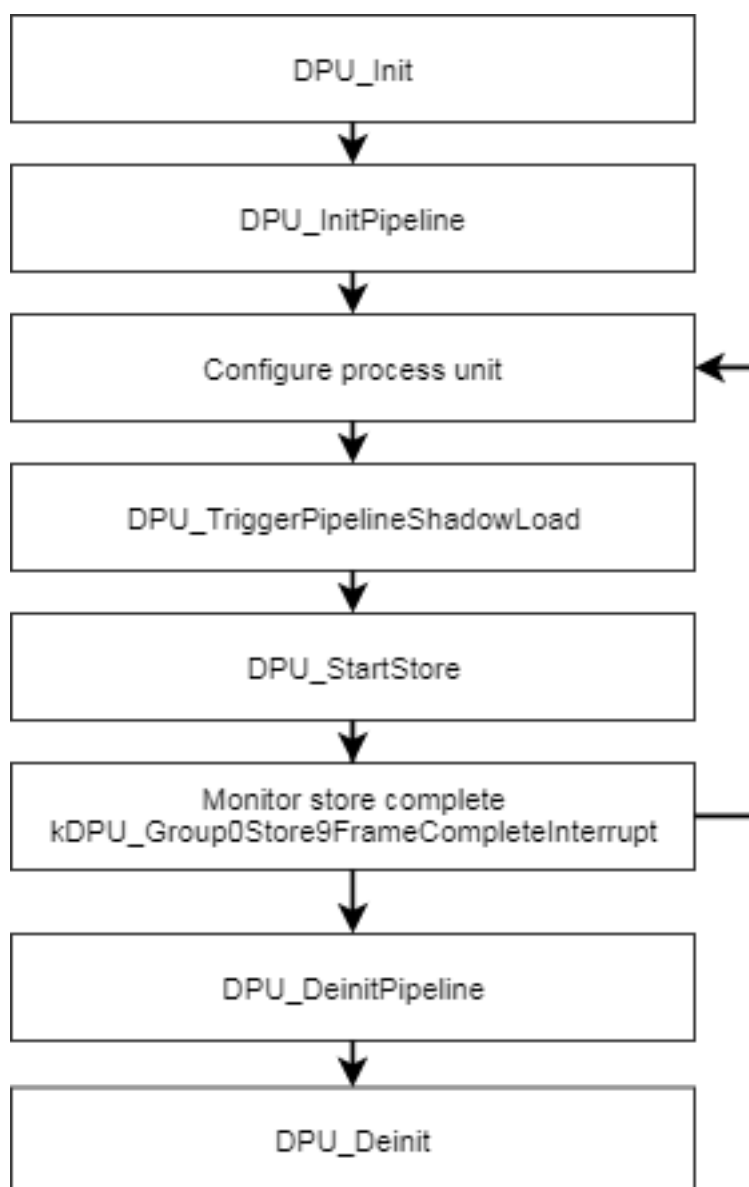


Figure 13.2.2: DPU blit engine workflow 1

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 interrupts. The function [DPU_TriggerPipelineCompleteInterrupt](#) should be used in this case. The workflow is:

1. Configure the blit engine units.
2. Trigger the blit 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:

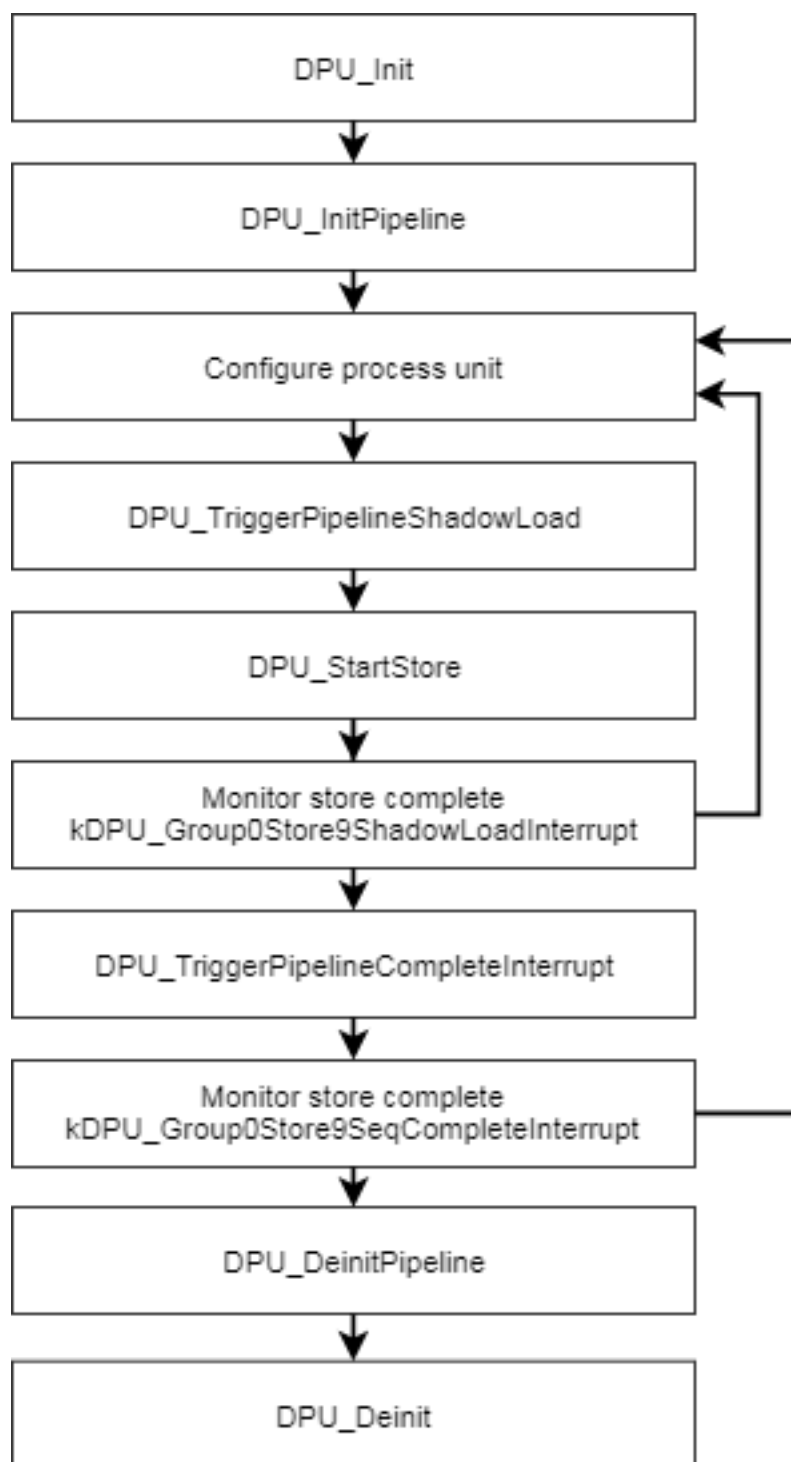


Figure 13.2.3: DPU blit engine workflow 2

Path configuration

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

Path configuration

The blit engine diagram is:

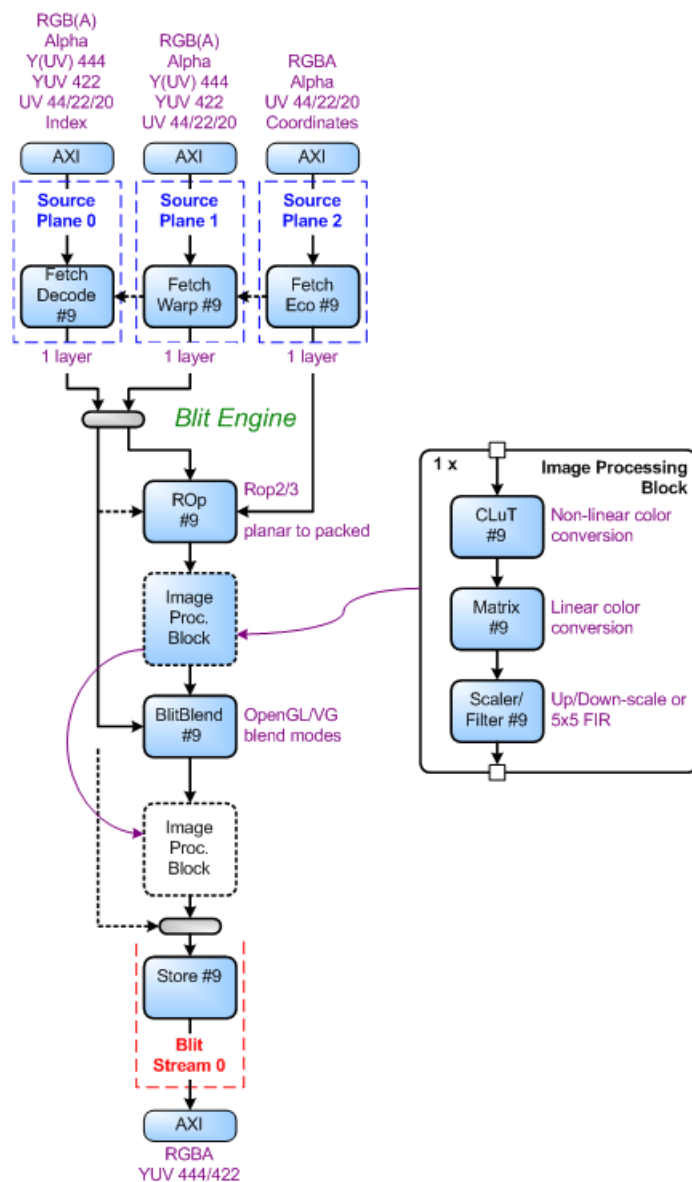


Figure 13.3.1: Blit Engine Block Diagram

The display controller block diagram is:

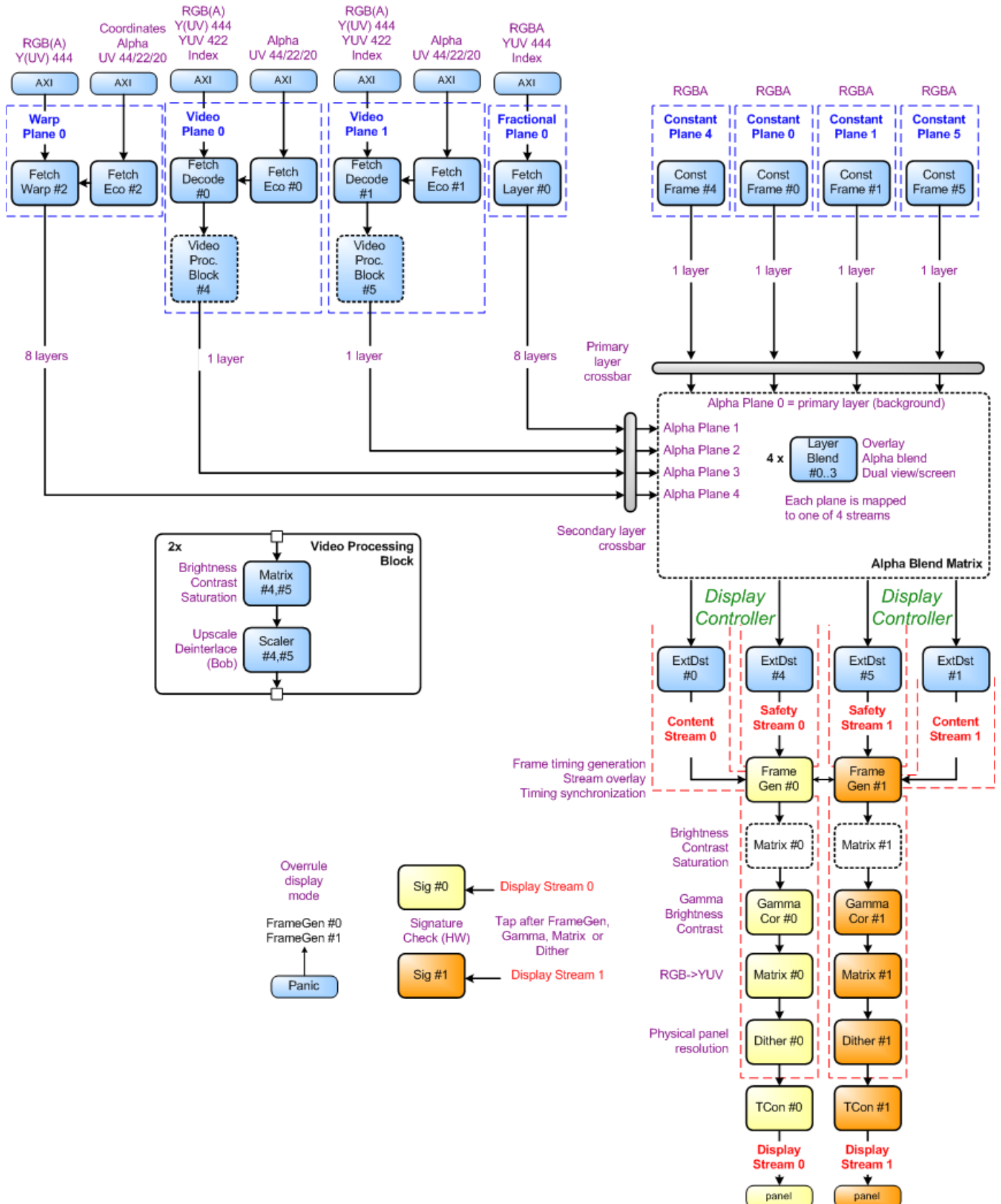


Figure 13.3.2: Display Controller Block Diagram

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:

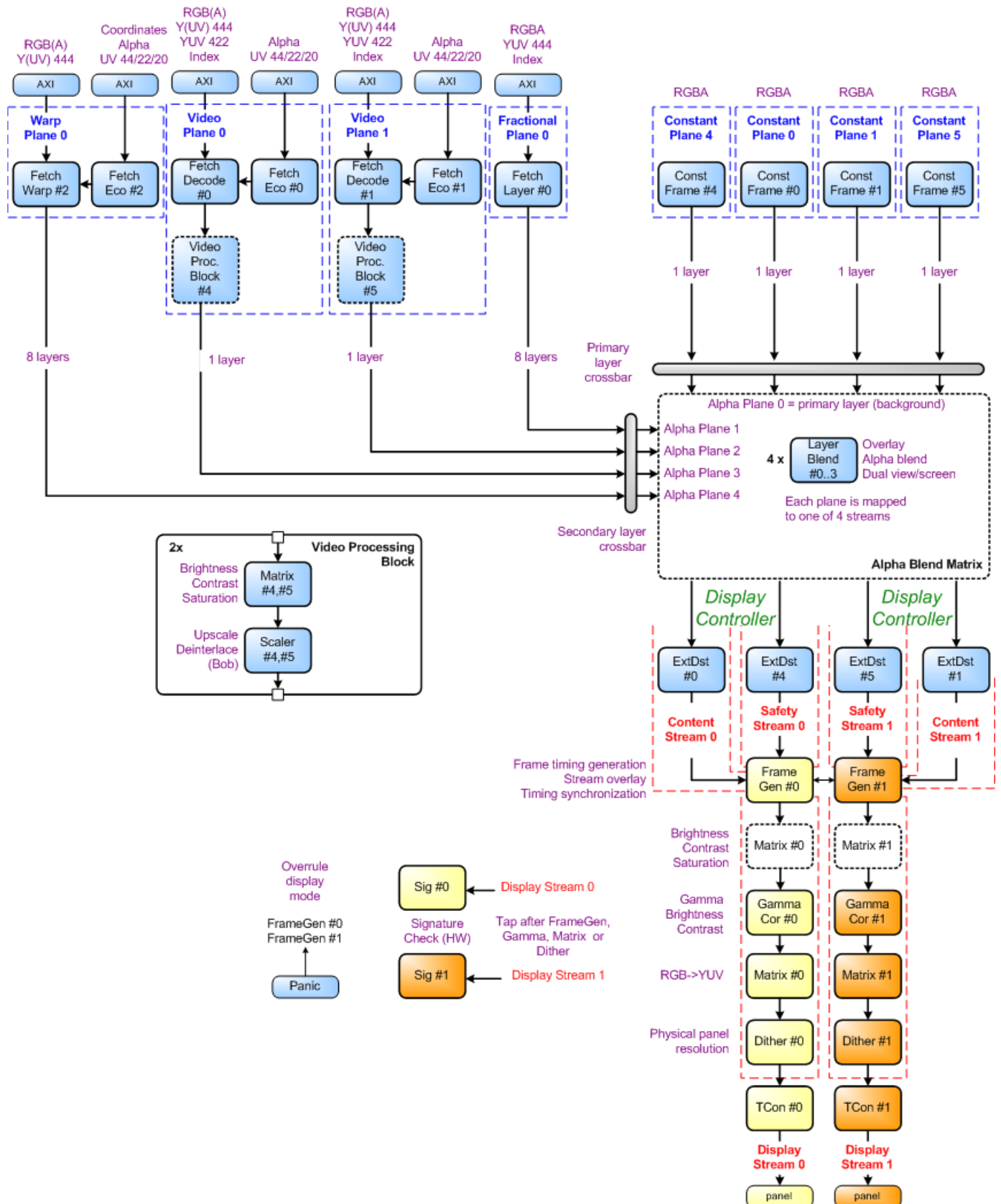


Figure 13.3.3: Default Path Configuration

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_blit_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](#) (256U)
DPU palette entry number.
- #define [DPU_FETCH_UNIT_BURST_LENGTH](#) (16U)
DPU fetch unit burst length, should be less than 16.
- #define [DPU_FETCH_UNIT_BURST_SIZE](#) (8U * DPU_FETCH_UNIT_BURST_LENGTH)
DPU fetch unit burst size.
- #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_blit_blend_func_t {
kDPU_BlitBlendFuncGIZero = 0,
kDPU_BlitBlendFuncGIOne = 1,
kDPU_BlitBlendFuncGISrcColor = 0x0300,
kDPU_BlitBlendFuncGIOneMinusSrcColor = 0x0301,
kDPU_BlitBlendFuncGISrcAlpha = 0x0302,
kDPU_BlitBlendFuncGIOneMinusSrcAlpha = 0x0303,
kDPU_BlitBlendFuncGIDstAlpha = 0x0304,
kDPU_BlitBlendFuncGIOneMinusDstAlpha = 0x0305,
kDPU_BlitBlendFuncGIDstColor = 0x0306,
kDPU_BlitBlendFuncGIOneMinusDstColor = 0x0307,
kDPU_BlitBlendFuncGISrcAlphaSaturate = 0x0308,
kDPU_BlitBlendFuncGIConstColor = 0x8001,
kDPU_BlitBlendFuncGIOneMinusConstColor = 0x8002,
kDPU_BlitBlendFuncGIConstAlpha = 0x8003,
kDPU_BlitBlendFuncGIOneMinusConstAlpha = 0x8004 }
BlitBlend blend function.
 - enum dpu_blit_blend_mode_t {

Path configuration

```
kDPU_BlitBlendModeGIFuncAdd = 0x8006,  
kDPU_BlitBlendModeGIFuncMin = 0x8007,  
kDPU_BlitBlendModeGIFuncMax = 0x8008,  
kDPU_BlitBlendModeGIFuncSubtract = 0x800A,  
kDPU_BlitBlendModeGIFuncReverseSubtract = 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_blit_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, 1, 1))
Driver version.

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 colnum colNum.
- #define `DPU_FRAME_GEN_INT_PER_FRAME`(rowNum) ((1U << 31U) | (((uint32_t)rowNum) & 0x3FFF0000U))

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_PreparePathConfig](#) (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.

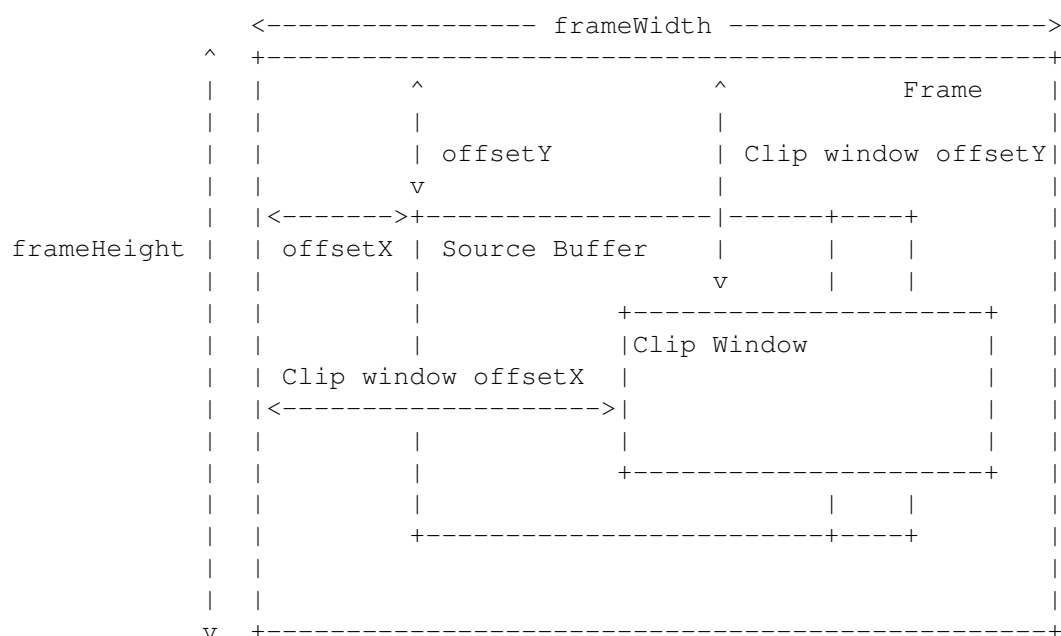
Pipeline.

- 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)

- *Trigger the pipeline shadow load.*
- 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

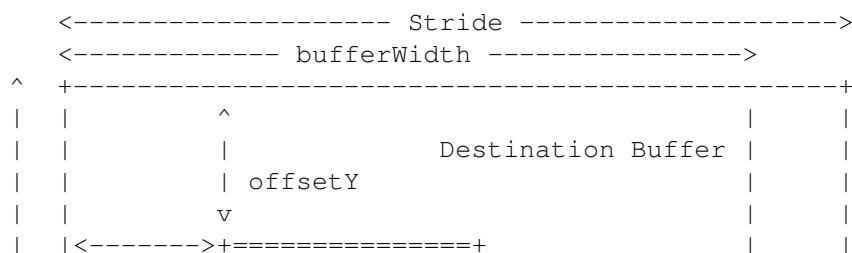
- Get the default warp configuration for FetchWarp unit.*
 - `status_t DPU_InitFetchUnitWarp` (IRIS_MVPL_Type *base, `dpu_unit_t` unit, const `dpu_warp_config_t` *config)
- Initialize the Warp function for FetchWarp unit.*
 - void `DPU_SrcBufferGetDefaultConfig` (`dpu_src_buffer_config_t` *config)
- 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.*

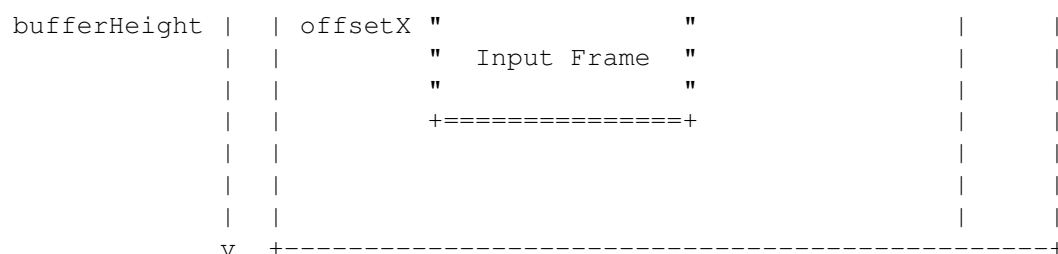
ExtDst Units

- 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:





- 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:

```
+-----+
|                                             |
|                                             |
|      +-----+                             |
|      | Window 0 xxxxxxxxxx|               |
|      |xxxxxxxxxxxxxxxxxxxx|               |
|      |xxxxxxxx+-----+                   |
|      |xxxxxxxx|   Window 1 (Skipped)       |
|      |xxxxxxxx|                           |
|      +-----+                             |
|                                             |
|               +-----+                   |
|                                             |
+-----+
```

In this example, windows 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.

Data Structure Documentation

13.4.1 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_REG1](#).
- uint16_t [frameHeight](#)
Frame height.
- uint16_t [frameWidth](#)
Frame width.

13.4.1.0.0.8 Field Documentation

13.4.1.0.0.8.1 uint32_t dpu_fetch_unit_config_t::srcReg

13.4.1.0.0.8.2 uint16_t dpu_fetch_unit_config_t::frameHeight

13.4.1.0.0.8.3 uint16_t dpu_fetch_unit_config_t::frameWidth

13.4.2 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

13.4.2.0.0.9 Field Documentation

13.4.2.0.0.9.1 uint8_t dpu_coordinates_config_t::bitsPerPixel

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

13.4.2.0.0.9.2 uint16_t dpu_coordinates_config_t::strideBytes

13.4.2.0.0.9.3 uint32_t dpu_coordinates_config_t::baseAddr

13.4.2.0.0.9.4 uint16_t dpu_coordinates_config_t::frameHeight

13.4.2.0.0.9.5 uint16_t dpu_coordinates_config_t::frameWidth

13.4.3 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_REGI](#).
- 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.

13.4.3.0.0.10 Field Documentation**13.4.3.0.0.10.1 uint32_t dpu_warp_config_t::srcReg****13.4.3.0.0.10.2 uint16_t dpu_warp_config_t::frameHeight****13.4.3.0.0.10.3 uint16_t dpu_warp_config_t::frameWidth****13.4.3.0.0.10.4 uint8_t dpu_warp_config_t::warpBitsPerPixel****13.4.3.0.0.10.5 bool dpu_warp_config_t::enableSymmetricOffset**

Recommended for small coordinate formats in DD_PNT mode.

13.4.3.0.0.10.6 dpu_warp_coordinate_mode_t dpu_warp_config_t::coordMode**13.4.3.0.0.10.7 uint32_t dpu_warp_config_t::arbStartX**

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

13.4.3.0.0.10.8 uint32_t dpu_warp_config_t::arbStartY

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

13.4.3.0.0.10.9 uint8_t dpu_warp_config_t::arbDeltaYY

Signed 3.5 fix-point. Used in DD_PNT.

13.4.3.0.0.10.10 uint8_t dpu_warp_config_t::arbDeltaYX

Signed 3.5 fix-point. Used in DD_PNT.

13.4.3.0.0.10.11 uint8_t dpu_warp_config_t::arbDeltaXY

Signed 3.5 fix-point. Used in DD_PNT.

13.4.3.0.0.10.12 uint8_t dpu_warp_config_t::arbDeltaXX

Signed 3.5 fix-point. Used in DD_PNT.

13.4.4 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 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](#).

13.4.4.0.0.11 Field Documentation

- 13.4.4.0.0.11.1 uint32_t [dpu_src_buffer_config_t::baseAddr](#)
- 13.4.4.0.0.11.2 uint16_t [dpu_src_buffer_config_t::strideBytes](#)
- 13.4.4.0.0.11.3 uint8_t [dpu_src_buffer_config_t::bitsPerPixel](#)
- 13.4.4.0.0.11.4 [dpu_pixel_format_t](#) [dpu_src_buffer_config_t::pixelFormat](#)
- 13.4.4.0.0.11.5 uint16_t [dpu_src_buffer_config_t::bufferHeight](#)
- 13.4.4.0.0.11.6 uint16_t [dpu_src_buffer_config_t::bufferWidth](#)
- 13.4.4.0.0.11.7 uint32_t [dpu_src_buffer_config_t::constColor](#)

13.4.5 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.

13.4.5.0.0.12 Field Documentation

13.4.5.0.0.12.1 `uint16_t dpu_clip_window_config_t::windowOffsetX`

13.4.5.0.0.12.2 `uint16_t dpu_clip_window_config_t::windowOffsetY`

13.4.5.0.0.12.3 `uint16_t dpu_clip_window_config_t::windowHeight`

13.4.5.0.0.12.4 `uint16_t dpu_clip_window_config_t::windowWidth`

13.4.6 `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

13.4.6.0.0.13 Field Documentation

13.4.6.0.0.13.1 `uint32_t dpu_dst_buffer_config_t::baseAddr`

13.4.6.0.0.13.2 `uint16_t dpu_dst_buffer_config_t::strideBytes`

13.4.6.0.0.13.3 `uint8_t dpu_dst_buffer_config_t::bitsPerPixel`

13.4.6.0.0.13.4 `dpu_pixel_format_t dpu_dst_buffer_config_t::pixelFormat`

13.4.6.0.0.13.5 `uint16_t dpu_dst_buffer_config_t::bufferHeight`

13.4.6.0.0.13.6 `uint16_t dpu_dst_buffer_config_t::bufferWidth`

13.4.7 `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.

13.4.7.0.0.14 Field Documentation**13.4.7.0.0.14.1** `uint8_t dpu_layer_blend_config_t::constAlpha`**13.4.7.0.0.14.2** `dpu_blend_mode_t dpu_layer_blend_config_t::secAlphaBlendMode`**13.4.7.0.0.14.3** `dpu_blend_mode_t dpu_layer_blend_config_t::primAlphaBlendMode`**13.4.7.0.0.14.4** `dpu_blend_mode_t dpu_layer_blend_config_t::secColorBlendMode`**13.4.7.0.0.14.5** `dpu_blend_mode_t dpu_layer_blend_config_t::primColorBlendMode`**13.4.7.0.0.14.6** `uint32_t dpu_layer_blend_config_t::srcReg`**13.4.7.0.0.14.7** `bool dpu_layer_blend_config_t::enableAlphaMask`**13.4.7.0.0.14.8** `dpu_alpha_mask_mode_t dpu_layer_blend_config_t::alphaMaskMode`**13.4.8 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.

13.4.8.0.0.15 Field Documentation

13.4.8.0.0.15.1 [uint8_t dpu_blit_blend_config_t::neutralBorderRightPixels](#)

13.4.8.0.0.15.2 [uint8_t dpu_blit_blend_config_t::neutralBorderLeftPixels](#)

13.4.8.0.0.15.3 [dpu_blit_blend_neutral_border_mode_t dpu_blit_blend_config_t::neutralBorderMode](#)

13.4.8.0.0.15.4 [uint32_t dpu_blit_blend_config_t::constColor](#)

13.4.9 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.

13.4.9.0.0.16 Field Documentation

13.4.9.0.0.16.1 [uint32_t dpu_rop_config_t::controlFlags](#)

13.4.9.0.0.16.2 [uint8_t dpu_rop_config_t::alphaIndex](#)

13.4.9.0.0.16.3 [uint8_t dpu_rop_config_t::blueIndex](#)

13.4.9.0.0.16.4 [uint8_t dpu_rop_config_t::greenIndex](#)

13.4.9.0.0.16.5 [uint8_t dpu_rop_config_t::redIndex](#)

13.4.10 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](#).

13.4.10.0.0.17 Field Documentation

13.4.10.0.0.17.1 uint16_t dpu_const_frame_config_t::frameHeight

13.4.10.0.0.17.2 uint16_t dpu_const_frame_config_t::frameWidth

13.4.10.0.0.17.3 uint32_t dpu_const_frame_config_t::constColor

13.4.11 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](#)
Vrtical front porch.
- uint16_t [vbp](#)
Vertical back porch.

13.4.11.0.0.18 Field Documentation

13.4.11.0.0.18.1 `uint16_t dpu_display_timing_config_t::flags`

13.4.11.0.0.18.2 `uint16_t dpu_display_timing_config_t::width`

13.4.11.0.0.18.3 `uint16_t dpu_display_timing_config_t::hsw`

13.4.11.0.0.18.4 `uint16_t dpu_display_timing_config_t::hfp`

13.4.11.0.0.18.5 `uint16_t dpu_display_timing_config_t::hbp`

13.4.11.0.0.18.6 `uint16_t dpu_display_timing_config_t::height`

13.4.11.0.0.18.7 `uint16_t dpu_display_timing_config_t::vsw`

13.4.11.0.0.18.8 `uint16_t dpu_display_timing_config_t::vfp`

13.4.11.0.0.18.9 `uint16_t dpu_display_timing_config_t::vbp`

13.4.12 `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.

13.4.12.0.0.19 Field Documentation

13.4.12.0.0.19.1 bool dpu_display_config_t::enablePrimAlpha

13.4.12.0.0.19.2 bool dpu_display_config_t::enableSecAlpha

13.4.12.0.0.19.3 dpu_display_mode_t dpu_display_config_t::displayMode

13.4.12.0.0.19.4 bool dpu_display_config_t::enablePrimAlphaInPanic

13.4.12.0.0.19.5 bool dpu_display_config_t::enableSecAlphaInPanic

13.4.12.0.0.19.6 dpu_display_mode_t dpu_display_config_t::displayModeInPanic

13.4.12.0.0.19.7 uint16_t dpu_display_config_t::constRed

13.4.12.0.0.19.8 uint16_t dpu_display_config_t::constGreen

13.4.12.0.0.19.9 uint16_t dpu_display_config_t::constBlue

13.4.12.0.0.19.10 uint8_t dpu_display_config_t::constAlpha

13.4.12.0.0.19.11 uint16_t dpu_display_config_t::primAreaStartX

14-bit , start from 1.

13.4.12.0.0.19.12 uint16_t dpu_display_config_t::primAreaStartY

14-bit, start from 1.

13.4.12.0.0.19.13 uint16_t dpu_display_config_t::secAreaStartX

14-bit, start from 1.

13.4.12.0.0.19.14 uint16_t dpu_display_config_t::secAreaStartY

14-bit, start from 1.

13.4.13 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_REGI](#).
- uint16_t [inputSize](#)

Data Structure Documentation

- *For HScaler, it is frame width, for VScaler, it is frame height.*
uint16_t [outputSize](#)
For HScaler, it is frame width, for VScaler, it is frame height.

13.4.13.0.0.20 Field Documentation

13.4.13.0.0.20.1 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 ->.

13.4.13.0.0.20.2 uint16_t dpu_scaler_config_t::inputSize

13.4.13.0.0.20.3 uint16_t dpu_scaler_config_t::outputSize

13.4.14 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.

13.4.14.0.0.21 Field Documentation

13.4.14.0.0.21.1 uint8_t dpu_signature_config_t::errorThreshold

13.4.14.0.0.21.2 uint8_t dpu_signature_config_t::errorResetThreshold

13.4.14.0.0.21.3 uint8_t dpu_signature_config_t::panicRed

13.4.14.0.0.21.4 uint8_t dpu_signature_config_t::panicGreen

13.4.14.0.0.21.5 uint8_t dpu_signature_config_t::panicBlue

13.4.14.0.0.21.6 uint8_t dpu_signature_config_t::panicAlpha

Must be 0 or 1

13.4.15 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.

13.4.15.0.0.22 Field Documentation

13.4.15.0.0.22.1 uint32_t dpu_signature_window_config_t::controlFlags

13.4.15.0.0.22.2 uint16_t dpu_signature_window_config_t::upperLeftX

13.4.15.0.0.22.3 uint16_t dpu_signature_window_config_t::upperLeftY

13.4.15.0.0.22.4 uint16_t dpu_signature_window_config_t::lowerRightX

13.4.15.0.0.22.5 uint16_t dpu_signature_window_config_t::lowerRightY

Macro Definition Documentation

13.5.1 #define FSL_DPU_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

13.5.2 #define DPU_PALETTE_ENTRY_NUM (256U)

13.5.3 #define DPU_FETCH_UNIT_BURST_LENGTH (16U)

13.5.4 #define DPU_FETCH_UNIT_BURST_SIZE (8U * DPU_FETCH_UNIT_BURST_LENGTH)

If prefetch is used, the frame buffer stride and base address should be aligned to the burst size.

Enumeration Type Documentation

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

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

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

Value:

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

13.5.8 `#define DPU_MAKE_CONST_COLOR(red, green, blue, alpha) (((uint32_t)(red)) << 24U) | (((uint32_t)(green)) << 16U) | (((uint32_t)(blue)) << 8U) | ((uint32_t)(alpha)))`

13.5.9 `#define DPU_FRAME_GEN_INT_DISABLE 0U`

13.5.10 `#define DPU_FRAME_GEN_INT_PER_LINE(colNum) ((1U << 31U) | (1U << 15U) | (((uint32_t)colNum) & (0x3FFFU)))`

13.5.11 `#define DPU_FRAME_GEN_INT_PER_FRAME(rowNum) ((1U << 31U) | (((uint32_t)rowNum) & 0x3FFF0000U))`

Enumeration Type Documentation

13.6.1 `enum dpu_unit_t`

13.6.2 `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.

13.6.3 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.

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.

13.6.4 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.

13.6.5 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.

13.6.6 enum dpu_clip_color_mode_t

Enumerator

kDPU_ClipColorNull Use null color.
kDPU_ClipColorSublayer Use color of sublayer.

13.6.7 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.

13.6.8 enum dpu_blend_mode_t

Enumerator

kDPU_BlendZero OUT = IN * 0.

Enumeration Type Documentation

kDPU_BlendOne $\text{OUT} = \text{IN} * 1.$
kDPU_BlendPrimAlpha $\text{OUT} = \text{IN} * \text{ALPHA_primary}.$
kDPU_BlendPrimAlphaInv $\text{OUT} = \text{IN} * (1 - \text{ALPHA_primary}).$
kDPU_BlendSecAlpha $\text{OUT} = \text{IN} * \text{ALPHA_secondary}.$
kDPU_BlendSecAlphaInv $\text{OUT} = \text{IN} * (1 - \text{ALPHA_secondary}).$
kDPU_BlendConstAlpha $\text{OUT} = \text{IN} * \text{ALPHA_const}.$
kDPU_BlendConstAlphaInv $\text{OUT} = \text{IN} * (1 - \text{ALPHA_const}).$

13.6.9 enum dpu_blit_blend_func_t

Enumerator

kDPU_BlitBlendFuncGlZero $\text{GL_ZERO}.$
kDPU_BlitBlendFuncGlOne $\text{GL_ONE}.$
kDPU_BlitBlendFuncGlSrcColor $\text{GL_SRC_COLOR}.$
kDPU_BlitBlendFuncGlOneMinusSrcColor $\text{GL_ONE_MINUS_SRC_COLOR}.$
kDPU_BlitBlendFuncGlSrcAlpha $\text{GL_SRC_ALPHA}.$
kDPU_BlitBlendFuncGlOneMinusSrcAlpha $\text{GL_ONE_MINUS_SRC_ALPHA}.$
kDPU_BlitBlendFuncGlDstAlpha $\text{GL_DST_ALPHA}.$
kDPU_BlitBlendFuncGlOneMinusDstAlpha $\text{GL_ONE_MINUS_DST_ALPHA}.$
kDPU_BlitBlendFuncGlDstColor $\text{GL_DST_COLOR}.$
kDPU_BlitBlendFuncGlOneMinusDstColor $\text{GL_ONE_MINUS_DST_COLOR}.$
kDPU_BlitBlendFuncGlSrcAlphaSaturate $\text{GL_SRC_ALPHA_SATURATE}.$
kDPU_BlitBlendFuncGlConstColor $\text{GL_CONSTANT_COLOR}.$
kDPU_BlitBlendFuncGlOneMinusConstColor $\text{GL_ONE_MINUS_CONSTANT_COLOR}.$
kDPU_BlitBlendFuncGlConstAlpha $\text{GL_CONSTANT_ALPHA}.$
kDPU_BlitBlendFuncGlOneMinusConstAlpha $\text{GL_ONE_MINUS_CONSTANT_ALPHA}.$

13.6.10 enum dpu_blit_blend_mode_t

Enumerator

kDPU_BlitBlendModeGlFuncAdd $\text{GL_FUNC_ADD}.$
kDPU_BlitBlendModeGlMin $\text{GL_MIN}.$
kDPU_BlitBlendModeGlMax $\text{GL_MAX}.$
kDPU_BlitBlendModeGlFuncSubtract $\text{GL_FUNC_SUBTRACT}.$
kDPU_BlitBlendModeGlFuncReverseSubtract $\text{GL_FUNC_REVERSE_SUBTRACT}.$
kDPU_BlitBlendModeVgBlendSrc $\text{VG_BLEND_SRC}.$
kDPU_BlitBlendModeVgBlendSrcOver $\text{VG_BLEND_SRC_OVER}.$
kDPU_BlitBlendModeVgBlendDstOver $\text{VG_BLEND_DST_OVER}.$
kDPU_BlitBlendModeVgBlendSrcIn $\text{VG_BLEND_SRC_IN}.$
kDPU_BlitBlendModeVgBlendDstIn $\text{VG_BLEND_DST_IN}.$

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.

13.6.11 enum dpu_blit_blend_neutral_border_mode_t

Enumerator

kDPU_BlitBlendNeutralBorderPrim Bypasses primary pixel.
kDPU_BlitBlendNeutralBorderSec Bypasses secondary pixel.

13.6.12 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.

13.6.13 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.

Function Documentation

13.6.14 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.

13.6.15 enum _dpu_signature_window_flags

Enumerator

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

13.6.16 enum _dpu_signature_status

Enumerator

kDPU_SignatureIdle Signature unit is in idle status.
kDPU_SignatureValid Signature unit is in idle status.

Function Documentation

13.7.1 void DPU_Init (IRIS_MVPL_Type * *base*)

This function ungates the DPU clock.

Parameters

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

13.7.2 void DPU_Deinit (IRIS_MVPL_Type * *base*)

This function gates the DPU clock.

Parameters

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

13.7.3 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.
-------------	------------------------------

13.7.4 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.

Function Documentation

<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.

13.7.5 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

<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.

13.7.6 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](#).

13.7.7 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.

13.7.8 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

Function Documentation

<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.

13.7.9 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 usage are the same.

Parameters

<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 .

13.7.10 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 .

13.7.11 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 .

13.7.12 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](#).

13.7.13 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 .

13.7.14 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](#).

Function Documentation

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 .

13.7.15 **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);
```

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.

13.7.16 **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.

13.7.17 **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.

13.7.18 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.

13.7.19 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.

13.7.20 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.

Function Documentation

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 .

13.7.21 void DPU_FetchUnitGetDefaultConfig (dpu_fetch_unit_config_t * *config*)

The default value is:

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

Parameters

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

13.7.22 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.

13.7.23 **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.

13.7.24 **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> .

Function Documentation

Return values

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

13.7.25 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.

13.7.26 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.
---------------	---

13.7.27 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.

13.7.28 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.
---------------	---

13.7.29 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](#)

Function Documentation

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.

13.7.30 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.
---------------	---

13.7.31 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.

13.7.32 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.

13.7.33 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.

13.7.34 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.

13.7.35 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.

13.7.36 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.
---------------	---

13.7.37 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.

13.7.38 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.

13.7.39 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_ClipColor-Sublayer .

13.7.40 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 select 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

13.7.41 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 .

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

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.

13.7.43 **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.
---------------	-------------------------------

13.7.44 **void DPU_SetStoreDstBufferAddr (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, uint32_t *baseAddr*)**

This function is run time used for better performance.

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>baseAddr</i>	Base address of the Destination buffer to set.

13.7.45 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

13.7.46 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.
...

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.

Parameters

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

13.7.47 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 .

Function Documentation

13.7.48 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.
---------------	---

13.7.49 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.

13.7.50 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.

13.7.51 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](#)

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 .

13.7.52 void DPU_BlitBlendGetDefaultConfig (dpu_blit_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.
---------------	---

13.7.53 void DPU_SetBlitBlendConfig (IRIS_MVPL_Type * *base*, dpu_unit_t *unit*, const dpu_blit_blend_config_t * *config*)

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

13.7.54 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.

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.

13.7.55 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.
---------------	---

13.7.56 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 .

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

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

13.7.58 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.

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.

13.7.59 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.

13.7.60 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.
---------------	---

13.7.61 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.

13.7.62 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.

13.7.63 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.
---------------	---

13.7.64 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](#)

Function Documentation

- [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.

13.7.65 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.
---------------	---

13.7.66 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.

13.7.67 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.
---------------	---

13.7.68 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.

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

Function Documentation

Parameters

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

13.7.70 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.

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

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.

13.7.72 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 beginning of next frame.

Parameters

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

13.7.73 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.
---------------	-------------------------------

13.7.74 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.

13.7.75 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.
---------------	-------------------------------

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

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>config</i>	Pointer to the configuration.

13.7.77 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)

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.

13.7.78 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.

13.7.79 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.

13.7.80 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>	Referencesignature value of green.
<i>blueCRC</i>	Reference signature value of blue.

13.7.81 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)
{
    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);
```

Function Documentation

```
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.

13.7.82 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 14

DPU IRQSTEER: Interrupt Request Steering Driver

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))

Function Documentation

14.2.1 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.

14.2.2 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.

14.2.3 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 15

ENET: Ethernet MAC Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the 10/100 Mbps Ethernet MAC (ENET) module of MCUXpresso SDK devices.

ENET: Ethernet MAC Driver {EthernetMACDriver}

Operations of Ethernet MAC Driver

15.2.1 MII interface Operation

The MII interface is the interface connected with MAC and PHY. the Serial management interface - MII management interface should be set before any access to the external PHY chip register. Call [ENET_SetSMI\(\)](#) to initialize the MII management interface. Use [ENET_StartSMIRead\(\)](#), [ENET_StartSMIWrite\(\)](#), and [ENET_ReadSMIData\(\)](#) to read/write to PHY registers. This function group sets up the MII and serial management SMI interface, gets data from the SMI interface, and starts the SMI read and write command. Use [ENET_SetMII\(\)](#) to configure the MII before successfully getting data from the external PHY.

15.2.2 MAC address filter

This group sets/gets the ENET mac address and the multicast group address filter. [ENET_AddMulticastGroup\(\)](#) should be called to add the ENET MAC to the multicast group. The IEEE 1588 feature requires receiving the PTP message.

15.2.3 Other Basic control Operations

This group has the receive active API [ENET_ActiveRead\(\)](#) for single and multiple rings. The [ENET_AVBConfigure\(\)](#) is provided to configure the AVB features to support the AVB frames transmission. Note that due to the AVB frames transmission scheme being a credit-based TX scheme, it is only supported with the Enhanced buffer descriptors. Because of this, the AVB configuration should only be done with the Enhanced buffer descriptor. When the AVB feature is required, make sure the "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" is defined before using this feature.

15.2.4 Transactional Operation

For ENET receive, the [ENET_GetRxFrameSize\(\)](#) function needs to be called to get the received data size. Then, call the [ENET_ReadFrame\(\)](#) function to get the received data. If the received error occurs, call the

Typical use case

[ENET_GetRxErrBeforeReadFrame\(\)](#) function after [ENET_GetRxFrameSize\(\)](#) and before [ENET_ReadFrame\(\)](#) functions to get the detailed error information.

For ENET transmit, call the [ENET_SendFrame\(\)](#) function to send the data out. The transmit data error information is only accessible for the IEEE 1588 enhanced buffer descriptor mode. When the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined, the [ENET_GetTxErrAfterSendFrame\(\)](#) can be used to get the detail transmit error information. The transmit error information can only be updated by uDMA after the data is transmitted. The [ENET_GetTxErrAfterSendFrame\(\)](#) function is recommended to be called on the transmit interrupt handler.

If send/read frame with zero-copy mechanism is needed, there're special APIs like [ENET_GetRxBuffer\(\)](#), [ENET_ReleaseRxBuffer\(\)](#), [ENET_SendFrameZeroCopy\(\)](#) and [ENET_SetTxBuffer\(\)](#). The send frame zero-copy APIs can't be used mixed with [ENET_SendFrame\(\)](#) for the same ENET peripheral, same as read frame zero-copy APIs.

15.2.5 PTP IEEE 1588 Feature Operation

This function group configures the PTP IEEE 1588 feature, starts/stops/gets/sets/adjusts the PTP IEEE 1588 timer, gets the receive/transmit frame timestamp, and PTP IEEE 1588 timer channel feature setting.

The [ENET_Ptp1588Configure\(\)](#) function needs to be called when the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined and the IEEE 1588 feature is required.

Typical use case

15.3.1 ENET Initialization, receive, and transmit operations

For the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` undefined use case, use the legacy type buffer descriptor transmit/receive the frame as follows. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet` For the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` defined use case, add the PTP IEEE 1588 configuration to enable the PTP IEEE 1588 feature. The initialization occurs as follows. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet`

Data Structures

- struct [enet_rx_bd_struct_t](#)
Defines the receive buffer descriptor structure for the little endian system. [More...](#)
- struct [enet_tx_bd_struct_t](#)
Defines the enhanced transmit buffer descriptor structure for the little endian system. [More...](#)
- struct [enet_data_error_stats_t](#)
Defines the ENET data error statistics structure. [More...](#)
- struct [enet_frame_info_t](#)
Defines the frame info structure. [More...](#)
- struct [enet_tx_dirty_ring_t](#)
Defines the ENET transmit dirty addresses ring/queue structure. [More...](#)
- struct [enet_buffer_config_t](#)

- Defines the receive buffer descriptor configuration structure. [More...](#)
- struct [enet_intcoalesce_config_t](#)
Defines the interrupt coalescing configure structure. [More...](#)
- struct [enet_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_tx_bd_ring_t](#)
Defines the ENET transmit buffer descriptor ring/queue structure. [More...](#)
- struct [enet_rx_bd_ring_t](#)
Defines the ENET receive buffer descriptor ring/queue structure. [More...](#)
- struct [enet_handle_t](#)
Defines the ENET handler structure. [More...](#)

Macros

- #define [ENET_BUFFDESCRIPTOR_RX_ERR_MASK](#)
Defines the receive error status flag mask.

Typedefs

- typedef void(* [enet_callback_t](#))(ENET_Type *base, enet_handle_t *handle, uint32_t ringId, [enet_event_t](#) event, [enet_frame_info_t](#) *frameInfo, void *userData)
ENET callback function.
- typedef void(* [enet_isr_ring_t](#))(ENET_Type *base, enet_handle_t *handle, uint32_t ringId)
Define interrupt IRQ handler.

Enumerations

- enum {
[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) }
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.

Typical use case

- enum `enet_mii_duplex_t` {
 `kENET_MiiHalfDuplex` = 0U,
 `kENET_MiiFullDuplex` }
 Defines the half or full duplex for the MII data interface.
- enum `enet_mii_write_t` {
 `kENET_MiiWriteNoCompliant` = 0U,
 `kENET_MiiWriteValidFrame` }
 Define the MII opcode for normal MDIO_CLAUSES_22 Frame.
- enum `enet_mii_read_t` {
 `kENET_MiiReadValidFrame` = 2U,
 `kENET_MiiReadNoCompliant` = 3U }
 Defines the read operation for the MII management frame.
- enum `enet_mii_extend_opcode` {
 `kENET_MiiAddrWrite_C45` = 0U,
 `kENET_MiiWriteFrame_C45` = 1U,
 `kENET_MiiReadFrame_C45` = 3U }
 Define the MII opcode for extended MDIO_CLAUSES_45 Frame.
- enum `enet_special_control_flag_t` {
 `kENET_ControlFlowControlEnable` = 0x0001U,
 `kENET_ControlRxPayloadCheckEnable` = 0x0002U,
 `kENET_ControlRxPadRemoveEnable` = 0x0004U,
 `kENET_ControlRxBroadCastRejectEnable` = 0x0008U,
 `kENET_ControlMacAddrInsert` = 0x0010U,
 `kENET_ControlStoreAndFwdDisable` = 0x0020U,
 `kENET_ControlSMIPreambleDisable` = 0x0040U,
 `kENET_ControlPromiscuousEnable` = 0x0080U,
 `kENET_ControlMIILoopEnable` = 0x0100U,
 `kENET_ControlVLANTagEnable` = 0x0200U,
 `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` {

Typical use case

```
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.

Functions

- uint32_t `ENET_GetInstance` (ENET_Type *base)
Get the ENET instance from peripheral base address.

Driver version

- #define `FSL_ENET_DRIVER_VERSION` (`MAKE_VERSION`(2, 3, 0))
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_LENVIOLATE_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_TRANMITCRC_MASK` 0x0400U
Transmit CRC 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.

Typical use case

- #define `ENET_PHY_MAXADDRESS` (`ENET_MMFR_PA_MASK >> ENET_MMFR_PA_SHIFT`)
Maximum PHY address.
- #define `ENET_TX_INTERRUPT`
Enet Tx interrupt flag.
- #define `ENET_RX_INTERRUPT`
Enet Rx interrupt flag.
- #define `ENET_TS_INTERRUPT` (`((uint32_t)kENET_TsTimerInterrupt | (uint32_t)kENET_TsAvailInterrupt)`)
Enet timestamp interrupt flag.
- #define `ENET_ERR_INTERRUPT`
Enet error interrupt flag.

Defines Tx operation flags.

- #define `ENET_TX_LAST_BD_FLAG` `0x01U`
Tx set last buffer descriptor flag.
- #define `ENET_TX_TIMESTAMP_FLAG` `0x02U`
Tx timestamp flag.

Initialization and De-initialization

- void `ENET_GetDefaultConfig` (`enet_config_t *config`)
Gets the ENET default configuration structure.
- void `ENET_Up` (`ENET_Type *base`, `enet_handle_t *handle`, const `enet_config_t *config`, const `enet_buffer_config_t *bufferConfig`, `uint8_t *macAddr`, `uint32_t srcClock_Hz`)
Initializes the ENET module.
- 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_Down` (`ENET_Type *base`)
Stops the ENET module.
- void `ENET_Deinit` (`ENET_Type *base`)
Deinitializes the ENET module.
- static void `ENET_Reset` (`ENET_Type *base`)
Resets the ENET module.

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.
- void [ENET_StartExtC45SMIWriteReg](#) (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg)
Starts the extended IEEE802.3 Clause 45 MDIO format SMI write register command.
- void [ENET_StartExtC45SMIWriteData](#) (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg, uint32_t data)
Starts the extended IEEE802.3 Clause 45 MDIO format SMI write data command.
- void [ENET_StartExtC45SMIReadData](#) (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg)
Starts the extended IEEE802.3 Clause 45 MDIO format SMI read data command.
- static void [ENET_SetRGMIIIClockDelay](#) (ENET_Type *base, bool txEnabled, bool rxEnabled)
Control the usage of the delayed tx/rx RGMII 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

- static void [ENET_ActiveRead](#) (ENET_Type *base)
Activates ENET read or receive.
- static void [ENET_EnableSleepMode](#) (ENET_Type *base, bool enable)
Enables/disables the MAC to enter sleep mode.
- static void [ENET_GetAccelFunction](#) (ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)
Gets ENET transmit and receive accelerator functions from MAC controller.

Interrupts.

- static void [ENET_EnableInterrupts](#) (ENET_Type *base, uint32_t mask)
Enables the ENET interrupt.
- static void [ENET_DisableInterrupts](#) (ENET_Type *base, uint32_t mask)
Disables the ENET interrupt.
- static uint32_t [ENET_GetInterruptStatus](#) (ENET_Type *base)
Gets the ENET interrupt status flag.
- static void [ENET_ClearInterruptStatus](#) (ENET_Type *base, uint32_t mask)
Clears the ENET interrupt events status flag.
- void [ENET_SetRxISRHandler](#) (ENET_Type *base, [enet_isr_ring_t](#) ISRHandler)

Typical use case

- *Set the second level Rx IRQ handler.*
void [ENET_SetTxISRHandler](#) (ENET_Type *base, [enet_isr_ring_t](#) ISRHandler)
- *Set the second level Tx IRQ handler.*
void [ENET_SetErrISRHandler](#) (ENET_Type *base, [enet_isr_t](#) ISRHandler)
- *Set the second level Err IRQ handler.*

Transactional operation

- void [ENET_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](#) *eError-Static, uint8_t ringId)
Gets the error statistics of a received frame for ENET specified ring.
- [status_t](#) [ENET_GetRxFrameSize](#) ([enet_handle_t](#) *handle, uint32_t *length, uint8_t ringId)
Gets the size of the read frame for specified ring.
- [status_t](#) [ENET_ReadFrame](#) (ENET_Type *base, [enet_handle_t](#) *handle, uint8_t *data, uint32_t length, uint8_t ringId, uint32_t *ts)
Reads a frame from the ENET device.
- [status_t](#) [ENET_SendFrame](#) (ENET_Type *base, [enet_handle_t](#) *handle, const uint8_t *data, uint32_t length, uint8_t ringId, bool tsFlag, void *context)
Transmits an ENET frame for specified ring.
- [status_t](#) [ENET_SetTxReclaim](#) ([enet_handle_t](#) *handle, bool isEnabled, uint8_t ringId)
Enable or disable tx descriptors reclaim mechanism.
- [status_t](#) [ENET_GetRxBuffer](#) (ENET_Type *base, [enet_handle_t](#) *handle, void **buffer, uint32_t *length, uint8_t ringId, bool *isLastBuff, uint32_t *ts)
Get a receive buffer pointer of the ENET device for specified ring.
- void [ENET_ReleaseRxBuffer](#) (ENET_Type *base, [enet_handle_t](#) *handle, void *buffer, uint8_t ringId)
Release receive buffer descriptor to DMA.
- [status_t](#) [ENET_SendFrameZeroCopy](#) (ENET_Type *base, [enet_handle_t](#) *handle, const uint8_t *data, uint32_t length, uint8_t ringId, bool tsFlag, void *context)
Transmits an ENET frame for specified ring with zero-copy.
- [status_t](#) [ENET_SetTxBuffer](#) (ENET_Type *base, [enet_handle_t](#) *handle, const uint8_t *data, uint32_t length, uint8_t ringId, uint8_t txFlag, void *context)
Set up ENET Tx buffer descriptor, preparing for one frame stores in scattered buffer.
- 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.

Data Structure Documentation

15.4.1 struct enet_rx_bd_struct_t

Data Fields

- uint16_t [length](#)
Buffer descriptor data length.
- uint16_t [control](#)
Buffer descriptor control and status.
- uint8_t * [buffer](#)
Data buffer pointer.

15.4.1.0.0.23 Field Documentation

15.4.1.0.0.23.1 uint16_t enet_rx_bd_struct_t::length

15.4.1.0.0.23.2 uint16_t enet_rx_bd_struct_t::control

15.4.1.0.0.23.3 uint8_t* enet_rx_bd_struct_t::buffer

15.4.2 struct enet_tx_bd_struct_t

Data Fields

- uint16_t [length](#)
Buffer descriptor data length.
- uint16_t [control](#)
Buffer descriptor control and status.
- uint8_t * [buffer](#)
Data buffer pointer.

15.4.2.0.0.24 Field Documentation

15.4.2.0.0.24.1 uint16_t enet_tx_bd_struct_t::length

15.4.2.0.0.24.2 uint16_t enet_tx_bd_struct_t::control

15.4.2.0.0.24.3 uint8_t* enet_tx_bd_struct_t::buffer

15.4.3 struct enet_data_error_stats_t

Data Fields

- uint32_t [statsRxLenGreaterErr](#)
Receive length greater than RCR[MAX_FL].
- uint32_t [statsRxAlignErr](#)
Receive non-octet alignment/.
- uint32_t [statsRxFcsErr](#)

Data Structure Documentation

- *Receive CRC error.*
uint32_t [statsRxOverRunErr](#)
- *Receive over run.*
uint32_t [statsRxTruncateErr](#)
- *Receive truncate.*

15.4.3.0.0.25 Field Documentation

15.4.3.0.0.25.1 uint32_t enet_data_error_stats_t::statsRxLenGreaterErr

15.4.3.0.0.25.2 uint32_t enet_data_error_stats_t::statsRxFcsErr

15.4.3.0.0.25.3 uint32_t enet_data_error_stats_t::statsRxOverRunErr

15.4.3.0.0.25.4 uint32_t enet_data_error_stats_t::statsRxTruncateErr

15.4.4 struct enet_frame_info_t

Data Fields

- void * [context](#)
User specified data.

15.4.5 struct enet_tx_dirty_ring_t

Data Fields

- [enet_frame_info_t](#) * [txDirtyBase](#)
Dirty buffer descriptor base address pointer.
- uint16_t [txGenIdx](#)
tx generate index.
- uint16_t [txConsumeIdx](#)
tx consume index.
- uint16_t [txRingLen](#)
tx ring length.
- bool [isFull](#)
tx ring is full flag.

15.4.5.0.0.26 Field Documentation

15.4.5.0.0.26.1 `enet_frame_info_t* enet_tx_dirty_ring_t::txDirtyBase`

15.4.5.0.0.26.2 `uint16_t enet_tx_dirty_ring_t::txGenIdx`

15.4.5.0.0.26.3 `uint16_t enet_tx_dirty_ring_t::txConsumIdx`

15.4.5.0.0.26.4 `uint16_t enet_tx_dirty_ring_t::txRingLen`

15.4.5.0.0.26.5 `bool enet_tx_dirty_ring_t::isFull`

15.4.6 struct enet_buffer_config_t

Note that for the internal DMA requirements, the buffers have a corresponding alignment requirements.

1. The aligned receive and transmit buffer size must be evenly divisible by ENET_BUFF_ALIGNMENT. when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.
2. The aligned transmit and receive buffer descriptor start address must be at least 64 bit aligned. However, it's recommended to be evenly divisible by ENET_BUFF_ALIGNMENT. buffer descriptors should be put in non-cacheable region when cache is enabled.
3. The aligned transmit and receive data buffer start address must be evenly divisible by ENET_BUFF_ALIGNMENT. Receive buffers should be continuous with the total size equal to "rxBdNumber * rxBuffSizeAlign". Transmit buffers should be continuous with the total size equal to "txBdNumber * txBuffSizeAlign". when the data buffers are in cacheable region when cache is enabled, all those size should be aligned to the maximum value of "ENET_BUFF_ALIGNMENT" and the cache line size.

Data Fields

- `uint16_t rxBdNumber`
Receive buffer descriptor number.
- `uint16_t txBdNumber`
Transmit buffer descriptor number.
- `uint16_t rxBuffSizeAlign`
Aligned receive data buffer size.
- `uint16_t txBuffSizeAlign`
Aligned transmit data buffer size.
- volatile `enet_rx_bd_struct_t * rxBdStartAddrAlign`
Aligned receive buffer descriptor start address: should be non-cacheable.
- volatile `enet_tx_bd_struct_t * txBdStartAddrAlign`
Aligned transmit buffer descriptor start address: should be non-cacheable.
- `uint8_t * rxBufferAlign`
Receive data buffer start address.
- `uint8_t * txBufferAlign`
Transmit data buffer start address.
- `bool rxMaintainEnable`

Data Structure Documentation

- *Receive buffer cache maintain.*
• bool [txMaintainEnable](#)
- *Transmit buffer cache maintain.*
• [enet_frame_info_t](#) * [txFrameInfo](#)
• *Transmit frame information start address.*

15.4.6.0.0.27 Field Documentation

- 15.4.6.0.0.27.1 uint16_t [enet_buffer_config_t::rxBdNumber](#)
- 15.4.6.0.0.27.2 uint16_t [enet_buffer_config_t::txBdNumber](#)
- 15.4.6.0.0.27.3 uint16_t [enet_buffer_config_t::rxBuffSizeAlign](#)
- 15.4.6.0.0.27.4 uint16_t [enet_buffer_config_t::txBuffSizeAlign](#)
- 15.4.6.0.0.27.5 volatile [enet_rx_bd_struct_t](#)* [enet_buffer_config_t::rxBdStartAddrAlign](#)
- 15.4.6.0.0.27.6 volatile [enet_tx_bd_struct_t](#)* [enet_buffer_config_t::txBdStartAddrAlign](#)
- 15.4.6.0.0.27.7 uint8_t* [enet_buffer_config_t::rxBufferAlign](#)
- 15.4.6.0.0.27.8 uint8_t* [enet_buffer_config_t::txBufferAlign](#)
- 15.4.6.0.0.27.9 bool [enet_buffer_config_t::rxMaintainEnable](#)
- 15.4.6.0.0.27.10 bool [enet_buffer_config_t::txMaintainEnable](#)
- 15.4.6.0.0.27.11 [enet_frame_info_t](#)* [enet_buffer_config_t::txFrameInfo](#)

15.4.7 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.*

15.4.7.0.0.28 Field Documentation

- 15.4.7.0.0.28.1 `uint8_t enet_intcoalesce_config_t::txCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- 15.4.7.0.0.28.2 `uint16_t enet_intcoalesce_config_t::txCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`
- 15.4.7.0.0.28.3 `uint8_t enet_intcoalesce_config_t::rxCoalesceFrameCount[FSL_FEATURE_ENET_QUEUE]`
- 15.4.7.0.0.28.4 `uint16_t enet_intcoalesce_config_t::rxCoalesceTimeCount[FSL_FEATURE_ENET_QUEUE]`

15.4.8 struct enet_avb_config_t

This is used for to configure the extended ring 1 and ring 2.

1. The classification match format is $(CMP3 \ll 12) \mid (CMP2 \ll 8) \mid (CMP1 \ll 4) \mid CMP0$. 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, $BW \text{ fraction} = 1 / (1 + 512/idleSlope)$. For avb configuration, the BW fraction of Class 1 and Class 2 combined must not exceed 0.75.

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 certian bandwidth fraction.

15.4.8.0.0.29 Field Documentation

- 15.4.8.0.0.29.1 `uint16_t enet_avb_config_t::rxClassifyMatch[FSL_FEATURE_ENET_QUEUE-1]`
- 15.4.8.0.0.29.2 `enet_idle_slope_t enet_avb_config_t::idleSlope[FSL_FEATURE_ENET_QUEUE-1]`

15.4.9 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_tx_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.
- [enet_intcoalesce_config_t](#) * [intCoalesceCfg](#)
If the interrupt coalesce is not required in the ring n(0,1,2), please set to NULL.

- uint8_t [ringNum](#)
Number of used rings.

15.4.9.0.0.30 Field Documentation

15.4.9.0.0.30.1 uint32_t enet_config_t::macSpecialConfig

A logical OR of "enet_special_control_flag_t".

15.4.9.0.0.30.2 uint32_t enet_config_t::interrupt

A logical OR of "enet_interrupt_enable_t".

15.4.9.0.0.30.3 uint16_t enet_config_t::rxMaxFrameLen

15.4.9.0.0.30.4 enet_mii_mode_t enet_config_t::miiMode

15.4.9.0.0.30.5 enet_mii_speed_t enet_config_t::miiSpeed

15.4.9.0.0.30.6 enet_mii_duplex_t enet_config_t::miiDuplex

15.4.9.0.0.30.7 uint8_t enet_config_t::rxAccelerConfig

15.4.9.0.0.30.8 uint8_t enet_config_t::txAccelerConfig

15.4.9.0.0.30.9 uint16_t enet_config_t::pauseDuration

15.4.9.0.0.30.10 uint8_t enet_config_t::rxFifoEmptyThreshold

15.4.9.0.0.30.11 uint8_t enet_config_t::rxFifoFullThreshold

15.4.9.0.0.30.12 uint8_t enet_config_t::txFifoWatermark

15.4.9.0.0.30.13 enet_intcoalesce_config_t* enet_config_t::intCoalesceCfg

15.4.9.0.0.30.14 uint8_t enet_config_t::ringNum

default with 1 – single ring.

15.4.10 struct enet_tx_bd_ring_t

Data Fields

- volatile [enet_tx_bd_struct_t](#) * txBdBase
Buffer descriptor base address pointer.
- uint16_t [txGenIdx](#)
The current available transmit buffer descriptor pointer.
- uint16_t [txConsumIdx](#)
Transmit consume index.

Data Structure Documentation

- volatile uint16_t [txDescUsed](#)
Transmit descriptor used number.
- uint16_t [txRingLen](#)
Transmit ring length.

15.4.10.0.0.31 Field Documentation

15.4.10.0.0.31.1 volatile enet_tx_bd_struct_t* enet_tx_bd_ring_t::txBdBase

15.4.10.0.0.31.2 uint16_t enet_tx_bd_ring_t::txGenIdx

15.4.10.0.0.31.3 uint16_t enet_tx_bd_ring_t::txConsumeIdx

15.4.10.0.0.31.4 volatile uint16_t enet_tx_bd_ring_t::txDescUsed

15.4.10.0.0.31.5 uint16_t enet_tx_bd_ring_t::txRingLen

15.4.11 struct enet_rx_bd_ring_t

Data Fields

- volatile [enet_rx_bd_struct_t](#) * [rxBdBase](#)
Buffer descriptor base address pointer.
- uint16_t [rxGenIdx](#)
The current available receive buffer descriptor pointer.
- uint16_t [rxRingLen](#)
Receive ring length.

15.4.11.0.0.32 Field Documentation

15.4.11.0.0.32.1 volatile enet_rx_bd_struct_t* enet_rx_bd_ring_t::rxBdBase

15.4.11.0.0.32.2 uint16_t enet_rx_bd_ring_t::rxGenIdx

15.4.11.0.0.32.3 uint16_t enet_rx_bd_ring_t::rxRingLen

15.4.12 struct _enet_handle

Data Fields

- [enet_rx_bd_ring_t rxBdRing](#) [FSL_FEATURE_ENET_QUEUE]
Receive buffer descriptor.
- [enet_tx_bd_ring_t txBdRing](#) [FSL_FEATURE_ENET_QUEUE]
Transmit buffer descriptor.
- uint16_t [rxBuffSizeAlign](#) [FSL_FEATURE_ENET_QUEUE]
Receive buffer size alignment.
- uint16_t [txBuffSizeAlign](#) [FSL_FEATURE_ENET_QUEUE]
Transmit buffer size alignment.
- bool [rxMaintainEnable](#) [FSL_FEATURE_ENET_QUEUE]

- *Receive buffer cache maintain.*
bool **txMaintainEnable** [FSL_FEATURE_ENET_QUEUE]
- *Transmit buffer cache maintain.*
uint8_t **ringNum**
Number of used rings.
- **enet_callback_t** *callback*
Callback function.
- void * **userData**
Callback function parameter.
- **enet_tx_dirty_ring_t** **txDirtyRing** [FSL_FEATURE_ENET_QUEUE]
Ring to store tx frame information.
- bool **TxReclaimEnable** [FSL_FEATURE_ENET_QUEUE]
Tx reclaim enable flag.

15.4.12.0.0.33 Field Documentation

15.4.12.0.0.33.1 **enet_rx_bd_ring_t** **enet_handle_t::rxBdRing**[FSL_FEATURE_ENET_QUEUE]

15.4.12.0.0.33.2 **enet_tx_bd_ring_t** **enet_handle_t::txBdRing**[FSL_FEATURE_ENET_QUEUE]

15.4.12.0.0.33.3 **uint16_t** **enet_handle_t::rxBuffSizeAlign**[FSL_FEATURE_ENET_QUEUE]

15.4.12.0.0.33.4 **uint16_t** **enet_handle_t::txBuffSizeAlign**[FSL_FEATURE_ENET_QUEUE]

15.4.12.0.0.33.5 **bool** **enet_handle_t::rxMaintainEnable**[FSL_FEATURE_ENET_QUEUE]

15.4.12.0.0.33.6 **bool** **enet_handle_t::txMaintainEnable**[FSL_FEATURE_ENET_QUEUE]

15.4.12.0.0.33.7 **uint8_t** **enet_handle_t::ringNum**

15.4.12.0.0.33.8 **enet_callback_t** **enet_handle_t::callback**

15.4.12.0.0.33.9 **void*** **enet_handle_t::userData**

15.4.12.0.0.33.10 **enet_tx_dirty_ring_t** **enet_handle_t::txDirtyRing**[FSL_FEATURE_ENET_QUEUE]

15.4.12.0.0.33.11 **bool** **enet_handle_t::TxReclaimEnable**[FSL_FEATURE_ENET_QUEUE]

Macro Definition Documentation

15.5.1 **#define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))**

Version 2.3.0.

- 15.5.2 **#define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U**
- 15.5.3 **#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U**
- 15.5.4 **#define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U**
- 15.5.5 **#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U**
- 15.5.6 **#define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U**
- 15.5.7 **#define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U**
- 15.5.8 **#define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U**
- 15.5.9 **#define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U**
- 15.5.10 **#define ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK 0x0020U**
- 15.5.11 **#define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U**
- 15.5.12 **#define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U**
- 15.5.13 **#define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U**
- 15.5.14 **#define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U**
- 15.5.15 **#define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U**
- 15.5.16 **#define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER1_MASK 0x4000U**
- 15.5.17 **#define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U**
- 15.5.18 **#define ENET_BUFFDESCRIPTOR_TX_SOFTOWENER2_MASK 0x1000U**
- 15.5.19 **#define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U**
- 15.5.20 **#define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U**
- 15.5.21 **#define ENET_BUFFDESCRIPTOR_RX_ERR_MASK**

Macro Definition Documentation

```
(ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK |  
    ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK | \  
    ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK |  
    ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK |  
    ENET_BUFFDESCRIPTOR_RX_CRC_MASK)
```

15.5.22 #define ENET_FRAME_MAX_FRAMELEN 1518U

15.5.23 #define ENET_FIFO_MIN_RX_FULL 5U

15.5.24 #define ENET_RX_MIN_BUFFERSIZE 256U

**15.5.25 #define ENET_PHY_MAXADDRESS (ENET_MMFR_PA_MASK >>
 ENET_MMFR_PA_SHIFT)**

15.5.26 #define ENET_TX_INTERRUPT

Value:

```
((uint32_t)kENET_TxFrameInterrupt | (uint32_t)  
    kENET_TxBufferInterrupt | (uint32_t)  
    kENET_TxFrame1Interrupt | \  
    (uint32_t)kENET_TxBuffer1Interrupt | (uint32_t)  
    kENET_TxFrame2Interrupt | \  
    (uint32_t)kENET_TxBuffer2Interrupt)
```

15.5.27 #define ENET_RX_INTERRUPT

Value:

```
((uint32_t)kENET_RxFrameInterrupt | (uint32_t)  
    kENET_RxBufferInterrupt | (uint32_t)  
    kENET_RxFrame1Interrupt | \  
    (uint32_t)kENET_RxBuffer1Interrupt | (uint32_t)  
    kENET_RxFrame2Interrupt | \  
    (uint32_t)kENET_RxBuffer2Interrupt)
```

**15.5.28 #define ENET_TS_INTERRUPT ((uint32_t)kENET_TsTimerInterrupt |
 (uint32_t)kENET_TsAvailInterrupt)**

15.5.29 #define ENET_ERR_INTERRUPT

Value:

```
((uint32_t)kENET_BabrInterrupt | (uint32_t)
kENET_BabtInterrupt | (uint32_t)kENET_EBuserInterrupt | \
(uint32_t)kENET_LateCollisionInterrupt | (uint32_t)
kENET_RetryLimitInterrupt | \
(uint32_t)kENET_UnderrunInterrupt | (uint32_t)
kENET_PayloadRxInterrupt)
```

15.5.30 #define ENET_TX_LAST_BD_FLAG 0x01U

15.5.31 #define ENET_TX_TIMESTAMP_FLAG 0x02U

Typedef Documentation

15.6.1 typedef void(* enet_callback_t)(ENET_Type *base, enet_handle_t *handle, uint32_t ringId, enet_event_t event, enet_frame_info_t *frameInfo, void *userData)

15.6.2 typedef void(* enet_isr_ring_t)(ENET_Type *base, enet_handle_t *handle, uint32_t ringId)

Enumeration Type Documentation

15.7.1 anonymous enum

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.

15.7.2 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.

Enumeration Type Documentation

15.7.3 enum enet_mii_speed_t

Notice: "kENET_MiiSpeed1000M" only supported when mii mode is "kENET_RgmiiMode".

Enumerator

kENET_MiiSpeed10M Speed 10 Mbps.
kENET_MiiSpeed100M Speed 100 Mbps.
kENET_MiiSpeed1000M Speed 1000M bps.

15.7.4 enum enet_mii_duplex_t

Enumerator

kENET_MiiHalfDuplex Half duplex mode.
kENET_MiiFullDuplex Full duplex mode.

15.7.5 enum enet_mii_write_t

Enumerator

kENET_MiiWriteNoCompliant Write frame operation, but not MII-compliant.
kENET_MiiWriteValidFrame Write frame operation for a valid MII management frame.

15.7.6 enum enet_mii_read_t

Enumerator

kENET_MiiReadValidFrame Read frame operation for a valid MII management frame.
kENET_MiiReadNoCompliant Read frame operation, but not MII-compliant.

15.7.7 enum enet_mii_extend_opcode

Enumerator

kENET_MiiAddrWrite_C45 Address Write operation.
kENET_MiiWriteFrame_C45 Write frame operation for a valid MII management frame.
kENET_MiiReadFrame_C45 Read frame operation for a valid MII management frame.

15.7.8 enum enet_special_control_flag_t

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the [enet_config_t](#). The kENET_ControlStoreAndFwdDisable is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure rxFifoFullThreshold and txFifoWatermark in the [enet_config_t](#).

Enumerator

kENET_ControlFlowControlEnable Enable ENET flow control: pause frame.
kENET_ControlRxPayloadCheckEnable Enable ENET receive payload length check.
kENET_ControlRxPadRemoveEnable Padding is removed from received frames.
kENET_ControlRxBroadCastRejectEnable Enable broadcast frame reject.
kENET_ControlMacAddrInsert Enable MAC address insert.
kENET_ControlStoreAndFwdDisable Enable FIFO store and forward.
kENET_ControlSMIPreambleDisable Enable SMI preamble.
kENET_ControlPromiscuousEnable Enable promiscuous mode.
kENET_ControlMIILoopEnable Enable ENET MII loop back.
kENET_ControlVLANTagEnable Enable normal VLAN (single vlan tag).
kENET_ControlSVLANEnable Enable S-VLAN.
kENET_ControlVLANUseSecondTag Enable extracting the second vlan tag for further processing.

15.7.9 enum enet_interrupt_enable_t

This enumeration uses one-hot 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.

Enumeration Type Documentation

kENET_RxFlush2Interrupt Rx DMA ring2 flush indication.
kENET_RxFlush1Interrupt Rx DMA ring1 flush indication.
kENET_RxFlush0Interrupt RX DMA ring0 flush indication.
kENET_TxFrame2Interrupt Tx frame interrupt for Tx ring/class 2.
kENET_TxBuffer2Interrupt Tx buffer interrupt for Tx ring/class 2.
kENET_RxFrame2Interrupt Rx frame interrupt for Rx ring/class 2.
kENET_RxBuffer2Interrupt Rx buffer interrupt for Rx ring/class 2.
kENET_TxFrame1Interrupt Tx frame interrupt for Tx ring/class 1.
kENET_TxBuffer1Interrupt Tx buffer interrupt for Tx ring/class 1.
kENET_RxFrame1Interrupt Rx frame interrupt for Rx ring/class 1.
kENET_RxBuffer1Interrupt Rx buffer interrupt for Rx ring/class 1.
kENET_TsAvailInterrupt TS AVAIL interrupt source for PTP.
kENET_TsTimerInterrupt TS WRAP interrupt source for PTP.

15.7.10 enum enet_event_t

Enumerator

kENET_RxEvent Receive event.
kENET_TxEvent Transmit event.
kENET_ErrEvent Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .
kENET_WakeUpEvent Wake up from sleep mode event.
kENET_TimeStampEvent Time stamp event.
kENET_TimeStampAvailEvent Time stamp available event.

15.7.11 enum enet_idle_slope_t

Enumerator

kENET_IdleSlope1 The bandwidth fraction is about 0.002.
kENET_IdleSlope2 The bandwidth fraction is about 0.003.
kENET_IdleSlope4 The bandwidth fraction is about 0.008.
kENET_IdleSlope8 The bandwidth fraction is about 0.02.
kENET_IdleSlope16 The bandwidth fraction is about 0.03.
kENET_IdleSlope32 The bandwidth fraction is about 0.06.
kENET_IdleSlope64 The bandwidth fraction is about 0.11.
kENET_IdleSlope128 The bandwidth fraction is about 0.20.
kENET_IdleSlope256 The bandwidth fraction is about 0.33.
kENET_IdleSlope384 The bandwidth fraction is about 0.43.
kENET_IdleSlope512 The bandwidth fraction is about 0.50.
kENET_IdleSlope640 The bandwidth fraction is about 0.56.
kENET_IdleSlope768 The bandwidth fraction is about 0.60.
kENET_IdleSlope896 The bandwidth fraction is about 0.64.

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.

15.7.12 enum enet_tx_accelerator_t

Enumerator

kENET_TxAccellShift16Enabled Transmit FIFO shift-16.
kENET_TxAccellIpCheckEnabled Insert IP header checksum.
kENET_TxAccelProtoCheckEnabled Insert protocol checksum.

15.7.13 enum enet_rx_accelerator_t

Enumerator

kENET_RxAccelPadRemoveEnabled Padding removal for short IP frames.
kENET_RxAccellIpCheckEnabled Discard with wrong IP header checksum.
kENET_RxAccelProtoCheckEnabled Discard with wrong protocol checksum.
kENET_RxAccelMacCheckEnabled Discard with Mac layer errors.
kENET_RxAccellisShift16Enabled Receive FIFO shift-16.

Function Documentation

15.8.1 uint32_t ENET_GetInstance (ENET_Type * *base*)

Parameters

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

Returns

ENET instance.

15.8.2 void ENET_GetDefaultConfig (enet_config_t * *config*)

The purpose of this API is to get the default ENET MAC controller configure structure for [ENET_Init\(\)](#). User may use the initialized structure unchanged in [ENET_Init\(\)](#), or modify some fields of the structure before calling [ENET_Init\(\)](#). Example:

```
enet_config_t config;
ENET_GetDefaultConfig(&config);
```

Function Documentation

Parameters

<i>config</i>	The ENET mac controller configuration structure pointer.
---------------	--

15.8.3 void ENET_Up (ENET_Type * *base*, enet_handle_t * *handle*, const enet_config_t * *config*, const enet_buffer_config_t * *bufferConfig*, uint8_t * *macAddr*, uint32_t *srcClock_Hz*)

This function initializes the module with the ENET configuration.

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET_Up\(\)](#).

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	ENET handler pointer.
<i>config</i>	ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
<i>bufferConfig</i>	ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.

<i>macAddr</i>	ENET mac address of Ethernet device. This MAC address should be provided.
<i>srcClock_Hz</i>	The internal module clock source for MII clock.

15.8.4 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.

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET_Init\(\)](#).

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	ENET handler pointer.
<i>config</i>	ENET mac configuration structure pointer. The "enet_config_t" type mac configuration return from ENET_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods.
<i>bufferConfig</i>	ENET buffer configuration structure pointer. The buffer configuration should be prepared for ENET Initialization. It is the start address of "ringNum" enet_buffer_config structures. To support added multi-ring features in some soc and compatible with the previous enet driver version. For single ring supported, this bufferConfig is a buffer configure structure pointer, for multi-ring supported and used case, this bufferConfig pointer should be a buffer configure structure array pointer.

Function Documentation

<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.

15.8.5 void ENET_Down (ENET_Type * *base*)

This function disables the ENET module.

Parameters

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

15.8.6 void ENET_Deinit (ENET_Type * *base*)

This function gates the module clock, clears ENET interrupts, and disables the ENET module.

Parameters

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

15.8.7 static void ENET_Reset (ENET_Type * *base*) [inline], [static]

This function restores the ENET module to reset state. Note that this function sets all registers to reset state. As a result, the ENET module can't work after calling this function.

Parameters

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

15.8.8 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.

15.8.9 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"> • true Enables the preamble. • false Disables the preamble.

15.8.10 static bool ENET_GetSMI (ENET_Type * *base*) [inline], [static]

This API is used to get the SMI configuration to check whether the MII management interface has been set.

Parameters

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

Returns

The SMI setup status true or false.

15.8.11 static uint32_t ENET_ReadSMIData (ENET_Type * *base*) [inline], [static]

Function Documentation

Parameters

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

Returns

The data read from PHY

15.8.12 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.

15.8.13 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.

15.8.14 void ENET_StartExtC45SMIRead (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*)

Deprecated Do not use this function. It has been superseded by [ENET_StartExtC45SMIWriteReg](#) and [ENET_StartExtC45SMIReadData](#).

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.

15.8.15 void ENET_StartExtC45SMIWrite (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*, uint32_t *data*)

Deprecated Do not use this function. It has been superceded by [ENET_StartExtC45SMIWriteReg](#) and [ENET_StartExtC45SMIWriteData](#).

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.

15.8.16 void ENET_StartExtC45SMIWriteReg (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.

Function Documentation

15.8.17 void ENET_StartExtC45SMIWriteData (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*, uint32_t *data*)

After writing MMFR register, we need to check whether the transmission is over. This is an example for whole procedure of clause 45 MDIO write.

```
* ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
* ENET_StartExtC45SMIWriteReg(base, phyAddr, phyReg);
* while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
* {
* }
* ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
* ENET_StartExtC45SMIWriteData(base, phyAddr, phyReg, data);
* while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
* {
* }
* ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*
```

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.

15.8.18 void ENET_StartExtC45SMIReadData (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*)

After writing MMFR register, we need to check whether the transmission is over. This is an example for whole procedure of clause 45 MDIO read.

```
* uint32_t data;
* ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
* ENET_StartExtC45SMIWriteReg(base, phyAddr, phyReg);
* while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
* {
* }
* ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
* ENET_StartExtC45SMIReadData(base, phyAddr, phyReg);
* while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
* {
* }
* ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
* data = ENET_ReadSMIData(base);
*
```


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.

15.8.19 static void ENET_SetRGMIIIClockDelay (ENET_Type * *base*, bool *txEnabled*, bool *rxEnabled*) [inline], [static]

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.

15.8.20 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.

15.8.21 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.

15.8.22 void ENET_AddMulticastGroup (ENET_Type * *base*, uint8_t * *address*)

Function Documentation

Parameters

<i>base</i>	ENET peripheral base address.
<i>address</i>	The six-byte multicast group address which is provided by application.

15.8.23 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.

15.8.24 static void ENET_ActiveRead (ENET_Type * *base*) [inline], [static]

This function is to active the enet read process.

Note

This must be called after the MAC configuration and state are ready. It must be called after the [ENET_Init\(\)](#) and [ENET_Ptp1588Configure\(\)](#). This should be called when the ENET receive required.

Parameters

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

15.8.25 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.

15.8.26 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.

15.8.27 static void ENET_EnableInterrupts (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of enumeration members. See [enet_interrupt_enable_t](#). For example, to enable the TX frame interrupt and RX frame interrupt, do the following.

```
*  ENET_EnableInterrupts(ENET, kENET_TxFrameInterrupt |
*  kENET_RxFrameInterrupt);
*
```

Parameters

<i>base</i>	ENET peripheral base address.
<i>mask</i>	ENET interrupts to enable. This is a logical OR of the enumeration enet_interrupt_enable_t .

15.8.28 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.

Function Documentation

```
* 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 .

15.8.29 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](#).

15.8.30 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 .

15.8.31 void ENET_SetRxISRHandler (ENET_Type * *base*, [enet_isr_ring_t](#) *ISRHandler*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>ISRHandler</i>	The handler to install.

15.8.32 void ENET_SetTxISRHandler (ENET_Type * *base*, enet_isr_ring_t *ISRHandler*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>ISRHandler</i>	The handler to install.

15.8.33 void ENET_SetErrISRHandler (ENET_Type * *base*, enet_isr_t *ISRHandler*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>ISRHandler</i>	The handler to install.

15.8.34 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.

Function Documentation

15.8.35 void ENET_GetRxErrBeforeReadFrame (enet_handle_t * *handle*, enet_data_error_stats_t * *eErrorStatic*, uint8_t *ringId*)

This API must be called after the ENET_GetRxFrameSize and before the [ENET_ReadFrame\(\)](#). If the ENET_GetRxFrameSize returns kStatus_ENET_RxFrameError, the ENET_GetRxErrBeforeReadFrame can be used to get the exact error statistics. This is an example.

```
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (status == kStatus_ENET_RxFrameError)
*      {
*          Comments: Get the error information of the received frame.
*          ENET_GetRxErrBeforeReadFrame(&g_handle, &eErrStatic, 0);
*          Comments: update the receive buffer.
*          ENET_ReadFrame(EXAMPLE_ENET, &g_handle, NULL, 0);
*      }
*
```

Parameters

<i>handle</i>	The ENET handler structure pointer. This is the same handler pointer used in the ENET_Init.
<i>eErrorStatic</i>	The error statistics structure pointer.
<i>ringId</i>	The ring index, range from 0 ~ FSL_FEATURE_ENET_QUEUE - 1.

15.8.36 status_t ENET_GetRxFrameSize (enet_handle_t * *handle*, uint32_t * *length*, uint8_t *ringId*)

This function gets a received frame size from the ENET buffer descriptors.

Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling ENET_GetRxFrameSize, [ENET_ReadFrame\(\)](#) should be called to 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.
<i>ringId</i>	The ring index or ring number.

Return values

<i>kStatus_ENET_RxFrame-Empty</i>	No frame received. Should not call ENET_ReadFrame to read frame.
<i>kStatus_ENET_RxFrame-Error</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.

15.8.37 status_t ENET_ReadFrame (ENET_Type * *base*, enet_handle_t * *handle*, uint8_t * *data*, uint32_t *length*, uint8_t *ringId*, uint32_t * *ts*)

This function reads a frame (both the data and the length) from the ENET buffer descriptors. User can get timestamp through ts pointer if the ts is not NULL. Note that it doesn't store the timestamp in the receive timestamp queue. The ENET_GetRxFrameSize should be used to get the size of the prepared data buffer. This is an example:

```
*      uint32_t length;
*      enet_handle_t g_handle;
*      Comments: Get the received frame size firstly.
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (length != 0)
*      {
*          Comments: Allocate memory here with the size of "length"
*          uint8_t *data = memory allocate interface;
*          if (!data)
*          {
*              ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
*              Comments: Add the console warning log.
*          }
*          else
*          {
*              status = ENET_ReadFrame(ENET, &g_handle, data, length, 0, NULL);
*              Comments: Call stack input API to deliver the data to stack
*          }
*      }
*      else if (status == kStatus_ENET_RxFrameError)
*      {
*          Comments: Update the received buffer when a error frame is received.
*          ENET_ReadFrame(ENET, &g_handle, NULL, 0, 0, NULL);
*      }
*
```

Function Documentation

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to store the frame which memory size should be at least "length".
<i>length</i>	The size of the data buffer which is still the length of the received frame.
<i>ringId</i>	The ring index or ring number.
<i>ts</i>	The timestamp address to store received timestamp.

Returns

The execute status, successful or failure.

15.8.38 `status_t ENET_SendFrame (ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length, uint8_t ringId, bool tsFlag, void * context)`

Note

The CRC is automatically appended to the data. Input the data to send without the CRC.

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to send.
<i>length</i>	The length of the data to send.
<i>ringId</i>	The ring index or ring number.
<i>tsFlag</i>	Timestamp enable flag.
<i>context</i>	Used by user to handle some events after transmit over.

Return values

<i>kStatus_Success</i>	Send frame succeed.
<i>kStatus_ENET_TxFrame-Busy</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> .

15.8.39 **status_t ENET_SetTxReclaim (enet_handle_t * *handle*, bool *isEnabled*, uint8_t *ringId*)**

Note

This function must be called when no pending send frame action. Set enable if you want to reclaim context or timestamp in interrupt.

Parameters

<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>isEnabled</i>	Enable or disable flag.
<i>ringId</i>	The ring index or ring number.

Return values

<i>kStatus_Success</i>	Succeed to enable/disable Tx reclaim.
<i>kStatus_Fail</i>	Fail to enable/disable Tx reclaim.

15.8.40 **status_t ENET_GetRxBuffer (ENET_Type * *base*, enet_handle_t * *handle*, void ** *buffer*, uint32_t * *length*, uint8_t *ringId*, bool * *isLastBuff*, uint32_t * *ts*)**

This function can get the data address which stores frame. Then can analyze these data directly without doing any memory copy. When the frame locates in multiple BD buffer, need to repeat calling this function until *isLastBuff*=true (need to store the temp buf pointer everytime call this function). After finishing the analysis of this frame, call ENET_ReleaseRxBuffer to release rxbuff memory to DMA. This is an example:

```
*      uint32_t length;
*      uint8_t *buf = NULL;
*      uint32_t data_len = 0;
*      bool isLastBuff = false;
*      enet_handle_t g_handle;
*      status_t status;
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (length != 0)
*      {
```

Function Documentation

```
*      ENET_GetRxBuffer (EXAMPLE_ENET, &g_handle, &buf, &data_len, 0, &isLastBuff, NULL
*      );
*      ENET_ReleaseRxBuffer (EXAMPLE_ENET, &g_handle, buf, 0);
*      }
*
```

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>buffer</i>	The data buffer pointer to store the frame.
<i>length</i>	The size of the data buffer. If isLastBuff=false, it represents data length of this buffer. If isLastBuff=true, it represents data length of total frame.
<i>ringId</i>	The ring index, range from 0 ~ FSL_FEATURE_ENET_QUEUE - 1.
<i>isLastBuff</i>	The flag represents whether this buffer is the last buffer to store frame.
<i>ts</i>	The 1588 timestamp value, valid in last buffer.

Return values

<i>kStatus_Success</i>	Get receive buffer succeed.
<i>kStatus_ENET_RxFrame-Fail</i>	Get receive buffer fails, it's owned by application, should wait app to release this buffer.

15.8.41 void ENET_ReleaseRxBuffer (ENET_Type * *base*, enet_handle_t * *handle*, void * *buffer*, uint8_t *ringId*)

This function can release specified BD owned by application, meanwhile it may rearrange the BD to let the no-owned BDs always in back of the index of DMA transfer. So for the situation that releasing order is not same as the getting order, the rearrangement makes all ready BDs can be used by DMA.

Note

This function can't be interrupted by ENET_GetRxBuffer, so in application must make sure ENET_GetRxBuffer is called before or after this function. And this function itself isn't thread safe due to BD content exchanging.

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>buffer</i>	The buffer address to store frame, using it to find the correspond BD and release it.
<i>ringId</i>	The ring index, range from 0 ~ FSL_FEATURE_ENET_QUEUE - 1.

15.8.42 `status_t ENET_SendFrameZeroCopy (ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length, uint8_t ringId, bool tsFlag, void * context)`

Note

The CRC is automatically appended to the data. Input the data to send without the CRC. The frame must store in continuous memory and need to check the buffer start address alignment based on your device, otherwise it has issue or can't get highest DMA transmit speed.

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to send.
<i>length</i>	The length of the data to send.
<i>ringId</i>	The ring index or ring number.
<i>tsFlag</i>	Timestamp enable flag.
<i>context</i>	Used by user to handle some events after transmit over.

Return values

<i>kStatus_Success</i>	Send frame succeed.
<i>kStatus_ENET_TxFrame-Busy</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 kStatus-ENET_TxFrameBusy.

Function Documentation

15.8.43 `status_t ENET_SetTxBuffer (ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length, uint8_t ringId, uint8_t txFlag, void * context)`

This function only set one Tx BD everytime calls, all ready data will be sent out with last flag sets or gets error. Send frame succeeds with last flag sets, then you can get context from frameInfo in callback.

Note

The CRC is automatically appended to the data. Input the data to send without the CRC. And if doesn't succeed to call this function, user can't get context in frameInfo of callback.

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to send.
<i>length</i>	The length of the data to send.
<i>ringId</i>	The ring index, range from 0 ~ FSL_FEATURE_ENET_QUEUE - 1.
<i>txFlag</i>	This function uses timestamp enable flag, last BD flag.
<i>context</i>	Used by user to handle some events after transmit over.

Return values

<i>kStatus_Success</i>	Send frame succeed.
<i>kStatus_ENET_TxFrame-OverLen</i>	Buffer length isn't enough to store data.
<i>kStatus_ENET_TxFrame-Busy</i>	Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity.

15.8.44 `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.

15.8.45 void ENET_ReceiveIRQHandler (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.

15.8.46 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.
-------------	-------------------------------

15.8.47 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.
-------------	-------------------------------

15.8.48 void ENET_ErrorIRQHandler (ENET_Type * *base*, enet_handle_t * *handle*)

Function Documentation

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer.

15.8.49 void ENET_CommonFrame0IRQHandler (ENET_Type * *base*)

This is used for the combined tx/rx/error interrupt for single/multi-ring (frame 0).

Parameters

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

Chapter 16

ESAI: Enhanced Serial Audio Interface

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.

Typical use case

16.2.1 ESAI Send/Receive using an interrupt method

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

16.2.2 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](#) (4U)
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 {
 [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, [_esai_status_t](#).
- 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`  
*Master 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 }
- ESAI shifter register shift direction.*
- enum `esai_clock_direction_t` {  
`kESAI_ClockInput` = 0x0,  
`kESAI_ClockOutput` = 0x1 }
- ESAI clock direction.*
- enum {  
`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, \_esai\_interrupt\_enable\_t.*
- enum {  
`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, \_esai\_flags.*
- enum {

## Typical use case

```
kESAI_TransmitOddRegEmpty = ESAI_SAIRS_TODFE_MASK,
kESAI_TransmitEvenRegEmpty = ESAI_SAIRS_TEDE_MASK,
kESAI_TransmitRegEmpty = ESAI_SAIRS_TDE_MASK,
kESAI_TransmitUnderrunError = ESAI_SAIRS_TUE_MASK,
kESAI_TransmitFrameSync = ESAI_SAIRS_TFS_MASK,
kESAI_ReceiveOddRegFull = ESAI_SAIRS_RODF_MASK,
kESAI_ReceiveEvenRegFull = ESAI_SAIRS_RDF_MASK,
kESAI_ReceiveOverrunError = ESAI_SAIRS_ROE_MASK,
kESAI_ReceiveFrameSync = ESAI_SAIRS_RFS_MASK,
kESAI_SerialInputFlag2 = ESAI_SAIRS_IF2_MASK,
kESAI_SerialInputFlag1 = ESAI_SAIRS_IF1_MASK,
kESAI_SerialInputFlag0 = ESAI_SAIRS_IF0_MASK }
```

*SAI interface port status flag, \_esai\_sai\_flags.*

- 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 }

```

*esai slot word length*

## Driver version

- #define **FSL\_ESAI\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 0, 2))  
*Version 2.0.2.*

## 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 sectionMap)

## Typical use case

- *Enables/disables ESAI Tx.*  
void [ESAI\\_RxEnable](#) (ESAI\_Type \*base, uint8\_t sectionMap)
- *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.*  
void [ESAI\\_AnalysisSlot](#) (esai\_slot\_format\_t slotFormat, uint8\_t \*slotLen, uint8\_t \*dataLen)
- *Get the data length and slot length from the input.*  
uint32\_t [ESAI\\_GetInstance](#) (ESAI\_Type \*base)
- *Get the instance number for ESAI.*

## 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 FIFO 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)  
*Tx interrupt handler.*

## Data Structure Documentation

### 16.3.1 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 shift direction, MSB or LSB.*
- bool [fsEarly](#)  
*If the frame sync one bit early.*
- bool [ifZeroPadding](#)  
*If padding zero.*
- bool [dataAlign](#)  
*Data left aligned or right aligned.*
- bool [fsOneBit](#)  
*If the frame sync one word length or one bit length.*
- uint8\_t [slotNum](#)  
*Slot number for the audio format.*

### 16.3.2 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](#)  
*Rx frame sync polarity.*
- [esai\\_clock\\_polarity\\_t rxSckPolarity](#)  
*Rx bit clock polarity.*
- [uint8\\_t txWatermark](#)  
*Tx transfer watermark.*
- [uint8\\_t rxWatermark](#)  
*Rx receive watermark.*

### 16.3.3 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 sectionMap](#)  
*The sections enabled, 0x1 means TE0 enabled, 0x2 means TE1 enabled, 0x4 means TE2, etc.*

### 16.3.4 struct esai\_transfer\_t

#### Data Fields

- [uint8\\_t \\* data](#)  
*Data start address to transfer.*
- [size\\_t dataSize](#)  
*Transfer size.*

## Enumeration Type Documentation

### 16.3.4.0.0.34 Field Documentation

#### 16.3.4.0.0.34.1 `uint8_t* esai_transfer_t::data`

#### 16.3.4.0.0.34.2 `size_t esai_transfer_t::dataSize`

### 16.3.5 `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.*
- `uint8_t slotLen`  
*Slot length of the audio data.*
- `uint8_t sectionMap`  
*Enabled section 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.*

## Macro Definition Documentation

### 16.4.1 `#define ESAI_XFER_QUEUE_SIZE (4U)`

## Enumeration Type Documentation

### 16.5.1 `anonymous enum`

#### 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.

### 16.5.2 enum esai\_mode\_t

Enumerator

***kESAI\_NormalMode*** Use normal mode.

***kESAI\_NetworkMode*** Network mode.

### 16.5.3 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.

### 16.5.4 enum esai\_master\_slave\_t

Enumerator

***kESAI\_Master*** Master mode.

***kESAI\_Slave*** Slave mode.

### 16.5.5 enum esai\_sync\_mode\_t

Enumerator

***kESAI\_ModeAsync*** Asynchronous mode.

***kESAI\_ModeSync*** Synchronous mode (with receiver or transmit)

### 16.5.6 enum esai\_clock\_polarity\_t

Enumerator

*kESAI\_ClockActiveHigh* Clock active while high.

*kESAI\_ClockActiveLow* Clock active while low.

### 16.5.7 enum esai\_shift\_direction\_t

Enumerator

*kESAI\_ShifterMSB* Data is shifted MSB first.

*kESAI\_ShifterLSB* Data is shifted LSB first.

### 16.5.8 enum esai\_clock\_direction\_t

Enumerator

*kESAI\_ClockInput* Clock direction is input.

*kESAI\_ClockOutput* Clock direction is output.

### 16.5.9 anonymous enum

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.

### 16.5.10 anonymous enum

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.

### 16.5.11 anonymous enum

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\_ReceiveOverrunError*** 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.

### 16.5.12 enum esai\_sample\_rate\_t

Enumerator

***kESAI\_SampleRate8KHz*** Sample rate 8000 Hz.  
***kESAI\_SampleRate11025Hz*** Sample rate 11025 Hz.  
***kESAI\_SampleRate12KHz*** Sample rate 12000 Hz.  
***kESAI\_SampleRate16KHz*** Sample rate 16000 Hz.  
***kESAI\_SampleRate22050Hz*** Sample rate 22050 Hz.  
***kESAI\_SampleRate24KHz*** Sample rate 24000 Hz.  
***kESAI\_SampleRate32KHz*** Sample rate 32000 Hz.  
***kESAI\_SampleRate44100Hz*** Sample rate 44100 Hz.  
***kESAI\_SampleRate48KHz*** Sample rate 48000 Hz.  
***kESAI\_SampleRate96KHz*** Sample rate 96000 Hz.

### 16.5.13 enum esai\_word\_width\_t

Enumerator

***kESAI\_WordWidth8bits*** Audio data width 8 bits.

## Function Documentation

*kESAI\_WordWidth16bits* Audio data width 16 bits.  
*kESAI\_WordWidth24bits* Audio data width 24 bits.  
*kESAI\_WordWidth32bits* Audio data width 32 bits.

### 16.5.14 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.  
*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.

## Function Documentation

### 16.6.1 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. |

**16.6.2 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 |
|---------------|-------------------------------------------|

**16.6.3 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 |
|-------------|-------------------|

**16.6.4 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. |

**16.6.5 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.

## Function Documentation

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | ESAI base pointer |
|-------------|-------------------|

### 16.6.6 void ESAI\_TxReset ( ESAI\_Type \* *base* )

This API only resets the core logic of tx and all tx sections.

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | ESAI base pointer |
|-------------|-------------------|

### 16.6.7 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 |
|-------------|-------------------|

### 16.6.8 static void ESAI\_TxResetFIFO ( ESAI\_Type \* *base* ) [inline], [static]

This function only resets the ESAI Tx FIFO.

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | ESAI base pointer |
|-------------|-------------------|

### 16.6.9 static void ESAI\_RxResetFIFO ( ESAI\_Type \* *base* ) [inline], [static]

This function only resets the ESAI Rx FIFO.

### Parameters

---

|             |                   |
|-------------|-------------------|
| <i>base</i> | ESAI base pointer |
|-------------|-------------------|

#### 16.6.10 void ESAI\_TxEnable ( ESAI\_Type \* *base*, uint8\_t *sectionMap* )

Parameters

|                   |                                                                                                                                                                          |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | ESAI base pointer                                                                                                                                                        |
| <i>sectionMap</i> | Which sections need to be enabled. 0 means all section disabled. This parameter can be a combination of each sections, every section N is 2 <sup>N</sup> in section map. |

#### 16.6.11 void ESAI\_RxEnable ( ESAI\_Type \* *base*, uint8\_t *sectionMap* )

Parameters

|                   |                                                                                                                                                                          |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | ESAI base pointer                                                                                                                                                        |
| <i>sectionMap</i> | Which sections need to be enabled. 0 means all section disabled. This parameter can be a combination of each sections, every section N is 2 <sup>N</sup> in section map. |

#### 16.6.12 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. |

#### 16.6.13 static void ESAI\_RxEnableFIFO ( ESAI\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

## Function Documentation

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | ESAI base pointer                               |
| <i>enable</i> | True means enable ESAI Rx, false means disable. |

**16.6.14 static void ESAI\_TxSetSlotMask ( 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 Tx. |

**16.6.15 static void ESAI\_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 |

**16.6.16 void ESAI\_AnalysisSlot ( esai\_slot\_format\_t *slotFormat*, uint8\_t \* *slotLen*,  
uint8\_t \* *dataLen* )**

This API sets the audio protocol defined by users.

Parameters

|                   |                                              |
|-------------------|----------------------------------------------|
| <i>slotFormat</i> | Slot type.                                   |
| <i>slotLen</i>    | Pointer to the return slot length value.     |
| <i>dataLen</i>    | Pointer to the return data length in a slot. |

**16.6.17 uint32\_t ESAI\_GetInstance ( ESAI\_Type \* *base* )**



## Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | ESAI base pointer. |
|-------------|--------------------|

**16.6.18** `static uint32_t ESAI_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.

**16.6.19** `static uint32_t ESAI_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.

**16.6.20** `static uint32_t ESAI_GetTxFIFOStatus ( ESAI_Type * base ) [inline],  
[static]`

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | ESAI base pointer |
|-------------|-------------------|

## Returns

ESAI Tx status flag value.

**16.6.21** `static uint32_t ESAI_GetRxFIFOStatus ( ESAI_Type * base ) [inline],  
[static]`

## Function Documentation

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | ESAI base pointer |
|-------------|-------------------|

### Returns

ESAI Rx status flag value.

**16.6.22** `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. |

**16.6.23** `static void ESAI_RxEnableInterrupts ( 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. |

**16.6.24** `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 <code>_esai_interrupt_enable_t</code> . |
|-------------|-------------------------------------------------------------------------------------------------------------|

#### 16.6.25 **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 <code>_esai_interrupt_enable_t</code> . |

#### 16.6.26 **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.

#### 16.6.27 **static uint32\_t ESAI\_RxGetDataRegisterAddress ( 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.

**16.6.28** 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.            |

### 16.6.29 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.            |

### 16.6.30 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.                   |

## Function Documentation

|             |                      |
|-------------|----------------------|
| <i>size</i> | Bytes to be written. |
|-------------|----------------------|

**16.6.31 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. |

**16.6.32 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.                                    |

**16.6.33 static uint32\_t ESAI\_ReadData ( ESAI\_Type \* *base*, uint32\_t *channel* )**  
**[inline], [static]**

Parameters

|                |                    |
|----------------|--------------------|
| <i>base</i>    | ESAI base pointer. |
| <i>channel</i> | Data channel used. |

Returns

Data in ESAI FIFO.

**16.6.34** 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.

## Function Documentation

### 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 |

### 16.6.35 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 |

### 16.6.36 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.

### 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`.

### 16.6.37 `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`.

### 16.6.38 `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.

## Function Documentation

### Parameters

|               |                                                                                 |
|---------------|---------------------------------------------------------------------------------|
| <i>base</i>   | ESAI base pointer                                                               |
| <i>handle</i> | pointer to <code>esai_handle_t</code> structure which stores the transfer state |
| <i>xfer</i>   | pointer to <a href="#">esai_transfer_t</a> 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.        |

### 16.6.39 `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 <code>esai_handle_t</code> structure which stores the transfer state |
| <i>xfer</i>   | pointer to <a href="#">esai_transfer_t</a> 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.        |

### 16.6.40 `status_t ESAI_TransferGetSendCount ( 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 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. |

### 16.6.41 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. |

### 16.6.42 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.

## Function Documentation

### Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | ESAI base pointer.                                                  |
| <i>handle</i> | pointer to esai_handle_t structure which stores the transfer state. |

### 16.6.43 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. |

### 16.6.44 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. |

### 16.6.45 void ESAI\_TransferRxHandleIRQ ( ESAI\_Type \* *base*, esai\_handle\_t \* *handle* )

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | ESAI base pointer.                  |
| <i>handle</i> | pointer to esai_handle_t structure. |

## ESAI eDMA Driver

### 16.7.1 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, 1, 0))  
*Version 2.1.0.*

#### 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.*

## 16.7.2 Data Structure Documentation

### 16.7.2.1 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 sectionMap](#)  
*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.*

### 16.7.2.1.0.35 Field Documentation

16.7.2.1.0.35.1 `uint8_t esai_edma_handle_t::nbytes`

16.7.2.1.0.35.2 `edma_tcd_t esai_edma_handle_t::tcd[ESAI_XFER_QUEUE_SIZE+1U]`

16.7.2.1.0.35.3 `esai_transfer_t esai_edma_handle_t::esaiQueue[ESAI_XFER_QUEUE_SIZE]`

16.7.2.1.0.35.4 `volatile uint8_t esai_edma_handle_t::queueUser`

### 16.7.3 Function Documentation

16.7.3.1 `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. |

**16.7.3.2 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. |

**16.7.3.3 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. |

**16.7.3.4 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. |

**16.7.3.5 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.           |

### 16.7.3.6 **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.            |

### 16.7.3.7 **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. |

### 16.7.3.8 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. |

### 16.7.3.9 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. |

### 16.7.3.10 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_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |



## Chapter 17

# FlexCAN: Flex Controller Area Network Driver

### 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

### 17.2.1 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.

### 17.2.2 Typical use case

#### 17.2.2.1 Message Buffer Send Operation

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

#### 17.2.2.2 Message Buffer Receive Operation

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

#### 17.2.2.3 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)

- *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 {  
`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),  
`kStatus_FLEXCAN_RxRemote` = MAKE\_STATUS(kStatusGroup\_FLEXCAN, 13) }  
*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,  
`kFLEXCAN_ClkSrc0` = 0x0U,  
`kFLEXCAN_ClkSrc1` = 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,`  
`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_FLTCONF_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 {`  
`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, 5, 2))  
*FlexCAN driver version.*

## Initialization and deinitialization

- void **FLEXCAN\_EnterFreezeMode** (CAN\_Type \*base)  
*Enter FlexCAN Freeze Mode.*
- void **FLEXCAN\_ExitFreezeMode** (CAN\_Type \*base)  
*Exit FlexCAN Freeze Mode.*
- uint32\_t **FLEXCAN\_GetInstance** (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 \*pTimingConfig)  
*Calculates the improved timing values by specific baudrates for classical CAN.*
- void **FLEXCAN\_Init** (CAN\_Type \*base, const flexcan\_config\_t \*pConfig, 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 \*pTimingConfig)  
*Calculates the improved timing values by specific baudrates for CANFD.*
- void **FLEXCAN\_FDInit** (CAN\_Type \*base, const flexcan\_config\_t \*pConfig, 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 \*pConfig)  
*Gets the default configuration structure.*

## Configuration.

- void **FLEXCAN\_SetTimingConfig** (CAN\_Type \*base, const flexcan\_timing\_config\_t \*pConfig)  
*Sets the FlexCAN protocol timing characteristic.*
- void **FLEXCAN\_SetFDTimingConfig** (CAN\_Type \*base, const flexcan\_timing\_config\_t \*pConfig)  
*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)

## FlexCAN Driver

- *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](#) \*pRxMbConfig, bool enable)
- *Configures a FlexCAN Receive Message Buffer.*  
void [FLEXCAN\\_SetFDRxMbConfig](#) (CAN\_Type \*base, uint8\_t mbIdx, const [flexcan\\_rx\\_mb\\_config\\_t](#) \*pRxMbConfig, bool enable)
- *Configures a FlexCAN Receive Message Buffer.*  
void [FLEXCAN\\_SetRxFifoConfig](#) (CAN\_Type \*base, const [flexcan\\_rx\\_fifo\\_config\\_t](#) \*pRxFifoConfig, 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\\_EnableRxFifoDMA](#) (CAN\_Type \*base, bool enable)  
*Enables or disables the FlexCAN Rx FIFO DMA request.*
- static uint32\_t [FLEXCAN\\_GetRxFifoHeadAddr](#) (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** \*pTxFrame)  
*Writes a FlexCAN Message to the Transmit Message Buffer.*
- **status\_t FLEXCAN\_ReadRxMb** (CAN\_Type \*base, uint8\_t mbIdx, **flexcan\_frame\_t** \*pRxFrame)  
*Reads a FlexCAN Message from Receive Message Buffer.*
- **status\_t FLEXCAN\_WriteFDTxMb** (CAN\_Type \*base, uint8\_t mbIdx, const **flexcan\_fd\_frame\_t** \*pTxFrame)  
*Writes a FlexCAN FD Message to the Transmit Message Buffer.*
- **status\_t FLEXCAN\_ReadFDRxMb** (CAN\_Type \*base, uint8\_t mbIdx, **flexcan\_fd\_frame\_t** \*pRxFrame)  
*Reads a FlexCAN FD Message from Receive Message Buffer.*
- **status\_t FLEXCAN\_ReadRxFifo** (CAN\_Type \*base, **flexcan\_frame\_t** \*pRxFrame)  
*Reads a FlexCAN Message from Rx FIFO.*

## Transactional

- **status\_t FLEXCAN\_TransferFDSendBlocking** (CAN\_Type \*base, uint8\_t mbIdx, **flexcan\_fd\_frame\_t** \*pTxFrame)  
*Performs a polling send transaction on the CAN bus.*
- **status\_t FLEXCAN\_TransferFDReceiveBlocking** (CAN\_Type \*base, uint8\_t mbIdx, **flexcan\_fd\_frame\_t** \*pRxFrame)  
*Performs a polling receive transaction on the CAN bus.*
- **status\_t FLEXCAN\_TransferFDSendNonBlocking** (CAN\_Type \*base, **flexcan\_handle\_t** \*handle, **flexcan\_mb\_transfer\_t** \*pMbXfer)  
*Sends a message using IRQ.*
- **status\_t FLEXCAN\_TransferFDReceiveNonBlocking** (CAN\_Type \*base, **flexcan\_handle\_t** \*handle, **flexcan\_mb\_transfer\_t** \*pMbXfer)  
*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** \*pTxFrame)  
*Performs a polling send transaction on the CAN bus.*
- **status\_t FLEXCAN\_TransferReceiveBlocking** (CAN\_Type \*base, uint8\_t mbIdx, **flexcan\_frame\_t** \*pRxFrame)  
*Performs a polling receive transaction on the CAN bus.*
- **status\_t FLEXCAN\_TransferReceiveFifoBlocking** (CAN\_Type \*base, **flexcan\_frame\_t** \*pRxFrame)  
*Performs a polling receive transaction from Rx FIFO on the CAN bus.*
- void **FLEXCAN\_TransferCreateHandle** (CAN\_Type \*base, **flexcan\_handle\_t** \*handle, **flexcan\_**

## FlexCAN Driver

`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 \*pMbXfer)  
*Sends a message using IRQ.*
- `status_t FLEXCAN_TransferReceiveNonBlocking` (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_mb\_transfer\_t \*pMbXfer)  
*Receives a message using IRQ.*
- `status_t FLEXCAN_TransferReceiveFifoNonBlocking` (CAN\_Type \*base, flexcan\_handle\_t \*handle, flexcan\_fifo\_transfer\_t \*pFifoXfer)  
*Receives a message from Rx FIFO using IRQ.*
- `uint32_t FLEXCAN_GetTimeStamp` (flexcan\_handle\_t \*handle, uint8\_t mbIdx)  
*Gets the detail index of Mailbox's Timestamp by handle.*
- `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.*



## 17.2.3 Data Structure Documentation

### 17.2.3.1 struct flexcan\_frame\_t

#### 17.2.3.1.0.36 Field Documentation

17.2.3.1.0.36.1 uint32\_t flexcan\_frame\_t::timestamp

17.2.3.1.0.36.2 uint32\_t flexcan\_frame\_t::length

17.2.3.1.0.36.3 uint32\_t flexcan\_frame\_t::type

17.2.3.1.0.36.4 uint32\_t flexcan\_frame\_t::format

17.2.3.1.0.36.5 uint32\_t flexcan\_frame\_t::\_\_pad0\_\_

17.2.3.1.0.36.6 uint32\_t flexcan\_frame\_t::idhit

17.2.3.1.0.36.7 uint32\_t flexcan\_frame\_t::id

17.2.3.1.0.36.8 uint32\_t flexcan\_frame\_t::dataWord0

17.2.3.1.0.36.9 uint32\_t flexcan\_frame\_t::dataWord1

17.2.3.1.0.36.10 uint8\_t flexcan\_frame\_t::dataByte3

17.2.3.1.0.36.11 uint8\_t flexcan\_frame\_t::dataByte2

17.2.3.1.0.36.12 uint8\_t flexcan\_frame\_t::dataByte1

17.2.3.1.0.36.13 uint8\_t flexcan\_frame\_t::dataByte0

17.2.3.1.0.36.14 uint8\_t flexcan\_frame\_t::dataByte7

17.2.3.1.0.36.15 uint8\_t flexcan\_frame\_t::dataByte6

17.2.3.1.0.36.16 uint8\_t flexcan\_frame\_t::dataByte5

17.2.3.1.0.36.17 uint8\_t flexcan\_frame\_t::dataByte4

### 17.2.3.2 struct flexcan\_fd\_frame\_t

#### 17.2.3.2.0.37 Field Documentation

17.2.3.2.0.37.1 uint32\_t flexcan\_fd\_frame\_t::timestamp

17.2.3.2.0.37.2 uint32\_t flexcan\_fd\_frame\_t::length

17.2.3.2.0.37.3 uint32\_t flexcan\_fd\_frame\_t::type

17.2.3.2.0.37.4 uint32\_t flexcan\_fd\_frame\_t::format

17.2.3.2.0.37.5 uint32\_t flexcan\_fd\_frame\_t::idhit

17.2.3.2.0.37.6 uint32\_t flexcan\_fd\_frame\_t::\_\_pad0\_\_



- `uint8_t rJumpwidth`  
*Clock Pre-scaler Division Factor.*
- `uint8_t phaseSeg1`  
*Re-sync Jump Width.*
- `uint8_t phaseSeg2`  
*Phase Segment 1.*
- `uint8_t propSeg`  
*Phase Segment 2.*
- `uint16_t fpreDivider`  
*Propagation Segment.*
- `uint8_t frJumpwidth`  
*Fast Clock Pre-scaler Division Factor.*
- `uint8_t fphaseSeg1`  
*Fast Re-sync Jump Width.*
- `uint8_t fphaseSeg2`  
*Fast Phase Segment 1.*
- `uint8_t fpropSeg`  
*Fast Phase Segment 2.*
- `uint8_t fpropSeg`  
*Fast Propagation Segment.*

#### 17.2.3.3.0.38 Field Documentation

17.2.3.3.0.38.1 `uint16_t flexcan_timing_config_t::preDivider`

17.2.3.3.0.38.2 `uint8_t flexcan_timing_config_t::rJumpwidth`

17.2.3.3.0.38.3 `uint8_t flexcan_timing_config_t::phaseSeg1`

17.2.3.3.0.38.4 `uint8_t flexcan_timing_config_t::phaseSeg2`

17.2.3.3.0.38.5 `uint8_t flexcan_timing_config_t::propSeg`

17.2.3.3.0.38.6 `uint16_t flexcan_timing_config_t::fpreDivider`

17.2.3.3.0.38.7 `uint8_t flexcan_timing_config_t::frJumpwidth`

17.2.3.3.0.38.8 `uint8_t flexcan_timing_config_t::fphaseSeg1`

17.2.3.3.0.38.9 `uint8_t flexcan_timing_config_t::fphaseSeg2`

17.2.3.3.0.38.10 `uint8_t flexcan_timing_config_t::fpropSeg`

#### 17.2.3.4 `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`

- *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.*  
[bool](#) disableSelfReception
- *Enable or Disable Self Reflection.*  
[bool](#) enableListenOnlyMode
- *Enable or Disable Listen Only Mode.*  
[bool](#) enableDoze
- *Enable or Disable Doze Mode.*

### 17.2.3.4.0.39 Field Documentation

17.2.3.4.0.39.1 [uint32\\_t](#) flexcan\_config\_t::baudRate

17.2.3.4.0.39.2 [uint32\\_t](#) flexcan\_config\_t::baudRateFD

17.2.3.4.0.39.3 [flexcan\\_clock\\_source\\_t](#) flexcan\_config\_t::clkSrc

17.2.3.4.0.39.4 [flexcan\\_wake\\_up\\_source\\_t](#) flexcan\_config\_t::wakeupSrc

17.2.3.4.0.39.5 [uint8\\_t](#) flexcan\_config\_t::maxMbNum

17.2.3.4.0.39.6 [bool](#) flexcan\_config\_t::enableLoopBack

17.2.3.4.0.39.7 [bool](#) flexcan\_config\_t::enableTimerSync

17.2.3.4.0.39.8 [bool](#) flexcan\_config\_t::enableSelfWakeup

17.2.3.4.0.39.9 [bool](#) flexcan\_config\_t::enableIndividMask

17.2.3.4.0.39.10 [bool](#) flexcan\_config\_t::disableSelfReception

17.2.3.4.0.39.11 [bool](#) flexcan\_config\_t::enableListenOnlyMode

17.2.3.4.0.39.12 [bool](#) flexcan\_config\_t::enableDoze

### 17.2.3.5 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 of Extend).*
- `flexcan_frame_type_t type`  
*CAN Frame Type(Data or Remote).*

### 17.2.3.5.0.40 Field Documentation

17.2.3.5.0.40.1 `uint32_t flexcan_rx_mb_config_t::id`

17.2.3.5.0.40.2 `flexcan_frame_format_t flexcan_rx_mb_config_t::format`

17.2.3.5.0.40.3 `flexcan_frame_type_t flexcan_rx_mb_config_t::type`

### 17.2.3.6 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.*

### 17.2.3.6.0.41 Field Documentation

17.2.3.6.0.41.1 `uint32_t* flexcan_rx_fifo_config_t::idFilterTable`

17.2.3.6.0.41.2 `uint8_t flexcan_rx_fifo_config_t::idFilterNum`

17.2.3.6.0.41.3 `flexcan_rx_fifo_filter_type_t flexcan_rx_fifo_config_t::idFilterType`

17.2.3.6.0.41.4 `flexcan_rx_fifo_priority_t flexcan_rx_fifo_config_t::priority`

### 17.2.3.7 struct flexcan\_mb\_transfer\_t

## Data Fields

- `flexcan_frame_t * frame`  
*The buffer of CAN Message to be transfer.*
- `uint8_t mbIdx`

## FlexCAN Driver

*The index of Message buffer used to transfer Message.*

### 17.2.3.7.0.42 Field Documentation

17.2.3.7.0.42.1 flexcan\_frame\_t\* flexcan\_mb\_transfer\_t::frame

17.2.3.7.0.42.2 uint8\_t flexcan\_mb\_transfer\_t::mbldx

### 17.2.3.8 struct flexcan\_fifo\_transfer\_t

#### Data Fields

- flexcan\_frame\_t \* frame  
*The buffer of CAN Message to be received from Rx FIFO.*

### 17.2.3.8.0.43 Field Documentation

17.2.3.8.0.43.1 flexcan\_frame\_t\* flexcan\_fifo\_transfer\_t::frame

### 17.2.3.9 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.*
- volatile uint32\_t timestamp [CAN\_WORD1\_COUNT]  
*Mailbox transfer timestamp.*

### 17.2.3.9.0.44 Field Documentation

17.2.3.9.0.44.1 `flexcan_transfer_callback_t flexcan_handle_t::callback`

17.2.3.9.0.44.2 `void* flexcan_handle_t::userData`

17.2.3.9.0.44.3 `flexcan_fd_frame_t* volatile flexcan_handle_t::mbFDFrameBuf[CAN_WORD1_COUNT]`

17.2.3.9.0.44.4 `flexcan_frame_t* volatile flexcan_handle_t::rxFifoFrameBuf`

17.2.3.9.0.44.5 `volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]`

17.2.3.9.0.44.6 `volatile uint8_t flexcan_handle_t::rxFifoState`

17.2.3.9.0.44.7 `volatile uint32_t flexcan_handle_t::timestamp[CAN_WORD1_COUNT]`

### 17.2.4 Macro Definition Documentation

17.2.4.1 `#define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))`

17.2.4.2 `#define FLEXCAN_ID_STD( id ) (((uint32_t)((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)`

Standard Frame ID helper macro.

17.2.4.3 `#define FLEXCAN_ID_EXT( id )`

Value:

```
((uint32_t)((uint32_t)(id)) << CAN_ID_EXT_SHIFT)) & \
(CAN_ID_EXT_MASK | CAN_ID_STD_MASK)
```

17.2.4.4 `#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.

17.2.4.5 `#define FLEXCAN_RX_MB_EXT_MASK( id, rtr, ide )`

Value:

## FlexCAN Driver

```
((uint32_t)((uint32_t)(rtr) << 31) | (uint32_t)((uint32_t)(ide) << 30)) | \
 FLEXCAN_ID_EXT(id)
```

### 17.2.4.6 #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.

### 17.2.4.7 #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)
```

### 17.2.4.8 #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)
```

### 17.2.4.9 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_HIGH( *id* ) (((uint32\_t)(id) & 0x7F8) << 21)

### 17.2.4.10 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_HIGH( *id* ) (((uint32\_t)(id) & 0x7F8) << 13)

### 17.2.4.11 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_MID\_LOW( *id* ) (((uint32\_t)(id) & 0x7F8) << 5)

### 17.2.4.12 #define FLEXCAN\_RX\_FIFO\_STD\_MASK\_TYPE\_C\_LOW( *id* ) (((uint32\_t)(id) & 0x7F8) >> 3)

### 17.2.4.13 #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)
```

**17.2.4.14 #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)) | \
 ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) << 1))
```

**17.2.4.15 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )****Value:**

```
((uint32_t)((uint32_t)(rtr) << 15) | (uint32_t)((uint32_t)(ide) << 14)) | \
 ((FLEXCAN_ID_EXT(id) & 0x1FFF8000) >> 15))
```

**17.2.4.16 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_HIGH( *id* ) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) << 3)****17.2.4.17 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_HIGH( *id* )****Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 5)
```

**17.2.4.18 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_MID\_LOW( *id* )****Value:**

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 13)
```

**17.2.4.19 #define FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_C\_LOW( *id* ) ((FLEXCAN\_ID\_EXT(id) & 0x1FE00000) >> 21)****17.2.4.20 #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.

### 17.2.4.21 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(\
 id, rtr, ide)
```

### 17.2.4.22 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(\
 id, rtr, ide)
```

### 17.2.4.23 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_HIGH( *id* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(\
 id)
```

### 17.2.4.24 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(\
 id)
```

### 17.2.4.25 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_MID\_LOW( *id* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(\
 id)
```

### 17.2.4.26 #define FLEXCAN\_RX\_FIFO\_STD\_FILTER\_TYPE\_C\_LOW( *id* )

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(\
 id)
```



**17.2.4.27 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_A( *id*, *rtr*, *ide*  
 ) FLEXCAN\_RX\_FIFO\_EXT\_MASK\_TYPE\_A(id, rtr, ide)**

**17.2.4.28 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_HIGH( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(\
 id, rtr, ide)
```

**17.2.4.29 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_B\_LOW( *id*, *rtr*, *ide* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(\
 id, rtr, ide)
```

**17.2.4.30 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(\
 id)
```

**17.2.4.31 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_HIGH( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(\
 id)
```

**17.2.4.32 #define FLEXCAN\_RX\_FIFO\_EXT\_FILTER\_TYPE\_C\_MID\_LOW( *id* )**

**Value:**

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(\
 id)
```

```
17.2.4.33 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(id
) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)
```

## 17.2.5 Typedef Documentation

```
17.2.5.1 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.

## 17.2.6 Enumeration Type Documentation

### 17.2.6.1 anonymous enum

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* UnHandled Interrupt asserted.  
*kStatus\_FLEXCAN\_RxRemote* Rx Remote Message Received in Mail box.

### 17.2.6.2 enum flexcan\_frame\_format\_t

Enumerator

*kFLEXCAN\_FrameFormatStandard* Standard frame format attribute.  
*kFLEXCAN\_FrameFormatExtend* Extend frame format attribute.

### 17.2.6.3 enum flexcan\_frame\_type\_t

Enumerator

***kFLEXCAN\_FrameTypeData*** Data frame type attribute.

***kFLEXCAN\_FrameTypeRemote*** Remote frame type attribute.

### 17.2.6.4 enum flexcan\_clock\_source\_t

**Deprecated** Do not use the **kFLEXCAN\_ClkSrcOs**. It has been superceded **kFLEXCAN\_ClkSrc0**  
Do not use the **kFLEXCAN\_ClkSrcPeri**. It has been superceded **kFLEXCAN\_ClkSrc1**

Enumerator

***kFLEXCAN\_ClkSrcOsc*** FlexCAN Protocol Engine clock from Oscillator.

***kFLEXCAN\_ClkSrcPeri*** FlexCAN Protocol Engine clock from Peripheral Clock.

***kFLEXCAN\_ClkSrc0*** FlexCAN Protocol Engine clock selected by user as SRC == 0.

***kFLEXCAN\_ClkSrc1*** FlexCAN Protocol Engine clock selected by user as SRC == 1.

### 17.2.6.5 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.

### 17.2.6.6 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.

### 17.2.6.7 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.

### 17.2.6.8 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 filter) 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.

### 17.2.6.9 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.

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.

### 17.2.6.10 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.

#### 17.2.6.11 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.

#### 17.2.6.12 anonymous enum

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.

### 17.2.7 Function Documentation

#### 17.2.7.1 void FLEXCAN\_EnterFreezeMode ( CAN\_Type \* *base* )

This function makes the FlexCAN work under Freeze Mode.

## Parameters

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

**17.2.7.2 void FLEXCAN\_ExitFreezeMode ( CAN\_Type \* *base* )**

This function makes the FlexCAN leave Freeze Mode.

## Parameters

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

**17.2.7.3 uint32\_t FLEXCAN\_GetInstance ( CAN\_Type \* *base* )**

## Parameters

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

## Returns

FlexCAN instance.

**17.2.7.4 bool FLEXCAN\_CalculateImprovedTimingValues ( uint32\_t *baudRate*, uint32\_t *sourceClock\_Hz*, flexcan\_timing\_config\_t \* *pTimingConfig* )**

## Parameters

|                       |                                                                        |
|-----------------------|------------------------------------------------------------------------|
| <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>pTimingConfig</i>  | Pointer to the FlexCAN timing configuration structure.                 |

## Returns

TRUE if timing configuration found, FALSE if failed to find configuration

## FlexCAN Driver

### 17.2.7.5 void FLEXCAN\_Init ( CAN\_Type \* *base*, const flexcan\_config\_t \* *pConfig*, 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_ClkSrc0;
* flexcanConfig.baudRate = 1000000U;
* flexcanConfig.maxMbNum = 16;
* flexcanConfig.enableLoopBack = false;
* flexcanConfig.enableSelfWakeup = false;
* flexcanConfig.enableIndividMask = false;
* flexcanConfig.enableDoze = false;
* flexcanConfig.disableSelfReception = false;
* flexcanConfig.enableListenOnlyMode = false;
* flexcanConfig.timingConfig = timingConfig;
* FLEXCAN_Init(CAN0, &flexcanConfig, 8000000UL);
*
```

#### Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                      |
| <i>pConfig</i>        | Pointer to the user-defined configuration structure.  |
| <i>sourceClock_Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz. |

### 17.2.7.6 bool FLEXCAN\_FDCalculatImprovedTimingValues ( uint32\_t *baudRate*, uint32\_t *baudRateFD*, uint32\_t *sourceClock\_Hz*, flexcan\_timing\_config\_t \* *pTimingConfig* )

#### 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>pTimingConfig</i>  | Pointer to the FlexCAN timing configuration structure.                 |

#### Returns

TRUE if timing configuration found, FALSE if failed to find configuration



### 17.2.7.7 void FLEXCAN\_FDInit ( CAN\_Type \* *base*, const flexcan\_config\_t \* *pConfig*, 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_ClkSrc0;
* flexcanConfig.baudRate = 1000000U;
* flexcanConfig.baudRateFD = 2000000U;
* flexcanConfig.maxMbNum = 16;
* flexcanConfig.enableLoopBack = false;
* flexcanConfig.enableSelfWakeup = false;
* flexcanConfig.enableIndividMask = false;
* flexcanConfig.disableSelfReception = false;
* flexcanConfig.enableListenOnlyMode = false;
* flexcanConfig.enableDoze = false;
* flexcanConfig.timingConfig = timingConfig;
* FLEXCAN_FDInit(CAN0, &flexcanConfig, 8000000UL,
* kFLEXCAN_16BperMB, false);
*
```

#### Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>base</i>           | FlexCAN peripheral base address.                      |
| <i>pConfig</i>        | Pointer to the user-defined configuration structure.  |
| <i>sourceClock_Hz</i> | FlexCAN Protocol Engine clock source frequency in Hz. |
| <i>dataSize</i>       | FlexCAN FD frame payload size.                        |
| <i>brs</i>            | If bitrate switch is enabled in FD mode.              |

### 17.2.7.8 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. |
|-------------|----------------------------------|

### 17.2.7.9 void FLEXCAN\_GetDefaultConfig ( flexcan\_config\_t \* *pConfig* )

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. `flexcanConfig->clkSrc = kFLEXCAN_ClkSrc0`; `flexcanConfig->baudRate = 1000000U`; `flexcanConfig->baudRateFD = 2000000U`; `flexcanConfig->maxMbNum = 16`; `flexcanConfig->enableLoopBack = false`; `flexcanConfig->enableSelfWakeup = false`; `flexcanConfig->enableIndividMask =`

## FlexCAN Driver

```
false; flexcanConfig->disableSelfReception = false; flexcanConfig->enableListenOnlyMode = false;
flexcanConfig->enableDoze = false; flexcanConfig.timingConfig = timingConfig;
```

## Parameters

|                |                                                 |
|----------------|-------------------------------------------------|
| <i>pConfig</i> | Pointer to the FlexCAN configuration structure. |
|----------------|-------------------------------------------------|

#### 17.2.7.10 void FLEXCAN\_SetTimingConfig ( CAN\_Type \* *base*, const flexcan\_timing\_config\_t \* *pConfig* )

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>pConfig</i> | Pointer to the timing configuration structure. |

#### 17.2.7.11 void FLEXCAN\_SetFDTimingConfig ( CAN\_Type \* *base*, const flexcan\_timing\_config\_t \* *pConfig* )

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>pConfig</i> | Pointer to the timing configuration structure. |

#### 17.2.7.12 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

---

## FlexCAN Driver

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.     |
| <i>mask</i> | Rx Message Buffer Global Mask value. |

### 17.2.7.13 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.       |

### 17.2.7.14 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.

Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>base</i>    | FlexCAN peripheral base address. |
| <i>maskIdx</i> | The Index of individual Mask.    |
| <i>mask</i>    | Rx Individual Mask value.        |

### 17.2.7.15 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> |

#### 17.2.7.16 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> |

#### 17.2.7.17 void FLEXCAN\_SetRxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_rx\_mb\_config\_t \* *pRxMbConfig*, 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>pRxMbConfig</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> |

**17.2.7.18** void FLEXCAN\_SetFDRxMbConfig ( CAN\_Type \* *base*, uint8\_t *mbldx*, const flexcan\_rx\_mb\_config\_t \* *pRxMbConfig*, 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>pRxMbConfig</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> |

#### 17.2.7.19 void FLEXCAN\_SetRxFifoConfig ( CAN\_Type \* *base*, const flexcan\_rx\_fifo\_config\_t \* *pRxFifoConfig*, bool *enable* )

This function configures the Rx FIFO with given Rx FIFO configuration.

## Parameters

|                      |                                                                                                                                      |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | FlexCAN peripheral base address.                                                                                                     |
| <i>pRxFifoConfig</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> |

#### 17.2.7.20 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](#).

### 17.2.7.21 `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> . |

**17.2.7.22** `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. |

**17.2.7.23** `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.

**17.2.7.24** `static void FLEXCAN_ClearMbStatusFlags ( CAN_Type * base, uint64_t mask )  
[inline], [static]`

This function clears the interrupt flags of a given Message Buffers.

## FlexCAN Driver

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

#### 17.2.7.25 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> . |

#### 17.2.7.26 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](#).

### Parameters

|             |                                                                                      |
|-------------|--------------------------------------------------------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_flexcan_interrupt_enable</a> . |

#### 17.2.7.27 static void FLEXCAN\_EnableMblInterrupts ( 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. |

**17.2.7.28** `static void FLEXCAN_DisableMbInterrupts ( CAN_Type * base, uint64_t mask )`  
`[inline], [static]`

This function disables the interrupts of given Message Buffers.

## FlexCAN Driver

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | FlexCAN peripheral base address.      |
| <i>mask</i> | The ORed FlexCAN Message Buffer mask. |

### 17.2.7.29 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. |

### 17.2.7.30 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.

### 17.2.7.31 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. |

**17.2.7.32** `status_t FLEXCAN_WriteTxMb ( CAN_Type * base, uint8_t mbldx, const flexcan_frame_t * pTxFrame )`

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.

## FlexCAN Driver

### Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.         |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.        |
| <i>pTxFrame</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. |

#### 17.2.7.33 **status\_t FLEXCAN\_ReadRxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *pRxFrame* )**

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.

### Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.                      |
| <i>mbIdx</i>    | The FlexCAN Message Buffer index.                     |
| <i>pRxFrame</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.                                             |

#### 17.2.7.34 **status\_t FLEXCAN\_WriteFDTxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, const flexcan\_fd\_frame\_t \* *pTxFrame* )**

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>pTxFrame</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. |

### 17.2.7.35 status\_t FLEXCAN\_ReadFDRxMb ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_fd\_frame\_t \* *pRxFrame* )

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>pRxFrame</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.                                             |

### 17.2.7.36 status\_t FLEXCAN\_ReadRxFifo ( CAN\_Type \* *base*, flexcan\_frame\_t \* *pRxFrame* )

This function reads a CAN message from the FlexCAN build-in Rx FIFO.

## FlexCAN Driver

### Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base address.                      |
| <i>pRxFrame</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.                 |

### 17.2.7.37 **status\_t FLEXCAN\_TransferFDSendBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_fd\_frame\_t \* *pTxFrame* )**

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>pTxFrame</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. |

### 17.2.7.38 **status\_t FLEXCAN\_TransferFDReceiveBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_fd\_frame\_t \* *pRxFrame* )**

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>pRxFrame</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.                                             |

### 17.2.7.39 status\_t FLEXCAN\_TransferFDSendNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )

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>pMbXfer</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.                          |

### 17.2.7.40 status\_t FLEXCAN\_TransferFDReceiveNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )

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>pMbXfer</i> | FlexCAN FD Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

## FlexCAN Driver

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.                            |

### 17.2.7.41 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. |

### 17.2.7.42 void FLEXCAN\_TransferFDAbortReceive ( 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 FD Message Buffer index. |

### 17.2.7.43 status\_t FLEXCAN\_TransferSendBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *pTxFrame* )

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>pTxFrame</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. |

#### 17.2.7.44 **status\_t FLEXCAN\_TransferReceiveBlocking ( CAN\_Type \* *base*, uint8\_t *mbIdx*, flexcan\_frame\_t \* *pRxFrame* )**

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>pRxFrame</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.                                             |

#### 17.2.7.45 **status\_t FLEXCAN\_TransferReceiveFifoBlocking ( CAN\_Type \* *base*, flexcan\_frame\_t \* *pRxFrame* )**

Note that a transfer handle does not need to be created before calling this API.

Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>base</i>     | FlexCAN peripheral base pointer.                      |
| <i>pRxFrame</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.                 |

**17.2.7.46 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. |

#### 17.2.7.47 status\_t FLEXCAN\_TransferSendNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )

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>pMbXfer</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.                          |

#### 17.2.7.48 status\_t FLEXCAN\_TransferReceiveNonBlocking ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle*, flexcan\_mb\_transfer\_t \* *pMbXfer* )

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>pMbXfer</i> | FlexCAN Message Buffer transfer structure. See the <a href="#">flexcan_mb_transfer_t</a> . |

## FlexCAN Driver

### 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.                            |

#### 17.2.7.49 status\_t FLEXCAN\_TransferReceiveFifoNonBlocking ( CAN\_Type \* base, flexcan\_handle\_t \* handle, flexcan\_fifo\_transfer\_t \* pFifoXfer )

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>pFifoXfer</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.                  |

#### 17.2.7.50 uint32\_t FLEXCAN\_GetTimeStamp ( flexcan\_handle\_t \* handle, uint8\_t mbIdx )

This function can only be used when calling non-blocking Data transfer (TX/RX) API. After TX/RX data transfer is done (User can get the status by handler's callback function), we can get the detail index of Mailbox's timestamp by handle. Detail non-blocking data transfer API (TX/RX) contain: -FLEXCAN\_TransferSendNonBlocking -FLEXCAN\_TransferFDSendNonBlocking -FLEXCAN\_TransferReceiveNonBlocking -FLEXCAN\_TransferFDReceiveNonBlocking -FLEXCAN\_TransferReceiveFifoNonBlocking

### Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>handle</i> | FlexCAN handle pointer.              |
| <i>mbIdx</i>  | The FlexCAN FD Message Buffer index. |

Return values

|            |                                                     |
|------------|-----------------------------------------------------|
| <i>the</i> | index of mailbox 's timestamp stored in the handle. |
|------------|-----------------------------------------------------|

#### 17.2.7.51 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. |

#### 17.2.7.52 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. |

#### 17.2.7.53 void FLEXCAN\_TransferAbortReceiveFifo ( CAN\_Type \* *base*, flexcan\_handle\_t \* *handle* )

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | FlexCAN peripheral base address. |
| <i>handle</i> | FlexCAN handle pointer.          |

### 17.2.7.54 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.          |

## FlexCAN eDMA Driver

### 17.3.1 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, 5, 2))  
*FlexCAN EDMA driver version.*

#### 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.*
- void [FLEXCAN\\_PrepareTransfConfiguration](#) (CAN\_Type \*base, [flexcan\\_fifo\\_transfer\\_t](#) \*pFifoXfer, [edma\\_transfer\\_config\\_t](#) \*pEdmaConfig)  
*Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.*
- [status\\_t](#) [FLEXCAN\\_StartTransferDatafromRxFIFO](#) (CAN\_Type \*base, flexcan\_edma\_handle\_t \*handle, [edma\\_transfer\\_config\\_t](#) \*pEdmaConfig)  
*Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.*
- [status\\_t](#) [FLEXCAN\\_TransferReceiveFifoEDMA](#) (CAN\_Type \*base, flexcan\_edma\_handle\_t \*handle, [flexcan\\_fifo\\_transfer\\_t](#) \*pFifoXfer)  
*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.*

## 17.3.2 Data Structure Documentation

### 17.3.2.1 struct \_flexcan\_edma\_handle

#### Data Fields

- [flexcan\\_edma\\_transfer\\_callback\\_t](#) *callback*  
*Callback function.*
- void \* [userData](#)  
*FlexCAN callback function parameter.*
- [edma\\_handle\\_t](#) \* [rxFifoEdmaHandle](#)  
*The EDMA Rx FIFO channel used.*
- volatile uint8\_t [rxFifoState](#)  
*Rx FIFO transfer state.*

#### 17.3.2.1.0.45 Field Documentation

17.3.2.1.0.45.1 [flexcan\\_edma\\_transfer\\_callback\\_t](#) [flexcan\\_edma\\_handle\\_t::callback](#)

17.3.2.1.0.45.2 void\* [flexcan\\_edma\\_handle\\_t::userData](#)

17.3.2.1.0.45.3 [edma\\_handle\\_t](#)\* [flexcan\\_edma\\_handle\\_t::rxFifoEdmaHandle](#)

17.3.2.1.0.45.4 volatile uint8\_t [flexcan\\_edma\\_handle\\_t::rxFifoState](#)

## 17.3.3 Macro Definition Documentation

17.3.3.1 #define FSL\_FLEXCAN\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 5, 2))

## 17.3.4 Typedef Documentation

17.3.4.1 typedef void(\* [flexcan\\_edma\\_transfer\\_callback\\_t](#))(CAN\_Type \*base,  
[flexcan\\_edma\\_handle\\_t](#) \*handle, status\_t status, void \*userData)

## 17.3.5 Function Documentation

17.3.5.1 void FLEXCAN\_TransferCreateHandleEDMA ( CAN\_Type \* base,  
[flexcan\\_edma\\_handle\\_t](#) \* handle, [flexcan\\_edma\\_transfer\\_callback\\_t](#) callback,  
void \* userData, [edma\\_handle\\_t](#) \* [rxFifoEdmaHandle](#) )

Parameters

---

## FlexCAN eDMA Driver

|                          |                                                     |
|--------------------------|-----------------------------------------------------|
| <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>rxFifoEdma-Handle</i> | User-requested DMA handle for Rx FIFO DMA transfer. |

### 17.3.5.2 void FLEXCAN\_PrepareTransfConfiguration ( CAN\_Type \* *base*, flexcan\_fifo\_transfer\_t \* *pFifoXfer*, edma\_transfer\_config\_t \* *pEdmaConfig* )

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

|                    |                                                                                        |
|--------------------|----------------------------------------------------------------------------------------|
| <i>base</i>        | FlexCAN peripheral base address.                                                       |
| <i>pFifoXfer</i>   | FlexCAN Rx FIFO EDMA transfer structure, see <a href="#">flexcan_fifo_transfer_t</a> . |
| <i>pEdmaConfig</i> | The user configuration structure of type edma_transfer_t.                              |

### 17.3.5.3 status\_t FLEXCAN\_StartTransferDatafromRxFIFO ( CAN\_Type \* *base*, flexcan\_edma\_handle\_t \* *handle*, edma\_transfer\_config\_t \* *pEdmaConfig* )

This function to Update edma transfer configuration and Start eDMA transfer

Parameters

|                    |                                                           |
|--------------------|-----------------------------------------------------------|
| <i>base</i>        | FlexCAN peripheral base address.                          |
| <i>handle</i>      | Pointer to flexcan_edma_handle_t structure.               |
| <i>pEdmaConfig</i> | The user configuration structure of type edma_transfer_t. |

Return values

|                                    |                            |
|------------------------------------|----------------------------|
| <i>kStatus_Success</i>             | if succeed, others failed. |
| <i>kStatus_FLEXCAN_Rx-FifoBusy</i> | Previous transfer ongoing. |

**17.3.5.4** `status_t FLEXCAN_TransferReceiveFifoEDMA ( CAN_Type * base,  
flexcan_edma_handle_t * handle, flexcan_fifo_transfer_t * pFifoXfer )`

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.

## FlexCAN eDMA Driver

### Parameters

|                  |                                                                                        |
|------------------|----------------------------------------------------------------------------------------|
| <i>base</i>      | FlexCAN peripheral base address.                                                       |
| <i>handle</i>    | Pointer to flexcan_edma_handle_t structure.                                            |
| <i>pFifoXfer</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. |

### 17.3.5.5 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 18

# FLEXSPI: Flexible Serial Peripheral Interface Driver

### Overview

The MCUXpresso SDK provides a peripheral driver for the Flexible Serial Peripheral Interface (FLEXSPI) module of MCUXpresso SDK/i.MX devices.

FLEXSPI driver includes functional APIs and interrupt/EDMA non-blocking transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FLEXSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FLEXSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FLEXSPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `flexspi_handle_t`/`flexspi_edma_handle_t` as the second parameter. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandle` API. Initialize the handle for interrupt non-blocking transfer by calling the `FLEXSPI_TransferCreateHandleEDMA` API.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXSPI\\_TransferNonBlocking\(\)](#) and `FLEXSPI_TransferEDMA()` set up data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXSPI_Idle` status.

### Data Structures

- struct [flexspi\\_config\\_t](#)  
*FLEXSPI configuration structure. [More...](#)*
- struct [flexspi\\_device\\_config\\_t](#)  
*External device configuration items. [More...](#)*
- struct [flexspi\\_transfer\\_t](#)  
*Transfer structure for FLEXSPI. [More...](#)*
- struct [flexspi\\_handle\\_t](#)  
*Transfer handle structure for FLEXSPI. [More...](#)*

### Macros

- #define [FLEXSPI\\_LUT\\_SEQ](#)(cmd0, pad0, op0, cmd1, pad1, op1)  
*Formula to form FLEXSPI instructions in LUT table.*

### Typedefs

- typedef void(\* flexspi\_transfer\_callback\_t )(FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle, status\_t status, void \*userData)  
*FLEXSPI transfer callback function.*

### Enumerations

- enum {  
kStatus\_FLEXSPI\_Busy = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 0),  
kStatus\_FLEXSPI\_SequenceExecutionTimeout = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 1),  
kStatus\_FLEXSPI\_IpCommandSequenceError = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 2),  
kStatus\_FLEXSPI\_IpCommandGrantTimeout = MAKE\_STATUS(kStatusGroup\_FLEXSPI, 3) }  
*Status structure of FLEXSPI.*
- enum {  
kFLEXSPI\_Command\_STOP = 0x00U,  
kFLEXSPI\_Command\_SDR = 0x01U,  
kFLEXSPI\_Command\_RADDR\_SDR = 0x02U,  
kFLEXSPI\_Command\_CADDR\_SDR = 0x03U,  
kFLEXSPI\_Command\_MODE1\_SDR = 0x04U,  
kFLEXSPI\_Command\_MODE2\_SDR = 0x05U,  
kFLEXSPI\_Command\_MODE4\_SDR = 0x06U,  
kFLEXSPI\_Command\_MODE8\_SDR = 0x07U,  
kFLEXSPI\_Command\_WRITE\_SDR = 0x08U,  
kFLEXSPI\_Command\_READ\_SDR = 0x09U,  
kFLEXSPI\_Command\_LEARN\_SDR = 0x0AU,  
kFLEXSPI\_Command\_DATSZ\_SDR = 0x0BU,  
kFLEXSPI\_Command\_DUMMY\_SDR = 0x0CU,  
kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR = 0x0DU,  
kFLEXSPI\_Command\_DDR = 0x21U,  
kFLEXSPI\_Command\_RADDR\_DDR = 0x22U,  
kFLEXSPI\_Command\_CADDR\_DDR = 0x23U,  
kFLEXSPI\_Command\_MODE1\_DDR = 0x24U,  
kFLEXSPI\_Command\_MODE2\_DDR = 0x25U,  
kFLEXSPI\_Command\_MODE4\_DDR = 0x26U,  
kFLEXSPI\_Command\_MODE8\_DDR = 0x27U,  
kFLEXSPI\_Command\_WRITE\_DDR = 0x28U,  
kFLEXSPI\_Command\_READ\_DDR = 0x29U,  
kFLEXSPI\_Command\_LEARN\_DDR = 0x2AU,  
kFLEXSPI\_Command\_DATSZ\_DDR = 0x2BU,  
kFLEXSPI\_Command\_DUMMY\_DDR = 0x2CU,  
kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR = 0x2DU,  
kFLEXSPI\_Command\_JUMP\_ON\_CS = 0x1FU }  
*CMD definition of FLEXSPI, use to form LUT instruction, \_flexspi\_command.*
- enum flexspi\_pad\_t {



```

kFLEXSPI_1PAD = 0x00U,
kFLEXSPI_2PAD = 0x01U,
kFLEXSPI_4PAD = 0x02U,
kFLEXSPI_8PAD = 0x03U }

```

*pad definition of FLEXSPI, use to form LUT instruction.*

- enum flexspi\_flags\_t {
 

```

kFLEXSPI_SequenceExecutionTimeoutFlag = FLEXSPI_INTEN_SEQTIMEOUTEN_MASK,
kFLEXSPI_AhbBusTimeoutFlag = FLEXSPI_INTEN_AHBBUSTIMEOUTEN_MASK,
kFLEXSPI_SckStoppedBecauseTxEmptyFlag,
kFLEXSPI_SckStoppedBecauseRxFullFlag,
kFLEXSPI_DataLearningFailedFlag = FLEXSPI_INTEN_DATALEARNFAILEN_MASK,
kFLEXSPI_IpTxFifoWatermarkEmptyFlag = FLEXSPI_INTEN_IPTXWEEN_MASK,
kFLEXSPI_IpRxFifoWatermarkAvailableFlag = FLEXSPI_INTEN_IPRXWAEN_MASK,
kFLEXSPI_AhbCommandSequenceErrorFlag,
kFLEXSPI_IpCommandSequenceErrorFlag = FLEXSPI_INTEN_IPCMDERREN_MASK,
kFLEXSPI_AhbCommandGrantTimeoutFlag,
kFLEXSPI_IpCommandGrantTimeoutFlag,
kFLEXSPI_IpCommandExecutionDoneFlag,
kFLEXSPI_AllInterruptFlags = 0xFFFU }

```

*FLEXSPI interrupt status flags.*

- enum flexspi\_read\_sample\_clock\_t {
 

```

kFLEXSPI_ReadSampleClkLoopbackInternally = 0x0U,
kFLEXSPI_ReadSampleClkLoopbackFromDqsPad = 0x1U,
kFLEXSPI_ReadSampleClkLoopbackFromSckPad = 0x2U,
kFLEXSPI_ReadSampleClkExternalInputFromDqsPad = 0x3U }

```

*FLEXSPI sample clock source selection for Flash Reading.*

- enum flexspi\_cs\_interval\_cycle\_unit\_t {
 

```

kFLEXSPI_CsIntervalUnit1SckCycle = 0x0U,
kFLEXSPI_CsIntervalUnit256SckCycle = 0x1U }

```

*FLEXSPI interval unit for flash device select.*

- enum flexspi\_ahb\_write\_wait\_unit\_t {
 

```

kFLEXSPI_AhbWriteWaitUnit2AhbCycle = 0x0U,
kFLEXSPI_AhbWriteWaitUnit8AhbCycle = 0x1U,
kFLEXSPI_AhbWriteWaitUnit32AhbCycle = 0x2U,
kFLEXSPI_AhbWriteWaitUnit128AhbCycle = 0x3U,
kFLEXSPI_AhbWriteWaitUnit512AhbCycle = 0x4U,
kFLEXSPI_AhbWriteWaitUnit2048AhbCycle = 0x5U,
kFLEXSPI_AhbWriteWaitUnit8192AhbCycle = 0x6U,
kFLEXSPI_AhbWriteWaitUnit32768AhbCycle = 0x7U }

```

*FLEXSPI AHB wait interval unit for writing.*

- enum flexspi\_ip\_error\_code\_t {

## Overview

```
kFLEXSPI_IpCmdErrorNoError = 0x0U,
kFLEXSPI_IpCmdErrorJumpOnCsInIpCmd = 0x2U,
kFLEXSPI_IpCmdErrorUnknownOpCode = 0x3U,
kFLEXSPI_IpCmdErrorSdrDummyInDdrSequence = 0x4U,
kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence = 0x5U,
kFLEXSPI_IpCmdErrorInvalidAddress = 0x6U,
kFLEXSPI_IpCmdErrorSequenceExecutionTimeout = 0xEU,
kFLEXSPI_IpCmdErrorFlashBoundaryAcrosss = 0xFU }
```

*Error Code when IP command Error detected.*

- enum flexspi\_ahb\_error\_code\_t {  
kFLEXSPI\_AhbCmdErrorNoError = 0x0U,  
kFLEXSPI\_AhbCmdErrorJumpOnCsInWriteCmd = 0x2U,  
kFLEXSPI\_AhbCmdErrorUnknownOpCode = 0x3U,  
kFLEXSPI\_AhbCmdErrorSdrDummyInDdrSequence = 0x4U,  
kFLEXSPI\_AhbCmdErrorDdrDummyInSdrSequence = 0x5U,  
kFLEXSPI\_AhbCmdSequenceExecutionTimeout = 0x6U }

*Error Code when AHB command Error detected.*

- enum flexspi\_port\_t {  
kFLEXSPI\_PortA1 = 0x0U,  
kFLEXSPI\_PortA2,  
kFLEXSPI\_PortB1,  
kFLEXSPI\_PortB2 }
- FLEXSPI operation port select.*
- enum flexspi\_arb\_command\_source\_t  
*Trigger source of current command sequence granted by arbitrator.*
- enum flexspi\_command\_type\_t {  
kFLEXSPI\_Command,  
kFLEXSPI\_Config }

*Command type.*

## Driver version

- #define FSL\_FLEXSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 2))  
*FLEXSPI driver version 2.2.2.*

## 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

- void [FLEXSPI\\_UpdateRxSampleClock](#) (FLEXSPI\_Type \*base, [flexspi\\_read\\_sample\\_clock\\_t](#) clockSource)  
*Update read sample clock source.*
- static void [FLEXSPI\\_EnableIPParallelMode](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables/disables the FLEXSPI IP command parallel mode.*
- static void [FLEXSPI\\_EnableAHBParallelMode](#) (FLEXSPI\_Type \*base, bool enable)  
*Enables/disables the FLEXSPI AHB 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.*

## Data Structure Documentation

### 18.2.1 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 execution finished.
- bool [enableAHBCachable](#)  
 Enable AHB bus cachable read access support.

### 18.2.1.0.0.46 Field Documentation

- 18.2.1.0.0.46.1 `flexspi_read_sample_clock_t flexspi_config_t::rxSampleClock`
- 18.2.1.0.0.46.2 `bool flexspi_config_t::enableSckFreeRunning`
- 18.2.1.0.0.46.3 `bool flexspi_config_t::enableCombination`
- 18.2.1.0.0.46.4 `bool flexspi_config_t::enableDoze`
- 18.2.1.0.0.46.5 `bool flexspi_config_t::enableHalfSpeedAccess`
- 18.2.1.0.0.46.6 `bool flexspi_config_t::enableSckBDiffOpt`
- 18.2.1.0.0.46.7 `bool flexspi_config_t::enableSameConfigForAll`
- 18.2.1.0.0.46.8 `uint16_t flexspi_config_t::seqTimeoutCycle`
- 18.2.1.0.0.46.9 `uint8_t flexspi_config_t::ipGrantTimeoutCycle`
- 18.2.1.0.0.46.10 `uint8_t flexspi_config_t::txWatermark`
- 18.2.1.0.0.46.11 `uint8_t flexspi_config_t::rxWatermark`
- 18.2.1.0.0.46.12 `bool flexspi_config_t::enableAHBWritelpTxFifo`
- 18.2.1.0.0.46.13 `bool flexspi_config_t::enableAHBWritelpRxFifo`
- 18.2.1.0.0.46.14 `uint8_t flexspi_config_t::ahbGrantTimeoutCycle`
- 18.2.1.0.0.46.15 `uint16_t flexspi_config_t::ahbBusTimeoutCycle`
- 18.2.1.0.0.46.16 `uint8_t flexspi_config_t::resumeWaitCycle`
- 18.2.1.0.0.46.17 `flexspi_ahbBuffer_config_t flexspi_config_t::buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT]`
- 18.2.1.0.0.46.18 `bool flexspi_config_t::enableClearAHBBufferOpt`
- 18.2.1.0.0.46.19 `bool flexspi_config_t::enableReadAddressOpt`

when enable, there is no AHB read burst start address alignment limitation.

18.2.1.0.0.46.20 **bool** flexspi\_config\_t::enableAHBPrefetch

18.2.1.0.0.46.21 **bool** flexspi\_config\_t::enableAHBBufferable

18.2.1.0.0.46.22 **bool** flexspi\_config\_t::enableAHBCachable

## 18.2.2 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, multiply CS interval unit to get the CS line assert interval cycles.*
- uint8\_t [CSHoldTime](#)  
*CS line hold time.*
- uint8\_t [CSSetupTime](#)  
*CS line setup time.*
- uint8\_t [dataValidTime](#)  
*Data valid time for external device.*
- uint8\_t [columnspace](#)  
*Column space size.*
- bool [enableWordAddress](#)  
*If enable word address.*
- uint8\_t [AWRSeqIndex](#)  
*Sequence ID for AHB write command.*
- uint8\_t [AWRSeqNumber](#)  
*Sequence number for AHB write command.*
- uint8\_t [ARDSeqIndex](#)  
*Sequence ID for AHB read command.*
- uint8\_t [ARDSeqNumber](#)  
*Sequence number for AHB read command.*
- [flexspi\\_ahb\\_write\\_wait\\_unit\\_t](#) AHBWriteWaitUnit  
*AHB write wait unit.*
- uint16\_t [AHBWriteWaitInterval](#)  
*AHB write wait interval, multiply AHB write interval unit to get the AHB write wait cycles.*
- bool [enableWriteMask](#)  
*Enable/Disable FLEXSPI drive DQS pin as write mask when writing to external device.*

### 18.2.2.0.0.47 Field Documentation

- 18.2.2.0.0.47.1 `uint32_t flexspi_device_config_t::flexspiRootClk`
- 18.2.2.0.0.47.2 `bool flexspi_device_config_t::isSck2Enabled`
- 18.2.2.0.0.47.3 `uint32_t flexspi_device_config_t::flashSize`
- 18.2.2.0.0.47.4 `flexspi_cs_interval_cycle_unit_t flexspi_device_config_t::CSIntervalUnit`
- 18.2.2.0.0.47.5 `uint16_t flexspi_device_config_t::CSInterval`
- 18.2.2.0.0.47.6 `uint8_t flexspi_device_config_t::CSHoldTime`
- 18.2.2.0.0.47.7 `uint8_t flexspi_device_config_t::CSSetupTime`
- 18.2.2.0.0.47.8 `uint8_t flexspi_device_config_t::dataValidTime`
- 18.2.2.0.0.47.9 `uint8_t flexspi_device_config_t::columnspace`
- 18.2.2.0.0.47.10 `bool flexspi_device_config_t::enableWordAddress`
- 18.2.2.0.0.47.11 `uint8_t flexspi_device_config_t::AWRSeqIndex`
- 18.2.2.0.0.47.12 `uint8_t flexspi_device_config_t::AWRSeqNumber`
- 18.2.2.0.0.47.13 `uint8_t flexspi_device_config_t::ARDSeqIndex`
- 18.2.2.0.0.47.14 `uint8_t flexspi_device_config_t::ARDSeqNumber`
- 18.2.2.0.0.47.15 `flexspi_ahb_write_wait_unit_t flexspi_device_config_t::AHBWriteWaitUnit`
- 18.2.2.0.0.47.16 `uint16_t flexspi_device_config_t::AHBWriteWaitInterval`
- 18.2.2.0.0.47.17 `bool flexspi_device_config_t::enableWriteMask`

### 18.2.3 `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.*

#### 18.2.3.0.0.48 Field Documentation

18.2.3.0.0.48.1 uint32\_t flexspi\_transfer\_t::deviceAddress

18.2.3.0.0.48.2 flexspi\_port\_t flexspi\_transfer\_t::port

18.2.3.0.0.48.3 flexspi\_command\_type\_t flexspi\_transfer\_t::cmdType

18.2.3.0.0.48.4 uint8\_t flexspi\_transfer\_t::seqIndex

18.2.3.0.0.48.5 uint8\_t flexspi\_transfer\_t::SeqNumber

18.2.3.0.0.48.6 uint32\_t\* flexspi\_transfer\_t::data

18.2.3.0.0.48.7 size\_t flexspi\_transfer\_t::dataSize

### 18.2.4 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.*

#### 18.2.4.0.0.49 Field Documentation

18.2.4.0.0.49.1 uint32\_t\* flexspi\_handle\_t::data

18.2.4.0.0.49.2 size\_t flexspi\_handle\_t::dataSize

18.2.4.0.0.49.3 size\_t flexspi\_handle\_t::transferTotalSize

18.2.4.0.0.49.4 void\* flexspi\_handle\_t::userData

## Enumeration Type Documentation

## Macro Definition Documentation

### 18.3.1 #define FSL\_FLEXSPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 2))

### 18.3.2 #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))
```

## Typedef Documentation

### 18.4.1 typedef void(\* flexspi\_transfer\_callback\_t)(FLEXSPI\_Type \*base, flexspi\_handle\_t \*handle, status\_t status, void \*userData)

## Enumeration Type Documentation

### 18.5.1 anonymous enum

Enumerator

- kStatus\_FLEXSPI\_Busy* FLEXSPI is busy.
- kStatus\_FLEXSPI\_SequenceExecutionTimeout* Sequence execution timeout error occurred during FLEXSPI transfer.
- kStatus\_FLEXSPI\_IpCommandSequenceError* IP command Sequence execution timeout error occurred during FLEXSPI transfer.
- kStatus\_FLEXSPI\_IpCommandGrantTimeout* IP command grant timeout error occurred during FLEXSPI transfer.

### 18.5.2 anonymous enum

Enumerator

- kFLEXSPI\_Command\_STOP* Stop execution, deassert CS.
- kFLEXSPI\_Command\_SDR* Transmit Command code to Flash, using SDR mode.
- kFLEXSPI\_Command\_RADDR\_SDR* Transmit Row Address to Flash, using SDR mode.
- kFLEXSPI\_Command\_CADDR\_SDR* Transmit Column Address to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE1\_SDR* Transmit 1-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE2\_SDR* Transmit 2-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE4\_SDR* Transmit 4-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_MODE8\_SDR* Transmit 8-bit Mode bits to Flash, using SDR mode.
- kFLEXSPI\_Command\_WRITE\_SDR* Transmit Programming Data to Flash, using SDR mode.
- kFLEXSPI\_Command\_READ\_SDR* Receive Read Data from Flash, using SDR mode.

***kFLEXSPI\_Command\_LEARN\_SDR*** Receive Read Data or Preamble bit from Flash, SDR mode.

***kFLEXSPI\_Command\_DATSZ\_SDR*** Transmit Read/Program Data size (byte) to Flash, SDR mode.

***kFLEXSPI\_Command\_DUMMY\_SDR*** Leave data lines undriven by FlexSPI controller.

***kFLEXSPI\_Command\_DUMMY\_RWDS\_SDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

***kFLEXSPI\_Command\_DDR*** Transmit Command code to Flash, using DDR mode.

***kFLEXSPI\_Command\_RADDR\_DDR*** Transmit Row Address to Flash, using DDR mode.

***kFLEXSPI\_Command\_CADDR\_DDR*** Transmit Column Address to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE1\_DDR*** Transmit 1-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE2\_DDR*** Transmit 2-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE4\_DDR*** Transmit 4-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_MODE8\_DDR*** Transmit 8-bit Mode bits to Flash, using DDR mode.

***kFLEXSPI\_Command\_WRITE\_DDR*** Transmit Programming Data to Flash, using DDR mode.

***kFLEXSPI\_Command\_READ\_DDR*** Receive Read Data from Flash, using DDR mode.

***kFLEXSPI\_Command\_LEARN\_DDR*** Receive Read Data or Preamble bit from Flash, DDR mode.

***kFLEXSPI\_Command\_DATSZ\_DDR*** Transmit Read/Program Data size (byte) to Flash, DDR mode.

***kFLEXSPI\_Command\_DUMMY\_DDR*** Leave data lines undriven by FlexSPI controller.

***kFLEXSPI\_Command\_DUMMY\_RWDS\_DDR*** Leave data lines undriven by FlexSPI controller, dummy cycles decided by RWDS.

***kFLEXSPI\_Command\_JUMP\_ON\_CS*** Stop execution, deassert CS and save operand[7:0] as the instruction start pointer for next sequence.

### 18.5.3 enum flexspi\_pad\_t

Enumerator

***kFLEXSPI\_1PAD*** Transmit command/address and transmit/receive data only through DATA0/D-ATA1.

***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].

### 18.5.4 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.

### 18.5.5 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.

### 18.5.6 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.

### 18.5.7 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.

### 18.5.8 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.

### 18.5.9 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

### 18.5.10 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.

### 18.5.11 enum flexspi\_arb\_command\_source\_t

### 18.5.12 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.

## Function Documentation

### 18.6.1 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.     |

### 18.6.2 void FLEXSPI\_GetDefaultConfig ( flexspi\_config\_t \* *config* )

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>config</i> | FLEXSPI configuration structure. |
|---------------|----------------------------------|

### 18.6.3 void FLEXSPI\_Deinit ( FLEXSPI\_Type \* *base* )

Clears the FLEXSPI state and FLEXSPI module registers.

## Parameters

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

#### 18.6.4 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.          |

#### 18.6.5 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. |
|-------------|----------------------------------|

#### 18.6.6 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. |

#### 18.6.7 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.        |

**18.6.8 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.        |

**18.6.9 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. |

**18.6.10 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. |

**18.6.11 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. |
|------------|------------------|

### 18.6.12 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. |
|------------|------------------|

### 18.6.13 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.      |

### 18.6.14 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.  |

### 18.6.15 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. |
|------------------|---------------------------------------------------------------------------------------------------|

### 18.6.16 static void FLEXSPI\_ClearInterruptStatusFlags ( FLEXSPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | FLEXSPI peripheral base address. |
| <i>mask</i> | FLEXSPI interrupt source.        |

### 18.6.17 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. |

**18.6.18** `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. |
|----------------|-------------------------------------|

### 18.6.19 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. |
|--------------|--------------------------------------|

### 18.6.20 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. |
|--------------|---------------------------------------|

### 18.6.21 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. |

### 18.6.22 void FLEXSPI\_UpdateRxSampleClock ( FLEXSPI\_Type \* *base*, flexspi\_read\_sample\_clock\_t *clockSource* )

## Parameters

|                    |                                                                 |
|--------------------|-----------------------------------------------------------------|
| <i>base</i>        | FLEXSPI peripheral base address.                                |
| <i>clockSource</i> | clockSource of type <a href="#">flexspi_read_sample_clock_t</a> |

### 18.6.23 static void FLEXSPI\_EnableIPParallelMode ( 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. |

### 18.6.24 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. |

### 18.6.25 void FLEXSPI\_UpdateLUT ( FLEXSPI\_Type \* *base*, uint32\_t *index*, const uint32\_t \* *cmd*, uint32\_t *count* )

## Function Documentation

### Parameters

|              |                                                                                                                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>  | FLEXSPI peripheral base address.                                                                                                                                                                                       |
| <i>index</i> | From which index start to update. It could be any index of the LUT table, which also allows user to update command content inside a command. Each command consists of up to 8 instructions and occupy 4*32-bit memory. |
| <i>cmd</i>   | Command sequence array.                                                                                                                                                                                                |
| <i>count</i> | Number of sequences.                                                                                                                                                                                                   |

**18.6.26** `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.         |

**18.6.27** `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.

**18.6.28** `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 sequence error detected |
| <i>kStatus_FLEXSPI_IpCommandGrantTimeout</i>    | IP command grant timeout detected  |

### 18.6.29 **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 |

## Function Documentation

|                                                |                                     |
|------------------------------------------------|-------------------------------------|
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i> | IP command sequencen error detected |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>  | IP command grant timeout detected   |

### 18.6.30 **status\_t FLEXSPI\_TransferBlocking ( FLEXSPI\_Type \* *base*, flexspi\_transfer\_t \* *xfer* )**

#### Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | FLEXSPI peripheral base address    |
| <i>xfer</i> | pointer to the transfer structure. |

#### Return values

|                                                   |                                        |
|---------------------------------------------------|----------------------------------------|
| <i>kStatus_Success</i>                            | command transfer success without error |
| <i>kStatus_FLEXSPI_-SequenceExecution-Timeout</i> | sequence execution timeout             |
| <i>kStatus_FLEXSPI_Ip-CommandSequenceError</i>    | IP command sequence error detected     |
| <i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>     | IP command grant timeout detected      |

### 18.6.31 **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.                    |



### 18.6.32 **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. |

### 18.6.33 **status\_t FLEXSPI\_TransferGetCount ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle*, size\_t \* *count* )**

#### Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                       |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.    |

#### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

### 18.6.34 **void FLEXSPI\_TransferAbort ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle* )**

## Function Documentation

### Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

### Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.                                      |
| <i>handle</i> | pointer to flexspi_handle_t structure which stores the transfer state |

### 18.6.35 void FLEXSPI\_TransferHandleIRQ ( FLEXSPI\_Type \* *base*, flexspi\_handle\_t \* *handle* )

### Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | FLEXSPI peripheral base address.       |
| <i>handle</i> | pointer to flexspi_handle_t structure. |

## Chapter 19

# FTM: FlexTimer Driver

### Overview

The MCUXpresso SDK provides a driver for the FlexTimer Module (FTM) of MCUXpresso SDK devices.

### 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.

#### 19.2.1 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.

#### 19.2.2 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.

#### 19.2.3 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.

### 19.2.4 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.

### 19.2.5 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.

### 19.2.6 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.

## 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/ftm` Multiple 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 synchnronization 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`

## Typical use case

### 19.4.1 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](#),  
  [kFTM\\_ComplementaryPwm](#) }  
*FTM PWM operation modes.*

## Typical use case

- enum `ftm_pwm_level_select_t` {  
    `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\_SHIFT)),  
    `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` }  
    *FlexTimer quadrature decode modes.*
- 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_Chnl6Trigger`,  
`kFTM_Chnl7Trigger`,  
`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 {  
    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.*
- static void **FTM\_EnableDmaTransfer** (FTM\_Type \*base, **ftm\_chnl\_t** chnlNumber, bool enable)  
*Enable DMA transfer or not.*

## Driver version

- #define **FSL\_FTM\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 2, 3))  
*FTM driver version 2.2.3.*

## Initialization and deinitialization

- **status\_t FTM\_Init** (FTM\_Type \*base, const **ftm\_config\_t** \*config)  
*Un-gates 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)

## Typical use case

- 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 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.*

## Data Structure Documentation

### 19.5.1 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.*
- bool [enableDeadtime](#)  
*true: The deadtime insertion in this pair of channels is enabled; false: The deadtime insertion in this pair of channels is disabled.*

#### 19.5.1.0.0.50 Field Documentation

##### 19.5.1.0.0.50.1 [ftm\\_chnl\\_t](#) [ftm\\_chnl\\_pwm\\_signal\\_param\\_t::chnlNumber](#)

In combined mode, this represents the channel pair number.

##### 19.5.1.0.0.50.2 [ftm\\_pwm\\_level\\_select\\_t](#) [ftm\\_chnl\\_pwm\\_signal\\_param\\_t::level](#)

##### 19.5.1.0.0.50.3 [uint8\\_t](#) [ftm\\_chnl\\_pwm\\_signal\\_param\\_t::dutyCyclePercent](#)

100 = always active signal (100% duty cycle).

## Data Structure Documentation

### 19.5.1.0.0.50.4 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

### 19.5.1.0.0.50.5 bool ftm\_chnl\_pwm\_signal\_param\_t::enableDeadtime

## 19.5.2 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.*

### 19.5.2.0.0.51 Field Documentation

#### 19.5.2.0.0.51.1 ftm\_chnl\_t ftm\_chnl\_pwm\_config\_param\_t::chnlNumber

In combined mode, this represents the channel pair number.

#### 19.5.2.0.0.51.2 ftm\_pwm\_level\_select\_t ftm\_chnl\_pwm\_config\_param\_t::level

#### 19.5.2.0.0.51.3 uint16\_t ftm\_chnl\_pwm\_config\_param\_t::dutyValue

#### 19.5.2.0.0.51.4 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.

## 19.5.3 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.*

### 19.5.4 struct ftm\_phase\_params\_t

#### Data Fields

- bool [enablePhaseFilter](#)  
*True: enable phase filter; false: disable filter.*
- uint32\_t [phaseFilterVal](#)  
*Filter value, used only if phase filter is enabled.*
- [ftm\\_phase\\_polarity\\_t](#) [phasePolarity](#)  
*Phase polarity.*

### 19.5.5 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.*

### 19.5.6 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.*

## Enumeration Type Documentation

- [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`  
*Defines the initialization value of the channels in `OUTINT` register.*
- [uint8\\_t](#) `chnlPolarity`  
*Defines the output polarity of the channels in `POL` register.*
- [bool](#) `useGlobalTimeBase`  
*True: Use of an external global time base is enabled; False: disabled.*

### 19.5.6.0.0.52 Field Documentation

#### 19.5.6.0.0.52.1 [uint32\\_t](#) `ftm_config_t::pwmSyncMode`

#### 19.5.6.0.0.52.2 [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](#).

#### 19.5.6.0.0.52.3 [uint32\\_t](#) `ftm_config_t::deadTimeValue`

#### 19.5.6.0.0.52.4 [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](#).

## Macro Definition Documentation

### 19.6.1 `#define FSL_FTM_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))`

## Enumeration Type Documentation

### 19.7.1 [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.

### 19.7.2 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.

### 19.7.3 enum ftm\_pwm\_mode\_t

Enumerator

*kFTM\_EdgeAlignedPwm* Edge-aligned PWM.  
*kFTM\_CenterAlignedPwm* Center-aligned PWM.  
*kFTM\_CombinedPwm* Combined PWM.  
*kFTM\_ComplementaryPwm* Complementary PWM.

### 19.7.4 enum ftm\_pwm\_level\_select\_t

Enumerator

*kFTM\_NoPwmSignal* No PWM output on pin.  
*kFTM\_LowTrue* Low true pulses.  
*kFTM\_HighTrue* High true pulses.

### 19.7.5 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.

### 19.7.6 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.

### 19.7.7 enum ftm\_dual\_edge\_capture\_mode\_t

Enumerator

*kFTM\_OneShot* One-shot capture mode.  
*kFTM\_Continuous* Continuous capture mode.

### 19.7.8 enum ftm\_quad\_decode\_mode\_t

Enumerator

*kFTM\_QuadPhaseEncode* Phase A and Phase B encoding mode.  
*kFTM\_QuadCountAndDir* Count and direction encoding mode.

### 19.7.9 enum ftm\_phase\_polarity\_t

Enumerator

*kFTM\_QuadPhaseNormal* Phase input signal is not inverted.  
*kFTM\_QuadPhaseInvert* Phase input signal is inverted.

### 19.7.10 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.



### 19.7.11 enum ftm\_clock\_source\_t

Enumerator

*kFTM\_SystemClock* System clock selected.  
*kFTM\_FixedClock* Fixed frequency clock.  
*kFTM\_ExternalClock* External clock.

### 19.7.12 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.

### 19.7.13 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.

### 19.7.14 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.

## Enumeration Type Documentation

### 19.7.15 enum ftm\_external\_trigger\_t

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\_Chnl6Trigger*** Available on certain SoC's, generate trigger when counter equals chnl 6 CnV reg.
- kFTM\_Chnl7Trigger*** Available on certain SoC's, generate trigger when counter equals chnl 7 CnV reg.
- kFTM\_InitTrigger*** Generate Trigger when counter is updated with CNTIN.
- kFTM\_ReloadInitTrigger*** Available on certain SoC's, trigger on reload point.

### 19.7.16 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.

### 19.7.17 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.

### 19.7.18 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.

### 19.7.19 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.

## Function Documentation

***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.

### 19.7.20 anonymous enum

Enumerator

***kFTM\_QuadDecoderCountingIncreaseFlag*** Counting direction is increasing (FTM counter increment), or the direction is decreasing.  
***kFTM\_QuadDecoderCountingOverflowOnTopFlag*** Indicates if the TOF bit was set on the top or the bottom of counting.

## Function Documentation

### 19.8.1 `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. If the FTM instance has only TPM features, please use the TPM 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.

### 19.8.2 `void FTM_Deinit ( FTM_Type * base )`

Parameters

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

### 19.8.3 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. |
|---------------|----------------------------------------------|

### 19.8.4 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                                                          |

## Function Documentation

|                    |                         |
|--------------------|-------------------------|
| <i>srcClock_Hz</i> | FTM counter clock in Hz |
|--------------------|-------------------------|

Returns

kStatus\_Success if the PWM setup was successful kStatus\_Error on failure

**19.8.5 void FTM\_UpdatePwmDutycycle ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, ftm\_pwm\_mode\_t *currentPwmMode*, uint8\_t *dutyCyclePercent* )**

Parameters

|                          |                                                                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <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) |

**19.8.6 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. |

**19.8.7 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

### 19.8.8 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. |

### 19.8.9 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

---

## Function Documentation

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| <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.                            |

### 19.8.10 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.

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. |

### 19.8.11 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 |

### 19.8.12 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> |

**19.8.13 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> |

**19.8.14 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](#)

**19.8.15 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](#)

**19.8.16 void FTM\_ClearStatusFlags ( FTM\_Type \* *base*, uint32\_t *mask* )**

## Function Documentation

### 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> |

### 19.8.17 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.

### 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. |

### 19.8.18 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

**19.8.19** `static void FTM_StartTimer ( FTM_Type * base, ftm_clock_source_t  
clockSource ) [inline], [static]`

## Function Documentation

### Parameters

|                    |                                                                              |
|--------------------|------------------------------------------------------------------------------|
| <i>base</i>        | FTM peripheral base address                                                  |
| <i>clockSource</i> | FTM clock source; After the clock source is set, the counter starts running. |

### 19.8.20 static void FTM\_StopTimer ( FTM\_Type \* *base* ) [inline], [static]

### Parameters

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

### 19.8.21 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<br>false: channel output is unaffected by software output control |

### 19.8.22 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 |

### 19.8.23 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 |

#### 19.8.24 static void FTM\_SetOutputMask ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, bool *mask* ) [inline], [static]

## 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 |

#### 19.8.25 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 |

#### 19.8.26 static void FTM\_SetFaultControlEnable ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlPairNumber*, bool *value* ) [inline], [static]

## Parameters

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

## Function Documentation

|                        |                                                                           |
|------------------------|---------------------------------------------------------------------------|
| <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 |

**19.8.27 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 |

**19.8.28 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 |

**19.8.29 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    |

**19.8.30** void FTM\_SetupQuadDecode ( FTM\_Type \* *base*, const  
ftm\_phase\_params\_t \* *phaseAParams*, const ftm\_phase\_params\_t \*  
*phaseBParams*, ftm\_quad\_decode\_mode\_t *quadMode* )

## Function Documentation

### 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 |

### 19.8.31 static uint32\_t FTM\_GetQuadDecoderFlags ( FTM\_Type \* *base* ) [inline], [static]

### Parameters

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

### Returns

Flag mask of FTM Quad Decoder, see `_ftm_quad_decoder_flags`.

### 19.8.32 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. |

### 19.8.33 static uint32\_t FTM\_GetQuadDecoderCounterValue ( FTM\_Type \* *base* ) [inline], [static]



## Parameters

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

## Returns

Current quad Decoder counter value.

### 19.8.34 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. |
|-------------|------------------------------|

### 19.8.35 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 |

### 19.8.36 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 |

### 19.8.37 static void FTM\_EnableDmaTransfer ( FTM\_Type \* *base*, ftm\_chnl\_t *chnlNumber*, bool *enable* ) [inline], [static]

Note: CHnIE bit needs to be set when calling this API. The channel DMA transfer request is generated and the channel interrupt is not generated if (CHnF = 1) when DMA and CHnIE bits are set.

## Function Documentation

### Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>base</i>       | FTM peripheral base address.     |
| <i>chnlNumber</i> | Channel to be configured         |
| <i>enable</i>     | true to enable, false to disable |

## Chapter 20

# GPT: General Purpose Timer

### Overview

The MCUXpresso SDK provides a driver for the General Purpose Timer (GPT) of MCUXpresso SDK devices.

### Function groups

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

#### 20.2.1 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.

### Typical use case

#### 20.3.1 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, 2)`)

## 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.*
- static void [GPT\\_SetOutputOperationMode](#) (GPT\_Type \*base, gpt\_output\_compare\_channel\_t channel, gpt\_output\_operation\_mode\_t mode)

## Data Structure Documentation

- 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.*

## Data Structure Documentation

### 20.4.1 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.

### 20.4.1.0.0.53 Field Documentation

20.4.1.0.0.53.1 `gpt_clock_source_t gpt_config_t::clockSource`

20.4.1.0.0.53.2 `uint32_t gpt_config_t::divider`

20.4.1.0.0.53.3 `bool gpt_config_t::enableFreeRun`

20.4.1.0.0.53.4 `bool gpt_config_t::enableRunInWait`

20.4.1.0.0.53.5 `bool gpt_config_t::enableRunInStop`

20.4.1.0.0.53.6 `bool gpt_config_t::enableRunInDoze`

20.4.1.0.0.53.7 `bool gpt_config_t::enableRunInDbg`

20.4.1.0.0.53.8 `bool gpt_config_t::enableMode`

## Enumeration Type Documentation

### 20.5.1 `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.

### 20.5.2 `enum gpt_input_capture_channel_t`

Enumerator

***kGPT\_InputCapture\_Channel1*** GPT Input Capture Channel1.

***kGPT\_InputCapture\_Channel2*** GPT Input Capture Channel2.

### 20.5.3 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.

### 20.5.4 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.

### 20.5.5 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.

### 20.5.6 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.



## 20.5.7 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.

## Function Documentation

### 20.6.1 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. |

### 20.6.2 void GPT\_Deinit ( GPT\_Type \* *base* )

Parameters

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

### 20.6.3 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 = false;
* config->enableMode = true;
*

```

## Function Documentation

### Parameters

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

### 20.6.4 static void GPT\_SoftwareReset ( GPT\_Type \* *base* ) [inline], [static]

### Parameters

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

### 20.6.5 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). |

### 20.6.6 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).

### 20.6.7 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).     |

### 20.6.8 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).

### 20.6.9 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).           |

### 20.6.10 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).

### 20.6.11 static void GPT\_StartTimer ( GPT\_Type \* *base* ) [inline], [static]

## Function Documentation

### Parameters

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

### 20.6.12 static void GPT\_StopTimer ( GPT\_Type \* *base* ) [inline], [static]

### Parameters

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

### 20.6.13 static uint32\_t GPT\_GetCurrentTimerCount ( GPT\_Type \* *base* ) [inline], [static]

### Parameters

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

### Returns

Current GPT counter value.

### 20.6.14 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). |

### 20.6.15 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).

**20.6.16** `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.

**20.6.17** `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).   |

**20.6.18** `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).

**20.6.19 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.                                                                          |

**20.6.20 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.

**20.6.21 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). |

**20.6.22 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> |

**20.6.23 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> |

**20.6.24 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](#)

**20.6.25**   `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.

**20.6.26 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). |



## Chapter 21

# GPIO: General-Purpose Input/Output Driver

### Overview

The MCUXpresso SDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of MCUXpresso SDK devices.

### Typical use case

#### 21.2.1 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, 3)`)  
*GPIO driver version 2.0.3.*

### 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.*
- 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.*

## Data Structure Documentation

### 21.3.1 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](#).*

#### 21.3.1.0.0.54 Field Documentation

21.3.1.0.0.54.1 [gpio\\_pin\\_direction\\_t gpio\\_pin\\_config\\_t::direction](#)

21.3.1.0.0.54.2 [gpio\\_interrupt\\_mode\\_t gpio\\_pin\\_config\\_t::interruptMode](#)

## Macro Definition Documentation

21.4.1 **#define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))**

## Enumeration Type Documentation

### 21.5.1 enum gpio\_pin\_direction\_t

Enumerator

**kGPIO\_DigitalInput** Set current pin as digital input.  
**kGPIO\_DigitalOutput** Set current pin as digital output.

### 21.5.2 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.

## Function Documentation

21.6.1 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>Config</i> | pointer to a <a href="#">gpio_pin_config_t</a> structure that contains the configuration information. |

### 21.6.2 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> | GPIO pin output logic level. <ul style="list-style-type: none"> <li>• 0: corresponding pin output low-logic level.</li> <li>• 1: corresponding pin output high-logic level.</li> </ul> |

### 21.6.3 static void GPIO\_WritePinOutput ( GPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinWrite](#).

### 21.6.4 static void GPIO\_PortSet ( 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                                          |

### 21.6.5 static void GPIO\_SetPinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PortSet](#).

**21.6.6** `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                                          |

**21.6.7 static void GPIO\_ClearPinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

**Deprecated** Do not use this function. It has been superceded by [GPIO\\_PortClear](#).

**21.6.8 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                                          |

**21.6.9 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. |
|-------------|-------------------|

**21.6.10 static uint32\_t GPIO\_ReadPinInput ( GPIO\_Type \* *base*, uint32\_t *pin* )**  
**[inline], [static]**

**Deprecated** Do not use this function. It has been superceded by [GPIO\\_PinRead](#).

**21.6.11**   `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. |
|-------------|-----------------------|

**21.6.12** `static uint8_t GPIO_ReadPadStatus ( GPIO_Type * base, uint32_t pin )  
[inline], [static]`

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinReadPadStatus](#).

**21.6.13** `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 <a href="#">gpio_interrupt_mode_t</a> structure that contains the interrupt mode information. |

**21.6.14** `static void GPIO_SetPinInterruptConfig ( GPIO_Type * base, uint32_t pin,  
gpio_interrupt_mode_t pinInterruptMode ) [inline], [static]`

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PinSetInterruptConfig](#).

**21.6.15** `static void GPIO_PortEnableInterrupts ( GPIO_Type * base, uint32_t mask  
) [inline], [static]`

## Function Documentation

### Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

**21.6.16 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. |

**21.6.17 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. |

**21.6.18 static void GPIO\_DisableInterrupts ( GPIO\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

**Deprecated** Do not use this function. It has been superseded by [GPIO\\_PortDisableInterrupts](#).

**21.6.19 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. |
|----------------|----------------------------|

**21.6.20 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. |
|----------------|----------------------------|

**21.6.21 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. |

**21.6.22 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

---

## Function Documentation

|             |                        |
|-------------|------------------------|
| <i>base</i> | GPIO base pointer.     |
| <i>mask</i> | GPIO pin number macro. |

## Chapter 22

# INTMUX: Interrupt Multiplexer Driver

### Overview

The MCUXpresso SDK provides a peripheral driver for the Interrupt Multiplexer (INTMUX) module of MCUXpresso SDK devices.

### Typical use case

#### 22.2.1 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, 3))  
    *< Version 2.0.3.*

### 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.*

## Macro Definition Documentation

### 22.3.1 #define FSL\_INTMUX\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))

## Enumeration Type Documentation

### 22.4.1 enum intmux\_channel\_logic\_mode\_t

Enumerator

***kINTMUX\_ChannelLogicOR*** Logic OR all enabled interrupt inputs.

***kINTMUX\_ChannelLogicAND*** Logic AND all enabled interrupt inputs.

## Function Documentation

### 22.5.1 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. |
|-------------|---------------------------------|

### 22.5.2 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. |
|-------------|---------------------------------|

### 22.5.3 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.      |

#### 22.5.4 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.  |

#### 22.5.5 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 INTMUX source. |

#### 22.5.6 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.                 |

**22.5.7 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 23

# IRQSTEER: Interrupt Request Steering Driver

### 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 - (uint32\_t)FSL\_FEATURE\_IRQSTEER\_IRQ\_START\_INDEX) % `IRQSTEER_INT_SRC_REG_WIDTH`)  
*IRQSTEER interrupt source mapping bit offset.*
- #define `IRQSTEER_INT_SRC_NUM`(regIndex, bitOffset) (((uint32\_t)FSL\_FEATURE\_IRQSTEER\_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.*

## Overview

- enum [irqsteer\\_int\\_master\\_t](#) {  
    [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, 2))  
    *< Version 2.0.2.*

## 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.*

## Macro Definition Documentation

**23.2.1 #define FSL\_IRQSTEER\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))**

**23.2.2 #define IRQSTEER\_INT\_SRC\_REG\_WIDTH 32U**

**23.2.3 #define IRQSTEER\_INT\_SRC\_REG\_INDEX( *irq* )**

**Value:**

```
((uint32_t)FSL_FEATURE_IRQSTEER_CHn_MASK_COUNT - 1U) - \
((irq - (uint32_t)FSL_FEATURE_IRQSTEER_IRQ_START_INDEX) /
IRQSTEER_INT_SRC_REG_WIDTH)
```

**23.2.4 #define IRQSTEER\_INT\_SRC\_BIT\_OFFSET( *irq* ) ((irq - (uint32\_t)FSL\_FEATURE\_IRQSTEER\_IRQ\_START\_INDEX) % IRQSTEER\_INT\_SRC\_REG\_WIDTH)**

**23.2.5 #define IRQSTEER\_INT\_SRC\_NUM( *regIndex*, *bitOffset* ) (((uint32\_t)FSL\_FEATURE\_IRQSTEER\_CHn\_MASK\_COUNT - 1U - (regIndex)) \* (IRQSTEER\_INT\_SRC\_REG\_WIDTH)) + (bitOffset))**

## Enumeration Type Documentation

**23.3.1 enum irqsteer\_int\_group\_t**

Enumerator

|                                          |                                                 |
|------------------------------------------|-------------------------------------------------|
| <b><i>kIRQSTEER_InterruptGroup0</i></b>  | Interrupt Group 0: interrupt source 31 - 0.     |
| <b><i>kIRQSTEER_InterruptGroup1</i></b>  | Interrupt Group 1: interrupt source 63 - 32.    |
| <b><i>kIRQSTEER_InterruptGroup2</i></b>  | Interrupt Group 2: interrupt source 95 - 64.    |
| <b><i>kIRQSTEER_InterruptGroup3</i></b>  | Interrupt Group 3: interrupt source 127 - 96.   |
| <b><i>kIRQSTEER_InterruptGroup4</i></b>  | Interrupt Group 4: interrupt source 159 - 128.  |
| <b><i>kIRQSTEER_InterruptGroup5</i></b>  | Interrupt Group 5: interrupt source 191 - 160.  |
| <b><i>kIRQSTEER_InterruptGroup6</i></b>  | Interrupt Group 6: interrupt source 223 - 192.  |
| <b><i>kIRQSTEER_InterruptGroup7</i></b>  | Interrupt Group 7: interrupt source 255 - 224.  |
| <b><i>kIRQSTEER_InterruptGroup8</i></b>  | Interrupt Group 8: interrupt source 287 - 256.  |
| <b><i>kIRQSTEER_InterruptGroup9</i></b>  | Interrupt Group 9: interrupt source 319 - 288.  |
| <b><i>kIRQSTEER_InterruptGroup10</i></b> | Interrupt Group 10: interrupt source 351 - 320. |
| <b><i>kIRQSTEER_InterruptGroup11</i></b> | Interrupt Group 11: interrupt source 383 - 352. |
| <b><i>kIRQSTEER_InterruptGroup12</i></b> | Interrupt Group 12: interrupt source 415 - 384. |

## Function Documentation

***kIRQSTEER\_InterruptGroup13*** Interrupt Group 13: interrupt source 447 - 416.  
***kIRQSTEER\_InterruptGroup14*** Interrupt Group 14: interrupt source 479 - 448.  
***kIRQSTEER\_InterruptGroup15*** Interrupt Group 15: interrupt source 511 - 480.

### 23.3.2 enum irqsteer\_int\_master\_t

Enumerator

***kIRQSTEER\_InterruptMaster0*** Interrupt Master 0: interrupt source 63 - 0.  
***kIRQSTEER\_InterruptMaster1*** Interrupt Master 1: interrupt source 127 - 64.  
***kIRQSTEER\_InterruptMaster2*** Interrupt Master 2: interrupt source 191 - 128.  
***kIRQSTEER\_InterruptMaster3*** Interrupt Master 3: interrupt source 255 - 192.  
***kIRQSTEER\_InterruptMaster4*** Interrupt Master 4: interrupt source 319 - 256.  
***kIRQSTEER\_InterruptMaster5*** Interrupt Master 5: interrupt source 383 - 320.  
***kIRQSTEER\_InterruptMaster6*** Interrupt Master 6: interrupt source 447 - 384.  
***kIRQSTEER\_InterruptMaster7*** Interrupt Master 7: interrupt source 511 - 448.

## Function Documentation

### 23.4.1 void IRQSTEER\_Init ( IRQSTEER\_Type \* *base* )

This function enables the clock gate for the specified IRQSTEER.

Parameters

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

### 23.4.2 void IRQSTEER\_Deinit ( IRQSTEER\_Type \* *base* )

The clock gate for the specified IRQSTEER is disabled.

Parameters

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

### 23.4.3 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. |

#### 23.4.4 static void IRQSTEER\_DisableInterrupt ( IRQSTEER\_Type \* *base*, IRQn\_Type *irq* ) [inline], [static]

## Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | IRQSTEER peripheral base address.                                  |
| <i>irq</i>  | Interrupt source number. The interrupt must be an IRQSTEER source. |

#### 23.4.5 static void IRQSTEER\_SetInterrupt ( IRQSTEER\_Type \* *base*, IRQn\_Type *irq*, bool *set* ) [inline], [static]

## Parameters

|             |                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------|
| <i>base</i> | IRQSTEER peripheral base address.                                                                        |
| <i>irq</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](#).

#### 23.4.6 static void IRQSTEER\_EnableMasterInterrupt ( IRQSTEER\_Type \* *base*, irqsteer\_int\_master\_t *intMasterIndex* ) [inline], [static]

By default, all the master interrupts are enabled.

## Function Documentation

### Parameters

|                       |                                                                                                                          |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | IRQSTEER peripheral base address.                                                                                        |
| <i>intMasterIndex</i> | Master index of interrupt sources to be routed, options available in enumeration <a href="#">irqsteer_int_master_t</a> . |

For example, to enable the interrupt sources of master 1:

```
* IRQSTEER_EnableMasterInterrupt (IRQSTEER_M4_0,
* kIRQSTEER_InterruptMaster1);
*
```

### 23.4.7 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, options available in enumeration <a href="#">irqsteer_int_master_t</a> . |

For example, to disable the interrupt sources of master 1:

```
* IRQSTEER_DisableMasterInterrupt (IRQSTEER_M4_0,
* kIRQSTEER_InterruptMaster1);
*
```

### 23.4.8 static bool IRQSTEER\_IsInterruptSet ( IRQSTEER\_Type \* *base*, IRQn\_Type *irq* ) [inline], [static]

### Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | 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.

For example, to check whether interrupt from output 0 of Display 1 is set:

```
* if (IRQSTEER_IsInterruptSet (IRQSTEER_DISPLAY1_INT_OUT0)
* {
* ...
* }
*
```



### 23.4.9 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](#).

### 23.4.10 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.

### 23.4.11 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, options available in enumeration <a href="#">irqsteer_int_-master_t</a> . |

### Returns

The current set next interrupt source number of one specific master. Return IRQSTEER\_INT\_Invalid if no interrupt set.

## Chapter 24

# ISI: Image Sensing Interface

### 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.

### Typical use case

#### 24.2.1 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_OutputRGBA8888;

 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)
 {
 }
}
```

## 24.2.2 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...](#)

### 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_AxiWriteErrorUInterrupt`,  
    `kISI_AxiWriteErrorYInterrupt`,  
    `kISI_AxiReadErrorInterrupt` = `ISI_CHNL_IER_AXI_RD_ERR_EN_MASK`,  
    `kISI_OverflowAlertVInterrupt`,  
    `kISI_ExcessOverflowVInterrupt`,  
    `kISI_OverflowVInterrupt` = `ISI_CHNL_IER_OFLW_V_BUF_EN_MASK`,  
    `kISI_OverflowAlertUInterrupt`,  
    `kISI_ExcessOverflowUInterrupt`,  
    `kISI_OverflowUInterrupt` = `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 = 60U,
kISI_OutputYUV420_3P12 = 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

- Get the ISI interrupt pending flags.*
- 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.
- 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.

Data Structure Documentation

24.3.1 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

24.3.1.0.0.55 Field Documentation

- 24.3.1.0.0.55.1 `bool isi_config_t::isChannelBypassed`
- 24.3.1.0.0.55.2 `bool isi_config_t::isSourceMemory`
- 24.3.1.0.0.55.3 `bool isi_config_t::isYCbCr`
- 24.3.1.0.0.55.4 `isi_chain_mode_t isi_config_t::chainMode`
- 24.3.1.0.0.55.5 `isi_deint_mode_t isi_config_t::deintMode`
- 24.3.1.0.0.55.6 `uint8_t isi_config_t::blankPixel`
- 24.3.1.0.0.55.7 `uint8_t isi_config_t::sourcePort`
- 24.3.1.0.0.55.8 `uint8_t isi_config_t::mipiChannel`
- 24.3.1.0.0.55.9 `uint16_t isi_config_t::inputHeight`
- 24.3.1.0.0.55.10 `uint16_t isi_config_t::inputWidth`
- 24.3.1.0.0.55.11 `isi_output_format_t isi_config_t::outputFormat`
- 24.3.1.0.0.55.12 `isi_threshold_t isi_config_t::thresholdY`
- 24.3.1.0.0.55.13 `isi_threshold_t isi_config_t::thresholdU`
- 24.3.1.0.0.55.14 `isi_threshold_t isi_config_t::thresholdV`

24.3.2 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

24.3.2.0.0.56 Field Documentation

24.3.2.0.0.56.1 `isi_csc_mode_t` `isi_csc_config_t::mode`

24.3.2.0.0.56.2 `float` `isi_csc_config_t::A1`

24.3.2.0.0.56.3 `float` `isi_csc_config_t::A2`

24.3.2.0.0.56.4 `float` `isi_csc_config_t::A3`

24.3.2.0.0.56.5 `float` `isi_csc_config_t::B1`

24.3.2.0.0.56.6 `float` `isi_csc_config_t::B2`

24.3.2.0.0.56.7 `float` `isi_csc_config_t::B3`

24.3.2.0.0.56.8 `float` `isi_csc_config_t::C1`

24.3.2.0.0.56.9 `float` `isi_csc_config_t::C2`

24.3.2.0.0.56.10 `float` `isi_csc_config_t::C3`

24.3.2.0.0.56.11 `int16_t` `isi_csc_config_t::D1`

24.3.2.0.0.56.12 `int16_t` `isi_csc_config_t::D2`

24.3.2.0.0.56.13 `int16_t` `isi_csc_config_t::D3`

24.3.3 `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.

24.3.3.0.0.57 Field Documentation**24.3.3.0.0.57.1** uint16_t isi_crop_config_t::upperLeftX**24.3.3.0.0.57.2** uint16_t isi_crop_config_t::upperLeftY**24.3.3.0.0.57.3** uint16_t isi_crop_config_t::lowerRightX**24.3.3.0.0.57.4** uint16_t isi_crop_config_t::lowerRightY**24.3.4 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.

24.3.4.0.0.58 Field Documentation**24.3.4.0.0.58.1** uint16_t isi_region_alpha_config_t::upperLeftX**24.3.4.0.0.58.2** uint16_t isi_region_alpha_config_t::upperLeftY**24.3.4.0.0.58.3** uint16_t isi_region_alpha_config_t::lowerRightX**24.3.4.0.0.58.4** uint16_t isi_region_alpha_config_t::lowerRightY**24.3.4.0.0.58.5** uint8_t isi_region_alpha_config_t::alpha**24.3.5 struct isi_input_mem_config_t****Data Fields**

- uint32_t [addr](#)
Address of the input memory.
- uint16_t [linePitchBytes](#)
Line pitch in bytes.
- uint16_t [framePitchBytes](#)
Frame pitch in bytes.
- [isi_input_mem_format_t](#) [format](#)
Image format of the input memory.

Enumeration Type Documentation

24.3.5.0.0.59 Field Documentation

24.3.5.0.0.59.1 `uint32_t isi_input_mem_config_t::addr`

24.3.5.0.0.59.2 `uint16_t isi_input_mem_config_t::linePitchBytes`

24.3.5.0.0.59.3 `uint16_t isi_input_mem_config_t::framePitchBytes`

24.3.5.0.0.59.4 `isi_input_mem_format_t isi_input_mem_config_t::format`

Macro Definition Documentation

24.4.1 `#define FSL_ISI_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

Version 2.0.1.

Enumeration Type Documentation

24.5.1 `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_OverflowUInterrupt 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.

24.5.2 `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).

24.5.3 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.

24.5.4 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.

24.5.5 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.

24.5.6 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.

24.5.7 enum isi_flip_mode_t

Enumerator

kISI_FlipDisable Flip disabled.

kISI_FlipHorizontal Horizontal flip.

kISI_FlipVertical Vertical flip.

kISI_FlipBoth Flip both direction.

24.5.8 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).

Function Documentation

24.6.1 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.
-------------	------------------------------

24.6.2 void ISI_Deinit (ISI_Type * *base*)

This function gates the ISI clock.

Parameters

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

24.6.3 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.
-------------	------------------------------

24.6.4 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 _isi_interrupt .

Returns

OR'ed value of the enabled interrupts before calling this function.

24.6.5 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 _isi_interrupt .

Returns

OR'ed value of the enabled interrupts before calling this function.

Function Documentation

24.6.6 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](#).

24.6.7 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 _isi_interrupt .

24.6.8 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.

24.6.9 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.

24.6.10 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.
---------------	---

24.6.11 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.

24.6.12 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.

24.6.13 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.
---------------	---

24.6.14 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.

24.6.15 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.

24.6.16 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.
---------------	---

24.6.17 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.

24.6.18 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.

24.6.19 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.

24.6.20 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.

24.6.21 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.
---------------	---

24.6.22 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.

24.6.23 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.

24.6.24 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.
---------------	---

24.6.25 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.

24.6.26 void ISI_TriggerInputMemRead (ISI_Type * *base*)

Parameters

<i>base</i>	ISI peripheral base address
-------------	-----------------------------

24.6.27 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.

24.6.28 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.

24.6.29 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
-------------	-----------------------------

24.6.30 static void ISI_Stop (ISI_Type * *base*) [inline], [static]

Parameters

<i>base</i>	ISI peripheral base address
-------------	-----------------------------

Chapter 25

LDB: LVDS Display Bridge

Overview

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.

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.

Function Documentation

Driver version

- #define **FSL_LDB_DRIVER_VERSION** (**MAKE_VERSION**(2, 0, 2))
LDB driver version.

Data Structure Documentation

25.2.1 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.

25.2.1.0.0.60 Field Documentation

25.2.1.0.0.60.1 **ldb_output_bus_t** ldb_channel_config_t::outputBus

25.2.1.0.0.60.2 **uint32_t** ldb_channel_config_t::inputFlag

25.2.1.0.0.60.3 **uint32_t** ldb_channel_config_t::pixelClock_Hz

Macro Definition Documentation

25.3.1 #define **FSL_LDB_DRIVER_VERSION** (**MAKE_VERSION**(2, 0, 2))

Enumeration Type Documentation

25.4.1 enum **ldb_output_bus_t**

25.4.2 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.

Function Documentation

25.5.1 void **LDB_Init** (**LDB_Type** * **base**)

Parameters

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

25.5.2 void LDB_Deinit (LDB_Type * *base*)

Parameters

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

25.5.3 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.

25.5.4 void LDB_DeinitChannel (LDB_Type * *base*, uint8_t *channel*)

Parameters

<i>base</i>	LDB peripheral base address.
<i>channel</i>	Channel index.

Chapter 26

LPADC: 12-bit SAR Analog-to-Digital Converter Driver

Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit SAR Analog-to-Digital Converter (LP-ADC) module of MCUXpresso SDK devices.

Typical use case

26.2.1 Polling Configuration

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

26.2.2 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_STATUE](#)(statusVal) ((statusVal & ADC_STAT_TRGACT_MASK) >> ADC_STAT_TRGACT_SHIFT)
Define the MACRO function to get trigger status from status value.

Enumerations

- enum `_lpadc_status_flags` {
 `kLPADC_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_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 }
    Define enumeration of trigger priority policy.

```

Driver version

- #define **FSL_LPADC_DRIVER_VERSION** (**MAKE_VERSION**(2, 2, 2))
LPADC driver version 2.2.2.

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)
Enable interrupts.
- 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.

Data Structure Documentation

26.3.1 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.

26.3.1.0.0.61 Field Documentation

26.3.1.0.0.61.1 `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.

26.3.1.0.0.61.2 `bool lpadc_config_t::enableAnalogPreliminary`

26.3.1.0.0.61.3 `uint32_t lpadc_config_t::powerUpDelay`

The startup delay count of $(\text{powerUpDelay} * 4)$ ADCK cycles must result in a longer delay than the analog startup time.

26.3.1.0.0.61.4 `lpadc_reference_voltage_source_t lpadc_config_t::referenceVoltageSource`

26.3.1.0.0.61.5 `lpadc_power_level_mode_t lpadc_config_t::powerLevelMode`

26.3.1.0.0.61.6 `lpadc_trigger_priority_policy_t lpadc_config_t::triggerPriorityPolicy`

26.3.1.0.0.61.7 `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.

26.3.1.0.0.61.8 `uint32_t lpadc_config_t::convPauseDelay`

The pause delay is a count of $(\text{convPauseDelay} * 4)$ ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

26.3.1.0.0.61.9 `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.

26.3.2 `struct lpadc_conv_command_config_t`

Data Fields

- `lpadc_sample_scale_mode_t sampleScaleMode`
Sample scale mode.

Data Structure Documentation

- `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.

26.3.2.0.0.62 Field Documentation

26.3.2.0.0.62.1 `lpadc_sample_scale_mode_t` `lpadc_conv_command_config_t::sampleScaleMode`

26.3.2.0.0.62.2 `lpadc_sample_channel_mode_t` `lpadc_conv_command_config_t::sampleChannelMode`

26.3.2.0.0.62.3 `uint32_t` `lpadc_conv_command_config_t::channelNumber`

26.3.2.0.0.62.4 `uint32_t` `lpadc_conv_command_config_t::chainedNextCommandNumber`

1-15 is available, 0 is to terminate the chain after this command.

26.3.2.0.0.62.5 `bool` `lpadc_conv_command_config_t::enableAutoChannelIncrement`

26.3.2.0.0.62.6 `uint32_t` `lpadc_conv_command_config_t::loopCount`

Command executes LOOP+1 times. 0-15 is available.

26.3.2.0.0.62.7 `lpadc_hardware_average_mode_t` `lpadc_conv_command_config_t::hardwareAverageMode`

26.3.2.0.0.62.8 `lpadc_sample_time_mode_t` `lpadc_conv_command_config_t::sampleTimeMode`

26.3.2.0.0.62.9 `lpadc_hardware_compare_mode_t` `lpadc_conv_command_config_t::hardwareCompareMode`

26.3.2.0.0.62.10 `uint32_t` `lpadc_conv_command_config_t::hardwareCompareValueHigh`

The available value range is in 16-bit.

26.3.2.0.0.62.11 `uint32_t` `lpadc_conv_command_config_t::hardwareCompareValueLow`

The available value range is in 16-bit.

26.3.3 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.

26.3.3.0.0.63 Field Documentation

26.3.3.0.0.63.1 `uint32_t` `lpadc_conv_trigger_config_t::targetCommandId`

26.3.3.0.0.63.2 `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.

26.3.3.0.0.63.3 `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.

26.3.3.0.0.63.4 `bool` `lpadc_conv_trigger_config_t::enableHardwareTrigger`

The software trigger is always available.

Enumeration Type Documentation

26.3.4 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.

26.3.4.0.0.64 Field Documentation

26.3.4.0.0.64.1 uint32_t lpadc_conv_result_t::commandIdSource

26.3.4.0.0.64.2 uint32_t lpadc_conv_result_t::loopCountIndex

26.3.4.0.0.64.3 uint32_t lpadc_conv_result_t::triggerIdSource

26.3.4.0.0.64.4 uint16_t lpadc_conv_result_t::convValue

Macro Definition Documentation

26.4.1 #define FSL_LPADC_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))

26.4.2 #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\(\)](#).

26.4.3 #define LPADC_GET_ACTIVE_TRIGGER_STATUE(*statusVal*) ((statusVal & ADC_STAT_TRGACT_MASK) >> ADC_STAT_TRGACT_SHIFT)

The statusVal is the return value from [LPADC_GetStatusFlags\(\)](#).

Enumeration Type Documentation

26.5.1 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.

26.5.2 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.

26.5.3 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).

26.5.4 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 minue side.

kLPADC_SampleChannelDiffBothSideBA Differential mode, using B as plus side and A as minue side.

26.5.5 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.

26.5.6 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.

26.5.7 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.

26.5.8 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.

26.5.9 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.

26.5.10 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.

Function Documentation**26.6.1 void LPADC_Init (ADC_Type * *base*, const lpadc_config_t * *config*)**Parameters

Function Documentation

<i>base</i>	LPADC peripheral base address.
<i>config</i>	Pointer to configuration structure. See "lpadc_config_t".

26.6.2 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.
---------------	-------------------------------------

26.6.3 void LPADC_Deinit (ADC_Type * *base*)

Parameters

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

26.6.4 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.
---------------	-------------------------

**26.6.5 static void LPADC_DoResetFIFO (ADC_Type * *base*) [inline],
[static]**

Parameters

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

**26.6.6 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.
-------------	--------------------------------

**26.6.7 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](#).

**26.6.8 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 _lpadc_status_flags .

26.6.9 static void LPADC_EnableInterrupts (ADC_Type * *base*, uint32_t *mask*)
[inline], [static]

Parameters

<i>base</i>	LPADC peripheral base address.
<i>mask</i>	Mask value for interrupt events. See to _lpadc_interrupt_enable .

26.6.10 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 _lpadc_interrupt_enable .

26.6.11 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.

26.6.12 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.

26.6.13 **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.

26.6.14 **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 lpadc_conv_trigger_config_t .

26.6.15 **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.
---------------	-------------------------------------

26.6.16 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.

26.6.17 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 lpadc_conv_command_config_t .

26.6.18 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              = 0U;
* config->chainedNextCmdNumber       = 0U;
* config->enableAutoChannelIncrement = false;
* config->loopCount                  = 0U;
* config->hardwareAverageMode        = kLPADC_HardwareAverageCount1;
* config->sampleTimeMode             = kLPADC_SampleTimeADCK3;
* config->hardwareCompareMode        = kLPADC_HardwareCompareDisabled;
* config->hardwareCompareValueHigh   = 0U;
* config->hardwareCompareValueLow    = 0U;
* config->conversionResolutionMode   = kLPADC_ConversionResolutionStandard;
* config->enableWaitTrigger          = false;
*
```



Parameters

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

Chapter 27

LPI2C: Low Power Inter-Integrated Circuit Driver

Overview

Modules

- LPI2C CMSIS Driver
- LPI2C FreeRTOS Driver
- LPI2C Master DMA Driver
- LPI2C Master Driver
- LPI2C Slave Driver

Macros

- #define `I2C_RETRY_TIMES` 0U /* Define to zero means keep waiting until the flag is asserted/deassert. */
Retry times for waiting flag.

Enumerations

- enum {
 `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, 11))
LPI2C driver version 2.1.11.

Enumeration Type Documentation

Macro Definition Documentation

27.2.1 `#define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 1, 11))`

27.2.2 `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

Enumeration Type Documentation

27.3.1 anonymous enum

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.

LPI2C Master Driver

27.4.1 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 {
[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 baud-Rate_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.

27.4.2 Data Structure Documentation

27.4.2.1 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.

27.4.2.1.0.65 Field Documentation

27.4.2.1.0.65.1 bool `lpi2c_master_config_t::enableMaster`

27.4.2.1.0.65.2 bool `lpi2c_master_config_t::enableDoze`

27.4.2.1.0.65.3 bool `lpi2c_master_config_t::debugEnable`

27.4.2.1.0.65.4 bool `lpi2c_master_config_t::ignoreAck`

27.4.2.1.0.65.5 `lpi2c_master_pin_config_t` `lpi2c_master_config_t::pinConfig`

27.4.2.1.0.65.6 `uint32_t` `lpi2c_master_config_t::baudRate_Hz`

27.4.2.1.0.65.7 `uint32_t` `lpi2c_master_config_t::busIdleTimeout_ns`

Set to 0 to disable.

LPI2C Master Driver

27.4.2.1.0.65.8 `uint32_t lpi2c_master_config_t::pinLowTimeout_ns`

Set to 0 to disable.

27.4.2.1.0.65.9 `uint8_t lpi2c_master_config_t::sdaGlitchFilterWidth_ns`

Set to 0 to disable.

27.4.2.1.0.65.10 `uint8_t lpi2c_master_config_t::sclGlitchFilterWidth_ns`

Set to 0 to disable.

27.4.2.1.0.65.11 `bool lpi2c_master_config_t::enable`

27.4.2.1.0.65.12 `lpi2c_host_request_source_t lpi2c_master_config_t::source`

27.4.2.1.0.65.13 `lpi2c_host_request_polarity_t lpi2c_master_config_t::polarity`

27.4.2.1.0.65.14 `struct { ... } lpi2c_master_config_t::hostRequest`

27.4.2.2 `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.

27.4.2.2.0.66 Field Documentation

27.4.2.2.0.66.1 `lpi2c_data_match_config_mode_t lpi2c_data_match_config_t::matchMode`

27.4.2.2.0.66.2 `bool lpi2c_data_match_config_t::rxDataMatchOnly`

27.4.2.2.0.66.3 `uint32_t lpi2c_data_match_config_t::match0`

27.4.2.2.0.66.4 `uint32_t lpi2c_data_match_config_t::match1`

27.4.2.3 `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`
Bit mask of options for the transfer.
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.

27.4.2.3.0.67 Field Documentation

27.4.2.3.0.67.1 `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.

27.4.2.3.0.67.2 `uint16_t lpi2c_master_transfer_t::slaveAddress`

27.4.2.3.0.67.3 `lpi2c_direction_t lpi2c_master_transfer_t::direction`

27.4.2.3.0.67.4 `uint32_t lpi2c_master_transfer_t::subaddress`

Transferred MSB first.

27.4.2.3.0.67.5 `size_t lpi2c_master_transfer_t::subaddressSize`

Maximum size is 4 bytes.

27.4.2.3.0.67.6 `void* lpi2c_master_transfer_t::data`

27.4.2.3.0.67.7 `size_t lpi2c_master_transfer_t::dataSize`

27.4.2.4 `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

- *Buffer pointer for current state.*
uint16_t [commandBuffer](#) [7]
- *LPI2C command sequence.*
lpi2c_master_transfer_t [transfer](#)
- *Copy of the current transfer info.*
lpi2c_master_transfer_callback_t [completionCallback](#)
- *Callback function pointer.*
void * [userData](#)
- *Application data passed to callback.*

27.4.2.4.0.68 Field Documentation

27.4.2.4.0.68.1 uint8_t lpi2c_master_handle_t::state

27.4.2.4.0.68.2 uint16_t lpi2c_master_handle_t::remainingBytes

27.4.2.4.0.68.3 uint8_t* lpi2c_master_handle_t::buf

27.4.2.4.0.68.4 uint16_t lpi2c_master_handle_t::commandBuffer[7]

27.4.2.4.0.68.5 lpi2c_master_transfer_t lpi2c_master_handle_t::transfer

27.4.2.4.0.68.6 lpi2c_master_transfer_callback_t lpi2c_master_handle_t::completionCallback

27.4.2.4.0.68.7 void* lpi2c_master_handle_t::userData

27.4.3 Typedef Documentation

27.4.3.1 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

<i>base</i>	The LPI2C peripheral base address.
<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.

27.4.4 Enumeration Type Documentation

27.4.4.1 anonymous enum

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.

27.4.4.2 enum lpi2c_direction_t

Enumerator

kLPI2C_Write Master transmit.
kLPI2C_Read Master receive.

27.4.4.3 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)

27.4.4.4 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.

27.4.4.5 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.

27.4.4.6 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.

27.4.4.7 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.

27.4.5 Function Documentation

27.4.5.1 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	<i>masterConfig</i>	User provided configuration structure for default values. Refer to <code>lpi2c_master_config_t</code> .
-----	---------------------	---

27.4.5.2 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 LPI2C_MasterGetDefaultConfig() 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.

27.4.5.3 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.
-------------	------------------------------------

27.4.5.4 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.

27.4.5.5 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.
-------------	------------------------------------

27.4.5.6 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.

27.4.5.7 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](#)

27.4.5.8 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 <code>_lpi2c_master_flags</code> enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags() .

See Also

`_lpi2c_master_flags`.

27.4.5.9 **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 <code>_lpi2c_master_flags</code> for the set of constants that should be OR'd together to form the bit mask.

27.4.5.10 **static void LPI2C_MasterDisableInterrupts (LPI2C_Type * *base*, uint32_t *interruptMask*) [inline], [static]**

All flags except [kLPI2C_MasterBusyFlag](#) and [kLPI2C_MasterBusBusyFlag](#) can be enabled as interrupts.

Parameters

<i>base</i>	The LPI2C peripheral base address.
<i>interruptMask</i>	Bit mask of interrupts to disable. See <code>_lpi2c_master_flags</code> for the set of constants that should be OR'd together to form the bit mask.

27.4.5.11 **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.

27.4.5.12 `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.

27.4.5.13 `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.

27.4.5.14 `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.

27.4.5.15 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 kLPI2C_MasterTxReadyFlag 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 kLPI2C_MasterRxReadyFlag 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.

27.4.5.16 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.

27.4.5.17 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.

27.4.5.18 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.

27.4.5.19 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 kLPI2C_Read or kLPI2C_Write . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

<i>kStatus_Success</i>	START signal and address were successfully enqueued in the transmit FIFO.
<i>kStatus_LPI2C_Busy</i>	Another master is currently utilizing the bus.

27.4.5.20 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

<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 kLPI2C_Read or kLPI2C_Write . This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

<i>kStatus_Success</i>	Repeated START signal and address were successfully enqueued in the transmit FIFO.
<i>kStatus_LPI2C_Busy</i>	Another master is currently utilizing the bus.

27.4.5.21 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.

27.4.5.22 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.

27.4.5.23 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.

27.4.5.24 `status_t LPI2C_MasterTransferBlocking (LPI2C_Type * base, lpi2c_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_PinLow-Timeout</i>	SCL or SDA were held low longer than the timeout.

**27.4.5.25 void LPI2C_MasterTransferCreateHandle (LPI2C_Type * *base*,
lpi2c_master_handle_t * *handle*, lpi2c_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.

**27.4.5.26 status_t LPI2C_MasterTransferNonBlocking (LPI2C_Type * *base*,
lpi2c_master_handle_t * *handle*, lpi2c_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

<i>kStatus_Success</i>	The transaction was started successfully.
<i>kStatus_LPI2C_Busy</i>	Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

27.4.5.27 `status_t LPI2C_MasterTransferGetCount (LPI2C_Type * base,
lpi2c_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

<i>kStatus_Success</i>	
<i>kStatus_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

27.4.5.28 `void LPI2C_MasterTransferAbort (LPI2C_Type * base, lpi2c_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

<i>kStatus_Success</i>	A transaction was successfully aborted.
<i>kStatus_LPI2C_Idle</i>	There is not a non-blocking transaction currently in progress.

27.4.5.29 void LPI2C_MasterTransferHandleIRQ (LPI2C_Type * *base*,
lpi2c_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

27.5.1 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 enable-Rx, 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.

27.5.2 Data Structure Documentation

27.5.2.1 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

27.5.2.1.0.69 Field Documentation

27.5.2.1.0.69.1 `bool lpi2c_slave_config_t::enableSlave`

27.5.2.1.0.69.2 `uint8_t lpi2c_slave_config_t::address0`

27.5.2.1.0.69.3 `uint8_t lpi2c_slave_config_t::address1`

27.5.2.1.0.69.4 `lpi2c_slave_address_match_t lpi2c_slave_config_t::addressMatchMode`

27.5.2.1.0.69.5 `bool lpi2c_slave_config_t::filterDozeEnable`

27.5.2.1.0.69.6 `bool lpi2c_slave_config_t::filterEnable`

27.5.2.1.0.69.7 `bool lpi2c_slave_config_t::enableGeneralCall`

27.5.2.1.0.69.8 `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`.

27.5.2.1.0.69.9 `bool lpi2c_slave_config_t::enableTx`

27.5.2.1.0.69.10 `bool lpi2c_slave_config_t::enableRx`

27.5.2.1.0.69.11 `bool lpi2c_slave_config_t::enableAddress`

27.5.2.1.0.69.12 `bool lpi2c_slave_config_t::ignoreAck`

27.5.2.1.0.69.13 `bool lpi2c_slave_config_t::enableReceivedAddressRead`

27.5.2.1.0.69.14 `uint32_t lpi2c_slave_config_t::sdaGlitchFilterWidth_ns`

27.5.2.1.0.69.15 `uint32_t lpi2c_slave_config_t::sclGlitchFilterWidth_ns`

27.5.2.1.0.69.16 `uint32_t lpi2c_slave_config_t::dataValidDelay_ns`

27.5.2.1.0.69.17 `uint32_t lpi2c_slave_config_t::clockHoldTime_ns`

27.5.2.2 `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.

27.5.2.2.0.70 Field Documentation

27.5.2.2.0.70.1 [lpi2c_slave_transfer_event_t lpi2c_slave_transfer_t::event](#)

27.5.2.2.0.70.2 [uint8_t lpi2c_slave_transfer_t::receivedAddress](#)

27.5.2.2.0.70.3 [status_t lpi2c_slave_transfer_t::completionStatus](#)

Only applies for [kLPI2C_SlaveCompletionEvent](#).

27.5.2.2.0.70.4 [size_t lpi2c_slave_transfer_t::transferredCount](#)

27.5.2.3 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

27.5.2.3.0.71 Field Documentation

27.5.2.3.0.71.1 `lpi2c_slave_transfer_t lpi2c_slave_handle_t::transfer`

27.5.2.3.0.71.2 `bool lpi2c_slave_handle_t::isBusy`

27.5.2.3.0.71.3 `bool lpi2c_slave_handle_t::wasTransmit`

27.5.2.3.0.71.4 `uint32_t lpi2c_slave_handle_t::eventMask`

27.5.2.3.0.71.5 `uint32_t lpi2c_slave_handle_t::transferredCount`

27.5.2.3.0.71.6 `lpi2c_slave_transfer_callback_t lpi2c_slave_handle_t::callback`

27.5.2.3.0.71.7 `void* lpi2c_slave_handle_t::userData`

27.5.3 Typedef Documentation

27.5.3.1 `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.

27.5.4 Enumeration Type Documentation

27.5.4.1 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.

27.5.4.2 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.

27.5.4.3 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.

27.5.5 Function Documentation

27.5.5.1 void LPI2C_SlaveGetDefaultConfig (lpi2c_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

out	<i>slaveConfig</i>	User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t .
-----	--------------------	--

27.5.5.2 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 LPI2C_SlaveGetDefaultConfig() 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.

27.5.5.3 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.
-------------	------------------------------------

27.5.5.4 static void LPI2C_SlaveReset (LPI2C_Type * *base*) [inline], [static]

Parameters

LPI2C Slave Driver

<i>base</i>	The LPI2C peripheral base address.
-------------	------------------------------------

27.5.5.5 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.

27.5.5.6 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](#)

27.5.5.7 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 _lpi2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags() .

See Also

[_lpi2c_slave_flags](#).

27.5.5.8 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 _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

27.5.5.9 static void LPI2C_SlaveDisableInterrupts (LPI2C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

All flags except [kLPI2C_SlaveBusyFlag](#) and [kLPI2C_SlaveBusBusyFlag](#) can be enabled as interrupts.

Parameters

<i>base</i>	The LPI2C peripheral base address.
<i>interruptMask</i>	Bit mask of interrupts to disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

27.5.5.10 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.

27.5.5.11 `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.

27.5.5.12 `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.

27.5.5.13 `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.

27.5.5.14 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.

27.5.5.15 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.

27.5.5.16 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.

**27.5.5.17 void LPI2C_SlaveTransferCreateHandle (LPI2C_Type * *base*,
lpi2c_slave_handle_t * *handle*, lpi2c_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.

**27.5.5.18 status_t LPI2C_SlaveTransferNonBlocking (LPI2C_Type * *base*,
lpi2c_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 [lpi2c_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 lpi2c_slave_transfer_event_t 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 kLPI2C_SlaveAllEvents to enable all events.

Return values

<i>kStatus_Success</i>	Slave transfers were successfully started.
<i>kStatus_LPI2C_Busy</i>	Slave transfers have already been started on this handle.

27.5.5.19 `status_t LPI2C_SlaveTransferGetCount (LPI2C_Type * base,
lpi2c_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.
out	<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

<i>kStatus_Success</i>	
<i>kStatus_NoTransferInProgress</i>	

27.5.5.20 `void LPI2C_SlaveTransferAbort (LPI2C_Type * base, lpi2c_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

<i>kStatus_Success</i>	
<i>kStatus_LPI2C_Idle</i>	

27.5.5.21 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.

LPI2C Master DMA Driver

27.6.1 Overview

Data Structures

- struct [lpi2c_master_edma_handle_t](#)
Driver handle for master DMA APIs. [More...](#)

Typedefs

- typedef void(* [lpi2c_master_edma_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.

27.6.2 Data Structure Documentation

27.6.2.1 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** [3]
- *Software TCD.*

27.6.2.1.0.72 Field Documentation

27.6.2.1.0.72.1 LPI2C_Type* lpi2c_master_edma_handle_t::base

27.6.2.1.0.72.2 bool lpi2c_master_edma_handle_t::isBusy

27.6.2.1.0.72.3 uint8_t lpi2c_master_edma_handle_t::nbytes

27.6.2.1.0.72.4 uint16_t lpi2c_master_edma_handle_t::commandBuffer[7]

27.6.2.1.0.72.5 lpi2c_master_transfer_t lpi2c_master_edma_handle_t::transfer

27.6.2.1.0.72.6 lpi2c_master_edma_transfer_callback_t lpi2c_master_edma_handle_t::completionCallback

27.6.2.1.0.72.7 void* lpi2c_master_edma_handle_t::userData

27.6.2.1.0.72.8 edma_handle_t* lpi2c_master_edma_handle_t::rx

27.6.2.1.0.72.9 edma_handle_t* lpi2c_master_edma_handle_t::tx

27.6.2.1.0.72.10 edma_tcd_t lpi2c_master_edma_handle_t::tcds[3]

Three are allocated to provide enough room to align to 32-bytes.

27.6.3 Typedef Documentation

27.6.3.1 `typedef void(* lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,
lpi2c_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.

27.6.4 Function Documentation

27.6.4.1 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*)

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.

27.6.4.2 status_t LPI2C_MasterTransferEDMA (LPI2C_Type * *base*, lpi2c_master_edma_handle_t * *handle*, lpi2c_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

<i>kStatus_Success</i>	The transaction was started successfully.
<i>kStatus_LPI2C_Busy</i>	Either another master is currently utilizing the bus, or another DMA transaction is already in progress.

27.6.4.3 **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.
out	<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Return values

<i>kStatus_Success</i>	
<i>kStatus_NoTransferInProgress</i>	There is not a DMA transaction currently in progress.

27.6.4.4 **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.

LPI2C FreeRTOS Driver

27.7.1 Overview

Driver version

- #define **FSL_LPI2C_FREERTOS_DRIVER_VERSION** (MAKE_VERSION(2, 1, 11))
LPI2C FreeRTOS driver version 2.1.11.

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.

27.7.2 Macro Definition Documentation

27.7.2.1 #define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 11))

27.7.3 Function Documentation

27.7.3.1 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.

27.7.3.2 status_t LPI2C_RTOS_Deinit (lpi2c_rtos_handle_t * *handle*)

This function deinitializes the LPI2C module and related RTOS context.

Parameters

<i>handle</i>	The RTOS LPI2C handle.
---------------	------------------------

**27.7.3.3 status_t LPI2C_RTOS_Transfer (lpi2c_rtos_handle_t * *handle*,
lpi2c_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.

LPI2C CMSIS Driver

This section describes the programming interface of the LPI2C Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method see <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The LPI2C CMSIS driver includes transactional APIs.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code accessing the hardware registers.

27.8.1 LPI2C CMSIS Driver

27.8.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}

/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
Driver_I2C0.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

27.8.1.2 Master Operation in DMA transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    /* Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}

/* DMAMux init and EDMA init. */
DMAMUX_Init(EXAMPLE_LPI2C_DMAMUX_BASEADDR);
```

```

edma_config_t edmaConfig;
EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(EXAMPLE_LPI2C_DMA_BASEADDR, &edmaConfig);

/*Init I2C0*/
Driver_I2C0.Initialize(I2C_MasterSignalEvent_t);

Driver_I2C0.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
Driver_I2C0.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transfer*/
Driver_I2C0.MasterReceive(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;

```

27.8.1.3 Slave Operation in interrupt transactional method

```

void I2C_SlaveSignalEvent_t(uint32_t event)
{
    /* Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_SlaveCompletionFlag = true;
    }
}

/*Init I2C1*/
Driver_I2C1.Initialize(I2C_SlaveSignalEvent_t);

Driver_I2C1.PowerControl(ARM_POWER_FULL);

/*config slave addr*/
Driver_I2C1.Control(ARM_I2C_OWN_ADDRESS, I2C_MASTER_SLAVE_ADDR);

/*start transfer*/
Driver_I2C1.SlaveReceive(g_slave_buff, I2C_DATA_LENGTH);

/* Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
g_SlaveCompletionFlag = false;

```


Chapter 28

LPIT: Low-Power Interrupt Timer

Overview

The MCUXpresso SDK provides a driver for the Low-Power Interrupt Timer (LPIT) of MCUXpresso SDK devices.

Function groups

The LPIT driver supports operating the module as a time counter.

28.2.1 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.

28.2.2 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

28.2.3 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

28.2.4 Status

Provides functions to get and clear the LPIT status.

28.2.5 Interrupt

Provides functions to enable/disable LPIT interrupts and get current enabled interrupts.

Typical use case

28.3.1 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, 1))
Version 2.0.1.

Initialization and deinitialization

- void `LPIT_Init` (LPIT_Type *base, const `lpit_config_t` *config)
Un-gates 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.

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.

Data Structure Documentation

28.4.1 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

28.4.1.0.0.73 Field Documentation

28.4.1.0.0.73.1 lpit_timer_modes_t lpit_chnl_params_t::timerMode

28.4.1.0.0.73.2 lpit_trigger_source_t lpit_chnl_params_t::triggerSource

28.4.2 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

Enumeration Type Documentation

28.5.1 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.

28.5.2 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

28.5.3 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.

28.5.4 enum lpit_trigger_source_t

Enumerator

<i>kLPIT_TriggerSource_External</i>	Use external trigger input.
<i>kLPIT_TriggerSource_Internal</i>	Use internal trigger.

28.5.5 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.

28.5.6 enum lpit_status_flags_t

Note

Number of timer channels are SoC-specific. See the SoC Reference Manual.

Enumerator

kLPIT_Channel0TimerFlag Channel 0 Timer interrupt flag.
kLPIT_Channel1TimerFlag Channel 1 Timer interrupt flag.
kLPIT_Channel2TimerFlag Channel 2 Timer interrupt flag.
kLPIT_Channel3TimerFlag Channel 3 Timer interrupt flag.

Function Documentation

28.6.1 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.

28.6.2 void LPIT_Deinit (LPIT_Type * *base*)

Parameters

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

28.6.3 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.
---------------	--

28.6.4 **status_t LPIT_SetupChannel (LPIT_Type * *base*, lpit_chnl_t *channel*, const lpit_chnl_params_t * *chnlSetup*)**

This function sets up the operation mode to one of the options available in the enumeration [lpit_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.

28.6.5 **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 lpit_interrupt_enable_t

28.6.6 **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 lpit_interrupt_enable_t

28.6.7 `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](#)

28.6.8 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](#)

28.6.9 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 lpit_status_flags_t

28.6.10 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.

28.6.11 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.

28.6.12 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.

28.6.13 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.

28.6.14 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 29

LPSPI: Low Power Serial Peripheral Interface

Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power Serial Peripheral Interface (LPSPI) module of MCUXpresso SDK devices.

Modules

- [LPSPI CMSIS Driver](#)
- [LPSPI FreeRTOS Driver](#)
- [LPSPI Peripheral driver](#)
- [LPSPI eDMA Driver](#)

LPSPI Peripheral driver

LPSPI Peripheral driver

29.2.1 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.

29.2.2 Function groups

29.2.2.1 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.

29.2.2.2 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.

29.2.2.3 LPSPI Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

29.2.2.4 LPSPI Status Operation

This function group gets/clears the LPSPI status.

29.2.2.5 LPSPI Block Transfer Operation

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

29.2.3 Typical use case

29.2.3.1 Master Operation

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

29.2.3.2 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 `SPI_RETRY_TIMES` 0U /* Define to zero means keep waiting until the flag is assert/deassert. */
Retry times for waiting flag.
- #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 {
 kStatus_LPSPi_Busy = MAKE_STATUS(kStatusGroup_LPSPi, 0),
 kStatus_LPSPi_Error = MAKE_STATUS(kStatusGroup_LPSPi, 1),
 kStatus_LPSPi_Idle = MAKE_STATUS(kStatusGroup_LPSPi, 2),
 kStatus_LPSPi_OutOfRange = MAKE_STATUS(kStatusGroup_LPSPi, 3),
 kStatus_LPSPi_Timeout = MAKE_STATUS(kStatusGroup_LPSPi, 4) }
 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` {

LPSPi Peripheral driver

```
kLPSPI_SingleBitXfer = 0U,  
kLPSPI_TwoBitXfer = 1U,  
kLPSPI_FourBitXfer = 2U }
```

LPSPi transfer width configuration.

- enum `lpspi_delay_type_t` {
 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, 5))
LPSPi driver version 2.0.5.

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.*
- uint32_t [LPSPI_GetInstance](#) ([LPSPI_Type](#) *base)
 - Get the LPSPI instance from peripheral base address.*
- 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 uint8_t [LPSPI_GetTxFifoSize](#) ([LPSPI_Type](#) *base)
 - Gets the LPSPI Tx FIFO size.*
- static uint8_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)

LPSPi Peripheral driver

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)
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.

29.2.4 Data Structure Documentation

29.2.4.1 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

LPSPi Peripheral driver

- 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).

29.2.4.1.0.74 Field Documentation

29.2.4.1.0.74.1 `uint32_t lpspi_master_config_t::baudRate`

29.2.4.1.0.74.2 `uint32_t lpspi_master_config_t::bitsPerFrame`

29.2.4.1.0.74.3 `lpspi_clock_polarity_t lpspi_master_config_t::cpol`

29.2.4.1.0.74.4 `lpspi_clock_phase_t lpspi_master_config_t::cpha`

29.2.4.1.0.74.5 `lpspi_shift_direction_t lpspi_master_config_t::direction`

29.2.4.1.0.74.6 `uint32_t lpspi_master_config_t::pcsToSckDelayInNanoSec`

It sets the boundary value if out of range.

29.2.4.1.0.74.7 `uint32_t lpspi_master_config_t::lastSckToPcsDelayInNanoSec`

It sets the boundary value if out of range.

29.2.4.1.0.74.8 `uint32_t lpspi_master_config_t::betweenTransferDelayInNanoSec`

It sets the boundary value if out of range.

29.2.4.1.0.74.9 `lpspi_which_pcs_t lpspi_master_config_t::whichPcs`

29.2.4.1.0.74.10 `lpspi_pin_config_t lpspi_master_config_t::pinCfg`

29.2.4.1.0.74.11 `lpspi_data_out_config_t lpspi_master_config_t::dataOutConfig`

29.2.4.2 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).

29.2.4.2.0.75 Field Documentation

29.2.4.2.0.75.1 [uint32_t lpspi_slave_config_t::bitsPerFrame](#)

29.2.4.2.0.75.2 [lpspi_clock_polarity_t lpspi_slave_config_t::cpol](#)

29.2.4.2.0.75.3 [lpspi_clock_phase_t lpspi_slave_config_t::cpha](#)

29.2.4.2.0.75.4 [lpspi_shift_direction_t lpspi_slave_config_t::direction](#)

29.2.4.2.0.75.5 [lpspi_pin_config_t lpspi_slave_config_t::pinCfg](#)

29.2.4.2.0.75.6 [lpspi_data_out_config_t lpspi_slave_config_t::dataOutConfig](#)

29.2.4.3 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.

29.2.4.3.0.76 Field Documentation

29.2.4.3.0.76.1 [uint8_t* lpspi_transfer_t::txData](#)

29.2.4.3.0.76.2 [uint8_t* lpspi_transfer_t::rxData](#)

29.2.4.3.0.76.3 [volatile size_t lpspi_transfer_t::dataSize](#)

29.2.4.3.0.76.4 [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.

29.2.4.4 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`
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.

29.2.4.4.0.77 Field Documentation

- 29.2.4.4.0.77.1 volatile bool `lpspi_master_handle_t::isPcsContinuous`
- 29.2.4.4.0.77.2 volatile bool `lpspi_master_handle_t::writeTcrInlSr`
- 29.2.4.4.0.77.3 volatile bool `lpspi_master_handle_t::isByteSwap`
- 29.2.4.4.0.77.4 volatile uint8_t `lpspi_master_handle_t::fifoSize`
- 29.2.4.4.0.77.5 volatile uint8_t `lpspi_master_handle_t::rxWatermark`
- 29.2.4.4.0.77.6 volatile uint8_t `lpspi_master_handle_t::bytesEachWrite`
- 29.2.4.4.0.77.7 volatile uint8_t `lpspi_master_handle_t::bytesEachRead`
- 29.2.4.4.0.77.8 uint8_t* volatile `lpspi_master_handle_t::txData`
- 29.2.4.4.0.77.9 uint8_t* volatile `lpspi_master_handle_t::rxData`
- 29.2.4.4.0.77.10 volatile size_t `lpspi_master_handle_t::txRemainingByteCount`
- 29.2.4.4.0.77.11 volatile size_t `lpspi_master_handle_t::rxRemainingByteCount`
- 29.2.4.4.0.77.12 volatile uint32_t `lpspi_master_handle_t::writeRegRemainingTimes`
- 29.2.4.4.0.77.13 volatile uint32_t `lpspi_master_handle_t::readRegRemainingTimes`
- 29.2.4.4.0.77.14 uint32_t `lpspi_master_handle_t::txBuffIfNull`
- 29.2.4.4.0.77.15 volatile uint8_t `lpspi_master_handle_t::state`
- 29.2.4.4.0.77.16 `lpspi_master_transfer_callback_t` `lpspi_master_handle_t::callback`
- 29.2.4.4.0.77.17 void* `lpspi_master_handle_t::userData`

29.2.4.5 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.

29.2.4.5.0.78 Field Documentation

- 29.2.4.5.0.78.1 volatile bool lpspi_slave_handle_t::isByteSwap
- 29.2.4.5.0.78.2 volatile uint8_t lpspi_slave_handle_t::fifoSize
- 29.2.4.5.0.78.3 volatile uint8_t lpspi_slave_handle_t::rxWatermark
- 29.2.4.5.0.78.4 volatile uint8_t lpspi_slave_handle_t::bytesEachWrite
- 29.2.4.5.0.78.5 volatile uint8_t lpspi_slave_handle_t::bytesEachRead
- 29.2.4.5.0.78.6 uint8_t* volatile lpspi_slave_handle_t::txData
- 29.2.4.5.0.78.7 uint8_t* volatile lpspi_slave_handle_t::rxData
- 29.2.4.5.0.78.8 volatile size_t lpspi_slave_handle_t::txRemainingByteCount
- 29.2.4.5.0.78.9 volatile size_t lpspi_slave_handle_t::rxRemainingByteCount
- 29.2.4.5.0.78.10 volatile uint32_t lpspi_slave_handle_t::writeRegRemainingTimes
- 29.2.4.5.0.78.11 volatile uint32_t lpspi_slave_handle_t::readRegRemainingTimes
- 29.2.4.5.0.78.12 volatile uint8_t lpspi_slave_handle_t::state
- 29.2.4.5.0.78.13 volatile uint32_t lpspi_slave_handle_t::errorCount
- 29.2.4.5.0.78.14 lpspi_slave_transfer_callback_t lpspi_slave_handle_t::callback
- 29.2.4.5.0.78.15 void* lpspi_slave_handle_t::userData

29.2.5 Macro Definition Documentation

29.2.5.1 #define FSL_LPSPi_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

29.2.5.2 #define LPSPi_DUMMY_DATA (0x00U)

Dummy data used for tx if there is not txData.

LPSPI Peripheral driver

29.2.5.3 `#define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

29.2.5.4 `#define LPSPI_MASTER_PCS_SHIFT (4U)`

29.2.5.5 `#define LPSPI_MASTER_PCS_MASK (0xF0U)`

29.2.5.6 `#define LPSPI_SLAVE_PCS_SHIFT (4U)`

29.2.5.7 `#define LPSPI_SLAVE_PCS_MASK (0xF0U)`

29.2.6 Typedef Documentation

29.2.6.1 `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.

29.2.6.2 `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.

29.2.7 Enumeration Type Documentation

29.2.7.1 anonymous enum

Enumerators

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.
kStatus_LPSPI_Timeout LPSPI timeout polling status flags.

29.2.7.2 enum _lpspi_flags

Enumerators

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)

LPSPi Peripheral driver

kLPSPi_DataMatchFlag Data Match flag.

kLPSPi_ModuleBusyFlag Module Busy flag.

kLPSPi_AllStatusFlag Used for clearing all w1c status flags.

29.2.7.3 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.

29.2.7.4 enum _lpspi_dma_enable

Enumerator

kLPSPi_TxDmaEnable Transmit data DMA enable.

kLPSPi_RxDmaEnable Receive data DMA enable.

29.2.7.5 enum lpspi_master_slave_mode_t

Enumerator

kLPSPi_Master LPSPi peripheral operates in master mode.

kLPSPi_Slave LPSPi peripheral operates in slave mode.

29.2.7.6 enum lpspi_which_pcs_t

Enumerator

kLPSPi_Pcs0 PCS[0].

kLPSPi_Pcs1 PCS[1].

kLPSPi_Pcs2 PCS[2].

kLPSPi_Pcs3 PCS[3].

29.2.7.7 enum `lpspi_pcs_polarity_config_t`

Enumerator

kLPSPi_PcsActiveHigh PCS Active High (idles low)
kLPSPi_PcsActiveLow PCS Active Low (idles high)

29.2.7.8 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).

29.2.7.9 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)

29.2.7.10 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.

29.2.7.11 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

29.2.7.12 enum lpspi_host_request_select_t

Enumerator

kLPSPI_HostReqExtPin Host Request is an ext pin.

kLPSPI_HostReqInternalTrigger Host Request is an internal trigger.

29.2.7.13 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.

29.2.7.14 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.

29.2.7.15 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.

29.2.7.16 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]

29.2.7.17 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.

29.2.7.18 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.

29.2.7.19 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.

29.2.7.20 enum _lpspi_transfer_state

Enumerator

kLPSPi_Idle Nothing in the transmitter/receiver.
kLPSPi_Busy Transfer queue is not finished.
kLPSPi_Error Transfer error.

29.2.8 Function Documentation

29.2.8.1 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 lpspi_master_config_t .
<i>srcClock_Hz</i>	Module source input clock in Hertz

29.2.8.2 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 lpspi_master_config_t structure
---------------------	--

29.2.8.3 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 lpspi_slave_config_t .

29.2.8.4 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 lpspi_slave_config_t structure.
--------------------	--

29.2.8.5 void LPSPI_Deinit (LPSPI_Type * *base*)

Call this API to disable the LPSPI clock.

Parameters

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

29.2.8.6 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.
-------------	---------------------------

29.2.8.7 uint32_t LPSPI_GetInstance (LPSPI_Type * *base*)

LPSPI Peripheral driver

Parameters

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

Returns

LPSPI instance.

29.2.8.8 `static void LPSPI_Enable (LPSPI_Type * base, bool enable) [inline],
[static]`

Parameters

<i>base</i>	LPSPI peripheral address.
<i>enable</i>	Pass true to enable module, false to disable module.

29.2.8.9 `static uint32_t LPSPI_GetStatusFlags (LPSPI_Type * base) [inline],
[static]`

Parameters

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

Returns

The LPSPI status(in SR register).

29.2.8.10 `static uint8_t LPSPI_GetTxFifoSize (LPSPI_Type * base) [inline],
[static]`

Parameters

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

Returns

The LPSPI Tx FIFO size.

29.2.8.11 `static uint8_t LPSPI_GetRxFifoSize (LPSPI_Type * base) [inline],
[static]`

Parameters

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

Returns

The LPSPI Rx FIFO size.

29.2.8.12 `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.

29.2.8.13 `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.

29.2.8.14 `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);
*
```

LPSPi Peripheral driver

Parameters

<i>base</i>	LPSPi peripheral address.
<i>statusFlags</i>	The status flag used from type <code>_lpspi_flags</code> .

< The status flags are cleared by writing 1 (w1c).

29.2.8.15 `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 );  
*
```

Parameters

<i>base</i>	LPSPi peripheral address.
<i>mask</i>	The interrupt mask; Use the enum <code>_lpspi_interrupt_enable</code> .

29.2.8.16 `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 <code>_lpspi_interrupt_enable</code> .

29.2.8.17 `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 <code>_lpspi_dma_enable</code> .

29.2.8.18 **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 <code>_lpspi_dma_enable</code> .

29.2.8.19 **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.

29.2.8.20 **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.

LPSPI Peripheral driver

Parameters

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

Returns

The LPSPI Receive Data Register address.

29.2.8.21 **bool LPSPI_CheckTransferArgument (lpspi_transfer_t * *transfer*, uint32_t *bitsPerFrame*, uint32_t *bytesPerFrame*)**

Parameters

<i>transfer</i>	the transfer struct to be used.
<i>bitsPerFrame</i>	The bit size of one frame.
<i>bytesPerFrame</i>	The byte size of one frame.

Returns

Return true for right and false for wrong.

29.2.8.22 **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.

29.2.8.23 **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.

29.2.8.24 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.

29.2.8.25 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.

29.2.8.26 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);
*
```

LPSPI Peripheral driver

Parameters

<i>base</i>	LPSPI peripheral address.
<i>mask</i>	The PCS polarity mask; Use the enum <code>_lpspi_pcs_polarity</code> .

29.2.8.27 `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 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.

29.2.8.28 `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.

29.2.8.29 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 lpspi_delay_type_t.

29.2.8.30 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

LPSPi Peripheral driver

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 $\text{delayTime} = \text{LPSPi_clockSource} / (\text{PRESCALE} * \text{Delay_scaler})$.

Parameters

<i>base</i>	LPSPi peripheral address.
<i>delayTimeInNanoSec</i>	The desired delay value in nano-seconds.
<i>whichDelay</i>	The desired delay to configuration, which must be of type <code>lpspi_delay_type_t</code> .
<i>srcClock_Hz</i>	Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

29.2.8.31 `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.

29.2.8.32 `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.

29.2.8.33 void LPSPI_SetDummyData (LPSPI_Type * *base*, uint8_t *dummyData*)

LPSPI Peripheral driver

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.

**29.2.8.34 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 lpspi_master_handle_t.
<i>callback</i>	DSPI callback.
<i>userData</i>	callback function parameter.

**29.2.8.35 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 lpspi_transfer_t structure.

Returns

status of status_t.

29.2.8.36 **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 lpspi_transfer_t structure.

Returns

status of status_t.

29.2.8.37 **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.

29.2.8.38 **void LPSPI_MasterTransferAbort (LPSPI_Type * *base*, lpspi_master_handle_t * *handle*)**

This function aborts a transfer which uses an interrupt method.

LPSPI Peripheral driver

Parameters

<i>base</i>	LPSPI peripheral address.
<i>handle</i>	pointer to <code>lpspi_master_handle_t</code> structure which stores the transfer state.

29.2.8.39 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 <code>lpspi_master_handle_t</code> structure which stores the transfer state.

29.2.8.40 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 <code>lpspi_slave_handle_t</code> .
<i>callback</i>	DSPI callback.
<i>userData</i>	callback function parameter.

29.2.8.41 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.

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`.

29.2.8.42 `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`.

29.2.8.43 `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.

29.2.8.44 `void LPSPi_SlaveTransferHandleIRQ (LPSPi_Type * base, lpspi_slave_handle_t * handle)`

This function processes the LPSPi transmit and receives an IRQ.

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.

29.2.9 Variable Documentation

29.2.9.1 `volatile uint8_t g_lpspiDummyData[]`

LPSPI eDMA Driver

29.3.1 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)

LPSPi eDMA Driver

Gets the slave eDMA transfer remaining bytes.

Driver version

- #define [FSL_LPSPi_EDMA_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 0, 5))
LPSPi EDMA driver version 2.0.5.

29.3.2 Data Structure Documentation

29.3.2.1 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 bool [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.

- `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.

LPSPI eDMA Driver

29.3.2.1.0.79 Field Documentation

- 29.3.2.1.0.79.1 volatile bool `lpspi_master_edma_handle_t::isPcsContinuous`
 - 29.3.2.1.0.79.2 volatile bool `lpspi_master_edma_handle_t::isByteSwap`
 - 29.3.2.1.0.79.3 volatile uint8_t `lpspi_master_edma_handle_t::fifoSize`
 - 29.3.2.1.0.79.4 volatile uint8_t `lpspi_master_edma_handle_t::rxWatermark`
 - 29.3.2.1.0.79.5 volatile uint8_t `lpspi_master_edma_handle_t::bytesEachWrite`
 - 29.3.2.1.0.79.6 volatile uint8_t `lpspi_master_edma_handle_t::bytesEachRead`
 - 29.3.2.1.0.79.7 volatile uint8_t `lpspi_master_edma_handle_t::bytesLastRead`
 - 29.3.2.1.0.79.8 volatile bool `lpspi_master_edma_handle_t::isThereExtraRxBytes`
 - 29.3.2.1.0.79.9 uint8_t* volatile `lpspi_master_edma_handle_t::txData`
 - 29.3.2.1.0.79.10 uint8_t* volatile `lpspi_master_edma_handle_t::rxData`
 - 29.3.2.1.0.79.11 volatile size_t `lpspi_master_edma_handle_t::txRemainingByteCount`
 - 29.3.2.1.0.79.12 volatile size_t `lpspi_master_edma_handle_t::rxRemainingByteCount`
 - 29.3.2.1.0.79.13 volatile uint32_t `lpspi_master_edma_handle_t::writeRegRemainingTimes`
 - 29.3.2.1.0.79.14 volatile uint32_t `lpspi_master_edma_handle_t::readRegRemainingTimes`
 - 29.3.2.1.0.79.15 uint32_t `lpspi_master_edma_handle_t::txBuffIfNull`
 - 29.3.2.1.0.79.16 uint32_t `lpspi_master_edma_handle_t::rxBuffIfNull`
 - 29.3.2.1.0.79.17 uint32_t `lpspi_master_edma_handle_t::transmitCommand`
 - 29.3.2.1.0.79.18 volatile uint8_t `lpspi_master_edma_handle_t::state`
 - 29.3.2.1.0.79.19 uint8_t `lpspi_master_edma_handle_t::nbytes`
 - 29.3.2.1.0.79.20 `lpspi_master_edma_transfer_callback_t` `lpspi_master_edma_handle_t::callback`
 - 29.3.2.1.0.79.21 void* `lpspi_master_edma_handle_t::userData`
- ### 29.3.2.2 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 bool [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.

29.3.2.2.0.80 Field Documentation

- 29.3.2.2.0.80.1 volatile bool lpspi_slave_edma_handle_t::isByteSwap
- 29.3.2.2.0.80.2 volatile uint8_t lpspi_slave_edma_handle_t::fifoSize
- 29.3.2.2.0.80.3 volatile uint8_t lpspi_slave_edma_handle_t::rxWatermark
- 29.3.2.2.0.80.4 volatile uint8_t lpspi_slave_edma_handle_t::bytesEachWrite
- 29.3.2.2.0.80.5 volatile uint8_t lpspi_slave_edma_handle_t::bytesEachRead
- 29.3.2.2.0.80.6 volatile uint8_t lpspi_slave_edma_handle_t::bytesLastRead
- 29.3.2.2.0.80.7 volatile bool lpspi_slave_edma_handle_t::isThereExtraRxBytes
- 29.3.2.2.0.80.8 uint8_t lpspi_slave_edma_handle_t::nbytes
- 29.3.2.2.0.80.9 uint8_t* volatile lpspi_slave_edma_handle_t::txData
- 29.3.2.2.0.80.10 uint8_t* volatile lpspi_slave_edma_handle_t::rxData
- 29.3.2.2.0.80.11 volatile size_t lpspi_slave_edma_handle_t::txRemainingByteCount
- 29.3.2.2.0.80.12 volatile size_t lpspi_slave_edma_handle_t::rxRemainingByteCount
- 29.3.2.2.0.80.13 volatile uint32_t lpspi_slave_edma_handle_t::writeRegRemainingTimes
- 29.3.2.2.0.80.14 volatile uint32_t lpspi_slave_edma_handle_t::readRegRemainingTimes
- 29.3.2.2.0.80.15 uint32_t lpspi_slave_edma_handle_t::txBuffIfNull
- 29.3.2.2.0.80.16 uint32_t lpspi_slave_edma_handle_t::rxBuffIfNull
- 29.3.2.2.0.80.17 volatile uint8_t lpspi_slave_edma_handle_t::state
- 29.3.2.2.0.80.18 uint32_t lpspi_slave_edma_handle_t::errorCount
- 29.3.2.2.0.80.19 lpspi_slave_edma_transfer_callback_t lpspi_slave_edma_handle_t::callback
- 29.3.2.2.0.80.20 void* lpspi_slave_edma_handle_t::userData

29.3.3 Macro Definition Documentation

- 29.3.3.1 #define FSL_LPSPi_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

29.3.4 Typedef Documentation

- 29.3.4.1 typedef void(* lpspi_master_edma_transfer_callback_t)(LPSPi_Type *base, lpspi_master_edma_handle_t *handle, status_t status, void *userData)

LPSPI eDMA Driver

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.

29.3.4.2 `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.

29.3.5 Function Documentation

29.3.5.1 `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 *edmaIntermediaryToTxRegHandle*. (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 <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 edma_handle_t .
<i>edmaTxData-ToTxReg-Handle</i>	edmaTxDataToTxRegHandle pointer to edma_handle_t .

29.3.5.2 `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 lpspi_transfer_t structure.

Returns

status of `status_t`.

29.3.5.3 `void LPSPI_MasterTransferAbortEDMA (LPSPI_Type * base, lpspi_master_edma_handle_t * handle)`

This function aborts a transfer which is using eDMA.

Parameters

LPSPI eDMA Driver

<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.

29.3.5.4 `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 <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the EDMA transaction.

Returns

status of `status_t`.

29.3.5.5 `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 <code>lpspi_slave_edma_handle_t</code> .

<i>callback</i>	LPSPI callback.
<i>userData</i>	callback function parameter.
<i>edmaRxRegTo-RxDataHandle</i>	edmaRxRegToRxDataHandle pointer to edma_handle_t .
<i>edmaTxData-ToTxReg-Handle</i>	edmaTxDataToTxRegHandle pointer to edma_handle_t .

29.3.5.6 **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 lpspi_slave_edma_handle_t structure which stores the transfer state.
<i>transfer</i>	pointer to lpspi_transfer_t structure.

Returns

status of status_t.

29.3.5.7 **void LPSPI_SlaveTransferAbortEDMA (LPSPI_Type * *base*, lpspi_slave_edma_handle_t * *handle*)**

This function aborts a transfer which is using eDMA.

Parameters

LPSPI eDMA Driver

<i>base</i>	LPSPI peripheral base address.
<i>handle</i>	pointer to <code>lpspi_slave_edma_handle_t</code> structure which stores the transfer state.

29.3.5.8 `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 <code>lpspi_slave_edma_handle_t</code> structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the eDMA transaction.

Returns

status of `status_t`.

LPSPI FreeRTOS Driver

29.4.1 Overview

Driver version

- #define **FSL_LPSPI_FREERTOS_DRIVER_VERSION** (**MAKE_VERSION**(2, 0, 5))
LPSPI FreeRTOS driver version 2.0.5.

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.

29.4.2 Macro Definition Documentation

29.4.2.1 #define FSL_LPSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

29.4.3 Function Documentation

29.4.3.1 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.

29.4.3.2 `status_t LPSPI_RTOS_Deinit (lpspi_rtos_handle_t * handle)`

This function deinitializes the LPSPI module and related RTOS context.

Parameters

<i>handle</i>	The RTOS LPSPI handle.
---------------	------------------------

29.4.3.3 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.

LPSPi CMSIS Driver

This section describes the programming interface of the LPSPi Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

29.5.1 Function groups

29.5.1.1 LPSPi CMSIS GetVersion Operation

This function group will return the DSPi CMSIS Driver version to user.

29.5.1.2 LPSPi CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

29.5.1.3 LPSPi CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the instance in master mode or slave mode. And this API must be called before you configure an instance or after you Deinit an instance. The right steps to start an instance is that you must initialize the instance which been selected firstly, then you can power on the instance. After these all have been done, you can configure the instance by using control operation. If you want to Uninitialize the instance, you must power off the instance first.

29.5.1.4 LPSPi Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

29.5.1.5 LPSPi Status Operation

This function group gets the LPSPi transfer status.

29.5.1.6 LPSPi CMSIS Control Operation

This function can select instance as master mode or slave mode, set baudrate for master mode transfer, get current baudrate of master mode transfer, set transfer data bits and set other control command.

29.5.2 Typical use case

29.5.2.1 Master Operation

```

/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*DSPI master init*/
Driver_SPI0.Initialize(DSPI_MasterSignalEvent_t);
Driver_SPI0.PowerControl(ARM_POWER_FULL);
Driver_SPI0.Control(ARM_SPI_MODE_MASTER, TRANSFER_BAUDRATE);

/* Start master transfer */
Driver_SPI0.Transfer(masterTxData, masterRxData, TRANSFER_SIZE);

/* Master power off */
Driver_SPI0.PowerControl(ARM_POWER_OFF);

/* Master uninitialized */
Driver_SPI0.Uninitialize();

```

29.5.2.2 Slave Operation

```

/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*DSPI slave init*/
Driver_SPI2.Initialize(DSPI_SlaveSignalEvent_t);
Driver_SPI2.PowerControl(ARM_POWER_FULL);
Driver_SPI2.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
Driver_SPI2.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
Driver_SPI2.PowerControl(ARM_POWER_OFF);

/* slave uninitialized */
Driver_SPI2.Uninitialize();

```




Chapter 30

LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver

Overview

Modules

- [LPUART CMSIS Driver](#)
- [LPUART Driver](#)
- [LPUART FreeRTOS Driver](#)
- [LPUART eDMA Driver](#)

LPUART Driver

LPUART Driver

30.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

30.2.2 Typical use case

30.2.2.1 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...](#)

Macros

- #define [UART_RETRY_TIMES](#) 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */
Retry times for waiting flag.

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 {
 - 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),
 - kStatus_LPUART_Timeout = MAKE_STATUS(kStatusGroup_LPUART, 15) }

Error codes for the LPUART driver.
- enum lpuart_parity_mode_t {
 - kLPUART_ParityDisabled = 0x0U,
 - kLPUART_ParityEven = 0x2U,
 - kLPUART_ParityOdd = 0x3U }

LPUART parity mode.
- enum lpuart_data_bits_t {
 - kLPUART_EightDataBits = 0x0U,
 - kLPUART_SevenDataBits = 0x1U }

LPUART data bits count.
- enum lpuart_stop_bit_count_t {
 - kLPUART_OneStopBit = 0U,
 - kLPUART_TwoStopBit = 1U }

LPUART stop bit count.
- enum lpuart_transmit_cts_source_t {
 - kLPUART_CtsSourcePin = 0U,
 - kLPUART_CtsSourceMatchResult = 1U }

LPUART transmit CTS source.
- enum lpuart_transmit_cts_config_t {
 - kLPUART_CtsSampleAtStart = 0U,
 - kLPUART_CtsSampleAtIdle = 1U }

LPUART transmit CTS configure.
- enum lpuart_idle_type_select_t {
 - kLPUART_IdleTypeStartBit = 0U,
 - kLPUART_IdleTypeStopBit = 1U }

LPUART idle flag type defines when the receiver starts counting.
- enum lpuart_idle_config_t {

```
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_LBKDIE_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, 3, 0))
LPUART driver version 2.3.0.

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.
- `status_t 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 sent out to bus.
- `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.

30.2.3 Data Structure Documentation

30.2.3.1 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

30.2.3.1.0.81 Field Documentation

30.2.3.1.0.81.1 lpuart_idle_type_select_t lpuart_config_t::rxIdleType

30.2.3.1.0.81.2 lpuart_idle_config_t lpuart_config_t::rxIdleConfig

30.2.3.2 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.

30.2.3.2.0.82 Field Documentation

30.2.3.2.0.82.1 uint8_t* lpuart_transfer_t::data

30.2.3.2.0.82.2 size_t lpuart_transfer_t::dataSize

30.2.3.3 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.

30.2.3.3.0.83 Field Documentation

- 30.2.3.3.0.83.1 `uint8_t* volatile lpuart_handle_t::txData`
- 30.2.3.3.0.83.2 `volatile size_t lpuart_handle_t::txDataSize`
- 30.2.3.3.0.83.3 `size_t lpuart_handle_t::txDataSizeAll`
- 30.2.3.3.0.83.4 `uint8_t* volatile lpuart_handle_t::rxData`
- 30.2.3.3.0.83.5 `volatile size_t lpuart_handle_t::rxDataSize`
- 30.2.3.3.0.83.6 `size_t lpuart_handle_t::rxDataSizeAll`
- 30.2.3.3.0.83.7 `uint8_t* lpuart_handle_t::rxRingBuffer`
- 30.2.3.3.0.83.8 `size_t lpuart_handle_t::rxRingBufferSize`
- 30.2.3.3.0.83.9 `volatile uint16_t lpuart_handle_t::rxRingBufferHead`
- 30.2.3.3.0.83.10 `volatile uint16_t lpuart_handle_t::rxRingBufferTail`
- 30.2.3.3.0.83.11 `lpuart_transfer_callback_t lpuart_handle_t::callback`
- 30.2.3.3.0.83.12 `void* lpuart_handle_t::userData`
- 30.2.3.3.0.83.13 `volatile uint8_t lpuart_handle_t::txState`
- 30.2.3.3.0.83.14 `volatile uint8_t lpuart_handle_t::rxState`
- 30.2.3.3.0.83.15 `bool lpuart_handle_t::isSevenDataBits`

30.2.4 Macro Definition Documentation

- 30.2.4.1 `#define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`
- 30.2.4.2 `#define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */`

30.2.5 Typedef Documentation

- 30.2.5.1 `typedef void(* lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle, status_t status, void *userData)`

30.2.6 Enumeration Type Documentation

30.2.6.1 anonymous enum

Enumerator

LPUART Driver

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_RxRingBufferOverflow LPUART RX software ring buffer overrun.
kStatus_LPUART_RxHardwareOverflow 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.
kStatus_LPUART_Timeout LPUART times out.

30.2.6.2 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.

30.2.6.3 enum lpuart_data_bits_t

Enumerator

kLPUART_EightDataBits Eight data bit.
kLPUART_SevenDataBits Seven data bit.

30.2.6.4 enum lpuart_stop_bit_count_t

Enumerator

kLPUART_OneStopBit One stop bit.
kLPUART_TwoStopBit Two stop bits.

30.2.6.5 enum lpuart_transmit_cts_source_t

Enumerator

kLPUART_CtsSourcePin CTS resource is the LPUART_CTS pin.

kLPUART_CtsSourceMatchResult CTS resource is the match result.

30.2.6.6 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.

30.2.6.7 enum lpuart_idle_type_select_t

Enumerator

kLPUART_IdleTypeStartBit Start counting after a valid start bit.

kLPUART_IdleTypeStopBit Start counting after a stop bit.

30.2.6.8 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.

30.2.6.9 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.

30.2.6.10 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.

30.2.7 Function Documentation

30.2.7.1 `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.

LPUART Driver

Parameters

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

30.2.7.2 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_GetDefault-Config\(\)](#) 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

30.2.7.3 void LPUART_Deinit (LPUART_Type * *base*)

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

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

30.2.7.4 void LPUART_GetDefaultConfig (lpuart_config_t * *config*)

This function initializes the LPUART configuration structure to a default value. The default values are:
 : 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; lpuartConfig->rxIdleType = kLPUART_IdleTypeStartBit; lpuartConfig->rxIdleConfig = kLPUART_IdleCharacter1; lpuartConfig->enableTx = false; lpuartConfig->enableRx = false;

Parameters

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

30.2.7.5 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, 200000000U);
*
```

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.

30.2.7.6 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:

LPUART Driver

```
*      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`.

30.2.7.7 `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 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 <code>_lpuart_status_flag_t</code> to do the OR operation and get the mask.

Returns

0 succeed, others failed.

Return values

<i>kStatus_LPUART_Flag- CannotClearManually</i>	The flag can't be cleared by this function but it is cleared automatically by hardware.
---	---

<i>kStatus_Success</i>	Status in the mask are cleared.
------------------------	---------------------------------

30.2.7.8 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 the enumeration _uart_interrupt_enable .

30.2.7.9 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 _lpuart_interrupt_enable .

30.2.7.10 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:

LPUART Driver

```

*      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](#).

30.2.7.11 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.

30.2.7.12 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.

30.2.7.13 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.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>enable</i>	True to enable, false to disable.

30.2.7.14 uint32_t LPUART_GetInstance (LPUART_Type * *base*)

Parameters

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

Returns

LPUART instance.

**30.2.7.15 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.

**30.2.7.16 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.

**30.2.7.17 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.

LPUART Driver

Parameters

<i>base</i>	LPUART peripheral base address.
<i>data</i>	Data write to the TX register.

30.2.7.18 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.

30.2.7.19 status_t LPUART_WriteBlocking (LPUART_Type * *base*, const uint8_t * *data*, size_t *length*)

This function polls the transmitter register, first waits for the register to be empty or TX FIFO to have room, and writes data to the transmitter buffer, then waits for the data to be sent out to the bus.

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.

Return values

<i>kStatus_LPUART_Timeout</i>	Transmission timed out and was aborted.
-------------------------------	---

<i>kStatus_Success</i>	Successfully wrote all data.
------------------------	------------------------------

30.2.7.20 **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.

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_LPUART_-Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully received all data.

30.2.7.21 **void LPUART_TransferCreateHandle (LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_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`.

LPUART Driver

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>callback</i>	Callback function.
<i>userData</i>	User data.

30.2.7.22 **status_t LPUART_TransferSendNonBlocking (LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_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 lpuart_transfer_t .

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.

30.2.7.23 **void LPUART_TransferStartRingBuffer (LPUART_Type * *base*, lpuart_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.

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.

30.2.7.24 void LPUART_TransferStopRingBuffer (LPUART_Type * *base*, lpuart_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.

30.2.7.25 size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type * *base*, lpuart_handle_t * *handle*)

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

Returns

Length of received data in RX ring buffer.

30.2.7.26 void LPUART_TransferAbortSend (LPUART_Type * *base*, lpuart_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.

LPUART Driver

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.

30.2.7.27 **status_t LPUART_TransferGetSendCount (LPUART_Type * *base*, lpuart_handle_t * *handle*, uint32_t * *count*)**

This function gets the number of bytes that have been sent out to bus 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> ;

30.2.7.28 **status_t LPUART_TransferReceiveNonBlocking (LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_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.

Parameters

<i>base</i>	LPUART peripheral base address.
<i>handle</i>	LPUART handle pointer.
<i>xfer</i>	LPUART transfer structure, see <code>uart_transfer_t</code> .
<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.

30.2.7.29 void LPUART_TransferAbortReceive (LPUART_Type * *base*, lpuart_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.

30.2.7.30 status_t LPUART_TransferGetReceiveCount (LPUART_Type * *base*, lpuart_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.

LPUART Driver

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;

30.2.7.31 void LPUART_TransferHandleIRQ (LPUART_Type * *base*, lpuart_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.

30.2.7.32 void LPUART_TransferHandleErrorIRQ (LPUART_Type * *base*, lpuart_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.

LPUART eDMA Driver

30.3.1 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, 3, 0))
LPUART EDMA driver version 2.3.0.

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.

30.3.2 Data Structure Documentation

30.3.2.1 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.

30.3.2.1.0.84 Field Documentation**30.3.2.1.0.84.1** `lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback`**30.3.2.1.0.84.2** `void* lpuart_edma_handle_t::userData`**30.3.2.1.0.84.3** `size_t lpuart_edma_handle_t::rxDataSizeAll`**30.3.2.1.0.84.4** `size_t lpuart_edma_handle_t::txDataSizeAll`**30.3.2.1.0.84.5** `edma_handle_t* lpuart_edma_handle_t::txEdmaHandle`**30.3.2.1.0.84.6** `edma_handle_t* lpuart_edma_handle_t::rxEdmaHandle`**30.3.2.1.0.84.7** `uint8_t lpuart_edma_handle_t::nbytes`**30.3.2.1.0.84.8** `volatile uint8_t lpuart_edma_handle_t::txState`**30.3.3 Macro Definition Documentation****30.3.3.1** `#define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`**30.3.4 Typedef Documentation****30.3.4.1** `typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base,
lpuart_edma_handle_t *handle, status_t status, void *userData)`**30.3.5 Function Documentation****30.3.5.1** `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 <code>lpuart_edma_handle_t</code> 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.

30.3.5.2 `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 lpuart_transfer_t .

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.

30.3.5.3 `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 <code>lpuart_edma_handle_t</code> structure.
<i>xfer</i>	LPUART eDMA transfer structure, see lpuart_transfer_t .

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.

30.3.5.4 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 <code>lpuart_edma_handle_t</code> structure.

30.3.5.5 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 <code>lpuart_edma_handle_t</code> structure.

30.3.5.6 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_NoTransferInProgress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <code>count</code> ;

30.3.5.7 `status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type * base, lpuart_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_NoTransferInProgress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <code>count</code> ;

LPUART FreeRTOS Driver

30.4.1 Overview

Data Structures

- struct [lpuart_rtos_config_t](#)
LPUART RTOS configuration structure. [More...](#)

Driver version

- #define [FSL_LPUART_FREERTOS_DRIVER_VERSION](#) (MAKE_VERSION(2, 3, 0))
LPUART FreeRTOS driver version 2.3.0.

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.

30.4.2 Data Structure Documentation

30.4.2.1 struct lpuart_rtos_config_t

Data Fields

- LPUART_Type * [base](#)
UART base address.
- uint32_t [srcclk](#)
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.
- `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.

30.4.3 Macro Definition Documentation

30.4.3.1 `#define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))`

30.4.4 Function Documentation

30.4.4.1 `int LPUART_RTOS_Init (lpuart_rtos_handle_t * handle, lpuart_handle_t * t_handle, const lpuart_rtos_config_t * cfg)`

Parameters

<i>handle</i>	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
<i>t_handle</i>	The pointer to an allocated space to store the transactional layer internal state.
<i>cfg</i>	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

30.4.4.2 `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

<i>handle</i>	The RTOS LPUART handle.
---------------	-------------------------

30.4.4.3 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.

30.4.4.4 int LPUART_RTOS_Receive (lpuart_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.

LPUART CMSIS Driver

This section describes the programming interface of the LPUART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to <http://www.keil.com/pack/doc/cmsis/Driver/html/index.html>.

The LPUART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

30.5.1 Function groups

30.5.1.1 LPUART CMSIS GetVersion Operation

This function group will return the LPUART CMSIS Driver version to user.

30.5.1.2 LPUART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

30.5.1.3 LPUART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize the lpuart instance . And this API must be called before you configure a lpuart instance or after you Deinit a lpuart instance. The right steps to start an instance is that you must initialize the instance which been selected firstly, then you can power on the instance. After these all have been done, you can configure the instance by using control operation. If you want to Uninitialize the instance, you must power off the instance first.

30.5.1.4 LPUART CMSIS Transfer Operation

This function group controls the transfer, send/receive data.

30.5.1.5 LPUART CMSIS Status Operation

This function group gets the LPUART transfer status.

30.5.1.6 LPUART CMSIS Control Operation

This function can configure an instance ,set baudrate for lpuart, get current baudrate ,set transfer data bits and other control command.

Chapter 31

MIPI CSI2 RX: MIPI CSI2 RX Driver

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, 3))
CSI2RX driver version.

Data Structure Documentation

31.2.1 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

31.2.1.0.0.85 Field Documentation

31.2.1.0.0.85.1 uint8_t csi2rx_config_t::laneNum

31.2.1.0.0.85.2 uint8_t csi2rx_config_t::tHsSettle_EscClk

The T_HS_SETTLE should be in the range of 85ns + 6UI to 145ns + 10UI.

Macro Definition Documentation

31.3.1 #define FSL_CSI2RX_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Enumeration Type Documentation

31.4.1 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.

31.4.2 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.

31.4.3 enum _csi2rx_bit_error

Enumerator

<i>kCSI2RX_BitErrorEccTwoBit</i>	ECC two bit error has occurred.
<i>kCSI2RX_BitErrorEccOneBit</i>	ECC one bit error has occurred.

31.4.4 enum csi2rx_ppi_error_t

Enumerator

<i>kCSI2RX_PpiErrorSotHs</i>	CSI2RX DPHY PPI error ErrSotHS.
<i>kCSI2RX_PpiErrorSotSyncHs</i>	CSI2RX DPHY PPI error ErrSotSync_HS.
<i>kCSI2RX_PpiErrorEsc</i>	CSI2RX DPHY PPI error ErrEsc.
<i>kCSI2RX_PpiErrorSyncEsc</i>	CSI2RX DPHY PPI error ErrSyncEsc.
<i>kCSI2RX_PpiErrorControl</i>	CSI2RX DPHY PPI error ErrControl.

31.4.5 enum _csi2rx_interrupt

31.4.6 enum _csi2rx_ulps_status

Enumerator

<i>kCSI2RX_ClockLaneUlps</i>	Clock lane is in ULPS state.
<i>kCSI2RX_DataLane0Ulps</i>	Data lane 0 is in ULPS state.
<i>kCSI2RX_DataLane1Ulps</i>	Data lane 1 is in ULPS state.
<i>kCSI2RX_DataLane2Ulps</i>	Data lane 2 is in ULPS state.
<i>kCSI2RX_DataLane3Ulps</i>	Data lane 3 is in ULPS state.
<i>kCSI2RX_ClockLaneMark</i>	Clock lane is in mark state.
<i>kCSI2RX_DataLane0Mark</i>	Data lane 0 is in mark state.
<i>kCSI2RX_DataLane1Mark</i>	Data lane 1 is in mark state.
<i>kCSI2RX_DataLane2Mark</i>	Data lane 2 is in mark state.
<i>kCSI2RX_DataLane3Mark</i>	Data lane 3 is in mark state.

Function Documentation

31.5.1 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.

31.5.2 void CSI2RX_Deinit (MIPI_CSI2RX_Type * *base*)

Parameters

<i>base</i>	CSI2RX peripheral address.
-------------	----------------------------

**31.5.3 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;

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

31.5.4 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 CSI2RX_GetBitError .
-----------------	--

Returns

The position of error bit.

31.5.5 static uint32_t CSI2RX_GetUlpStatus (MIPI_CSI2RX_Type * *base*) [inline], [static]

Example to check whether data lane 0 is in ULPS status.

```
uint32_t status = CSI2RX_GetUlpStatus(MIPI_CSI2RX);
if (kCSI2RX_DataLane0Ulp & 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](#).

31.5.6 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.

31.5.7 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 <code>_csi2rx_interrupt</code> .

31.5.8 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 _csi2rx_interrupt .
-------------	--

31.5.9 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](#).



Chapter 32

MIPI_DSI: MIPI DSI Host Controller

Overview

The MCUXpresso SDK provides a peripheral driver for the MIPI DSI

Modules

- [MIPI DSI Driver](#)

The MIPI DSI driver supports both video mode and command mode.

MIPI DSI Driver

The MIPI DSI driver supports both video mode and command mode.

32.2.1 Overview

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](#).

32.2.2 Command mode data transfer

DSI driver provides polling method and interrupt method for the command mode data transfer, they are [DSI_TransferBlocking](#) and [DSI_TransferNonBlocking](#). The transfer is specified by the structure [dsi_transfer_t](#).

There are two ways to construct the [dsi_transfer_t](#).

1. Include the DSC command in TX data array. In this case, the DSC command is the first byte of TX data array. The parameter `sendDscCmd` is set to false, the `dscCmd` is not used.
2. The DSC command is not in TX data array, but specified by parameter `dscCmd`. In this case, the parameter `sendDscCmd` is set to true, the `dscCmd` is the DSC command to send. The TX data array is sent after `dscCmd`.

There is an example that send DSC command `set_column_address (0x2A)`. The two methods are actually the same.

Method 1: Include DSC command in TX data array.

```
dsi_transfer_t dsiXfer = {0};
uint8_t txData[4];

dsiXfer.virtualChannel = 0;
dsiXfer.txDataType     = kDSI_TxDataDcsLongWr;
dsiXfer.txDataSize     = 4;
dsiXfer.txData         = txData;
dsiXfer.sendDscCmd     = false;
dsiXfer.dscCmd         = 0; /* Not used. */

txData[0] = 0x2A;
txData[1] = (startX >> 8U) & 0xFFU;
txData[2] = startX & 0xFFU;
```



```
txData[3] = (endX >> 8U) & 0xFFU;
txData[4] = endX & 0xFFU;

DSI_TransferBlocking(MIPI_DSI, &dsiXfer);
```

Method 2: Don't include DSC command in TX data array.

```
dsi_transfer_t dsiXfer = {0};
uint8_t txData[5];

dsiXfer.virtualChannel = 0;
dsiXfer.txDataType     = kDSI_TxDataDcsLongWr;
dsiXfer.txDataSize     = 5;
dsiXfer.txData         = txData;
dsiXfer.sendDscCmd     = true;
dsiXfer.dscCmd         = 0x2A;

txData[0] = (startX >> 8U) & 0xFFU;
txData[1] = startX & 0xFFU;
txData[2] = (endX >> 8U) & 0xFFU;
txData[3] = endX & 0xFFU;

DSI_TransferBlocking(MIPI_DSI, &dsiXfer);
```

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 {
[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) }

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 {
 `kDSI_DpiVsyncActiveLow` = 0U,
 `kDSI_DpiHsyncActiveLow` = 0U,
 `kDSI_DpiVsyncActiveHigh` = (1U << 0U),
 `kDSI_DpiHsyncActiveHigh` = (1U << 1U) }

_dsi_dpi_polarity_flag 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 {
 `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) }

_dsi_apb_status Status of APB to packet interface.

- enum {

```

kDSI_RxErrorEccOneBit = (1U << 0U),
kDSI_RxErrorEccMultiBit = (1U << 1U),
kDSI_RxErrorCrc = (1U << 7U),
kDSI_RxErrorHtxTo = (1U << 8U),
kDSI_RxErrorLrxTo = (1U << 9U),
kDSI_RxErrorBtaTo = (1U << 10U) }

```

_dsi_rx_error_status 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 {

```

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 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 {

MIPI DSI Driver

```
kDSI_TransferUseHighSpeed = (1U << 0U),  
kDSI_TransferPerformBTA = (1U << 1U) }  
_dsi_transfer_flags DSI transfer control flags.
```

Driver version

- #define **FSL_MIPI_DSI_DRIVER_VERSION** ([MAKE_VERSION](#)(2, 0, 4))

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 [dsi_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 [dsi_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](#) ([dsi_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_WriteApbTxPayloadExt](#) (MIPI_DSI_HOST_Type *base, const uint8_t *payload, uint16_t payloadSize, bool sendDscCmd, uint8_t dscCmd)
Extended function to fill the payload to TX FIFO.
- 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.

MIPI DSI Driver

- 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.

32.2.3 Data Structure Documentation

32.2.3.1 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.

32.2.3.1.0.86 Field Documentation

- 32.2.3.1.0.86.1** `uint8_t dsi_config_t::numLanes`
- 32.2.3.1.0.86.2** `bool dsi_config_t::enableNonContinuousHsClk`
- 32.2.3.1.0.86.3** `bool dsi_config_t::enableTxUlps`
- 32.2.3.1.0.86.4** `bool dsi_config_t::autoInsertEoTp`
- 32.2.3.1.0.86.5** `uint8_t dsi_config_t::numExtraEoTp`
- 32.2.3.1.0.86.6** `uint32_t dsi_config_t::htxTo_ByteClk`
- 32.2.3.1.0.86.7** `uint32_t dsi_config_t::lrxHostTo_ByteClk`
- 32.2.3.1.0.86.8** `uint32_t dsi_config_t::btaTo_ByteClk`

32.2.3.2 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.

32.2.3.2.0.87 Field Documentation

32.2.3.2.0.87.1 uint16_t dsi_dpi_config_t::pixelPayloadSize

Recommended that the line size (in pixels) is evenly divisible by this parameter.

32.2.3.2.0.87.2 dsi_dpi_color_coding_t dsi_dpi_config_t::dpiColorCoding

32.2.3.2.0.87.3 dsi_dpi_pixel_packet_t dsi_dpi_config_t::pixelPacket

32.2.3.2.0.87.4 dsi_dpi_video_mode_t dsi_dpi_config_t::videoMode

32.2.3.2.0.87.5 dsi_dpi_bllp_mode_t dsi_dpi_config_t::bllpMode

32.2.3.2.0.87.6 uint8_t dsi_dpi_config_t::polarityFlags

32.2.3.2.0.87.7 uint16_t dsi_dpi_config_t::hfp

32.2.3.2.0.87.8 uint16_t dsi_dpi_config_t::hbp

32.2.3.2.0.87.9 uint16_t dsi_dpi_config_t::hsw

32.2.3.2.0.87.10 uint8_t dsi_dpi_config_t::vfp

32.2.3.2.0.87.11 uint8_t dsi_dpi_config_t::vbp

32.2.3.2.0.87.12 uint16_t dsi_dpi_config_t::panelHeight

32.2.3.2.0.87.13 uint8_t dsi_dpi_config_t::virtualChannel

32.2.3.3 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.*

32.2.3.3.0.88 Field Documentation

32.2.3.3.0.88.1 uint32_t dsi_dphy_config_t::txHsBitClk_Hz

32.2.3.3.0.88.2 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.

32.2.3.3.0.88.3 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.

32.2.3.3.0.88.4 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.

32.2.3.3.0.88.5 uint32_t dsi_dphy_config_t::tWakeup_EscClk

32.2.3.3.0.88.6 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.

32.2.3.3.0.88.7 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.

32.2.3.3.0.88.8 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.

32.2.3.3.0.88.9 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.

32.2.3.3.0.88.10 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.

32.2.3.3.0.88.11 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.

32.2.3.4 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.
- bool [sendDscCmd](#)
If set to true, the DSC command is specified by [dscCmd](#), otherwise the DSC command is included in the [txData](#).
- uint8_t [dscCmd](#)
The DSC command to send, only valid when [sendDscCmd](#) is true.

32.2.3.4.0.89 Field Documentation**32.2.3.4.0.89.1** `uint8_t dsi_transfer_t::virtualChannel`**32.2.3.4.0.89.2** `dsi_tx_data_type_t dsi_transfer_t::txDataType`**32.2.3.4.0.89.3** `uint8_t dsi_transfer_t::flags`**32.2.3.4.0.89.4** `const uint8_t* dsi_transfer_t::txData`**32.2.3.4.0.89.5** `uint8_t* dsi_transfer_t::rxData`**32.2.3.4.0.89.6** `uint16_t dsi_transfer_t::txDataSize`**32.2.3.4.0.89.7** `uint16_t dsi_transfer_t::rxDataSize`**32.2.3.4.0.89.8** `bool dsi_transfer_t::sendDscCmd`**32.2.3.4.0.89.9** `uint8_t dsi_transfer_t::dscCmd`**32.2.3.5 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.

32.2.3.5.0.90 Field Documentation**32.2.3.5.0.90.1** `volatile bool dsi_handle_t::isBusy`**32.2.3.5.0.90.2** `dsi_transfer_t dsi_handle_t::xfer`**32.2.4 Typedef Documentation****32.2.4.1** `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.

32.2.5 Enumeration Type Documentation

32.2.5.1 anonymous enum

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.

32.2.5.2 enum dsi_dpi_color_coding_t

Enumerator

`kDSI_Dpi16BitConfig1` 16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_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: XXXXXXRR_RRRRGGGG_GGBBB-BBB.
`kDSI_Dpi18BitConfig2` 18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBB-BBB.
`kDSI_Dpi24Bit` 24-bit.

32.2.5.3 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.

32.2.5.4 anonymous enum

Enumerator

kDSI_DpiVsyncActiveLow VSYNC active low.
kDSI_DpiHsyncActiveLow HSYNC active low.
kDSI_DpiVsyncActiveHigh VSYNC active high.
kDSI_DpiHsyncActiveHigh HSYNC active high.

32.2.5.5 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.

32.2.5.6 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.

32.2.5.7 anonymous enum

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.

32.2.5.8 anonymous enum

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.

32.2.5.9 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.
kDSI_HostProtocolViolation DSI protocol violation.
kDSI_HostResetTriggerReceived Reset trigger received.
kDSI_HostTearTriggerReceived Tear effect trigger receive.
kDSI_HostAckTriggerReceived Acknowledge trigger message received.

32.2.5.10 anonymous enum

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_InterruptGroup1ProtocalViolation 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.

32.2.5.11 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.

32.2.5.12 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.

32.2.5.13 anonymous enum

Enumerator

kDSI_TransferUseHighSpeed Use high speed mode or not.
kDSI_TransferPerformBTA Perform BTA or not.

32.2.6 Function Documentation

32.2.6.1 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.

MIPI DSI Driver

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>config</i>	Pointer to a user-defined configuration structure.

32.2.6.2 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.
-------------	--

32.2.6.3 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.
---------------	--

32.2.6.4 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 dsi_dpi_config_t .
<i>dpiPixelClkFreq_Hz</i>	The DPI pixel clock frequency in Hz.
<i>dsiHsBitClkFreq_Hz</i>	The DSI high speed bit clock frequency in Hz. It is the same with DPHY PLL output.

32.2.6.5 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](#).

For some platforms there is not dedicated D-PHY PLL, indicated by the macro FSL_FEATURE_MIPIDSI_NO_DPHY_PLL. For these platforms, the *refClkFreq_Hz* is useless.

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.

32.2.6.6 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.
-------------	--

32.2.6.7 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.

MIPI DSI Driver

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.

32.2.6.8 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.

32.2.6.9 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.

32.2.6.10 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.

32.2.6.11 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.

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 <code>_dsi_transfer_flags</code> .

32.2.6.12 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.

32.2.6.13 void DSI_WriteApbTxPayloadExt (MIPI_DSI_HOST_Type * *base*, const uint8_t * *payload*, uint16_t *payloadSize*, bool *sendDscCmd*, uint8_t *dscCmd*)

Write the long packet payload to TX FIFO. This function could be used in two ways

1. Include the DSC command in parameter `payload`. In this case, the DSC command is the first byte of `payload`. The parameter `sendDscCmd` is set to false, the `dscCmd` is not used. This function is the same as [DSI_WriteApbTxPayload](#) when used in this way.

MIPI DSI Driver

2. The DSC command is not in parameter `payload`, but specified by parameter `dscCmd`. In this case, the parameter `sendDscCmd` is set to true, the `dscCmd` is the DSC command to send. The `payload` is sent after `dscCmd`.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>payload</i>	Pointer to the payload.
<i>payloadSize</i>	Payload size in byte.
<i>sendDscCmd</i>	If set to true, the DSC command is specified by <code>dscCmd</code> , otherwise the DSC command is included in the <code>payload</code> .
<i>dscCmd</i>	The DSC command to send, only used when <code>sendDscCmd</code> is true.

32.2.6.14 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.

32.2.6.15 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.
-------------	--

32.2.6.16 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.

32.2.6.17 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.

32.2.6.18 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);
}
```

Parameters

<i>rxErrorStatus</i>	The error status returned by DSI_GetRxErrorStatus .
----------------------	---

Returns

The 1-bit ECC error position.

32.2.6.19 static uint32_t DSI_GetAndClearHostStatus (MIPI_DSI_HOST_Type * *base*)
[inline], [static]

The host status are returned as mask value of `_dsi_host_status`.

MIPI DSI Driver

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

Returns

The DSI host status.

32.2.6.20 `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.

32.2.6.21 `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 DSI_GetRxPacketHeader .
--------------------	---

Returns

The RX packet type.

32.2.6.22 `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 DSI_GetRxPacketHeader .
--------------------	---

Returns

For long packet, return the payload word count (byte). For short packet, return the (data0 << 8) | data1.

32.2.6.23 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 DSI_GetRxPacketHeader .
--------------------	---

Returns

The virtual channel.

32.2.6.24 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.

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.
------------------------	---------------------------------------

MIPI DSI Driver

<i>kStatus_Timeout</i>	Transfer failed because of timeout.
<i>kStatus_DSI_RxData-Error</i>	RX data error, user could use DSI_GetRxErrorStatus to check the error details.
<i>kStatus_DSI_Error-ReportReceived</i>	Error Report packet received, user could use DSI_GetAndClearHostStatus 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.

32.2.6.25 **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.

32.2.6.26 **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 pervious transfer.
<i>kStatus_DSI_Not-Supported</i>	Transfer format not supported.

32.2.6.27 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.

32.2.6.28 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.

Chapter 33

MU: Messaging Unit

Overview

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

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.

33.2.1 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.

33.2.2 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

33.2.3 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.

33.2.4 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.

33.2.5 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_TEn_SHIFT + 3U)),
`kMU_Tx1EmptyFlag` = (1U << (MU_SR_TEn_SHIFT + 2U)),
`kMU_Tx2EmptyFlag` = (1U << (MU_SR_TEn_SHIFT + 1U)),
`kMU_Tx3EmptyFlag` = (1U << (MU_SR_TEn_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` = (int)(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` = (int)(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, 4))
MU driver version 2.0.3.

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 uint32_t [MU_GetInterruptsPending](#) (MU_Type *base)
Gets the MU IRQ pending status.
- 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)
Holds the other core reset.
- static void [MU_ResetBothSides](#) (MU_Type *base)
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.

Macro Definition Documentation

33.3.1 #define FSL_MU_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

Enumeration Type Documentation

33.4.1 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.

33.4.2 enum _mu_interrupt_enable

Enumerator

kMU_Tx0EmptyInterruptEnable TX0 empty.
kMU_Tx1EmptyInterruptEnable TX1 empty.
kMU_Tx2EmptyInterruptEnable TX2 empty.
kMU_Tx3EmptyInterruptEnable TX3 empty.

Function Documentation

kMU_Rx0FullInterruptEnable RX0 full.
kMU_Rx1FullInterruptEnable RX1 full.
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.

33.4.3 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.

Function Documentation

33.5.1 void MU_Init (MU_Type * *base*)

This function enables the MU clock only.

Parameters

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

33.5.2 void MU_Deinit (MU_Type * *base*)

This function disables the MU clock only.

Parameters

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

33.5.3 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.

33.5.4 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.

33.5.5 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.
*
```

Function Documentation

Parameters

<i>base</i>	MU peripheral base address.
<i>regIndex</i>	TX register index.

Returns

The received message.

33.5.6 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.

33.5.7 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.

33.5.8 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.

33.5.9 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.

33.5.10 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);
* }
```

Function Documentation

```
* if (kMU_Tx1EmptyFlag & flags)
* {
*     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`.

33.5.11 `static uint32_t MU_GetInterruptsPending (MU_Type * base) [inline], [static]`

This function returns the bit mask of the pending MU IRQs.

Parameters

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

Returns

Bit mask of the MU IRQs pending.

33.5.12 `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"> • <code>kMU_Tx0EmptyFlag</code> • <code>kMU_Tx1EmptyFlag</code> • <code>kMU_Tx2EmptyFlag</code> • <code>kMU_Tx3EmptyFlag</code> • <code>kMU_Rx0FullFlag</code> • <code>kMU_Rx1FullFlag</code> • <code>kMU_Rx2FullFlag</code> • <code>kMU_Rx3FullFlag</code> • <code>kMU_EventPendingFlag</code> • <code>kMU_FlagsUpdatingFlag</code> • <code>kMU_OtherSideInResetFlag</code>

33.5.13 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 <code>_mu_interrupt_enable</code> .

33.5.14 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);
*
```

Function Documentation

Parameters

<i>base</i>	MU peripheral base address.
<i>mask</i>	Bit mask of the MU interrupts. See <code>_mu_interrupt_enable</code> .

33.5.15 `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.  
* }  
*
```

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.

33.5.16 `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.
-------------	-----------------------------

33.5.17 `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.

33.5.18 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.

33.5.19 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.

33.5.20 static void MU_HoldOtherCoreReset (MU_Type * *base*) [inline], [static]

This function causes the other core to be held in reset following any reset event.

Function Documentation

Parameters

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

33.5.21 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

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

Note

For some platforms, only MU side A could use this function, check reference manual for details.

33.5.22 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_Mask-HardwareReset. 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"> • true: Wait until the other core enters reset, if the other core has masked the hardware reset, then this function will be blocked. • false: Don't wait the reset.
<i>holdReset</i>	Hold the other core reset or not. <ul style="list-style-type: none"> • true: Hold the other core in reset, this function returns directly when the other core enters reset. • false: Don't hold the other core in reset, this function waits until the other core out of reset.
<i>bootMode</i>	Boot mode of the other core, if <i>holdReset</i> is true, this parameter is useless.

33.5.23 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.

33.5.24 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.

Chapter 34

PRG: Prefetch Resolve Gasket

Overview

The MCUXpresso SDK provides a peripheral driver for the PRG module of MCUXpresso SDK devices. The PRG works with Display Prefetch Resolve (DPR) to prefetch the frame buffer data for display controller.

Data Structures

- struct `prg_buffer_config_t`
Frame buffer configuration. [More...](#)

Macros

- #define `FSL_PRG_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)
Driver version.

Enumerations

- enum `prg_data_type_t` {
 `kPRG_DataType32Bpp` = 0x0U,
 `kPRG_DataType24Bpp`,
 `kPRG_DataType16Bpp`,
 `kPRG_DataType8Bpp` }
Data type of the frame buffer.

Functions

- void `PRG_Init` (`PRG_Type *base`)
Enables and configures the PRG peripheral module.
- void `PRG_Deinit` (`PRG_Type *base`)
Disables the PRG peripheral module.
- static void `PRG_Enable` (`PRG_Type *base`, bool enable)
Enable or disable the PRG.
- static void `PRG_EnableShadowLoad` (`PRG_Type *base`, bool enable)
Enable or disable the shadow load.
- static void `PRG_UpdateRegister` (`PRG_Type *base`)
Update the registers.
- void `PRG_SetBufferConfig` (`PRG_Type *base`, const `prg_buffer_config_t *config`)
Set the frame buffer configuration.
- void `PRG_BufferGetDefaultConfig` (`prg_buffer_config_t *config`)
Get the frame buffer default configuration.
- static void `PRG_SetBufferAddr` (`PRG_Type *base`, `uint32_t addr`)
Set the frame buffer address.

Function Documentation

Data Structure Documentation

34.2.1 struct prg_buffer_config_t

Data Fields

- uint16_t [width](#)
Frame buffer width.
- uint16_t [height](#)
Frame buffer height.
- uint16_t [strideBytes](#)
Stride, must be 8 bytes aligned.
- [prg_data_type_t](#) [dataType](#)
Data type.

34.2.1.0.0.91 Field Documentation

34.2.1.0.0.91.1 uint16_t prg_buffer_config_t::width

34.2.1.0.0.91.2 uint16_t prg_buffer_config_t::height

34.2.1.0.0.91.3 uint16_t prg_buffer_config_t::strideBytes

34.2.1.0.0.91.4 prg_data_type_t prg_buffer_config_t::dataType

Macro Definition Documentation

34.3.1 #define FSL_PRG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Enumeration Type Documentation

34.4.1 enum prg_data_type_t

Enumerator

kPRG_DataType32Bpp 32 bits per pixel.

kPRG_DataType24Bpp 24 bits per pixel.

kPRG_DataType16Bpp 16 bits per pixel.

kPRG_DataType8Bpp 8 bits per pixel.

Function Documentation

34.5.1 void PRG_Init (PRG_Type * *base*)

Parameters

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

34.5.2 void PRG_Deinit (PRG_Type * *base*)

Parameters

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

34.5.3 static void PRG_Enable (PRG_Type * *base*, bool *enable*) [inline], [static]

If enabled, display controller fetches data from PRG. If disabled, display controller fetches data from frame buffer.

Parameters

<i>base</i>	PRG peripheral address.
<i>enable</i>	Pass in true to enable, false to disable

34.5.4 static void PRG_EnableShadowLoad (PRG_Type * *base*, bool *enable*) [inline], [static]

If disabled, the function [PRG_UpdateRegister](#) makes the new configurations take effect immediately. If enabled, after calling [PRG_UpdateRegister](#), the new configurations take effect at next frame.

Parameters

<i>base</i>	PRG peripheral address.
<i>enable</i>	Pass in true to enable, false to disable

34.5.5 static void PRG_UpdateRegister (PRG_Type * *base*) [inline], [static]

New configurations set to PRG registers will not take effect immediately until this function is called. If the shadow is disabled by [PRG_EnableShadowLoad](#), the new configurations take effect immediately after

Function Documentation

this function is called. If the shadow is enabled by [PRG_EnableShadowLoad](#), the new configurations take effect at next frame after this function is called.

Parameters

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

34.5.6 void PRG_SetBufferConfig (PRG_Type * *base*, const prg_buffer_config_t * *config*)

Parameters

<i>base</i>	PRG peripheral address.
<i>config</i>	Pointer to the configuration.

34.5.7 void PRG_BufferGetDefaultConfig (prg_buffer_config_t * *config*)

The default configuration is:

```
config->width = 1080U;
config->height = 1920U;
config->strideBytes = 4U * 1080U;
config->dataType = kPRG_DataType32Bpp;
```

Parameters

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

34.5.8 static void PRG_SetBufferAddr (PRG_Type * *base*, uint32_t *addr*) [inline], [static]

Parameters

<i>base</i>	PRG peripheral address.
<i>addr</i>	Frame buffer address.

Chapter 35

RGPIO: Rapid General-Purpose Input/Output Driver

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, 2))
RGPIO driver version 2.0.2.

Data Structure Documentation

35.2.1 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

Macro Definition Documentation

35.3.1 #define FSL_RGPIODRIVER_VERSION (MAKE_VERSION(2, 0, 2))

Enumeration Type Documentation

35.4.1 enum rgpio_pin_direction_t

Enumerator

kRGPIODigitalInput Set current pin as digital input.

kRGPIODigitalOutput Set current pin as digital output.

RGPIO Driver

35.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Rapid General-Purpose Input/Output (RGPIO) module of MCUXpresso SDK devices.

35.5.2 Typical use case

35.5.2.1 Output Operation

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

35.5.2.2 Input Operation

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

RGPIO 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.

RGPIO 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 Driver

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.

35.5.3 Function Documentation

35.5.3.1 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

35.5.3.2 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
--------------	----------

35.5.3.3 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"> • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

35.5.3.4 static void RGPIO_WritePinOutput (RGPIO_Type * *base*, uint32_t *pin*, uint8_t *output*) [inline], [static]

Deprecated Do not use this function. It has been superceded by [RGPIO_PinWrite](#).

35.5.3.5 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

35.5.3.6 static void RGPIO_SetPinsOutput (RGPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by [RGPIO_PortSet](#).

35.5.3.7 `static void RGPIO_PortClear (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

35.5.3.8 `static void RGPIO_ClearPinsOutput (RGPIO_Type * base, uint32_t mask)`
`[inline], [static]`

Deprecated Do not use this function. It has been superceded by [RGPIO_PortClear](#).

Parameters

<i>base</i>	RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
<i>mask</i>	RGPIO pin number macro

35.5.3.9 `static void RGPIO_PortToggle (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

35.5.3.10 `static void RGPIO_TogglePinsOutput (RGPIO_Type * base, uint32_t mask)`
`[inline], [static]`

Deprecated Do not use this function. It has been superceded by [RGPIO_PortToggle](#).

35.5.3.11 `static uint32_t RGPIO_PinRead (RGPIO_Type * base, uint32_t pin)`
`[inline], [static]`

Parameters

RGPIO Driver

<i>base</i>	RGPIO peripheral base pointer (RGPIOA, RGPIOB, RGPIOC, and so on.)
<i>pin</i>	RGPIO pin number

Return values

<i>RGPIO</i>	port input value <ul style="list-style-type: none">• 0: corresponding pin input low-logic level.• 1: corresponding pin input high-logic level.
--------------	---

35.5.3.12 `static uint32_t RGPIO_ReadPinInput (RGPIO_Type * base, uint32_t pin)`
`[inline], [static]`

Deprecated Do not use this function. It has been superseded by [RGPIO_PinRead](#).

FGPIO Driver

This section 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.

35.6.1 Typical use case

35.6.1.1 Output Operation

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

35.6.1.2 Input Operation

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

Chapter 36

SAI: Serial Audio Interface

Overview

The MCUXpresso SDK provides a peripheral driver for the Serial Audio Interface (SAI) module of MCUXpresso SDK devices.

SAI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SAI initialization, configuration and operation, and for optimization and customization purposes. Using the functional API requires the knowledge of the SAI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SAI functional operation groups provide the functional API set.

Transactional APIs 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 `sai_handle_t` as the first parameter. Initialize the handle by calling the [SAI_TransferTxCreateHandle\(\)](#) or [SAI_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SAI_TransferSendNonBlocking\(\)](#) and [SAI_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SAI_TxIdle` and `kStatus_SAI_RxIdle` status.

Typical configurations

Bit width configuration

SAI driver support 8/16/24/32bits stereo/mono raw audio data transfer. SAI EDMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI DMA driver support 8/16/32bits stereo/mono raw audio data transfer, since the EDMA doesn't support 24bit data width, so application should pre-convert the 24bit data to 32bit. SAI SDMA driver support 8/16/24/32bits stereo/mono raw audio data transfer.

Frame configuration

SAI driver support I2S, DSP, Left justified, Right justified, TDM mode. Application can call the api directly: `SAI_GetClassicI2SConfig` `SAI_GetLeftJustifiedConfig` `SAI_GetRightJustifiedConfig` `SAI_GetTDMConfig` `SAI_GetDSPConfig`

Typical use case

Typical use case

36.3.1 SAI Send/receive using an interrupt method

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

36.3.2 SAI Send/receive using a DMA method

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

Modules

- [SAI Driver](#)
- [SAI EDMA Driver](#)

SAI Driver

36.4.1 Overview

Data Structures

- struct [sai_config_t](#)
SAI user configuration structure. [More...](#)
- struct [sai_transfer_format_t](#)
sai transfer format [More...](#)
- struct [sai_fifo_t](#)
sai fifo configurations [More...](#)
- struct [sai_bit_clock_t](#)
sai bit clock configurations [More...](#)
- struct [sai_frame_sync_t](#)
sai frame sync configurations [More...](#)
- struct [sai_serial_data_t](#)
sai serial data configurations [More...](#)
- struct [sai_transceiver_t](#)
sai transceiver configurations [More...](#)
- struct [sai_transfer_t](#)
SAI transfer structure. [More...](#)
- struct [sai_handle_t](#)
SAI handle structure. [More...](#)

Macros

- #define [SAI_XFER_QUEUE_SIZE](#) (4U)
SAI transfer queue size, user can refine it according to use case.
- #define [FSL_SAI_HAS_FIFO_EXTEND_FEATURE](#) 1
sai fifo feature

Typedefs

- typedef void(* [sai_transfer_callback_t](#))(I2S_Type *base, sai_handle_t *handle, [status_t](#) status, void *userData)
SAI transfer callback prototype.

Enumerations

- enum {
`kStatus_SAI_TxBusy` = MAKE_STATUS(kStatusGroup_SAI, 0),
`kStatus_SAI_RxBusy` = MAKE_STATUS(kStatusGroup_SAI, 1),
`kStatus_SAI_TxError` = MAKE_STATUS(kStatusGroup_SAI, 2),
`kStatus_SAI_RxError` = MAKE_STATUS(kStatusGroup_SAI, 3),
`kStatus_SAI_QueueFull` = MAKE_STATUS(kStatusGroup_SAI, 4),
`kStatus_SAI_TxIdle` = MAKE_STATUS(kStatusGroup_SAI, 5),
`kStatus_SAI_RxIdle` = MAKE_STATUS(kStatusGroup_SAI, 6) }
_sai_status_t, SAI return status.
- enum {
`kSAI_Channel0Mask` = 1 << 0U,
`kSAI_Channel1Mask` = 1 << 1U,
`kSAI_Channel2Mask` = 1 << 2U,
`kSAI_Channel3Mask` = 1 << 3U,
`kSAI_Channel4Mask` = 1 << 4U,
`kSAI_Channel5Mask` = 1 << 5U,
`kSAI_Channel6Mask` = 1 << 6U,
`kSAI_Channel7Mask` = 1 << 7U }
_sai_channel_mask, sai channel mask value, actual channel numbers is depend soc specific
- enum `sai_protocol_t` {
`kSAI_BusLeftJustified` = 0x0U,
`kSAI_BusRightJustified`,
`kSAI_BusI2S`,
`kSAI_BusPCMA`,
`kSAI_BusPCMB` }
Define the SAI bus type.
- enum `sai_master_slave_t` {
`kSAI_Master` = 0x0U,
`kSAI_Slave` = 0x1U,
`kSAI_Bclk_Master_FrameSync_Slave` = 0x2U,
`kSAI_Bclk_Slave_FrameSync_Master` = 0x3U }
Master or slave mode.
- enum `sai_mono_stereo_t` {
`kSAI_Stereo` = 0x0U,
`kSAI_MonoRight`,
`kSAI_MonoLeft` }
Mono or stereo audio format.
- enum `sai_data_order_t` {
`kSAI_DataLSB` = 0x0U,
`kSAI_DataMSB` }
SAI data order, MSB or LSB.
- enum `sai_clock_polarity_t` {

- kSAI_PolarityActiveHigh = 0x0U,
 - kSAI_PolarityActiveLow = 0x1U,
 - kSAI_SampleOnFallingEdge = 0x0U,
 - kSAI_SampleOnRisingEdge = 0x1U }

SAI clock polarity, active high or low.
- enum sai_sync_mode_t {
 - kSAI_ModeAsync = 0x0U,
 - kSAI_ModeSync }

Synchronous or asynchronous mode.
- enum sai_bclk_source_t {
 - kSAI_BclkSourceBusclk = 0x0U,
 - kSAI_BclkSourceMclkOption1 = 0x1U,
 - kSAI_BclkSourceMclkOption2 = 0x2U,
 - kSAI_BclkSourceMclkOption3 = 0x3U,
 - kSAI_BclkSourceMclkDiv = 0x1U,
 - kSAI_BclkSourceOtherSai0 = 0x2U,
 - kSAI_BclkSourceOtherSai1 = 0x3U }

Bit clock source.
- enum {
 - kSAI_WordStartInterruptEnable,
 - kSAI_SyncErrorInterruptEnable = I2S_TCSR_SEIE_MASK,
 - kSAI_FIFOWarningInterruptEnable = I2S_TCSR_FWIE_MASK,
 - kSAI_FIFOErrorInterruptEnable = I2S_TCSR_FEIE_MASK,
 - kSAI_FIFORequestInterruptEnable = I2S_TCSR_FRIE_MASK }

_sai_interrupt_enable_t, The SAI interrupt enable flag
- enum {
 - kSAI_FIFOWarningDMAEnable = I2S_TCSR_FWDE_MASK,
 - kSAI_FIFORequestDMAEnable = I2S_TCSR_FRDE_MASK }

_sai_dma_enable_t, The DMA request sources
- enum {
 - kSAI_WordStartFlag = I2S_TCSR_WSF_MASK,
 - kSAI_SyncErrorFlag = I2S_TCSR_SEF_MASK,
 - kSAI_FIFOErrorFlag = I2S_TCSR_FEF_MASK,
 - kSAI_FIFORequestFlag = I2S_TCSR_FRF_MASK,
 - kSAI_FIFOWarningFlag = I2S_TCSR_FWF_MASK }

_sai_flags, The SAI status flag
- enum sai_reset_type_t {
 - kSAI_ResetTypeSoftware = I2S_TCSR_SR_MASK,
 - kSAI_ResetTypeFIFO = I2S_TCSR_FR_MASK,
 - kSAI_ResetAll = I2S_TCSR_SR_MASK | I2S_TCSR_FR_MASK }

The reset type.
- enum sai_fifo_packing_t {
 - kSAI_FifoPackingDisabled = 0x0U,
 - kSAI_FifoPacking8bit = 0x2U,
 - kSAI_FifoPacking16bit = 0x3U }

The SAI packing mode The mode includes 8 bit and 16 bit packing.
- enum sai_sample_rate_t {

SAI Driver

```
kSAI_SampleRate8KHz = 8000U,  
kSAI_SampleRate11025Hz = 11025U,  
kSAI_SampleRate12KHz = 12000U,  
kSAI_SampleRate16KHz = 16000U,  
kSAI_SampleRate22050Hz = 22050U,  
kSAI_SampleRate24KHz = 24000U,  
kSAI_SampleRate32KHz = 32000U,  
kSAI_SampleRate44100Hz = 44100U,  
kSAI_SampleRate48KHz = 48000U,  
kSAI_SampleRate96KHz = 96000U,  
kSAI_SampleRate192KHz = 192000U,  
kSAI_SampleRate384KHz = 384000U }
```

Audio sample rate.

- enum `sai_word_width_t` {
 `kSAI_WordWidth8bits` = 8U,
 `kSAI_WordWidth16bits` = 16U,
 `kSAI_WordWidth24bits` = 24U,
 `kSAI_WordWidth32bits` = 32U }

Audio word width.

- enum `sai_transceiver_type_t` {
 `kSAI_Transmitter` = 0U,
 `kSAI_Receiver` = 1U }

sai transceiver type

- enum `sai_frame_sync_len_t` {
 `kSAI_FrameSyncLenOneBitClk` = 0U,
 `kSAI_FrameSyncLenPerWordWidth` = 1U }

sai frame sync len

Driver version

- #define `FSL_SAI_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 1)`)
 Version 2.3.1.

Initialization and deinitialization

- void `SAI_TxInit` (I2S_Type *base, const `sai_config_t` *config)
 Initializes the SAI Tx peripheral.
- void `SAI_RxInit` (I2S_Type *base, const `sai_config_t` *config)
 Initializes the SAI Rx peripheral.
- void `SAI_TxGetDefaultConfig` (`sai_config_t` *config)
 Sets the SAI Tx configuration structure to default values.
- void `SAI_RxGetDefaultConfig` (`sai_config_t` *config)
 Sets the SAI Rx configuration structure to default values.
- void `SAI_Init` (I2S_Type *base)
 Initializes the SAI peripheral.
- void `SAI_Deinit` (I2S_Type *base)

- *De-initializes the SAI peripheral.*
- void **SAI_TxReset** (I2S_Type *base)
Resets the SAI Tx.
- void **SAI_RxReset** (I2S_Type *base)
Resets the SAI Rx.
- void **SAI_TxEnable** (I2S_Type *base, bool enable)
Enables/disables the SAI Tx.
- void **SAI_RxEnable** (I2S_Type *base, bool enable)
Enables/disables the SAI Rx.
- static void **SAI_TxSetBitClockDirection** (I2S_Type *base, sai_master_slave_t masterSlave)
Set Rx bit clock direction.
- static void **SAI_RxSetBitClockDirection** (I2S_Type *base, sai_master_slave_t masterSlave)
Set Rx bit clock direction.
- static void **SAI_RxSetFrameSyncDirection** (I2S_Type *base, sai_master_slave_t masterSlave)
Set Rx frame sync direction.
- static void **SAI_TxSetFrameSyncDirection** (I2S_Type *base, sai_master_slave_t masterSlave)
Set Tx frame sync direction.
- void **SAI_TxSetBitClockRate** (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)
Transmitter bit clock rate configurations.
- void **SAI_RxSetBitClockRate** (I2S_Type *base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)
Receiver bit clock rate configurations.
- void **SAI_TxSetBitclockConfig** (I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)
Transmitter Bit clock configurations.
- void **SAI_RxSetBitclockConfig** (I2S_Type *base, sai_master_slave_t masterSlave, sai_bit_clock_t *config)
Receiver Bit clock configurations.
- void **SAI_TxSetFifoConfig** (I2S_Type *base, sai_fifo_t *config)
SAI transmitter fifo configurations.
- void **SAI_RxSetFifoConfig** (I2S_Type *base, sai_fifo_t *config)
SAI receiver fifo configurations.
- void **SAI_TxSetFrameSyncConfig** (I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
SAI transmitter Frame sync configurations.
- void **SAI_RxSetFrameSyncConfig** (I2S_Type *base, sai_master_slave_t masterSlave, sai_frame_sync_t *config)
SAI receiver Frame sync configurations.
- void **SAI_TxSetSerialDataConfig** (I2S_Type *base, sai_serial_data_t *config)
SAI transmitter Serial data configurations.
- void **SAI_RxSetSerialDataConfig** (I2S_Type *base, sai_serial_data_t *config)
SAI receiver Serial data configurations.
- void **SAI_TxSetConfig** (I2S_Type *base, sai_transceiver_t *config)
SAI transmitter configurations.
- void **SAI_RxSetConfig** (I2S_Type *base, sai_transceiver_t *config)
SAI receiver configurations.
- void **SAI_GetClassicI2SConfig** (sai_transceiver_t *config, sai_word_width_t bitWidth, sai_mono_stereo_t mode, uint32_t saiChannelMask)
Get classic I2S mode configurations.

SAI Driver

- void [SAI_GetLeftJustifiedConfig](#) ([sai_transceiver_t](#) *config, [sai_word_width_t](#) bitWidth, [sai_mono_stereo_t](#) mode, [uint32_t](#) saiChannelMask)
Get left justified mode configurations.
- void [SAI_GetRightJustifiedConfig](#) ([sai_transceiver_t](#) *config, [sai_word_width_t](#) bitWidth, [sai_mono_stereo_t](#) mode, [uint32_t](#) saiChannelMask)
Get right justified mode configurations.
- void [SAI_GetTDMConfig](#) ([sai_transceiver_t](#) *config, [sai_frame_sync_len_t](#) frameSyncWidth, [sai_word_width_t](#) bitWidth, [uint32_t](#) dataWordNum, [uint32_t](#) saiChannelMask)
Get TDM mode configurations.
- void [SAI_GetDSPConfig](#) ([sai_transceiver_t](#) *config, [sai_frame_sync_len_t](#) frameSyncWidth, [sai_word_width_t](#) bitWidth, [sai_mono_stereo_t](#) mode, [uint32_t](#) saiChannelMask)
Get DSP mode configurations.

Status

- static [uint32_t](#) [SAI_TxGetStatusFlag](#) ([I2S_Type](#) *base)
Gets the SAI Tx status flag state.
- static void [SAI_TxClearStatusFlags](#) ([I2S_Type](#) *base, [uint32_t](#) mask)
Clears the SAI Tx status flag state.
- static [uint32_t](#) [SAI_RxGetStatusFlag](#) ([I2S_Type](#) *base)
Gets the SAI Rx status flag state.
- static void [SAI_RxClearStatusFlags](#) ([I2S_Type](#) *base, [uint32_t](#) mask)
Clears the SAI Rx status flag state.
- void [SAI_TxSoftwareReset](#) ([I2S_Type](#) *base, [sai_reset_type_t](#) type)
Do software reset or FIFO reset .
- void [SAI_RxSoftwareReset](#) ([I2S_Type](#) *base, [sai_reset_type_t](#) type)
Do software reset or FIFO reset .
- void [SAI_TxSetChannelFIFOMask](#) ([I2S_Type](#) *base, [uint8_t](#) mask)
Set the Tx channel FIFO enable mask.
- void [SAI_RxSetChannelFIFOMask](#) ([I2S_Type](#) *base, [uint8_t](#) mask)
Set the Rx channel FIFO enable mask.
- void [SAI_TxSetDataOrder](#) ([I2S_Type](#) *base, [sai_data_order_t](#) order)
Set the Tx data order.
- void [SAI_RxSetDataOrder](#) ([I2S_Type](#) *base, [sai_data_order_t](#) order)
Set the Rx data order.
- void [SAI_TxSetBitClockPolarity](#) ([I2S_Type](#) *base, [sai_clock_polarity_t](#) polarity)
Set the Tx data order.
- void [SAI_RxSetBitClockPolarity](#) ([I2S_Type](#) *base, [sai_clock_polarity_t](#) polarity)
Set the Rx data order.
- void [SAI_TxSetFrameSyncPolarity](#) ([I2S_Type](#) *base, [sai_clock_polarity_t](#) polarity)
Set the Tx data order.
- void [SAI_RxSetFrameSyncPolarity](#) ([I2S_Type](#) *base, [sai_clock_polarity_t](#) polarity)
Set the Rx data order.
- void [SAI_TxSetFIFOPacking](#) ([I2S_Type](#) *base, [sai_fifo_packing_t](#) pack)
Set Tx FIFO packing feature.
- void [SAI_RxSetFIFOPacking](#) ([I2S_Type](#) *base, [sai_fifo_packing_t](#) pack)
Set Rx FIFO packing feature.
- static void [SAI_TxSetFIFOErrorContinue](#) ([I2S_Type](#) *base, bool isEnabled)
Set Tx FIFO error continue.

- static void [SAI_RxSetFIFOErrorContinue](#) (I2S_Type *base, bool isEnabled)
Set Rx FIFO error continue.

Interrupts

- static void [SAI_TxEnableInterrupts](#) (I2S_Type *base, uint32_t mask)
Enables the SAI Tx interrupt requests.
- static void [SAI_RxEnableInterrupts](#) (I2S_Type *base, uint32_t mask)
Enables the SAI Rx interrupt requests.
- static void [SAI_TxDisableInterrupts](#) (I2S_Type *base, uint32_t mask)
Disables the SAI Tx interrupt requests.
- static void [SAI_RxDisableInterrupts](#) (I2S_Type *base, uint32_t mask)
Disables the SAI Rx interrupt requests.

DMA Control

- static void [SAI_TxEnableDMA](#) (I2S_Type *base, uint32_t mask, bool enable)
Enables/disables the SAI Tx DMA requests.
- static void [SAI_RxEnableDMA](#) (I2S_Type *base, uint32_t mask, bool enable)
Enables/disables the SAI Rx DMA requests.
- static uint32_t [SAI_TxGetDataRegisterAddress](#) (I2S_Type *base, uint32_t channel)
Gets the SAI Tx data register address.
- static uint32_t [SAI_RxGetDataRegisterAddress](#) (I2S_Type *base, uint32_t channel)
Gets the SAI Rx data register address.

Bus Operations

- void [SAI_TxSetFormat](#) (I2S_Type *base, [sai_transfer_format_t](#) *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Tx audio format.
- void [SAI_RxSetFormat](#) (I2S_Type *base, [sai_transfer_format_t](#) *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Rx audio format.
- void [SAI_WriteBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Sends data using a blocking method.
- void [SAI_WriteMultiChannelBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Sends data to multi channel using a blocking method.
- static void [SAI_WriteData](#) (I2S_Type *base, uint32_t channel, uint32_t data)
Writes data into SAI FIFO.
- void [SAI_ReadBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t bitWidth, uint8_t *buffer, uint32_t size)
Receives data using a blocking method.
- void [SAI_ReadMultiChannelBlocking](#) (I2S_Type *base, uint32_t channel, uint32_t channelMask, uint32_t bitWidth, uint8_t *buffer, uint32_t size)

SAI Driver

Receives multi channel data using a blocking method.

- static uint32_t [SAI_ReadData](#) (I2S_Type *base, uint32_t channel)
Reads data from the SAI FIFO.

Transactional

- void [SAI_TransferTxCreateHandle](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t callback, void *userData)
Initializes the SAI Tx handle.
- void [SAI_TransferRxCreateHandle](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_callback_t callback, void *userData)
Initializes the SAI Rx handle.
- void [SAI_TransferTxSetConfig](#) (I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)
SAI transmitter transfer configurations.
- void [SAI_TransferRxSetConfig](#) (I2S_Type *base, sai_handle_t *handle, sai_transceiver_t *config)
SAI receiver transfer configurations.
- status_t [SAI_TransferTxSetFormat](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Tx audio format.
- status_t [SAI_TransferRxSetFormat](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_format_t *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Rx audio format.
- status_t [SAI_TransferSendNonBlocking](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)
Performs an interrupt non-blocking send transfer on SAI.
- status_t [SAI_TransferReceiveNonBlocking](#) (I2S_Type *base, sai_handle_t *handle, sai_transfer_t *xfer)
Performs an interrupt non-blocking receive transfer on SAI.
- status_t [SAI_TransferGetSendCount](#) (I2S_Type *base, sai_handle_t *handle, size_t *count)
Gets a set byte count.
- status_t [SAI_TransferGetReceiveCount](#) (I2S_Type *base, sai_handle_t *handle, size_t *count)
Gets a received byte count.
- void [SAI_TransferAbortSend](#) (I2S_Type *base, sai_handle_t *handle)
Aborts the current send.
- void [SAI_TransferAbortReceive](#) (I2S_Type *base, sai_handle_t *handle)
Aborts the current IRQ receive.
- void [SAI_TransferTerminateSend](#) (I2S_Type *base, sai_handle_t *handle)
Terminate all SAI send.
- void [SAI_TransferTerminateReceive](#) (I2S_Type *base, sai_handle_t *handle)
Terminate all SAI receive.
- void [SAI_TransferTxHandleIRQ](#) (I2S_Type *base, sai_handle_t *handle)
Tx interrupt handler.
- void [SAI_TransferRxHandleIRQ](#) (I2S_Type *base, sai_handle_t *handle)
Rx interrupt handler.

36.4.2 Data Structure Documentation

36.4.2.1 struct sai_config_t

Data Fields

- [sai_protocol_t protocol](#)
Audio bus protocol in SAI.
- [sai_sync_mode_t syncMode](#)
SAI sync mode, control Tx/Rx clock sync.
- [sai_bclk_source_t bclkSource](#)
Bit Clock source.
- [sai_master_slave_t masterSlave](#)
Master or slave.

36.4.2.2 struct sai_transfer_format_t

Data Fields

- uint32_t [sampleRate_Hz](#)
Sample rate of audio data.
- uint32_t [bitWidth](#)
Data length of audio data, usually 8/16/24/32 bits.
- [sai_mono_stereo_t stereo](#)
Mono or stereo.
- uint8_t [watermark](#)
Watermark value.
- uint8_t [channel](#)
Transfer start channel.
- uint8_t [channelMask](#)
enabled channel mask value, reference `_sai_channel_mask`
- uint8_t [endChannel](#)
end channel number
- uint8_t [channelNums](#)
Total enabled channel numbers.
- [sai_protocol_t protocol](#)
Which audio protocol used.
- bool [isFrameSyncCompact](#)
True means Frame sync length is configurable according to bitWidth, false means frame sync length is 64 times of bit clock.

SAI Driver

36.4.2.2.0.92 Field Documentation

36.4.2.2.0.92.1 bool sai_transfer_format_t::isFrameSyncCompact

36.4.2.3 struct sai_fifo_t

Data Fields

- bool [fifoContinueOnError](#)
fifo continues when error occur
- [sai_fifo_packing_t](#) [fifoPacking](#)
fifo packing mode
- uint8_t [fifoWatermark](#)
fifo watermark

36.4.2.4 struct sai_bit_clock_t

Data Fields

- bool [bclkSrcSwap](#)
bit clock source swap
- bool [bclkInputDelay](#)
bit clock actually used by the transmitter is delayed by the pad output delay, this has effect of decreasing the data input setup time, but increasing the data output valid time .
- [sai_clock_polarity_t](#) [bclkPolarity](#)
bit clock polarity
- [sai_bclk_source_t](#) [bclkSource](#)
bit Clock source

36.4.2.4.0.93 Field Documentation

36.4.2.4.0.93.1 bool sai_bit_clock_t::bclkInputDelay

36.4.2.5 struct sai_frame_sync_t

Data Fields

- uint8_t [frameSyncWidth](#)
frame sync width in number of bit clocks
- bool [frameSyncEarly](#)
TRUE is frame sync assert one bit before the first bit of frame FALSE is frame sync assert with the first bit of the frame.
- [sai_clock_polarity_t](#) [frameSyncPolarity](#)
frame sync polarity

36.4.2.6 struct sai_serial_data_t

Data Fields

- [sai_data_order_t dataOrder](#)
configure whether the LSB or MSB is transmitted first
- [uint8_t dataWord0Length](#)
configure the number of bits in the first word in each frame
- [uint8_t dataWordNLength](#)
configure the number of bits in the each word in each frame, except the first word
- [uint8_t dataWordLength](#)
used to record the data length for dma transfer
- [uint8_t dataFirstBitShifted](#)
Configure the bit index for the first bit transmitted for each word in the frame.
- [uint8_t dataWordNum](#)
configure the number of words in each frame
- [uint32_t dataMaskedWord](#)
configure whether the transmit word is masked

36.4.2.7 struct sai_transceiver_t

Data Fields

- [sai_serial_data_t serialData](#)
serial data configurations
- [sai_frame_sync_t frameSync](#)
ws configurations
- [sai_bit_clock_t bitClock](#)
bit clock configurations
- [sai_fifo_t fifo](#)
fifo configurations
- [sai_master_slave_t masterSlave](#)
transceiver is master or slave
- [sai_sync_mode_t syncMode](#)
transceiver sync mode
- [uint8_t startChannel](#)
Transfer start channel.
- [uint8_t channelMask](#)
enabled channel mask value, reference `_sai_channel_mask`
- [uint8_t endChannel](#)
end channel number
- [uint8_t channelNums](#)
Total enabled channel numbers.

36.4.2.8 struct sai_transfer_t

Data Fields

- [uint8_t * data](#)

SAI Driver

- *Data start address to transfer.*
size_t [dataSize](#)
Transfer size.

36.4.2.8.0.94 Field Documentation

36.4.2.8.0.94.1 uint8_t* sai_transfer_t::data

36.4.2.8.0.94.2 size_t sai_transfer_t::dataSize

36.4.2.9 struct _sai_handle

Data Fields

- I2S_Type * [base](#)
base address
- uint32_t [state](#)
Transfer status.
- [sai_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.
- uint8_t [channel](#)
Transfer start channel.
- uint8_t [channelMask](#)
enabled channel mask value, refernece _sai_channel_mask
- uint8_t [endChannel](#)
end channel number
- uint8_t [channelNums](#)
Total enabled channel numbers.
- [sai_transfer_t](#) [saiQueue](#) [[SAI_XFER_QUEUE_SIZE](#)]
Transfer queue storing queued transfer.
- size_t [transferSize](#) [[SAI_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.

36.4.3 Macro Definition Documentation

36.4.3.1 #define SAI_XFER_QUEUE_SIZE (4U)

36.4.4 Enumeration Type Documentation

36.4.4.1 anonymous enum

Enumerator

kStatus_SAI_TxBusy SAI Tx is busy.
kStatus_SAI_RxBusy SAI Rx is busy.
kStatus_SAI_TxError SAI Tx FIFO error.
kStatus_SAI_RxError SAI Rx FIFO error.
kStatus_SAI_QueueFull SAI transfer queue is full.
kStatus_SAI_TxIdle SAI Tx is idle.
kStatus_SAI_RxIdle SAI Rx is idle.

36.4.4.2 anonymous enum

Enumerator

kSAI_Channel0Mask channel 0 mask value
kSAI_Channel1Mask channel 1 mask value
kSAI_Channel2Mask channel 2 mask value
kSAI_Channel3Mask channel 3 mask value
kSAI_Channel4Mask channel 4 mask value
kSAI_Channel5Mask channel 5 mask value
kSAI_Channel6Mask channel 6 mask value
kSAI_Channel7Mask channel 7 mask value

36.4.4.3 enum sai_protocol_t

Enumerator

kSAI_BusLeftJustified Uses left justified format.
kSAI_BusRightJustified Uses right justified format.
kSAI_BusI2S Uses I2S format.
kSAI_BusPCMA Uses I2S PCM A format.
kSAI_BusPCMB Uses I2S PCM B format.

SAI Driver

36.4.4.4 enum sai_master_slave_t

Enumerator

kSAI_Master Master mode include bclk and frame sync.

kSAI_Slave Slave mode include bclk and frame sync.

kSAI_Bclk_Master_FrameSync_Slave bclk in master mode, frame sync in slave mode

kSAI_Bclk_Slave_FrameSync_Master bclk in slave mode, frame sync in master mode

36.4.4.5 enum sai_mono_stereo_t

Enumerator

kSAI_Stereo Stereo sound.

kSAI_MonoRight Only Right channel have sound.

kSAI_MonoLeft Only left channel have sound.

36.4.4.6 enum sai_data_order_t

Enumerator

kSAI_DataLSB LSB bit transferred first.

kSAI_DataMSB MSB bit transferred first.

36.4.4.7 enum sai_clock_polarity_t

Enumerator

kSAI_PolarityActiveHigh Drive outputs on rising edge.

kSAI_PolarityActiveLow Drive outputs on falling edge.

kSAI_SampleOnFallingEdge Sample inputs on falling edge.

kSAI_SampleOnRisingEdge Sample inputs on rising edge.

36.4.4.8 enum sai_sync_mode_t

Enumerator

kSAI_ModeAsync Asynchronous mode.

kSAI_ModeSync Synchronous mode (with receiver or transmit)

36.4.4.9 enum sai_bclk_source_t

Enumerator

kSAI_BclkSourceBusclk Bit clock using bus clock.
kSAI_BclkSourceMclkOption1 Bit clock MCLK option 1.
kSAI_BclkSourceMclkOption2 Bit clock MCLK option2.
kSAI_BclkSourceMclkOption3 Bit clock MCLK option3.
kSAI_BclkSourceMclkDiv Bit clock using master clock divider.
kSAI_BclkSourceOtherSai0 Bit clock from other SAI device.
kSAI_BclkSourceOtherSai1 Bit clock from other SAI device.

36.4.4.10 anonymous enum

Enumerator

kSAI_WordStartInterruptEnable Word start flag, means the first word in a frame detected.
kSAI_SyncErrorInterruptEnable Sync error flag, means the sync error is detected.
kSAI_FIFOWarningInterruptEnable FIFO warning flag, means the FIFO is empty.
kSAI_FIFOErrorInterruptEnable FIFO error flag.
kSAI_FIFORequestInterruptEnable FIFO request, means reached watermark.

36.4.4.11 anonymous enum

Enumerator

kSAI_FIFOWarningDMAEnable FIFO warning caused by the DMA request.
kSAI_FIFORequestDMAEnable FIFO request caused by the DMA request.

36.4.4.12 anonymous enum

Enumerator

kSAI_WordStartFlag Word start flag, means the first word in a frame detected.
kSAI_SyncErrorFlag Sync error flag, means the sync error is detected.
kSAI_FIFOErrorFlag FIFO error flag.
kSAI_FIFORequestFlag FIFO request flag.
kSAI_FIFOWarningFlag FIFO warning flag.

36.4.4.13 enum sai_reset_type_t

Enumerator

kSAI_ResetTypeSoftware Software reset, reset the logic state.
kSAI_ResetTypeFIFO FIFO reset, reset the FIFO read and write pointer.

SAI Driver

kSAI_ResetAll All reset.

36.4.4.14 enum sai_fifo_packing_t

Enumerator

kSAI_FifoPackingDisabled Packing disabled.
kSAI_FifoPacking8bit 8 bit packing enabled
kSAI_FifoPacking16bit 16bit packing enabled

36.4.4.15 enum sai_sample_rate_t

Enumerator

kSAI_SampleRate8KHz Sample rate 8000 Hz.
kSAI_SampleRate11025Hz Sample rate 11025 Hz.
kSAI_SampleRate12KHz Sample rate 12000 Hz.
kSAI_SampleRate16KHz Sample rate 16000 Hz.
kSAI_SampleRate22050Hz Sample rate 22050 Hz.
kSAI_SampleRate24KHz Sample rate 24000 Hz.
kSAI_SampleRate32KHz Sample rate 32000 Hz.
kSAI_SampleRate44100Hz Sample rate 44100 Hz.
kSAI_SampleRate48KHz Sample rate 48000 Hz.
kSAI_SampleRate96KHz Sample rate 96000 Hz.
kSAI_SampleRate192KHz Sample rate 192000 Hz.
kSAI_SampleRate384KHz Sample rate 384000 Hz.

36.4.4.16 enum sai_word_width_t

Enumerator

kSAI_WordWidth8bits Audio data width 8 bits.
kSAI_WordWidth16bits Audio data width 16 bits.
kSAI_WordWidth24bits Audio data width 24 bits.
kSAI_WordWidth32bits Audio data width 32 bits.

36.4.4.17 enum sai_transceiver_type_t

Enumerator

kSAI_Transmitter sai transmitter
kSAI_Receiver sai receiver

36.4.4.18 enum sai_frame_sync_len_t

Enumerator

kSAI_FrameSyncLenOneBitClk 1 bit clock frame sync len for DSP mode
kSAI_FrameSyncLenPerWordWidth Frame sync length decided by word width.

36.4.5 Function Documentation

36.4.5.1 void SAI_TxInit (I2S_Type * *base*, const sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_Init](#)

Ungates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI_TxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

Parameters

<i>base</i>	SAI base pointer
<i>config</i>	SAI configuration structure.

36.4.5.2 void SAI_RxInit (I2S_Type * *base*, const sai_config_t * *config*)

Deprecated Do not use this function. It has been superceded by [SAI_Init](#)

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI_RxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

SAI Driver

Parameters

<i>base</i>	SAI base pointer
<i>config</i>	SAI configuration structure.

36.4.5.3 void SAI_TxGetDefaultConfig (sai_config_t * *config*)

Deprecated Do not use this function. It has been superseded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeftJustifiedConfig](#) , [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

This API initializes the configuration structure for use in SAI_TxConfig(). The initialized structure can remain unchanged in SAI_TxConfig(), or it can be modified before calling SAI_TxConfig(). This is an example.

```
sai_config_t config;  
SAI_TxGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

36.4.5.4 void SAI_RxGetDefaultConfig (sai_config_t * *config*)

Deprecated Do not use this function. It has been superseded by [SAI_GetClassicI2SConfig](#), [SAI_GetLeftJustifiedConfig](#) , [SAI_GetRightJustifiedConfig](#), [SAI_GetDSPConfig](#), [SAI_GetTDMConfig](#)

This API initializes the configuration structure for use in SAI_RxConfig(). The initialized structure can remain unchanged in SAI_RxConfig() or it can be modified before calling SAI_RxConfig(). This is an example.

```
sai_config_t config;  
SAI_RxGetDefaultConfig(&config);
```

Parameters

<i>config</i>	pointer to master configuration structure
---------------	---

36.4.5.5 void SAI_Init (I2S_Type * *base*)

This API gates the SAI clock. The SAI module can't operate unless SAI_Init is called to enable the clock.

Parameters

<i>base</i>	SAI base pointer.
-------------	-------------------

36.4.5.6 void SAI_Deinit (I2S_Type * *base*)

This API gates the SAI clock. The SAI module can't operate unless SAI_TxInit or SAI_RxInit is called to enable the clock.

Parameters

<i>base</i>	SAI base pointer.
-------------	-------------------

36.4.5.7 void SAI_TxReset (I2S_Type * *base*)

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

36.4.5.8 void SAI_RxReset (I2S_Type * *base*)

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

36.4.5.9 void SAI_TxEnable (I2S_Type * *base*, bool *enable*)

Parameters

<i>base</i>	SAI base pointer.
<i>enable</i>	True means enable SAI Tx, false means disable.

36.4.5.10 void SAI_RxEnable (I2S_Type * *base*, bool *enable*)

SAI Driver

Parameters

<i>base</i>	SAI base pointer.
<i>enable</i>	True means enable SAI Rx, false means disable.

36.4.5.11 static void SAI_TxSetBitClockDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select bit clock direction, master or slave.

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	reference sai_master_slave_t.

36.4.5.12 static void SAI_RxSetBitClockDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select bit clock direction, master or slave.

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	reference sai_master_slave_t.

36.4.5.13 static void SAI_RxSetFrameSyncDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select frame sync direction, master or slave.

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	reference sai_master_slave_t.

36.4.5.14 static void SAI_TxSetFrameSyncDirection (I2S_Type * *base*, sai_master_slave_t *masterSlave*) [inline], [static]

Select frame sync direction, master or slave.

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	reference sai_master_slave_t.

36.4.5.15 void SAI_TxSetBitClockRate (I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers*)

Parameters

<i>base</i>	SAI base pointer.
<i>sourceClockHz</i>	Bit clock source frequency.
<i>sampleRate</i>	Audio data sample rate.
<i>bitWidth</i>	Audio data bitWidth.
<i>channel-Numbers</i>	Audio channel numbers.

36.4.5.16 void SAI_RxSetBitClockRate (I2S_Type * *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers*)

Parameters

<i>base</i>	SAI base pointer.
<i>sourceClockHz</i>	Bit clock source frequency.
<i>sampleRate</i>	Audio data sample rate.
<i>bitWidth</i>	Audio data bitWidth.
<i>channel-Numbers</i>	Audio channel numbers.

36.4.5.17 void SAI_TxSetBitclockConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config*)

SAI Driver

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	bit clock other configurations, can be NULL in slave mode.

36.4.5.18 void SAI_RxSetBitclockConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_bit_clock_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	bit clock other configurations, can be NULL in slave mode.

36.4.5.19 void SAI_TxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	fifo configurations.

36.4.5.20 void SAI_RxSetFifoConfig (I2S_Type * *base*, sai_fifo_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	fifo configurations.

36.4.5.21 void SAI_TxSetFrameSyncConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	frame sync configurations, can be NULL in slave mode.

36.4.5.22 void SAI_RxSetFrameSyncConfig (I2S_Type * *base*, sai_master_slave_t *masterSlave*, sai_frame_sync_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>masterSlave</i>	master or slave.
<i>config</i>	frame sync configurations, can be NULL in slave mode.

36.4.5.23 void SAI_TxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	serial data configurations.

36.4.5.24 void SAI_RxSetSerialDataConfig (I2S_Type * *base*, sai_serial_data_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	serial data configurations.

36.4.5.25 void SAI_TxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

SAI Driver

<i>base</i>	SAI base pointer.
<i>config</i>	transmitter configurations.

36.4.5.26 void SAI_RxSetConfig (I2S_Type * *base*, sai_transceiver_t * *config*)

Parameters

<i>base</i>	SAI base pointer.
<i>config</i>	receiver configurations.

36.4.5.27 void SAI_GetClassicI2SConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

<i>config</i>	transceiver configurations.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannel-Mask</i>	mask value of the channel to be enable.

36.4.5.28 void SAI_GetLeftJustifiedConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

<i>config</i>	transceiver configurations.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannel-Mask</i>	mask value of the channel to be enable.

36.4.5.29 void SAI_GetRightJustifiedConfig (sai_transceiver_t * *config*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

<i>config</i>	transceiver configurations.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.
<i>saiChannel-Mask</i>	mask value of the channel to be enable.

36.4.5.30 void SAI_GetTDMConfig (sai_transceiver_t * *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, uint32_t *dataWordNum*, uint32_t *saiChannelMask*)

Parameters

<i>config</i>	transceiver configurations.
<i>frameSync-Width</i>	length of frame sync.
<i>bitWidth</i>	audio data word width.
<i>dataWordNum</i>	word number in one frame.
<i>saiChannel-Mask</i>	mask value of the channel to be enable.

36.4.5.31 void SAI_GetDSPConfig (sai_transceiver_t * *config*, sai_frame_sync_len_t *frameSyncWidth*, sai_word_width_t *bitWidth*, sai_mono_stereo_t *mode*, uint32_t *saiChannelMask*)

Parameters

<i>config</i>	transceiver configurations.
<i>frameSync-Width</i>	length of frame sync.
<i>bitWidth</i>	audio data bitWidth.
<i>mode</i>	audio data channel.

SAI Driver

<i>saiChannel-Mask</i>	mask value of the channel to enable.
------------------------	--------------------------------------

36.4.5.32 `static uint32_t SAI_TxGetStatusFlag (I2S_Type * base) [inline], [static]`

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

36.4.5.33 `static void SAI_TxClearStatusFlags (I2S_Type * base, uint32_t mask) [inline], [static]`

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	State mask. It can be a combination of the following source if defined: <ul style="list-style-type: none">• kSAI_WordStartFlag• kSAI_SyncErrorFlag• kSAI_FIFOErrorFlag

36.4.5.34 `static uint32_t SAI_RxGetStatusFlag (I2S_Type * base) [inline], [static]`

Parameters

<i>base</i>	SAI base pointer
-------------	------------------

Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

36.4.5.35 `static void SAI_RxClearStatusFlags (I2S_Type * base, uint32_t mask) [inline], [static]`

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	State mask. It can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_WordStartFlag • kSAI_SyncErrorFlag • kSAI_FIFOErrorFlag

36.4.5.36 void SAI_TxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *type*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

<i>base</i>	SAI base pointer
<i>type</i>	Reset type, FIFO reset or software reset

36.4.5.37 void SAI_RxSoftwareReset (I2S_Type * *base*, sai_reset_type_t *type*)

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

<i>base</i>	SAI base pointer
<i>type</i>	Reset type, FIFO reset or software reset

36.4.5.38 void SAI_TxSetChannelFIFOMask (I2S_Type * *base*, uint8_t *mask*)

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

36.4.5.39 void SAI_RxSetChannelFIFOMask (I2S_Type * *base*, uint8_t *mask*)

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	Channel enable mask, 0 means all channel FIFO disabled, 1 means channel 0 enabled, 3 means both channel 0 and channel 1 enabled.

36.4.5.40 void SAI_TxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

36.4.5.41 void SAI_RxSetDataOrder (I2S_Type * *base*, sai_data_order_t *order*)

Parameters

<i>base</i>	SAI base pointer
<i>order</i>	Data order MSB or LSB

36.4.5.42 void SAI_TxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

36.4.5.43 void SAI_RxSetBitClockPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

36.4.5.44 void SAI_TxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

36.4.5.45 void SAI_RxSetFrameSyncPolarity (I2S_Type * *base*, sai_clock_polarity_t *polarity*)

Parameters

<i>base</i>	SAI base pointer
<i>polarity</i>	

36.4.5.46 void SAI_TxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

<i>base</i>	SAI base pointer.
<i>pack</i>	FIFO pack type. It is element of sai_fifo_packing_t.

36.4.5.47 void SAI_RxSetFIFOPacking (I2S_Type * *base*, sai_fifo_packing_t *pack*)

Parameters

<i>base</i>	SAI base pointer.
<i>pack</i>	FIFO pack type. It is element of sai_fifo_packing_t.

**36.4.5.48 static void SAI_TxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*)
[inline], [static]**

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in TCSR register.

SAI Driver

Parameters

<i>base</i>	SAI base pointer.
<i>isEnabled</i>	Is FIFO error continue enabled, true means enable, false means disable.

36.4.5.49 static void SAI_RxSetFIFOErrorContinue (I2S_Type * *base*, bool *isEnabled*) [inline], [static]

FIFO error continue mode means SAI will keep running while FIFO error occurred. If this feature not enabled, SAI will hang and users need to clear FEF flag in RCSR register.

Parameters

<i>base</i>	SAI base pointer.
<i>isEnabled</i>	Is FIFO error continue enabled, true means enable, false means disable.

36.4.5.50 static void SAI_TxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none">• kSAI_WordStartInterruptEnable• kSAI_SyncErrorInterruptEnable• kSAI_FIFOWarningInterruptEnable• kSAI_FIFORequestInterruptEnable• kSAI_FIFOErrorInterruptEnable

36.4.5.51 static void SAI_RxEnableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

36.4.5.52 static void SAI_TxDisableInterrupts (I2S_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kSAI_WordStartInterruptEnable • kSAI_SyncErrorInterruptEnable • kSAI_FIFOWarningInterruptEnable • kSAI_FIFORequestInterruptEnable • kSAI_FIFOErrorInterruptEnable

SAI Driver

36.4.5.53 static void SAI_RxDisableInterrupts (I2S_Type * *base*, uint32_t *mask*)
[inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none">• kSAI_WordStartInterruptEnable• kSAI_SyncErrorInterruptEnable• kSAI_FIFOWarningInterruptEnable• kSAI_FIFOREquestInterruptEnable• kSAI_FIFOErrorInterruptEnable

36.4.5.54 static void SAI_TxEnableDMA (I2S_Type * *base*, uint32_t *mask*, bool *enable*)
[inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	DMA source The parameter can be combination of the following sources if defined. <ul style="list-style-type: none">• kSAI_FIFOWarningDMAEnable• kSAI_FIFOREquestDMAEnable
<i>enable</i>	True means enable DMA, false means disable DMA.

36.4.5.55 static void SAI_RxEnableDMA (I2S_Type * *base*, uint32_t *mask*, bool *enable*)
[inline], [static]

Parameters

<i>base</i>	SAI base pointer
<i>mask</i>	DMA source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none">• kSAI_FIFOWarningDMAEnable• kSAI_FIFOREquestDMAEnable

<i>enable</i>	True means enable DMA, false means disable DMA.
---------------	---

36.4.5.56 **static uint32_t SAI_TxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]**

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

36.4.5.57 **static uint32_t SAI_RxGetDataRegisterAddress (I2S_Type * *base*, uint32_t *channel*) [inline], [static]**

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Which data channel used.

Returns

data register address.

36.4.5.58 **void SAI_TxSetFormat (I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)**

Deprecated Do not use this function. It has been superceded by [SAI_TxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

SAI Driver

Parameters

<i>base</i>	SAI base pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

36.4.5.59 void SAI_RxSetFormat (I2S_Type * *base*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

Deprecated Do not use this function. It has been superseded by [SAI_RxSetConfig](#)

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>	SAI base pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz.

36.4.5.60 void SAI_WriteBlocking (I2S_Type * *base*, uint32_t *channel*, 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>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

36.4.5.61 void SAI_WriteMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, 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>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be written.
<i>size</i>	Bytes to be written.

**36.4.5.62 static void SAI_WriteData (I2S_Type * *base*, uint32_t *channel*, uint32_t *data*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>data</i>	Data needs to be written.

36.4.5.63 void SAI_ReadBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *bitWidth*, uint8_t * *buffer*, uint32_t *size*)

Note

This function blocks by polling until data is ready to be sent.

SAI Driver

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

36.4.5.64 void SAI_ReadMultiChannelBlocking (I2S_Type * *base*, uint32_t *channel*, uint32_t *channelMask*, 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>	SAI base pointer.
<i>channel</i>	Data channel used.
<i>channelMask</i>	channel mask.
<i>bitWidth</i>	How many bits in an audio word; usually 8/16/24/32 bits.
<i>buffer</i>	Pointer to the data to be read.
<i>size</i>	Bytes to be read.

**36.4.5.65 static uint32_t SAI_ReadData (I2S_Type * *base*, uint32_t *channel*)
[inline], [static]**

Parameters

<i>base</i>	SAI base pointer.
<i>channel</i>	Data channel used.

Returns

Data in SAI FIFO.

36.4.5.66 void SAI_TransferTxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*,
sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Tx handle for the SAI Tx transactional APIs. Call this function once to get the handle initialized.

SAI Driver

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function

36.4.5.67 void SAI_TransferRxCreateHandle (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_callback_t *callback*, void * *userData*)

This function initializes the Rx handle for the SAI Rx transactional APIs. Call this function once to get the handle initialized.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>callback</i>	Pointer to the user callback function.
<i>userData</i>	User parameter passed to the callback function.

36.4.5.68 void SAI_TransferTxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Tx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>config</i>	transmitter configurations.

36.4.5.69 void SAI_TransferRxSetConfig (I2S_Type * *base*, sai_handle_t * *handle*, sai_transceiver_t * *config*)

This function initializes the Rx, include bit clock, frame sync, master clock, serial data and fifo configurations.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>config</i>	receiver configurations.

36.4.5.70 `status_t SAI_TransferTxSetFormat (I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)`

Deprecated Do not use this function. It has been superceded by [SAI_TransferTxSetConfig](#)

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>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format.

Returns

Status of this function. Return value is the status_t.

36.4.5.71 `status_t SAI_TransferRxSetFormat (I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)`

Deprecated Do not use this function. It has been superceded by [SAI_TransferRxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

SAI Driver

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI handle pointer.
<i>format</i>	Pointer to the SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format.

Returns

Status of this function. Return value is one of status_t.

36.4.5.72 status_t SAI_TransferSendNonBlocking (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the sai_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SAI_TxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

36.4.5.73 status_t SAI_TransferReceiveNonBlocking (I2S_Type * *base*, sai_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This API returns immediately after the transfer initiates. Call the SAI_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus_SAI_Busy, the transfer is finished.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the sai_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully started the data receive.
<i>kStatus_SAI_RxBusy</i>	Previous receive still not finished.
<i>kStatus_InvalidArgument</i>	The input parameter is invalid.

36.4.5.74 status_t SAI_TransferGetSendCount (I2S_Type * *base*, sai_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_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.

36.4.5.75 status_t SAI_TransferGetReceiveCount (I2S_Type * *base*, sai_handle_t * *handle*, size_t * *count*)

SAI Driver

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_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.

36.4.5.76 void SAI_TransferAbortSend (I2S_Type * *base*, sai_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>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.

36.4.5.77 void SAI_TransferAbortReceive (I2S_Type * *base*, sai_handle_t * *handle*)

Note

This API can be called when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	Pointer to the sai_handle_t structure which stores the transfer state.

36.4.5.78 void SAI_TransferTerminateSend (I2S_Type * *base*, sai_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortSend.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

36.4.5.79 void SAI_TransferTerminateReceive (I2S_Type * *base*, sai_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call SAI_TransferAbortReceive.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

36.4.5.80 void SAI_TransferTxHandleIRQ (I2S_Type * *base*, sai_handle_t * *handle*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure.

36.4.5.81 void SAI_TransferRxHandleIRQ (I2S_Type * *base*, sai_handle_t * *handle*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	Pointer to the sai_handle_t structure.

SAI EDMA Driver

36.5.1 Overview

Data Structures

- struct [sai_edma_handle_t](#)
SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef void(* [sai_edma_callback_t](#))(I2S_Type *base, sai_edma_handle_t *handle, [status_t](#) status, void *userData)
SAI eDMA transfer callback function for finish and error.

Driver version

- #define [FSL_SAI_EDMA_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 3, 1))
Version 2.3.1.

eDMA Transactional

- void [SAI_TransferTxCreateHandleEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_edma_callback_t](#) callback, void *userData, [edma_handle_t](#) *txDmaHandle)
Initializes the SAI eDMA handle.
- void [SAI_TransferRxCreateHandleEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_edma_callback_t](#) callback, void *userData, [edma_handle_t](#) *rxDmaHandle)
Initializes the SAI Rx eDMA handle.
- void [SAI_TransferTxSetFormatEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_transfer_format_t](#) *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Tx audio format.
- void [SAI_TransferRxSetFormatEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_transfer_format_t](#) *format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz)
Configures the SAI Rx audio format.
- void [SAI_TransferTxSetConfigEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_transceiver_t](#) *saiConfig)
Configures the SAI Tx.
- void [SAI_TransferRxSetConfigEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_transceiver_t](#) *saiConfig)
Configures the SAI Rx.
- [status_t](#) [SAI_TransferSendEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_transfer_t](#) *xfer)
Performs a non-blocking SAI transfer using DMA.
- [status_t](#) [SAI_TransferReceiveEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, [sai_transfer_t](#) *xfer)

- *Performs a non-blocking SAI receive using eDMA.*
- void [SAI_TransferTerminateSendEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
Terminate all SAI send.
- void [SAI_TransferTerminateReceiveEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
Terminate all SAI receive.
- void [SAI_TransferAbortSendEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
Aborts a SAI transfer using eDMA.
- void [SAI_TransferAbortReceiveEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle)
Aborts a SAI receive using eDMA.
- [status_t SAI_TransferGetSendCountEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
Gets byte count sent by SAI.
- [status_t SAI_TransferGetReceiveCountEDMA](#) (I2S_Type *base, sai_edma_handle_t *handle, size_t *count)
Gets byte count received by SAI.

36.5.2 Data Structure Documentation

36.5.2.1 struct sai_edma_handle

Data Fields

- [edma_handle_t](#) * [dmaHandle](#)
DMA handler for SAI send.
- uint8_t [nbytes](#)
eDMA minor byte transfer count initially configured.
- uint8_t [bytesPerFrame](#)
Bytes in a frame.
- uint8_t [channel](#)
Which data channel.
- uint8_t [count](#)
The transfer data count in a DMA request.
- uint32_t [state](#)
Internal state for SAI eDMA transfer.
- [sai_edma_callback_t](#) [callback](#)
Callback for users while transfer finish or error occurs.
- void * [userData](#)
User callback parameter.
- uint8_t [tcd](#) [(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]
TCD pool for eDMA transfer.
- [sai_transfer_t](#) [saiQueue](#) [SAI_XFER_QUEUE_SIZE]
Transfer queue storing queued transfer.
- size_t [transferSize](#) [SAI_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.

SAI EDMA Driver

36.5.2.1.0.95 Field Documentation

36.5.2.1.0.95.1 `uint8_t sai_edma_handle_t::nbytes`

36.5.2.1.0.95.2 `uint8_t sai_edma_handle_t::tcd[(SAI_XFER_QUEUE_SIZE+1U)*sizeof(edma_tcd_t)]`

36.5.2.1.0.95.3 `sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`

36.5.2.1.0.95.4 `volatile uint8_t sai_edma_handle_t::queueUser`

36.5.3 Function Documentation

36.5.3.1 `void SAI_TransferTxCreateHandleEDMA (I2S_Type * base, sai_edma_handle_t * handle, sai_edma_callback_t callback, void * userData, edma_handle_t * txDmaHandle)`

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>base</i>	SAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>txDmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

36.5.3.2 void SAI_TransferRxCreateHandleEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_edma_callback_t *callback*, void * *userData*, edma_handle_t * *rxDmaHandle*)

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>base</i>	SAI peripheral base address.
<i>callback</i>	Pointer to user callback function.
<i>userData</i>	User parameter passed to the callback function.
<i>rxDmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

36.5.3.3 void SAI_TransferTxSetFormatEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

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>	SAI base pointer.
-------------	-------------------

SAI EDMA Driver

<i>handle</i>	SAI eDMA handle pointer.
<i>format</i>	Pointer to SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

36.5.3.4 void SAI_TransferRxSetFormatEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_format_t * *format*, uint32_t *mclkSourceClockHz*, uint32_t *bclkSourceClockHz*)

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>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>format</i>	Pointer to SAI audio data format structure.
<i>mclkSource-ClockHz</i>	SAI master clock source frequency in Hz.
<i>bclkSource-ClockHz</i>	SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format.

Return values

<i>kStatus_Success</i>	Audio format set successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.

36.5.3.5 void SAI_TransferTxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

36.5.3.6 void SAI_TransferRxSetConfigEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transceiver_t * *saiConfig*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>saiConfig</i>	sai configurations.

36.5.3.7 status_t SAI_TransferSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*)

Note

This interface returns immediately after the transfer initiates. Call SAI_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to the DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA send successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_TxBusy</i>	SAI is busy sending data.

36.5.3.8 status_t SAI_TransferReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, sai_transfer_t * *xfer*)

SAI EDMA Driver

Note

This interface returns immediately after the transfer initiates. Call the `SAI_GetReceiveRemainingBytes` to poll the transfer status and check whether the SAI transfer is finished.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a SAI eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_RxBusy</i>	SAI is busy receiving data.

36.5.3.9 void SAI_TransferTerminateSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call `SAI_TransferAbortSendEDMA`.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

36.5.3.10 void SAI_TransferTerminateReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the sai queue. If users only want to abort the current transfer slot, please call `SAI_TransferAbortReceiveEDMA`.

Parameters

<i>base</i>	SAI base pointer.
-------------	-------------------

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

36.5.3.11 void SAI_TransferAbortSendEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateSendEDMA.

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.

36.5.3.12 void SAI_TransferAbortReceiveEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*)

This function only aborts the current transfer slots, the other transfer slots' information still kept in the handler. If users want to terminate all transfer slots, just call SAI_TransferTerminateReceiveEDMA.

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.

36.5.3.13 status_t SAI_TransferGetSendCountEDMA (I2S_Type * *base*, sai_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	SAI base pointer.
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count sent by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

36.5.3.14 `status_t SAI_TransferGetReceiveCountEDMA (I2S_Type * base,
sai_edma_handle_t * handle, size_t * count)`

Parameters

<i>base</i>	SAI base pointer
<i>handle</i>	SAI eDMA handle pointer.
<i>count</i>	Bytes count received by SAI.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

Chapter 37

SEMA42: Hardware Semaphores Driver

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.

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) (((volatile uint8_t *)(&((base)->GATE3)))[(n) ^ 3U])
SEMA42 gate n register address.

Enumerations

- enum {
 [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, 2))
SEMA42 driver version.

Macro Definition Documentation

37.3.1 #define SEMA42_GATE_NUM_RESET_ALL (64U)

37.3.2 #define SEMA42_GATEn(base, n) (((volatile uint8_t *)(&((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: $0 \wedge 3 = 3$ $1 \wedge 3 = 2$ $2 \wedge 3 = 1$ $3 \wedge 3 = 0$ $4 \wedge 3 = 7$ $5 \wedge 3 = 6$ $6 \wedge 3 = 5$ $7 \wedge 3 = 4$...

Enumeration Type Documentation

37.4.1 anonymous enum

Enumerator

kStatus_SEMA42_Busy SEMA42 gate has been locked by other processor.

kStatus_SEMA42_Reseting SEMA42 gate reseting is ongoing.

37.4.2 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

Function Documentation

37.5.1 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.
-------------	---------------------------------

37.5.2 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.
-------------	---------------------------------

37.5.3 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.

37.5.4 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.

37.5.5 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.

37.5.6 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.

37.5.7 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.

37.5.8 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.

Chapter 38

TPM: Timer PWM Module

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.

Introduction of TPM

38.2.1 Initialization and deinitialization

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.

38.2.2 PWM Operations

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.

Introduction of TPM

38.2.3 Input capture operations

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.

38.2.4 Output compare operations

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.

38.2.5 Quad decode

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 input signal.

38.2.6 Fault operation

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.

38.2.7 Status

Provides functions to get and clear the TPM status.

38.2.8 Interrupt

Provides functions to enable/disable TPM interrupts and get current enabled interrupts.

Typical use case

38.3.1 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.

Typical use case

- enum `tpm_output_compare_mode_t` {
 `kTPM_NoOutputSignal` = (1U << TPM_CnSC_MSA_SHIFT),
 `kTPM_ToggleOnMatch` = ((1U << TPM_CnSC_MSA_SHIFT) | (1U << TPM_CnSC_ELSA_SHIFT)),
 `kTPM_ClearOnMatch` = ((1U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_SHIFT)),
 `kTPM_SetOnMatch` = ((1U << TPM_CnSC_MSA_SHIFT) | (3U << TPM_CnSC_ELSA_SHIFT)),
 `kTPM_HighPulseOutput` = ((3U << TPM_CnSC_MSA_SHIFT) | (1U << TPM_CnSC_ELSA_SHIFT)),
 `kTPM_LowPulseOutput` = ((3U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_SHIFT)) }
 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` }
 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, 7))
Version 2.0.7.

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)

Typical use case

- *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

38.4.1 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.

38.4.1.0.0.96 Field Documentation

38.4.1.0.0.96.1 tpm_chnl_t tpm_chnl_pwm_signal_param_t::chnlNumber

In combined mode (available in some SoC's, this represents the channel pair number

38.4.1.0.0.96.2 uint8_t tpm_chnl_pwm_signal_param_t::dutyCyclePercent

100=always active signal (100% duty cycle)

38.4.1.0.0.96.3 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

38.4.2 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.

Enumeration Type Documentation

38.4.3 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.

38.4.4 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

38.4.4.0.0.97 Field Documentation

38.4.4.0.0.97.1 tpm_trigger_source_t tpm_config_t::triggerSource

Enumeration Type Documentation

38.5.1 enum tpm_chnl_t

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.

38.5.2 enum tpm_pwm_mode_t

Enumerator

kTPM_EdgeAlignedPwm Edge aligned PWM.
kTPM_CenterAlignedPwm Center aligned PWM.
kTPM_CombinedPwm Combined PWM.

38.5.3 enum tpm_pwm_level_select_t

Enumerator

kTPM_NoPwmSignal No PWM output on pin.
kTPM_LowTrue Low true pulses.
kTPM_HighTrue High true pulses.

38.5.4 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.

Enumeration Type Documentation

38.5.5 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.

38.5.6 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.

38.5.7 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.

38.5.8 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.

38.5.9 enum tpm_phase_polarity_t

Enumerator

kTPM_QuadPhaseNormal Phase input signal is not inverted.

kTPM_QuadPhaseInvert Phase input signal is inverted.

38.5.10 enum tpm_clock_source_t

Enumerator

kTPM_SystemClock System clock.

kTPM_ExternalClock External clock.

38.5.11 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.

38.5.12 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.

Function Documentation

38.5.13 enum tpm_status_flags_t

Enumerator

kTPM_Chnl0Flag Channel 0 flag.
kTPM_Chnl1Flag Channel 1 flag.
kTPM_Chnl2Flag Channel 2 flag.
kTPM_Chnl3Flag Channel 3 flag.
kTPM_Chnl4Flag Channel 4 flag.
kTPM_Chnl5Flag Channel 5 flag.
kTPM_Chnl6Flag Channel 6 flag.
kTPM_Chnl7Flag Channel 7 flag.
kTPM_TimeOverflowFlag Time overflow flag.

Function Documentation

38.6.1 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.

38.6.2 void TPM_Deinit (TPM_Type * *base*)

Parameters

<i>base</i>	TPM peripheral base address
-------------	-----------------------------

38.6.3 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;
```

```

*#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.
---------------	---

38.6.4 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 tpm_pwm_mode_t
<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

38.6.5 void TPM_UpdatePwmDutycycle (TPM_Type * *base*, tpm_chnl_t *chnlNumber*, tpm_pwm_mode_t *currentPwmMode*, uint8_t *dutyCyclePercent*)

Function Documentation

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)

38.6.6 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.

38.6.7 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

38.6.8 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.

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.

38.6.9 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.

38.6.10 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

38.6.11 void TPM_EnableInterrupts (TPM_Type * *base*, uint32_t *mask*)

Function Documentation

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 tpm_interrupt_enable_t

38.6.12 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 tpm_interrupt_enable_t

38.6.13 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](#)

38.6.14 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](#)

38.6.15 static void TPM_ClearStatusFlags (TPM_Type * *base*, uint32_t *mask*) [inline], [static]

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 tpm_status_flags_t

38.6.16 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.

38.6.17 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

38.6.18 `static void TPM_StartTimer (TPM_Type * base, tpm_clock_source_t clockSource) [inline], [static]`

Parameters

<i>base</i>	TPM peripheral base address
<i>clockSource</i>	TPM clock source; once clock source is set the counter will start running

38.6.19 static void TPM_StopTimer (TPM_Type * *base*) [inline], [static]

Parameters

<i>base</i>	TPM peripheral base address
-------------	-----------------------------

38.6.20 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 39

TSTMR: Timestamp Timer Driver

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.

Function Documentation

39.2.1 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.

39.2.2 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 40

WDOG32: 32-bit Watchdog Timer

Overview

The MCUXpresso SDK provides a peripheral driver for the WDOG32 module of MCUXpresso SDK devices.

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) & 0xFFFFU)
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) & 0xFFFFU)
Second word of refresh sequence.

Driver version

- #define [FSL_WDOG32_DRIVER_VERSION](#) (MAKE_VERSION(2, 0, 3))
WDOG32 driver version.

WDOG32 Initialization and De-initialization

- void [WDOG32_GetDefaultConfig](#) (wdog32_config_t *config)
Initializes the WDOG32 configuration structure.
- 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.
- 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.

Data Structure Documentation

40.3.1 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.

40.3.2 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.

Macro Definition Documentation

40.4.1 #define FSL_WDOG32_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

Enumeration Type Documentation

Enumeration Type Documentation

40.5.1 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.

40.5.2 enum wdog32_clock_prescaler_t

Enumerator

kWDOG32_ClockPrescalerDivide1 Divided by 1.
kWDOG32_ClockPrescalerDivide256 Divided by 256.

40.5.3 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.

40.5.4 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.

40.5.5 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.

Function Documentation

40.6.1 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](#)

40.6.2 void WDOG32_Init (WDOG_Type * *base*, const wdog32_config_t * *config*)

This function initializes the WDOG32. To reconfigure the WDOG32 without forcing a reset first, enable-Update 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);
*
```

Parameters

Function Documentation

<i>base</i>	WDOG32 peripheral base address.
<i>config</i>	The configuration of the WDOG32.

40.6.3 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.
-------------	---------------------------------

40.6.4 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.
-------------	---------------------------------

40.6.5 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
-------------	--------------------------------

40.6.6 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.

Function Documentation

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">• kWDOG32_InterruptEnable

40.6.7 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">• kWDOG32_InterruptEnable

40.6.8 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;  
*
```

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.

40.6.9 void WDOG32_ClearStatusFlags (WDOG_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"> • kWDOG32_InterruptFlag

40.6.10 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.

Function Documentation

Parameters

<i>base</i>	WDOG32 peripheral base address
<i>timeoutCount</i>	WDOG32 timeout value, count of WDOG32 clock ticks.

40.6.11 **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.

40.6.12 **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
-------------	--------------------------------

40.6.13 **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.

Parameters

<i>base</i>	WDOG32 peripheral base address
-------------	--------------------------------

**40.6.14 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 41

Debug Console

Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data. The below picture shows the layout of debug console.

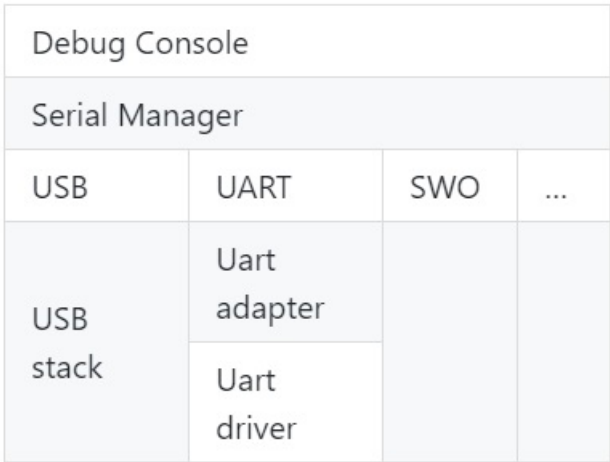


Figure 41.1.1: Debug console overview

Function groups

41.2.1 Initialization

To initialize the debug console, call the [DbgConsole_Init\(\)](#) function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate,  
    serial_port_type_t device, uint32_t clkSrcFreq);
```

Select the supported debug console hardware device type, such as

```
typedef enum _serial_port_type  
{  
    kSerialPort_Uart = 1U,
```

Function groups

```
kSerialPort_UsbCdc,  
kSerialPort_Swo,  
kSerialPort_UsbCdcVirtual,  
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral.

This example shows how to call the `DbgConsole_Init()` given the user configuration structure.

```
DbgConsole_Init (BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUDRATE, BOARD_DEBUG_UART_TYPE,  
BOARD_DEBUG_UART_CLK_FREQ);
```

41.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	Description
.number	For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

length	Description
Do not support	

specifier	Description
d or i	Signed decimal integer
f	Decimal floating point
F	Decimal floating point capital letters
x	Unsigned hexadecimal integer

Function groups

specifier	Description
X	Unsigned hexadecimal integer capital letters
o	Signed octal
b	Binary value
p	Pointer address
u	Unsigned decimal integer
c	Character
s	String of characters
n	Nothing printed

- Support a format specifier for SCANF following this prototype " %[*][width][length]specifier", which is explained below

*	Description
An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument.	

width	Description
This specifies the maximum number of characters to be read in the current reading operation.	

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).

length	Description
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
```

Typical use case

```
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
    version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
    toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */
```

NOTE: The macro `SDK_DEBUGCONSOLE_UART` is used to decide whether provide low level IO implementation to toolchain `printf` and `scanf`. For example, within `MCUXpresso`, if the macro `SDK_DEBUGCONSOLE_UART` is defined, **`sys_write` and `__sys_readc` will be used when `__REDLIB` is defined;** `_write` and `_read` will be used in other cases. If the macro `SDK_DEBUGCONSOLE_UART` is not defined, the semihosting will be used.

Typical use case

Some examples use the PUTCHAR & GETCHAR function

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

Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalent 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\r\nTime: %u ticks %2.5f milliseconds\r\n\r\nDONE\r\n", "1 day", 86400, 86.4);
```

Some examples use the SCANF function

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


Print out failure messages using MCUXpresso SDK `__assert_func`:

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

Note:

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

Modules

- [SWO](#)
- [Semihosting](#)

Macros

- #define [DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN](#) 0U
Definition select redirect toolchain printf, scanf to uart or not.
- #define [DEBUGCONSOLE_REDIRECT_TO_SDK](#) 1U
Select SDK version printf, scanf.
- #define [DEBUGCONSOLE_DISABLE](#) 2U
Disable debugconsole function.
- #define [SDK_DEBUGCONSOLE](#) 1U
Definition to select sdk or toolchain printf, scanf.
- #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.

Variables

- [serial_handle_t g_serialHandle](#)
serial manager handle

Macro Definition Documentation

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.
- int [DbgConsole_BlockingPrintf](#) (const char *formatString,...)
Writes formatted output to the standard output stream with the blocking mode.
- [status_t DbgConsole_Flush](#) (void)
Debug console flush.

Macro Definition Documentation

41.4.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

41.4.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

41.4.3 #define DEBUGCONSOLE_DISABLE 2U

41.4.4 #define SDK_DEBUGCONSOLE 1U

The macro only support to be redefined in project setting.

41.4.5 #define PRINTF DbgConsole_Printf

if SDK_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK_DEBUGCONSOLE defined to 1,it represents select SDK version printf, scanf. if SDK_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

Function Documentation

41.5.1 `status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)`

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

Parameters

<i>instance</i>	The instance of the module.
<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 • kSerialPort_UsbCdcVirtual.
<i>clkSrcFreq</i>	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

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

41.5.2 `status_t DbgConsole_Deinit (void)`

Call this function to disable debug log messages to be output via the specified peripheral initialized by the serial manager module.

Returns

Indicates whether de-initialization was successful or not.

41.5.3 `int DbgConsole_Printf (const char * formatString, ...)`

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

Function Documentation

Parameters

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

Returns

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

41.5.4 int DbgConsole_Putchar (int *ch*)

Call this function to write a character to stdout.

Parameters

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

Returns

Returns the character written.

41.5.5 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_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Parameters

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

Returns

Returns the number of fields successfully converted and assigned.

41.5.6 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_BLOCKING` is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function `DbgConsole_TryGetchar` to get the input char.

Returns

Returns the character read.

41.5.7 int DbgConsole_BlockingPrintf (const char * *formatString*, ...)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` set or not. The function could be used in system ISR mode with `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` set.

Parameters

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

Returns

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

41.5.8 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.

Function Documentation

41.5.9 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 callbck function pointer

Returns

Number of characters to be print

41.5.10 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.

Return values

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

Semihosting

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

41.6.1 Guide Semihosting for IAR

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

Step 1: Setting up the environment

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

Step 2: Building the project

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

Step 3: Starting semihosting

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

41.6.2 Guide Semihosting for Keil µVision

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

41.6.3 Guide Semihosting for MCUXpresso IDE

Step 1: Setting up the environment

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

Step 2: Building the project

1. Compile and link the project.

Step 3: Starting semihosting

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

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

41.6.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

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

Add to "CMakeLists.txt"

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

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

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

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


Step 2: Building the project

1. Change "CMakeLists.txt":

Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "\${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=nano.specs")"

to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "\${CMAKE_EXE_LINKER_FLAGS_RELEASE} -specs=rdimon.specs")"

Replace paragraph

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "\${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")

To

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

Remove

target_link_libraries(semihosting_ARMGCC.elf debug nosys)

2. Run "build_debug.bat" to build project

Semihosting

Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

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

SWO

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

41.7.1 Guide SWO for SDK

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

Step 1: Setting up the environment

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

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

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

Step 2: Building the project

Step 3: Download and run project

41.7.1.1 Guide SWO for IAR

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

Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK, and click the Activate button to enable trace data collection.
6. There are three cases for this SDK_DEBUGCONSOLE_UART whether or not defined. a: if use uppercase PRINTF to output log, The SDK_DEBUGCONSOLE_UART defined or not defined will not effect debug function. b: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to zero, then debug function ok. c: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to one, then debug function ok.

SWO

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

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

Step 2: Building the project

Step 3: Starting swo

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

41.7.2 Guide SWO for Keil µVision

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

Step 1: Setting up the environment

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

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

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

Step 3: Building the project

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

Step 4: Run the project

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

41.7.3 Guide SWO for MCUXpresso IDE

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

41.7.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 42

Notification Framework

Overview

This section describes the programming interface of the Notifier driver.

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.

Data Structure Documentation

42.3.1 struct `notifier_notification_block_t`

Data Fields

- `notifier_user_config_t * targetConfig`
Pointer to target configuration.
- `notifier_policy_t policy`
Configure transition policy.
- `notifier_notification_type_t notifyType`

Data Structure Documentation

Configure notification type.

42.3.1.0.0.98 Field Documentation

42.3.1.0.0.98.1 `notifier_user_config_t* notifier_notification_block_t::targetConfig`

42.3.1.0.0.98.2 `notifier_policy_t notifier_notification_block_t::policy`

42.3.1.0.0.98.3 `notifier_notification_type_t notifier_notification_block_t::notifyType`

42.3.2 struct `notifier_callback_config_t`

This structure holds the configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains the following application-defined data. `callback` - pointer to the callback function `callbackType` - specifies when the callback is called `callbackData` - pointer to the data passed to the callback.

Data Fields

- [notifier_callback_t callback](#)
Pointer to the callback function.
- [notifier_callback_type_t callbackType](#)
Callback type.
- `void * callbackData`
Pointer to the data passed to the callback.

42.3.2.0.0.99 Field Documentation

42.3.2.0.0.99.1 `notifier_callback_t notifier_callback_config_t::callback`

42.3.2.0.0.99.2 `notifier_callback_type_t notifier_callback_config_t::callbackType`

42.3.2.0.0.99.3 `void* notifier_callback_config_t::callbackData`

42.3.3 struct `notifier_handle_t`

Notifier handle structure. Contains data necessary for the Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data, and other internal data. [NOTIFIER_CreateHandle\(\)](#) must be called to initialize this handle.

Data Fields

- [notifier_user_config_t ** configsTable](#)
Pointer to configure table.
- `uint8_t configsNumber`
Number of configurations.

- [notifier_callback_config_t](#) * [callbacksTable](#)
Pointer to callback table.
- [uint8_t](#) [callbacksNumber](#)
Maximum number of callback configurations.
- [uint8_t](#) [errorCallbackIndex](#)
Index of callback returns error.
- [uint8_t](#) [currentConfigIndex](#)
Index of current configuration.
- [notifier_user_function_t](#) [userFunction](#)
User function.
- [void](#) * [userData](#)
User data passed to user function.

42.3.3.0.0.100 Field Documentation

42.3.3.0.0.100.1 [notifier_user_config_t](#)** [notifier_handle_t::configsTable](#)

42.3.3.0.0.100.2 [uint8_t](#) [notifier_handle_t::configsNumber](#)

42.3.3.0.0.100.3 [notifier_callback_config_t](#)* [notifier_handle_t::callbacksTable](#)

42.3.3.0.0.100.4 [uint8_t](#) [notifier_handle_t::callbacksNumber](#)

42.3.3.0.0.100.5 [uint8_t](#) [notifier_handle_t::errorCallbackIndex](#)

42.3.3.0.0.100.6 [uint8_t](#) [notifier_handle_t::currentConfigIndex](#)

42.3.3.0.0.100.7 [notifier_user_function_t](#) [notifier_handle_t::userFunction](#)

42.3.3.0.0.100.8 [void](#)* [notifier_handle_t::userData](#)

Typedef Documentation

42.4.1 [typedef void notifier_user_config_t](#)

Reference of the user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

42.4.2 [typedef status_t\(* notifier_user_function_t\)\(notifier_user_config_t *targetConfig, void *userData\)](#)

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, [NOTIFIER_SwitchConfig\(\)](#) exits.

Enumeration Type Documentation

Parameters

<i>targetConfig</i>	target Configuration.
<i>userData</i>	Refers to other specific data passed to user function.

Returns

An error code or `kStatus_Success`.

42.4.3 `typedef status_t(* notifier_callback_t)(notifier_notification_block_t *notify, void *data)`

Declaration of a callback. It is common for registered callbacks. Reference to function of this type is part of the `notifier_callback_config_t` callback configuration structure. Depending on callback type, function of this prototype is called (see `NOTIFIER_SwitchConfig()`) before configuration switch, after it or in both use cases to notify about the switch progress (see `notifier_callback_type_t`). When called, the type of the notification is passed as a parameter along with the reference to the target configuration structure (see `notifier_notification_block_t`) and any data passed during the callback registration. When notified before the configuration switch, depending on the configuration switch policy (see `notifier_policy_t`), the callback may deny the execution of the user function by returning an error code different than `kStatus_Success` (see `NOTIFIER_SwitchConfig()`).

Parameters

<i>notify</i>	Notification block.
<i>data</i>	Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

Returns

An error code or `kStatus_Success`.

Enumeration Type Documentation

42.5.1 `enum _notifier_status`

Used as return value of Notifier functions.

Enumerator

`kStatus_NOTIFIER_ErrorNotificationBefore` An error occurs during send "BEFORE" notification.

`kStatus_NOTIFIER_ErrorNotificationAfter` An error occurs during send "AFTER" notification.

42.5.2 enum notifier_policy_t

Defines whether the user function execution is forced or not. For `kNOTIFIER_PolicyForcible`, the user function is executed regardless of the callback results, while `kNOTIFIER_PolicyAgreement` policy is used to exit `NOTIFIER_SwitchConfig()` when any of the callbacks returns error code. See also `NOTIFIER_SwitchConfig()` description.

Enumerator

kNOTIFIER_PolicyAgreement `NOTIFIER_SwitchConfig()` method is exited when any of the callbacks returns error code.

kNOTIFIER_PolicyForcible The user function is executed regardless of the results.

42.5.3 enum notifier_notification_type_t

Used to notify registered callbacks

Enumerator

kNOTIFIER_NotifyRecover Notify IP to recover to previous work state.

kNOTIFIER_NotifyBefore Notify IP that configuration setting is going to change.

kNOTIFIER_NotifyAfter Notify IP that configuration setting has been changed.

42.5.4 enum notifier_callback_type_t

Used in the callback configuration structure (`notifier_callback_config_t`) to specify when the registered callback is called during configuration switch initiated by the `NOTIFIER_SwitchConfig()`. Callback can be invoked in following situations.

- Before the configuration switch (Callback return value can affect `NOTIFIER_SwitchConfig()` execution. See the `NOTIFIER_SwitchConfig()` and `notifier_policy_t` documentation).
- After an unsuccessful attempt to switch configuration
- After a successful configuration switch

Enumerator

kNOTIFIER_CallbackBefore Callback handles BEFORE notification.

kNOTIFIER_CallbackAfter Callback handles AFTER notification.

kNOTIFIER_CallbackBeforeAfter Callback handles BEFORE and AFTER notification.

Function Documentation

42.6.1 `status_t NOTIFIER_CreateHandle (notifier_handle_t * notifierHandle,
notifier_user_config_t ** configs, uint8_t configsNumber, notifier_callback-
_config_t * callbacks, uint8_t callbacksNumber, notifier_user_function_t
userFunction, void * userData)`

Parameters

<i>notifierHandle</i>	A pointer to the notifier handle.
<i>configs</i>	A pointer to an array with references to all configurations which is handled by the Notifier.
<i>configsNumber</i>	Number of configurations. Size of the configuration array.
<i>callbacks</i>	A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value.
<i>callbacks-Number</i>	Number of registered callbacks. Size of the callbacks array.
<i>userFunction</i>	User function.
<i>userData</i>	User data passed to user function.

Returns

An error Code or kStatus_Success.

42.6.2 **status_t NOTIFIER_SwitchConfig (notifier_handle_t * *notifierHandle*, uint8_t *configIndex*, notifier_policy_t *policy*)**

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER_PolicyForcible) or exited (kNOTIFIER_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If an agreement is required, if any callback returns an error code, further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returns an error code, an error code indicating in which phase the error occurred is returned when [NOTIFIER_SwitchConfig\(\)](#) exits.

Parameters

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.

42.6.3 uint8_t NOTIFIER_GetErrorCallbackIndex (notifier_handle_t * *notifierHandle*)

This function returns an index of the last callback that failed during the configuration switch while the last [NOTIFIER_SwitchConfig\(\)](#) was called. If the last [NOTIFIER_SwitchConfig\(\)](#) call ended successfully value equal to callbacks number is returned. The returned value represents an index in the array of static call-backs.

Parameters

<i>notifierHandle</i>	Pointer to the notifier handle
-----------------------	--------------------------------

Returns

Callback Index of the last failed callback or value equal to callbacks count.

Chapter 43

Shell

Overview

This section describes the programming interface of the Shell middleware.

Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

Function groups

43.2.1 Initialization

To initialize the Shell middleware, call the [SHELL_Init\(\)](#) function with these parameters. This function automatically enables the middleware.

```
shell_status_t SHELL_Init(shell_handle_t shellHandle,  
                           serial_handle_t serialHandle, char *prompt);
```

Then, after the initialization was successful, call a command to control MCUs.

This example shows how to call the [SHELL_Init\(\)](#) given the user configuration structure.

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
```

43.2.2 Advanced Feature

- Support to get a character from standard input devices.

```
static shell_status_t SHELL_GetChar(shell_context_handle_t *shellContextHandle, uint8_t *ch);
```

Commands	Description
help	List all the registered commands.
exit	Exit program.

43.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");  
SHELL_Task((s_shellHandle);
```

Function groups

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_USE_COMMON_TASK](#) (1U)
Macro to determine whether use common task.
- #define [SHELL_TASK_PRIORITY](#) (2U)
Macro to set shell task priority.
- #define [SHELL_TASK_STACK_SIZE](#) (1000U)
Macro to set shell task stack size.
- #define [SHELL_HANDLE_DEFINE](#)(name) uint32_t name[(([SHELL_HANDLE_SIZE](#) + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the shell handle.
- #define [SHELL_COMMAND_DEFINE](#)(command, descriptor, callback, paramCount)
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 status.

Shell functional operation

- `shell_status_t SHELL_Init (shell_handle_t shellHandle, serial_handle_t serialHandle, char *prompt)`
Initializes the shell module.
- `shell_status_t SHELL_RegisterCommand (shell_handle_t shellHandle, shell_command_t *shellCommand)`
Registers the shell command.
- `shell_status_t SHELL_UnregisterCommand (shell_command_t *shellCommand)`
Unregisters the shell command.
- `shell_status_t SHELL_Write (shell_handle_t shellHandle, 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.

Data Structure Documentation

43.3.1 struct shell_command_t

Data Fields

- `const char * pcCommand`
The command that is executed.
- `char * pcHelpString`
String that describes how to use the command.
- `const cmd_function_t pFuncCallBack`
A pointer to the callback function that returns the output generated by the command.
- `uint8_t cExpectedNumberOfParameters`
Commands expect a fixed number of parameters, which may be zero.
- `list_element_t link`
link of the element

43.3.1.0.0.101 Field Documentation

43.3.1.0.0.101.1 const char* shell_command_t::pcCommand

For example "help". It must be all lower case.

43.3.1.0.0.101.2 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".

43.3.1.0.0.101.3 const cmd_function_t shell_command_t::pFuncCallBack

43.3.1.0.0.101.4 uint8_t shell_command_t::cExpectedNumberOfParameters

Macro Definition Documentation

43.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE

43.4.2 #define SHELL_AUTO_COMPLETE (1U)

43.4.3 #define SHELL_BUFFER_SIZE (64U)

43.4.4 #define SHELL_MAX_ARGS (8U)

43.4.5 #define SHELL_HISTORY_COUNT (3U)

43.4.6 #define SHELL_HANDLE_SIZE (520U)

It is the sum of the $SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE$

43.4.7 #define SHELL_USE_COMMON_TASK (1U)

43.4.8 #define SHELL_TASK_PRIORITY (2U)

43.4.9 #define SHELL_TASK_STACK_SIZE (1000U)

**43.4.10 #define SHELL_HANDLE_DEFINE(*name*) uint32_t
name[(((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t)))]**

This macro is used to define a 4 byte aligned shell handle. Then use "(shell_handle_t)name" to get the shell handle.

The macro should be global and could be optional. You could also define shell handle by yourself.

This is an example,

```
* SHELL_HANDLE_DEFINE(shellHandle);  
*
```

Parameters

<i>name</i>	The name string of the shell handle.
-------------	--------------------------------------

43.4.11 #define SHELL_COMMAND_DEFINE(*command*, *descriptor*, *callback*, *paramCount*)

Value:

```
\
    shell_command_t g_shellCommand##command = {
        (#command), (descriptor), (callback), (paramCount), {0},
    }
\
```

This macro is used to define the shell command structure [shell_command_t](#). And then uses the macro SHELL_COMMAND to get the command structure pointer. The macro should not be used in any function.

This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

<i>command</i>	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".
<i>descriptor</i>	The description of the command is used for showing the command usage when "help" is typing.
<i>callback</i>	The callback of the command is used to handle the command line when the input command is matched.
<i>paramCount</i>	The max parameter count of the current command.

43.4.12 #define SHELL_COMMAND(*command*) &g_shellCommand##command

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL_COMMAND_DEFINE is used.

Function Documentation

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".
----------------	--

Typedef Documentation

43.5.1 typedef shell_status_t(* cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char **argv)

Enumeration Type Documentation

43.6.1 enum shell_status_t

Enumerator

kStatus_SHELL_Success Success.
kStatus_SHELL_Error Failed.
kStatus_SHELL_OpenWriteHandleFailed Open write handle failed.
kStatus_SHELL_OpenReadHandleFailed Open read handle failed.

Function Documentation

43.7.1 shell_status_t SHELL_Init (shell_handle_t *shellHandle*, serial_handle_t *serialHandle*, char * *prompt*)

This function must be called before calling all other Shell functions. Call operation the Shell commands with user-defined settings. The example below shows how to set up the Shell and how to call the SHELL_Init function by passing in these parameters. This is an example.

```
* static SHELL_HANDLE_DEFINE(s_shellHandle);  
* SHELL_Init((shell_handle_t)s_shellHandle, (  
*     serial_handle_t)s_serialHandle, "Test@SHELL>");  
*
```

Parameters

<i>shellHandle</i>	Pointer to point to a memory space of size SHELL_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SHELL_HANDLE_DEFINE(shellHandle) ; or <code>uint32_t shellHandle[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
--------------------	---

<i>serialHandle</i>	The serial manager module handle pointer.
<i>prompt</i>	The string prompt pointer of Shell. Only the global variable can be passed.

Return values

<i>kStatus_SHELL_Success</i>	The shell initialization succeed.
<i>kStatus_SHELL_Error</i>	An error occurred when the shell is initialized.
<i>kStatus_SHELL_Open-WriteHandleFailed</i>	Open the write handle failed.
<i>kStatus_SHELL_Open-ReadHandleFailed</i>	Open the read handle failed.

43.7.2 **shell_status_t** SHELL_RegisterCommand (**shell_handle_t** *shellHandle*, **shell_command_t** * *shellCommand*)

This function is used to register the shell command by using the command configuration `shell_command_config_t`. This is a example,

```
* SHELL_COMMAND_DEFINE(exit, "\r\n\"exit\": Exit program\r\n", SHELL_ExitCommand, 0);
* SHELL_RegisterCommand(s_shellHandle, SHELL_COMMAND(exit));
*
```

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>shellCommand</i>	The command element.

Return values

<i>kStatus_SHELL_Success</i>	Successfully register the command.
<i>kStatus_SHELL_Error</i>	An error occurred.

43.7.3 **shell_status_t** SHELL_UnregisterCommand (**shell_command_t** * *shellCommand*)

This function is used to unregister the shell command.

Function Documentation

Parameters

<i>shellCommand</i>	The command element.
---------------------	----------------------

Return values

<i>kStatus_SHELL_Success</i>	Successfully unregister the command.
------------------------------	--------------------------------------

43.7.4 shell_status_t SHELL_Write (shell_handle_t *shellHandle*, char * *buffer*, uint32_t *length*)

This function is used to send data to the shell output stream.

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>buffer</i>	Start address of the data to write.
<i>length</i>	Length of the data to write.

Return values

<i>kStatus_SHELL_Success</i>	Successfully send data.
<i>kStatus_SHELL_Error</i>	An error occurred.

43.7.5 int SHELL_Printf (shell_handle_t *shellHandle*, const char * *formatString*, ...)

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

Parameters

<i>shellHandle</i>	The shell module handle pointer.
<i>formatString</i>	Format string.

Returns

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

43.7.6 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.

Function Documentation

Parameters

<i>shellHandle</i>	The shell module handle pointer.
--------------------	----------------------------------



Chapter 44

CODEC codec Driver

Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

Modules

- [codec common Driver](#)
- [cs42888 Driver](#)
- [da7212 Driver](#)
- [sgtl5000 Driver](#)
- [wm8904 Driver](#)
- [wm8960 Driver](#)

codec common Driver

44.2.1 Overview

The codec common driver provide codec control abstraction interface.

Data Structures

- struct `codec_config_t`
Initialize structure of the codec. [More...](#)
- struct `codec_capability_t`
codec capability [More...](#)
- struct `codec_handle_t`
Codec handle definition. [More...](#)

Macros

- #define `CODEC_VOLUME_MAX_VALUE` (100U)
codec maximum volume range

Enumerations

- enum `_codec_status` {
 `kStatus_CODEC_NotSupport` = MAKE_STATUS(kStatusGroup_CODEC, 0U),
 `kStatus_CODEC_DeviceNotRegistered` = MAKE_STATUS(kStatusGroup_CODEC, 1U),
 `kStatus_CODEC_I2CBusInitialFailed`,
 `kStatus_CODEC_I2CCommandTransferFailed` }
CODEC status.
- enum `codec_audio_protocol_t` {
 `kCODEC_BusI2S` = 0U,
 `kCODEC_BusLeftJustified` = 1U,
 `kCODEC_BusRightJustified` = 2U,
 `kCODEC_BusPCMA` = 3U,
 `kCODEC_BusPCMB` = 4U,
 `kCODEC_BusTDM` = 5U }
AUDIO format definition.
- enum `_codec_audio_sample_rate` {

```

kCODEC_AudioSampleRate8KHz = 8000U,
kCODEC_AudioSampleRate11025Hz = 11025U,
kCODEC_AudioSampleRate12KHz = 12000U,
kCODEC_AudioSampleRate16KHz = 16000U,
kCODEC_AudioSampleRate22050Hz = 22050U,
kCODEC_AudioSampleRate24KHz = 24000U,
kCODEC_AudioSampleRate32KHz = 32000U,
kCODEC_AudioSampleRate44100Hz = 44100U,
kCODEC_AudioSampleRate48KHz = 48000U,
kCODEC_AudioSampleRate96KHz = 96000U,
kCODEC_AudioSampleRate192KHz = 192000U,
kCODEC_AudioSampleRate384KHz = 384000U }

    audio sample rate definition
• enum _codec_audio_bit_width {
    kCODEC_AudioBitWidth16bit = 16U,
    kCODEC_AudioBitWidth20bit = 20U,
    kCODEC_AudioBitWidth24bit = 24U,
    kCODEC_AudioBitWidth32bit = 32U }

    audio bit width
• enum codec_module_t {
    kCODEC_ModuleADC = 0U,
    kCODEC_ModuleDAC = 1U,
    kCODEC_ModulePGA = 2U,
    kCODEC_ModuleHeadphone = 3U,
    kCODEC_ModuleSpeaker = 4U,
    kCODEC_ModuleLinein = 5U,
    kCODEC_ModuleLineout = 6U,
    kCODEC_ModuleVref = 7U,
    kCODEC_ModuleMicbias = 8U,
    kCODEC_ModuleMic = 9U,
    kCODEC_ModuleI2SIn = 10U,
    kCODEC_ModuleI2SOut = 11U,
    kCODEC_ModuleMxier = 12U }

    audio codec module
• enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }

    audio codec module control cmd
• enum _codec_module_ctrl_i2s_in_interface {
    kCODEC_ModuleI2SInInterfacePCM = 0U,
    kCODEC_ModuleI2SInInterfaceDSD = 1U }

    audio codec module digital interface
• enum _codec_record_source {
    kCODEC_RecordSourceDifferentialLine = 1U,
    kCODEC_RecordSourceLineInput = 2U,
    kCODEC_RecordSourceDifferentialMic = 4U,
    kCODEC_RecordSourceDigitalMic = 8U,
    kCODEC_RecordSourceSingleEndMic = 16U }

```

codec common Driver

- audio codec module record source value*
 - enum `_codec_reocrd_channel` {
 - `kCODEC_RecordChannelLeft1` = 1U,
 - `kCODEC_RecordChannelLeft2` = 2U,
 - `kCODEC_RecordChannelLeft3` = 4U,
 - `kCODEC_RecordChannelRight1` = 1U,
 - `kCODEC_RecordChannelRight2` = 2U,
 - `kCODEC_RecordChannelRight3` = 4U,
 - `kCODEC_RecordChannelDifferentialPositive1` = 1U,
 - `kCODEC_RecordChannelDifferentialPositive2` = 2U,
 - `kCODEC_RecordChannelDifferentialPositive3` = 4U,
 - `kCODEC_RecordChannelDifferentialNegative1` = 8U,
 - `kCODEC_RecordChannelDifferentialNegative2` = 16U,
 - `kCODEC_RecordChannelDifferentialNegative3` = 32U }
- audio codec record channel*
 - enum `_codec_play_source` {
 - `kCODEC_PlaySourcePGA` = 1U,
 - `kCODEC_PlaySourceInput` = 2U,
 - `kCODEC_PlaySourceDAC` = 4U,
 - `kCODEC_PlaySourceMixerIn` = 1U,
 - `kCODEC_PlaySourceMixerInLeft` = 2U,
 - `kCODEC_PlaySourceMixerInRight` = 4U,
 - `kCODEC_PlaySourceAux` = 8U }
- audio codec module play source value*
 - enum `_codec_play_channel` {
 - `kCODEC_PlayChannelHeadphoneLeft` = 1U,
 - `kCODEC_PlayChannelHeadphoneRight` = 2U,
 - `kCODEC_PlayChannelSpeakerLeft` = 4U,
 - `kCODEC_PlayChannelSpeakerRight` = 8U,
 - `kCODEC_PlayChannelLineOutLeft` = 16U,
 - `kCODEC_PlayChannelLineOutRight` = 32U,
 - `kCODEC_PlayChannelLeft0` = 1U,
 - `kCODEC_PlayChannelRight0` = 2U,
 - `kCODEC_PlayChannelLeft1` = 4U,
 - `kCODEC_PlayChannelRight1` = 8U,
 - `kCODEC_PlayChannelLeft2` = 16U,
 - `kCODEC_PlayChannelRight2` = 32U,
 - `kCODEC_PlayChannelLeft3` = 64U,
 - `kCODEC_PlayChannelRight3` = 128U }
- codec play channel*
 - enum `_codec_capability_flag` {

```

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

```

audio codec capability

Functions

- [status_t CODEC_Init](#) (codec_handle_t *handle, [codec_config_t](#) *config)
Codec initialization.
- [status_t CODEC_Deinit](#) (codec_handle_t *handle)
Codec de-initialization.
- [status_t CODEC_SetFormat](#) (codec_handle_t *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- [status_t CODEC_ModuleControl](#) (codec_handle_t *handle, [codec_module_ctrl_cmd_t](#) cmd, uint32_t data)
codec module control.
- [status_t CODEC_SetVolume](#) (codec_handle_t *handle, uint32_t channel, uint32_t volume)
set audio codec pl volume.
- [status_t CODEC_SetMute](#) (codec_handle_t *handle, uint32_t channel, bool mute)
set audio codec module mute.
- [status_t CODEC_SetPower](#) (codec_handle_t *handle, [codec_module_t](#) module, bool powerOn)
set audio codec power.
- [status_t CODEC_SetRecord](#) (codec_handle_t *handle, uint32_t recordRource)
codec set record source.
- [status_t CODEC_SetRecordChannel](#) (codec_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- [status_t CODEC_SetPlay](#) (codec_handle_t *handle, uint32_t playSource)
codec set play source.

Driver version

- #define [FSL_CODEC_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 2, 0))
CLOCK driver version 2.2.0.

44.2.2 Data Structure Documentation

44.2.2.1 struct codec_config_t

Data Fields

- uint32_t [codecDevType](#)
codec type
- void * [codecDevConfig](#)
Codec device specific configuration.

44.2.2.2 struct codec_capability_t

Data Fields

- uint32_t [codecModuleCapability](#)
codec module capability
- uint32_t [codecPlayCapability](#)
codec play capability
- uint32_t [codecRecordCapability](#)
codec record capability

44.2.2.3 struct _codec_handle

codec handle declaration

- Application should allocate a buffer with CODEC_HANDLE_SIZE for handle definition, such as uint8_t codecHandleBuffer[CODEC_HANDLE_SIZE]; codec_handle_t *codecHandle = codecHandleBuffer;

Data Fields

- [codec_config_t](#) * [codecConfig](#)
codec configuration function pointer
- const [codec_capability_t](#) * [codecCapability](#)
codec capability
- uint8_t [codecDevHandle](#) [HAL_CODEC_HANDLER_SIZE]
codec device handle

44.2.3 Macro Definition Documentation

44.2.3.1 #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

44.2.4 Enumeration Type Documentation

44.2.4.1 enum _codec_status

Enumerator

kStatus_CODEC_NotSupport CODEC not support status.

kStatus_CODEC_DeviceNotRegistered CODEC device register failed status.

kStatus_CODEC_I2CBusInitialFailed CODEC i2c bus initialization failed status.

kStatus_CODEC_I2CCommandTransferFailed CODEC i2c bus command transfer failed status.

44.2.4.2 enum codec_audio_protocol_t

Enumerator

kCODEC_BusI2S I2S type.
kCODEC_BusLeftJustified Left justified mode.
kCODEC_BusRightJustified Right justified mode.
kCODEC_BusPCMA DSP/PCM A mode.
kCODEC_BusPCMB DSP/PCM B mode.
kCODEC_BusTDM TDM mode.

44.2.4.3 enum _codec_audio_sample_rate

Enumerator

kCODEC_AudioSampleRate8KHz Sample rate 8000 Hz.
kCODEC_AudioSampleRate11025Hz Sample rate 11025 Hz.
kCODEC_AudioSampleRate12KHz Sample rate 12000 Hz.
kCODEC_AudioSampleRate16KHz Sample rate 16000 Hz.
kCODEC_AudioSampleRate22050Hz Sample rate 22050 Hz.
kCODEC_AudioSampleRate24KHz Sample rate 24000 Hz.
kCODEC_AudioSampleRate32KHz Sample rate 32000 Hz.
kCODEC_AudioSampleRate44100Hz Sample rate 44100 Hz.
kCODEC_AudioSampleRate48KHz Sample rate 48000 Hz.
kCODEC_AudioSampleRate96KHz Sample rate 96000 Hz.
kCODEC_AudioSampleRate192KHz Sample rate 192000 Hz.
kCODEC_AudioSampleRate384KHz Sample rate 384000 Hz.

44.2.4.4 enum _codec_audio_bit_width

Enumerator

kCODEC_AudioBitWidth16bit audio bit width 16
kCODEC_AudioBitWidth20bit audio bit width 20
kCODEC_AudioBitWidth24bit audio bit width 24
kCODEC_AudioBitWidth32bit audio bit width 32

44.2.4.5 enum codec_module_t

Enumerator

kCODEC_ModuleADC codec module ADC
kCODEC_ModuleDAC codec module DAC
kCODEC_ModulePGA codec module PGA

kCODEC_ModuleHeadphone codec module headphone
kCODEC_ModuleSpeaker codec module speaker
kCODEC_ModuleLinein codec module linein
kCODEC_ModuleLineout codec module lineout
kCODEC_ModuleVref codec module VREF
kCODEC_ModuleMicbias codec module MIC BIAS
kCODEC_ModuleMic codec module MIC
kCODEC_ModuleI2SIn codec module I2S in
kCODEC_ModuleI2SOut codec module I2S out
kCODEC_ModuleMxier codec module mixer

44.2.4.6 enum codec_module_ctrl_cmd_t

Enumerator

kCODEC_ModuleSwitchI2SInInterface module digital interface siwtch.

44.2.4.7 enum _codec_module_ctrl_i2s_in_interface

Enumerator

kCODEC_ModuleI2SInInterfacePCM Pcm interface.
kCODEC_ModuleI2SInInterfaceDSD DSD interface.

44.2.4.8 enum _codec_record_source

Enumerator

kCODEC_RecordSourceDifferentialLine record source from differential line
kCODEC_RecordSourceLineInput record source from line input
kCODEC_RecordSourceDifferentialMic record source from differential mic
kCODEC_RecordSourceDigitalMic record source from digital microphone
kCODEC_RecordSourceSingleEndMic record source from single microphone

44.2.4.9 enum _codec_reocrd_channel

Enumerator

kCODEC_RecordChannelLeft1 left record channel 1
kCODEC_RecordChannelLeft2 left record channel 2
kCODEC_RecordChannelLeft3 left record channel 3
kCODEC_RecordChannelRight1 right record channel 1

codec common Driver

kCODEC_RecordChannelRight2 right record channel 2
kCODEC_RecordChannelRight3 right record channel 3
kCODEC_RecordChannelDifferentialPositive1 differential positive record channel 1
kCODEC_RecordChannelDifferentialPositive2 differential positive record channel 2
kCODEC_RecordChannelDifferentialPositive3 differential positive record channel 3
kCODEC_RecordChannelDifferentialNegative1 differential negative record channel 1
kCODEC_RecordChannelDifferentialNegative2 differential negative record channel 2
kCODEC_RecordChannelDifferentialNegative3 differential negative record channel 3

44.2.4.10 enum _codec_play_source

Enumerator

kCODEC_PlaySourcePGA play source PGA, bypass ADC
kCODEC_PlaySourceInput play source Input3
kCODEC_PlaySourceDAC play source DAC
kCODEC_PlaySourceMixerIn play source mixer in
kCODEC_PlaySourceMixerInLeft play source mixer in left
kCODEC_PlaySourceMixerInRight play source mixer in right
kCODEC_PlaySourceAux play source mixer in AUx

44.2.4.11 enum _codec_play_channel

Enumerator

kCODEC_PlayChannelHeadphoneLeft play channel headphone left
kCODEC_PlayChannelHeadphoneRight play channel headphone right
kCODEC_PlayChannelSpeakerLeft play channel speaker left
kCODEC_PlayChannelSpeakerRight play channel speaker right
kCODEC_PlayChannelLineOutLeft play channel lineout left
kCODEC_PlayChannelLineOutRight play channel lineout right
kCODEC_PlayChannelLeft0 play channel left0
kCODEC_PlayChannelRight0 play channel right0
kCODEC_PlayChannelLeft1 play channel left1
kCODEC_PlayChannelRight1 play channel right1
kCODEC_PlayChannelLeft2 play channel left2
kCODEC_PlayChannelRight2 play channel right2
kCODEC_PlayChannelLeft3 play channel left3
kCODEC_PlayChannelRight3 play channel right3

44.2.4.12 enum_codec_capability_flag

Enumerator

kCODEC_SupportModuleADC codec capability of module ADC
kCODEC_SupportModuleDAC codec capability of module DAC
kCODEC_SupportModulePGA codec capability of module PGA
kCODEC_SupportModuleHeadphone codec capability of module headphone
kCODEC_SupportModuleSpeaker codec capability of module speaker
kCODEC_SupportModuleLinein codec capability of module linein
kCODEC_SupportModuleLineout codec capability of module lineout
kCODEC_SupportModuleVref codec capability of module vref
kCODEC_SupportModuleMicbias codec capability of module mic bias
kCODEC_SupportModuleMic codec capability of module mic bias
kCODEC_SupportModuleI2SIn codec capability of module I2S in
kCODEC_SupportModuleI2SOut codec capability of module I2S out
kCODEC_SupportModuleMixer codec capability of module mixer
kCODEC_SupportModuleI2SInSwitchInterface codec capability of module I2S in switch interface

kCODEC_SupportPlayChannelLeft0 codec capability of play channel left 0
kCODEC_SupportPlayChannelRight0 codec capability of play channel right 0
kCODEC_SupportPlayChannelLeft1 codec capability of play channel left 1
kCODEC_SupportPlayChannelRight1 codec capability of play channel right 1
kCODEC_SupportPlayChannelLeft2 codec capability of play channel left 2
kCODEC_SupportPlayChannelRight2 codec capability of play channel right 2
kCODEC_SupportPlayChannelLeft3 codec capability of play channel left 3
kCODEC_SupportPlayChannelRight3 codec capability of play channel right 3
kCODEC_SupportPlaySourcePGA codec capability of set playback source PGA
kCODEC_SupportPlaySourceInput codec capability of set playback source INPUT
kCODEC_SupportPlaySourceDAC codec capability of set playback source DAC
kCODEC_SupportPlaySourceMixerIn codec capability of set play source Mixer in
kCODEC_SupportPlaySourceMixerInLeft codec capability of set play source Mixer in left
kCODEC_SupportPlaySourceMixerInRight codec capability of set play source Mixer in right
kCODEC_SupportPlaySourceAux codec capability of set play source aux
kCODEC_SupportRecordSourceDifferentialLine codec capability of record source differential line

kCODEC_SupportRecordSourceLineInput codec capability of record source line input
kCODEC_SupportRecordSourceDifferentialMic codec capability of record source differential mic

kCODEC_SupportRecordSourceDigitalMic codec capability of record digital mic
kCODEC_SupportRecordSourceSingleEndMic codec capability of single end mic
kCODEC_SupportRecordChannelLeft1 left record channel 1
kCODEC_SupportRecordChannelLeft2 left record channel 2
kCODEC_SupportRecordChannelLeft3 left record channel 3
kCODEC_SupportRecordChannelRight1 right record channel 1

codec common Driver

kCODEC_SupportRecordChannelRight2 right record channel 2

kCODEC_SupportRecordChannelRight3 right record channel 3

44.2.5 Function Documentation

44.2.5.1 **status_t CODEC_Init (codec_handle_t * *handle*, codec_config_t * *config*)**

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configurations.

Returns

kStatus_Success is success, else de-initial failed.

44.2.5.2 **status_t CODEC_Deinit (codec_handle_t * *handle*)**

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

44.2.5.3 **status_t CODEC_SetFormat (codec_handle_t * *handle*, uint32_t *mclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)**

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.

<i>bitWidth</i>	bit width.
-----------------	------------

Returns

kStatus_Success is success, else configure failed.

44.2.5.4 **status_t CODEC_ModuleControl (codec_handle_t * *handle*, codec_module_ctrl_cmd_t *cmd*, uint32_t *data*)**

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature.

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

44.2.5.5 **status_t CODEC_SetVolume (codec_handle_t * *handle*, uint32_t *channel*, uint32_t *volume*)**

Parameters

<i>handle</i>	codec handle.
<i>channel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

44.2.5.6 **status_t CODEC_SetMute (codec_handle_t * *handle*, uint32_t *channel*, bool *mute*)**

codec common Driver

Parameters

<i>handle</i>	codec handle.
<i>channel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>mute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

44.2.5.7 **status_t CODEC_SetPower (codec_handle_t * *handle*, codec_module_t *module*, bool *powerOn*)**

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

44.2.5.8 **status_t CODEC_SetRecord (codec_handle_t * *handle*, uint32_t *recordRource*)**

Parameters

<i>handle</i>	codec handle.
<i>recordRource</i>	audio codec record source, can be a value or combine value of <code>_codec_record_source</code> .

Returns

kStatus_Success is success, else configure failed.

44.2.5.9 **status_t CODEC_SetRecordChannel (codec_handle_t * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)**

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

44.2.5.10 status_t CODEC_SetPlay (codec_handle_t * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

cs42888 Driver

44.3.1 Overview

The cs42888 driver provide codec control interface.

Modules

- [cs42888 adapter](#)

Data Structures

- struct [cs42888_audio_format_t](#)
cs42888 audio format [More...](#)
- struct [cs42888_config_t](#)
Initialize structure of CS42888. [More...](#)
- struct [cs42888_handle_t](#)
cs42888 handler [More...](#)

Macros

- #define [CS42888_I2C_HANDLER_SIZE](#) CODEC_I2C_MASTER_HANDLER_SIZE
CS42888 handle size.
- #define [CS42888_ID](#) 0x01
Define the register address of CS42888.
- #define [CS42888_CACHEREGNUM](#) 28
Cache register number.
- #define [CS42888_I2C_ADDR](#) 0x48
CS42888 I2C address.
- #define [CS42888_I2C_BITRATE](#) (1000000U)
CS42888 I2C baudrate.

Typedefs

- typedef void(* [cs42888_reset](#))(bool state)
cs42888 reset function pointer

Enumerations

- enum [cs42888_func_mode](#) {
 [kCS42888_ModeMasterSSM](#) = 0x0,
 [kCS42888_ModeMasterDSM](#) = 0x1,
 [kCS42888_ModeMasterQSM](#) = 0x2,
 [kCS42888_ModeSlave](#) = 0x3 }
}

- CS42888 support modes.*
- enum `cs42888_module_t` {
`kCS42888_ModuleDACPair1` = 0x2,
`kCS42888_ModuleDACPair2` = 0x4,
`kCS42888_ModuleDACPair3` = 0x8,
`kCS42888_ModuleDACPair4` = 0x10,
`kCS42888_ModuleADCPair1` = 0x20,
`kCS42888_ModuleADCPair2` = 0x40 }

- Modules in CS42888 board.*
- enum `cs42888_bus_t` {
`kCS42888_BusLeftJustified` = 0x0,
`kCS42888_BusI2S` = 0x1,
`kCS42888_BusRightJustified` = 0x2,
`kCS42888_BusOL1` = 0x4,
`kCS42888_BusOL2` = 0x5,
`kCS42888_BusTDM` = 0x6 }

- CS42888 supported audio bus type.*
- enum `_cs42888_play_channel` {
`kCS42888_AOUT1` = 1U,
`kCS42888_AOUT2` = 2U,
`kCS42888_AOUT3` = 3U,
`kCS42888_AOUT4` = 4U,
`kCS42888_AOUT5` = 5U,
`kCS42888_AOUT6` = 6U,
`kCS42888_AOUT7` = 7U,
`kCS42888_AOUT8` = 8U }

CS42888 play channel.

Functions

- `status_t CS42888_Init` (`cs42888_handle_t` *handle, `cs42888_config_t` *config)
CS42888 initialize function.
- `status_t CS42888_Deinit` (`cs42888_handle_t` *handle)
Deinit the CS42888 codec.
- `status_t CS42888_SetProtocol` (`cs42888_handle_t` *handle, `cs42888_bus_t` protocol, `uint32_t` bit-Width)
Set the audio transfer protocol.
- `void CS42888_SetFuncMode` (`cs42888_handle_t` *handle, `cs42888_func_mode` mode)
Set CS42888 to differernt working mode.
- `status_t CS42888_SelectFunctionalMode` (`cs42888_handle_t` *handle, `cs42888_func_mode` adc-Mode, `cs42888_func_mode` dacMode)
Set CS42888 to differernt functional mode.
- `status_t CS42888_SetAOUTVolume` (`cs42888_handle_t` *handle, `uint8_t` channel, `uint8_t` volume)
Set the volume of different modules in CS42888.
- `status_t CS42888_SetAINVolume` (`cs42888_handle_t` *handle, `uint8_t` channel, `uint8_t` volume)
Set the volume of different modules in CS42888.
- `uint8_t CS42888_GetAOUTVolume` (`cs42888_handle_t` *handle, `uint8_t` channel)

cs42888 Driver

- Get the volume of different AOUT channel in CS42888.*
- `uint8_t CS42888_GetAINVolume (cs42888_handle_t *handle, uint8_t channel)`
Get the volume of different AIN channel in CS42888.
- `status_t CS42888_SetMute (cs42888_handle_t *handle, uint8_t channelMask)`
Mute modules in CS42888.
- `status_t CS42888_SetChannelMute (cs42888_handle_t *handle, uint8_t channel, bool isMute)`
Mute channel modules in CS42888.
- `status_t CS42888_SetModule (cs42888_handle_t *handle, cs42888_module_t module, bool isEnabled)`
Enable/disable expected devices.
- `status_t CS42888_ConfigDataFormat (cs42888_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)`
Configure the data format of audio data.
- `status_t CS42888_WriteReg (cs42888_handle_t *handle, uint8_t reg, uint8_t val)`
Write register to CS42888 using I2C.
- `status_t CS42888_ReadReg (cs42888_handle_t *handle, uint8_t reg, uint8_t *val)`
Read register from CS42888 using I2C.
- `status_t CS42888_ModifyReg (cs42888_handle_t *handle, uint8_t reg, uint8_t mask, uint8_t val)`
Modify some bits in the register using I2C.

Driver version

- `#define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`
cs42888 driver version 2.1.1.

44.3.2 Data Structure Documentation

44.3.2.1 struct cs42888_audio_format_t

Data Fields

- `uint32_t mclk_HZ`
master clock frequency
- `uint32_t sampleRate`
sample rate
- `uint32_t bitWidth`
bit width

44.3.2.2 struct cs42888_config_t

Data Fields

- `cs42888_bus_t bus`
Audio transfer protocol.
- `cs42888_audio_format_t format`
cs42888 audio format

- `cs42888_func_mode` `ADCMode`
CS42888 ADC function mode.
- `cs42888_func_mode` `DACMode`
CS42888 DAC function mode.
- `bool` `master`
true is master, false is slave
- `codec_i2c_config_t` `i2cConfig`
i2c bus configuration
- `uint8_t` `slaveAddress`
slave address
- `cs42888_reset` `reset`
reset function pointer

44.3.2.2.0.102 Field Documentation

44.3.2.2.0.102.1 `cs42888_func_mode` `cs42888_config_t::ADCMode`

44.3.2.2.0.102.2 `cs42888_func_mode` `cs42888_config_t::DACMode`

44.3.2.3 struct `cs42888_handle_t`

Data Fields

- `cs42888_config_t` * `config`
cs42888 config pointer
- `uint8_t` `i2cHandle` [`CS42888_I2C_HANDLER_SIZE`]
i2c handle pointer

44.3.3 Macro Definition Documentation

44.3.3.1 `#define` `FSL_CS42888_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)

44.3.3.2 `#define` `CS42888_ID` `0x01`

44.3.3.3 `#define` `CS42888_I2C_ADDR` `0x48`

44.3.4 Enumeration Type Documentation

44.3.4.1 enum `cs42888_func_mode`

Enumerator

`kCS42888_ModeMasterSSM` master single speed mode
`kCS42888_ModeMasterDSM` master dual speed mode
`kCS42888_ModeMasterQSM` master quad speed mode
`kCS42888_ModeSlave` master single speed mode

44.3.4.2 enum cs42888_module_t

Enumerator

kCS42888_ModuleDACPair1 DAC pair1 (AOUT1 and AOUT2) module in CS42888.
kCS42888_ModuleDACPair2 DAC pair2 (AOUT3 and AOUT4) module in CS42888.
kCS42888_ModuleDACPair3 DAC pair3 (AOUT5 and AOUT6) module in CS42888.
kCS42888_ModuleDACPair4 DAC pair4 (AOUT7 and AOUT8) module in CS42888.
kCS42888_ModuleADCPair1 ADC pair1 (AIN1 and AIN2) module in CS42888.
kCS42888_ModuleADCPair2 ADC pair2 (AIN3 and AIN4) module in CS42888.

44.3.4.3 enum cs42888_bus_t

Enumerator

kCS42888_BusLeftJustified Left justified format, up to 24 bits.
kCS42888_BusI2S I2S format, up to 24 bits.
kCS42888_BusRightJustified Right justified, can support 16bits and 24 bits.
kCS42888_BusOL1 One-Line #1 mode.
kCS42888_BusOL2 One-Line #2 mode.
kCS42888_BusTDM TDM mode.

44.3.4.4 enum _cs42888_play_channel

Enumerator

kCS42888_AOUT1 aout1
kCS42888_AOUT2 aout2
kCS42888_AOUT3 aout3
kCS42888_AOUT4 aout4
kCS42888_AOUT5 aout5
kCS42888_AOUT6 aout6
kCS42888_AOUT7 aout7
kCS42888_AOUT8 aout8

44.3.5 Function Documentation

44.3.5.1 status_t CS42888_Init (cs42888_handle_t * *handle*, cs42888_config_t * *config*)

The second parameter is NULL to CS42888 in this version. If users want to change the settings, they have to use cs42888_write_reg() or cs42888_modify_reg() to set the register value of CS42888. Note: If the codec_config is NULL, it would initialize CS42888 using default settings. The default setting: codec_config->bus = kCS42888_BusI2S codec_config->ADCmode = kCS42888_ModeSlave codec_config->DACmode = kCS42888_ModeSlave

Parameters

<i>handle</i>	CS42888 handle structure.
<i>config</i>	CS42888 configuration structure.

44.3.5.2 status_t CS42888_Deinit (cs42888_handle_t * *handle*)

This function close all modules in CS42888 to save power.

Parameters

<i>handle</i>	CS42888 handle structure pointer.
---------------	-----------------------------------

44.3.5.3 status_t CS42888_SetProtocol (cs42888_handle_t * *handle*, cs42888_bus_t *protocol*, uint32_t *bitWidth*)

CS42888 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>protocol</i>	Audio data transfer protocol.
<i>bitWidth</i>	bit width

44.3.5.4 void CS42888_SetFuncMode (cs42888_handle_t * *handle*, cs42888_func_mode *mode*)

Deprecated api, Do not use it anymore. It has been superceded by [CS42888_SelectFunctionalMode](#).

Parameters

<i>handle</i>	CS42888 handle structure.
<i>mode</i>	different working mode of CS42888.

44.3.5.5 status_t CS42888_SelectFunctionalMode (cs42888_handle_t * *handle*, cs42888_func_mode *adcMode*, cs42888_func_mode *dacMode*)

cs42888 Driver

Parameters

<i>handle</i>	CS42888 handle structure.
<i>adcMode</i>	different working mode of CS42888.
<i>dacMode</i>	different working mode of CS42888.

44.3.5.6 **status_t CS42888_SetAOUTVolume (cs42888_handle_t * *handle*, uint8_t *channel*, uint8_t *volume*)**

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	AOUT channel, it shall be 1~8.
<i>volume</i>	Volume value need to be set.

44.3.5.7 **status_t CS42888_SetAINVolume (cs42888_handle_t * *handle*, uint8_t *channel*, uint8_t *volume*)**

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	AIN channel, it shall be 1~4.
<i>volume</i>	Volume value need to be set.

44.3.5.8 **uint8_t CS42888_GetAOUTVolume (cs42888_handle_t * *handle*, uint8_t *channel*)**

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	AOUT channel, it shall be 1~8.

44.3.5.9 uint8_t CS42888_GetAINVolume (cs42888_handle_t * *handle*, uint8_t *channel*)

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	AIN channel, it shall be 1~4.

44.3.5.10 status_t CS42888_SetMute (cs42888_handle_t * *handle*, uint8_t *channelMask*)

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channelMask</i>	Channel mask for mute. Mute channel 0, it shall be 0x1, while mute channel 0 and 1, it shall be 0x3. Mute all channel, it shall be 0xFF. Each bit represent one channel, 1 means mute, 0 means unmute.

44.3.5.11 status_t CS42888_SetChannelMute (cs42888_handle_t * *handle*, uint8_t *channel*, bool *isMute*)

Parameters

<i>handle</i>	CS42888 handle structure.
<i>channel</i>	reference _cs42888_play_channel.
<i>isMute</i>	true is mute, false is unmute.

44.3.5.12 status_t CS42888_SetModule (cs42888_handle_t * *handle*, cs42888_module_t *module*, bool *isEnabled*)

cs42888 Driver

Parameters

<i>handle</i>	CS42888 handle structure.
<i>module</i>	Module expected to enable.
<i>isEnabled</i>	Enable or disable moudles.

44.3.5.13 **status_t CS42888_ConfigDataFormat (cs42888_handle_t * *handle*, uint32_t *mclk*, uint32_t *sample_rate*, uint32_t *bits*)**

This function would configure the registers about the sample rate, bit depths.

Parameters

<i>handle</i>	CS42888 handle structure pointer.
<i>mclk</i>	Master clock frequency of I2S.
<i>sample_rate</i>	Sample rate of audio file running in CS42888. CS42888 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
<i>bits</i>	Bit depth of audio file (CS42888 only supports 16bit, 20bit, 24bit and 32 bit in HW).

44.3.5.14 **status_t CS42888_WriteReg (cs42888_handle_t * *handle*, uint8_t *reg*, uint8_t *val*)**

Parameters

<i>handle</i>	CS42888 handle structure.
<i>reg</i>	The register address in CS42888.
<i>val</i>	Value needs to write into the register.

44.3.5.15 **status_t CS42888_ReadReg (cs42888_handle_t * *handle*, uint8_t *reg*, uint8_t * *val*)**

Parameters

<i>handle</i>	CS42888 handle structure.
<i>reg</i>	The register address in CS42888.
<i>val</i>	Value written to.

44.3.5.16 `status_t CS42888_ModifyReg (cs42888_handle_t * handle, uint8_t reg, uint8_t mask, uint8_t val)`

Parameters

<i>handle</i>	CS42888 handle structure.
<i>reg</i>	The register address in CS42888.
<i>mask</i>	The mask code for the bits want to write. The bit you want to write should be 0.
<i>val</i>	Value needs to write into the register.

44.3.6 cs42888 adapter

44.3.6.1 Overview

The cs42888 adapter provide codec unify control interface .

Macros

- #define [HAL_CODEC_HANDLER_SIZE](#) ([CS42888_I2C_HANDLER_SIZE](#) + 4)
codec handler size

Enumerations

- enum [_codec_type](#) {
 [kCODEC_CS42888](#),
 [kCODEC_DA7212](#),
 [kCODEC_WM8904](#),
 [kCODEC_WM8960](#),
 [kCODEC_WM8524](#),
 [kCODEC_SGTL5000](#),
 [kCODEC_DA7212](#),
 [kCODEC_CS42888](#),
 [kCODEC_AK4497](#),
 [kCODEC_AK4458](#),
 [kCODEC_TFA9XXX](#),
 [kCODEC_TFA9896](#),
 [kCODEC_SGTL5000](#),
 [kCODEC_WM8904](#),
 [kCODEC_WM8960](#) }
codec type

Functions

- [status_t HAL_CODEC_Init](#) (void *handle, void *config)
Codec initialization.
- [status_t HAL_CODEC_Deinit](#) (void *handle)
Codec de-initialization.
- [status_t HAL_CODEC_SetFormat](#) (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bit-Width)
set audio data format.
- [status_t HAL_CODEC_SetVolume](#) (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- [status_t HAL_CODEC_SetMute](#) (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- [status_t HAL_CODEC_SetPower](#) (void *handle, uint32_t module, bool powerOn)

- *set audio codec module power.*
status_t HAL_CODEC_SetRecord (void *handle, uint32_t recordSource)
- *codec set record source.*
status_t HAL_CODEC_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
- *codec set record channel.*
status_t HAL_CODEC_SetPlay (void *handle, uint32_t playSource)
- *codec set play source.*
status_t HAL_CODEC_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
- *codec module control.*

44.3.6.2 Enumeration Type Documentation

44.3.6.2.1 enum_codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960

44.3.6.3 Function Documentation

44.3.6.3.1 status_t HAL_CODEC_Init (void * handle, void * config)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

cs42888 Driver

<i>config</i>	codec configuration.
---------------	----------------------

Returns

kStatus_Success is success, else initial failed.

44.3.6.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

44.3.6.3.3 status_t HAL_CODEC_SetFormat (void * *handle*, uint32_t *mclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

44.3.6.3.4 status_t HAL_CODEC_SetVolume (void * *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

44.3.6.3.5 status_t HAL_CODEC_SetMute (void * *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

44.3.6.3.6 status_t HAL_CODEC_SetPower (void * *handle*, uint32_t *module*, bool *powerOn*)

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

44.3.6.3.7 status_t HAL_CODEC_SetRecord (void * *handle*, uint32_t *recordSource*)

cs42888 Driver

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

44.3.6.3.8 status_t HAL_CODEC_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

44.3.6.3.9 status_t HAL_CODEC_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

44.3.6.3.10 status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference <code>_codec_module_ctrl_cmd</code> .
<i>data</i>	value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations.

Returns

`kStatus_Success` is success, else configure failed.

da7212 Driver

da7212 Driver

44.4.1 Overview

The da7212 driver provide codec control interface.

Modules

- [da7212 adapter](#)

Data Structures

- struct [da7212_pll_config_t](#)
da7212 pll configuration [More...](#)
- struct [da7212_audio_format_t](#)
da7212 audio format [More...](#)
- struct [da7212_config_t](#)
DA7212 configure structure. [More...](#)
- struct [da7212_handle_t](#)
da7212 codec handler [More...](#)

Macros

- #define [DA7212_I2C_HANDLER_SIZE](#) CODEC_I2C_MASTER_HANDLER_SIZE
da7212 handle size
- #define [DA7212_ADDRESS](#) (0x1A)
DA7212 I2C address.

Enumerations

- enum [da7212_Input_t](#) {
 [kDA7212_Input_AUX](#) = 0x0,
 [kDA7212_Input_MIC1_Dig](#),
 [kDA7212_Input_MIC1_An](#),
 [kDA7212_Input_MIC2](#) }
DA7212 input source select.
- enum [_da7212_play_channel](#) {
 [kDA7212_HeadphoneLeft](#) = 1U,
 [kDA7212_HeadphoneRight](#) = 2U,
 [kDA7212_Speaker](#) = 4U }
da7212 play channel
- enum [da7212_Output_t](#) {
 [kDA7212_Output_HP](#) = 0x0,
 [kDA7212_Output_SP](#) }

- DA7212 output device select.*
 - enum `_da7212_module` {
`kDA7212_ModuleADC`,
`kDA7212_ModuleDAC`,
`kDA7212_ModuleHeadphone`,
`kDA7212_ModuleSpeaker` }
 - DA7212 module.*
 - enum `da7212_dac_source_t` {
`kDA7212_DACSourceADC` = 0x0U,
`kDA7212_DACSourceInputStream` = 0x3U }
 - DA7212 functionality.*
 - enum `da7212_volume_t` {
`kDA7212_DACGainMute` = 0x7,
`kDA7212_DACGainM72DB` = 0x17,
`kDA7212_DACGainM60DB` = 0x1F,
`kDA7212_DACGainM54DB` = 0x27,
`kDA7212_DACGainM48DB` = 0x2F,
`kDA7212_DACGainM42DB` = 0x37,
`kDA7212_DACGainM36DB` = 0x3F,
`kDA7212_DACGainM30DB` = 0x47,
`kDA7212_DACGainM24DB` = 0x4F,
`kDA7212_DACGainM18DB` = 0x57,
`kDA7212_DACGainM12DB` = 0x5F,
`kDA7212_DACGainM6DB` = 0x67,
`kDA7212_DACGain0DB` = 0x6F,
`kDA7212_DACGain6DB` = 0x77,
`kDA7212_DACGain12DB` = 0x7F }
 - DA7212 volume.*
 - enum `da7212_protocol_t` {
`kDA7212_BusI2S` = 0x0,
`kDA7212_BusLeftJustified`,
`kDA7212_BusRightJustified`,
`kDA7212_BusDSPMode` }
 - The audio data transfer protocol choice.*
 - enum `da7212_sys_clk_source_t` {
`kDA7212_SysClkSourceMCLK` = 0U,
`kDA7212_SysClkSourcePLL` = 1U << 14 }
 - da7212 system clock source*
 - enum `da7212_pll_clk_source_t` { `kDA7212_PLLClkSourceMCLK` = 0U }
 - DA7212 pll clock source.*
 - enum `da7212_pll_out_clk_t` {
`kDA7212_PLLOutputClk11289600` = 11289600U,
`kDA7212_PLLOutputClk12288000` = 12288000U }
 - DA7212 output clock frequency.*
 - enum `da7212_master_bits_t` {

da7212 Driver

```
kDA7212_MasterBits32PerFrame = 0U,  
kDA7212_MasterBits64PerFrame = 1U,  
kDA7212_MasterBits128PerFrame = 2U,  
kDA7212_MasterBits256PerFrame = 3U }  
    master mode bits per frame
```

Functions

- **status_t DA7212_Init** (da7212_handle_t *handle, da7212_config_t *config)
DA7212 initialize function.
- **status_t DA7212_ConfigAudioFormat** (da7212_handle_t *handle, uint32_t masterClock_Hz, uint32_t sampleRate_Hz, uint32_t dataBits)
Set DA7212 audio format.
- **status_t DA7212_SetPLLConfig** (da7212_handle_t *handle, da7212_pll_config_t *config)
DA7212 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.
- **void DA7212_ChangeHPVolume** (da7212_handle_t *handle, da7212_volume_t volume)
Set DA7212 playback volume.
- **void DA7212_Mute** (da7212_handle_t *handle, bool isMuted)
Mute or unmute DA7212.
- **void DA7212_ChangeInput** (da7212_handle_t *handle, da7212_input_t DA7212_Input)
Set the input data source of DA7212.
- **void DA7212_ChangeOutput** (da7212_handle_t *handle, da7212_output_t DA7212_Output)
Set the output device of DA7212.
- **status_t DA7212_SetChannelVolume** (da7212_handle_t *handle, uint32_t module, uint32_t volume)
Set module volume.
- **status_t DA7212_SetChannelMute** (da7212_handle_t *handle, uint32_t module, bool isMute)
Set module mute.
- **status_t DA7212_SetProtocol** (da7212_handle_t *handle, da7212_protocol_t protocol)
Set protocol for DA7212.
- **status_t DA7212_SetMasterModeBits** (da7212_handle_t *handle, uint32_t bitWidth)
Set master mode bits per frame for DA7212.
- **status_t DA7212_WriteRegister** (da7212_handle_t *handle, uint8_t u8Register, uint8_t u8RegisterData)
Write a register for DA7212.
- **status_t DA7212_ReadRegister** (da7212_handle_t *handle, uint8_t u8Register, uint8_t *pu8RegisterData)
Get a register value of DA7212.
- **status_t DA7212_Deinit** (da7212_handle_t *handle)
Deinit DA7212.

Driver version

- **#define FSL_DA7212_DRIVER_VERSION** (MAKE_VERSION(2, 2, 0))
CLOCK driver version 2.2.0.

44.4.2 Data Structure Documentation

44.4.2.1 struct da7212_pll_config_t

Data Fields

- [da7212_pll_clk_source_t](#) *source*
pll reference clock source
- [uint32_t](#) [refClock_HZ](#)
pll reference clock frequency
- [da7212_pll_out_clk_t](#) [outputClock_HZ](#)
pll output clock frequency

44.4.2.2 struct da7212_audio_format_t

Data Fields

- [uint32_t](#) [mclk_HZ](#)
master clock frequency
- [uint32_t](#) [sampleRate](#)
sample rate
- [uint32_t](#) [bitWidth](#)
bit width
- [bool](#) [isBclkInvert](#)
bit clock interval

44.4.2.3 struct da7212_config_t

Data Fields

- [bool](#) [isMaster](#)
If DA7212 is master, true means master, false means slave.
- [da7212_protocol_t](#) [protocol](#)
Audio bus format, can be I2S, LJ, RJ or DSP mode.
- [da7212_dac_source_t](#) [dacSource](#)
DA7212 data source.
- [da7212_audio_format_t](#) [format](#)
audio format
- [uint8_t](#) [slaveAddress](#)
device address
- [codec_i2c_config_t](#) [i2cConfig](#)
i2c configuration
- [da7212_sys_clk_source_t](#) [sysClkSource](#)
system clock source
- [da7212_pll_config_t](#) * [pll](#)
pll configuration

da7212 Driver

44.4.2.3.0.1 Field Documentation

44.4.2.3.0.1.1 `bool da7212_config_t::isMaster`

44.4.2.3.0.1.2 `da7212_protocol_t da7212_config_t::protocol`

44.4.2.3.0.1.3 `da7212_dac_source_t da7212_config_t::dacSource`

44.4.2.4 `struct da7212_handle_t`

Data Fields

- `da7212_config_t * config`
da7212 config pointer
- `uint8_t i2cHandle [DA7212_I2C_HANDLER_SIZE]`
i2c handle

44.4.3 Macro Definition Documentation

44.4.3.1 `#define FSL_DA7212_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

44.4.4 Enumeration Type Documentation

44.4.4.1 `enum da7212_Input_t`

Enumerator

kDA7212_Input_AUX Input from AUX.
kDA7212_Input_MIC1_Dig Input from MIC1 Digital.
kDA7212_Input_MIC1_An Input from Mic1 Analog.
kDA7212_Input_MIC2 Input from MIC2.

44.4.4.2 `enum _da7212_play_channel`

Enumerator

kDA7212_HeadphoneLeft headphone left
kDA7212_HeadphoneRight headphone right
kDA7212_Speaker speaker channel

44.4.4.3 `enum da7212_Output_t`

Enumerator

kDA7212_Output_HP Output to headphone.

kDA7212_Output_SP Output to speaker.

44.4.4.4 enum _da7212_module

Enumerator

kDA7212_ModuleADC module ADC
kDA7212_ModuleDAC module DAC
kDA7212_ModuleHeadphone module headphone
kDA7212_ModuleSpeaker module speaker

44.4.4.5 enum da7212_dac_source_t

Enumerator

kDA7212_DACSourceADC DAC source from ADC.
kDA7212_DACSourceInputStream DAC source from.

44.4.4.6 enum da7212_volume_t

Enumerator

kDA7212_DACGainMute Mute DAC.
kDA7212_DACGainM72DB DAC volume -72db.
kDA7212_DACGainM60DB DAC volume -60db.
kDA7212_DACGainM54DB DAC volume -54db.
kDA7212_DACGainM48DB DAC volume -48db.
kDA7212_DACGainM42DB DAC volume -42db.
kDA7212_DACGainM36DB DAC volume -36db.
kDA7212_DACGainM30DB DAC volume -30db.
kDA7212_DACGainM24DB DAC volume -24db.
kDA7212_DACGainM18DB DAC volume -18db.
kDA7212_DACGainM12DB DAC volume -12db.
kDA7212_DACGainM6DB DAC volume -6db.
kDA7212_DACGain0DB DAC volume +0db.
kDA7212_DACGain6DB DAC volume +6db.
kDA7212_DACGain12DB DAC volume +12db.

44.4.4.7 enum da7212_protocol_t

Enumerator

kDA7212_BusI2S I2S Type.

da7212 Driver

kDA7212_BusLeftJustified Left justified.
kDA7212_BusRightJustified Right Justified.
kDA7212_BusDSPMode DSP mode.

44.4.4.8 enum da7212_sys_clk_source_t

Enumerator

kDA7212_SysClkSourceMCLK da7212 system clock soure from MCLK
kDA7212_SysClkSourcePLL da7212 system clock soure from pLL

44.4.4.9 enum da7212_pll_clk_source_t

Enumerator

kDA7212_PLLClkSourceMCLK DA7212 PLL clock source from MCLK.

44.4.4.10 enum da7212_pll_out_clk_t

Enumerator

kDA7212_PLLOutputClk11289600 output 112896000U
kDA7212_PLLOutputClk12288000 output 12288000U

44.4.4.11 enum da7212_master_bits_t

Enumerator

kDA7212_MasterBits32PerFrame master mode bits32 per frame
kDA7212_MasterBits64PerFrame master mode bits64 per frame
kDA7212_MasterBits128PerFrame master mode bits128 per frame
kDA7212_MasterBits256PerFrame master mode bits256 per frame

44.4.5 Function Documentation

44.4.5.1 status_t DA7212_Init (da7212_handle_t * *handle*, da7212_config_t * *config*)

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>config</i>	Codec configure structure. This parameter can be NULL, if NULL, set as default settings. The default setting: <pre> * sgtl_init_t codec_config * codec_config.route = kDA7212_RoutePlayback * codec_config.bus = kDA7212_BusI2S * codec_config.isMaster = false * </pre>

44.4.5.2 **status_t DA7212_ConfigAudioFormat (da7212_handle_t * *handle*, uint32_t *masterClock_Hz*, uint32_t *sampleRate_Hz*, uint32_t *dataBits*)**

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>masterClock_Hz</i>	Master clock frequency in Hz. If DA7212 is slave, use the frequency of master, if DA7212 as master, it should be 1228000 while sample rate frequency is 8k/12K/16-K/24K/32K/48K/96K, 11289600 while sample rate is 11.025K/22.05K/44.1K
<i>sampleRate_Hz</i>	Sample rate frequency in Hz.
<i>dataBits</i>	How many bits in a word of a audio frame, DA7212 only supports 16/20/24/32 bits.

44.4.5.3 **status_t DA7212_SetPLLConfig (da7212_handle_t * *handle*, da7212_pll_config_t * *config*)**

Parameters

<i>handle</i>	DA7212 handler pointer.
<i>config</i>	PLL configuration pointer.

44.4.5.4 **void DA7212_ChangeHPVolume (da7212_handle_t * *handle*, da7212_volume_t *volume*)**

da7212 Driver

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>volume</i>	The volume of playback.

44.4.5.5 void DA7212_Mute (da7212_handle_t * *handle*, bool *isMuted*)

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>isMuted</i>	True means mute, false means unmute.

44.4.5.6 void DA7212_ChangeInput (da7212_handle_t * *handle*, da7212_Input_t *DA7212_Input*)

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>DA7212_Input</i>	Input data source.

44.4.5.7 void DA7212_ChangeOutput (da7212_handle_t * *handle*, da7212_Output_t *DA7212_Output*)

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>DA7212_Output</i>	Output device of DA7212.

44.4.5.8 status_t DA7212_SetChannelVolume (da7212_handle_t * *handle*, uint32_t *module*, uint32_t *volume*)

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>module</i>	shoule be a value of _da7212_module
<i>volume</i>	volume range 0 - 100, 0 is mute, 100 is the maximum value.

44.4.5.9 **status_t DA7212_SetChannelMute (da7212_handle_t * *handle*, uint32_t *module*, bool *isMute*)**

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>module</i>	shoule be a value of _da7212_module
<i>isMute</i>	true is mute, false is unmute.

44.4.5.10 **status_t DA7212_SetProtocol (da7212_handle_t * *handle*, da7212_protocol_t *protocol*)**

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>protocol</i>	da7212_protocol_t.

44.4.5.11 **status_t DA7212_SetMasterModeBits (da7212_handle_t * *handle*, uint32_t *bitWidth*)**

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>bitWidth</i>	audio data bitwidth.

44.4.5.12 **status_t DA7212_WriteRegister (da7212_handle_t * *handle*, uint8_t *u8Register*, uint8_t *u8RegisterData*)**

da7212 Driver

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>u8Register</i>	DA7212 register address to be written.
<i>u8RegisterData</i>	Data to be written into register

44.4.5.13 `status_t DA7212_ReadRegister (da7212_handle_t * handle, uint8_t u8Register, uint8_t * pu8RegisterData)`

Parameters

<i>handle</i>	DA7212 handle pointer.
<i>u8Register</i>	DA7212 register address to be read.
<i>pu8Register-Data</i>	Pointer where the read out value to be stored.

44.4.5.14 `status_t DA7212_Deinit (da7212_handle_t * handle)`

Parameters

<i>handle</i>	DA7212 handle pointer.
---------------	------------------------

44.4.6 da7212 adapter

44.4.6.1 Overview

The da7212 adapter provide codec unify control interface .

Macros

- #define `HAL_CODEC_HANDLER_SIZE` (`DA7212_I2C_HANDLER_SIZE` + 4)
codec handler size

Enumerations

- enum `_codec_type` {
`kCODEC_CS42888`,
`kCODEC_DA7212`,
`kCODEC_WM8904`,
`kCODEC_WM8960`,
`kCODEC_WM8524`,
`kCODEC_SGTL5000`,
`kCODEC_DA7212`,
`kCODEC_CS42888`,
`kCODEC_AK4497`,
`kCODEC_AK4458`,
`kCODEC_TFA9XXX`,
`kCODEC_TFA9896`,
`kCODEC_SGTL5000`,
`kCODEC_WM8904`,
`kCODEC_WM8960` }
codec type

Functions

- `status_t HAL_CODEC_Init` (void *handle, void *config)
Codec initialization.
- `status_t HAL_CODEC_Deinit` (void *handle)
Codec de-initialization.
- `status_t HAL_CODEC_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bit-Width)
set audio data format.
- `status_t HAL_CODEC_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- `status_t HAL_CODEC_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- `status_t HAL_CODEC_SetPower` (void *handle, uint32_t module, bool powerOn)

da7212 Driver

- set audio codec module power.*
 - [status_t HAL_CODEC_SetRecord](#) (void *handle, uint32_t recordSource)
- codec set record source.*
 - [status_t HAL_CODEC_SetRecordChannel](#) (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
- codec set record channel.*
 - [status_t HAL_CODEC_SetPlay](#) (void *handle, uint32_t playSource)
- codec set play source.*
 - [status_t HAL_CODEC_ModuleControl](#) (void *handle, uint32_t cmd, uint32_t data)
- codec module control.*

44.4.6.2 Enumeration Type Documentation

44.4.6.2.1 enum_codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960

44.4.6.3 Function Documentation

44.4.6.3.1 status_t HAL_CODEC_Init (void * *handle*, void * *config*)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

<i>config</i>	codec configuration.
---------------	----------------------

Returns

kStatus_Success is success, else initial failed.

44.4.6.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

44.4.6.3.3 status_t HAL_CODEC_SetFormat (void * *handle*, uint32_t *mclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

44.4.6.3.4 status_t HAL_CODEC_SetVolume (void * *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

da7212 Driver

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

`kStatus_Success` is success, else configure failed.

44.4.6.3.5 `status_t HAL_CODEC_SetMute (void * handle, uint32_t playChannel, bool isMute)`

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>isMute</i>	true is mute, false is unmute.

Returns

`kStatus_Success` is success, else configure failed.

44.4.6.3.6 `status_t HAL_CODEC_SetPower (void * handle, uint32_t module, bool powerOn)`

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

`kStatus_Success` is success, else configure failed.

44.4.6.3.7 `status_t HAL_CODEC_SetRecord (void * handle, uint32_t recordSource)`

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

44.4.6.3.8 status_t HAL_CODEC_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

44.4.6.3.9 status_t HAL_CODEC_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

44.4.6.3.10 status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

da7212 Driver

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

sgtl5000 Driver

44.5.1 Overview

The sgtl5000 driver provide codec control interface.

Modules

- [sgtl5000 adapter](#)

Data Structures

- struct [sgtl_audio_format_t](#)
Audio format configuration. [More...](#)
- struct [sgtl_config_t](#)
Initailize structure of sgtl5000. [More...](#)
- struct [sgtl_handle_t](#)
SGTL codec handler. [More...](#)

Macros

- #define [CHIP_ID](#) 0x0000
Define the register address of sgtl5000.
- #define [SGTL5000_I2C_ADDR](#) 0x0A
SGTL5000 I2C address.
- #define [SGTL_I2C_HANDLER_SIZE](#) CODEC_I2C_MASTER_HANDLER_SIZE
sgtl handle size
- #define [SGTL_I2C_BITRATE](#) 100000U
sgtl i2c baudrate

Enumerations

- enum [sgtl_module_t](#) {
[kSGTL_ModuleADC](#) = 0x0,
[kSGTL_ModuleDAC](#),
[kSGTL_ModuleDAP](#),
[kSGTL_ModuleHP](#),
[kSGTL_ModuleI2SIN](#),
[kSGTL_ModuleI2SOUT](#),
[kSGTL_ModuleLineIn](#),
[kSGTL_ModuleLineOut](#),
[kSGTL_ModuleMicin](#) }
Modules in Sgtl5000 board.

sgtl5000 Driver

- enum `sgtl_route_t` {
 `kSGTL_RouteBypass` = 0x0,
 `kSGTL_RoutePlayback`,
 `kSGTL_RoutePlaybackandRecord`,
 `kSGTL_RoutePlaybackwithDAP`,
 `kSGTL_RoutePlaybackwithDAPandRecord`,
 `kSGTL_RouteRecord` }
 Sgtl5000 data route.
- enum `sgtl_protocol_t` {
 `kSGTL_BusI2S` = 0x0,
 `kSGTL_BusLeftJustified`,
 `kSGTL_BusRightJustified`,
 `kSGTL_BusPCMA`,
 `kSGTL_BusPCMB` }
 The audio data transfer protocol choice.
- enum `_sgtl_play_channel` {
 `kSGTL_HeadphoneLeft` = 0,
 `kSGTL_HeadphoneRight` = 1,
 `kSGTL_LineoutLeft` = 2,
 `kSGTL_LineoutRight` = 3 }
 sgtl play channel
- enum `_sgtl_record_source` {
 `kSGTL_RecordSourceLineIn` = 0U,
 `kSGTL_RecordSourceMic` = 1U }
 sgtl record source
- enum `_stgl_play_source` {
 `kSGTL_PlaySourceLineIn` = 0U,
 `kSGTL_PlaySourceDAC` = 1U }
 sgtl play source
- enum `sgtl_sclk_edge_t` {
 `kSGTL_SclkValidEdgeRising` = 0U,
 `kSGTL_SclkValidEdgeFalling` = 1U }
 SGTL SCLK valid edge.

Functions

- `status_t SGTL_Init` (`sgtl_handle_t` *handle, `sgtl_config_t` *config)
 sgtl5000 initialize function.
- `status_t SGTL_SetDataRoute` (`sgtl_handle_t` *handle, `sgtl_route_t` route)
 Set audio data route in sgtl5000.
- `status_t SGTL_SetProtocol` (`sgtl_handle_t` *handle, `sgtl_protocol_t` protocol)
 Set the audio transfer protocol.
- `void SGTL_SetMasterSlave` (`sgtl_handle_t` *handle, bool master)
 Set sgtl5000 as master or slave.
- `status_t SGTL_SetVolume` (`sgtl_handle_t` *handle, `sgtl_module_t` module, uint32_t volume)
 Set the volume of different modules in sgtl5000.
- `uint32_t SGTL_GetVolume` (`sgtl_handle_t` *handle, `sgtl_module_t` module)

- *Get the volume of different modules in sgtl5000.*
- `status_t SGTL_SetMute (sgtl_handle_t *handle, sgtl_module_t module, bool mute)`
Mute/unmute modules in sgtl5000.
- `status_t SGTL_EnableModule (sgtl_handle_t *handle, sgtl_module_t module)`
Enable expected devices.
- `status_t SGTL_DisableModule (sgtl_handle_t *handle, sgtl_module_t module)`
Disable expected devices.
- `status_t SGTL_Deinit (sgtl_handle_t *handle)`
Deinit the sgtl5000 codec.
- `status_t SGTL_ConfigDataFormat (sgtl_handle_t *handle, uint32_t mclk, uint32_t sample_rate, uint32_t bits)`
Configure the data format of audio data.
- `status_t SGTL_SetPlay (sgtl_handle_t *handle, uint32_t playSource)`
select SGTL codec play source.
- `status_t SGTL_SetRecord (sgtl_handle_t *handle, uint32_t recordSource)`
select SGTL codec record source.
- `status_t SGTL_WriteReg (sgtl_handle_t *handle, uint16_t reg, uint16_t val)`
Write register to sgtl using I2C.
- `status_t SGTL_ReadReg (sgtl_handle_t *handle, uint16_t reg, uint16_t *val)`
Read register from sgtl using I2C.
- `status_t SGTL_ModifyReg (sgtl_handle_t *handle, uint16_t reg, uint16_t clr_mask, uint16_t val)`
Modify some bits in the register using I2C.

Driver version

- `#define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`
CLOCK driver version 2.1.0.

44.5.2 Data Structure Documentation

44.5.2.1 struct sgtl_audio_format_t

Data Fields

- `uint32_t mclk_HZ`
master clock
- `uint32_t sampleRate`
Sample rate.
- `uint32_t bitWidth`
Bit width.
- `sgtl_sclk_edge_t sclkEdge`
sclk valid edge

44.5.2.2 struct sgtl_config_t

Data Fields

- [sgtl_route_t route](#)
Audio data route.
- [sgtl_protocol_t bus](#)
Audio transfer protocol.
- bool [master_slave](#)
Master or slave.
- [sgtl_audio_format_t format](#)
audio format
- uint8_t [slaveAddress](#)
code device slave address
- [codec_i2c_config_t i2cConfig](#)
i2c bus configuration

44.5.2.2.0.1 Field Documentation

44.5.2.2.0.1.1 sgtl_route_t sgtl_config_t::route

44.5.2.2.0.1.2 bool sgtl_config_t::master_slave

True means master, false means slave.

44.5.2.3 struct sgtl_handle_t

Data Fields

- [sgtl_config_t * config](#)
sgtl config pointer
- uint8_t [i2cHandle](#) [SGTL_I2C_HANDLER_SIZE]
i2c handle

44.5.3 Macro Definition Documentation

44.5.3.1 `#define FSL_SGTL5000_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

44.5.3.2 `#define CHIP_ID 0x0000`

44.5.3.3 `#define SGTL5000_I2C_ADDR 0x0A`

44.5.4 Enumeration Type Documentation

44.5.4.1 enum sgtl_module_t

Enumerator

kSGTL_ModuleADC ADC module in SGTL5000.
kSGTL_ModuleDAC DAC module in SGTL5000.
kSGTL_ModuleDAP DAP module in SGTL5000.
kSGTL_ModuleHP Headphone module in SGTL5000.
kSGTL_ModuleI2SIN I2S-IN module in SGTL5000.
kSGTL_ModuleI2SOUT I2S-OUT module in SGTL5000.
kSGTL_ModuleLineIn Line-in module in SGTL5000.
kSGTL_ModuleLineOut Line-out module in SGTL5000.
kSGTL_ModuleMicin Micphone module in SGTL5000.

44.5.4.2 enum sgtl_route_t

Note

Only provide some typical data route, not all route listed. Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

kSGTL_RouteBypass LINEIN->Headphone.
kSGTL_RoutePlayback I2SIN->DAC->Headphone.
kSGTL_RoutePlaybackandRecord I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.
kSGTL_RoutePlaybackwithDAP I2SIN->DAP->DAC->Headphone.
kSGTL_RoutePlaybackwithDAPandRecord I2SIN->DAP->DAC->HP, LINEIN->ADC->I2SOUT.
kSGTL_RouteRecord LINEIN->ADC->I2SOUT.

44.5.4.3 enum sgtl_protocol_t

Sgtl5000 only supports I2S format and PCM format.

sgtl5000 Driver

Enumerator

kSGTL_BusI2S I2S Type.
kSGTL_BusLeftJustified Left justified.
kSGTL_BusRightJustified Right Justified.
kSGTL_BusPCMA PCMA.
kSGTL_BusPCMB PCMB.

44.5.4.4 enum _sgtl_play_channel

Enumerator

kSGTL_HeadphoneLeft headphone left channel
kSGTL_HeadphoneRight headphone right channel
kSGTL_LineoutLeft lineout left channel
kSGTL_LineoutRight lineout right channel

44.5.4.5 enum _sgtl_record_source

Enumerator

kSGTL_RecordSourceLineIn record source line in
kSGTL_RecordSourceMic record source single end

44.5.4.6 enum _stgl_play_source

Enumerator

kSGTL_PlaySourceLineIn play source line in
kSGTL_PlaySourceDAC play source line in

44.5.4.7 enum sgtl_sclk_edge_t

Enumerator

kSGTL_SclkValidEdgeRising SCLK valid edge.
kSGTL_SclkValidEdgeFailling SCLK failling edge.

44.5.5 Function Documentation

44.5.5.1 status_t SGTL_Init (sgtl_handle_t * *handle*, sgtl_config_t * *config*)

This function calls SGTL_I2CInit(), and in this function, some configurations are fixed. The second parameter can be NULL. If users want to change the SGTL5000 settings, a configure structure should be

prepared.

Note

If the `codec_config` is NULL, it would initialize sgtl5000 using default settings. The default setting:

```
* sgtl_init_t codec_config
* codec_config.route = kSGTL_RoutePlaybackandRecord
* codec_config.bus = kSGTL_BusI2S
* codec_config.master = slave
*
```

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>config</i>	sgtl5000 configuration structure. If this pointer equals to NULL, it means using the default configuration.

Returns

Initialization status

44.5.5.2 status_t SGTL_SetDataRoute (sgtl_handle_t * *handle*, sgtl_route_t *route*)

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules.

Note

If a new route is set, the previous route would not work.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>route</i>	Audio data route in sgtl5000.

44.5.5.3 status_t SGTL_SetProtocol (sgtl_handle_t * *handle*, sgtl_protocol_t *protocol*)

Sgtl5000 only supports I2S, I2S left, I2S right, PCM A, PCM B format.

sgtl5000 Driver

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>protocol</i>	Audio data transfer protocol.

44.5.5.4 void SGTL_SetMasterSlave (sgtl_handle_t * *handle*, bool *master*)

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>master</i>	1 represent master, 0 represent slave.

44.5.5.5 status_t SGTL_SetVolume (sgtl_handle_t * *handle*, sgtl_module_t *module*, uint32_t *volume*)

This function would set the volume of sgtl5000 modules. This interface set module volume. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.
<i>volume</i>	Volume value need to be set. The value is the exact value in register.

44.5.5.6 uint32_t SGTL_GetVolume (sgtl_handle_t * *handle*, sgtl_module_t *module*)

This function gets the volume of sgtl5000 modules. This interface get DAC module volume. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.

Returns

Module value, the value is exact value in register.

44.5.5.7 `status_t SGTL_SetMute (sgtl_handle_t * handle, sgtl_module_t module, bool mute)`

sgtl5000 Driver

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Sgtl5000 module, such as DAC, ADC and etc.
<i>mute</i>	True means mute, and false means unmute.

44.5.5.8 status_t SGTL_EnableModule (sgtl_handle_t * *handle*, sgtl_module_t *module*)

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Module expected to enable.

44.5.5.9 status_t SGTL_DisableModule (sgtl_handle_t * *handle*, sgtl_module_t *module*)

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>module</i>	Module expected to enable.

44.5.5.10 status_t SGTL_Deinit (sgtl_handle_t * *handle*)

Shut down Sgtl5000 modules.

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
---------------	------------------------------------

44.5.5.11 status_t SGTL_ConfigDataFormat (sgtl_handle_t * *handle*, uint32_t *mclk*, uint32_t *sample_rate*, uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

--

<i>handle</i>	Sgtl5000 handle structure pointer.
<i>mclk</i>	Master clock frequency of I2S.
<i>sample_rate</i>	Sample rate of audio file running in sgtl5000. Sgtl5000 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
<i>bits</i>	Bit depth of audio file (Sgtl5000 only supports 16bit, 20bit, 24bit and 32 bit in HW).

44.5.5.12 status_t SGTL_SetPlay (sgtl_handle_t * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
<i>playSource</i>	play source value, reference _sgtl_play_source.

Returns

kStatus_Success, else failed.

44.5.5.13 status_t SGTL_SetRecord (sgtl_handle_t * *handle*, uint32_t *recordSource*)

Parameters

<i>handle</i>	Sgtl5000 handle structure pointer.
<i>recordSource</i>	record source value, reference _sgtl_record_source.

Returns

kStatus_Success, else failed.

44.5.5.14 status_t SGTL_WriteReg (sgtl_handle_t * *handle*, uint16_t *reg*, uint16_t *val*)

Parameters

<i>handle</i>	Sgtl5000 handle structure.
---------------	----------------------------

sgtl5000 Driver

<i>reg</i>	The register address in sgtl.
<i>val</i>	Value needs to write into the register.

44.5.5.15 `status_t SGTL_ReadReg (sgtl_handle_t * handle, uint16_t reg, uint16_t * val)`

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>reg</i>	The register address in sgtl.
<i>val</i>	Value written to.

44.5.5.16 `status_t SGTL_ModifyReg (sgtl_handle_t * handle, uint16_t reg, uint16_t clr_mask, uint16_t val)`

Parameters

<i>handle</i>	Sgtl5000 handle structure.
<i>reg</i>	The register address in sgtl.
<i>clr_mask</i>	The mask code for the bits want to write. The bit you want to write should be 0.
<i>val</i>	Value needs to write into the register.

44.5.6 sgtl5000 adapter

44.5.6.1 Overview

The sgtl5000 adapter provide codec unify control interface .

Macros

- #define `HAL_CODEC_HANDLER_SIZE` (`SGTL_I2C_HANDLER_SIZE` + 4)
codec handler size

Enumerations

- enum `_codec_type` {
`kCODEC_CS42888`,
`kCODEC_DA7212`,
`kCODEC_WM8904`,
`kCODEC_WM8960`,
`kCODEC_WM8524`,
`kCODEC_SGTL5000`,
`kCODEC_DA7212`,
`kCODEC_CS42888`,
`kCODEC_AK4497`,
`kCODEC_AK4458`,
`kCODEC_TFA9XXX`,
`kCODEC_TFA9896`,
`kCODEC_SGTL5000`,
`kCODEC_WM8904`,
`kCODEC_WM8960` }
codec type

Functions

- `status_t HAL_CODEC_Init` (void *handle, void *config)
Codec initialization.
- `status_t HAL_CODEC_Deinit` (void *handle)
Codec de-initialization.
- `status_t HAL_CODEC_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bit-Width)
set audio data format.
- `status_t HAL_CODEC_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- `status_t HAL_CODEC_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- `status_t HAL_CODEC_SetPower` (void *handle, uint32_t module, bool powerOn)

sgtl5000 Driver

- set audio codec module power.*
 - [status_t HAL_CODEC_SetRecord](#) (void *handle, uint32_t recordSource)
codec set record source.
- codec set record channel.*
 - [status_t HAL_CODEC_SetRecordChannel](#) (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- codec set play source.*
 - [status_t HAL_CODEC_SetPlay](#) (void *handle, uint32_t playSource)
codec set play source.
- codec module control.*
 - [status_t HAL_CODEC_ModuleControl](#) (void *handle, uint32_t cmd, uint32_t data)

44.5.6.2 Enumeration Type Documentation

44.5.6.2.1 enum_codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960

44.5.6.3 Function Documentation

44.5.6.3.1 status_t HAL_CODEC_Init (void * *handle*, void * *config*)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

<i>config</i>	codec configuration.
---------------	----------------------

Returns

kStatus_Success is success, else initial failed.

44.5.6.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

44.5.6.3.3 status_t HAL_CODEC_SetFormat (void * *handle*, uint32_t *mclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

44.5.6.3.4 status_t HAL_CODEC_SetVolume (void * *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

sgtl5000 Driver

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

44.5.6.3.5 status_t HAL_CODEC_SetMute (void * *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

44.5.6.3.6 status_t HAL_CODEC_SetPower (void * *handle*, uint32_t *module*, bool *powerOn*)

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

44.5.6.3.7 status_t HAL_CODEC_SetRecord (void * *handle*, uint32_t *recordSource*)

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

44.5.6.3.8 status_t HAL_CODEC_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

44.5.6.3.9 status_t HAL_CODEC_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

44.5.6.3.10 status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

sgtl5000 Driver

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference _codec_module_ctrl_cmd.
<i>data</i>	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

wm8960 Driver

44.6.1 Overview

The wm8960 driver provide codec control interface.

Modules

- [wm8960 adapter](#)

Data Structures

- struct [wm8960_audio_format_t](#)
wm8960 audio format [More...](#)
- struct [wm8960_config_t](#)
Initialize structure of WM8960. [More...](#)
- struct [wm8960_handle_t](#)
wm8960 codec handler [More...](#)

Macros

- #define [WM8960_I2C_HANDLER_SIZE](#) CODEC_I2C_MASTER_HANDLER_SIZE
wm8960 handle size
- #define [WM8960_LINVOL](#) 0x0
Define the register address of WM8960.
- #define [WM8960_CACHEREGNUM](#) 56
Cache register number.
- #define [WM8960_IFACE1_FORMAT_MASK](#) 0x03
WM8960_IFACE1 FORMAT bits.
- #define [WM8960_IFACE1_WL_MASK](#) 0x0C
WM8960_IFACE1 WL bits.
- #define [WM8960_IFACE1_LRP_MASK](#) 0x10
WM8960_IFACE1 LRP bit.
- #define [WM8960_IFACE1_DLRSWAP_MASK](#) 0x20
WM8960_IFACE1 DLRSWAP bit.
- #define [WM8960_IFACE1_MS_MASK](#) 0x40
WM8960_IFACE1 MS bit.
- #define [WM8960_IFACE1_BCLKINV_MASK](#) 0x80
WM8960_IFACE1 BCLKINV bit.
- #define [WM8960_IFACE1_ALRSWAP_MASK](#) 0x100
WM8960_IFACE1 ALRSWAP bit.
- #define [WM8960_POWER1_VREF_MASK](#) 0x40
WM8960_POWER1.
- #define [WM8960_POWER2_DACL_MASK](#) 0x100
WM8960_POWER2.
- #define [WM8960_I2C_ADDR](#) 0x1A
WM8960 I2C address.

wm8960 Driver

- #define WM8960_I2C_BAUDRATE (100000U)
WM8960 I2C baudrate.

Enumerations

- enum wm8960_module_t {
 kWM8960_ModuleADC = 0,
 kWM8960_ModuleDAC = 1,
 kWM8960_ModuleVREF = 2,
 kWM8960_ModuleHP = 3,
 kWM8960_ModuleMICB = 4,
 kWM8960_ModuleMIC = 5,
 kWM8960_ModuleLineIn = 6,
 kWM8960_ModuleLineOut = 7,
 kWM8960_ModuleSpeaker = 8,
 kWM8960_ModuleOMIX = 9 }
Modules in WM8960 board.
- enum _wm8960_play_channel {
 kWM8960_HeadphoneLeft = 1,
 kWM8960_HeadphoneRight = 2,
 kWM8960_SpeakerLeft = 4,
 kWM8960_SpeakerRight = 8 }
wm8960 play channel
- enum wm8960_play_source_t {
 kWM8960_PlaySourcePGA = 1,
 kWM8960_PlaySourceInput = 2,
 kWM8960_PlaySourceDAC = 4 }
wm8960 play source
- enum wm8960_route_t {
 kWM8960_RouteBypass = 0,
 kWM8960_RoutePlayback = 1,
 kWM8960_RoutePlaybackandRecord = 2,
 kWM8960_RouteRecord = 5 }
WM8960 data route.
- enum wm8960_protocol_t {
 kWM8960_BusI2S = 2,
 kWM8960_BusLeftJustified = 1,
 kWM8960_BusRightJustified = 0,
 kWM8960_BusPCMA = 3,
 kWM8960_BusPCMB = 3 | (1 << 4) }
The audio data transfer protocol choice.
- enum wm8960_input_t {

```

kWM8960_InputClosed = 0,
kWM8960_InputSingleEndedMic = 1,
kWM8960_InputDifferentialMicInput2 = 2,
kWM8960_InputDifferentialMicInput3 = 3,
kWM8960_InputLineINPUT2 = 4,
kWM8960_InputLineINPUT3 = 5 }
    wm8960 input source
• enum _wm8960_sample_rate {
    kWM8960_AudioSampleRate8KHz = 8000U,
    kWM8960_AudioSampleRate11025Hz = 11025U,
    kWM8960_AudioSampleRate12KHz = 12000U,
    kWM8960_AudioSampleRate16KHz = 16000U,
    kWM8960_AudioSampleRate22050Hz = 22050U,
    kWM8960_AudioSampleRate24KHz = 24000U,
    kWM8960_AudioSampleRate32KHz = 32000U,
    kWM8960_AudioSampleRate44100Hz = 44100U,
    kWM8960_AudioSampleRate48KHz = 48000U,
    kWM8960_AudioSampleRate96KHz = 96000U,
    kWM8960_AudioSampleRate192KHz = 192000U,
    kWM8960_AudioSampleRate384KHz = 384000U }
    audio sample rate definition
• enum _wm8960_audio_bit_width {
    kWM8960_AudioBitWidth16bit = 16U,
    kWM8960_AudioBitWidth20bit = 20U,
    kWM8960_AudioBitWidth24bit = 24U,
    kWM8960_AudioBitWidth32bit = 32U }
    audio bit width

```

Functions

- `status_t WM8960_Init (wm8960_handle_t *handle, const wm8960_config_t *wm8960Config)`
WM8960 initialize function.
- `status_t WM8960_Deinit (wm8960_handle_t *handle)`
Deinit the WM8960 codec.
- `status_t WM8960_SetDataRoute (wm8960_handle_t *handle, wm8960_route_t route)`
Set audio data route in WM8960.
- `status_t WM8960_SetLeftInput (wm8960_handle_t *handle, wm8960_input_t input)`
Set left audio input source in WM8960.
- `status_t WM8960_SetRightInput (wm8960_handle_t *handle, wm8960_input_t input)`
Set right audio input source in WM8960.
- `status_t WM8960_SetProtocol (wm8960_handle_t *handle, wm8960_protocol_t protocol)`
Set the audio transfer protocol.
- `void WM8960_SetMasterSlave (wm8960_handle_t *handle, bool master)`
Set WM8960 as master or slave.
- `status_t WM8960_SetVolume (wm8960_handle_t *handle, wm8960_module_t module, uint32_t volume)`

wm8960 Driver

- *Set the volume of different modules in WM8960.*
• `uint32_t WM8960_GetVolume (wm8960_handle_t *handle, wm8960_module_t module)`
Get the volume of different modules in WM8960.
- `status_t WM8960_SetMute (wm8960_handle_t *handle, wm8960_module_t module, bool isEnabled)`
Mute modules in WM8960.
- `status_t WM8960_SetModule (wm8960_handle_t *handle, wm8960_module_t module, bool isEnabled)`
Enable/disable expected devices.
- `status_t WM8960_SetPlay (wm8960_handle_t *handle, uint32_t playSource)`
SET the WM8960 play source.
- `status_t WM8960_ConfigDataFormat (wm8960_handle_t *handle, uint32_t sysclk, uint32_t sample_rate, uint32_t bits)`
Configure the data format of audio data.
- `status_t WM8960_SetJackDetect (wm8960_handle_t *handle, bool isEnabled)`
Enable/disable jack detect feature.
- `status_t WM8960_WriteReg (wm8960_handle_t *handle, uint8_t reg, uint16_t val)`
Write register to WM8960 using I2C.
- `status_t WM8960_ReadReg (uint8_t reg, uint16_t *val)`
Read register from WM8960 using I2C.
- `status_t WM8960_ModifyReg (wm8960_handle_t *handle, uint8_t reg, uint16_t mask, uint16_t val)`
Modify some bits in the register using I2C.

Driver version

- `#define FSL_WM8960_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`
CLOCK driver version 2.1.1.

44.6.2 Data Structure Documentation

44.6.2.1 struct wm8960_audio_format_t

Data Fields

- `uint32_t mclk_HZ`
master clock frequency
- `uint32_t sampleRate`
sample rate
- `uint32_t bitWidth`
bit width

44.6.2.2 struct wm8960_config_t

Data Fields

- [wm8960_route_t route](#)
Audio data route.
- [wm8960_protocol_t bus](#)
Audio transfer protocol.
- [wm8960_audio_format_t format](#)
Audio format.
- bool [master_slave](#)
Master or slave.
- bool [enableSpeaker](#)
True means enable class D speaker as output, false means no.
- [wm8960_input_t leftInputSource](#)
Left input source for WM8960.
- [wm8960_input_t rightInputSource](#)
Right input source for wm8960.
- [wm8960_play_source_t playSource](#)
play source
- uint8_t [slaveAddress](#)
wm8960 device address
- [codec_i2c_config_t i2cConfig](#)
i2c configuration

44.6.2.2.0.1 Field Documentation

44.6.2.2.0.1.1 [wm8960_route_t wm8960_config_t::route](#)

44.6.2.2.0.1.2 [bool wm8960_config_t::master_slave](#)

44.6.2.3 struct wm8960_handle_t

Data Fields

- const [wm8960_config_t * config](#)
wm8904 config pointer
- uint8_t [i2cHandle](#) [[WM8960_I2C_HANDLER_SIZE](#)]
i2c handle

44.6.3 Macro Definition Documentation

44.6.3.1 **#define WM8960_LINVOL 0x0**

44.6.3.2 **#define WM8960_I2C_ADDR 0x1A**

44.6.4 Enumeration Type Documentation

44.6.4.1 **enum wm8960_module_t**

Enumerator

kWM8960_ModuleADC ADC module in WM8960.

kWM8960_ModuleDAC DAC module in WM8960.

kWM8960_ModuleVREF VREF module.

kWM8960_ModuleHP Headphone.

kWM8960_ModuleMICB Mic bias.

kWM8960_ModuleMIC Input Mic.

kWM8960_ModuleLineIn Analog in PGA.

kWM8960_ModuleLineOut Line out module.

kWM8960_ModuleSpeaker Speaker module.

kWM8960_ModuleOMIX Output mixer.

44.6.4.2 **enum _wm8960_play_channel**

Enumerator

kWM8960_HeadphoneLeft wm8960 headphone left channel

kWM8960_HeadphoneRight wm8960 headphone right channel

kWM8960_SpeakerLeft wm8960 speaker left channel

kWM8960_SpeakerRight wm8960 speaker right channel

44.6.4.3 **enum wm8960_play_source_t**

Enumerator

kWM8960_PlaySourcePGA wm8960 play source PGA

kWM8960_PlaySourceInput wm8960 play source Input

kWM8960_PlaySourceDAC wm8960 play source DAC

44.6.4.4 enum wm8960_route_t

Only provide some typical data route, not all route listed. Note: Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

kWM8960_RouteBypass LINEIN->Headphone.
kWM8960_RoutePlayback I2SIN->DAC->Headphone.
kWM8960_RoutePlaybackandRecord I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.
kWM8960_RouteRecord LINEIN->ADC->I2SOUT.

44.6.4.5 enum wm8960_protocol_t

WM8960 only supports I2S format and PCM format.

Enumerator

kWM8960_BusI2S I2S type.
kWM8960_BusLeftJustified Left justified mode.
kWM8960_BusRightJustified Right justified mode.
kWM8960_BusPCMA PCM A mode.
kWM8960_BusPCMB PCM B mode.

44.6.4.6 enum wm8960_input_t

Enumerator

kWM8960_InputClosed Input device is closed.
kWM8960_InputSingleEndedMic Input as single ended mic, only use L/RINPUT1.
kWM8960_InputDifferentialMicInput2 Input as differential mic, use L/RINPUT1 and L/RINPUT2.
kWM8960_InputDifferentialMicInput3 Input as differential mic, use L/RINPUT1 and L/RINPUT3.
kWM8960_InputLineINPUT2 Input as line input, only use L/RINPUT2.
kWM8960_InputLineINPUT3 Input as line input, only use L/RINPUT3.

44.6.4.7 enum _wm8960_sample_rate

Enumerator

kWM8960_AudioSampleRate8KHz Sample rate 8000 Hz.
kWM8960_AudioSampleRate11025Hz Sample rate 11025 Hz.
kWM8960_AudioSampleRate12KHz Sample rate 12000 Hz.

wm8960 Driver

kWM8960_AudioSampleRate16KHz Sample rate 16000 Hz.
kWM8960_AudioSampleRate22050Hz Sample rate 22050 Hz.
kWM8960_AudioSampleRate24KHz Sample rate 24000 Hz.
kWM8960_AudioSampleRate32KHz Sample rate 32000 Hz.
kWM8960_AudioSampleRate44100Hz Sample rate 44100 Hz.
kWM8960_AudioSampleRate48KHz Sample rate 48000 Hz.
kWM8960_AudioSampleRate96KHz Sample rate 96000 Hz.
kWM8960_AudioSampleRate192KHz Sample rate 192000 Hz.
kWM8960_AudioSampleRate384KHz Sample rate 384000 Hz.

44.6.4.8 enum _wm8960_audio_bit_width

Enumerator

kWM8960_AudioBitWidth16bit audio bit width 16
kWM8960_AudioBitWidth20bit audio bit width 20
kWM8960_AudioBitWidth24bit audio bit width 24
kWM8960_AudioBitWidth32bit audio bit width 32

44.6.5 Function Documentation

44.6.5.1 status_t WM8960_Init (wm8960_handle_t * *handle*, const wm8960_config_t * *wm8960Config*)

The second parameter is NULL to WM8960 in this version. If users want to change the settings, they have to use wm8960_write_reg() or wm8960_modify_reg() to set the register value of WM8960. Note: If the codec_config is NULL, it would initialize WM8960 using default settings. The default setting: codec_config->route = kWM8960_RoutePlaybackandRecord codec_config->bus = kWM8960_BusI2S codec_config->master = slave

Parameters

<i>handle</i>	WM8960 handle structure.
<i>wm8960Config</i>	WM8960 configuration structure.

44.6.5.2 status_t WM8960_Deinit (wm8960_handle_t * *handle*)

This function close all modules in WM8960 to save power.

Parameters

<i>handle</i>	WM8960 handle structure pointer.
---------------	----------------------------------

44.6.5.3 **status_t WM8960_SetDataRoute (wm8960_handle_t * *handle*, wm8960_route_t *route*)**

This function would set the data route according to route. The route cannot be combined, as all route would enable different modules. Note: If a new route is set, the previous route would not work.

Parameters

<i>handle</i>	WM8960 handle structure.
<i>route</i>	Audio data route in WM8960.

44.6.5.4 **status_t WM8960_SetLeftInput (wm8960_handle_t * *handle*, wm8960_input_t *input*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>input</i>	Audio input source.

44.6.5.5 **status_t WM8960_SetRightInput (wm8960_handle_t * *handle*, wm8960_input_t *input*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>input</i>	Audio input source.

44.6.5.6 **status_t WM8960_SetProtocol (wm8960_handle_t * *handle*, wm8960_protocol_t *protocol*)**

WM8960 only supports I2S, left justified, right justified, PCM A, PCM B format.

wm8960 Driver

Parameters

<i>handle</i>	WM8960 handle structure.
<i>protocol</i>	Audio data transfer protocol.

44.6.5.7 void WM8960_SetMasterSlave (wm8960_handle_t * *handle*, bool *master*)

Parameters

<i>handle</i>	WM8960 handle structure.
<i>master</i>	1 represent master, 0 represent slave.

44.6.5.8 status_t WM8960_SetVolume (wm8960_handle_t * *handle*, wm8960_module_t *module*, uint32_t *volume*)

This function would set the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Module to set volume, it can be ADC, DAC, Headphone and so on.
<i>volume</i>	Volume value need to be set.

44.6.5.9 uint32_t WM8960_GetVolume (wm8960_handle_t * *handle*, wm8960_module_t *module*)

This function gets the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Module to set volume, it can be ADC, DAC, Headphone and so on.

Returns

Volume value of the module.

44.6.5.10 `status_t WM8960_SetMute (wm8960_handle_t * handle, wm8960_module_t module, bool isEnabled)`

wm8960 Driver

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Modules need to be mute.
<i>isEnabled</i>	Mute or unmute, 1 represent mute.

44.6.5.11 **status_t WM8960_SetModule (wm8960_handle_t * *handle*, wm8960_module_t *module*, bool *isEnabled*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Module expected to enable.
<i>isEnabled</i>	Enable or disable moudles.

44.6.5.12 **status_t WM8960_SetPlay (wm8960_handle_t * *handle*, uint32_t *playSource*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>playSource</i>	play source , can be a value combine of kWM8960_ModuleHeadphoneSourcePGA, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

44.6.5.13 **status_t WM8960_ConfigDataFormat (wm8960_handle_t * *handle*, uint32_t *sysclk*, uint32_t *sample_rate*, uint32_t *bits*)**

This function would configure the registers about the sample rate, bit depths.

Parameters

<i>handle</i>	WM8960 handle structure pointer.
<i>sysclk</i>	system clock of the codec which can be generated by MCLK or PLL output.
<i>sample_rate</i>	Sample rate of audio file running in WM8960. WM8960 now supports 8k, 11.025k, 12k, 16k, 22.05k, 24k, 32k, 44.1k, 48k and 96k sample rate.
<i>bits</i>	Bit depth of audio file (WM8960 only supports 16bit, 20bit, 24bit and 32 bit in HW).

44.6.5.14 `status_t WM8960_SetJackDetect (wm8960_handle_t * handle, bool isEnabled)`

Parameters

<i>handle</i>	WM8960 handle structure.
<i>isEnabled</i>	Enable or disable moudles.

44.6.5.15 `status_t WM8960_WriteReg (wm8960_handle_t * handle, uint8_t reg, uint16_t val)`

Parameters

<i>handle</i>	WM8960 handle structure.
<i>reg</i>	The register address in WM8960.
<i>val</i>	Value needs to write into the register.

44.6.5.16 `status_t WM8960_ReadReg (uint8_t reg, uint16_t * val)`

Parameters

<i>reg</i>	The register address in WM8960.
<i>val</i>	Value written to.

44.6.5.17 `status_t WM8960_ModifyReg (wm8960_handle_t * handle, uint8_t reg, uint16_t mask, uint16_t val)`

wm8960 Driver

Parameters

<i>handle</i>	WM8960 handle structure.
<i>reg</i>	The register address in WM8960.
<i>mask</i>	The mask code for the bits want to write. The bit you want to write should be 0.
<i>val</i>	Value needs to write into the register.

44.6.6 wm8960 adapter

44.6.6.1 Overview

The wm8960 adapter provide codec unify control interface .

Macros

- #define `HAL_CODEC_HANDLER_SIZE` (`WM8960_I2C_HANDLER_SIZE` + 4)
codec handler size

Enumerations

- enum `_codec_type` {
`kCODEC_CS42888`,
`kCODEC_DA7212`,
`kCODEC_WM8904`,
`kCODEC_WM8960`,
`kCODEC_WM8524`,
`kCODEC_SGTL5000`,
`kCODEC_DA7212`,
`kCODEC_CS42888`,
`kCODEC_AK4497`,
`kCODEC_AK4458`,
`kCODEC_TFA9XXX`,
`kCODEC_TFA9896`,
`kCODEC_SGTL5000`,
`kCODEC_WM8904`,
`kCODEC_WM8960` }
codec type

Functions

- `status_t HAL_CODEC_Init` (void *handle, void *config)
Codec initialization.
- `status_t HAL_CODEC_Deinit` (void *handle)
Codec de-initialization.
- `status_t HAL_CODEC_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bit-Width)
set audio data format.
- `status_t HAL_CODEC_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- `status_t HAL_CODEC_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- `status_t HAL_CODEC_SetPower` (void *handle, uint32_t module, bool powerOn)

- set audio codec module power.*
 - [status_t HAL_CODEC_SetRecord](#) (void *handle, uint32_t recordSource)
codec set record source.
 - [status_t HAL_CODEC_SetRecordChannel](#) (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
 - [status_t HAL_CODEC_SetPlay](#) (void *handle, uint32_t playSource)
codec set play source.
 - [status_t HAL_CODEC_ModuleControl](#) (void *handle, uint32_t cmd, uint32_t data)
codec module control.

44.6.6.2 Enumeration Type Documentation

44.6.6.2.1 enum_codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960

44.6.6.3 Function Documentation

44.6.6.3.1 status_t HAL_CODEC_Init (void * *handle*, void * *config*)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

<i>config</i>	codec configuration.
---------------	----------------------

Returns

kStatus_Success is success, else initial failed.

44.6.6.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

44.6.6.3.3 status_t HAL_CODEC_SetFormat (void * *handle*, uint32_t *mclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

44.6.6.3.4 status_t HAL_CODEC_SetVolume (void * *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

wm8960 Driver

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

44.6.6.3.5 status_t HAL_CODEC_SetMute (void * *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

44.6.6.3.6 status_t HAL_CODEC_SetPower (void * *handle*, uint32_t *module*, bool *powerOn*)

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

44.6.6.3.7 status_t HAL_CODEC_SetRecord (void * *handle*, uint32_t *recordSource*)

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

44.6.6.3.8 status_t HAL_CODEC_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

44.6.6.3.9 status_t HAL_CODEC_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

44.6.6.3.10 status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

wm8960 Driver

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference <code>_codec_module_ctrl_cmd</code> .
<i>data</i>	value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations.

Returns

`kStatus_Success` is success, else configure failed.

wm8904 Driver

44.7.1 Overview

The wm8904 driver provide codec control interface.

Modules

- [wm8904 adapter](#)

Data Structures

- struct [wm8904_fl_config_t](#)
wm8904 fl configuration [More...](#)
- struct [wm8904_audio_format_t](#)
Audio format configuration. [More...](#)
- struct [wm8904_config_t](#)
Configuration structure of WM8904. [More...](#)
- struct [wm8904_handle_t](#)
wm8904 codec handler [More...](#)

Macros

- #define [WM8904_I2C_HANDLER_SIZE](#) (CODEC_I2C_MASTER_HANDLER_SIZE)
wm8904 handle size
- #define [WM8904_DEBUG_REGISTER](#) 0
wm8904 debug macro
- #define [WM8904_RESET](#) (0x00)
WM8904 register map.
- #define [WM8904_I2C_ADDRESS](#) (0x1A)
WM8904 I2C address.
- #define [WM8904_I2C_BITRATE](#) (400000U)
WM8904 I2C bit rate.

Enumerations

- enum [_wm8904_status](#) {
 [kStatus_WM8904_Success](#) = 0x0,
 [kStatus_WM8904_Fail](#) = 0x1 }
WM8904 status return codes.
- enum [_wm8904_lrc_polarity](#) {
 [kWM8904_LRCPolarityNormal](#) = 0U,
 [kWM8904_LRCPolarityInverted](#) = 1U << 4U }
WM8904 lrc polarity.

wm8904 Driver

- enum `wm8904_module_t` {
 `kWM8904_ModuleADC` = 0,
 `kWM8904_ModuleDAC` = 1,
 `kWM8904_ModulePGA` = 2,
 `kWM8904_ModuleHeadphone` = 3,
 `kWM8904_ModuleLineout` = 4 }
 wm8904 module value
- enum `_wm8904_play_channel`
 wm8904 play channel
- enum `wm8904_timeslot_t` {
 `kWM8904_TimeSlot0` = 0U,
 `kWM8904_TimeSlot1` = 1U }
 WM8904 time slot.
- enum `wm8904_protocol_t` {
 `kWM8904_ProtocolI2S` = 0x2,
 `kWM8904_ProtocolLeftJustified` = 0x1,
 `kWM8904_ProtocolRightJustified` = 0x0,
 `kWM8904_ProtocolPCMA` = 0x3,
 `kWM8904_ProtocolPCMB` = 0x3 | (1 << 4) }
 The audio data transfer protocol.
- enum `wm8904_fs_ratio_t` {
 `kWM8904_FsRatio64X` = 0x0,
 `kWM8904_FsRatio128X` = 0x1,
 `kWM8904_FsRatio192X` = 0x2,
 `kWM8904_FsRatio256X` = 0x3,
 `kWM8904_FsRatio384X` = 0x4,
 `kWM8904_FsRatio512X` = 0x5,
 `kWM8904_FsRatio768X` = 0x6,
 `kWM8904_FsRatio1024X` = 0x7,
 `kWM8904_FsRatio1408X` = 0x8,
 `kWM8904_FsRatio1536X` = 0x9 }
 The SYSCLK / fs ratio.
- enum `wm8904_sample_rate_t` {
 `kWM8904_SampleRate8kHz` = 0x0,
 `kWM8904_SampleRate12kHz` = 0x1,
 `kWM8904_SampleRate16kHz` = 0x2,
 `kWM8904_SampleRate24kHz` = 0x3,
 `kWM8904_SampleRate32kHz` = 0x4,
 `kWM8904_SampleRate48kHz` = 0x5 }
 Sample rate.
- enum `wm8904_bit_width_t` {
 `kWM8904_BitWidth16` = 0x0,
 `kWM8904_BitWidth20` = 0x1,
 `kWM8904_BitWidth24` = 0x2,
 `kWM8904_BitWidth32` = 0x3 }
 Bit width.

- enum `_wm8904_record_source` {
`kWM8904_RecordSourceDifferentialLine` = 1U,
`kWM8904_RecordSourceLineInput` = 2U,
`kWM8904_RecordSourceDifferentialMic` = 4U,
`kWM8904_RecordSourceDigitalMic` = 8U }
wm8904 record source
- enum `_wm8904_record_channel` {
`kWM8904_RecordChannelLeft1` = 1U,
`kWM8904_RecordChannelLeft2` = 2U,
`kWM8904_RecordChannelLeft3` = 4U,
`kWM8904_RecordChannelRight1` = 1U,
`kWM8904_RecordChannelRight2` = 2U,
`kWM8904_RecordChannelRight3` = 4U,
`kWM8904_RecordChannelDifferentialPositive1` = 1U,
`kWM8904_RecordChannelDifferentialPositive2` = 2U,
`kWM8904_RecordChannelDifferentialPositive3` = 4U,
`kWM8904_RecordChannelDifferentialNegative1` = 8U,
`kWM8904_RecordChannelDifferentialNegative2` = 16U,
`kWM8904_RecordChannelDifferentialNegative3` = 32U }
wm8904 record channel
- enum `_wm8904_play_source` {
`kWM8904_PlaySourcePGA` = 1U,
`kWM8904_PlaySourceDAC` = 4U }
wm8904 play source
- enum `wm8904_sys_clk_source_t` {
`kWM8904_SysClkSourceMCLK` = 0U,
`kWM8904_SysClkSourceFLL` = 1U << 14 }
wm8904 system clock source
- enum `wm8904_flr_clk_source_t` { `kWM8904_FLLClkSourceMCLK` = 0U }
wm8904 flr clock source

Functions

- `status_t WM8904_WriteRegister` (`wm8904_handle_t` *handle, `uint8_t` reg, `uint16_t` value)
WM8904 write register.
- `status_t WM8904_ReadRegister` (`wm8904_handle_t` *handle, `uint8_t` reg, `uint16_t` *value)
WM8904 read register.
- `status_t WM8904_ModifyRegister` (`wm8904_handle_t` *handle, `uint8_t` reg, `uint16_t` mask, `uint16_t` value)
WM8904 modify register.
- `status_t WM8904_Init` (`wm8904_handle_t` *handle, `wm8904_config_t` *wm8904_config)
Initializes WM8904.
- `status_t WM8904_Deinit` (`wm8904_handle_t` *handle)
Deinitializes the WM8904 codec.
- void `WM8904_GetDefaultConfig` (`wm8904_config_t` *config)
Fills the configuration structure with default values.
- `status_t WM8904_SetMasterSlave` (`wm8904_handle_t` *handle, bool master)

wm8904 Driver

- Sets WM8904 as master or slave.*
- `status_t WM8904_SeMasterClock` (`wm8904_handle_t` *handle, `uint32_t` sysclk, `uint32_t` sampleRate, `uint32_t` bitWidth)
Sets WM8904 master clock configuration.
- `status_t WM8904_SetFLLConfig` (`wm8904_handle_t` *handle, `wm8904_fl_config_t` *config)
WM8904 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fl output clock frequency from WM8904 GPIO1.
- `status_t WM8904_SetProtocol` (`wm8904_handle_t` *handle, `wm8904_protocol_t` protocol)
Sets the audio data transfer protocol.
- `status_t WM8904_SetAudioFormat` (`wm8904_handle_t` *handle, `uint32_t` sysclk, `uint32_t` sampleRate, `uint32_t` bitWidth)
Sets the audio data format.
- `status_t WM8904_CheckAudioFormat` (`wm8904_handle_t` *handle, `wm8904_audio_format_t` *format, `uint32_t` mclkFreq)
check and update the audio data format.
- `status_t WM8904_SetVolume` (`wm8904_handle_t` *handle, `uint16_t` volumeLeft, `uint16_t` volumeRight)
Sets the module output volume.
- `status_t WM8904_SetMute` (`wm8904_handle_t` *handle, `bool` muteLeft, `bool` muteRight)
Sets the headphone output mute.
- `status_t WM8904_SelectLRCPolarity` (`wm8904_handle_t` *handle, `uint32_t` polarity)
Select LRC polarity.
- `status_t WM8904_EnableDACTDMMMode` (`wm8904_handle_t` *handle, `wm8904_timeslot_t` timeSlot)
Enable WM8904 DAC time slot.
- `status_t WM8904_EnableADCTDMMMode` (`wm8904_handle_t` *handle, `wm8904_timeslot_t` timeSlot)
Enable WM8904 ADC time slot.
- `status_t WM8904_SetModulePower` (`wm8904_handle_t` *handle, `wm8904_module_t` module, `bool` isEnabled)
brief SET the module output power.
- `status_t WM8904_SetChannelVolume` (`wm8904_handle_t` *handle, `uint32_t` channel, `uint32_t` volume)
Sets the channel output volume.
- `status_t WM8904_SetRecord` (`wm8904_handle_t` *handle, `uint32_t` recordSource)
SET the WM8904 record source.
- `status_t WM8904_SetRecordChannel` (`wm8904_handle_t` *handle, `uint32_t` leftRecordChannel, `uint32_t` rightRecordChannel)
SET the WM8904 record source.
- `status_t WM8904_SetPlay` (`wm8904_handle_t` *handle, `uint32_t` playSource)
SET the WM8904 play source.
- `status_t WM8904_SetChannelMute` (`wm8904_handle_t` *handle, `uint32_t` channel, `bool` isMute)
Sets the channel mute.

Driver version

- `#define FSL_WM8904_DRIVER_VERSION` (`MAKE_VERSION`(2, 4, 1))
WM8904 driver version 2.4.1.

44.7.2 Data Structure Documentation

44.7.2.1 struct wm8904_fll_config_t

Data Fields

- [wm8904_fll_clk_source_t](#) source
fll reference clock source
- [uint32_t](#) [refClock_HZ](#)
fll reference clock frequency
- [uint32_t](#) [outputClock_HZ](#)
fll output clock frequency

44.7.2.2 struct wm8904_audio_format_t

Data Fields

- [wm8904_fs_ratio_t](#) fsRatio
SYSCLK / fs ratio.
- [wm8904_sample_rate_t](#) sampleRate
Sample rate.
- [wm8904_bit_width_t](#) bitWidth
Bit width.

44.7.2.3 struct wm8904_config_t

Data Fields

- [bool](#) [master](#)
Master or slave.
- [wm8904_sys_clk_source_t](#) sysClkSource
system clock source
- [wm8904_fll_config_t](#) * fll
fll configuration
- [wm8904_protocol_t](#) protocol
Audio transfer protocol.
- [wm8904_audio_format_t](#) format
Audio format.
- [uint32_t](#) [mclk_HZ](#)
MCLK frequency value.
- [uint16_t](#) [recordSource](#)
record source
- [uint16_t](#) [recordChannelLeft](#)
record channel
- [uint16_t](#) [recordChannelRight](#)
record channel
- [uint16_t](#) [playSource](#)
play source

wm8904 Driver

- `uint8_t slaveAddress`
code device slave address
- `codec_i2c_config_t i2cConfig`
i2c bus configuration

44.7.2.4 struct wm8904_handle_t

Data Fields

- `wm8904_config_t * config`
wm8904 config pointer
- `uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]`
i2c handle

44.7.3 Macro Definition Documentation

44.7.3.1 `#define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))`

44.7.3.2 `#define WM8904_I2C_ADDRESS (0x1A)`

44.7.3.3 `#define WM8904_I2C_BITRATE (400000U)`

44.7.4 Enumeration Type Documentation

44.7.4.1 enum _wm8904_status

Enumerator

kStatus_WM8904_Success Success.
kStatus_WM8904_Fail Failure.

44.7.4.2 enum _wm8904_lrc_polarity

Enumerator

kWM8904_LRCPolarityNormal LRC polarity normal.
kWM8904_LRCPolarityInverted LRC polarity inverted.

44.7.4.3 enum wm8904_module_t

Enumerator

kWM8904_ModuleADC module ADC

kWM8904_ModuleDAC module DAC
kWM8904_ModulePGA module PGA
kWM8904_ModuleHeadphone module headphone
kWM8904_ModuleLineout module line out

44.7.4.4 enum wm8904_timeslot_t

Enumerator

kWM8904_TimeSlot0 time slot0
kWM8904_TimeSlot1 time slot1

44.7.4.5 enum wm8904_protocol_t

Enumerator

kWM8904_ProtocolI2S I2S type.
kWM8904_ProtocolLeftJustified Left justified mode.
kWM8904_ProtocolRightJustified Right justified mode.
kWM8904_ProtocolPCMA PCM A mode.
kWM8904_ProtocolPCMB PCM B mode.

44.7.4.6 enum wm8904_fs_ratio_t

Enumerator

kWM8904_FsRatio64X SYSCLK is $64 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio128X SYSCLK is $128 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio192X SYSCLK is $192 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio256X SYSCLK is $256 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio384X SYSCLK is $384 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio512X SYSCLK is $512 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio768X SYSCLK is $768 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio1024X SYSCLK is $1024 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio1408X SYSCLK is $1408 * \text{sample rate} * \text{frame width}$.
kWM8904_FsRatio1536X SYSCLK is $1536 * \text{sample rate} * \text{frame width}$.

44.7.4.7 enum wm8904_sample_rate_t

Enumerator

kWM8904_SampleRate8kHz 8 kHz

wm8904 Driver

kWM8904_SampleRate12kHz 11.025kHz, 12kHz
kWM8904_SampleRate16kHz 16kHz
kWM8904_SampleRate24kHz 22.05kHz, 24kHz
kWM8904_SampleRate32kHz 32kHz
kWM8904_SampleRate48kHz 44.1kHz, 48kHz

44.7.4.8 enum wm8904_bit_width_t

Enumerator

kWM8904_BitWidth16 16 bits
kWM8904_BitWidth20 20 bits
kWM8904_BitWidth24 24 bits
kWM8904_BitWidth32 32 bits

44.7.4.9 enum _wm8904_record_source

Enumerator

kWM8904_RecordSourceDifferentialLine record source from differential line
kWM8904_RecordSourceLineInput record source from line input
kWM8904_RecordSourceDifferentialMic record source from differential mic
kWM8904_RecordSourceDigitalMic record source from digital microphone

44.7.4.10 enum _wm8904_record_channel

Enumerator

kWM8904_RecordChannelLeft1 left record channel 1
kWM8904_RecordChannelLeft2 left record channel 2
kWM8904_RecordChannelLeft3 left record channel 3
kWM8904_RecordChannelRight1 right record channel 1
kWM8904_RecordChannelRight2 right record channel 2
kWM8904_RecordChannelRight3 right record channel 3
kWM8904_RecordChannelDifferentialPositive1 differential positive record channel 1
kWM8904_RecordChannelDifferentialPositive2 differential positive record channel 2
kWM8904_RecordChannelDifferentialPositive3 differential positive record channel 3
kWM8904_RecordChannelDifferentialNegative1 differential negative record channel 1
kWM8904_RecordChannelDifferentialNegative2 differential negative record channel 2
kWM8904_RecordChannelDifferentialNegative3 differential negative record channel 3

44.7.4.11 enum _wm8904_play_source

Enumerator

kWM8904_PlaySourcePGA play source PGA, bypass ADC
kWM8904_PlaySourceDAC play source Input3

44.7.4.12 enum wm8904_sys_clk_source_t

Enumerator

kWM8904_SysClkSourceMCLK wm8904 system clock soure from MCLK
kWM8904_SysClkSourceFLL wm8904 system clock soure from FLL

44.7.4.13 enum wm8904_fl_clk_source_t

Enumerator

kWM8904_FLLClkSourceMCLK wm8904 FLL clock source from MCLK

44.7.5 Function Documentation

44.7.5.1 status_t WM8904_WriteRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t *value*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>reg</i>	register address.
<i>value</i>	value to write.

Returns

kStatus_Success, else failed.

44.7.5.2 status_t WM8904_ReadRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t * *value*)

wm8904 Driver

Parameters

<i>handle</i>	WM8904 handle structure.
<i>reg</i>	register address.
<i>value</i>	value to read.

Returns

kStatus_Success, else failed.

44.7.5.3 status_t WM8904_ModifyRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t *mask*, uint16_t *value*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>reg</i>	register address.
<i>mask</i>	register bits mask.
<i>value</i>	value to write.

Returns

kStatus_Success, else failed.

44.7.5.4 status_t WM8904_Init (wm8904_handle_t * *handle*, wm8904_config_t * *wm8904_config*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>wm8904_config</i>	WM8904 configuration structure.

44.7.5.5 status_t WM8904_Deinit (wm8904_handle_t * *handle*)

This function resets WM8904.

Parameters

<i>handle</i>	WM8904 handle structure.
---------------	--------------------------

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.6 void WM8904_GetDefaultConfig (wm8904_config_t * *config*)

The default values are:

master = false; protocol = kWM8904_ProtocolI2S; format.fsRatio = kWM8904_FsRatio64X; format.sampleRate = kWM8904_SampleRate48kHz; format.bitWidth = kWM8904_BitWidth16;

Parameters

<i>config</i>	default configurations of wm8904.
---------------	-----------------------------------

44.7.5.7 status_t WM8904_SetMasterSlave (wm8904_handle_t * *handle*, bool *master*)

Deprecated DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY [WM8904_SeMasterClock](#)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>master</i>	true for master, false for slave.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.8 status_t WM8904_SeMasterClock (wm8904_handle_t * *handle*, uint32_t *sysclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

wm8904 Driver

Parameters

<i>handle</i>	WM8904 handle structure.
<i>sysclk</i>	system clock rate.
<i>sampleRate</i>	sample rate
<i>bitWidth</i>	bit width

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.9 status_t WM8904_SetFLLConfig (wm8904_handle_t * *handle*, wm8904_fll_config_t * *config*)

Parameters

<i>handle</i>	wm8904 handler pointer.
<i>config</i>	FLL configuration pointer.

44.7.5.10 status_t WM8904_SetProtocol (wm8904_handle_t * *handle*, wm8904_protocol_t *protocol*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>protocol</i>	Audio transfer protocol.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.11 status_t WM8904_SetAudioFormat (wm8904_handle_t * *handle*, uint32_t *sysclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>sysclk</i>	System clock frequency for codec, user should pay attention to this parater, sysclk is caculate as $SYSCLK = MCLK / MCLKDIV$, MCLKDIV is bit0 of WM8904_CLK_RATES_0.
<i>sampleRate</i>	Sample rate frequency in Hz.
<i>bitWidth</i>	Audio data bit width.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.12 status_t WM8904_CheckAudioFormat (wm8904_handle_t * *handle*, wm8904_audio_format_t * *format*, uint32_t *mclkFreq*)

This api is used check the fsRatio setting based on the mclk and sample rate, if fsRatio setting is not correct, it will correct it according to mclk and sample rate.

Parameters

<i>handle</i>	WM8904 handle structure.
<i>format</i>	audio data format
<i>mclkFreq</i>	mclk frequency

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.13 status_t WM8904_SetVolume (wm8904_handle_t * *handle*, uint16_t *volumeLeft*, uint16_t *volumeRight*)

The parameter should be from 0 to 100. The resulting volume will be. 0 for mute, 100 for maximum volume value.

Parameters

wm8904 Driver

<i>handle</i>	WM8904 handle structure.
<i>volumeLeft</i>	left channel volume.
<i>volumeRight</i>	right channel volume.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.14 status_t WM8904_SetMute (wm8904_handle_t * *handle*, bool *muteLeft*, bool *muteRight*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>muteLeft</i>	true to mute left channel, false to unmute.
<i>muteRight</i>	true to mute right channel, false to unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.15 status_t WM8904_SelectLRCPolarity (wm8904_handle_t * *handle*, uint32_t *polarity*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>polarity</i>	LRC clock polarity.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.16 status_t WM8904_EnableDACTDMMMode (wm8904_handle_t * *handle*, wm8904_timeslot_t *timeSlot*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>timeSlot</i>	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.17 status_t WM8904_EnableADCTDMMMode (wm8904_handle_t * *handle*, wm8904_timeslot_t *timeSlot*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>timeSlot</i>	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.18 status_t WM8904_SetModulePower (wm8904_handle_t * *handle*, wm8904_module_t *module*, bool *isEnabled*)

param handle WM8904 handle structure. param module wm8904 module. param isEnabled, true is power on, false is power down.

return kStatus_WM8904_Success if successful, different code otherwise..

44.7.5.19 status_t WM8904_SetChannelVolume (wm8904_handle_t * *handle*, uint32_t *channel*, uint32_t *volume*)

The parameter should be from 0 to 100. The resulting volume will be. 0 for mute, 100 for maximum volume value.

Parameters

wm8904 Driver

<i>handle</i>	codec handle structure.
<i>channel</i>	codec channel.
<i>volume</i>	volume value.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.20 status_t WM8904_SetRecord (wm8904_handle_t * *handle*, uint32_t *recordSource*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>recordSource</i>	record source , can be a value of kCODEC_ModuleRecordSourceDifferential-Line, kCODEC_ModuleRecordSourceDifferentialMic, kCODEC_ModuleRecordSourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.5.21 status_t WM8904_SetRecordChannel (wm8904_handle_t * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>leftRecord-Channel</i>	channel number of left record channel when using differential source, channel number of single end left channel when using single end source, channel number of digital mic when using digital mic source.

<i>rightRecord-Channel</i>	channel number of right record channel when using differential source, channel number of single end right channel when using single end source.
----------------------------	---

Returns

kStatus_WM8904_Success if successful, different code otherwise..

44.7.5.22 status_t WM8904_SetPlay (wm8904_handle_t * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	WM8904 handle structure.
<i>playSource</i>	play source , can be a value of kCODEC_ModuleHeadphoneSourcePGA, kCODEC_ModuleHeadphoneSourceDAC, kCODEC_ModuleLineoutSourcePGA, kCODEC_ModuleLineoutSourceDAC.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

44.7.5.23 status_t WM8904_SetChannelMute (wm8904_handle_t * *handle*, uint32_t *channel*, bool *isMute*)

Parameters

<i>handle</i>	codec handle structure.
<i>channel</i>	codec module name.
<i>isMute</i>	true is mute, false unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

44.7.6 wm8904 adapter

44.7.6.1 Overview

The wm8904 adapter provide codec unify control interface .

Macros

- #define [HAL_CODEC_HANDLER_SIZE](#) ([WM8904_I2C_HANDLER_SIZE](#) + 4)
codec handler size

Enumerations

- enum [_codec_type](#) {
 [kCODEC_CS42888](#),
 [kCODEC_DA7212](#),
 [kCODEC_WM8904](#),
 [kCODEC_WM8960](#),
 [kCODEC_WM8524](#),
 [kCODEC_SGTL5000](#),
 [kCODEC_DA7212](#),
 [kCODEC_CS42888](#),
 [kCODEC_AK4497](#),
 [kCODEC_AK4458](#),
 [kCODEC_TFA9XXX](#),
 [kCODEC_TFA9896](#),
 [kCODEC_SGTL5000](#),
 [kCODEC_WM8904](#),
 [kCODEC_WM8960](#) }
codec type

Functions

- [status_t HAL_CODEC_Init](#) (void *handle, void *config)
Codec initialization.
- [status_t HAL_CODEC_Deinit](#) (void *handle)
Codec de-initialization.
- [status_t HAL_CODEC_SetFormat](#) (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bit-Width)
set audio data format.
- [status_t HAL_CODEC_SetVolume](#) (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- [status_t HAL_CODEC_SetMute](#) (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- [status_t HAL_CODEC_SetPower](#) (void *handle, uint32_t module, bool powerOn)

- *set audio codec module power.*
status_t HAL_CODEC_SetRecord (void *handle, uint32_t recordSource)
- *codec set record source.*
status_t HAL_CODEC_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
- *codec set record channel.*
status_t HAL_CODEC_SetPlay (void *handle, uint32_t playSource)
- *codec set play source.*
status_t HAL_CODEC_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
- *codec module control.*

44.7.6.2 Enumeration Type Documentation

44.7.6.2.1 enum_codec_type

Enumerator

kCODEC_CS42888 CS42888.
kCODEC_DA7212 da7212
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896
kCODEC_SGTL5000 sgtl5000
kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960

44.7.6.3 Function Documentation

44.7.6.3.1 status_t HAL_CODEC_Init (void * handle, void * config)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

wm8904 Driver

<i>config</i>	codec configuration.
---------------	----------------------

Returns

kStatus_Success is success, else initial failed.

44.7.6.3.2 status_t HAL_CODEC_Deinit (void * *handle*)

Parameters

<i>handle</i>	codec handle.
---------------	---------------

Returns

kStatus_Success is success, else de-initial failed.

44.7.6.3.3 status_t HAL_CODEC_SetFormat (void * *handle*, uint32_t *mclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

44.7.6.3.4 status_t HAL_CODEC_SetVolume (void * *handle*, uint32_t *playChannel*, uint32_t *volume*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

44.7.6.3.5 status_t HAL_CODEC_SetMute (void * *handle*, uint32_t *playChannel*, bool *isMute*)

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

44.7.6.3.6 status_t HAL_CODEC_SetPower (void * *handle*, uint32_t *module*, bool *powerOn*)

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

44.7.6.3.7 status_t HAL_CODEC_SetRecord (void * *handle*, uint32_t *recordSource*)

wm8904 Driver

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

44.7.6.3.8 status_t HAL_CODEC_SetRecordChannel (void * *handle*, uint32_t *leftRecordChannel*, uint32_t *rightRecordChannel*)

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
<i>rightRecord-Channel</i>	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.

Returns

kStatus_Success is success, else configure failed.

44.7.6.3.9 status_t HAL_CODEC_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

44.7.6.3.10 status_t HAL_CODEC_ModuleControl (void * *handle*, uint32_t *cmd*, uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference <code>_codec_module_ctrl_cmd</code> .
<i>data</i>	value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations.

Returns

`kStatus_Success` is success, else configure failed.

Chapter 45

Serial_Manager

Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are UART, USB CDC and SWO.

Modules

- [Serial Port SWO](#)
- [Serial Port Uart](#)

Data Structures

- struct [serial_manager_config_t](#)
serial manager config structure [More...](#)
- struct [serial_manager_callback_message_t](#)
Callback message structure. [More...](#)

Macros

- #define [SERIAL_MANAGER_NON_BLOCKING_MODE](#) (0U)
Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)
- #define [SERIAL_PORT_TYPE_UART](#) (0U)
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_PORT_TYPE_USBCDC_VIRTUAL](#) (0U)
Enable or disable USB CDC virtual 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.
- #define [SERIAL_MANAGER_HANDLE_DEFINE](#)(name) uint32_t name[(([SERIAL_MANAGER_HANDLE_SIZE](#) + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager handle.
- #define [SERIAL_MANAGER_WRITE_HANDLE_DEFINE](#)(name) uint32_t name[(([SERIAL_MANAGER_WRITE_HANDLE_SIZE](#) + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager write handle.

Overview

- #define [SERIAL_MANAGER_READ_HANDLE_DEFINE](#)(name) uint32_t name[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
Defines the serial manager read handle.
- #define [SERIAL_MANAGER_USE_COMMON_TASK](#) (1U)
Macro to determine whether use common task.
- #define [SERIAL_MANAGER_TASK_PRIORITY](#) (2U)
Macro to set serial manager task priority.
- #define [SERIAL_MANAGER_TASK_STACK_SIZE](#) (1000U)
Macro to set serial manager task stack size.

Typedefs

- typedef void * [serial_handle_t](#)
The handle of the serial manager module.
- typedef void * [serial_write_handle_t](#)
The write handle of the serial manager module.
- typedef void * [serial_read_handle_t](#)
The read handle of the serial manager module.
- typedef void(* [serial_manager_callback_t](#))(void *callbackParam, [serial_manager_callback_message_t](#) *message, [serial_manager_status_t](#) status)
callback function

Enumerations

- enum [serial_port_type_t](#) {
 [kSerialPort_Uart](#) = 1U,
 [kSerialPort_UsbCdc](#),
 [kSerialPort_Swo](#),
 [kSerialPort_UsbCdcVirtual](#) }
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](#),
 [kStatus_SerialManager_NotConnected](#) = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7) }
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.

Data Structure Documentation

45.2.1 struct serial_manager_config_t

Data Fields

- [serial_port_type_t](#) type
Serial port type.
- void * [portConfig](#)
Serial port configuration.

45.2.2 struct serial_manager_callback_message_t

Data Fields

- [uint8_t](#) * [buffer](#)
Transferred buffer.
- [uint32_t](#) [length](#)
Transferred data length.

Macro Definition Documentation

45.3.1 **#define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)**

Definition of serial manager handle size.

45.3.2 **#define SERIAL_MANAGER_HANDLE_DEFINE(*name*) uint32_t
name[(((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) /
sizeof(uint32_t)))]**

This macro is used to define a 4 byte aligned serial manager handle. Then use "(serial_handle_t)name" to get the serial manager handle.

The macro should be global and could be optional. You could also define serial manager handle by yourself.

This is an example,

```
* SERIAL_MANAGER_HANDLE_DEFINE(serialManagerHandle);  
*
```

Parameters

<i>name</i>	The name string of the serial manager handle.
-------------	---

45.3.3 **#define SERIAL_MANAGER_WRITE_HANDLE_DEFINE(*name*) uint32_t
name[(((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) -
1U) / sizeof(uint32_t)))]**

This macro is used to define a 4 byte aligned serial manager write handle. Then use "(serial_write_handle_t)name" to get the serial manager write handle.

The macro should be global and could be optional. You could also define serial manager write handle by yourself.

This is an example,

```
* SERIAL_MANAGER_WRITE_HANDLE_DEFINE(serialManagerwriteHandle);  
*
```

Parameters

<i>name</i>	The name string of the serial manager write handle.
-------------	---

45.3.4 **#define SERIAL_MANAGER_READ_HANDLE_DEFINE(*name*) uint32_t name[(((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]**

This macro is used to define a 4 byte aligned serial manager read handle. Then use "(serial_read_handle_t)name" to get the serial manager read handle.

The macro should be global and could be optional. You could also define serial manager read handle by yourself.

This is an example,

```
* SERIAL_MANAGER_READ_HANDLE_DEFINE(serialManagerReadHandle);
*
```

Parameters

<i>name</i>	The name string of the serial manager read handle.
-------------	--

45.3.5 **#define SERIAL_MANAGER_USE_COMMON_TASK (1U)**

45.3.6 **#define SERIAL_MANAGER_TASK_PRIORITY (2U)**

45.3.7 **#define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)**

Enumeration Type Documentation

45.4.1 **enum serial_port_type_t**

Enumerator

kSerialPort_Uart Serial port UART.

kSerialPort_UsbCdc Serial port USB CDC.

kSerialPort_Swo Serial port SWO.

kSerialPort_UsbCdcVirtual Serial port USB CDC Virtual.

Function Documentation

45.4.2 enum serial_manager_status_t

Enumerator

kStatus_SerialManager_Success Success.
kStatus_SerialManager_Error Failed.
kStatus_SerialManager_Busy Busy.
kStatus_SerialManager_Notify Ring buffer is not empty.
kStatus_SerialManager_Canceled the non-blocking request is canceled
kStatus_SerialManager_HandleConflict The handle is opened.
kStatus_SerialManager_RingBufferOverflow The ring buffer is overflowed.
kStatus_SerialManager_NotConnected The host is not connected.

Function Documentation

45.5.1 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 three types of serial port, UART (includes UART, USART, LPUART, etc), USB CDC and sww. Please refer to [serial_port_type_t](#) for serial port setting. These three types can be set by using [serial_manager_config_t](#).

Example below shows how to use this API to configure the Serial Manager. For UART,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_uart_config_t uartConfig;
* config.type = kSerialPort_Uart;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* uartConfig.instance = 0;
* uartConfig.clockRate = 24000000;
* uartConfig.baudRate = 115200;
* uartConfig.parityMode = kSerialManager_UartParityDisabled;
* uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
* uartConfig.enableRx = 1;
* uartConfig.enableTx = 1;
* config.portConfig = &uartConfig;
* SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

For USB CDC,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_usb_cdc_config_t usbCdcConfig;
```

```

* config.type = kSerialPort_UsbCdc;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* usbCdcConfig.controllerIndex = kSerialManager_UsbControllerKhci0;
* config.portConfig = &usbCdcConfig;
* SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

Parameters

<i>serialHandle</i>	Pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_HANDLE_DEFINE(serialHandle) ; or <code>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<i>config</i>	Pointer to user-defined configuration structure.

Return values

<i>kStatus_SerialManager_Error</i>	An error occurred.
<i>kStatus_SerialManager_Success</i>	The Serial Manager module initialization succeed.

45.5.2 serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)

This function de-initializes the serial manager module instance. If the opened writing or reading handle is not closed, the function will return `kStatus_SerialManager_Busy`.

Parameters

<i>serialHandle</i>	The serial manager module handle pointer.
---------------------	---

Return values

<i>kStatus_SerialManager_Success</i>	The serial manager de-initialization succeed.
--------------------------------------	---

Function Documentation

<i>kStatus_SerialManager_Busy</i>	Opened reading or writing handle is not closed.
-----------------------------------	---

45.5.3 serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_write_handle_t writeHandle)

This function Opens a writing handle for the serial manager module. If the serial manager needs to be used in different tasks, the task should open a dedicated write handle for itself by calling [SerialManager_OpenWriteHandle](#). Since there can only one buffer for transmission for the writing handle at the same time, multiple writing handles need to be opened when the multiple transmission is needed for a task.

Parameters

<i>serialHandle</i>	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
<i>writeHandle</i>	The serial manager module writing handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: SERIAL_MANAGER_WRITE_HANDLE_DEFINE(writeHandle) ; or <code>uint32_t writeHandle[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>

Return values

<i>kStatus_SerialManager_Error</i>	An error occurred.
<i>kStatus_SerialManager_HandleConflict</i>	The writing handle was opened.
<i>kStatus_SerialManager_Success</i>	The writing handle is opened.

Example below shows how to use this API to write data. For task 1,

```
*  static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle1);
*  static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
*  SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
*      , (serial_write_handle_t)s_serialWriteHandle1);
*  SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle1,
*      Task1_SerialManagerTxCallback,
*      s_serialWriteHandle1);
*  SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle1,
*      s_nonBlockingWelcome1,
*      sizeof(s_nonBlockingWelcome1) - 1U);
*
```

For task 2,

```
*  static SERIAL_MANAGER_WRITE_HANDLE_DEFINE(s_serialWriteHandle2);
```



```

*  static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
*  SerialManager_OpenWriteHandle((serial_handle_t)serialHandle
*      , (serial_write_handle_t)s_serialWriteHandle2);
*  SerialManager_InstallTxCallback((serial_write_handle_t)s_serialWriteHandle2,
*      Task2_SerialManagerTxCallback,
*      s_serialWriteHandle2);
*  SerialManager_WriteNonBlocking((serial_write_handle_t)s_serialWriteHandle2,
*      s_nonBlockingWelcome2,
*      sizeof(s_nonBlockingWelcome2) - 1U);
*

```

45.5.4 serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)

This function Closes a writing handle for the serial manager module.

Parameters

<i>writeHandle</i>	The serial manager module writing handle pointer.
--------------------	---

Return values

<i>kStatus_SerialManager_Success</i>	The writing handle is closed.
--------------------------------------	-------------------------------

45.5.5 serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_read_handle_t readHandle)

This function Opens a reading handle for the serial manager module. The reading handle can not be opened multiple at the same time. The error code `kStatus_SerialManager_Busy` would be returned when the previous reading handle is not closed. And there can only be one buffer for receiving for the reading handle at the same time.

Parameters

<i>serialHandle</i>	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
<i>readHandle</i>	The serial manager module reading handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <code>SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle);</code> or <code>uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>

Function Documentation

Return values

<i>kStatus_SerialManager_Error</i>	An error occurred.
<i>kStatus_SerialManager_Success</i>	The reading handle is opened.
<i>kStatus_SerialManager_Busy</i>	Previous reading handle is not closed.

Example below shows how to use this API to read data.

```
*  static SERIAL_MANAGER_READ_HANDLE_DEFINE(s_serialReadHandle);
*  SerialManager_OpenReadHandle((serial_handle_t)serialHandle,
*    (serial_read_handle_t)s_serialReadHandle);
*  static uint8_t s_nonBlockingBuffer[64];
*  SerialManager_InstallRxCallback((serial_read_handle_t)s_serialReadHandle,
*    APP_SerialManagerRxCallback,
*    s_serialReadHandle);
*  SerialManager_ReadNonBlocking((serial_read_handle_t)s_serialReadHandle,
*    s_nonBlockingBuffer,
*    sizeof(s_nonBlockingBuffer));
*
```

45.5.6 serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t *readHandle*)

This function Closes a reading for the serial manager module.

Parameters

<i>readHandle</i>	The serial manager module reading handle pointer.
-------------------	---

Return values

<i>kStatus_SerialManager_Success</i>	The reading handle is closed.
--------------------------------------	-------------------------------

45.5.7 serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t *writeHandle*, uint8_t * *buffer*, uint32_t *length*)

This is a blocking function, which polls the sending queue, waits for the sending queue to be empty. This function sends data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for transmission for the writing handle at the same time.

Note

The function [SerialManager_WriteBlocking](#) and the function `SerialManager_WriteNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelWriting` cannot be used to abort the transmission of this function.

Parameters

<i>writeHandle</i>	The serial manager module handle pointer.
<i>buffer</i>	Start address of the data to write.
<i>length</i>	Length of the data to write.

Return values

<i>kStatus_SerialManager_- Success</i>	Successfully sent all data.
<i>kStatus_SerialManager_- Busy</i>	Previous transmission still not finished; data not all sent yet.
<i>kStatus_SerialManager_- Error</i>	An error occurred.

45.5.8 serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t * buffer, uint32_t length)

This is a blocking function, which polls the receiving buffer, waits for the receiving buffer to be full. This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

Note

The function [SerialManager_ReadBlocking](#) and the function `SerialManager_ReadNonBlocking` cannot be used at the same time. And, the function `SerialManager_CancelReading` cannot be used to abort the transmission of this function.

Parameters

Function Documentation

<i>readHandle</i>	The serial manager module handle pointer.
<i>buffer</i>	Start address of the data to store the received data.
<i>length</i>	The length of the data to be received.

Return values

<i>kStatus_SerialManager_- Success</i>	Successfully received all data.
<i>kStatus_SerialManager_- Busy</i>	Previous transmission still not finished; data not all received yet.
<i>kStatus_SerialManager_- Error</i>	An error occurred.

45.5.9 serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)

This function is used to prepare to enter low power consumption.

Parameters

<i>serialHandle</i>	The serial manager module handle pointer.
---------------------	---

Return values

<i>kStatus_SerialManager_- Success</i>	Successful operation.
--	-----------------------

45.5.10 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.
--	-----------------------

Serial Port Uart

Serial Port Uart

45.6.1 Overview

Data Structures

- struct [serial_port_uart_config_t](#)
serial port uart config struct [More...](#)

Macros

- #define [SERIAL_PORT_UART_HANDLE_SIZE](#) (HAL_UART_HANDLE_SIZE)
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

45.6.2 Data Structure Documentation

45.6.2.1 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](#)

Enable TX.

45.6.2.1.0.1 Field Documentation

45.6.2.1.0.1.1 `uint8_t serial_port_uart_config_t::instance`

45.6.3 Enumeration Type Documentation

45.6.3.1 `enum serial_port_uart_parity_mode_t`

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

45.6.3.2 `enum serial_port_uart_stop_bit_count_t`

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

Serial Port SWO

Serial Port SWO

45.7.1 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

45.7.2 Data Structure Documentation

45.7.2.1 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.

45.7.3 Enumeration Type Documentation

45.7.3.1 enum serial_port_swo_protocol_t

Enumerator

kSerialManager_SwoProtocolManchester SWO Manchester protocol.
kSerialManager_SwoProtocolNrz SWO UART/NRZ protocol.

Chapter 46

Cache

Overview

Macros

- #define `L1CODEBUSCACHE_LINESIZE_BYTE` `FSL_FEATURE_L1ICACHE_LINESIZE_BYTE`
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, 3))`
cache driver version 2.0.3.

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.
- 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)

Macro Definition Documentation

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

Macro Definition Documentation

46.2.1 #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

46.2.2 #define L1CODEBUSCACHE_LINESIZE_BYTE FSL_FEATURE_L1ICACHE_LINESIZE_BYTE

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

46.2.3 #define L1SYSTEMBUSCACHE_LINESIZE_BYTE L1CODEBUSCACHE_LINESIZE_BYTE

Function Documentation

46.3.1 void L1CACHE_InvalidateCodeCacheByRange (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 invalidated.

Note

Address and size should be aligned to "L1CODCACHE_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.

46.3.2 void L1CACHE_CleanCodeCacheByRange (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 "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.

46.3.3 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.

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

Function Documentation

Parameters

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

46.3.5 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.

46.3.6 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.

46.3.7 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.

46.3.8 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.
---------------	---

46.3.9 void L1CACHE_InvalidateICacheByRange (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_BYTE) aligned.

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

Function Documentation

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_BYTE) aligned.

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

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_BYTE) aligned.

46.3.12 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_BYTE) aligned.

46.3.13 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_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.

46.3.14 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.

46.3.15 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.

46.3.16 `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.

Chapter 47

Data Structure Documentation

47.0.17 codec_i2c_config_t Struct Reference

CODEC I2C configurations structure.

```
#include <fsl_codec_i2c.h>
```

Data Fields

- uint32_t [codecI2CInstance](#)
i2c bus instance
- uint32_t [codecI2CSourceClock](#)
i2c bus source clock frequency



How to Reach Us:**Home Page:**nxp.com**Web Support:**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.

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.

© 2018 NXP B.V.