Document Number: MCUXSDKAPIRM
Rev 2.11.0
Jan 2022

MCUXpresso SDK API Reference Manual

NXP Semiconductors

Contents

Chapter 1 Introduction

Chapter 2 Trademarks

Chapter 3 Architectural Overview

Chapter 4 Clock Driver

4.1 Overview **7**

4.2 Data Structure Documentation **24**
4.2.1 struct clock_sys_pll_config_t 24
4.2.2 struct clock_audio_pll_config_t 24
4.2.3 struct clock_frg_clk_config_t 25

4.3 Macro Definition Documentation **25**
4.3.1 MIPI_DSI_HOST_CLOCKS 25
4.3.2 LCDIF_CLOCKS 25
4.3.3 SCT_CLOCKS 26
4.3.4 USBD_CLOCKS 26
4.3.5 LPADC_CLOCKS 26
4.3.6 CMP_CLOCKS 26
4.3.7 WWDT_CLOCKS 26
4.3.8 UTICK_CLOCKS 27
4.3.9 FLEXIO_CLOCKS 27
4.3.10 FLEXCOMM_CLOCKS 27
4.3.11 USART_CLOCKS 27
4.3.12 I2C_CLOCKS 28
4.3.13 SPI_CLOCKS 28
4.3.14 I2S_CLOCKS 28
4.3.15 DMIC_CLOCKS 28
4.3.16 DMA_CLOCKS 29
4.3.17 CRC_CLOCKS 29
4.3.18 GPIO_CLOCKS 29
4.3.19 PINT_CLOCKS 29
4.3.20 I3C_CLOCKS 29
4.3.21 MRT_CLOCKS 30
4.3.22 RTC_CLOCKS 30

4.3.23 CTIMER_CLOCKS ... 30
4.3.24 CLK_GATE_REG_OFFSET_SHIFT ... 30
4.3.25 SYSPLL0CLKSEL_OFFSET ... 30

4.4 Enumeration Type Documentation ... 30
4.4.1 clock_ip_name_t ... 30
4.4.2 clock_name_t ... 34
4.4.3 clock_pfd_t ... 35
4.4.4 clock_attach_id_t ... 35
4.4.5 clock_div_name_t ... 41
4.4.6 sys_pll_src_t ... 41
4.4.7 sys_pll_mult_t ... 42
4.4.8 audio_pll_src_t ... 42
4.4.9 audio_pll_mult_t ... 42
4.4.10 clock_fro_output_en_t ... 42
4.4.11 clock_fro_freq_t ... 43

4.5 Function Documentation ... 43
4.5.1 CLOCK_AttachClk ... 43
4.5.2 CLOCK_SetClkDiv ... 43
4.5.3 CLOCK_GetFreq ... 43
4.5.4 CLOCK_GetFRGClock ... 44
4.5.5 CLOCK_SetFRGClock ... 44
4.5.6 CLOCK_GetSysPllFreq ... 44
4.5.7 CLOCK_GetSysPfdFreq ... 44
4.5.8 CLOCK_GetAudioPllFreq ... 44
4.5.9 CLOCK_GetAudioPfdFreq ... 44
4.5.10 CLOCK_GetMainClkFreq ... 45
4.5.11 CLOCK_GetDspMainClkFreq ... 45
4.5.12 CLOCK_GetAcmpClkFreq ... 45
4.5.13 CLOCK_GetDmicClkFreq ... 45
4.5.14 CLOCK_GetUsbClkFreq ... 45
4.5.15 CLOCK_GetSdioClkFreq ... 45
4.5.16 CLOCK_GetI3cClkFreq ... 46
4.5.17 CLOCK_GetSystickClkFreq ... 46
4.5.18 CLOCK_GetWdtClkFreq ... 46
4.5.19 CLOCK_GetMclkClkFreq ... 46
4.5.20 CLOCK_GetSctClkFreq ... 47
4.5.21 CLOCK_EnableSysOscClk ... 47
4.5.22 CLOCK_EnableFroClk ... 47
4.5.23 CLOCK_EnableFroClkRange ... 47
4.5.24 CLOCK_GetXtalInClkFreq ... 47
4.5.25 CLOCK_GetMclkInClkFreq ... 48
4.5.26 CLOCK_GetLpOscFreq ... 48
4.5.27 CLOCK_GetOsc32KFreq ... 48

4.5.28 CLOCK_EnableOsc32K ... 48
4.5.29 CLOCK_GetWakeClk32KFreq ... 48
4.5.30 CLOCK_SetXtalFreq ... 48
4.5.31 CLOCK_SetClkinFreq ... 49
4.5.32 CLOCK_SetMclkFreq ... 49
4.5.33 CLOCK_GetFlexcommClkFreq ... 49
4.5.34 CLOCK_GetFlexioClkFreq ... 49
4.5.35 CLOCK_GetCtimerClkFreq ... 49
4.5.36 CLOCK_GetClockOutClkFreq ... 50
4.5.37 CLOCK_GetAdcClkFreq ... 50
4.5.38 CLOCK_GetFlexspiClkFreq ... 50
4.5.39 CLOCK_GetGpuClkFreq ... 50
4.5.40 CLOCK_GetDcPixelClkFreq ... 51
4.5.41 CLOCK_GetMipiDphyClkFreq ... 51
4.5.42 CLOCK_GetMipiDphyEscRxClkFreq ... 51
4.5.43 CLOCK_GetMipiDphyEscTxClkFreq ... 51
4.5.44 CLOCK_InitSysPll ... 51
4.5.45 CLOCK_DeinitSysPll ... 51
4.5.46 CLOCK_InitSysPfd ... 51
4.5.47 CLOCK_DeinitSysPfd ... 52
4.5.48 CLOCK_InitAudioPll ... 52
4.5.49 CLOCK_DeinitAudioPll ... 52
4.5.50 CLOCK_InitAudioPfd ... 52
4.5.51 CLOCK_DeinitAudioPfd ... 52
4.5.52 CLOCK_EnableFroTuning ... 53
4.5.53 CLOCK_EnableUsbHs0DeviceClock ... 53
4.5.54 CLOCK_DisableUsbHs0DeviceClock ... 53
4.5.55 CLOCK_EnableUsbHs0HostClock ... 53
4.5.56 CLOCK_DisableUsbHs0HostClock ... 53
4.5.57 CLOCK_EnableUsbHs0PhyPllClock ... 53
4.5.58 CLOCK_DisableUsbHs0PhyPllClock ... 53

4.6 Variable Documentation ... 53
4.6.1 g_xtalFreq ... 53
4.6.2 g_clkinFreq ... 54
4.6.3 g_mclkFreq ... 54

Chapter 5 DSP Driver

5.1 Overview ... 55

5.2 Data Structure Documentation ... 55
5.2.1 struct dsp_copy_image_t ... 55

5.3 Macro Definition Documentation ... 56

Section No. Title Page No.

5.3.1 FSL_DSP_DRIVER_VERSION ... 56

5.4 Function Documentation ... 56
5.4.1 DSP_Init ... 56
5.4.2 DSP_SetVecRemap ... 56
5.4.3 DSP_CopyImage ... 56

Chapter 6 Power Driver

6.1 Overview ... 57

6.2 Data Structure Documentation ... 60
6.2.1 struct power_pad_vrange_t ... 60

6.3 Macro Definition Documentation ... 61
6.3.1 FSL_POWER_DRIVER_VERSION ... 61
6.3.2 PMIC_VDDCORE_RECOVERY_TIME_IGNORE ... 61
6.3.3 POWER_INVALID_VOLT_LEVEL ... 61
6.3.4 POWER_FREQ_LEVELS_NUM ... 61

6.4 Enumeration Type Documentation ... 61
6.4.1 _pmc_interrupt ... 61
6.4.2 _pmc_event_flags ... 61
6.4.3 power_deep_sleep_clk_t ... 62
6.4.4 power_pad_vrange_val_t ... 62
6.4.5 power_lvd_falling_trip_vol_val_t ... 62

6.5 Function Documentation ... 63
6.5.1 POWER_EnablePD ... 63
6.5.2 POWER_DisablePD ... 63
6.5.3 POWER_UpdateOscSettlingTime ... 63
6.5.4 POWER_UpdatePmicRecoveryTime ... 63
6.5.5 POWER_ClearEventFlags ... 64
6.5.6 POWER_GetEventFlags ... 64
6.5.7 POWER_EnableInterrupts ... 64
6.5.8 POWER_DisableInterrupts ... 64
6.5.9 POWER_SetAnalogBuffer ... 64
6.5.10 POWER_GetPmicMode ... 65
6.5.11 POWER_GetBodyBiasMode ... 65
6.5.12 POWER_SetPadVolRange ... 65
6.5.13 POWER_SetLdoVoltageForFreq ... 66
6.5.14 POWER_SetLvdFallingTripVoltage ... 66
6.5.15 POWER_GetLvdFallingTripVoltage ... 66
6.5.16 POWER_SetPmicMode ... 66
6.5.17 POWER_SetDeepSleepClock ... 67

6.5.18 POWER_EnterDeepSleep ... 67
6.5.19 POWER_EnterDeepPowerDown ... 67
6.5.20 POWER_EnterFullDeepPowerDown ... 67
6.5.21 POWER_EnterPowerMode ... 68
6.5.22 EnableDeepSleepIRQ ... 68
6.5.23 DisableDeepSleepIRQ ... 68
6.5.24 POWER_GetLibVersion ... 69

6.6 Variable Documentation ... 69
6.6.1 powerFreqLevel ... 69

Chapter 7 Reset Driver

7.1 Overview ... 70

7.2 Macro Definition Documentation ... 72
7.2.1 FSL_RESET_DRIVER_VERSION ... 72
7.2.2 ADC_RSTS ... 72

7.3 Enumeration Type Documentation ... 72
7.3.1 RSTCTL_RSTn_t ... 72

7.4 Function Documentation ... 74
7.4.1 RESET_SetPeripheralReset ... 74
7.4.2 RESET_ClearPeripheralReset ... 74
7.4.3 RESET_PeripheralReset ... 74

Chapter 8 ACMP: Analog Comparator Driver

8.1 Overview ... 76

8.2 Typical use case ... 76
8.2.1 Normal Configuration ... 76
8.2.2 Interrupt Configuration ... 76
8.2.3 Round robin Configuration ... 76

8.3 Data Structure Documentation ... 79
8.3.1 struct acmp_config_t ... 79
8.3.2 struct acmp_channel_config_t ... 80
8.3.3 struct acmp_filter_config_t ... 80
8.3.4 struct acmp_dac_config_t ... 81
8.3.5 struct acmp_round_robin_config_t ... 81
8.3.6 struct acmp_discrete_mode_config_t ... 82

8.4 Macro Definition Documentation ... 82
8.4.1 FSL_ACMP_DRIVER_VERSION ... 82

8.4.2 CMP_C0_CFx_MASK ... 82

8.5 Enumeration Type Documentation ... 82
8.5.1 _acmp_interrupt_enable ... 83
8.5.2 _acmp_status_flags ... 83
8.5.3 acmp_hysteresis_mode_t ... 83
8.5.4 acmp_reference_voltage_source_t ... 83
8.5.5 acmp_fixed_port_t ... 83
8.5.6 acmp_dac_work_mode_t ... 84
8.5.7 acmp_discrete_clock_source_t ... 84
8.5.8 acmp_discrete_sample_time_t ... 84
8.5.9 acmp_discrete_phase_time_t ... 84

8.6 Function Documentation ... 85
8.6.1 ACMP_Init ... 85
8.6.2 ACMP_Deinit ... 85
8.6.3 ACMP_GetDefaultConfig ... 85
8.6.4 ACMP_Enable ... 85
8.6.5 ACMP_EnableLinkToDAC ... 86
8.6.6 ACMP_SetChannelConfig ... 86
8.6.7 ACMP_EnableDMA ... 86
8.6.8 ACMP_EnableWindowMode ... 87
8.6.9 ACMP_SetFilterConfig ... 87
8.6.10 ACMP_SetDACConfig ... 87
8.6.11 ACMP_SetRoundRobinConfig ... 88
8.6.12 ACMP_SetRoundRobinPreState ... 88
8.6.13 ACMP_GetRoundRobinStatusFlags ... 88
8.6.14 ACMP_ClearRoundRobinStatusFlags ... 89
8.6.15 ACMP_GetRoundRobinResult ... 89
8.6.16 ACMP_EnableInterrupts ... 89
8.6.17 ACMP_DisableInterrupts ... 90
8.6.18 ACMP_GetStatusFlags ... 90
8.6.19 ACMP_ClearStatusFlags ... 90
8.6.20 ACMP_SetDiscreteModeConfig ... 90
8.6.21 ACMP_GetDefaultDiscreteModeConfig ... 91

Chapter 9 CACHE: CACHE Memory Controller

9.1 Overview ... 92

9.2 Function groups ... 92
9.2.1 CACHE Operation ... 92

9.3 Data Structure Documentation ... 93
9.3.1 struct cache64_config_t ... 94

Section No. Title Page No.

9.4 Macro Definition Documentation ... **94**
9.4.1 FSL_CACHE_DRIVER_VERSION ... 94
9.4.2 CACHE64_LINESIZE_BYTE ... 94
9.4.3 CACHE64_REGION_NUM ... 94
9.4.4 CACHE64_REGION_ALIGNMENT ... 94

9.5 Enumeration Type Documentation ... **94**
9.5.1 cache64_policy_t ... 94

9.6 Function Documentation ... **94**
9.6.1 CACHE64_GetInstance ... 94
9.6.2 CACHE64_GetInstanceByAddr ... 95
9.6.3 CACHE64_Init ... 95
9.6.4 CACHE64_GetDefaultConfig ... 95
9.6.5 CACHE64_EnableCache ... 96
9.6.6 CACHE64_DisableCache ... 96
9.6.7 CACHE64_InvalidateCache ... 96
9.6.8 CACHE64_InvalidateCacheByRange ... 96
9.6.9 CACHE64_CleanCache ... 97
9.6.10 CACHE64_CleanCacheByRange ... 98
9.6.11 CACHE64_CleanInvalidateCache ... 98
9.6.12 CACHE64_CleanInvalidateCacheByRange ... 98
9.6.13 CACHE64_EnableWriteBuffer ... 99
9.6.14 ICACHE_InvalidateByRange ... 100
9.6.15 DCACHE_InvalidateByRange ... 100
9.6.16 DCACHE_CleanByRange ... 100
9.6.17 DCACHE_CleanInvalidateByRange ... 101

Chapter 10 CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing

10.1 Overview ... **102**

10.2 CASPER Driver Initialization and deinitialization ... **102**

10.3 Comments about API usage in RTOS ... **102**

10.4 Comments about API usage in interrupt handler ... **102**

10.5 CASPER Driver Examples ... **102**
10.5.1 Simple examples ... 102

10.6 casper_driver ... **104**
10.6.1 Overview ... 104

10.6.2 Macro Definition Documentation ... 104
10.6.3 Enumeration Type Documentation ... 106
10.6.4 Function Documentation ... 106

10.7 casper_driver_pkha ... 108
10.7.1 Overview ... 108
10.7.2 Function Documentation ... 108

Chapter 11 Common Driver

11.1 Overview ... 114

11.2 Macro Definition Documentation ... 116
11.2.1 FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ ... 116
11.2.2 MAKE_STATUS ... 116
11.2.3 MAKE_VERSION ... 117
11.2.4 FSL_COMMON_DRIVER_VERSION ... 117
11.2.5 DEBUG_CONSOLE_DEVICE_TYPE_NONE ... 117
11.2.6 DEBUG_CONSOLE_DEVICE_TYPE_UART ... 117
11.2.7 DEBUG_CONSOLE_DEVICE_TYPE_LPUART ... 117
11.2.8 DEBUG_CONSOLE_DEVICE_TYPE_LPSCI ... 117
11.2.9 DEBUG_CONSOLE_DEVICE_TYPE_USBCDC ... 117
11.2.10 DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM ... 117
11.2.11 DEBUG_CONSOLE_DEVICE_TYPE_IUART ... 117
11.2.12 DEBUG_CONSOLE_DEVICE_TYPE_VUSART ... 117
11.2.13 DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART ... 117
11.2.14 DEBUG_CONSOLE_DEVICE_TYPE_SWO ... 117
11.2.15 DEBUG_CONSOLE_DEVICE_TYPE_QSCI ... 117
11.2.16 ARRAY_SIZE ... 117

11.3 Typedef Documentation ... 117
11.3.1 status_t ... 117

11.4 Enumeration Type Documentation ... 118
11.4.1 _status_groups ... 118
11.4.2 anonymous enum ... 120

11.5 Function Documentation ... 120
11.5.1 SDK_Malloc ... 121
11.5.2 SDK_Free ... 122
11.5.3 SDK_DelayAtLeastUs ... 122

Chapter 12 CTIMER: Standard counter/timers

12.1 Overview ... 123

Section No.	Title	Page No.

12.2 Function groups **123**
12.2.1 Initialization and deinitialization 123
12.2.2 PWM Operations 123
12.2.3 Match Operation 123
12.2.4 Input capture operations 123

12.3 Typical use case **124**
12.3.1 Match example 124
12.3.2 PWM output example 124

12.4 Data Structure Documentation **127**
12.4.1 struct ctimer_match_config_t 127
12.4.2 struct ctimer_config_t 127

12.5 Enumeration Type Documentation **128**
12.5.1 ctimer_capture_channel_t 128
12.5.2 ctimer_capture_edge_t 128
12.5.3 ctimer_match_t 128
12.5.4 ctimer_external_match_t 128
12.5.5 ctimer_match_output_control_t 128
12.5.6 ctimer_interrupt_enable_t 129
12.5.7 ctimer_status_flags_t 129
12.5.8 ctimer_callback_type_t 129

12.6 Function Documentation **130**
12.6.1 CTIMER_Init 130
12.6.2 CTIMER_Deinit 130
12.6.3 CTIMER_GetDefaultConfig 130
12.6.4 CTIMER_SetupPwmPeriod 130
12.6.5 CTIMER_SetupPwm 131
12.6.6 CTIMER_UpdatePwmPulsePeriod 132
12.6.7 CTIMER_UpdatePwmDutycycle 132
12.6.8 CTIMER_SetupMatch 133
12.6.9 CTIMER_GetOutputMatchStatus 133
12.6.10 CTIMER_SetupCapture 133
12.6.11 CTIMER_GetTimerCountValue 134
12.6.12 CTIMER_RegisterCallBack 134
12.6.13 CTIMER_EnableInterrupts 134
12.6.14 CTIMER_DisableInterrupts 135
12.6.15 CTIMER_GetEnabledInterrupts 135
12.6.16 CTIMER_GetStatusFlags 135
12.6.17 CTIMER_ClearStatusFlags 136
12.6.18 CTIMER_StartTimer 137
12.6.19 CTIMER_StopTimer 137
12.6.20 CTIMER_Reset 137

Chapter 13 DMIC: Digital Microphone

13.1 Overview **138**

13.2 Function groups **138**
13.2.1 Initialization and deinitialization 138
13.2.2 Configuration 138
13.2.3 DMIC Data and status 138
13.2.4 DMIC Interrupt Functions 138
13.2.5 DMIC HWVAD Functions 139
13.2.6 DMIC HWVAD Interrupt Functions 139

13.3 Typical use case **139**
13.3.1 DMIC DMA Configuration 139
13.3.2 DMIC use case 139

13.4 DMIC Driver **140**
13.4.1 Overview 140
13.4.2 Data Structure Documentation 144
13.4.3 Macro Definition Documentation 144
13.4.4 Typedef Documentation 144
13.4.5 Enumeration Type Documentation 145
13.4.6 Function Documentation 147

13.5 DMIC DMA Driver **158**
13.5.1 Overview 158
13.5.2 Data Structure Documentation 159
13.5.3 Typedef Documentation 160
13.5.4 Function Documentation 160

Chapter 14 FLEXCOMM: FLEXCOMM Driver

14.1 Overview **164**

14.2 FLEXCOMM Driver **165**
14.2.1 Overview 165
14.2.2 Macro Definition Documentation 165
14.2.3 Typedef Documentation 166
14.2.4 Enumeration Type Documentation 166
14.2.5 Function Documentation 166
14.2.6 Variable Documentation 166

Chapter 15 I2C: Inter-Integrated Circuit Driver

15.1 Overview **167**

Section No. Title Page No.

15.2 Typical use case ... **167**
15.2.1 Master Operation in functional method ... 167
15.2.2 Master Operation in interrupt transactional method ... 168
15.2.3 Master Operation in DMA transactional method ... 169
15.2.4 Slave Operation in functional method ... 169
15.2.5 Slave Operation in interrupt transactional method ... 170

15.3 I2C Driver ... **172**
15.3.1 Overview ... 172
15.3.2 Macro Definition Documentation ... 174
15.3.3 Enumeration Type Documentation ... 174

15.4 I2C Master Driver ... **177**
15.4.1 Overview ... 177
15.4.2 Data Structure Documentation ... 179
15.4.3 Typedef Documentation ... 181
15.4.4 Enumeration Type Documentation ... 183
15.4.5 Function Documentation ... 183

15.5 I2C Slave Driver ... **195**
15.5.1 Overview ... 195
15.5.2 Data Structure Documentation ... 197
15.5.3 Typedef Documentation ... 200
15.5.4 Enumeration Type Documentation ... 200
15.5.5 Function Documentation ... 201

15.6 I2C DMA Driver ... **210**
15.6.1 Overview ... 210
15.6.2 Data Structure Documentation ... 211
15.6.3 Macro Definition Documentation ... 211
15.6.4 Typedef Documentation ... 212
15.6.5 Function Documentation ... 212

15.7 I2C FreeRTOS Driver ... **214**
15.7.1 Overview ... 214
15.7.2 Data Structure Documentation ... 214
15.7.3 Macro Definition Documentation ... 214
15.7.4 Function Documentation ... 215

15.8 I2C CMSIS Driver ... **216**
15.8.1 I2C CMSIS Driver ... 216

Chapter 16 I2S: I2S Driver

16.1 Overview ... **218**

16.2 I2S Driver Initialization and Configuration **218**

16.3 I2S Transmit Data **218**

16.4 I2S Interrupt related functions **219**

16.5 I2S Other functions **219**

16.6 I2S Data formats **219**
16.6.1 DMA mode 219
16.6.2 Interrupt mode 222

16.7 I2S Driver Examples **223**
16.7.1 Interrupt mode examples 223
16.7.2 DMA mode examples 224

16.8 I2S Driver **226**
16.8.1 Overview 226
16.8.2 Data Structure Documentation 228
16.8.3 Macro Definition Documentation 230
16.8.4 Typedef Documentation 230
16.8.5 Enumeration Type Documentation 231
16.8.6 Function Documentation 232

Chapter 17 SPI: Serial Peripheral Interface Driver

17.1 Overview **242**

17.2 Typical use case **242**
17.2.1 SPI master transfer using an interrupt method 242
17.2.2 SPI Send/receive using a DMA method 243

17.3 SPI Driver **245**
17.3.1 Overview 245
17.3.2 Data Structure Documentation 248
17.3.3 Macro Definition Documentation 251
17.3.4 Typedef Documentation 251
17.3.5 Enumeration Type Documentation 252
17.3.6 Variable Documentation 254

17.4 SPI DMA Driver **255**
17.4.1 Overview 255
17.4.2 Data Structure Documentation 256
17.4.3 Macro Definition Documentation 256
17.4.4 Typedef Documentation 256
17.4.5 Function Documentation 257

Section No. Title Page No.

17.5 SPI FreeRTOS driver **262**
17.5.1 Overview 262
17.5.2 Data Structure Documentation 262
17.5.3 Macro Definition Documentation 263
17.5.4 Function Documentation 263

17.6 SPI CMSIS driver **265**
17.6.1 Function groups 265
17.6.2 Typical use case 266

Chapter 18 USART: Universal Synchronous/Asynchronous Receiver/Transmitter Driver

18.1 Overview **267**

18.2 Typical use case **268**
18.2.1 USART Send/receive using a polling method 268
18.2.2 USART Send/receive using an interrupt method 268
18.2.3 USART Receive using the ringbuffer feature 269
18.2.4 USART Send/Receive using the DMA method 270

18.3 USART Driver **272**
18.3.1 Overview 272
18.3.2 Data Structure Documentation 276
18.3.3 Macro Definition Documentation 279
18.3.4 Typedef Documentation 279
18.3.5 Enumeration Type Documentation 279
18.3.6 Function Documentation 281

18.4 USART DMA Driver **297**
18.4.1 Overview 297
18.4.2 Data Structure Documentation 298
18.4.3 Macro Definition Documentation 298
18.4.4 Typedef Documentation 299
18.4.5 Function Documentation 299

18.5 USART FreeRTOS Driver **303**
18.5.1 Overview 303
18.5.2 Data Structure Documentation 303
18.5.3 Macro Definition Documentation 304
18.5.4 Function Documentation 304

18.6 USART CMSIS Driver **307**
18.6.1 USART Send Methods 307

Section No.	Title	Page No.

Chapter 19 FlexIO: FlexIO Driver

19.1 Overview **309**

19.2 FlexIO Driver **310**
19.2.1 Overview 310
19.2.2 Data Structure Documentation 314
19.2.3 Macro Definition Documentation 317
19.2.4 Typedef Documentation 317
19.2.5 Enumeration Type Documentation 317
19.2.6 Function Documentation 322
19.2.7 Variable Documentation 332

19.3 FlexIO I2C Master Driver **333**
19.3.1 Overview 333
19.3.2 Typical use case 333
19.3.3 Data Structure Documentation 337
19.3.4 Macro Definition Documentation 339
19.3.5 Typedef Documentation 339
19.3.6 Enumeration Type Documentation 340
19.3.7 Function Documentation 340

19.4 FlexIO MCU Interface LCD Driver **350**
19.4.1 Overview 350
19.4.2 Typical use case 350
19.4.3 Data Structure Documentation 356
19.4.4 Macro Definition Documentation 359
19.4.5 Typedef Documentation 360
19.4.6 Enumeration Type Documentation 360
19.4.7 Function Documentation 361
19.4.8 FlexIO DMA MCU Interface LCD Driver 374
19.4.9 FlexIO SMARTDMA MCU Interface LCD Driver 379

19.5 FlexIO SPI Driver **384**
19.5.1 Overview 384
19.5.2 Typical use case 384
19.5.3 Data Structure Documentation 390
19.5.4 Macro Definition Documentation 394
19.5.5 Typedef Documentation 394
19.5.6 Enumeration Type Documentation 394
19.5.7 Function Documentation 396

19.6 FlexIO UART Driver **409**
19.6.1 Overview 409
19.6.2 Typical use case 409
19.6.3 Data Structure Documentation 417

19.6.4 Macro Definition Documentation ... 420
19.6.5 Typedef Documentation ... 420
19.6.6 Enumeration Type Documentation ... 421
19.6.7 Function Documentation ... 421

Chapter 20 FLEXSPI: Flexible Serial Peripheral Interface Driver

20.1 Overview ... 433

20.2 Data Structure Documentation ... 438
20.2.1 struct flexspi_config_t ... 439
20.2.2 struct flexspi_device_config_t ... 441
20.2.3 struct flexspi_transfer_t ... 442
20.2.4 struct _flexspi_handle ... 443

20.3 Macro Definition Documentation ... 443
20.3.1 FSL_FLEXSPI_DRIVER_VERSION ... 443
20.3.2 FLEXSPI_LUT_SEQ ... 443

20.4 Typedef Documentation ... 444
20.4.1 flexspi_transfer_callback_t ... 444

20.5 Enumeration Type Documentation ... 444
20.5.1 anonymous enum ... 444
20.5.2 anonymous enum ... 444
20.5.3 flexspi_pad_t ... 445
20.5.4 flexspi_flags_t ... 445
20.5.5 flexspi_read_sample_clock_t ... 446
20.5.6 flexspi_cs_interval_cycle_unit_t ... 446
20.5.7 flexspi_ahb_write_wait_unit_t ... 446
20.5.8 flexspi_ip_error_code_t ... 447
20.5.9 flexspi_ahb_error_code_t ... 447
20.5.10 flexspi_port_t ... 447
20.5.11 flexspi_arb_command_source_t ... 447
20.5.12 flexspi_command_type_t ... 448

20.6 Function Documentation ... 448
20.6.1 FLEXSPI_GetInstance ... 448
20.6.2 FLEXSPI_CheckAndClearError ... 448
20.6.3 FLEXSPI_Init ... 448
20.6.4 FLEXSPI_GetDefaultConfig ... 448
20.6.5 FLEXSPI_Deinit ... 449
20.6.6 FLEXSPI_UpdateDllValue ... 449
20.6.7 FLEXSPI_SetFlashConfig ... 449
20.6.8 FLEXSPI_SoftwareReset ... 450

20.6.9 FLEXSPI_Enable ... 451
20.6.10 FLEXSPI_EnableInterrupts ... 451
20.6.11 FLEXSPI_DisableInterrupts ... 451
20.6.12 FLEXSPI_EnableTxDMA ... 451
20.6.13 FLEXSPI_EnableRxDMA ... 452
20.6.14 FLEXSPI_GetTxFifoAddress ... 452
20.6.15 FLEXSPI_GetRxFifoAddress ... 452
20.6.16 FLEXSPI_ResetFifos ... 453
20.6.17 FLEXSPI_GetFifoCounts ... 454
20.6.18 FLEXSPI_GetInterruptStatusFlags ... 454
20.6.19 FLEXSPI_ClearInterruptStatusFlags ... 454
20.6.20 FLEXSPI_GetDataLearningPhase ... 455
20.6.21 FLEXSPI_GetArbitratorCommandSource ... 456
20.6.22 FLEXSPI_GetIPCommandErrorCode ... 456
20.6.23 FLEXSPI_GetAHBCommandErrorCode ... 456
20.6.24 FLEXSPI_GetBusIdleStatus ... 457
20.6.25 FLEXSPI_UpdateRxSampleClock ... 457
20.6.26 FLEXSPI_EnableIPParallelMode ... 457
20.6.27 FLEXSPI_EnableAHBParallelMode ... 458
20.6.28 FLEXSPI_UpdateLUT ... 459
20.6.29 FLEXSPI_WriteData ... 459
20.6.30 FLEXSPI_ReadData ... 459
20.6.31 FLEXSPI_WriteBlocking ... 460
20.6.32 FLEXSPI_ReadBlocking ... 460
20.6.33 FLEXSPI_TransferBlocking ... 461
20.6.34 FLEXSPI_TransferCreateHandle ... 461
20.6.35 FLEXSPI_TransferNonBlocking ... 462
20.6.36 FLEXSPI_TransferGetCount ... 462
20.6.37 FLEXSPI_TransferAbort ... 463
20.6.38 FLEXSPI_TransferHandleIRQ ... 463

20.7 FLEXSPI DMA Driver ... 464
20.7.1 Overview ... 464
20.7.2 Data Structure Documentation ... 465
20.7.3 Macro Definition Documentation ... 465
20.7.4 Enumeration Type Documentation ... 466
20.7.5 Function Documentation ... 466

Chapter 21 FMEAS: Frequency Measure Driver

21.1 Overview ... 469

21.2 Frequency Measure Driver operation ... 469

21.3 Typical use case ... 469

21.4 Macro Definition Documentation **470**
21.4.1 FSL_FMEAS_DRIVER_VERSION 470

21.5 Function Documentation **470**
21.5.1 FMEAS_StartMeasure 470
21.5.2 FMEAS_IsMeasureComplete 470
21.5.3 FMEAS_GetFrequency 470

Chapter 22 Hashcrypt: The Cryptographic Accelerator

22.1 Overview **471**

22.2 Hashcrypt Driver Initialization and deinitialization **471**

22.3 Comments about API usage in RTOS **471**

22.4 Comments about API usage in interrupt handler **471**

22.5 Hashcrypt Driver Examples **471**
22.5.1 Simple examples 471

22.6 Hashcrypt AES **473**
22.6.1 Overview 473
22.6.2 Data Structure Documentation 474
22.6.3 Enumeration Type Documentation 474
22.6.4 Function Documentation 475

22.7 Hashcrypt HASH **481**
22.7.1 Overview 481
22.7.2 Data Structure Documentation 481
22.7.3 Macro Definition Documentation 482
22.7.4 Typedef Documentation 482
22.7.5 Function Documentation 482

22.8 Hashcrypt Background HASH **485**
22.8.1 Overview 485
22.8.2 Function Documentation 485

Chapter 23 I3C: I3C Driver

23.1 Overview **487**

23.2 Data Structure Documentation **488**
23.2.1 struct i3c_device_info_t 488

23.3 Macro Definition Documentation **489**

23.3.1 I3C_RETRY_TIMES ... 489

23.4 Enumeration Type Documentation ... 489
23.4.1 anonymous enum ... 489
23.4.2 i3c_hdr_mode_t ... 489

23.5 I3C Master Driver ... 490
23.5.1 Overview ... 490
23.5.2 Data Structure Documentation ... 496
23.5.3 Typedef Documentation ... 499
23.5.4 Enumeration Type Documentation ... 500
23.5.5 Function Documentation ... 503

23.6 I3C Slave Driver ... 529
23.6.1 Overview ... 529
23.6.2 Data Structure Documentation ... 532
23.6.3 Typedef Documentation ... 535
23.6.4 Enumeration Type Documentation ... 535
23.6.5 Function Documentation ... 538

23.7 I3C Master DMA Driver ... 552

23.8 I3C FreeRTOS Driver ... 553

Chapter 24 IAP: In Application Programming Driver

24.1 Overview ... 554

24.2 Macro Definition Documentation ... 554
24.2.1 FSL_IAP_DRIVER_VERSION ... 554

24.3 IAP Boot Driver ... 555
24.3.1 Overview ... 555
24.3.2 Data Structure Documentation ... 555
24.3.3 Function Documentation ... 555

24.4 IAP FlexSPI Driver ... 557
24.4.1 Overview ... 557
24.4.2 Data Structure Documentation ... 560
24.4.3 Macro Definition Documentation ... 564
24.4.4 Enumeration Type Documentation ... 564
24.4.5 Function Documentation ... 566

24.5 IAP OTP Driver ... 571
24.5.1 Overview ... 571
24.5.2 Enumeration Type Documentation ... 571

24.5.3 Function Documentation ... 572

Chapter 25 INPUTMUX: Input Multiplexing Driver

25.1 Overview ... 575

25.2 Input Multiplexing Driver operation ... 575

25.3 Typical use case ... 575

25.4 Enumeration Type Documentation ... 576
25.4.1 inputmux_connection_t ... 576
25.4.2 inputmux_signal_t ... 577

25.5 Function Documentation ... 577
25.5.1 INPUTMUX_Init ... 577
25.5.2 INPUTMUX_AttachSignal ... 577
25.5.3 INPUTMUX_EnableSignal ... 578
25.5.4 INPUTMUX_Deinit ... 578

Chapter 26 LCDIF: LCD interface

26.1 Overview ... 579

26.2 Data Structure Documentation ... 582
26.2.1 struct lcdif_dpi_config_t ... 582
26.2.2 struct lcdif_fb_config_t ... 583
26.2.3 struct lcdif_cursor_config_t ... 583
26.2.4 struct lcdif_dither_config_t ... 584

26.3 Macro Definition Documentation ... 585
26.3.1 LCDIF_MAKE_CURSOR_COLOR ... 585
26.3.2 LCDIF_MAKE_GAMMA_VALUE ... 585
26.3.3 LCDIF_FB_ALIGN ... 585
26.3.4 LCDIF_GAMMA_INDEX_MAX ... 585

26.4 Enumeration Type Documentation ... 585
26.4.1 _lcdif_polarity_flags ... 585
26.4.2 lcdif_output_format_t ... 585
26.4.3 lcdif_fb_format_t ... 586
26.4.4 _lcdif_interrupt ... 586
26.4.5 lcdif_cursor_format_t ... 586

26.5 Function Documentation ... 586
26.5.1 LCDIF_Init ... 586
26.5.2 LCDIF_Deinit ... 587

26.5.3 LCDIF_DpiModeGetDefaultConfig ... 587
26.5.4 LCDIF_DpiModeSetConfig ... 587
26.5.5 LCDIF_SetFrameBufferConfig ... 588
26.5.6 LCDIF_FrameBufferGetDefaultConfig ... 588
26.5.7 LCDIF_SetFrameBufferAddr ... 588
26.5.8 LCDIF_SetFrameBufferStride ... 589
26.5.9 LCDIF_SetDitherConfig ... 589
26.5.10 LCDIF_SetGammaData ... 589
26.5.11 LCDIF_EnableInterrupts ... 590
26.5.12 LCDIF_DisableInterrupts ... 590
26.5.13 LCDIF_GetAndClearInterruptPendingFlags ... 590
26.5.14 LCDIF_CursorGetDefaultConfig ... 591
26.5.15 LCDIF_SetCursorConfig ... 591
26.5.16 LCDIF_SetCursorHotspotPosition ... 591
26.5.17 LCDIF_SetCursorBufferAddress ... 591
26.5.18 LCDIF_SetCursorColor ... 592

Chapter 27 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

27.1 Overview ... 593

27.2 Typical use case ... 593
27.2.1 Polling Configuration ... 593
27.2.2 Interrupt Configuration ... 593

27.3 Data Structure Documentation ... 596
27.3.1 struct lpadc_config_t ... 596
27.3.2 struct lpadc_conv_command_config_t ... 598
27.3.3 struct lpadc_conv_trigger_config_t ... 599
27.3.4 struct lpadc_conv_result_t ... 600

27.4 Macro Definition Documentation ... 600
27.4.1 FSL_LPADC_DRIVER_VERSION ... 600
27.4.2 LPADC_GET_ACTIVE_COMMAND_STATUS ... 600
27.4.3 LPADC_GET_ACTIVE_TRIGGER_STATUE ... 600

27.5 Enumeration Type Documentation ... 601
27.5.1 _lpadc_status_flags ... 601
27.5.2 _lpadc_interrupt_enable ... 601
27.5.3 lpadc_sample_scale_mode_t ... 601
27.5.4 lpadc_sample_channel_mode_t ... 602
27.5.5 lpadc_hardware_average_mode_t ... 602
27.5.6 lpadc_sample_time_mode_t ... 602
27.5.7 lpadc_hardware_compare_mode_t ... 603
27.5.8 lpadc_reference_voltage_source_t ... 603

27.5.9 lpadc_power_level_mode_t ... 603
27.5.10 lpadc_trigger_priority_policy_t ... 604

27.6 Function Documentation ... 604
27.6.1 LPADC_Init ... 604
27.6.2 LPADC_GetDefaultConfig ... 604
27.6.3 LPADC_Deinit ... 605
27.6.4 LPADC_Enable ... 605
27.6.5 LPADC_DoResetFIFO0 ... 605
27.6.6 LPADC_DoResetFIFO1 ... 605
27.6.7 LPADC_DoResetConfig ... 606
27.6.8 LPADC_GetStatusFlags ... 607
27.6.9 LPADC_ClearStatusFlags ... 607
27.6.10 LPADC_EnableInterrupts ... 607
27.6.11 LPADC_DisableInterrupts ... 607
27.6.12 LPADC_EnableFIFO0WatermarkDMA ... 608
27.6.13 LPADC_EnableFIFO1WatermarkDMA ... 608
27.6.14 LPADC_GetConvResultCount ... 608
27.6.15 LPADC_GetConvResult ... 609
27.6.16 LPADC_SetConvTriggerConfig ... 609
27.6.17 LPADC_GetDefaultConvTriggerConfig ... 609
27.6.18 LPADC_DoSoftwareTrigger ... 609
27.6.19 LPADC_SetConvCommandConfig ... 610
27.6.20 LPADC_GetDefaultConvCommandConfig ... 610

Chapter 28 CRC: Cyclic Redundancy Check Driver

28.1 Overview ... 611

28.2 CRC Driver Initialization and Configuration ... 611

28.3 CRC Write Data ... 611

28.4 CRC Get Checksum ... 611

28.5 Comments about API usage in RTOS ... 612

28.6 Data Structure Documentation ... 613
28.6.1 struct crc_config_t ... 613

28.7 Macro Definition Documentation ... 614
28.7.1 FSL_CRC_DRIVER_VERSION ... 614
28.7.2 CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT ... 614

28.8 Enumeration Type Documentation ... 614
28.8.1 crc_polynomial_t ... 614

28.9 Function Documentation ... **614**
28.9.1 CRC_Init ... 615
28.9.2 CRC_Deinit ... 616
28.9.3 CRC_Reset ... 616
28.9.4 CRC_WriteSeed ... 616
28.9.5 CRC_GetDefaultConfig ... 616
28.9.6 CRC_GetConfig ... 617
28.9.7 CRC_WriteData ... 617
28.9.8 CRC_Get32bitResult ... 617
28.9.9 CRC_Get16bitResult ... 618

Chapter 29 DMA: Direct Memory Access Controller Driver

29.1 Overview ... **620**

29.2 Typical use case ... **620**
29.2.1 DMA Operation ... 620

29.3 Data Structure Documentation ... **625**
29.3.1 struct dma_descriptor_t ... 625
29.3.2 struct dma_xfercfg_t ... 625
29.3.3 struct dma_channel_trigger_t ... 626
29.3.4 struct dma_channel_config_t ... 626
29.3.5 struct dma_transfer_config_t ... 627
29.3.6 struct dma_handle_t ... 627

29.4 Macro Definition Documentation ... **627**
29.4.1 FSL_DMA_DRIVER_VERSION ... 627
29.4.2 FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE ... 628
29.4.3 DMA_ALLOCATE_HEAD_DESCRIPTORS ... 628
29.4.4 DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE ... 628
29.4.5 DMA_ALLOCATE_LINK_DESCRIPTORS ... 628
29.4.6 DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE ... 628
29.4.7 DMA_DESCRIPTOR_END_ADDRESS ... 629
29.4.8 DMA_CHANNEL_XFER ... 629

29.5 Typedef Documentation ... **630**
29.5.1 dma_callback ... 630

29.6 Enumeration Type Documentation ... **630**
29.6.1 anonymous enum ... 630
29.6.2 anonymous enum ... 630
29.6.3 anonymous enum ... 630
29.6.4 dma_priority_t ... 630
29.6.5 dma_irq_t ... 631

29.6.6 dma_trigger_type_t ... 631
29.6.7 anonymous enum ... 631
29.6.8 dma_trigger_burst_t ... 632
29.6.9 dma_burst_wrap_t ... 632
29.6.10 dma_transfer_type_t ... 632

29.7 Function Documentation ... 632
29.7.1 DMA_Init ... 632
29.7.2 DMA_Deinit ... 633
29.7.3 DMA_InstallDescriptorMemory ... 633
29.7.4 DMA_ChannelIsActive ... 633
29.7.5 DMA_ChannelIsBusy ... 633
29.7.6 DMA_EnableChannelInterrupts ... 634
29.7.7 DMA_DisableChannelInterrupts ... 634
29.7.8 DMA_EnableChannel ... 634
29.7.9 DMA_DisableChannel ... 634
29.7.10 DMA_EnableChannelPeriphRq ... 635
29.7.11 DMA_DisableChannelPeriphRq ... 635
29.7.12 DMA_ConfigureChannelTrigger ... 635
29.7.13 DMA_SetChannelConfig ... 636
29.7.14 DMA_GetRemainingBytes ... 637
29.7.15 DMA_SetChannelPriority ... 637
29.7.16 DMA_GetChannelPriority ... 637
29.7.17 DMA_SetChannelConfigValid ... 638
29.7.18 DMA_DoChannelSoftwareTrigger ... 638
29.7.19 DMA_LoadChannelTransferConfig ... 638
29.7.20 DMA_CreateDescriptor ... 638
29.7.21 DMA_SetupDescriptor ... 639
29.7.22 DMA_SetupChannelDescriptor ... 639
29.7.23 DMA_LoadChannelDescriptor ... 640
29.7.24 DMA_AbortTransfer ... 640
29.7.25 DMA_CreateHandle ... 640
29.7.26 DMA_SetCallback ... 641
29.7.27 DMA_PrepareTransfer ... 642
29.7.28 DMA_PrepareChannelTransfer ... 642
29.7.29 DMA_SubmitTransfer ... 643
29.7.30 DMA_SubmitChannelTransferParameter ... 643
29.7.31 DMA_SubmitChannelDescriptor ... 644
29.7.32 DMA_SubmitChannelTransfer ... 645
29.7.33 DMA_StartTransfer ... 646
29.7.34 DMA_IRQHandle ... 646

Chapter 30 GPIO: General Purpose I/O

30.1 Overview ... 648

Section No. Title Page No.

30.2 Function groups ... **648**
30.2.1 Initialization and deinitialization ... 648
30.2.2 Pin manipulation ... 648
30.2.3 Port manipulation ... 648
30.2.4 Port masking ... 648

30.3 Typical use case ... **648**

30.4 Data Structure Documentation ... **650**
30.4.1 struct gpio_pin_config_t ... 650
30.4.2 struct gpio_interrupt_config_t ... 650

30.5 Macro Definition Documentation ... **650**
30.5.1 FSL_GPIO_DRIVER_VERSION ... 650

30.6 Enumeration Type Documentation ... **650**
30.6.1 gpio_pin_direction_t ... 650
30.6.2 gpio_pin_enable_mode_t ... 650
30.6.3 gpio_pin_enable_polarity_t ... 651
30.6.4 gpio_interrupt_index_t ... 651

30.7 Function Documentation ... **651**
30.7.1 GPIO_PortInit ... 651
30.7.2 GPIO_PinInit ... 651
30.7.3 GPIO_PinWrite ... 653
30.7.4 GPIO_PinRead ... 653
30.7.5 GPIO_PortSet ... 654
30.7.6 GPIO_PortClear ... 655
30.7.7 GPIO_PortToggle ... 655

Chapter 31 IOPCTL: Input/Output Pad Controller

31.1 Overview ... **656**

31.2 Data Structure Documentation ... **657**
31.2.1 struct iopctl_group_t ... 657

31.3 Macro Definition Documentation ... **657**
31.3.1 LPC_IOPCTL_DRIVER_VERSION ... 657
31.3.2 IOPCTL_FUNC0 ... 657

31.4 Function Documentation ... **658**
31.4.1 IOPCTL_PinMuxSet ... 658
31.4.2 IOPCTL_SetPinMuxing ... 658

Section No.	Title	Page No.

Chapter 32 RTC: Real Time Clock

32.1 Overview **659**

32.2 Function groups **659**
32.2.1 Initialization and deinitialization 659
32.2.2 Set & Get Datetime 659
32.2.3 Set & Get Alarm 659
32.2.4 Start & Stop timer 659
32.2.5 Status 660
32.2.6 Interrupt 660
32.2.7 High resolution timer 660

32.3 Typical use case **660**
32.3.1 RTC tick example 660

32.4 Data Structure Documentation **662**
32.4.1 struct rtc_datetime_t 662

32.5 Enumeration Type Documentation **663**
32.5.1 rtc_interrupt_enable_t 663
32.5.2 rtc_status_flags_t 663

32.6 Function Documentation **663**
32.6.1 RTC_Init 663
32.6.2 RTC_Deinit 663
32.6.3 RTC_SetDatetime 663
32.6.4 RTC_GetDatetime 664
32.6.5 RTC_SetAlarm 664
32.6.6 RTC_GetAlarm 664
32.6.7 RTC_EnableWakeupTimer 665
32.6.8 RTC_GetEnabledWakeupTimer 665
32.6.9 RTC_SetSecondsTimerMatch 665
32.6.10 RTC_GetSecondsTimerMatch 666
32.6.11 RTC_SetSecondsTimerCount 666
32.6.12 RTC_GetSecondsTimerCount 666
32.6.13 RTC_SetWakeupCount 666
32.6.14 RTC_GetWakeupCount 667
32.6.15 RTC_EnableWakeUpTimerInterruptFromDPD 667
32.6.16 RTC_EnableAlarmTimerInterruptFromDPD 667
32.6.17 RTC_EnableInterrupts 668
32.6.18 RTC_DisableInterrupts 668
32.6.19 RTC_GetEnabledInterrupts 668
32.6.20 RTC_GetStatusFlags 669
32.6.21 RTC_ClearStatusFlags 669
32.6.22 RTC_EnableTimer 669

32.6.23 RTC_StartTimer ... 670
32.6.24 RTC_StopTimer ... 670
32.6.25 RTC_Reset ... 670

Chapter 33 MIPI_DSI: MIPI DSI Host Controller

33.1 Overview ... 672

33.2 MIPI DSI Driver ... 673
33.2.1 Overview ... 673
33.2.2 Command mode data transfer ... 673
33.2.3 Data Structure Documentation ... 681
33.2.4 Typedef Documentation ... 685
33.2.5 Enumeration Type Documentation ... 686
33.2.6 Function Documentation ... 691

33.3 MIPI DSI SMARTDMA driver ... 701
33.3.1 Overview ... 701
33.3.2 Data Structure Documentation ... 702
33.3.3 Typedef Documentation ... 703
33.3.4 Enumeration Type Documentation ... 703
33.3.5 Function Documentation ... 703

Chapter 34 MRT: Multi-Rate Timer

34.1 Overview ... 706

34.2 Function groups ... 706
34.2.1 Initialization and deinitialization ... 706
34.2.2 Timer period Operations ... 706
34.2.3 Start and Stop timer operations ... 706
34.2.4 Get and release channel ... 707
34.2.5 Status ... 707
34.2.6 Interrupt ... 707

34.3 Typical use case ... 707
34.3.1 MRT tick example ... 707

34.4 Data Structure Documentation ... 709
34.4.1 struct mrt_config_t ... 709

34.5 Enumeration Type Documentation ... 709
34.5.1 mrt_chnl_t ... 709
34.5.2 mrt_timer_mode_t ... 709
34.5.3 mrt_interrupt_enable_t ... 710
34.5.4 mrt_status_flags_t ... 710

34.6 Function Documentation ... **710**
34.6.1 MRT_Init ... 710
34.6.2 MRT_Deinit ... 710
34.6.3 MRT_GetDefaultConfig ... 710
34.6.4 MRT_SetupChannelMode ... 711
34.6.5 MRT_EnableInterrupts ... 711
34.6.6 MRT_DisableInterrupts ... 711
34.6.7 MRT_GetEnabledInterrupts ... 711
34.6.8 MRT_GetStatusFlags ... 712
34.6.9 MRT_ClearStatusFlags ... 712
34.6.10 MRT_UpdateTimerPeriod ... 712
34.6.11 MRT_GetCurrentTimerCount ... 713
34.6.12 MRT_StartTimer ... 713
34.6.13 MRT_StopTimer ... 714
34.6.14 MRT_GetIdleChannel ... 714
34.6.15 MRT_ReleaseChannel ... 714

Chapter 35 MU: Messaging Unit

35.1 Overview ... **715**

35.2 Function description ... **715**
35.2.1 MU initialization ... 715
35.2.2 MU message ... 715
35.2.3 MU flags ... 716
35.2.4 Status and interrupt ... 716
35.2.5 MU misc functions ... 716

35.3 Macro Definition Documentation ... **719**
35.3.1 FSL_MU_DRIVER_VERSION ... 719

35.4 Enumeration Type Documentation ... **719**
35.4.1 _mu_status_flags ... 719
35.4.2 _mu_interrupt_enable ... 719
35.4.3 _mu_interrupt_trigger ... 720

35.5 Function Documentation ... **720**
35.5.1 MU_Init ... 720
35.5.2 MU_Deinit ... 720
35.5.3 MU_SendMsgNonBlocking ... 720
35.5.4 MU_SendMsg ... 721
35.5.5 MU_ReceiveMsgNonBlocking ... 721
35.5.6 MU_ReceiveMsg ... 722
35.5.7 MU_SetFlagsNonBlocking ... 723
35.5.8 MU_SetFlags ... 723

35.5.9 MU_GetFlags ... 724
35.5.10 MU_GetStatusFlags ... 724
35.5.11 MU_GetInterruptsPending ... 725
35.5.12 MU_ClearStatusFlags ... 726
35.5.13 MU_EnableInterrupts ... 727
35.5.14 MU_DisableInterrupts ... 727
35.5.15 MU_TriggerInterrupts ... 727
35.5.16 MU_ResetBothSides ... 728
35.5.17 MU_GetOtherCorePowerMode ... 728

Chapter 36 OSTIMER: OS Event Timer Driver

36.1 Overview ... 730

36.2 Function groups ... 730
36.2.1 Initialization and deinitialization ... 730
36.2.2 OSTIMER status ... 730
36.2.3 OSTIMER set match value ... 730
36.2.4 OSTIMER get timer count ... 730

36.3 Typical use case ... 731

36.4 Macro Definition Documentation ... 732
36.4.1 FSL_OSTIMER_DRIVER_VERSION ... 732

36.5 Typedef Documentation ... 732
36.5.1 ostimer_callback_t ... 732

36.6 Enumeration Type Documentation ... 732
36.6.1 _ostimer_flags ... 732

36.7 Function Documentation ... 732
36.7.1 OSTIMER_Init ... 732
36.7.2 OSTIMER_Deinit ... 732
36.7.3 OSTIMER_GrayToDecimal ... 732
36.7.4 OSTIMER_DecimalToGray ... 733
36.7.5 OSTIMER_GetStatusFlags ... 733
36.7.6 OSTIMER_ClearStatusFlags ... 733
36.7.7 OSTIMER_SetMatchRawValue ... 734
36.7.8 OSTIMER_SetMatchValue ... 734
36.7.9 OSTIMER_SetMatchRegister ... 735
36.7.10 OSTIMER_EnableMatchInterrupt ... 735
36.7.11 OSTIMER_DisableMatchInterrupt ... 735
36.7.12 OSTIMER_GetCurrentTimerRawValue ... 736
36.7.13 OSTIMER_GetCurrentTimerValue ... 736
36.7.14 OSTIMER_GetCaptureRawValue ... 736

36.7.15 OSTIMER_GetCaptureValue ... 737
36.7.16 OSTIMER_HandleIRQ ... 737

Chapter 37 OTFAD: On The Fly AES-128 Decryption Driver

37.1 Overview ... 738

37.2 Data Structure Documentation ... 739
37.2.1 struct otfad_encryption_config_t ... 739
37.2.2 struct otfad_config_t ... 740

37.3 Macro Definition Documentation ... 740
37.3.1 FSL_OTFAD_DRIVER_VERSION ... 740

37.4 Enumeration Type Documentation ... 740
37.4.1 anonymous enum ... 740
37.4.2 anonymous enum ... 740
37.4.3 anonymous enum ... 740

37.5 Function Documentation ... 741
37.5.1 OTFAD_GetDefaultConfig ... 741
37.5.2 OTFAD_Init ... 741
37.5.3 OTFAD_GetOperateMode ... 741
37.5.4 OTFAD_GetStatus ... 741
37.5.5 OTFAD_SetEncryptionConfig ... 741
37.5.6 OTFAD_GetEncryptionConfig ... 742
37.5.7 OTFAD_HitDetermination ... 742

Chapter 38 PINT: Pin Interrupt and Pattern Match Driver

38.1 Overview ... 743

38.2 Pin Interrupt and Pattern match Driver operation ... 743
38.2.1 Pin Interrupt use case ... 743
38.2.2 Pattern match use case ... 743

38.3 Typedef Documentation ... 746
38.3.1 pint_cb_t ... 746

38.4 Enumeration Type Documentation ... 746
38.4.1 pint_pin_enable_t ... 746
38.4.2 pint_pin_int_t ... 746
38.4.3 pint_pmatch_input_src_t ... 747
38.4.4 pint_pmatch_bslice_t ... 747
38.4.5 pint_pmatch_bslice_cfg_t ... 747

Section No. Title Page No.

38.5 Function Documentation **748**
38.5.1 PINT_Init 748
38.5.2 PINT_PinInterruptConfig 748
38.5.3 PINT_PinInterruptGetConfig 748
38.5.4 PINT_PinInterruptClrStatus 749
38.5.5 PINT_PinInterruptGetStatus 749
38.5.6 PINT_PinInterruptClrStatusAll 749
38.5.7 PINT_PinInterruptGetStatusAll 750
38.5.8 PINT_PinInterruptClrFallFlag 750
38.5.9 PINT_PinInterruptGetFallFlag 750
38.5.10 PINT_PinInterruptClrFallFlagAll 751
38.5.11 PINT_PinInterruptGetFallFlagAll 751
38.5.12 PINT_PinInterruptClrRiseFlag 752
38.5.13 PINT_PinInterruptGetRiseFlag 753
38.5.14 PINT_PinInterruptClrRiseFlagAll 753
38.5.15 PINT_PinInterruptGetRiseFlagAll 753
38.5.16 PINT_PatternMatchConfig 754
38.5.17 PINT_PatternMatchGetConfig 754
38.5.18 PINT_PatternMatchGetStatus 755
38.5.19 PINT_PatternMatchGetStatusAll 755
38.5.20 PINT_PatternMatchResetDetectLogic 755
38.5.21 PINT_PatternMatchEnable 756
38.5.22 PINT_PatternMatchDisable 756
38.5.23 PINT_PatternMatchEnableRXEV 756
38.5.24 PINT_PatternMatchDisableRXEV 757
38.5.25 PINT_EnableCallback 757
38.5.26 PINT_DisableCallback 757
38.5.27 PINT_Deinit 758
38.5.28 PINT_EnableCallbackByIndex 758
38.5.29 PINT_DisableCallbackByIndex 758

Chapter 39 POWERQUAD: PowerQuad hardware accelerator

39.1 Overview **760**

39.2 Function groups **761**
39.2.1 POWERQUAD functional Operation 761

39.3 Data Structure Documentation **767**
39.3.1 struct pq_prescale_t 767
39.3.2 struct pq_config_t 768
39.3.3 struct pq_biquad_param_t 769
39.3.4 struct pq_biquad_state_t 769
39.3.5 struct pq_biquad_cascade_df2_instance 770
39.3.6 union pq_float_t 770

39.4 Macro Definition Documentation ... **770**
39.4.1 FSL_POWERQUAD_DRIVER_VERSION ... 770
39.4.2 PQ_Initiate_Vector_Func ... 770
39.4.3 PQ_End_Vector_Func ... 771
39.4.4 PQ_StartVector ... 771
39.4.5 PQ_StartVectorFixed16 ... 771
39.4.6 PQ_StartVectorQ15 ... 773
39.4.7 PQ_EndVector ... 773
39.4.8 PQ_Vector8F32 ... 773
39.4.9 PQ_Vector8Fixed32 ... 774
39.4.10 PQ_Vector8Fixed16 ... 774
39.4.11 PQ_Vector8Q15 ... 774
39.4.12 PQ_DF2_Vector8_FP ... 775
39.4.13 PQ_DF2_Vector8_FX ... 775
39.4.14 PQ_Vector8BiquadDf2F32 ... 776
39.4.15 PQ_Vector8BiquadDf2Fixed32 ... 776
39.4.16 PQ_Vector8BiquadDf2Fixed16 ... 777
39.4.17 PQ_DF2_Cascade_Vector8_FP ... 777
39.4.18 PQ_DF2_Cascade_Vector8_FX ... 778
39.4.19 PQ_Vector8BiqaudDf2CascadeF32 ... 779
39.4.20 PQ_Vector8BiqaudDf2CascadeFixed32 ... 779
39.4.21 PQ_Vector8BiqaudDf2CascadeFixed16 ... 780
39.4.22 POWERQUAD_MAKE_MATRIX_LEN ... 781
39.4.23 PQ_Q31_2_FLOAT ... 781
39.4.24 PQ_Q15_2_FLOAT ... 781

39.5 Enumeration Type Documentation ... **781**
39.5.1 pq_computationengine_t ... 781
39.5.2 pq_format_t ... 781
39.5.3 pq_cordic_iter_t ... 781

39.6 Function Documentation ... **781**
39.6.1 PQ_GetDefaultConfig ... 781
39.6.2 PQ_SetConfig ... 782
39.6.3 PQ_SetCoprocessorScaler ... 782
39.6.4 PQ_Init ... 782
39.6.5 PQ_Deinit ... 783
39.6.6 PQ_SetFormat ... 783
39.6.7 PQ_WaitDone ... 783
39.6.8 PQ_LnF32 ... 783
39.6.9 PQ_InvF32 ... 783
39.6.10 PQ_SqrtF32 ... 784
39.6.11 PQ_InvSqrtF32 ... 784
39.6.12 PQ_EtoxF32 ... 784
39.6.13 PQ_EtonxF32 ... 784

39.6.14 PQ_SinF32 ... 785
39.6.15 PQ_CosF32 ... 785
39.6.16 PQ_BiquadF32 ... 785
39.6.17 PQ_DivF32 ... 785
39.6.18 PQ_Biquad1F32 ... 786
39.6.19 PQ_LnFixed ... 786
39.6.20 PQ_InvFixed ... 786
39.6.21 PQ_SqrtFixed ... 786
39.6.22 PQ_InvSqrtFixed ... 787
39.6.23 PQ_EtoxFixed ... 787
39.6.24 PQ_EtonxFixed ... 787
39.6.25 PQ_SinQ31 ... 787
39.6.26 PQ_SinQ15 ... 788
39.6.27 PQ_CosQ31 ... 788
39.6.28 PQ_CosQ15 ... 788
39.6.29 PQ_BiquadFixed ... 788
39.6.30 PQ_VectorLnF32 ... 789
39.6.31 PQ_VectorInvF32 ... 789
39.6.32 PQ_VectorSqrtF32 ... 789
39.6.33 PQ_VectorInvSqrtF32 ... 789
39.6.34 PQ_VectorEtoxF32 ... 790
39.6.35 PQ_VectorEtonxF32 ... 790
39.6.36 PQ_VectorSinF32 ... 790
39.6.37 PQ_VectorCosF32 ... 790
39.6.38 PQ_VectorLnFixed32 ... 791
39.6.39 PQ_VectorInvFixed32 ... 791
39.6.40 PQ_VectorSqrtFixed32 ... 791
39.6.41 PQ_VectorInvSqrtFixed32 ... 791
39.6.42 PQ_VectorEtoxFixed32 ... 792
39.6.43 PQ_VectorEtonxFixed32 ... 792
39.6.44 PQ_VectorSinQ15 ... 792
39.6.45 PQ_VectorCosQ15 ... 792
39.6.46 PQ_VectorSinQ31 ... 793
39.6.47 PQ_VectorCosQ31 ... 793
39.6.48 PQ_VectorLnFixed16 ... 793
39.6.49 PQ_VectorInvFixed16 ... 793
39.6.50 PQ_VectorSqrtFixed16 ... 794
39.6.51 PQ_VectorInvSqrtFixed16 ... 794
39.6.52 PQ_VectorEtoxFixed16 ... 794
39.6.53 PQ_VectorEtonxFixed16 ... 795
39.6.54 PQ_VectorBiqaudDf2F32 ... 795
39.6.55 PQ_VectorBiqaudDf2Fixed32 ... 795
39.6.56 PQ_VectorBiqaudDf2Fixed16 ... 795
39.6.57 PQ_VectorBiqaudCascadeDf2F32 ... 796
39.6.58 PQ_VectorBiqaudCascadeDf2Fixed32 ... 796

39.6.59 PQ_VectorBiqaudCascadeDf2Fixed16 ... 796
39.6.60 PQ_ArctanFixed ... 797
39.6.61 PQ_ArctanhFixed ... 797
39.6.62 PQ_Biquad1Fixed ... 798
39.6.63 PQ_TransformCFFT ... 798
39.6.64 PQ_TransformRFFT ... 798
39.6.65 PQ_TransformIFFT ... 799
39.6.66 PQ_TransformCDCT ... 799
39.6.67 PQ_TransformRDCT ... 800
39.6.68 PQ_TransformIDCT ... 800
39.6.69 PQ_BiquadBackUpInternalState ... 800
39.6.70 PQ_BiquadRestoreInternalState ... 800
39.6.71 PQ_BiquadCascadeDf2Init ... 801
39.6.72 PQ_BiquadCascadeDf2F32 ... 801
39.6.73 PQ_BiquadCascadeDf2Fixed32 ... 801
39.6.74 PQ_BiquadCascadeDf2Fixed16 ... 802
39.6.75 PQ_FIR ... 803
39.6.76 PQ_FIRIncrement ... 803
39.6.77 PQ_MatrixAddition ... 804
39.6.78 PQ_MatrixSubtraction ... 805
39.6.79 PQ_MatrixMultiplication ... 805
39.6.80 PQ_MatrixProduct ... 806
39.6.81 PQ_VectorDotProduct ... 807
39.6.82 PQ_MatrixInversion ... 807
39.6.83 PQ_MatrixTranspose ... 808
39.6.84 PQ_MatrixScale ... 809

Chapter 40 PUF: Physical Unclonable Function

40.1 Overview ... 810

40.2 PUF Driver Initialization and deinitialization ... 810

40.3 Comments about API usage in RTOS ... 810

40.4 Comments about API usage in interrupt handler ... 810

40.5 PUF Driver Examples ... 810
40.5.1 Simple examples ... 810

40.6 Macro Definition Documentation ... 811
40.6.1 FSL_PUF_DRIVER_VERSION ... 811
40.6.2 PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE ... 812

40.7 Enumeration Type Documentation ... 812
40.7.1 puf_key_slot_t ... 812

40.7.2 anonymous enum ... 812

Chapter 41 SCTimer: SCTimer/PWM (SCT)

41.1 Overview ... 813

41.2 Function groups ... 813
41.2.1 Initialization and deinitialization ... 813
41.2.2 PWM Operations ... 813
41.2.3 Status ... 813
41.2.4 Interrupt ... 813

41.3 SCTimer State machine and operations ... 814
41.3.1 SCTimer event operations ... 814
41.3.2 SCTimer state operations ... 814
41.3.3 SCTimer action operations ... 814

41.4 16-bit counter mode ... 814

41.5 Typical use case ... 815
41.5.1 PWM output ... 815

41.6 Data Structure Documentation ... 820
41.6.1 struct sctimer_pwm_signal_param_t ... 820
41.6.2 struct sctimer_config_t ... 820

41.7 Typedef Documentation ... 821
41.7.1 sctimer_event_callback_t ... 821

41.8 Enumeration Type Documentation ... 821
41.8.1 sctimer_pwm_mode_t ... 821
41.8.2 sctimer_counter_t ... 822
41.8.3 sctimer_input_t ... 822
41.8.4 sctimer_out_t ... 822
41.8.5 sctimer_pwm_level_select_t ... 822
41.8.6 sctimer_clock_mode_t ... 823
41.8.7 sctimer_clock_select_t ... 823
41.8.8 sctimer_conflict_resolution_t ... 823
41.8.9 sctimer_event_active_direction_t ... 824
41.8.10 sctimer_interrupt_enable_t ... 824
41.8.11 sctimer_status_flags_t ... 824

41.9 Function Documentation ... 825
41.9.1 SCTIMER_Init ... 825
41.9.2 SCTIMER_Deinit ... 825
41.9.3 SCTIMER_GetDefaultConfig ... 825

41.9.4 SCTIMER_SetupPwm ... 826
41.9.5 SCTIMER_UpdatePwmDutycycle ... 826
41.9.6 SCTIMER_EnableInterrupts ... 827
41.9.7 SCTIMER_DisableInterrupts ... 827
41.9.8 SCTIMER_GetEnabledInterrupts ... 827
41.9.9 SCTIMER_GetStatusFlags ... 828
41.9.10 SCTIMER_ClearStatusFlags ... 828
41.9.11 SCTIMER_StartTimer ... 828
41.9.12 SCTIMER_StopTimer ... 829
41.9.13 SCTIMER_CreateAndScheduleEvent ... 829
41.9.14 SCTIMER_ScheduleEvent ... 830
41.9.15 SCTIMER_IncreaseState ... 830
41.9.16 SCTIMER_GetCurrentState ... 830
41.9.17 SCTIMER_SetCounterState ... 831
41.9.18 SCTIMER_GetCounterState ... 831
41.9.19 SCTIMER_SetupCaptureAction ... 831
41.9.20 SCTIMER_SetCallback ... 832
41.9.21 SCTIMER_SetupStateLdMethodAction ... 832
41.9.22 SCTIMER_SetupNextStateActionwithLdMethod ... 833
41.9.23 SCTIMER_SetupNextStateAction ... 833
41.9.24 SCTIMER_SetupEventActiveDirection ... 834
41.9.25 SCTIMER_SetupOutputSetAction ... 834
41.9.26 SCTIMER_SetupOutputClearAction ... 834
41.9.27 SCTIMER_SetupOutputToggleAction ... 835
41.9.28 SCTIMER_SetupCounterLimitAction ... 836
41.9.29 SCTIMER_SetupCounterStopAction ... 836
41.9.30 SCTIMER_SetupCounterStartAction ... 836
41.9.31 SCTIMER_SetupCounterHaltAction ... 837
41.9.32 SCTIMER_SetupDmaTriggerAction ... 837
41.9.33 SCTIMER_SetCOUNTValue ... 837
41.9.34 SCTIMER_GetCOUNTValue ... 838
41.9.35 SCTIMER_SetEventInState ... 838
41.9.36 SCTIMER_ClearEventInState ... 838
41.9.37 SCTIMER_GetEventInState ... 839
41.9.38 SCTIMER_EventHandleIRQ ... 839

Chapter 42 SEMA42: Hardware Semaphores Driver

42.1 Overview ... 840

42.2 Typical use case ... 840

42.3 Macro Definition Documentation ... 841
42.3.1 SEMA42_GATE_NUM_RESET_ALL ... 841
42.3.2 SEMA42_GATEn ... 842

Section No. Title Page No.

42.4 Enumeration Type Documentation ... **842**
42.4.1 anonymous enum ... 842
42.4.2 sema42_gate_status_t ... 842

42.5 Function Documentation ... **842**
42.5.1 SEMA42_Init ... 842
42.5.2 SEMA42_Deinit ... 843
42.5.3 SEMA42_TryLock ... 843
42.5.4 SEMA42_Lock ... 843
42.5.5 SEMA42_Unlock ... 844
42.5.6 SEMA42_GetGateStatus ... 844
42.5.7 SEMA42_ResetGate ... 844
42.5.8 SEMA42_ResetAllGates ... 845

Chapter 43 SMARTDMA: SMART DMA Driver

43.1 Overview ... **846**

43.2 Data Structure Documentation ... **848**
43.2.1 struct smartdma_flexio_mculcd_param_t ... 848
43.2.2 struct smartdma_flexio_onelane_mculcd_param_t ... 848
43.2.3 struct smartdma_dsi_param_t ... 848
43.2.4 struct smartdma_rgb565_rgb888_param_t ... 848

43.3 Macro Definition Documentation ... **848**
43.3.1 SMARTDMA_DISPLAY_MEM_ADDR ... 848
43.3.2 s_smartdmaFlexioMcuLcdFirmware ... 848

43.4 Typedef Documentation ... **848**
43.4.1 smartdma_callback_t ... 848

43.5 Enumeration Type Documentation ... **849**
43.5.1 _smartdma_flexio_mculcd_api ... 849

43.6 Function Documentation ... **849**
43.6.1 SMARTDMA_Init ... 849
43.6.2 SMARTDMA_InitWithoutFirmware ... 850
43.6.3 SMARTDMA_InstallFirmware ... 850
43.6.4 SMARTDMA_InstallCallback ... 850
43.6.5 SMARTDMA_Boot ... 851

43.7 Variable Documentation ... **851**
43.7.1 s_smartdmaDisplayFirmware ... 851

Section No.	Title	Page No.

Chapter 44 TRNG: True Random Number Generator

44.1 Overview **852**

44.2 TRNG Initialization **852**

44.3 Get random data from TRNG **852**

44.4 Data Structure Documentation **853**
44.4.1 struct trng_statistical_check_limit_t 853
44.4.2 struct trng_config_t 854

44.5 Macro Definition Documentation **856**
44.5.1 FSL_TRNG_DRIVER_VERSION 856

44.6 Enumeration Type Documentation **856**
44.6.1 trng_sample_mode_t 857
44.6.2 trng_clock_mode_t 857
44.6.3 trng_ring_osc_div_t 857

44.7 Function Documentation **857**
44.7.1 TRNG_GetDefaultConfig 857
44.7.2 TRNG_Init 858
44.7.3 TRNG_Deinit 858
44.7.4 TRNG_GetRandomData 859

Chapter 45 USDHC: Ultra Secured Digital Host Controller Driver

45.1 Overview **860**

45.2 Typical use case **860**
45.2.1 USDHC Operation 860

45.3 Data Structure Documentation **871**
45.3.1 struct usdhc_adma2_descriptor_t 871
45.3.2 struct usdhc_capability_t 871
45.3.3 struct usdhc_boot_config_t 872
45.3.4 struct usdhc_config_t 873
45.3.5 struct usdhc_command_t 873
45.3.6 struct usdhc_adma_config_t 874
45.3.7 struct usdhc_scatter_gather_data_list_t 874
45.3.8 struct usdhc_scatter_gather_data_t 875
45.3.9 struct usdhc_scatter_gather_transfer_t 875
45.3.10 struct usdhc_data_t 876
45.3.11 struct usdhc_transfer_t 877
45.3.12 struct usdhc_transfer_callback_t 877

45.3.13 struct _usdhc_handle ... 877
45.3.14 struct usdhc_host_t ... 878

45.4 Macro Definition Documentation ... 879
45.4.1 FSL_USDHC_DRIVER_VERSION ... 879
45.4.2 USDHC_ADMA1_ADDRESS_ALIGN ... 879
45.4.3 USDHC_ADMA1_LENGTH_ALIGN ... 879
45.4.4 USDHC_ADMA2_ADDRESS_ALIGN ... 879
45.4.5 USDHC_ADMA2_LENGTH_ALIGN ... 879
45.4.6 USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT ... 879
45.4.7 USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK ... 879
45.4.8 USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT ... 879
45.4.9 USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK ... 879
45.4.10 USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY ... 880
45.4.11 USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT ... 880
45.4.12 USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK ... 880
45.4.13 USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY ... 880

45.5 Typedef Documentation ... 880
45.5.1 usdhc_adma1_descriptor_t ... 881
45.5.2 usdhc_transfer_function_t ... 881

45.6 Enumeration Type Documentation ... 881
45.6.1 anonymous enum ... 881
45.6.2 anonymous enum ... 881
45.6.3 anonymous enum ... 882
45.6.4 anonymous enum ... 882
45.6.5 anonymous enum ... 882
45.6.6 anonymous enum ... 883
45.6.7 anonymous enum ... 883
45.6.8 anonymous enum ... 884
45.6.9 anonymous enum ... 884
45.6.10 anonymous enum ... 885
45.6.11 anonymous enum ... 885
45.6.12 anonymous enum ... 885
45.6.13 usdhc_transfer_direction_t ... 886
45.6.14 usdhc_data_bus_width_t ... 886
45.6.15 usdhc_endian_mode_t ... 886
45.6.16 usdhc_dma_mode_t ... 886
45.6.17 anonymous enum ... 887
45.6.18 usdhc_boot_mode_t ... 887
45.6.19 usdhc_card_command_type_t ... 887
45.6.20 usdhc_card_response_type_t ... 887
45.6.21 anonymous enum ... 888
45.6.22 anonymous enum ... 888

45.6.23 anonymous enum ... 888
45.6.24 usdhc_burst_len_t ... 889
45.6.25 anonymous enum ... 889

45.7 Function Documentation ... 889
45.7.1 USDHC_Init ... 889
45.7.2 USDHC_Deinit ... 890
45.7.3 USDHC_Reset ... 890
45.7.4 USDHC_SetAdmaTableConfig ... 890
45.7.5 USDHC_SetInternalDmaConfig ... 891
45.7.6 USDHC_SetADMA2Descriptor ... 891
45.7.7 USDHC_SetADMA1Descriptor ... 892
45.7.8 USDHC_EnableInternalDMA ... 892
45.7.9 USDHC_EnableInterruptStatus ... 893
45.7.10 USDHC_DisableInterruptStatus ... 893
45.7.11 USDHC_EnableInterruptSignal ... 893
45.7.12 USDHC_DisableInterruptSignal ... 893
45.7.13 USDHC_GetEnabledInterruptStatusFlags ... 894
45.7.14 USDHC_GetInterruptStatusFlags ... 894
45.7.15 USDHC_ClearInterruptStatusFlags ... 894
45.7.16 USDHC_GetAutoCommand12ErrorStatusFlags ... 895
45.7.17 USDHC_GetAdmaErrorStatusFlags ... 896
45.7.18 USDHC_GetPresentStatusFlags ... 896
45.7.19 USDHC_GetCapability ... 896
45.7.20 USDHC_ForceClockOn ... 897
45.7.21 USDHC_SetSdClock ... 897
45.7.22 USDHC_SetCardActive ... 897
45.7.23 USDHC_AssertHardwareReset ... 898
45.7.24 USDHC_SetDataBusWidth ... 898
45.7.25 USDHC_WriteData ... 898
45.7.26 USDHC_ReadData ... 898
45.7.27 USDHC_SendCommand ... 899
45.7.28 USDHC_EnableWakeupEvent ... 899
45.7.29 USDHC_CardDetectByData3 ... 899
45.7.30 USDHC_DetectCardInsert ... 899
45.7.31 USDHC_EnableSdioControl ... 900
45.7.32 USDHC_SetContinueRequest ... 900
45.7.33 USDHC_RequestStopAtBlockGap ... 900
45.7.34 USDHC_SetMmcBootConfig ... 900
45.7.35 USDHC_EnableMmcBoot ... 901
45.7.36 USDHC_SetForceEvent ... 901
45.7.37 UDSHC_SelectVoltage ... 901
45.7.38 USDHC_RequestTuningForSDR50 ... 901
45.7.39 USDHC_RequestReTuning ... 902
45.7.40 USDHC_EnableAutoTuning ... 902

45.7.41 USDHC_SetRetuningTimer ... 902
45.7.42 USDHC_EnableAutoTuningForCmdAndData ... 902
45.7.43 USDHC_EnableManualTuning ... 903
45.7.44 USDHC_GetTuningDelayStatus ... 903
45.7.45 USDHC_SetTuningDelay ... 903
45.7.46 USDHC_AdjustDelayForManualTuning ... 904
45.7.47 USDHC_SetStandardTuningCounter ... 904
45.7.48 USDHC_EnableStandardTuning ... 904
45.7.49 USDHC_GetExecuteStdTuningStatus ... 905
45.7.50 USDHC_CheckStdTuningResult ... 905
45.7.51 USDHC_CheckTuningError ... 905
45.7.52 USDHC_EnableDDRMode ... 905
45.7.53 USDHC_EnableHS400Mode ... 906
45.7.54 USDHC_ResetStrobeDLL ... 906
45.7.55 USDHC_EnableStrobeDLL ... 906
45.7.56 USDHC_ConfigStrobeDLL ... 906
45.7.57 USDHC_SetStrobeDllOverride ... 907
45.7.58 USDHC_GetStrobeDLLStatus ... 907
45.7.59 USDHC_SetDataConfig ... 907
45.7.60 USDHC_TransferCreateHandle ... 908
45.7.61 USDHC_TransferNonBlocking ... 908
45.7.62 USDHC_TransferBlocking ... 909
45.7.63 USDHC_TransferHandleIRQ ... 910

Chapter 46 UTICK: MictoTick Timer Driver

46.1 Overview ... 911

46.2 Typical use case ... 911

46.3 Macro Definition Documentation ... 912
46.3.1 FSL_UTICK_DRIVER_VERSION ... 912

46.4 Typedef Documentation ... 912
46.4.1 utick_callback_t ... 912

46.5 Enumeration Type Documentation ... 912
46.5.1 utick_mode_t ... 912

46.6 Function Documentation ... 912
46.6.1 UTICK_Init ... 912
46.6.2 UTICK_Deinit ... 912
46.6.3 UTICK_GetStatusFlags ... 912
46.6.4 UTICK_ClearStatusFlags ... 913
46.6.5 UTICK_SetTick ... 913

46.6.6 UTICK_HandleIRQ ... 913

Chapter 47 WWDT: Windowed Watchdog Timer Driver

47.1 Overview ... 915

47.2 Function groups ... 915
47.2.1 Initialization and deinitialization ... 915
47.2.2 Status ... 915
47.2.3 Interrupt ... 915
47.2.4 Watch dog Refresh ... 915

47.3 Typical use case ... 915

47.4 Data Structure Documentation ... 916
47.4.1 struct wwdt_config_t ... 917

47.5 Macro Definition Documentation ... 917
47.5.1 FSL_WWDT_DRIVER_VERSION ... 917

47.6 Enumeration Type Documentation ... 917
47.6.1 _wwdt_status_flags_t ... 917

47.7 Function Documentation ... 917
47.7.1 WWDT_GetDefaultConfig ... 918
47.7.2 WWDT_Init ... 918
47.7.3 WWDT_Deinit ... 918
47.7.4 WWDT_Enable ... 919
47.7.5 WWDT_Disable ... 919
47.7.6 WWDT_GetStatusFlags ... 919
47.7.7 WWDT_ClearStatusFlags ... 920
47.7.8 WWDT_SetWarningValue ... 920
47.7.9 WWDT_SetTimeoutValue ... 920
47.7.10 WWDT_SetWindowValue ... 922
47.7.11 WWDT_Refresh ... 922

Chapter 48 Debug Console

48.1 Overview ... 923

48.2 Function groups ... 923
48.2.1 Initialization ... 923
48.2.2 Advanced Feature ... 924
48.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART ... 928

48.3 Typical use case ... 929

Section No.	Title	Page No.

48.4 Function groups **930**
48.4.1 Initialization 930
48.4.2 Advanced Feature 930
48.4.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART 935

48.5 Typical use case **936**

48.6 Macro Definition Documentation **938**
48.6.1 DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 938
48.6.2 DEBUGCONSOLE_REDIRECT_TO_SDK 938
48.6.3 DEBUGCONSOLE_DISABLE 938
48.6.4 SDK_DEBUGCONSOLE 938
48.6.5 PRINTF 938

48.7 Function Documentation **938**
48.7.1 DbgConsole_Init 938
48.7.2 DbgConsole_Deinit 939
48.7.3 DbgConsole_EnterLowpower 939
48.7.4 DbgConsole_ExitLowpower 940
48.7.5 DbgConsole_Printf 940
48.7.6 DbgConsole_Vprintf 940
48.7.7 DbgConsole_Putchar 940
48.7.8 DbgConsole_Scanf 941
48.7.9 DbgConsole_Getchar 941
48.7.10 DbgConsole_BlockingPrintf 942
48.7.11 DbgConsole_BlockingVprintf 942
48.7.12 DbgConsole_Flush 942
48.7.13 StrFormatPrintf 943
48.7.14 StrFormatScanf 943

48.8 Semihosting **944**
48.8.1 Guide Semihosting for IAR 944
48.8.2 Guide Semihosting for Keil µVision 944
48.8.3 Guide Semihosting for MCUXpresso IDE 945
48.8.4 Guide Semihosting for ARMGCC 945
48.8.5 Guide Semihosting for IAR 947
48.8.6 Guide Semihosting for Keil µVision 948
48.8.7 Guide Semihosting for MCUXpresso IDE 948
48.8.8 Guide Semihosting for ARMGCC 948

48.9 SWO **951**
48.9.1 Guide SWO for SDK 951
48.9.2 Guide SWO for Keil µVision 952
48.9.3 Guide SWO for MCUXpresso IDE 953
48.9.4 Guide SWO for ARMGCC 953

Section No.	Title	Page No.

Chapter 49 Shell

49.1 Overview **954**

49.2 Function groups **954**
49.2.1 Initialization 954
49.2.2 Advanced Feature 954
49.2.3 Shell Operation 954

49.3 Data Structure Documentation **956**
49.3.1 struct shell_command_t 956

49.4 Macro Definition Documentation **957**
49.4.1 SHELL_NON_BLOCKING_MODE 957
49.4.2 SHELL_AUTO_COMPLETE 957
49.4.3 SHELL_BUFFER_SIZE 957
49.4.4 SHELL_MAX_ARGS 957
49.4.5 SHELL_HISTORY_COUNT 957
49.4.6 SHELL_HANDLE_SIZE 957
49.4.7 SHELL_USE_COMMON_TASK 957
49.4.8 SHELL_TASK_PRIORITY 957
49.4.9 SHELL_TASK_STACK_SIZE 957
49.4.10 SHELL_HANDLE_DEFINE 958
49.4.11 SHELL_COMMAND_DEFINE 958
49.4.12 SHELL_COMMAND 959

49.5 Typedef Documentation **959**
49.5.1 cmd_function_t 959

49.6 Enumeration Type Documentation **959**
49.6.1 shell_status_t 959

49.7 Function Documentation **959**
49.7.1 SHELL_Init 959
49.7.2 SHELL_RegisterCommand 960
49.7.3 SHELL_UnregisterCommand 961
49.7.4 SHELL_Write 961
49.7.5 SHELL_Printf 961
49.7.6 SHELL_WriteSynchronization 962
49.7.7 SHELL_PrintfSynchronization 962
49.7.8 SHELL_ChangePrompt 963
49.7.9 SHELL_PrintPrompt 963
49.7.10 SHELL_Task 963
49.7.11 SHELL_checkRunningInIsr 964

Section No.	Title	Page No.

Chapter 50 Cards: Secure Digital Card/Embedded MultiMedia Card/SDIO Card

50.1 Overview **965**

50.2 SDIO Card Driver **966**
50.2.1 Overview 966
50.2.2 SDIO CARD Operation 966
50.2.3 Data Structure Documentation 968
50.2.4 Macro Definition Documentation 970
50.2.5 Enumeration Type Documentation 970
50.2.6 Function Documentation 970

50.3 SD Card Driver **987**
50.3.1 Overview 987
50.3.2 SD CARD Operation 987
50.3.3 Data Structure Documentation 990
50.3.4 Macro Definition Documentation 991
50.3.5 Enumeration Type Documentation 991
50.3.6 Function Documentation 991

50.4 MMC Card Driver **1001**
50.4.1 Overview 1001
50.4.2 MMC CARD Operation 1001
50.4.3 Data Structure Documentation 1003
50.4.4 Macro Definition Documentation 1005
50.4.5 Enumeration Type Documentation 1005
50.4.6 Function Documentation 1005

50.5 SDMMC HOST Driver **1019**
50.5.1 Overview 1019

50.6 SDMMC OSA **1020**
50.6.1 Overview 1020
50.6.2 Data Structure Documentation 1021
50.6.3 Function Documentation 1021
50.6.4 USDHC HOST adapter Driver 1025

50.7 SDMMC Common **1038**
50.7.1 Overview 1038
50.7.2 Data Structure Documentation 1057
50.7.3 Macro Definition Documentation 1070
50.7.4 Enumeration Type Documentation 1070
50.7.5 Function Documentation 1087

Section No.	Title	Page No.

Chapter 51 CODEC Driver

51.1 Overview **1092**

51.2 CODEC Common Driver **1093**
51.2.1 Overview 1093
51.2.2 Data Structure Documentation 1098
51.2.3 Macro Definition Documentation 1099
51.2.4 Enumeration Type Documentation 1099
51.2.5 Function Documentation 1104

51.3 CODEC I2C Driver **1108**
51.3.1 Overview 1108
51.3.2 Data Structure Documentation 1109
51.3.3 Enumeration Type Documentation 1109
51.3.4 Function Documentation 1109

51.4 CS42888 Driver **1112**
51.4.1 Overview 1112
51.4.2 Data Structure Documentation 1114
51.4.3 Macro Definition Documentation 1115
51.4.4 Enumeration Type Documentation 1115
51.4.5 Function Documentation 1116
51.4.6 CS42888 Adapter 1122

51.5 TFA9896 Driver **1130**
51.5.1 Overview 1130
51.5.2 Data Structure Documentation 1133
51.5.3 Enumeration Type Documentation 1134
51.5.4 Function Documentation 1135
51.5.5 TFA9896 Adapter 1146

51.6 WM8904 Driver **1154**
51.6.1 Overview 1154
51.6.2 Data Structure Documentation 1158
51.6.3 Macro Definition Documentation 1159
51.6.4 Enumeration Type Documentation 1159
51.6.5 Function Documentation 1162
51.6.6 WM8904 Adapter 1171

Chapter 52 Serial Manager

52.1 Overview **1179**

52.2 Data Structure Documentation **1182**
52.2.1 struct serial_manager_config_t 1182

52.2.2 struct serial_manager_callback_message_t ... 1182

52.3 Macro Definition Documentation ... 1182
52.3.1 SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE ... 1183
52.3.2 SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE ... 1183
52.3.3 SERIAL_MANAGER_USE_COMMON_TASK ... 1183
52.3.4 SERIAL_MANAGER_HANDLE_SIZE ... 1183
52.3.5 SERIAL_MANAGER_HANDLE_DEFINE ... 1183
52.3.6 SERIAL_MANAGER_WRITE_HANDLE_DEFINE ... 1183
52.3.7 SERIAL_MANAGER_READ_HANDLE_DEFINE ... 1184
52.3.8 SERIAL_MANAGER_TASK_PRIORITY ... 1184
52.3.9 SERIAL_MANAGER_TASK_STACK_SIZE ... 1184

52.4 Enumeration Type Documentation ... 1184
52.4.1 serial_port_type_t ... 1184
52.4.2 serial_manager_type_t ... 1185
52.4.3 serial_manager_status_t ... 1185

52.5 Function Documentation ... 1185
52.5.1 SerialManager_Init ... 1185
52.5.2 SerialManager_Deinit ... 1187
52.5.3 SerialManager_OpenWriteHandle ... 1187
52.5.4 SerialManager_CloseWriteHandle ... 1188
52.5.5 SerialManager_OpenReadHandle ... 1189
52.5.6 SerialManager_CloseReadHandle ... 1190
52.5.7 SerialManager_WriteBlocking ... 1191
52.5.8 SerialManager_ReadBlocking ... 1192
52.5.9 SerialManager_EnterLowpower ... 1192
52.5.10 SerialManager_ExitLowpower ... 1193
52.5.11 SerialManager_needPollingIsr ... 1193

52.6 Serial Port Uart ... 1194
52.6.1 Overview ... 1194
52.6.2 Enumeration Type Documentation ... 1194

52.7 Serial Port USB ... 1195
52.7.1 Overview ... 1195
52.7.2 Data Structure Documentation ... 1195
52.7.3 Enumeration Type Documentation ... 1196
52.7.4 USB Device Configuration ... 1197

52.8 Serial Port SWO ... 1198
52.8.1 Overview ... 1198
52.8.2 Data Structure Documentation ... 1198
52.8.3 Enumeration Type Documentation ... 1198

Chapter 53 I2s_dma_driver

53.1 Overview **1199**

53.2 Data Structure Documentation **1200**
53.2.1 struct _i2s_dma_handle 1200

53.3 Macro Definition Documentation **1200**
53.3.1 FSL_I2S_DMA_DRIVER_VERSION 1200

53.4 Typedef Documentation **1200**
53.4.1 i2s_dma_transfer_callback_t 1200

53.5 Function Documentation **1201**
53.5.1 I2S_TxTransferCreateHandleDMA 1201
53.5.2 I2S_TxTransferSendDMA 1201
53.5.3 I2S_TransferAbortDMA 1202
53.5.4 I2S_RxTransferCreateHandleDMA 1202
53.5.5 I2S_RxTransferReceiveDMA 1202
53.5.6 I2S_DMACallback 1203
53.5.7 I2S_TransferInstallLoopDMADescriptorMemory 1203
53.5.8 I2S_TransferSendLoopDMA 1204
53.5.9 I2S_TransferReceiveLoopDMA 1204

Chapter 54 Hashcrypt_driver

54.1 Overview **1206**

54.2 Macro Definition Documentation **1206**
54.2.1 FSL_HASHCRYPT_DRIVER_VERSION 1206

54.3 Enumeration Type Documentation **1207**
54.3.1 hashcrypt_algo_t 1207

54.4 Function Documentation **1207**
54.4.1 HASHCRYPT_Init 1207
54.4.2 HASHCRYPT_Deinit 1208

Chapter 55 I3c_common_driver

55.1 Overview **1209**

55.2 Data Structure Documentation **1209**
55.2.1 struct i3c_config_t 1209

55.3 Function Documentation **1211**

55.3.1 I3C_GetInstance 1211
55.3.2 I3C_GetDefaultConfig 1211
55.3.3 I3C_Init 1212
55.3.4 CODEC Adapter 1213

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

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: 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.

© 2021 NXP B.V.

Chapter 3
Architectural Overview

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

Overview

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

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

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/DMA driver/eDMA driver to quickly enable the peripherals and perform transfers.

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

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

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

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

Interrupt handling for transactional APIs

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

```
        PUBWEAK SPI0_IRQHandler
        PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

The first level of the weak implementation are the functions defined in the vector table. In the devices/<DEVICE_NAME>/<TOOLCHAIN>/startup_<DEVICE_NAME>.s/.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0_DriverIRQHandler) jumps to itself (B). The MCUXpresso SDK drivers with transactional APIs provide the reimplementation of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0_DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

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

The limitation of the double weak mechanism is that it cannot be used for peripherals that share the same vector entry. For this use case, redefine the first layer weak function to enable the desired peripheral interrupt functionality. For example, if the MCU's UART0 and UART1 share the same vector entry, redefine the UART0_UART1_IRQHandler according to the use case requirements.

Feature Header Files

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

Application

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

Chapter 4
Clock Driver

4.1 Overview

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

The clock driver supports:

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

Files

- file fsl_clock.h

Data Structures

- struct clock_sys_pll_config_t
  PLL configuration for SYSPLL. More...
- struct clock_audio_pll_config_t
  PLL configuration for SYSPLL. More...
- struct clock_frg_clk_config_t
  PLL configuration for FRG. More...

Macros

- #define MIPI_DSI_HOST_CLOCKS
  Clock ip name array for MIPI DSI.
- #define LCDIF_CLOCKS
  Clock ip name array for LCDIF.
- #define SCT_CLOCKS
  Clock ip name array for SCT.
- #define USBD_CLOCKS
  Clock ip name array for USBD.
- #define FLEXSPI_CLOCKS
  Clock ip name array for FlexSPI.
- #define CACHE64_CLOCKS
  Clock ip name array for Cache64.
- #define TRNG_CLOCKS
  Clock ip name array for RNG.
- #define PUF_CLOCKS
  Clock ip name array for PUF.
- #define HASHCRYPT_CLOCKS
  Clock ip name array for HashCrypt.
- #define CASPER_CLOCKS

Clock ip name array for Casper.

- #define POWERQUAD_CLOCKS

  Clock ip name array for Powerquad.
- #define LPADC_CLOCKS

  Clock ip name array for ADC.
- #define CMP_CLOCKS

  Clock ip name array for ACMP.
- #define USDHC_CLOCKS

  Clock ip name array for uSDHC.
- #define WWDT_CLOCKS

  Clock ip name array for WWDT.
- #define UTICK_CLOCKS

  Clock ip name array for UTICK.
- #define FLEXIO_CLOCKS

  Clock ip name array for FlexIO.
- #define OSTIMER_CLOCKS

  Clock ip name array for OSTimer.
- #define FLEXCOMM_CLOCKS

  Clock ip name array for FLEXCOMM.
- #define USART_CLOCKS

  Clock ip name array for LPUART.
- #define I2C_CLOCKS

  Clock ip name array for I2C.
- #define SPI_CLOCKS

  Clock ip name array for SPI.
- #define I2S_CLOCKS

  Clock ip name array for FLEXI2S.
- #define DMIC_CLOCKS

  Clock ip name array for DMIC.
- #define SEMA42_CLOCKS

  Clock ip name array for SEMA.
- #define MU_CLOCKS

  Clock ip name array for MUA.
- #define DMA_CLOCKS

  Clock ip name array for DMA.
- #define CRC_CLOCKS

  Clock ip name array for CRC.
- #define GPIO_CLOCKS

  Clock ip name array for GPIO.
- #define PINT_CLOCKS

  Clock ip name array for PINT.
- #define I3C_CLOCKS

  Clock ip name array for I3C.
- #define MRT_CLOCKS

  Clock ip name array for MRT.
- #define RTC_CLOCKS

  Clock ip name array for RTC.
- #define CTIMER_CLOCKS

  Clock ip name array for CT32B.
- #define CLK_GATE_REG_OFFSET_SHIFT 8U

  Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

- #define SYSPLL0CLKSEL_OFFSET 0x200

  Clock Mux Switches The encoding is as follows each connection identified is 32bits wide starting from LSB upwards.

Enumerations

- enum clock_ip_name_t {
  kCLOCK_IpInvalid = 0U,
  kCLOCK_Dsp = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 1),
  kCLOCK_RomCtrlr = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 2),
  kCLOCK_AxiSwitch = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 3),
  kCLOCK_AxiCtrl = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 4),
  kCLOCK_PowerQuad = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 8),
  kCLOCK_Casper = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 9),
  kCLOCK_HashCrypt = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 10),
  kCLOCK_Puf = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 11),
  kCLOCK_Rng = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 12),
  kCLOCK_Flexspi0 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 16),
  kCLOCK_OtpCtrl = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 17),
  kCLOCK_Flexspi1 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 18),
  kCLOCK_UsbhsPhy = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 20),
  kCLOCK_UsbhsDevice = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 21),
  kCLOCK_UsbhsHost = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 22),
  kCLOCK_UsbhsSram = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 23),
  kCLOCK_Sct = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 24),
  kCLOCK_Gpu = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 26),
  kCLOCK_DisplayCtrl = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 27),
  kCLOCK_MipiDsiCtrl = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 28),
  kCLOCK_Smartdma = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL0, 30),
  kCLOCK_Sdio0 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL1, 2),
  kCLOCK_Sdio1 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL1, 3),
  kCLOCK_Acmp0 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL1, 15),
  kCLOCK_Adc0 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL1, 16),
  kCLOCK_ShsGpio0 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL1, 24),
  kCLOCK_Utick0 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL2, 0),
  kCLOCK_Wwdt0 = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL2, 1),
  kCLOCK_Pmc = CLK_GATE_DEFINE(CLK_CTL0_PSCCTL2, 29),
  kCLOCK_Flexcomm0 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 8),
  kCLOCK_Flexcomm1 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 9),
  kCLOCK_Flexcomm2 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 10),
  kCLOCK_Flexcomm3 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 11),
  kCLOCK_Flexcomm4 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 12),
  kCLOCK_Flexcomm5 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 13),
  kCLOCK_Flexcomm6 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 14),
  kCLOCK_Flexcomm7 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 15),
  kCLOCK_Flexcomm8 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 16),
  kCLOCK_Flexcomm9 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 17),
  kCLOCK_Flexcomm10 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 18),
  kCLOCK_Flexcomm11 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 19),
  kCLOCK_Flexcomm12 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 20),
  kCLOCK_Flexcomm13 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 21),
  kCLOCK_Flexcomm14 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 22),
  kCLOCK_Flexcomm15 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 23),
  kCLOCK_Flexcomm16 = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL0, 25),

kCLOCK_InputMux = CLK_GATE_DEFINE(CLK_CTL1_PSCCTL2, 31) }

Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.

- enum clock_name_t {
  kCLOCK_CoreSysClk,
  kCLOCK_BusClk,
  kCLOCK_MclkClk,
  kCLOCK_ClockOutClk,
  kCLOCK_AdcClk,
  kCLOCK_Flexspi0Clk,
  kCLOCK_Flexspi1Clk,
  kCLOCK_SctClk,
  kCLOCK_Wdt0Clk,
  kCLOCK_Wdt1Clk,
  kCLOCK_SystickClk,
  kCLOCK_Sdio0Clk,
  kCLOCK_Sdio1Clk,
  kCLOCK_I3cClk,
  kCLOCK_UsbClk,
  kCLOCK_DmicClk,
  kCLOCK_DspCpuClk,
  kCLOCK_AcmpClk,
  kCLOCK_Flexcomm0Clk,
  kCLOCK_Flexcomm1Clk,
  kCLOCK_Flexcomm2Clk,
  kCLOCK_Flexcomm3Clk,
  kCLOCK_Flexcomm4Clk,
  kCLOCK_Flexcomm5Clk,
  kCLOCK_Flexcomm6Clk,
  kCLOCK_Flexcomm7Clk,
  kCLOCK_Flexcomm8Clk,
  kCLOCK_Flexcomm9Clk,
  kCLOCK_Flexcomm10Clk,
  kCLOCK_Flexcomm11Clk,
  kCLOCK_Flexcomm12Clk,
  kCLOCK_Flexcomm13Clk,
  kCLOCK_Flexcomm14Clk,
  kCLOCK_Flexcomm15Clk,
  kCLOCK_Flexcomm16Clk,
  kCLOCK_FlexioClk,
  kCLOCK_GpuClk,
  kCLOCK_DcPixelClk,
  kCLOCK_MipiDphyClk,
  kCLOCK_MipiDphyEscRxClk,
  kCLOCK_MipiDphyEscTxClk }

  Clock name used to get clock frequency.

- enum clock_pfd_t {
  kCLOCK_Pfd0 = 0U,
  kCLOCK_Pfd1 = 1U,
  kCLOCK_Pfd2 = 2U,
  kCLOCK_Pfd3 = 3U }
  PLL PFD clock name.
- enum clock_attach_id_t {
  kFRO_DIV8_to_SYS_PLL = CLKCTL0_TUPLE_MUXA(SYSPLL0CLKSEL_OFFSET, 0),
  kOSC_CLK_to_SYS_PLL = CLKCTL0_TUPLE_MUXA(SYSPLL0CLKSEL_OFFSET, 1),
  kNONE_to_SYS_PLL = CLKCTL0_TUPLE_MUXA(SYSPLL0CLKSEL_OFFSET, 7),
  kFRO_DIV8_to_AUDIO_PLL = CLKCTL1_TUPLE_MUXA(AUDIOPLL0CLKSEL_OFFSET, 0),
  kOSC_CLK_to_AUDIO_PLL = CLKCTL1_TUPLE_MUXA(AUDIOPLL0CLKSEL_OFFSET, 1),
  kNONE_to_AUDIO_PLL = CLKCTL1_TUPLE_MUXA(AUDIOPLL0CLKSEL_OFFSET, 7),
  kLPOSC_to_MAIN_CLK,
  kFRO_DIV2_to_MAIN_CLK,
  kFRO_DIV4_to_MAIN_CLK,
  kFRO_DIV8_to_MAIN_CLK,
  kFRO_DIV16_to_MAIN_CLK,
  kOSC_CLK_to_MAIN_CLK,
  kFRO_DIV1_to_MAIN_CLK,
  kMAIN_PLL_to_MAIN_CLK = CLKCTL0_TUPLE_MUXA(MAINCLKSELB_OFFSET, 1),
  kOSC32K_to_MAIN_CLK = CLKCTL0_TUPLE_MUXA(MAINCLKSELB_OFFSET, 2),
  kFRO_DIV1_to_DSP_MAIN_CLK,
  kOSC_CLK_to_DSP_MAIN_CLK,
  kLPOSC_to_DSP_MAIN_CLK,
  kMAIN_PLL_to_DSP_MAIN_CLK = CLKCTL1_TUPLE_MUXA(DSPCPUCLKSELB_OFFSET, 1),
  kDSP_PLL_to_DSP_MAIN_CLK = CLKCTL1_TUPLE_MUXA(DSPCPUCLKSELB_OFFSET, 2),
  kOSC32K_to_DSP_MAIN_CLK = CLKCTL1_TUPLE_MUXA(DSPCPUCLKSELB_OFFSET, 3),
  kLPOSC_to_UTICK_CLK = CLKCTL0_TUPLE_MUXA(UTICKFCLKSEL_OFFSET, 0),
  kNONE_to_UTICK_CLK = CLKCTL0_TUPLE_MUXA(UTICKFCLKSEL_OFFSET, 7),
  kLPOSC_to_WDT0_CLK = CLKCTL0_TUPLE_MUXA(WDT0FCLKSEL_OFFSET, 0),
  kNONE_to_WDT0_CLK = CLKCTL0_TUPLE_MUXA(WDT0FCLKSEL_OFFSET, 7),
  kLPOSC_to_WDT1_CLK = CLKCTL1_TUPLE_MUXA(WDT1FCLKSEL_OFFSET, 0),
  kNONE_to_WDT1_CLK = CLKCTL1_TUPLE_MUXA(WDT1FCLKSEL_OFFSET, 7),
  kOSC32K_to_32KHZWAKE_CLK = CLKCTL0_TUPLE_MUXA(A32KHZWAKECLKSEL_OFFSET, 0),
  kLPOSC_DIV32_to_32KHZWAKE_CLK = CLKCTL0_TUPLE_MUXA(A32KHZWAKECLKSEL_OFFSET, 1),
  kNONE_to_32KHZWAKE_CLK = CLKCTL0_TUPLE_MUXA(A32KHZWAKECLKSEL_OF-

FSET, 7),
kMAIN_CLK_DIV_to_SYSTICK_CLK = CLKCTL0_TUPLE_MUXA(SYSTICKFCLKSEL_OFFSET, 0),
kLPOSC_to_SYSTICK_CLK = CLKCTL0_TUPLE_MUXA(SYSTICKFCLKSEL_OFFSET, 1),
kOSC32K_to_SYSTICK_CLK = CLKCTL0_TUPLE_MUXA(SYSTICKFCLKSEL_OFFSET, 2),
kNONE_to_SYSTICK_CLK = CLKCTL0_TUPLE_MUXA(SYSTICKFCLKSEL_OFFSET, 7),
kMAIN_CLK_to_SDIO0_CLK = CLKCTL0_TUPLE_MUXA(SDIO0FCLKSEL_OFFSET, 0),
kMAIN_PLL_to_SDIO0_CLK = CLKCTL0_TUPLE_MUXA(SDIO0FCLKSEL_OFFSET, 1),
kAUX0_PLL_to_SDIO0_CLK = CLKCTL0_TUPLE_MUXA(SDIO0FCLKSEL_OFFSET, 2),
kFRO_DIV2_to_SDIO0_CLK = CLKCTL0_TUPLE_MUXA(SDIO0FCLKSEL_OFFSET, 3),
kAUX1_PLL_to_SDIO0_CLK = CLKCTL0_TUPLE_MUXA(SDIO0FCLKSEL_OFFSET, 4),
kNONE_to_SDIO0_CLK = CLKCTL0_TUPLE_MUXA(SDIO0FCLKSEL_OFFSET, 7),
kMAIN_CLK_to_SDIO1_CLK = CLKCTL0_TUPLE_MUXA(SDIO1FCLKSEL_OFFSET, 0),
kMAIN_PLL_to_SDIO1_CLK = CLKCTL0_TUPLE_MUXA(SDIO1FCLKSEL_OFFSET, 1),
kAUX0_PLL_to_SDIO1_CLK = CLKCTL0_TUPLE_MUXA(SDIO1FCLKSEL_OFFSET, 2),
kFRO_DIV2_to_SDIO1_CLK = CLKCTL0_TUPLE_MUXA(SDIO1FCLKSEL_OFFSET, 3),
kAUX1_PLL_to_SDIO1_CLK = CLKCTL0_TUPLE_MUXA(SDIO1FCLKSEL_OFFSET, 4),
kNONE_to_SDIO1_CLK = CLKCTL0_TUPLE_MUXA(SDIO1FCLKSEL_OFFSET, 7),
kMAIN_CLK_to_CTIMER0 = CLKCTL1_TUPLE_MUXA(CT32BIT0FCLKSEL_OFFSET, 0),
kFRO_DIV1_to_CTIMER0 = CLKCTL1_TUPLE_MUXA(CT32BIT0FCLKSEL_OFFSET, 1),
kAUDIO_PLL_to_CTIMER0 = CLKCTL1_TUPLE_MUXA(CT32BIT0FCLKSEL_OFFSET, 2),
kMASTER_CLK_to_CTIMER0 = CLKCTL1_TUPLE_MUXA(CT32BIT0FCLKSEL_OFFSET, 3),
k32K_WAKE_CLK_to_CTIMER0 = CLKCTL1_TUPLE_MUXA(CT32BIT0FCLKSEL_OFFSET, 4),
kNONE_to_CTIMER0 = CLKCTL1_TUPLE_MUXA(CT32BIT0FCLKSEL_OFFSET, 7),
kMAIN_CLK_to_CTIMER1 = CLKCTL1_TUPLE_MUXA(CT32BIT1FCLKSEL_OFFSET, 0),
kFRO_DIV1_to_CTIMER1 = CLKCTL1_TUPLE_MUXA(CT32BIT1FCLKSEL_OFFSET, 1),
kAUDIO_PLL_to_CTIMER1 = CLKCTL1_TUPLE_MUXA(CT32BIT1FCLKSEL_OFFSET, 2),
kMASTER_CLK_to_CTIMER1 = CLKCTL1_TUPLE_MUXA(CT32BIT1FCLKSEL_OFFSET, 3),
k32K_WAKE_CLK_to_CTIMER1 = CLKCTL1_TUPLE_MUXA(CT32BIT1FCLKSEL_OFFSET, 4),
kNONE_to_CTIMER1 = CLKCTL1_TUPLE_MUXA(CT32BIT1FCLKSEL_OFFSET, 7),
kMAIN_CLK_to_CTIMER2 = CLKCTL1_TUPLE_MUXA(CT32BIT2FCLKSEL_OFFSET, 0),
kFRO_DIV1_to_CTIMER2 = CLKCTL1_TUPLE_MUXA(CT32BIT2FCLKSEL_OFFSET, 1),
kAUDIO_PLL_to_CTIMER2 = CLKCTL1_TUPLE_MUXA(CT32BIT2FCLKSEL_OFFSET, 2),
kMASTER_CLK_to_CTIMER2 = CLKCTL1_TUPLE_MUXA(CT32BIT2FCLKSEL_OFFSET, 3),
k32K_WAKE_CLK_to_CTIMER2 = CLKCTL1_TUPLE_MUXA(CT32BIT2FCLKSEL_OFFS-

ET, 4),
kNONE_to_CTIMER2 = CLKCTL1_TUPLE_MUXA(CT32BIT2FCLKSEL_OFFSET, 7),
kMAIN_CLK_to_CTIMER3 = CLKCTL1_TUPLE_MUXA(CT32BIT3FCLKSEL_OFFSET, 0),
kFRO_DIV1_to_CTIMER3 = CLKCTL1_TUPLE_MUXA(CT32BIT3FCLKSEL_OFFSET, 1),
kAUDIO_PLL_to_CTIMER3 = CLKCTL1_TUPLE_MUXA(CT32BIT3FCLKSEL_OFFSET, 2),
kMASTER_CLK_to_CTIMER3 = CLKCTL1_TUPLE_MUXA(CT32BIT3FCLKSEL_OFFSET, 3),
k32K_WAKE_CLK_to_CTIMER3 = CLKCTL1_TUPLE_MUXA(CT32BIT3FCLKSEL_OFFSET, 4),
kNONE_to_CTIMER3 = CLKCTL1_TUPLE_MUXA(CT32BIT3FCLKSEL_OFFSET, 7),
kMAIN_CLK_to_CTIMER4 = CLKCTL1_TUPLE_MUXA(CT32BIT4FCLKSEL_OFFSET, 0),
kFRO_DIV1_to_CTIMER4 = CLKCTL1_TUPLE_MUXA(CT32BIT4FCLKSEL_OFFSET, 1),
kAUDIO_PLL_to_CTIMER4 = CLKCTL1_TUPLE_MUXA(CT32BIT4FCLKSEL_OFFSET, 2),
kMASTER_CLK_to_CTIMER4 = CLKCTL1_TUPLE_MUXA(CT32BIT4FCLKSEL_OFFSET, 3),
k32K_WAKE_CLK_to_CTIMER4 = CLKCTL1_TUPLE_MUXA(CT32BIT4FCLKSEL_OFFSET, 4),
kNONE_to_CTIMER4 = CLKCTL1_TUPLE_MUXA(CT32BIT4FCLKSEL_OFFSET, 7),
kMAIN_CLK_to_FLEXSPI0_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKSEL_OFFSET, 0),
kMAIN_PLL_to_FLEXSPI0_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKSEL_OFFSET, 1),
kAUX0_PLL_to_FLEXSPI0_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKSEL_OFFSET, 2),
kFRO_DIV1_to_FLEXSPI0_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKSEL_OFFSET, 3),
kAUX1_PLL_to_FLEXSPI0_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKSEL_OFFSET, 4),
kFRO_DIV4_to_FLEXSPI0_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKSEL_OFFSET, 5),
kFRO_DIV8_to_FLEXSPI0_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKSEL_OFFSET, 6),
kNONE_to_FLEXSPI0_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKSEL_OFFSET, 7),
kMAIN_CLK_to_FLEXSPI1_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI1FCLKSEL_OFFSET, 0),
kMAIN_PLL_to_FLEXSPI1_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI1FCLKSEL_OFFSET, 1),
kAUX0_PLL_to_FLEXSPI1_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI1FCLKSEL_OFFSET, 2),
kFRO_DIV1_to_FLEXSPI1_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI1FCLKSEL_OFFSET, 3),
kAUX1_PLL_to_FLEXSPI1_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI1FCLKSEL_OFFSE-

T, 4),
kNONE_to_FLEXSPI1_CLK = CLKCTL0_TUPLE_MUXA(FLEXSPI1FCLKSEL_OFFSET, 7),
kOSC_CLK_to_USB_CLK = CLKCTL0_TUPLE_MUXA(USBHSFCLKSEL_OFFSET, 0),
kMAIN_CLK_to_USB_CLK = CLKCTL0_TUPLE_MUXA(USBHSFCLKSEL_OFFSET, 1),
kAUX0_PLL_to_USB_CLK = CLKCTL0_TUPLE_MUXA(USBHSFCLKSEL_OFFSET, 3),
kNONE_to_USB_CLK = CLKCTL0_TUPLE_MUXA(USBHSFCLKSEL_OFFSET, 7),
kMAIN_CLK_to_SCT_CLK = CLKCTL0_TUPLE_MUXA(SCTFCLKSEL_OFFSET, 0),
kMAIN_PLL_to_SCT_CLK = CLKCTL0_TUPLE_MUXA(SCTFCLKSEL_OFFSET, 1),
kAUX0_PLL_to_SCT_CLK = CLKCTL0_TUPLE_MUXA(SCTFCLKSEL_OFFSET, 2),
kFRO_DIV1_to_SCT_CLK = CLKCTL0_TUPLE_MUXA(SCTFCLKSEL_OFFSET, 3),
kAUX1_PLL_to_SCT_CLK = CLKCTL0_TUPLE_MUXA(SCTFCLKSEL_OFFSET, 4),
kAUDIO_PLL_to_SCT_CLK = CLKCTL0_TUPLE_MUXA(SCTFCLKSEL_OFFSET, 5),
kNONE_to_SCT_CLK = CLKCTL0_TUPLE_MUXA(SCTFCLKSEL_OFFSET, 7),
kLPOSC_to_OSTIMER_CLK = CLKCTL1_TUPLE_MUXA(OSEVENTTFCLKSEL_OFFSET, 0),
kOSC32K_to_OSTIMER_CLK = CLKCTL1_TUPLE_MUXA(OSEVENTTFCLKSEL_OFFSET, 1),
kHCLK_to_OSTIMER_CLK = CLKCTL1_TUPLE_MUXA(OSEVENTTFCLKSEL_OFFSET, 2),
kNONE_to_OSTIMER_CLK = CLKCTL1_TUPLE_MUXA(OSEVENTTFCLKSEL_OFFSET,

7),
kFRO_DIV8_to_MCLK_CLK = CLKCTL1_TUPLE_MUXA(AUDIOMCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_MCLK_CLK = CLKCTL1_TUPLE_MUXA(AUDIOMCLKSEL_OFFSET, 1),
kNONE_to_MCLK_CLK = CLKCTL1_TUPLE_MUXA(AUDIOMCLKSEL_OFFSET, 7),
kFRO_DIV4_to_DMIC = CLKCTL1_TUPLE_MUXA(DMIC0FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_DMIC = CLKCTL1_TUPLE_MUXA(DMIC0FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_DMIC = CLKCTL1_TUPLE_MUXA(DMIC0FCLKSEL_OFFSET, 2),
kLPOSC_to_DMIC = CLKCTL1_TUPLE_MUXA(DMIC0FCLKSEL_OFFSET, 3),
k32K_WAKE_CLK_to_DMIC = CLKCTL1_TUPLE_MUXA(DMIC0FCLKSEL_OFFSET, 4),
kNONE_to_DMIC = CLKCTL1_TUPLE_MUXA(DMIC0FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM0 = CLKCTL1_TUPLE_MUXA(FC0FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM0 = CLKCTL1_TUPLE_MUXA(FC0FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM0 = CLKCTL1_TUPLE_MUXA(FC0FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM0 = CLKCTL1_TUPLE_MUXA(FC0FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM0 = CLKCTL1_TUPLE_MUXA(FC0FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM1 = CLKCTL1_TUPLE_MUXA(FC1FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM1 = CLKCTL1_TUPLE_MUXA(FC1FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM1 = CLKCTL1_TUPLE_MUXA(FC1FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM1 = CLKCTL1_TUPLE_MUXA(FC1FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM1 = CLKCTL1_TUPLE_MUXA(FC1FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM2 = CLKCTL1_TUPLE_MUXA(FC2FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM2 = CLKCTL1_TUPLE_MUXA(FC2FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM2 = CLKCTL1_TUPLE_MUXA(FC2FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM2 = CLKCTL1_TUPLE_MUXA(FC2FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM2 = CLKCTL1_TUPLE_MUXA(FC2FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM3 = CLKCTL1_TUPLE_MUXA(FC3FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM3 = CLKCTL1_TUPLE_MUXA(FC3FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM3 = CLKCTL1_TUPLE_MUXA(FC3FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM3 = CLKCTL1_TUPLE_MUXA(FC3FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM3 = CLKCTL1_TUPLE_MUXA(FC3FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM4 = CLKCTL1_TUPLE_MUXA(FC4FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM4 = CLKCTL1_TUPLE_MUXA(FC4FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM4 = CLKCTL1_TUPLE_MUXA(FC4FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM4 = CLKCTL1_TUPLE_MUXA(FC4FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM4 = CLKCTL1_TUPLE_MUXA(FC4FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM5 = CLKCTL1_TUPLE_MUXA(FC5FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM5 = CLKCTL1_TUPLE_MUXA(FC5FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM5 = CLKCTL1_TUPLE_MUXA(FC5FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM5 = CLKCTL1_TUPLE_MUXA(FC5FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM5 = CLKCTL1_TUPLE_MUXA(FC5FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM6 = CLKCTL1_TUPLE_MUXA(FC6FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM6 = CLKCTL1_TUPLE_MUXA(FC6FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM6 = CLKCTL1_TUPLE_MUXA(FC6FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM6 = CLKCTL1_TUPLE_MUXA(FC6FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM6 = CLKCTL1_TUPLE_MUXA(FC6FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM7 = CLKCTL1_TUPLE_MUXA(FC7FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM7 = CLKCTL1_TUPLE_MUXA(FC7FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM7 = CLKCTL1_TUPLE_MUXA(FC7FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM7 = CLKCTL1_TUPLE_MUXA(FC7FCLKSEL_OFFSET, 3),

2),
kFRG_to_FLEXCOMM10 = CLKCTL1_TUPLE_MUXA(FC10FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM10 = CLKCTL1_TUPLE_MUXA(FC10FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM11 = CLKCTL1_TUPLE_MUXA(FC11FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM11 = CLKCTL1_TUPLE_MUXA(FC11FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM11 = CLKCTL1_TUPLE_MUXA(FC11FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM11 = CLKCTL1_TUPLE_MUXA(FC11FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM11 = CLKCTL1_TUPLE_MUXA(FC11FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM12 = CLKCTL1_TUPLE_MUXA(FC12FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM12 = CLKCTL1_TUPLE_MUXA(FC12FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM12 = CLKCTL1_TUPLE_MUXA(FC12FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM12 = CLKCTL1_TUPLE_MUXA(FC12FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM12 = CLKCTL1_TUPLE_MUXA(FC12FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM13 = CLKCTL1_TUPLE_MUXA(FC13FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM13 = CLKCTL1_TUPLE_MUXA(FC13FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM13 = CLKCTL1_TUPLE_MUXA(FC13FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM13 = CLKCTL1_TUPLE_MUXA(FC13FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM13 = CLKCTL1_TUPLE_MUXA(FC13FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM14 = CLKCTL1_TUPLE_MUXA(FC14FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM14 = CLKCTL1_TUPLE_MUXA(FC14FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM14 = CLKCTL1_TUPLE_MUXA(FC14FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM14 = CLKCTL1_TUPLE_MUXA(FC14FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM14 = CLKCTL1_TUPLE_MUXA(FC14FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM15 = CLKCTL1_TUPLE_MUXA(FC15FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM15 = CLKCTL1_TUPLE_MUXA(FC15FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM15 = CLKCTL1_TUPLE_MUXA(FC15FCLKSEL_OFFSET, 2),
kFRG_to_FLEXCOMM15 = CLKCTL1_TUPLE_MUXA(FC15FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM15 = CLKCTL1_TUPLE_MUXA(FC15FCLKSEL_OFFSET, 7),
kFRO_DIV4_to_FLEXCOMM16 = CLKCTL1_TUPLE_MUXA(FC16FCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXCOMM16 = CLKCTL1_TUPLE_MUXA(FC16FCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXCOMM16 = CLKCTL1_TUPLE_MUXA(FC16FCLKSEL_OFFSET,

2),
kFRG_to_FLEXCOMM16 = CLKCTL1_TUPLE_MUXA(FC16FCLKSEL_OFFSET, 3),
kNONE_to_FLEXCOMM16 = CLKCTL1_TUPLE_MUXA(FC16FCLKSEL_OFFSET, 7),
kFRO_DIV2_to_FLEXIO = CLKCTL1_TUPLE_MUXA(FLEXIOCLKSEL_OFFSET, 0),
kAUDIO_PLL_to_FLEXIO = CLKCTL1_TUPLE_MUXA(FLEXIOCLKSEL_OFFSET, 1),
kMASTER_CLK_to_FLEXIO = CLKCTL1_TUPLE_MUXA(FLEXIOCLKSEL_OFFSET, 2),
kFRG_to_FLEXIO = CLKCTL1_TUPLE_MUXA(FLEXIOCLKSEL_OFFSET, 3),
kNONE_to_FLEXIO = CLKCTL1_TUPLE_MUXA(FLEXIOCLKSEL_OFFSET, 7),
kMAIN_CLK_to_I3C_CLK = CLKCTL1_TUPLE_MUXA(I3C01FCLKSEL_OFFSET, 0),
kFRO_DIV8_to_I3C_CLK = CLKCTL1_TUPLE_MUXA(I3C01FCLKSEL_OFFSET, 1),
kNONE_to_I3C_CLK = CLKCTL1_TUPLE_MUXA(I3C01FCLKSEL_OFFSET, 7),
kI3C_CLK_to_I3C_TC_CLK = CLKCTL1_TUPLE_MUXA(I3C01FCLKSTCSEL_OFFSET, 0),
kLPOSC_to_I3C_TC_CLK = CLKCTL1_TUPLE_MUXA(I3C01FCLKSTCSEL_OFFSET, 1),
kNONE_to_I3C_TC_CLK = CLKCTL1_TUPLE_MUXA(I3C01FCLKSTCSEL_OFFSET, 7),
kMAIN_CLK_to_ACMP_CLK = CLKCTL1_TUPLE_MUXA(ACMP0FCLKSEL_OFFSET, 0),
kFRO_DIV4_to_ACMP_CLK = CLKCTL1_TUPLE_MUXA(ACMP0FCLKSEL_OFFSET, 1),
kAUX0_PLL_to_ACMP_CLK = CLKCTL1_TUPLE_MUXA(ACMP0FCLKSEL_OFFSET, 2),
kAUX1_PLL_to_ACMP_CLK = CLKCTL1_TUPLE_MUXA(ACMP0FCLKSEL_OFFSET, 3),
kNONE_to_ACMP_CLK = CLKCTL1_TUPLE_MUXA(ACMP0FCLKSEL_OFFSET, 7),
kOSC_CLK_to_ADC_CLK,
kLPOSC_to_ADC_CLK,
kFRO_DIV4_to_ADC_CLK,
kMAIN_PLL_to_ADC_CLK = CLKCTL0_TUPLE_MUXA(ADC0FCLKSEL1_OFFSET, 1),
kAUX0_PLL_to_ADC_CLK = CLKCTL0_TUPLE_MUXA(ADC0FCLKSEL1_OFFSET, 2),
kAUX1_PLL_to_ADC_CLK = CLKCTL0_TUPLE_MUXA(ADC0FCLKSEL1_OFFSET, 3),
kOSC_CLK_to_CLKOUT,
kLPOSC_to_CLKOUT,
kFRO_DIV2_to_CLKOUT,
kMAIN_CLK_to_CLKOUT,
kDSP_MAIN_to_CLKOUT,
kMAIN_PLL_to_CLKOUT = CLKCTL1_TUPLE_MUXA(CLKOUTSEL1_OFFSET, 1),
kAUX0_PLL_to_CLKOUT = CLKCTL1_TUPLE_MUXA(CLKOUTSEL1_OFFSET, 2),
kDSP_PLL_to_CLKOUT = CLKCTL1_TUPLE_MUXA(CLKOUTSEL1_OFFSET, 3),
kAUX1_PLL_to_CLKOUT = CLKCTL1_TUPLE_MUXA(CLKOUTSEL1_OFFSET, 4),
kAUDIO_PLL_to_CLKOUT = CLKCTL1_TUPLE_MUXA(CLKOUTSEL1_OFFSET, 5),
kOSC32K_to_CLKOUT = CLKCTL1_TUPLE_MUXA(CLKOUTSEL1_OFFSET, 6),
kNONE_to_CLKOUT = CLKCTL1_TUPLE_MUXA(CLKOUTSEL1_OFFSET, 7),
kMAIN_CLK_to_GPU_CLK = CLKCTL0_TUPLE_MUXA(GPUCLKSEL_OFFSET, 0),
kFRO_DIV1_to_GPU_CLK = CLKCTL0_TUPLE_MUXA(GPUCLKSEL_OFFSET, 1),
kMAIN_PLL_to_GPU_CLK = CLKCTL0_TUPLE_MUXA(GPUCLKSEL_OFFSET, 2),
kAUX0_PLL_to_GPU_CLK = CLKCTL0_TUPLE_MUXA(GPUCLKSEL_OFFSET, 3),
kAUX1_PLL_to_GPU_CLK = CLKCTL0_TUPLE_MUXA(GPUCLKSEL_OFFSET, 4),
kNONE_to_GPU_CLK = CLKCTL0_TUPLE_MUXA(GPUCLKSEL_OFFSET, 7),
kFRO_DIV1_to_MIPI_DPHY_CLK = CLKCTL0_TUPLE_MUXA(DPHYCLKSEL_OFFSET,

0),
kMAIN_PLL_to_MIPI_DPHY_CLK = CLKCTL0_TUPLE_MUXA(DPHYCLKSEL_OFFSET, 1),
kAUX0_PLL_to_MIPI_DPHY_CLK = CLKCTL0_TUPLE_MUXA(DPHYCLKSEL_OFFSET, 2),
kAUX1_PLL_to_MIPI_DPHY_CLK = CLKCTL0_TUPLE_MUXA(DPHYCLKSEL_OFFSET, 3),
kNONE_to_MIPI_DPHY_CLK = CLKCTL0_TUPLE_MUXA(DPHYCLKSEL_OFFSET, 7),
kFRO_DIV1_to_MIPI_DPHYESC_CLK = CLKCTL0_TUPLE_MUXA(DPHYESCCLKSEL_OFFSET, 0),
kFRO_DIV16_to_MIPI_DPHYESC_CLK = CLKCTL0_TUPLE_MUXA(DPHYESCCLKSEL_OFFSET, 1),
kAUX0_PLL_to_MIPI_DPHYESC_CLK = CLKCTL0_TUPLE_MUXA(DPHYESCCLKSEL_OFFSET, 2),
kAUX1_PLL_to_MIPI_DPHYESC_CLK = CLKCTL0_TUPLE_MUXA(DPHYESCCLKSEL_OFFSET, 3),
kMIPI_DPHY_CLK_to_DCPIXEL_CLK = CLKCTL0_TUPLE_MUXA(DCPIXELCLKSEL_OFFSET, 0),
kMAIN_CLK_to_DCPIXEL_CLK = CLKCTL0_TUPLE_MUXA(DCPIXELCLKSEL_OFFSET, 1),
kFRO_DIV1_to_DCPIXEL_CLK = CLKCTL0_TUPLE_MUXA(DCPIXELCLKSEL_OFFSET, 2),
kMAIN_PLL_to_DCPIXEL_CLK = CLKCTL0_TUPLE_MUXA(DCPIXELCLKSEL_OFFSET, 3),
kAUX0_PLL_to_DCPIXEL_CLK = CLKCTL0_TUPLE_MUXA(DCPIXELCLKSEL_OFFSET, 4),
kAUX1_PLL_to_DCPIXEL_CLK = CLKCTL0_TUPLE_MUXA(DCPIXELCLKSEL_OFFSET, 5),
kNONE_to_DCPIXEL_CLK = CLKCTL0_TUPLE_MUXA(DCPIXELCLKSEL_OFFSET, 7) }

The enumerator of clock attach Id.

- enum clock_div_name_t {
kCLOCK_DivAudioPllClk = CLKCTL1_TUPLE_MUXA(AUDIOPLLCLKDIV_OFFSET, 0),
kCLOCK_DivMainPllClk = CLKCTL0_TUPLE_MUXA(MAINPLLCLKDIV_OFFSET, 0),
kCLOCK_DivDspPllClk = CLKCTL0_TUPLE_MUXA(DSPPLLCLKDIV_OFFSET, 0),
kCLOCK_DivAux0PllClk = CLKCTL0_TUPLE_MUXA(AUX0PLLCLKDIV_OFFSET, 0),
kCLOCK_DivAux1PllClk = CLKCTL0_TUPLE_MUXA(AUX1PLLCLKDIV_OFFSET, 0),
kCLOCK_DivPfc0Clk = CLKCTL0_TUPLE_MUXA(PFC0CLKDIV_OFFSET, 0),
kCLOCK_DivPfc1Clk = CLKCTL0_TUPLE_MUXA(PFC1CLKDIV_OFFSET, 0),
kCLOCK_DivSysCpuAhbClk = CLKCTL0_TUPLE_MUXA(SYSCPUAHBCLKDIV_OFFSET, 0),
kCLOCK_Div32KhzWakeClk = CLKCTL0_TUPLE_MUXA(A32KHZWAKECLKDIV_OFFSE-

T, 0),
kCLOCK_DivSystickClk = CLKCTL0_TUPLE_MUXA(SYSTICKFCLKDIV_OFFSET, 0),
kCLOCK_DivSdio0Clk = CLKCTL0_TUPLE_MUXA(SDIO0FCLKDIV_OFFSET, 0),
kCLOCK_DivSdio1Clk = CLKCTL0_TUPLE_MUXA(SDIO1FCLKDIV_OFFSET, 0),
kCLOCK_DivFlexspi0Clk = CLKCTL0_TUPLE_MUXA(FLEXSPI0FCLKDIV_OFFSET, 0),
kCLOCK_DivFlexspi1Clk = CLKCTL0_TUPLE_MUXA(FLEXSPI1FCLKDIV_OFFSET, 0),
kCLOCK_DivUsbHsFclk = CLKCTL0_TUPLE_MUXA(USBHSFCLKDIV_OFFSET, 0),
kCLOCK_DivSctClk = CLKCTL0_TUPLE_MUXA(SCTFCLKDIV_OFFSET, 0),
kCLOCK_DivMclkClk = CLKCTL1_TUPLE_MUXA(AUDIOMCLKDIV_OFFSET, 0),
kCLOCK_DivDmicClk = CLKCTL1_TUPLE_MUXA(DMIC0FCLKDIV_OFFSET, 0),
kCLOCK_DivPLLFRGClk = CLKCTL1_TUPLE_MUXA(FRGPLLCLKDIV_OFFSET, 0),
kCLOCK_DivFlexioClk = CLKCTL1_TUPLE_MUXA(FLEXIOCLKDIV_OFFSET, 0),
kCLOCK_DivI3cClk = CLKCTL1_TUPLE_MUXA(I3C01FCLKDIV_OFFSET, 0),
kCLOCK_DivI3cTcClk = CLKCTL1_TUPLE_MUXA(I3C01FCLKSTCDIV_OFFSET, 0),
kCLOCK_DivI3cSlowClk = CLKCTL1_TUPLE_MUXA(I3C01FCLKSDIV_OFFSET, 0),
kCLOCK_DivDspCpuClk = CLKCTL1_TUPLE_MUXA(DSPCPUCLKDIV_OFFSET, 0),
kCLOCK_DivAcmpClk = CLKCTL1_TUPLE_MUXA(ACMP0FCLKDIV_OFFSET, 0),
kCLOCK_DivAdcClk = CLKCTL0_TUPLE_MUXA(ADC0FCLKDIV_OFFSET, 0),
kCLOCK_DivLowFreqClk = CLKCTL0_TUPLE_MUXA(LOWFREQCLKDIV_OFFSET, 0),
kCLOCK_DivClockOut = CLKCTL1_TUPLE_MUXA(CLKOUTFCLKDIV_OFFSET, 0),
kCLOCK_DivGpuClk = CLKCTL0_TUPLE_MUXA(GPUCLKDIV_OFFSET, 0),
kCLOCK_DivDcPixelClk = CLKCTL0_TUPLE_MUXA(DCPIXELCLKDIV_OFFSET, 0),
kCLOCK_DivDphyClk = CLKCTL0_TUPLE_MUXA(DPHYCLKDIV_OFFSET, 0),
kCLOCK_DivDphyEscRxClk = CLKCTL0_TUPLE_MUXA(DPHYESCRXCLKDIV_OFFSET, 0),
kCLOCK_DivDphyEscTxClk = CLKCTL0_TUPLE_MUXA(DPHYESCTXCLKDIV_OFFSET, 0) }

Clock dividers.

- enum sys_pll_src_t {
  kCLOCK_SysPllFroDiv8Clk = 0,
  kCLOCK_SysPllXtalIn = 1,
  kCLOCK_SysPllNone = 7 }

  SysPLL Reference Input Clock Source.
- enum sys_pll_mult_t {
  kCLOCK_SysPllMult16 = 0,
  kCLOCK_SysPllMult17,
  kCLOCK_SysPllMult18,
  kCLOCK_SysPllMult19,
  kCLOCK_SysPllMult20,
  kCLOCK_SysPllMult21,
  kCLOCK_SysPllMult22 }

  SysPLL Multiplication Factor.
- enum audio_pll_src_t {
  kCLOCK_AudioPllFroDiv8Clk = 0,
  kCLOCK_AudioPllXtalIn = 1,

kCLOCK_AudioPllNone = 7 }
AudioPll Reference Input Clock Source.

- enum audio_pll_mult_t {
  kCLOCK_AudioPllMult16 = 0,
  kCLOCK_AudioPllMult17,
  kCLOCK_AudioPllMult18,
  kCLOCK_AudioPllMult19,
  kCLOCK_AudioPllMult20,
  kCLOCK_AudioPllMult21,
  kCLOCK_AudioPllMult22 }
  AudioPll Multiplication Factor.
- enum clock_fro_output_en_t {
  kCLOCK_FroDiv1OutEn = CLKCTL0_FRODIVOEN_FRO_DIV1_O_EN_MASK,
  kCLOCK_FroDiv2OutEn = CLKCTL0_FRODIVOEN_FRO_DIV2_O_EN_MASK,
  kCLOCK_FroDiv4OutEn = CLKCTL0_FRODIVOEN_FRO_DIV4_O_EN_MASK,
  kCLOCK_FroDiv8OutEn = CLKCTL0_FRODIVOEN_FRO_DIV8_O_EN_MASK,
  kCLOCK_FroDiv16OutEn = CLKCTL0_FRODIVOEN_FRO_DIV16_O_EN_MASK }
  FRO output enable.
- enum clock_fro_freq_t {
  kCLOCK_Fro192M,
  kCLOCK_Fro96M }
  FRO frequence configuration.

Functions

- void CLOCK_AttachClk (clock_attach_id_t connection)
  Configure the clock selection muxes.
- void CLOCK_SetClkDiv (clock_div_name_t div_name, uint32_t divider)
  Setup peripheral clock dividers.
- uint32_t CLOCK_GetFreq (clock_name_t clockName)
  Return Frequency of selected clock.
- uint32_t CLOCK_GetFRGClock (uint32_t id)
  Return Input frequency for the Fractional baud rate generator.
- void CLOCK_SetFRGClock (const clock_frg_clk_config_t *config)
  Set output of the Fractional baud rate generator.
- uint32_t CLOCK_GetSysPllFreq (void)
  Return Frequency of SYSPLL.
- uint32_t CLOCK_GetSysPfdFreq (clock_pfd_t pfd)
  Get current output frequency of specific System PLL PFD.
- uint32_t CLOCK_GetAudioPllFreq (void)
  Return Frequency of AUDIO PLL.
- uint32_t CLOCK_GetAudioPfdFreq (clock_pfd_t pfd)
  Get current output frequency of specific Audio PLL PFD.
- uint32_t CLOCK_GetMainClkFreq (void)
  Return Frequency of main clk.
- uint32_t CLOCK_GetDspMainClkFreq (void)
  Return Frequency of DSP main clk.
- uint32_t CLOCK_GetAcmpClkFreq (void)
  Return Frequency of ACMP clk.

- uint32_t CLOCK_GetDmicClkFreq (void)
  Return Frequency of DMIC clk.
- uint32_t CLOCK_GetUsbClkFreq (void)
  Return Frequency of USB clk.
- uint32_t CLOCK_GetSdioClkFreq (uint32_t id)
  Return Frequency of SDIO clk.
- uint32_t CLOCK_GetI3cClkFreq (void)
  Return Frequency of I3C clk.
- uint32_t CLOCK_GetSystickClkFreq (void)
  Return Frequency of systick clk.
- uint32_t CLOCK_GetWdtClkFreq (uint32_t id)
  Return Frequency of WDT clk.
- uint32_t CLOCK_GetMclkClkFreq (void)
  Return output Frequency of mclk.
- uint32_t CLOCK_GetSctClkFreq (void)
  Return Frequency of sct.
- void CLOCK_EnableSysOscClk (bool enable, bool enableLowPower, uint32_t delay_us)
  Enable/Disable sys osc clock from external crystal clock.
- void CLOCK_EnableFroClk (uint32_t divOutEnable)
  Enable/Disable FRO clock output.
- void CLOCK_EnableFroClkRange (clock_fro_freq_t froFreq, uint32_t divOutEnable)
  Enable/Disable FRO192M or FRO96M clock output.
- void CLOCK_EnableLpOscClk (void)
  Enable LPOSC 1MHz clock.
- static uint32_t CLOCK_GetXtalInClkFreq (void)
  Return Frequency of sys osc Clock.
- static uint32_t CLOCK_GetMclkInClkFreq (void)
  Return Frequency of MCLK Input Clock.
- static uint32_t CLOCK_GetLpOscFreq (void)
  Return Frequency of Lower power osc.
- static uint32_t CLOCK_GetOsc32KFreq (void)
  Return Frequency of 32kHz osc.
- static void CLOCK_EnableOsc32K (bool enable)
  Enables and disables 32kHz osc.
- static uint32_t CLOCK_GetWakeClk32KFreq (void)
  Return Frequency of 32khz wake clk.
- static void CLOCK_SetXtalFreq (uint32_t freq)
  Set the XTALIN (system OSC) frequency based on board setting.
- static void CLOCK_SetClkinFreq (uint32_t freq)
  Set the CLKIN (CLKIN pin) frequency based on board setting.
- static void CLOCK_SetMclkFreq (uint32_t freq)
  Set the MCLK IN frequency based on board setting.
- uint32_t CLOCK_GetFlexcommClkFreq (uint32_t id)
  Return Frequency of Flexcomm functional Clock.
- uint32_t CLOCK_GetFlexioClkFreq (void)
  Return Frequency of Flexio functional Clock.
- uint32_t CLOCK_GetCtimerClkFreq (uint32_t id)
  Return Frequency of Ctimer Clock.
- uint32_t CLOCK_GetClockOutClkFreq (void)
  Return Frequency of ClockOut.
- uint32_t CLOCK_GetAdcClkFreq (void)

Return Frequency of Adc Clock.
- uint32_t CLOCK_GetFlexspiClkFreq (uint32_t id)
  Return Frequency of FLEXSPI Clock.
- uint32_t CLOCK_GetGpuClkFreq (void)
  Return Frequency of GPU functional Clock.
- uint32_t CLOCK_GetDcPixelClkFreq (void)
  Return Frequency of DCNano Pixel functional Clock.
- uint32_t CLOCK_GetMipiDphyClkFreq (void)
  Return Frequency of MIPI DPHY functional Clock.
- uint32_t CLOCK_GetMipiDphyEscRxClkFreq (void)
  Return Frequency of MIPI DPHY Esc RX functional Clock.
- uint32_t CLOCK_GetMipiDphyEscTxClkFreq (void)
  Return Frequency of MIPI DPHY Esc Tx functional Clock.
- void CLOCK_InitSysPll (const clock_sys_pll_config_t *config)
  Initialize the System PLL.
- static void CLOCK_DeinitSysPll (void)
  brief Deinit the System PLL.
- void CLOCK_InitSysPfd (clock_pfd_t pfd, uint8_t divider)
  Initialize the System PLL PFD.
- static void CLOCK_DeinitSysPfd (clock_pfd_t pfd)
  brief Disable the audio PLL PFD.
- void CLOCK_InitAudioPll (const clock_audio_pll_config_t *config)
  Initialize the audio PLL.
- static void CLOCK_DeinitAudioPll (void)
  brief Deinit the Audio PLL.
- void CLOCK_InitAudioPfd (clock_pfd_t pfd, uint8_t divider)
  Initialize the audio PLL PFD.
- static void CLOCK_DeinitAudioPfd (uint32_t pfd)
  brief Disable the audio PLL PFD.
- void CLOCK_EnableFroTuning (bool enable)
  Enable/Disable FRO tuning.
- void CLOCK_EnableUsbHs0DeviceClock (clock_attach_id_t src, uint8_t divider)
  Enable USB HS device clock.
- void CLOCK_DisableUsbHs0DeviceClock (void)
  Disable USB HS device clock.
- void CLOCK_EnableUsbHs0HostClock (clock_attach_id_t src, uint8_t divider)
  Enable USB HS host clock.
- void CLOCK_DisableUsbHs0HostClock (void)
  Disable USB HS host clock.
- bool CLOCK_EnableUsbHs0PhyPllClock (clock_attach_id_t src, uint32_t freq)
  brief Enable USB hs0PhyPll clock.
- void CLOCK_DisableUsbHs0PhyPllClock (void)
  Disable USB hs0PhyPll clock.

Variables

- volatile uint32_t g_xtalFreq
  External XTAL (SYSOSC) clock frequency.
- volatile uint32_t g_clkinFreq
  External CLK_IN pin clock frequency (clkin) clock frequency.
- volatile uint32_t g_mclkFreq

External MCLK IN clock frequency.

Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 6, 1))
  CLOCK driver version 2.6.1.

4.2 Data Structure Documentation

4.2.1 struct clock_sys_pll_config_t

Data Fields

- sys_pll_src_t sys_pll_src
  Reference Input Clock Source.
- uint32_t numerator
  30 bit numerator of fractional loop divider.
- uint32_t denominator
  30 bit numerator of fractional loop divider.
- sys_pll_mult_t sys_pll_mult
  Multiplication Factor.

Field Documentation

(1) uint32_t clock_sys_pll_config_t::numerator

(2) uint32_t clock_sys_pll_config_t::denominator

4.2.2 struct clock_audio_pll_config_t

Data Fields

- audio_pll_src_t audio_pll_src
  Reference Input Clock Source.
- uint32_t numerator
  30 bit numerator of fractional loop divider.
- uint32_t denominator
  30 bit numerator of fractional loop divider.
- audio_pll_mult_t audio_pll_mult
  Multiplication Factor.

Field Documentation

(1) uint32_t clock_audio_pll_config_t::numerator

(2) uint32_t clock_audio_pll_config_t::denominator

4.2.3 struct clock_frg_clk_config_t

Public Types

- enum {
  kCLOCK_FrgMainClk = 0,
  kCLOCK_FrgPllDiv,
  kCLOCK_FrgFroDiv4 }

Data Fields

- uint8_t num
  FRG clock, [0 - 16]: Flexcomm, [17]: Flexio.
- uint8_t divider
  Denominator of the fractional divider.
- uint8_t mult
  Numerator of the fractional divider.

Member Enumeration Documentation

(1) anonymous enum

Enumerator

kCLOCK_FrgMainClk Main System clock.
kCLOCK_FrgPllDiv Main pll clock divider.
kCLOCK_FrgFroDiv4 FRO_DIV4.

Field Documentation

(1) uint8_t clock_frg_clk_config_t::divider

(2) uint8_t clock_frg_clk_config_t::mult

4.3 Macro Definition Documentation

4.3.1 #define MIPI_DSI_HOST_CLOCKS

Value:

```
{                                   \
        kCLOCK_MipiDsiCtrl          \
    }
```

4.3.2 #define LCDIF_CLOCKS

Value:

```
{                       \
        kCLOCK_DisplayCtrl \
    }
```

4.3.3 #define SCT_CLOCKS

Value:

```
{               \
        kCLOCK_Sct \
    }
```

4.3.4 #define USBD_CLOCKS

Value:

```
{                       \
        kCLOCK_UsbhsDevice \
    }
```

4.3.5 #define LPADC_CLOCKS

Value:

```
{                 \
        kCLOCK_Adc0  \
    }
```

4.3.6 #define CMP_CLOCKS

Value:

```
{                 \
        kCLOCK_Acmp0 \
    }
```

4.3.7 #define WWDT_CLOCKS

Value:

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

4.3.8 #define UTICK_CLOCKS

Value:

```
{                   \
        kCLOCK_Utick0 \
    }
```

4.3.9 #define FLEXIO_CLOCKS

Value:

```
{                   \
        kCLOCK_Flexio \
    }
```

4.3.10 #define FLEXCOMM_CLOCKS

Value:

```
{
            \
        kCLOCK_Flexcomm0, kCLOCK_Flexcomm1,
      kCLOCK_Flexcomm2, kCLOCK_Flexcomm3,
      kCLOCK_Flexcomm4, kCLOCK_Flexcomm5, \
            kCLOCK_Flexcomm6, kCLOCK_Flexcomm7,
      kCLOCK_Flexcomm8, kCLOCK_Flexcomm9,
      kCLOCK_Flexcomm10,              \
            kCLOCK_Flexcomm11, kCLOCK_Flexcomm12,
      kCLOCK_Flexcomm13, kCLOCK_Flexcomm14,
      kCLOCK_Flexcomm15,          \
            kCLOCK_Flexcomm16
                                          \
    }
```

4.3.11 #define USART_CLOCKS

Value:

```
{
        \
        kCLOCK_Usart0, kCLOCK_Usart1,
      kCLOCK_Usart2, kCLOCK_Usart3, kCLOCK_Usart4,
      kCLOCK_Usart5, kCLOCK_Usart6, \
            kCLOCK_Usart7, kCLOCK_Usart8,
      kCLOCK_Usart9, kCLOCK_Usart10, kCLOCK_Usart11,
      kCLOCK_Usart12,         \
            kCLOCK_Usart13
                                    \
    }
```

4.3.12 #define I2C_CLOCKS

Value:

```
{
        \
         kCLOCK_I2c0, kCLOCK_I2c1, kCLOCK_I2c2,
      kCLOCK_I2c3, kCLOCK_I2c4, kCLOCK_I2c5,
      kCLOCK_I2c6, kCLOCK_I2c7, \
            kCLOCK_I2c8, kCLOCK_I2c9,
      kCLOCK_I2c10, kCLOCK_I2c11, kCLOCK_I2c12,
      kCLOCK_I2c13, kCLOCK_I2c15      \
    }
```

4.3.13 #define SPI_CLOCKS

Value:

```
{
        \
         kCLOCK_Spi0, kCLOCK_Spi1, kCLOCK_Spi2,
      kCLOCK_Spi3, kCLOCK_Spi4, kCLOCK_Spi5,
      kCLOCK_Spi6, kCLOCK_Spi7, \
            kCLOCK_Spi8, kCLOCK_Spi9,
      kCLOCK_Spi10, kCLOCK_Spi11, kCLOCK_Spi12,
      kCLOCK_Spi13, kCLOCK_Spi14,     \
            kCLOCK_Spi16
                                  \
    }
```

4.3.14 #define I2S_CLOCKS

Value:

```
{
        \
         kCLOCK_I2s0, kCLOCK_I2s1, kCLOCK_I2s2,
      kCLOCK_I2s3, kCLOCK_I2s4, kCLOCK_I2s5,
      kCLOCK_I2s6, kCLOCK_I2s7, \
            kCLOCK_I2s8, kCLOCK_I2s9,
      kCLOCK_I2s10, kCLOCK_I2s11, kCLOCK_I2s12,
      kCLOCK_I2s13                    \
    }
```

4.3.15 #define DMIC_CLOCKS

Value:

```
{                 \
         kCLOCK_Dmic0 \
    }
```

4.3.16 #define DMA_CLOCKS

Value:

```
{                                   \
        kCLOCK_Dmac0, kCLOCK_Dmac1 \
    }
```

4.3.17 #define CRC_CLOCKS

Value:

```
{                \
        kCLOCK_Crc \
    }
```

4.3.18 #define GPIO_CLOCKS

Value:

```
{
      \
        kCLOCK_HsGpio0, kCLOCK_HsGpio1,
      kCLOCK_HsGpio2, kCLOCK_HsGpio3, kCLOCK_HsGpio4,
      kCLOCK_HsGpio5, \
            kCLOCK_HsGpio6, kCLOCK_HsGpio7
                                                \
    }
```

4.3.19 #define PINT_CLOCKS

Value:

```
{                       \
        kCLOCK_GpioIntCtl \
    }
```

4.3.20 #define I3C_CLOCKS

Value:

```
{                                   \
        kCLOCK_I3c0, kCLOCK_I3c1 \
    }
```

4.3.21 #define MRT_CLOCKS

Value:

```
{                \
        kCLOCK_Mrt0 \
    }
```

4.3.22 #define RTC_CLOCKS

Value:

```
{               \
        kCLOCK_Rtc \
    }
```

4.3.23 #define CTIMER_CLOCKS

Value:

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

4.3.24 #define CLK_GATE_REG_OFFSET_SHIFT 8U

4.3.25 #define SYSPLL0CLKSEL_OFFSET 0x200

[31 30 29:28 27:25 24:14 13:11 10:0] [CLKCTL index]:[FRODIVSEL onoff]:[FRODIVSEL]:[MUXB choice]:[MUXB offset]:[MUXA choice]:[MUXA offset] FRODIVSEL onoff '1' means need to set FRODIVSEL. MUX offset 0 means end of descriptor.

4.4 Enumeration Type Documentation

4.4.1 enum clock_ip_name_t

Enumerator

kCLOCK_IpInvalid Invalid Ip Name.
kCLOCK_Dsp Clock gate name: Dsp.
kCLOCK_RomCtrlr Clock gate name: RomCtrlr.

kCLOCK_AxiSwitch Clock gate name: AxiSwitch.
kCLOCK_AxiCtrl Clock gate name: AxiCtrl.
kCLOCK_PowerQuad Clock gate name: PowerQuad.
kCLOCK_Casper Clock gate name: Casper.
kCLOCK_HashCrypt Clock gate name: HashCrypt.
kCLOCK_Puf Clock gate name: Puf.
kCLOCK_Rng Clock gate name: Rng.
kCLOCK_Flexspi0 Clock gate name: Flexspi0.
kCLOCK_OtpCtrl Clock gate name: OtpCtrl.
kCLOCK_Flexspi1 Clock gate name: Flexspi1.
kCLOCK_UsbhsPhy Clock gate name: UsbhsPhy.
kCLOCK_UsbhsDevice Clock gate name: UsbhsDevice.
kCLOCK_UsbhsHost Clock gate name: UsbhsHost.
kCLOCK_UsbhsSram Clock gate name: UsbhsSram.
kCLOCK_Sct Clock gate name: Sct.
kCLOCK_Gpu Clock gate name: Gpu.
kCLOCK_DisplayCtrl Clock gate name: DisplayCtrl.
kCLOCK_MipiDsiCtrl Clock gate name: MipiDsiCtrl.
kCLOCK_Smartdma Clock gate name: Smartdma.
kCLOCK_Sdio0 Clock gate name: Sdio0.
kCLOCK_Sdio1 Clock gate name: Sdio1.
kCLOCK_Acmp0 Clock gate name: Acmp0.
kCLOCK_Adc0 Clock gate name: Adc0.
kCLOCK_ShsGpio0 Clock gate name: ShsGpio0.
kCLOCK_Utick0 Clock gate name: Utick0.
kCLOCK_Wwdt0 Clock gate name: Wwdt0.
kCLOCK_Pmc Clock gate name: Pmc.
kCLOCK_Flexcomm0 Clock gate name: Flexcomm0.
kCLOCK_Flexcomm1 Clock gate name: Flexcomm1.
kCLOCK_Flexcomm2 Clock gate name: Flexcomm2.
kCLOCK_Flexcomm3 Clock gate name: Flexcomm3.
kCLOCK_Flexcomm4 Clock gate name: Flexcomm4.
kCLOCK_Flexcomm5 Clock gate name: Flexcomm5.
kCLOCK_Flexcomm6 Clock gate name: Flexcomm6.
kCLOCK_Flexcomm7 Clock gate name: Flexcomm7.
kCLOCK_Flexcomm8 Clock gate name: Flexcomm8.
kCLOCK_Flexcomm9 Clock gate name: Flexcomm9.
kCLOCK_Flexcomm10 Clock gate name: Flexcomm10.
kCLOCK_Flexcomm11 Clock gate name: Flexcomm11.
kCLOCK_Flexcomm12 Clock gate name: Flexcomm12.
kCLOCK_Flexcomm13 Clock gate name: Flexcomm13.
kCLOCK_Flexcomm14 Clock gate name: Flexcomm14.
kCLOCK_Flexcomm15 Clock gate name: Flexcomm15.
kCLOCK_Flexcomm16 Clock gate name: Flexcomm16.
kCLOCK_Usart0 Clock gate name: Usart0.

kCLOCK_Usart1 Clock gate name: Usart1.
kCLOCK_Usart2 Clock gate name: Usart2.
kCLOCK_Usart3 Clock gate name: Usart3.
kCLOCK_Usart4 Clock gate name: Usart4.
kCLOCK_Usart5 Clock gate name: Usart5.
kCLOCK_Usart6 Clock gate name: Usart6.
kCLOCK_Usart7 Clock gate name: Usart7.
kCLOCK_Usart8 Clock gate name: Usart8.
kCLOCK_Usart9 Clock gate name: Usart9.
kCLOCK_Usart10 Clock gate name: Usart10.
kCLOCK_Usart11 Clock gate name: Usart11.
kCLOCK_Usart12 Clock gate name: Usart12.
kCLOCK_Usart13 Clock gate name: Usart13.
kCLOCK_I2s0 Clock gate name: I2s0.
kCLOCK_I2s1 Clock gate name: I2s1.
kCLOCK_I2s2 Clock gate name: I2s2.
kCLOCK_I2s3 Clock gate name: I2s3.
kCLOCK_I2s4 Clock gate name: I2s4.
kCLOCK_I2s5 Clock gate name: I2s5.
kCLOCK_I2s6 Clock gate name: I2s6.
kCLOCK_I2s7 Clock gate name: I2s7.
kCLOCK_I2s8 Clock gate name: I2s8.
kCLOCK_I2s9 Clock gate name: I2s9.
kCLOCK_I2s10 Clock gate name: I2s10.
kCLOCK_I2s11 Clock gate name: I2s11.
kCLOCK_I2s12 Clock gate name: I2s12.
kCLOCK_I2s13 Clock gate name: I2s13.
kCLOCK_I2c0 Clock gate name: I2c0.
kCLOCK_I2c1 Clock gate name: I2c1.
kCLOCK_I2c2 Clock gate name: I2c2.
kCLOCK_I2c3 Clock gate name: I2c3.
kCLOCK_I2c4 Clock gate name: I2c4.
kCLOCK_I2c5 Clock gate name: I2c5.
kCLOCK_I2c6 Clock gate name: I2c6.
kCLOCK_I2c7 Clock gate name: I2c7.
kCLOCK_I2c8 Clock gate name: I2c8.
kCLOCK_I2c9 Clock gate name: I2c9.
kCLOCK_I2c10 Clock gate name: I2c10.
kCLOCK_I2c11 Clock gate name: I2c11.
kCLOCK_I2c12 Clock gate name: I2c12.
kCLOCK_I2c13 Clock gate name: I2c13.
kCLOCK_I2c15 Clock gate name: I2c15.
kCLOCK_Spi0 Clock gate name: Spi0.
kCLOCK_Spi1 Clock gate name: Spi1.
kCLOCK_Spi2 Clock gate name: Spi2.

kCLOCK_Spi3 Clock gate name: Spi3.
kCLOCK_Spi4 Clock gate name: Spi4.
kCLOCK_Spi5 Clock gate name: Spi5.
kCLOCK_Spi6 Clock gate name: Spi6.
kCLOCK_Spi7 Clock gate name: Spi7.
kCLOCK_Spi8 Clock gate name: Spi8.
kCLOCK_Spi9 Clock gate name: Spi9.
kCLOCK_Spi10 Clock gate name: Spi10.
kCLOCK_Spi11 Clock gate name: Spi11.
kCLOCK_Spi12 Clock gate name: Spi12.
kCLOCK_Spi13 Clock gate name: Spi13.
kCLOCK_Spi14 Clock gate name: Spi14.
kCLOCK_Spi16 Clock gate name: Spi16.
kCLOCK_Dmic0 Clock gate name: Dmic0.
kCLOCK_OsEventTimer Clock gate name: OsEventTimer.
kCLOCK_Flexio Clock gate name: Flexio.
kCLOCK_HsGpio0 Clock gate name: HsGpio0.
kCLOCK_HsGpio1 Clock gate name: HsGpio1.
kCLOCK_HsGpio2 Clock gate name: HsGpio2.
kCLOCK_HsGpio3 Clock gate name: HsGpio3.
kCLOCK_HsGpio4 Clock gate name: HsGpio4.
kCLOCK_HsGpio5 Clock gate name: HsGpio5.
kCLOCK_HsGpio6 Clock gate name: HsGpio6.
kCLOCK_HsGpio7 Clock gate name: HsGpio7.
kCLOCK_Crc Clock gate name: Crc.
kCLOCK_Dmac0 Clock gate name: Dmac0.
kCLOCK_Dmac1 Clock gate name: Dmac1.
kCLOCK_Mu Clock gate name: Mu.
kCLOCK_Sema Clock gate name: Sema.
kCLOCK_Freqme Clock gate name: Freqme.
kCLOCK_Ct32b0 Clock gate name: Ct32b0.
kCLOCK_Ct32b1 Clock gate name: Ct32b1.
kCLOCK_Ct32b2 Clock gate name: Ct32b2.
kCLOCK_Ct32b3 Clock gate name: Ct32b3.
kCLOCK_Ct32b4 Clock gate name: Ct32b4.
kCLOCK_Rtc Clock gate name: Rtc.
kCLOCK_Mrt0 Clock gate name: Mrt0.
kCLOCK_Wwdt1 Clock gate name: Wwdt1.
kCLOCK_I3c0 Clock gate name: I3c0.
kCLOCK_I3c1 Clock gate name: I3c1.
kCLOCK_Pint Clock gate name: Pint.
kCLOCK_InputMux Clock gate name: InputMux.

4.4.2 enum clock_name_t

Enumerator

kCLOCK_CoreSysClk Core clock (aka HCLK)
kCLOCK_BusClk Bus clock (AHB/APB clock, aka HCLK)
kCLOCK_MclkClk MCLK, to MCLK pin.
kCLOCK_ClockOutClk CLOCKOUT.
kCLOCK_AdcClk ADC.
kCLOCK_Flexspi0Clk FlexSpi0.
kCLOCK_Flexspi1Clk FlexSpi1.
kCLOCK_SctClk SCT.
kCLOCK_Wdt0Clk Watchdog0.
kCLOCK_Wdt1Clk Watchdog1.
kCLOCK_SystickClk Systick.
kCLOCK_Sdio0Clk SDIO0.
kCLOCK_Sdio1Clk SDIO1.
kCLOCK_I3cClk I3C0 and I3C1.
kCLOCK_UsbClk USB0.
kCLOCK_DmicClk Digital Mic clock.
kCLOCK_DspCpuClk DSP clock.
kCLOCK_AcmpClk Acmp clock.
kCLOCK_Flexcomm0Clk Flexcomm0Clock.
kCLOCK_Flexcomm1Clk Flexcomm1Clock.
kCLOCK_Flexcomm2Clk Flexcomm2Clock.
kCLOCK_Flexcomm3Clk Flexcomm3Clock.
kCLOCK_Flexcomm4Clk Flexcomm4Clock.
kCLOCK_Flexcomm5Clk Flexcomm5Clock.
kCLOCK_Flexcomm6Clk Flexcomm6Clock.
kCLOCK_Flexcomm7Clk Flexcomm7Clock.
kCLOCK_Flexcomm8Clk Flexcomm8Clock.
kCLOCK_Flexcomm9Clk Flexcomm9Clock.
kCLOCK_Flexcomm10Clk Flexcomm10Clock.
kCLOCK_Flexcomm11Clk Flexcomm11Clock.
kCLOCK_Flexcomm12Clk Flexcomm12Clock.
kCLOCK_Flexcomm13Clk Flexcomm13Clock.
kCLOCK_Flexcomm14Clk Flexcomm14Clock.
kCLOCK_Flexcomm15Clk Flexcomm15Clock.
kCLOCK_Flexcomm16Clk Flexcomm16Clock.
kCLOCK_FlexioClk FlexIO.
kCLOCK_GpuClk GPU Core.
kCLOCK_DcPixelClk DCNano Pixel Clock.
kCLOCK_MipiDphyClk MIPI D-PHY Bit Clock.
kCLOCK_MipiDphyEscRxClk MIPI D-PHY RX Clock.
kCLOCK_MipiDphyEscTxClk MIPI D-PHY TX Clock.

4.4.3 enum clock_pfd_t

Enumerator

kCLOCK_Pfd0 PLL PFD0.
kCLOCK_Pfd1 PLL PFD1.
kCLOCK_Pfd2 PLL PFD2.
kCLOCK_Pfd3 PLL PFD3.

4.4.4 enum clock_attach_id_t

Enumerator

kFRO_DIV8_to_SYS_PLL Attach FRO_DIV8 to SYS_PLL.
kOSC_CLK_to_SYS_PLL Attach OSC_CLK to SYS_PLL.
kNONE_to_SYS_PLL Attach NONE to SYS_PLL.
kFRO_DIV8_to_AUDIO_PLL Attach FRO_DIV8 to AUDIO_PLL.
kOSC_CLK_to_AUDIO_PLL Attach OSC_CLK to AUDIO_PLL.
kNONE_to_AUDIO_PLL Attach NONE to AUDIO_PLL.
kLPOSC_to_MAIN_CLK Attach LPOSC to MAIN_CLK.
kFRO_DIV2_to_MAIN_CLK Attach Fro_DIV2 to MAIN_CLK.
kFRO_DIV4_to_MAIN_CLK Attach Fro_DIV4 to MAIN_CLK.
kFRO_DIV8_to_MAIN_CLK Attach Fro_DIV8 to MAIN_CLK.
kFRO_DIV16_to_MAIN_CLK Attach Fro_DIV16 to MAIN_CLK.
kOSC_CLK_to_MAIN_CLK Attach OSC_CLK to MAIN_CLK.
kFRO_DIV1_to_MAIN_CLK Attach FRO_DIV1 to MAIN_CLK.
kMAIN_PLL_to_MAIN_CLK Attach MAIN_PLL to MAIN_CLK.
kOSC32K_to_MAIN_CLK Attach OSC32K to MAIN_CLK.
kFRO_DIV1_to_DSP_MAIN_CLK Attach Fro_DIV1 to DSP_MAIN_CLK.
kOSC_CLK_to_DSP_MAIN_CLK Attach OSC_CLK to DSP_MAIN_CLK.
kLPOSC_to_DSP_MAIN_CLK Attach LPOSC to DSP_MAIN_CLK.
kMAIN_PLL_to_DSP_MAIN_CLK Attach MAIN_PLL to DSP_MAIN_CLK.
kDSP_PLL_to_DSP_MAIN_CLK Attach DSP_PLL to DSP_MAIN_CLK.
kOSC32K_to_DSP_MAIN_CLK Attach OSC32K to DSP_MAIN_CLK.
kLPOSC_to_UTICK_CLK Attach LPOSC to UTICK_CLK.
kNONE_to_UTICK_CLK Attach NONE to UTICK_CLK.
kLPOSC_to_WDT0_CLK Attach LPOSC to WDT0_CLK.
kNONE_to_WDT0_CLK Attach NONE to WDT0_CLK.
kLPOSC_to_WDT1_CLK Attach LPOSC to WDT1_CLK.
kNONE_to_WDT1_CLK Attach NONE to WDT1_CLK.
kOSC32K_to_32KHZWAKE_CLK Attach OSC32K to 32KHZWAKE_CLK.
kLPOSC_DIV32_to_32KHZWAKE_CLK Attach LPOSC_DIV32 to 32KHZWAKE_CLK.
kNONE_to_32KHZWAKE_CLK Attach NONE to 32KHZWAKE_CLK.
kMAIN_CLK_DIV_to_SYSTICK_CLK Attach MAIN_CLK_DIV to SYSTICK_CLK.

kLPOSC_to_SYSTICK_CLK Attach LPOSC to SYSTICK_CLK.
kOSC32K_to_SYSTICK_CLK Attach OSC32K to SYSTICK_CLK.
kNONE_to_SYSTICK_CLK Attach NONE to SYSTICK_CLK.
kMAIN_CLK_to_SDIO0_CLK Attach MAIN_CLK to SDIO0_CLK.
kMAIN_PLL_to_SDIO0_CLK Attach MAIN_PLL to SDIO0_CLK.
kAUX0_PLL_to_SDIO0_CLK Attach AUX0_PLL to SDIO0_CLK.
kFRO_DIV2_to_SDIO0_CLK Attach FRO_DIV2 to SDIO0_CLK.
kAUX1_PLL_to_SDIO0_CLK Attach AUX1_PLL to SDIO0_CLK.
kNONE_to_SDIO0_CLK Attach NONE to SDIO0_CLK.
kMAIN_CLK_to_SDIO1_CLK Attach MAIN_CLK to SDIO1_CLK.
kMAIN_PLL_to_SDIO1_CLK Attach MAIN_PLL to SDIO1_CLK.
kAUX0_PLL_to_SDIO1_CLK Attach AUX0_PLL to SDIO1_CLK.
kFRO_DIV2_to_SDIO1_CLK Attach FRO_DIV2 to SDIO1_CLK.
kAUX1_PLL_to_SDIO1_CLK Attach AUX1_PLL to SDIO1_CLK.
kNONE_to_SDIO1_CLK Attach NONE to SDIO1_CLK.
kMAIN_CLK_to_CTIMER0 Attach MAIN_CLK to CTIMER0.
kFRO_DIV1_to_CTIMER0 Attach FRO_DIV1 to CTIMER0.
kAUDIO_PLL_to_CTIMER0 Attach AUDIO_PLL to CTIMER0.
kMASTER_CLK_to_CTIMER0 Attach MASTER_CLK to CTIMER0.
k32K_WAKE_CLK_to_CTIMER0 Attach 32K_WAKE_CLK to CTIMER0.
kNONE_to_CTIMER0 Attach NONE to CTIMER0.
kMAIN_CLK_to_CTIMER1 Attach MAIN_CLK to CTIMER1.
kFRO_DIV1_to_CTIMER1 Attach FRO_DIV1 to CTIMER1.
kAUDIO_PLL_to_CTIMER1 Attach AUDIO_PLL to CTIMER1.
kMASTER_CLK_to_CTIMER1 Attach MASTER_CLK to CTIMER1.
k32K_WAKE_CLK_to_CTIMER1 Attach 32K_WAKE_CLK to CTIMER1.
kNONE_to_CTIMER1 Attach NONE to CTIMER1.
kMAIN_CLK_to_CTIMER2 Attach MAIN_CLK to CTIMER2.
kFRO_DIV1_to_CTIMER2 Attach FRO_DIV1 to CTIMER2.
kAUDIO_PLL_to_CTIMER2 Attach AUDIO_PLL to CTIMER2.
kMASTER_CLK_to_CTIMER2 Attach MASTER_CLK to CTIMER2.
k32K_WAKE_CLK_to_CTIMER2 Attach 32K_WAKE_CLK to CTIMER2.
kNONE_to_CTIMER2 Attach NONE to CTIMER2.
kMAIN_CLK_to_CTIMER3 Attach MAIN_CLK to CTIMER3.
kFRO_DIV1_to_CTIMER3 Attach FRO_DIV1 to CTIMER3.
kAUDIO_PLL_to_CTIMER3 Attach AUDIO_PLL to CTIMER3.
kMASTER_CLK_to_CTIMER3 Attach MASTER_CLK to CTIMER3.
k32K_WAKE_CLK_to_CTIMER3 Attach 32K_WAKE_CLK to CTIMER3.
kNONE_to_CTIMER3 Attach NONE to CTIMER3.
kMAIN_CLK_to_CTIMER4 Attach MAIN_CLK to CTIMER4.
kFRO_DIV1_to_CTIMER4 Attach FRO_DIV1 to CTIMER4.
kAUDIO_PLL_to_CTIMER4 Attach AUDIO_PLL to CTIMER4.
kMASTER_CLK_to_CTIMER4 Attach MASTER_CLK to CTIMER4.
k32K_WAKE_CLK_to_CTIMER4 Attach 32K_WAKE_CLK to CTIMER4.
kNONE_to_CTIMER4 Attach NONE to CTIMER4.

kMAIN_CLK_to_FLEXSPI0_CLK Attach MAIN_CLK to FLEXSPI0_CLK.
kMAIN_PLL_to_FLEXSPI0_CLK Attach MAIN_PLL to FLEXSPI0_CLK.
kAUX0_PLL_to_FLEXSPI0_CLK Attach AUX0_PLL to FLEXSPI0_CLK.
kFRO_DIV1_to_FLEXSPI0_CLK Attach FRO_DIV1 to FLEXSPI0_CLK.
kAUX1_PLL_to_FLEXSPI0_CLK Attach AUX1_PLL to FLEXSPI0_CLK.
kFRO_DIV4_to_FLEXSPI0_CLK Attach FRO_DIV4 to FLEXSPI0_CLK.
kFRO_DIV8_to_FLEXSPI0_CLK Attach FRO_DIV8 to FLEXSPI0_CLK.
kNONE_to_FLEXSPI0_CLK Attach NONE to FLEXSPI0_CLK.
kMAIN_CLK_to_FLEXSPI1_CLK Attach MAIN_CLK to FLEXSPI1_CLK.
kMAIN_PLL_to_FLEXSPI1_CLK Attach MAIN_PLL to FLEXSPI1_CLK.
kAUX0_PLL_to_FLEXSPI1_CLK Attach AUX0_PLL to FLEXSPI1_CLK.
kFRO_DIV1_to_FLEXSPI1_CLK Attach FRO_DIV1 to FLEXSPI1_CLK.
kAUX1_PLL_to_FLEXSPI1_CLK Attach AUX1_PLL to FLEXSPI1_CLK.
kNONE_to_FLEXSPI1_CLK Attach NONE to FLEXSPI1_CLK.
kOSC_CLK_to_USB_CLK Attach OSC_CLK to USB_CLK.
kMAIN_CLK_to_USB_CLK Attach MAIN_CLK to USB_CLK.
kAUX0_PLL_to_USB_CLK Attach AUX0_PLL to USB_CLK.
kNONE_to_USB_CLK Attach NONE to USB_CLK.
kMAIN_CLK_to_SCT_CLK Attach MAIN_CLK to SCT_CLK.
kMAIN_PLL_to_SCT_CLK Attach MAIN_PLL to SCT_CLK.
kAUX0_PLL_to_SCT_CLK Attach AUX0_PLL to SCT_CLK.
kFRO_DIV1_to_SCT_CLK Attach FRO_DIV1 to SCT_CLK.
kAUX1_PLL_to_SCT_CLK Attach AUX1_PLL to SCT_CLK.
kAUDIO_PLL_to_SCT_CLK Attach AUDIO_PLL to SCT_CLK.
kNONE_to_SCT_CLK Attach NONE to SCT_CLK.
kLPOSC_to_OSTIMER_CLK Attach LPOSC to OSTIMER_CLK.
kOSC32K_to_OSTIMER_CLK Attach OSC32K to OSTIMER_CLK.
kHCLK_to_OSTIMER_CLK Attach HCLK to OSTIMER_CLK.
kNONE_to_OSTIMER_CLK Attach NONE to OSTIMER_CLK.
kFRO_DIV8_to_MCLK_CLK Attach FRO_DIV8 to MCLK_CLK.
kAUDIO_PLL_to_MCLK_CLK Attach AUDIO_PLL to MCLK_CLK.
kNONE_to_MCLK_CLK Attach NONE to MCLK_CLK.
kFRO_DIV4_to_DMIC Attach FRO_DIV4 to DMIC.
kAUDIO_PLL_to_DMIC Attach AUDIO_PLL to DMIC.
kMASTER_CLK_to_DMIC Attach MASTER_CLK to DMIC.
kLPOSC_to_DMIC Attach LPOSC to DMIC.
k32K_WAKE_CLK_to_DMIC Attach 32K_WAKE_CLK to DMIC.
kNONE_to_DMIC Attach NONE to DMIC.
kFRO_DIV4_to_FLEXCOMM0 Attach FRO_DIV4 to FLEXCOMM0.
kAUDIO_PLL_to_FLEXCOMM0 Attach AUDIO_PLL to FLEXCOMM0.
kMASTER_CLK_to_FLEXCOMM0 Attach MASTER_CLK to FLEXCOMM0.
kFRG_to_FLEXCOMM0 Attach FRG to FLEXCOMM0.
kNONE_to_FLEXCOMM0 Attach NONE to FLEXCOMM0.
kFRO_DIV4_to_FLEXCOMM1 Attach FRO_DIV4 to FLEXCOMM1.
kAUDIO_PLL_to_FLEXCOMM1 Attach AUDIO_PLL to FLEXCOMM1.

kMASTER_CLK_to_FLEXCOMM1 Attach MASTER_CLK to FLEXCOMM1.
kFRG_to_FLEXCOMM1 Attach FRG to FLEXCOMM1.
kNONE_to_FLEXCOMM1 Attach NONE to FLEXCOMM1.
kFRO_DIV4_to_FLEXCOMM2 Attach FRO_DIV4 to FLEXCOMM2.
kAUDIO_PLL_to_FLEXCOMM2 Attach AUDIO_PLL to FLEXCOMM2.
kMASTER_CLK_to_FLEXCOMM2 Attach MASTER_CLK to FLEXCOMM2.
kFRG_to_FLEXCOMM2 Attach FRG to FLEXCOMM2.
kNONE_to_FLEXCOMM2 Attach NONE to FLEXCOMM2.
kFRO_DIV4_to_FLEXCOMM3 Attach FRO_DIV4 to FLEXCOMM3.
kAUDIO_PLL_to_FLEXCOMM3 Attach AUDIO_PLL to FLEXCOMM3.
kMASTER_CLK_to_FLEXCOMM3 Attach MASTER_CLK to FLEXCOMM3.
kFRG_to_FLEXCOMM3 Attach FRG to FLEXCOMM3.
kNONE_to_FLEXCOMM3 Attach NONE to FLEXCOMM3.
kFRO_DIV4_to_FLEXCOMM4 Attach FRO_DIV4 to FLEXCOMM4.
kAUDIO_PLL_to_FLEXCOMM4 Attach AUDIO_PLL to FLEXCOMM4.
kMASTER_CLK_to_FLEXCOMM4 Attach MASTER_CLK to FLEXCOMM4.
kFRG_to_FLEXCOMM4 Attach FRG to FLEXCOMM4.
kNONE_to_FLEXCOMM4 Attach NONE to FLEXCOMM4.
kFRO_DIV4_to_FLEXCOMM5 Attach FRO_DIV4 to FLEXCOMM5.
kAUDIO_PLL_to_FLEXCOMM5 Attach AUDIO_PLL to FLEXCOMM5.
kMASTER_CLK_to_FLEXCOMM5 Attach MASTER_CLK to FLEXCOMM5.
kFRG_to_FLEXCOMM5 Attach FRG to FLEXCOMM5.
kNONE_to_FLEXCOMM5 Attach NONE to FLEXCOMM5.
kFRO_DIV4_to_FLEXCOMM6 Attach FRO_DIV4 to FLEXCOMM6.
kAUDIO_PLL_to_FLEXCOMM6 Attach AUDIO_PLL to FLEXCOMM6.
kMASTER_CLK_to_FLEXCOMM6 Attach MASTER_CLK to FLEXCOMM6.
kFRG_to_FLEXCOMM6 Attach FRG to FLEXCOMM6.
kNONE_to_FLEXCOMM6 Attach NONE to FLEXCOMM6.
kFRO_DIV4_to_FLEXCOMM7 Attach FRO_DIV4 to FLEXCOMM7.
kAUDIO_PLL_to_FLEXCOMM7 Attach AUDIO_PLL to FLEXCOMM7.
kMASTER_CLK_to_FLEXCOMM7 Attach MASTER_CLK to FLEXCOMM7.
kFRG_to_FLEXCOMM7 Attach FRG to FLEXCOMM7.
kNONE_to_FLEXCOMM7 Attach NONE to FLEXCOMM7.
kFRO_DIV4_to_FLEXCOMM8 Attach FRO_DIV4 to FLEXCOMM8.
kAUDIO_PLL_to_FLEXCOMM8 Attach AUDIO_PLL to FLEXCOMM8.
kMASTER_CLK_to_FLEXCOMM8 Attach MASTER_CLK to FLEXCOMM8.
kFRG_to_FLEXCOMM8 Attach FRG to FLEXCOMM8.
kNONE_to_FLEXCOMM8 Attach NONE to FLEXCOMM8.
kFRO_DIV4_to_FLEXCOMM9 Attach FRO_DIV4 to FLEXCOMM9.
kAUDIO_PLL_to_FLEXCOMM9 Attach AUDIO_PLL to FLEXCOMM9.
kMASTER_CLK_to_FLEXCOMM9 Attach MASTER_CLK to FLEXCOMM9.
kFRG_to_FLEXCOMM9 Attach FRG to FLEXCOMM9.
kNONE_to_FLEXCOMM9 Attach NONE to FLEXCOMM9.
kFRO_DIV4_to_FLEXCOMM10 Attach FRO_DIV4 to FLEXCOMM10.
kAUDIO_PLL_to_FLEXCOMM10 Attach AUDIO_PLL to FLEXCOMM10.

kMASTER_CLK_to_FLEXCOMM10 Attach MASTER_CLK to FLEXCOMM10.
kFRG_to_FLEXCOMM10 Attach FRG to FLEXCOMM10.
kNONE_to_FLEXCOMM10 Attach NONE to FLEXCOMM10.
kFRO_DIV4_to_FLEXCOMM11 Attach FRO_DIV4 to FLEXCOMM11.
kAUDIO_PLL_to_FLEXCOMM11 Attach AUDIO_PLL to FLEXCOMM11.
kMASTER_CLK_to_FLEXCOMM11 Attach MASTER_CLK to FLEXCOMM11.
kFRG_to_FLEXCOMM11 Attach FRG to FLEXCOMM11.
kNONE_to_FLEXCOMM11 Attach NONE to FLEXCOMM11.
kFRO_DIV4_to_FLEXCOMM12 Attach FRO_DIV4 to FLEXCOMM12.
kAUDIO_PLL_to_FLEXCOMM12 Attach AUDIO_PLL to FLEXCOMM12.
kMASTER_CLK_to_FLEXCOMM12 Attach MASTER_CLK to FLEXCOMM12.
kFRG_to_FLEXCOMM12 Attach FRG to FLEXCOMM12.
kNONE_to_FLEXCOMM12 Attach NONE to FLEXCOMM12.
kFRO_DIV4_to_FLEXCOMM13 Attach FRO_DIV4 to FLEXCOMM13.
kAUDIO_PLL_to_FLEXCOMM13 Attach AUDIO_PLL to FLEXCOMM13.
kMASTER_CLK_to_FLEXCOMM13 Attach MASTER_CLK to FLEXCOMM13.
kFRG_to_FLEXCOMM13 Attach FRG to FLEXCOMM13.
kNONE_to_FLEXCOMM13 Attach NONE to FLEXCOMM13.
kFRO_DIV4_to_FLEXCOMM14 Attach FRO_DIV4 to FLEXCOMM14.
kAUDIO_PLL_to_FLEXCOMM14 Attach AUDIO_PLL to FLEXCOMM14.
kMASTER_CLK_to_FLEXCOMM14 Attach MASTER_CLK to FLEXCOMM14.
kFRG_to_FLEXCOMM14 Attach FRG to FLEXCOMM14.
kNONE_to_FLEXCOMM14 Attach NONE to FLEXCOMM14.
kFRO_DIV4_to_FLEXCOMM15 Attach FRO_DIV4 to FLEXCOMM15.
kAUDIO_PLL_to_FLEXCOMM15 Attach AUDIO_PLL to FLEXCOMM15.
kMASTER_CLK_to_FLEXCOMM15 Attach MASTER_CLK to FLEXCOMM15.
kFRG_to_FLEXCOMM15 Attach FRG to FLEXCOMM15.
kNONE_to_FLEXCOMM15 Attach NONE to FLEXCOMM15.
kFRO_DIV4_to_FLEXCOMM16 Attach FRO_DIV4 to FLEXCOMM16.
kAUDIO_PLL_to_FLEXCOMM16 Attach AUDIO_PLL to FLEXCOMM16.
kMASTER_CLK_to_FLEXCOMM16 Attach MASTER_CLK to FLEXCOMM16.
kFRG_to_FLEXCOMM16 Attach FRG to FLEXCOMM16.
kNONE_to_FLEXCOMM16 Attach NONE to FLEXCOMM16.
kFRO_DIV2_to_FLEXIO Attach FRO_DIV2 to FLEXIO.
kAUDIO_PLL_to_FLEXIO Attach AUDIO_PLL to FLEXIO.
kMASTER_CLK_to_FLEXIO Attach MASTER_CLK to FLEXIO.
kFRG_to_FLEXIO Attach FRG to FLEXIO.
kNONE_to_FLEXIO Attach NONE to FLEXIO.
kMAIN_CLK_to_I3C_CLK Attach MAIN_CLK to I3C_CLK.
kFRO_DIV8_to_I3C_CLK Attach FRO_DIV8 to I3C_CLK.
kNONE_to_I3C_CLK Attach NONE to I3C_CLK.
kI3C_CLK_to_I3C_TC_CLK Attach I3C_CLK to I3C_TC_CLK.
kLPOSC_to_I3C_TC_CLK Attach LPOSC to I3C_TC_CLK.
kNONE_to_I3C_TC_CLK Attach NONE to I3C_TC_CLK.
kMAIN_CLK_to_ACMP_CLK Attach MAIN_CLK to ACMP_CLK.

kFRO_DIV4_to_ACMP_CLK Attach FRO_DIV4 to ACMP_CLK.
kAUX0_PLL_to_ACMP_CLK Attach AUX0_PLL to ACMP_CLK.
kAUX1_PLL_to_ACMP_CLK Attach AUX1_PLL to ACMP_CLK.
kNONE_to_ACMP_CLK Attach NONE to ACMP_CLK.
kOSC_CLK_to_ADC_CLK Attach OSC_CLK to ADC_CLK.
kLPOSC_to_ADC_CLK Attach LPOSC to ADC_CLK.
kFRO_DIV4_to_ADC_CLK Attach FRO_DIV4 to ADC_CLK.
kMAIN_PLL_to_ADC_CLK Attach MAIN_PLL to ADC_CLK.
kAUX0_PLL_to_ADC_CLK Attach AUX0_PLL to ADC_CLK.
kAUX1_PLL_to_ADC_CLK Attach AUX1_PLL to ADC_CLK.
kOSC_CLK_to_CLKOUT Attach OSC_CLK to CLKOUT.
kLPOSC_to_CLKOUT Attach LPOSC to CLKOUT.
kFRO_DIV2_to_CLKOUT Attach FRO_DIV2 to CLKOUT.
kMAIN_CLK_to_CLKOUT Attach MAIN_CLK to CLKOUT.
kDSP_MAIN_to_CLKOUT Attach DSP_MAIN to CLKOUT.
kMAIN_PLL_to_CLKOUT Attach MAIN_PLL to CLKOUT.
kAUX0_PLL_to_CLKOUT Attach AUX0_PLL to CLKOUT.
kDSP_PLL_to_CLKOUT Attach DSP_PLL to CLKOUT.
kAUX1_PLL_to_CLKOUT Attach AUX1_PLL to CLKOUT.
kAUDIO_PLL_to_CLKOUT Attach AUDIO_PLL to CLKOUT.
kOSC32K_to_CLKOUT Attach OSC32K to CLKOUT.
kNONE_to_CLKOUT Attach NONE to CLKOUT.
kMAIN_CLK_to_GPU_CLK Attach MAIN_CLK to GPU_CLK.
kFRO_DIV1_to_GPU_CLK Attach FRO_DIV1 to GPU_CLK.
kMAIN_PLL_to_GPU_CLK Attach MAIN_PLL to GPU_CLK.
kAUX0_PLL_to_GPU_CLK Attach AUX0_PLL to GPU_CLK.
kAUX1_PLL_to_GPU_CLK Attach AUX1_PLL to GPU_CLK.
kNONE_to_GPU_CLK Attach NONE to GPU_CLK.
kFRO_DIV1_to_MIPI_DPHY_CLK Attach FRO_DIV1 to MIPI_DPHY_CLK.
kMAIN_PLL_to_MIPI_DPHY_CLK Attach MAIN_PLL to MIPI_DPHY_CLK.
kAUX0_PLL_to_MIPI_DPHY_CLK Attach AUX0_PLL to MIPI_DPHY_CLK.
kAUX1_PLL_to_MIPI_DPHY_CLK Attach AUX1_PLL to MIPI_DPHY_CLK.
kNONE_to_MIPI_DPHY_CLK Attach NONE to MIPI_DPHY_CLK.
kFRO_DIV1_to_MIPI_DPHYESC_CLK Attach FRO_DIV1 to MIPI_DPHYESC_CLK.
kFRO_DIV16_to_MIPI_DPHYESC_CLK Attach FRO_DIV16 to MIPI_DPHYESC_CLK.
kAUX0_PLL_to_MIPI_DPHYESC_CLK Attach AUX0_PLL to MIPI_DPHYESC_CLK.
kAUX1_PLL_to_MIPI_DPHYESC_CLK Attach AUX1_PLL to MIPI_DPHYESC_CLK.
kMIPI_DPHY_CLK_to_DCPIXEL_CLK Attach MIPI_DPHY_CLK to DCPIXEL_CLK.
kMAIN_CLK_to_DCPIXEL_CLK Attach MAIN_CLK to DCPIXEL_CLK.
kFRO_DIV1_to_DCPIXEL_CLK Attach FRO_DIV1 to DCPIXEL_CLK.
kMAIN_PLL_to_DCPIXEL_CLK Attach MAIN_PLL to DCPIXEL_CLK.
kAUX0_PLL_to_DCPIXEL_CLK Attach AUX0_PLL to DCPIXEL_CLK.
kAUX1_PLL_to_DCPIXEL_CLK Attach AUX1_PLL to DCPIXEL_CLK.
kNONE_to_DCPIXEL_CLK Attach NONE to DCPIXEL_CLK.

4.4.5 enum clock_div_name_t

Enumerator

kCLOCK_DivAudioPllClk Audio Pll Clk Divider.
kCLOCK_DivMainPllClk Main Pll Clk Divider.
kCLOCK_DivDspPllClk Dsp Pll Clk Divider.
kCLOCK_DivAux0PllClk Aux0 Pll Clk Divider.
kCLOCK_DivAux1PllClk Aux1 Pll Clk Divider.
kCLOCK_DivPfc0Clk Pfc0 Clk Divider.
kCLOCK_DivPfc1Clk Pfc1 Clk Divider.
kCLOCK_DivSysCpuAhbClk Sys Cpu Ahb Clk Divider.
kCLOCK_Div32KhzWakeClk Khz Wake Clk Divider.
kCLOCK_DivSystickClk Systick Clk Divider.
kCLOCK_DivSdio0Clk Sdio0 Clk Divider.
kCLOCK_DivSdio1Clk Sdio1 Clk Divider.
kCLOCK_DivFlexspi0Clk Flexspi0 Clk Divider.
kCLOCK_DivFlexspi1Clk Flexspi1 Clk Divider.
kCLOCK_DivUsbHsFclk Usb Hs Fclk Divider.
kCLOCK_DivSctClk Sct Clk Divider.
kCLOCK_DivMclkClk Mclk Clk Divider.
kCLOCK_DivDmicClk Dmic Clk Divider.
kCLOCK_DivPLLFRGClk P L L F R G Clk Divider.
kCLOCK_DivFlexioClk Flexio Clk Divider.
kCLOCK_DivI3cClk I3c Clk Divider.
kCLOCK_DivI3cTcClk I3c Tc Clk Divider.
kCLOCK_DivI3cSlowClk I3c Slow Clk Divider.
kCLOCK_DivDspCpuClk Dsp Cpu Clk Divider.
kCLOCK_DivAcmpClk Acmp Clk Divider.
kCLOCK_DivAdcClk Adc Clk Divider.
kCLOCK_DivLowFreqClk Low Freq Clk Divider.
kCLOCK_DivClockOut Clock Out Divider.
kCLOCK_DivGpuClk Gpu Clk Divider.
kCLOCK_DivDcPixelClk Dc Pixel Clk Divider.
kCLOCK_DivDphyClk Dphy Clk Divider.
kCLOCK_DivDphyEscRxClk Dphy Esc Rx Clk Divider.
kCLOCK_DivDphyEscTxClk Dphy Esc Tx Clk Divider.

4.4.6 enum sys_pll_src_t

Enumerator

kCLOCK_SysPllFroDiv8Clk FRO_DIV8 clock.
kCLOCK_SysPllXtalIn OSC clock.

kCLOCK_SysPllNone Gated to reduce power.

4.4.7 enum sys_pll_mult_t

Enumerator

kCLOCK_SysPllMult16 Divide by 16.
kCLOCK_SysPllMult17 Divide by 17.
kCLOCK_SysPllMult18 Divide by 18.
kCLOCK_SysPllMult19 Divide by 19.
kCLOCK_SysPllMult20 Divide by 20.
kCLOCK_SysPllMult21 Divide by 21.
kCLOCK_SysPllMult22 Divide by 22.

4.4.8 enum audio_pll_src_t

Enumerator

kCLOCK_AudioPllFroDiv8Clk FRO_DIV8 clock.
kCLOCK_AudioPllXtalIn OSC clock.
kCLOCK_AudioPllNone Gated to reduce power.

4.4.9 enum audio_pll_mult_t

Enumerator

kCLOCK_AudioPllMult16 Divide by 16.
kCLOCK_AudioPllMult17 Divide by 17.
kCLOCK_AudioPllMult18 Divide by 18.
kCLOCK_AudioPllMult19 Divide by 19.
kCLOCK_AudioPllMult20 Divide by 20.
kCLOCK_AudioPllMult21 Divide by 21.
kCLOCK_AudioPllMult22 Divide by 22.

4.4.10 enum clock_fro_output_en_t

Enumerator

kCLOCK_FroDiv1OutEn Enable Fro Div1 output.
kCLOCK_FroDiv2OutEn Enable Fro Div2 output.

kCLOCK_FroDiv4OutEn Enable Fro Div4 output.
kCLOCK_FroDiv8OutEn Enable Fro Div8 output.
kCLOCK_FroDiv16OutEn Enable Fro Div16 output.

4.4.11 enum clock_fro_freq_t

Enumerator

kCLOCK_Fro192M 192MHz FRO clock.
kCLOCK_Fro96M 96MHz FRO clock.

4.5 Function Documentation

4.5.1 void CLOCK_AttachClk (clock_attach_id_t *connection*)

Parameters

connection	: Clock to be configured.

Returns

Nothing

4.5.2 void CLOCK_SetClkDiv (clock_div_name_t *div_name,* uint32_t *divider*)

Parameters

div_name	: Clock divider name
divider	: Value to be divided. Divided clock frequency = Undivided clock frequency / divider.

Returns

Nothing

4.5.3 uint32_t CLOCK_GetFreq (clock_name_t *clockName*)

Returns

Frequency of selected clock

4.5.4 uint32_t CLOCK_GetFRGClock (uint32_t *id*)

Returns

Input Frequency for FRG

4.5.5 void CLOCK_SetFRGClock (const clock_frg_clk_config_t ∗ *config*)

Parameters

config	: Configuration to set to FRGn clock.

4.5.6 uint32_t CLOCK_GetSysPllFreq (void)

Returns

Frequency of SYSPLL

4.5.7 uint32_t CLOCK_GetSysPfdFreq (clock_pfd_t *pfd*)

Parameters

pfd	: pfd name to get frequency.

Returns

Frequency of SYSPLL PFD.

4.5.8 uint32_t CLOCK_GetAudioPllFreq (void)

Returns

Frequency of AUDIO PLL

4.5.9 uint32_t CLOCK_GetAudioPfdFreq (clock_pfd_t *pfd*)

Parameters

pfd	: pfd name to get frequency.

Returns

Frequency of AUDIO PLL PFD.

4.5.10 uint32_t CLOCK_GetMainClkFreq (void)

Returns

Frequency of main clk

4.5.11 uint32_t CLOCK_GetDspMainClkFreq (void)

Returns

Frequency of DSP main clk

4.5.12 uint32_t CLOCK_GetAcmpClkFreq (void)

Returns

Frequency of ACMP clk

4.5.13 uint32_t CLOCK_GetDmicClkFreq (void)

Returns

Frequency of DMIC clk

4.5.14 uint32_t CLOCK_GetUsbClkFreq (void)

Returns

Frequency of USB clk

4.5.15 uint32_t CLOCK_GetSdioClkFreq (uint32_t *id*)

Parameters

id	: SDIO index to get frequency.

Returns

Frequency of SDIO clk

4.5.16 uint32_t CLOCK_GetI3cClkFreq (void)

Returns

Frequency of I3C clk

4.5.17 uint32_t CLOCK_GetSystickClkFreq (void)

Returns

Frequency of systick clk

4.5.18 uint32_t CLOCK_GetWdtClkFreq (uint32_t *id*)

Parameters

id	: WDT index to get frequency.

Returns

Frequency of WDT clk

4.5.19 uint32_t CLOCK_GetMclkClkFreq (void)

Returns

Frequency of mclk output clk

4.5.20 uint32_t CLOCK_GetSctClkFreq (void)

Returns

Frequency of sct clk

4.5.21 void CLOCK_EnableSysOscClk (bool *enable,* bool *enableLowPower,* uint32_t *delay_us*)

Parameters

enable	: true to enable system osc clock, false to bypass system osc.
enableLowPower	: true to enable low power mode, false to enable high gain mode.
delay_us	: Delay time after OSC power up.

4.5.22 void CLOCK_EnableFroClk (uint32_t *divOutEnable*)

Parameters

divOutEnable	: Or'ed value of clock_fro_output_en_t to enable certain clock freq output.

4.5.23 void CLOCK_EnableFroClkRange (clock_fro_freq_t *froFreq,* uint32_t *divOutEnable*)

Parameters

froFreq	: target fro frequency.
divOutEnable	: Or'ed value of clock_fro_output_en_t to enable certain clock freq output.

4.5.24 static uint32_t CLOCK_GetXtalInClkFreq (void) `[inline], [static]`

Returns

Frequency of sys osc Clock. Or CLK_IN pin frequency.

4.5.25 static uint32_t CLOCK_GetMclkInClkFreq (void) [inline], [static]

Returns

Frequency of MCLK input Clock.

4.5.26 static uint32_t CLOCK_GetLpOscFreq (void) [inline], [static]

Returns

Frequency of LPOSC

4.5.27 static uint32_t CLOCK_GetOsc32KFreq (void) [inline], [static]

Returns

Frequency of 32kHz osc

4.5.28 static void CLOCK_EnableOsc32K (bool *enable*) [inline], [static]

Parameters

enable	: true to enable 32k osc clock, false to disable clock

4.5.29 static uint32_t CLOCK_GetWakeClk32KFreq (void) [inline], [static]

Returns

Frequency of 32kHz wake clk

4.5.30 static void CLOCK_SetXtalFreq (uint32_t *freq*) [inline], [static]

Parameters

freq	: The XTAL input clock frequency in Hz.

4.5.31 static void CLOCK_SetClkinFreq (uint32_t *freq*) [inline], [static]

Parameters

freq	: The CLK_IN pin input clock frequency in Hz.

4.5.32 static void CLOCK_SetMclkFreq (uint32_t *freq*) [inline], [static]

Parameters

freq	: The MCLK input clock frequency in Hz.

4.5.33 uint32_t CLOCK_GetFlexcommClkFreq (uint32_t *id*)

Parameters

id	: flexcomm index to get frequency.

Returns

Frequency of Flexcomm functional Clock

4.5.34 uint32_t CLOCK_GetFlexioClkFreq (void)

Returns

Frequency of Flexcomm functional Clock

4.5.35 uint32_t CLOCK_GetCtimerClkFreq (uint32_t *id*)

Parameters

id	: ctimer index to get frequency.

Returns

Frequency of Ctimer Clock

4.5.36 uint32_t CLOCK_GetClockOutClkFreq (void)

Returns

Frequency of ClockOut

4.5.37 uint32_t CLOCK_GetAdcClkFreq (void)

Returns

Frequency of Adc Clock.

4.5.38 uint32_t CLOCK_GetFlexspiClkFreq (uint32_t *id*)

Parameters

id	: flexspi index to get frequency.

Returns

Frequency of Flexspi.

4.5.39 uint32_t CLOCK_GetGpuClkFreq (void)

Returns

Frequency of GPU functional Clock

4.5.40 uint32_t CLOCK_GetDcPixelClkFreq (void)

Returns

Frequency of DCNano pixel functional Clock

4.5.41 uint32_t CLOCK_GetMipiDphyClkFreq (void)

Returns

Frequency of MIPI DPHY functional Clock

4.5.42 uint32_t CLOCK_GetMipiDphyEscRxClkFreq (void)

Returns

Frequency of MIPI DPHY Esc RX functional Clock

4.5.43 uint32_t CLOCK_GetMipiDphyEscTxClkFreq (void)

Returns

Frequency of MIPI DPHY Esc Tx functional Clock

4.5.44 void CLOCK_InitSysPll (const clock_sys_pll_config_t * *config*)

Parameters

config	: Configuration to set to PLL.

4.5.45 static void CLOCK_DeinitSysPll (void) `[inline], [static]`

param none.

4.5.46 void CLOCK_InitSysPfd (clock_pfd_t *pfd,* uint8_t *divider*)

Parameters

pfd	: Which PFD clock to enable.
divider	: The PFD divider value.

Note

It is recommended that PFD settings are kept between 12-35.

4.5.47 static void CLOCK_DeinitSysPfd (clock_pfd_t *pfd*) [inline], [static]

param pfd : Which PFD clock to disable.

4.5.48 void CLOCK_InitAudioPll (const clock_audio_pll_config_t ∗ *config*)

Parameters

config	: Configuration to set to PLL.

4.5.49 static void CLOCK_DeinitAudioPll (void) [inline], [static]

param none.

4.5.50 void CLOCK_InitAudioPfd (clock_pfd_t *pfd,* uint8_t *divider*)

Parameters

pfd	: Which PFD clock to enable.
divider	: The PFD divider value.

Note

It is recommended that PFD settings are kept between 12-35.

4.5.51 static void CLOCK_DeinitAudioPfd (uint32_t *pfd*) [inline], [static]

param pfd : Which PFD clock to disable.

4.5.52 void CLOCK_EnableFroTuning (bool *enable*)

On enable, the function will wait until FRO is close to the target frequency.

4.5.53 void CLOCK_EnableUsbHs0DeviceClock (clock_attach_id_t *src,* uint8_t *divider*)

This function enables USB HS device clock.

4.5.54 void CLOCK_DisableUsbHs0DeviceClock (void)

This function disables USB HS device clock.

4.5.55 void CLOCK_EnableUsbHs0HostClock (clock_attach_id_t *src,* uint8_t *divider*)

This function enables USB HS host clock.

4.5.56 void CLOCK_DisableUsbHs0HostClock (void)

This function disables USB HS host clock.

4.5.57 bool CLOCK_EnableUsbHs0PhyPllClock (clock_attach_id_t *src,* uint32_t *freq*)

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

4.5.58 void CLOCK_DisableUsbHs0PhyPllClock (void)

This function disables USB hs0PhyPll clock.

4.6 Variable Documentation

4.6.1 volatile uint32_t g_xtalFreq

The XTAL (YSOSC) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetXtalFreq to set the value in to clock driver. For example, if XTAL is 16MHz,

```
* CLOCK_SetXtalFreq(160000000);
*
```

4.6.2 volatile uint32_t g_clkinFreq

The CLK_IN pin (clkin) clock frequency in Hz, when the clock is setup, use the function CLOCK_SetClkinFreq to set the value in to clock driver. For example, if CLK_IN is 16MHz,

```
* CLOCK_SetClkinFreq(160000000);
*
```

4.6.3 volatile uint32_t g_mclkFreq

The MCLK IN clock frequency in Hz, when the clock is setup, use the function CLOCK_SetMclkFreq to set the value in to clock driver. For example, if MCLK IN is 16MHz,

```
* CLOCK_SetMclkFreq(160000000);
*
```

Chapter 5
DSP Driver

5.1 Overview

This driver is used for cooperation with DSP.

Data Structures

- struct dsp_copy_image_t
  Structure for DSP copy image to destination address. More...

Enumerations

- enum dsp_static_vec_sel_t
  Fusion DSP vector table select.

Functions

- void DSP_Init (void)
  Initializing DSP core.
- static void DSP_SetVecRemap (dsp_static_vec_sel_t statVecSel, uint32_t remap)
  Set Fusion DSP static vector table remap.
- void DSP_CopyImage (dsp_copy_image_t *dspCopyImage)
  Copy DSP image to destination address.
- void DSP_Deinit (void)
  Deinitializing DSP core.
- static void DSP_Start (void)
  Start DSP core.
- static void DSP_Stop (void)
  Stop DSP core.

Driver version

- #define FSL_DSP_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  dsp driver version 2.0.1.

5.2 Data Structure Documentation

5.2.1 struct dsp_copy_image_t

Defines start and destination address for copying image with given size.

5.3 Macro Definition Documentation

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

5.4 Function Documentation

5.4.1 void DSP_Init (void)

Power up DSP Enable DSP clock Reset DSP peripheral

5.4.2 static void DSP_SetVecRemap (dsp_static_vec_sel_t *statVecSel,* uint32_t *remap*) [inline],[static]

Parameters

statVecSel	static vector base address selection
remap	static vector remap, valid value from 0x0U to 0xFFFU

5.4.3 void DSP_CopyImage (dsp_copy_image_t ∗ *dspCopyImage*)

Copy DSP image from source address to destination address with given size.

Parameters

dspCopyImage	Structure contains information for DSP copy image to destination address.

Chapter 6
Power Driver

6.1 Overview

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

Data Structures

- struct power_pad_vrange_t
  pad voltage range configuration. More...

Macros

- #define PMIC_VDDCORE_RECOVERY_TIME_IGNORE (0xFFFFFFFFU)
  PMIC is used but vddcore supply is always above LVD threshold.
- #define SYSCTL0_TUPLE_REG(reg) (∗((volatile uint32_t ∗)(((uint32_t)(SYSCTL0)) + (((uint32_t)(reg)) & 0xFFFU))))
  PMIC mode pin configuration API parameter.
- #define POWER_INVALID_VOLT_LEVEL (0xFFFFFFFFU)
  Invalid voltage level.
- #define POWER_FREQ_LEVELS_NUM (5U)
  Core frequency levels number.

Enumerations

- enum _pmc_interrupt {
  kPMC_INT_LVDCORE = PMC_CTRL_LVDCOREIE_MASK,
  kPMC_INT_HVDCORE = PMC_CTRL_HVDCOREIE_MASK,
  kPMC_INT_HVD1V8 = PMC_CTRL_HVD1V8IE_MASK,
  kPMC_INT_AUTOWK = PMC_CTRL_AUTOWKEN_MASK,
  kPMC_INT_INTRPAD }
  PMC event flags.
- enum _pmc_event_flags {
  kPMC_FLAGS_PORCORE = PMC_FLAGS_PORCOREF_MASK,
  kPMC_FLAGS_POR1V8,
  kPMC_FLAGS_PORAO18,
  kPMC_FLAGS_LVDCORE,
  kPMC_FLAGS_HVDCORE,
  kPMC_FLAGS_HVD1V8,
  kPMC_FLAGS_RTC,
  kPMC_FLAGS_AUTOWK,
  kPMC_FLAGS_INTNPADF = PMC_FLAGS_INTNPADF_MASK,
  kPMC_FLAGS_RESETNPAD = PMC_FLAGS_RESETNPADF_MASK,
  kPMC_FLAGS_DEEPPD = PMC_FLAGS_DEEPPDF_MASK }

PMC event flags.

- enum power_mode_cfg_t
  Power mode configuration API parameter.
- enum body_bias_mode_t
  Body Bias mode definition.
- enum power_deep_sleep_clk_t
  Clock source of main clock before entering deep sleep.
- enum power_pad_vrange_val_t {
  kPadVol_171_360 = 0U,
  kPadVol_Continuous = 0U,
  kPadVol_171_198 = 1U,
  kPadVol_300_360 = 2U }
  pad voltage range value.
- enum power_lvd_falling_trip_vol_val_t {
  kLvdFallingTripVol_720 = 0U,
  kLvdFallingTripVol_735 = 1U,
  kLvdFallingTripVol_750 = 2U,
  kLvdFallingTripVol_765 = 3U,
  kLvdFallingTripVol_780 = 4U,
  kLvdFallingTripVol_795 = 5U,
  kLvdFallingTripVol_810 = 6U,
  kLvdFallingTripVol_825 = 7U,
  kLvdFallingTripVol_840 = 8U,
  kLvdFallingTripVol_855 = 9U,
  kLvdFallingTripVol_870 = 10U,
  kLvdFallingTripVol_885 = 11U,
  kLvdFallingTripVol_900 = 12U,
  kLvdFallingTripVol_915 = 13U,
  kLvdFallingTripVol_930 = 14U,
  kLvdFallingTripVol_945 = 15U }
  LVD falling trip voltage value.

Functions

- void POWER_EnablePD (pd_bit_t en)
  API to enable PDRUNCFG bit in the Sysctl0.
- void POWER_DisablePD (pd_bit_t en)
  API to disable PDRUNCFG bit in the Sysctl0.
- static void POWER_EnableDeepSleep (void)
  API to enable deep sleep bit in the ARM Core.
- static void POWER_DisableDeepSleep (void)
  API to disable deep sleep bit in the ARM Core.
- void POWER_UpdateOscSettlingTime (uint32_t osc_delay)
  API to update XTAL oscillator settling time .
- void POWER_UpdatePmicRecoveryTime (uint32_t pmic_delay)
  API to update on-board PMIC vddcore recovery time.
- void POWER_ApplyPD (void)
  API to apply updated PMC PDRUNCFG bits in the Sysctl0.

- void POWER_ClearEventFlags (uint32_t statusMask)
  Clears the PMC event flags state.
- uint32_t POWER_GetEventFlags (void)
  Get the PMC event flags state.
- void POWER_EnableInterrupts (uint32_t interruptMask)
  Enable the PMC interrupt requests.
- void POWER_DisableInterrupts (uint32_t interruptMask)
  Disable the PMC interrupt requests.
- void POWER_SetAnalogBuffer (bool enable)
  Set the PMC analog buffer for references or ATX2.
- static uint32_t POWER_GetPmicMode (pmic_mode_reg_t reg)
  Get PMIC_MODE pins configure value.
- static body_bias_mode_t POWER_GetBodyBiasMode (pmic_mode_reg_t reg)
  Get RBB/FBB bit value.
- void POWER_SetPadVolRange (const power_pad_vrange_t *config)
  Configure pad voltage level.
- void POWER_EnterRbb (void)
  PMC Enter Rbb mode function call.
- void POWER_EnterFbb (void)
  PMC Enter Fbb mode function call.
- void POWER_EnterNbb (void)
  PMC exit Rbb & Fbb mode function call.
- bool POWER_SetLdoVoltageForFreq (uint32_t cm33_clk_freq, uint32_t dsp_clk_freq)
  PMC Set Ldo volatage for particular frequency. NOTE: If LVD falling trip voltage is higher than the required core voltage for particular frequency, LVD voltage will be decreased to safe level to avoid unexpected LVD reset or interrupt event.
- void POWER_SetLvdFallingTripVoltage (power_lvd_falling_trip_vol_val_t volt)
  Set vddcore low voltage detection falling trip voltage.
- power_lvd_falling_trip_vol_val_t POWER_GetLvdFallingTripVoltage (void)
  Get current vddcore low voltage detection falling trip voltage.
- void POWER_DisableLVD (void)
  Disable low voltage detection, no reset or interrupt is triggered when vddcore voltage drops below threshold. NOTE: This API is for internal use only. Application should not touch it.
- void POWER_RestoreLVD (void)
  Restore low voltage detection setting. NOTE: This API is for internal use only. Application should not touch it.
- void POWER_SetPmicMode (uint32_t mode, pmic_mode_reg_t reg)
  Set PMIC_MODE pins configure value.
- void POWER_SetDeepSleepClock (power_deep_sleep_clk_t clk)
  Set deep sleep clock source of main clock.
- void POWER_EnterSleep (void)
  Configures and enters in SLEEP low power mode.
- void POWER_EnterDeepSleep (const uint32_t exclude_from_pd[4])
  PMC Deep Sleep function call.
- void POWER_EnterDeepPowerDown (const uint32_t exclude_from_pd[4])
  PMC Deep Power Down function call.
- void POWER_EnterFullDeepPowerDown (const uint32_t exclude_from_pd[4])
  PMC Full Deep Power Down function call.
- void POWER_EnterPowerMode (power_mode_cfg_t mode, const uint32_t exclude_from_pd[4])
  Power Library API to enter different power mode.
- void EnableDeepSleepIRQ (IRQn_Type interrupt)

Enable specific interrupt for wake-up from deep-sleep mode.
- void DisableDeepSleepIRQ (IRQn_Type interrupt)
  Disable specific interrupt for wake-up from deep-sleep mode.
- uint32_t POWER_GetLibVersion (void)
  Power Library API to return the library version.

Variables

- const uint32_t powerFreqLevel [POWER_FREQ_LEVELS_NUM]
  Frequency levels defined in power library.

Driver version

- #define FSL_POWER_DRIVER_VERSION (MAKE_VERSION(2, 3, 3))
  power driver version 2.3.3.

6.2 Data Structure Documentation

6.2.1 struct power_pad_vrange_t

Data Fields

- uint32_t Vdde0Range: 2
  VDDE0 voltage range for VDDIO_0.
- uint32_t Vdde1Range: 2
  VDDE1 voltage range for VDDIO_1.
- uint32_t Vdde2Range: 2
  VDDE2 voltage range for VDDIO_2.
- uint32_t Vdde3Range: 2
  VDDE3 voltage range for VDDIO_3.
- uint32_t Vdde4Range: 2
  VDDE4 voltage range for VDDIO_4.
- uint32_t __pad0__: 22
  Reserved.

Field Documentation

(1) uint32_t power_pad_vrange_t::Vdde0Range

power_pad_vrange_val_t

(2) uint32_t power_pad_vrange_t::Vdde1Range

power_pad_vrange_val_t

(3) uint32_t power_pad_vrange_t::Vdde2Range

power_pad_vrange_val_t

(4) uint32_t power_pad_vrange_t::Vdde3Range

power_pad_vrange_val_t

(5) uint32_t power_pad_vrange_t::Vdde4Range

power_pad_vrange_val_t

(6) uint32_t power_pad_vrange_t::__pad0__

6.3 Macro Definition Documentation

6.3.1 #define FSL_POWER_DRIVER_VERSION (MAKE_VERSION(2, 3, 3))

6.3.2 #define PMIC_VDDCORE_RECOVERY_TIME_IGNORE (0xFFFFFFFFU)

6.3.3 #define POWER_INVALID_VOLT_LEVEL (0xFFFFFFFFU)

6.3.4 #define POWER_FREQ_LEVELS_NUM (5U)

6.4 Enumeration Type Documentation

6.4.1 enum _pmc_interrupt

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kPMC_INT_LVDCORE Vddcore Low-Voltage Detector Interrupt Enable.
kPMC_INT_HVDCORE Vddcore High-Voltage Detector Interrupt Enable.
kPMC_INT_HVD1V8 Vdd1v8 High-Voltage Detector Interrupt Enable.
kPMC_INT_AUTOWK PMC automatic wakeup enable and interrupt enable.
kPMC_INT_INTRPAD Interrupt pad deep powerdown and deep sleep wake up & interrupt enable.

6.4.2 enum _pmc_event_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kPMC_FLAGS_PORCORE POR triggered by the vddcore POR monitor (0 = no, 1 = yes).
kPMC_FLAGS_POR1V8 vdd1v8 power on event detected since last cleared(0 = no, 1 = yes).

kPMC_FLAGS_PORAO18 vdd_ao18 power on event detected since last cleared (0 = no, 1 = yes).
kPMC_FLAGS_LVDCORE LVD tripped since last time this bit was cleared (0 = no, 1 = yes).
kPMC_FLAGS_HVDCORE HVD tripped since last time this bit was cleared (0 = no, 1 = yes).
kPMC_FLAGS_HVD1V8 vdd1v8 HVD tripped since last time this bit was cleared (0 = no, 1 = yes).

kPMC_FLAGS_RTC RTC wakeup detected since last time flag was cleared (0 = no, 1 = yes).
kPMC_FLAGS_AUTOWK PMC Auto wakeup caused a deep sleep wakeup and interrupt (0 = no, 1 = yes).
kPMC_FLAGS_INTNPADF Pad interrupt caused a wakeup or interrupt event since the last time this flag was cleared (0 = no, 1 = yes).
kPMC_FLAGS_RESETNPAD Reset pad wakeup caused a wakeup or reset event since the last time this bit was cleared. (0 = no, 1 = yes).
kPMC_FLAGS_DEEPPD Deep powerdown was entered since the last time this flag was cleared (0 = no, 1 = yes).

6.4.3 enum power_deep_sleep_clk_t

6.4.4 enum power_pad_vrange_val_t

Note, refer to Reference Manual PMC GPIO VDDIO Range Selection Control (PADVRANGE) register's description for the supported voltage by different VDDDIO.

Enumerator

kPadVol_171_360 Deprecated! Voltage from 1.71V to 3.60V.
kPadVol_Continuous Continuous mode, VDDE detector on.
kPadVol_171_198 Voltage from 1.71V to 1.98V. VDDE detector off.
kPadVol_300_360 Voltage from 3.00V to 3.60V. VDDE detector off.

6.4.5 enum power_lvd_falling_trip_vol_val_t

Enumerator

kLvdFallingTripVol_720 Voltage 720mV.
kLvdFallingTripVol_735 Voltage 735mV.
kLvdFallingTripVol_750 Voltage 750mV.
kLvdFallingTripVol_765 Voltage 765mV.
kLvdFallingTripVol_780 Voltage 780mV.
kLvdFallingTripVol_795 Voltage 795mV.
kLvdFallingTripVol_810 Voltage 810mV.
kLvdFallingTripVol_825 Voltage 825mV.
kLvdFallingTripVol_840 Voltage 840mV.

kLvdFallingTripVol_855 Voltage 855mV.
kLvdFallingTripVol_870 Voltage 870mV.
kLvdFallingTripVol_885 Voltage 885mV.
kLvdFallingTripVol_900 Voltage 900mV.
kLvdFallingTripVol_915 Voltage 915mV.
kLvdFallingTripVol_930 Voltage 930mV.
kLvdFallingTripVol_945 Voltage 945mV.

6.5 Function Documentation

6.5.1 void POWER_EnablePD (pd_bit_t *en*)

Note that enabling the bit powers down the peripheral

Parameters

en	peripheral for which to enable the PDRUNCFG bit

6.5.2 void POWER_DisablePD (pd_bit_t *en*)

Note that disabling the bit powers up the peripheral

Parameters

en	peripheral for which to disable the PDRUNCFG bit

6.5.3 void POWER_UpdateOscSettlingTime (uint32_t *osc_delay*)

Parameters

osc_delay	: OSC stabilization time in unit of microsecond

6.5.4 void POWER_UpdatePmicRecoveryTime (uint32_t *pmic_delay*)

NOTE: If LDO is used instead of PMIC, don't call it. Otherwise it must be called to allow power library to well handle the deep sleep process.

Parameters

pmic_delay	: PMIC stabilization time in unit of microsecond, or PMIC_VDDCORE_RECOVERY_TIME_IGNORE if not care.

6.5.5 void POWER_ClearEventFlags (uint32_t *statusMask*)

Parameters

statusMask	: A bitmask of event flags that are to be cleared.

6.5.6 uint32_t POWER_GetEventFlags (void)

Returns

PMC FLAGS register value

6.5.7 void POWER_EnableInterrupts (uint32_t *interruptMask*)

Parameters

interruptMask	: A bitmask of of interrupts to enable.

6.5.8 void POWER_DisableInterrupts (uint32_t *interruptMask*)

Parameters

interruptMask	: A bitmask of of interrupts to disable.

6.5.9 void POWER_SetAnalogBuffer (bool *enable*)

Parameters

enable	: Set to true to enable analog buffer for references or ATX2, false to disable.

6.5.10 static uint32_t POWER_GetPmicMode (pmic_mode_reg_t *reg*) [inline], [static]

Parameters

reg	: PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns

PMIC_MODE pins value in PDSLEEPCFG0

6.5.11 static body_bias_mode_t POWER_GetBodyBiasMode (pmic_mode_reg_t *reg*) [inline], [static]

Parameters

reg	: PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns

Current body bias mode

6.5.12 void POWER_SetPadVolRange (const power_pad_vrange_t ∗ *config*)

Wide voltage range cost more power due to enabled voltage detector.

NOTE: BE CAUTIOUS TO CALL THIS API. IF THE PAD SUPPLY IS BEYOND THE SET RANGE, SILICON MIGHT BE DAMAGED.

Parameters

config	pad voltage range configuration.

6.5.13 bool POWER_SetLdoVoltageForFreq (uint32_t *cm33_clk_freq,* uint32_t *dsp_clk_freq*)

Parameters

cm33_clk_freq	: CM33 core frequency value
dsp_clk_freq	: dsp core frequency value

Returns

true for success and false for CPU frequency out of specified voltOpRange.

6.5.14 void POWER_SetLvdFallingTripVoltage (power_lvd_falling_trip_vol_val_t *volt*)

Parameters

volt	target LVD voltage to set.

6.5.15 power_lvd_falling_trip_vol_val_t POWER_GetLvdFallingTripVoltage (void)

Returns

Current LVD voltage.

6.5.16 void POWER_SetPmicMode (uint32_t *mode,* pmic_mode_reg_t *reg*)

Parameters

mode	: PMIC MODE pin value
reg	: PDSLEEPCFG0 or PDRUNCFG0 register offset

Returns

PMIC_MODE pins value in PDSLEEPCFG0

6.5.17 void POWER_SetDeepSleepClock (power_deep_sleep_clk_t *clk*)

Parameters

clk	: clock source of main clock.

6.5.18 void POWER_EnterDeepSleep (const uint32_t *exclude_from_pd[4]*)

Parameters

exclude_from_-pd	Bit mask of the PDRUNCFG0 ∼ PDRUNCFG3 that needs to be powered on during Deep Sleep mode selected.

6.5.19 void POWER_EnterDeepPowerDown (const uint32_t *exclude_from_pd[4]*)

Parameters

exclude_from_-pd	Bit mask of the PDRUNCFG0 ∼ PDRUNCFG3 that needs to be powered on during Deep Power Down mode selected.

6.5.20 void POWER_EnterFullDeepPowerDown (const uint32_t *exclude_from_pd[4]*)

Parameters

exclude_from_-pd	Bit mask of the PDRUNCFG0 ∼ PDRUNCFG3 that needs to be powered on during Full Deep Power Down mode selected.

6.5.21 void POWER_EnterPowerMode (power_mode_cfg_t *mode,* const uint32_t *exclude_from_pd[4]*)

Parameters

mode	Power mode to enter.
exclude_from_-pd	Bit mask of the PDRUNCFG0 ∼ PDRUNCFG3 that needs to be powered on during power mode selected.

6.5.22 void EnableDeepSleepIRQ (IRQn_Type *interrupt*)

Enable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note

This function also enables the interrupt in the NVIC (EnableIRQ() is called internally).

Parameters

interrupt	The IRQ number.

6.5.23 void DisableDeepSleepIRQ (IRQn_Type *interrupt*)

Disable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

Note

This function also disables the interrupt in the NVIC (DisableIRQ() is called internally).

Parameters

interrupt	The IRQ number.

6.5.24 uint32_t POWER_GetLibVersion (void)

Returns

version number of the power library

6.6 Variable Documentation

6.6.1 const uint32_t powerFreqLevel[POWER_FREQ_LEVELS_NUM]

Chapter 7
Reset Driver

7.1 Overview

Reset driver supports peripheral reset and system reset.

Macros

- #define RST_CTL0_PSCCTL0 0
  Reset control registers index.
- #define ADC_RSTS

Typedefs

- typedef RSTCTL_RSTn_t reset_ip_name_t
  IP reset handle.

Enumerations

- enum RSTCTL_RSTn_t {
  kDSP_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 1U,
  kAXI_SWITCH_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 3U,
  kPOWERQUAD_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 8U,
  kCASPER_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 9U,
  kHASHCRYPT_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 10U,
  kPUF_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 11U,
  kRNG_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 12U,
  kFLEXSPI0_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 16U,
  kFLEXSPI1_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 18U,
  kUSBHS_PHY_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 20U,
  kUSBHS_DEVICE_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 21U,
  kUSBHS_HOST_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 22U,
  kUSBHS_SRAM_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 23U,
  kSCT_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 24U,
  kGPU_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 26U,
  kDISP_CTRL_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 27U,
  kMIPI_DSI_CTRL_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 28U,
  kMIPI_DSI_PHY_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 29U,
  kSMART_DMA_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL0 << 8) | 30U,
  kSDIO0_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL1 << 8) | 2U,
  kSDIO1_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL1 << 8) | 3U,
  kACMP0_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL1 << 8) | 15U,
  kADC0_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL1 << 8) | 16U,
  kSHSGPIO0_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL1 << 8) | 24U,
  kUTICK0_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL2 << 8) | 0U,
  kWWDT0_RST_SHIFT_RSTn = (RST_CTL0_PSCCTL2 << 8) | 1U,
  kFC0_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 8U,
  kFC1_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 9U,
  kFC2_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 10U,
  kFC3_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 11U,
  kFC4_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 12U,
  kFC5_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 13U,
  kFC6_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 14U,
  kFC7_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 15U,
  kFC8_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 16U,
  kFC9_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 17U,
  kFC10_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 18U,
  kFC11_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 19U,
  kFC12_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 20U,
  kFC13_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 21U,
  kFC14_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 22U,
  kFC15_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 23U,
  kDMIC_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 24U,
  kFC16_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 25U,
  kOSEVENT_TIMER_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 27U,
  kHLEVIO_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL0 << 8) | 29U,
  kHSGPIO0_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL1 << 8) | 0U,

kINPUTMUX_RST_SHIFT_RSTn = (RST_CTL1_PSCCTL2 << 8) | 31U }
Enumeration for peripheral reset control bits.

Functions

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

Driver version

- #define FSL_RESET_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  reset driver version 2.0.1.

7.2 Macro Definition Documentation

7.2.1 #define FSL_RESET_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

7.2.2 #define ADC_RSTS

Value:

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

Array initializers with peripheral reset bits

7.3 Enumeration Type Documentation

7.3.1 enum RSTCTL_RSTn_t

Defines the enumeration for peripheral reset control bits in RSTCLTx registers

Enumerator

kDSP_RST_SHIFT_RSTn DSP reset control
kAXI_SWITCH_RST_SHIFT_RSTn AXI Switch reset control
kPOWERQUAD_RST_SHIFT_RSTn POWERQUAD reset control
kCASPER_RST_SHIFT_RSTn CASPER reset control
kHASHCRYPT_RST_SHIFT_RSTn HASHCRYPT reset control
kPUF_RST_SHIFT_RSTn Physical unclonable function reset control
kRNG_RST_SHIFT_RSTn Random number generator (RNG) reset control
kFLEXSPI0_RST_SHIFT_RSTn FLEXSPI0/OTFAD reset control

kFLEXSPI1_RST_SHIFT_RSTn FLEXSPI1 reset control
kUSBHS_PHY_RST_SHIFT_RSTn High speed USB PHY reset control
kUSBHS_DEVICE_RST_SHIFT_RSTn High speed USB Device reset control
kUSBHS_HOST_RST_SHIFT_RSTn High speed USB Host reset control
kUSBHS_SRAM_RST_SHIFT_RSTn High speed USB SRAM reset control
kSCT_RST_SHIFT_RSTn Standard ctimers reset control
kGPU_RST_SHIFT_RSTn GPU reset control
kDISP_CTRL_RST_SHIFT_RSTn Display controller reset control
kMIPI_DSI_CTRL_RST_SHIFT_RSTn MIPI DSI controller reset control
kMIPI_DSI_PHY_RST_SHIFT_RSTn MIPI DSI PHY reset control
kSMART_DMA_RST_SHIFT_RSTn Smart DMA reset control
kSDIO0_RST_SHIFT_RSTn SDIO0 reset control
kSDIO1_RST_SHIFT_RSTn SDIO1 reset control
kACMP0_RST_SHIFT_RSTn Grouped interrupt (PINT) reset control.
kADC0_RST_SHIFT_RSTn ADC0 reset control
kSHSGPIO0_RST_SHIFT_RSTn Security HSGPIO 0 reset control
kUTICK0_RST_SHIFT_RSTn Micro-tick timer reset control
kWWDT0_RST_SHIFT_RSTn Windowed Watchdog timer 0 reset control
kFC0_RST_SHIFT_RSTn Flexcomm Interface 0 reset control
kFC1_RST_SHIFT_RSTn Flexcomm Interface 1 reset control
kFC2_RST_SHIFT_RSTn Flexcomm Interface 2 reset control
kFC3_RST_SHIFT_RSTn Flexcomm Interface 3 reset control
kFC4_RST_SHIFT_RSTn Flexcomm Interface 4 reset control
kFC5_RST_SHIFT_RSTn Flexcomm Interface 5 reset control
kFC6_RST_SHIFT_RSTn Flexcomm Interface 6 reset control
kFC7_RST_SHIFT_RSTn Flexcomm Interface 7 reset control
kFC8_RST_SHIFT_RSTn Flexcomm Interface 8 reset control
kFC9_RST_SHIFT_RSTn Flexcomm Interface 9 reset control
kFC10_RST_SHIFT_RSTn Flexcomm Interface 10 reset control
kFC11_RST_SHIFT_RSTn Flexcomm Interface 11 reset control
kFC12_RST_SHIFT_RSTn Flexcomm Interface 12 reset control
kFC13_RST_SHIFT_RSTn Flexcomm Interface 13 reset control
kFC14_RST_SHIFT_RSTn Flexcomm Interface 14 reset control
kFC15_RST_SHIFT_RSTn Flexcomm Interface 15 reset control
kDMIC_RST_SHIFT_RSTn Digital microphone interface reset control
kFC16_RST_SHIFT_RSTn Flexcomm Interface 16 reset control
kOSEVENT_TIMER_RST_SHIFT_RSTn Osevent Timer reset control
kFLEXIO_RST_SHIFT_RSTn FlexIO reset control
kHSGPIO0_RST_SHIFT_RSTn HSGPIO 0 reset control
kHSGPIO1_RST_SHIFT_RSTn HSGPIO 1 reset control
kHSGPIO2_RST_SHIFT_RSTn HSGPIO 2 reset control
kHSGPIO3_RST_SHIFT_RSTn HSGPIO 3 reset control
kHSGPIO4_RST_SHIFT_RSTn HSGPIO 4 reset control
kHSGPIO5_RST_SHIFT_RSTn HSGPIO 5 reset control
kHSGPIO6_RST_SHIFT_RSTn HSGPIO 6 reset control

kHSGPIO7_RST_SHIFT_RSTn HSGPIO 7 reset control
kCRC_RST_SHIFT_RSTn CRC reset control
kDMAC0_RST_SHIFT_RSTn DMA Controller 0 reset control
kDMAC1_RST_SHIFT_RSTn DMA Controller 1 reset control
kMU_RST_SHIFT_RSTn Message Unit reset control
kSEMA_RST_SHIFT_RSTn Semaphore reset control
kFREQME_RST_SHIFT_RSTn Frequency Measure reset control
kCT32B0_RST_SHIFT_RSTn CT32B0 reset control
kCT32B1_RST_SHIFT_RSTn CT32B1 reset control
kCT32B2_RST_SHIFT_RSTn CT32B3 reset control
kCT32B3_RST_SHIFT_RSTn CT32B4 reset control
kCT32B4_RST_SHIFT_RSTn CT32B4 reset control
kMRT0_RST_SHIFT_RSTn Multi-rate timer (MRT) reset control
kWWDT1_RST_SHIFT_RSTn Windowed Watchdog timer 1 reset control
kI3C0_RST_SHIFT_RSTn I3C0 reset control
kI3C1_RST_SHIFT_RSTn I3C1 reset control
kPINT_RST_SHIFT_RSTn GPIO Pin interrupt reset control
kINPUTMUX_RST_SHIFT_RSTn Peripheral input muxes reset control

7.4 Function Documentation

7.4.1 void RESET_SetPeripheralReset (reset_ip_name_t *peripheral*)

Asserts reset signal to specified peripheral module.

Parameters

peripheral	Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

7.4.2 void RESET_ClearPeripheralReset (reset_ip_name_t *peripheral*)

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

Parameters

peripheral	Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register.

7.4.3 void RESET_PeripheralReset (reset_ip_name_t *peripheral*)

Reset peripheral module.

Parameters

peripheral	Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register.

Chapter 8
ACMP: Analog Comparator Driver

8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Comparator (ACMP) module of MCUXpresso SDK devices.

The ACMP driver is created to help the user operate the ACMP module better. This driver can be considered as a basic comparator with advanced features. The APIs for basic comparator can make the C-MP work as a general comparator, which compares the two input channel's voltage and creates the output of the comparator result immediately. The APIs for advanced feature can be used as the plug-in function based on the basic comparator, and can provide more ways to process the comparator's output.

8.2 Typical use case

8.2.1 Normal Configuration

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

8.2.2 Interrupt Configuration

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

8.2.3 Round robin Configuration

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

Data Structures

- struct acmp_config_t
  Configuration for ACMP. More...
- struct acmp_channel_config_t
  Configuration for channel. More...
- struct acmp_filter_config_t
  Configuration for filter. More...
- struct acmp_dac_config_t
  Configuration for DAC. More...
- struct acmp_round_robin_config_t
  Configuration for round robin mode. More...
- struct acmp_discrete_mode_config_t
  Configuration for discrete mode. More...

Macros

- #define CMP_C0_CFx_MASK (CMP_C0_CFR_MASK | CMP_C0_CFF_MASK)
  The mask of status flags cleared by writing 1.

Enumerations

- enum _acmp_interrupt_enable {
  kACMP_OutputRisingInterruptEnable = (1U << 0U),
  kACMP_OutputFallingInterruptEnable = (1U << 1U),
  kACMP_RoundRobinInterruptEnable = (1U << 2U) }
  Interrupt enable/disable mask.
- enum _acmp_status_flags {
  kACMP_OutputRisingEventFlag = CMP_C0_CFR_MASK,
  kACMP_OutputFallingEventFlag = CMP_C0_CFF_MASK,
  kACMP_OutputAssertEventFlag = CMP_C0_COUT_MASK }
  Status flag mask.
- enum acmp_hysteresis_mode_t {
  kACMP_HysteresisLevel0 = 0U,
  kACMP_HysteresisLevel1 = 1U,
  kACMP_HysteresisLevel2 = 2U,
  kACMP_HysteresisLevel3 = 3U }
  Comparator hard block hysteresis control.
- enum acmp_reference_voltage_source_t {
  kACMP_VrefSourceVin1 = 0U,
  kACMP_VrefSourceVin2 = 1U }
  CMP Voltage Reference source.
- enum acmp_fixed_port_t {
  kACMP_FixedPlusPort = 0U,
  kACMP_FixedMinusPort = 1U }
  Fixed mux port.
- enum acmp_dac_work_mode_t {
  kACMP_DACWorkLowSpeedMode = 0U,
  kACMP_DACWorkHighSpeedMode = 1U }
  Internal DAC's work mode.
- enum acmp_discrete_clock_source_t {
  kACMP_DiscreteClockSlow = 0U,
  kACMP_DiscreteClockFast = 1U }
  Discrete mode clock selection.
- enum acmp_discrete_sample_time_t {
  kACMP_DiscreteSampleTimeAs1T = 0U,
  kACMP_DiscreteSampleTimeAs2T = 1U,
  kACMP_DiscreteSampleTimeAs4T = 2U,
  kACMP_DiscreteSampleTimeAs8T = 3U,
  kACMP_DiscreteSampleTimeAs16T = 4U,
  kACMP_DiscreteSampleTimeAs32T = 5U,
  kACMP_DiscreteSampleTimeAs64T = 6U,
  kACMP_DiscreteSampleTimeAs256T = 7U }

ACMP discrete sample selection.

- enum acmp_discrete_phase_time_t {
  kACMP_DiscretePhaseTimeAlt0 = 0U,
  kACMP_DiscretePhaseTimeAlt1 = 1U,
  kACMP_DiscretePhaseTimeAlt2 = 2U,
  kACMP_DiscretePhaseTimeAlt3 = 3U,
  kACMP_DiscretePhaseTimeAlt4 = 4U,
  kACMP_DiscretePhaseTimeAlt5 = 5U,
  kACMP_DiscretePhaseTimeAlt6 = 6U,
  kACMP_DiscretePhaseTimeAlt7 = 7U }
  ACMP discrete phase time selection.

Driver version

- #define FSL_ACMP_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 6U))
  ACMP driver version 2.0.6.

Initialization and deinitialization

- void ACMP_Init (CMP_Type *base, const acmp_config_t *config)
  Initializes the ACMP.
- void ACMP_Deinit (CMP_Type *base)
  Deinitializes the ACMP.
- void ACMP_GetDefaultConfig (acmp_config_t *config)
  Gets the default configuration for ACMP.

Basic Operations

- void ACMP_Enable (CMP_Type *base, bool enable)
  Enables or disables the ACMP.
- void ACMP_EnableLinkToDAC (CMP_Type *base, bool enable)
  Enables the link from CMP to DAC enable.
- void ACMP_SetChannelConfig (CMP_Type *base, const acmp_channel_config_t *config)
  Sets the channel configuration.

Advanced Operations

- void ACMP_EnableDMA (CMP_Type *base, bool enable)
  Enables or disables DMA.
- void ACMP_EnableWindowMode (CMP_Type *base, bool enable)
  Enables or disables window mode.
- void ACMP_SetFilterConfig (CMP_Type *base, const acmp_filter_config_t *config)
  Configures the filter.
- void ACMP_SetDACConfig (CMP_Type *base, const acmp_dac_config_t *config)
  Configures the internal DAC.
- void ACMP_SetRoundRobinConfig (CMP_Type *base, const acmp_round_robin_config_t *config)
  Configures the round robin mode.
- void ACMP_SetRoundRobinPreState (CMP_Type *base, uint32_t mask)
  Defines the pre-set state of channels in round robin mode.

- static uint32_t ACMP_GetRoundRobinStatusFlags (CMP_Type *base)
  Gets the channel input changed flags in round robin mode.
- void ACMP_ClearRoundRobinStatusFlags (CMP_Type *base, uint32_t mask)
  Clears the channel input changed flags in round robin mode.
- static uint32_t ACMP_GetRoundRobinResult (CMP_Type *base)
  Gets the round robin result.

Interrupts

- void ACMP_EnableInterrupts (CMP_Type *base, uint32_t mask)
  Enables interrupts.
- void ACMP_DisableInterrupts (CMP_Type *base, uint32_t mask)
  Disables interrupts.

Status

- uint32_t ACMP_GetStatusFlags (CMP_Type *base)
  Gets status flags.
- void ACMP_ClearStatusFlags (CMP_Type *base, uint32_t mask)
  Clears status flags.

Discrete mode

- void ACMP_SetDiscreteModeConfig (CMP_Type *base, const acmp_discrete_mode_config_t *config)
  Configure the discrete mode.
- void ACMP_GetDefaultDiscreteModeConfig (acmp_discrete_mode_config_t *config)
  Get the default configuration for discrete mode setting.

8.3 Data Structure Documentation

8.3.1 struct acmp_config_t

Data Fields

- acmp_hysteresis_mode_t hysteresisMode
  Hysteresis mode.
- bool enableHighSpeed
  Enable High Speed (HS) comparison mode.
- bool enableInvertOutput
  Enable inverted comparator output.
- bool useUnfilteredOutput
  Set compare output(COUT) to equal COUTA(true) or COUT(false).
- bool enablePinOut
  The comparator output is available on the associated pin.

Field Documentation

(1) acmp_hysteresis_mode_t acmp_config_t::hysteresisMode

(2) bool acmp_config_t::enableHighSpeed

(3) bool acmp_config_t::enableInvertOutput

(4) bool acmp_config_t::useUnfilteredOutput

(5) bool acmp_config_t::enablePinOut

8.3.2 struct acmp_channel_config_t

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

Data Fields

- uint32_t plusMuxInput
  Plus mux input channel(0∼7).
- uint32_t minusMuxInput
  Minus mux input channel(0∼7).

Field Documentation

(1) uint32_t acmp_channel_config_t::plusMuxInput

(2) uint32_t acmp_channel_config_t::minusMuxInput

8.3.3 struct acmp_filter_config_t

Data Fields

- bool enableSample
  Using external SAMPLE as sampling clock input, or using divided bus clock.
- uint32_t filterCount
  Filter Sample Count.
- uint32_t filterPeriod
  Filter Sample Period.

Field Documentation

(1) bool acmp_filter_config_t::enableSample

(2) uint32_t acmp_filter_config_t::filterCount

Available range is 1-7, 0 would cause the filter disabled.

(3) uint32_t acmp_filter_config_t::filterPeriod

The divider to bus clock. Available range is 0-255.

8.3.4 struct acmp_dac_config_t

Data Fields

- acmp_reference_voltage_source_t referenceVoltageSource
  Supply voltage reference source.
- uint32_t DACValue
  Value for DAC Output Voltage.

Field Documentation

(1) acmp_reference_voltage_source_t acmp_dac_config_t::referenceVoltageSource

(2) uint32_t acmp_dac_config_t::DACValue

Available range is 0-255.

8.3.5 struct acmp_round_robin_config_t

Data Fields

- acmp_fixed_port_t fixedPort
  Fixed mux port.
- uint32_t fixedChannelNumber
  Indicates which channel is fixed in the fixed mux port.
- uint32_t checkerChannelMask
  Mask of checker channel index.
- uint32_t sampleClockCount
  Specifies how many round-robin clock cycles(0∼3) later the sample takes place.
- uint32_t delayModulus
  Comparator and DAC initialization delay modulus.

Field Documentation

(1) acmp_fixed_port_t acmp_round_robin_config_t::fixedPort

(2) uint32_t acmp_round_robin_config_t::fixedChannelNumber

(3) uint32_t acmp_round_robin_config_t::checkerChannelMask

Available range is channel0:0x01 to channel7:0x80 for round-robin checker.

(4) uint32_t acmp_round_robin_config_t::sampleClockCount

(5) uint32_t acmp_round_robin_config_t::delayModulus

8.3.6 struct acmp_discrete_mode_config_t

Data Fields

- bool enablePositiveChannelDiscreteMode
  Positive Channel Continuous Mode Enable.
- bool enableNegativeChannelDiscreteMode
  Negative Channel Continuous Mode Enable.
- bool enableResistorDivider
  Resistor Divider Enable is used to enable the resistor divider for the inputs when they come from 3v domain and their values are above 1.8v.
- acmp_discrete_clock_source_t clockSource
  Select the clock source in order to generate the requiried timing for comparator to work in discrete mode.
- acmp_discrete_sample_time_t sampleTime
  Select the ACMP total sampling time period.
- acmp_discrete_phase_time_t phase1Time
  Select the ACMP phase 1 sampling time.
- acmp_discrete_phase_time_t phase2Time
  Select the ACMP phase 2 sampling time.

Field Documentation

(1) bool acmp_discrete_mode_config_t::enablePositiveChannelDiscreteMode

By default, the continuous mode is used.

(2) bool acmp_discrete_mode_config_t::enableNegativeChannelDiscreteMode

By default, the continuous mode is used.

(3) bool acmp_discrete_mode_config_t::enableResistorDivider

(4) acmp_discrete_clock_source_t acmp_discrete_mode_config_t::clockSource

(5) acmp_discrete_sample_time_t acmp_discrete_mode_config_t::sampleTime

(6) acmp_discrete_phase_time_t acmp_discrete_mode_config_t::phase1Time

(7) acmp_discrete_phase_time_t acmp_discrete_mode_config_t::phase2Time

8.4 Macro Definition Documentation

8.4.1 #define FSL_ACMP_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 6U))

8.4.2 #define CMP_C0_CFx_MASK (CMP_C0_CFR_MASK | CMP_C0_CFF_MASK)

8.5 Enumeration Type Documentation

8.5.1 enum _acmp_interrupt_enable

Enumerator

kACMP_OutputRisingInterruptEnable Enable the interrupt when comparator outputs rising.
kACMP_OutputFallingInterruptEnable Enable the interrupt when comparator outputs falling.
kACMP_RoundRobinInterruptEnable Enable the Round-Robin interrupt.

8.5.2 enum _acmp_status_flags

Enumerator

kACMP_OutputRisingEventFlag Rising-edge on compare output has occurred.
kACMP_OutputFallingEventFlag Falling-edge on compare output has occurred.
kACMP_OutputAssertEventFlag Return the current value of the analog comparator output.

8.5.3 enum acmp_hysteresis_mode_t

See chip data sheet to get the actual hysteresis value with each level.

Enumerator

kACMP_HysteresisLevel0 Offset is level 0 and Hysteresis is level 0.
kACMP_HysteresisLevel1 Offset is level 0 and Hysteresis is level 1.
kACMP_HysteresisLevel2 Offset is level 0 and Hysteresis is level 2.
kACMP_HysteresisLevel3 Offset is level 0 and Hysteresis is level 3.

8.5.4 enum acmp_reference_voltage_source_t

Enumerator

kACMP_VrefSourceVin1 Vin1 is selected as resistor ladder network supply reference Vin.
kACMP_VrefSourceVin2 Vin2 is selected as resistor ladder network supply reference Vin.

8.5.5 enum acmp_fixed_port_t

Enumerator

kACMP_FixedPlusPort Only the inputs to the Minus port are swept in each round.
kACMP_FixedMinusPort Only the inputs to the Plus port are swept in each round.

8.5.6 enum acmp_dac_work_mode_t

Enumerator

kACMP_DACWorkLowSpeedMode DAC is selected to work in low speed and low power mode.
kACMP_DACWorkHighSpeedMode DAC is selected to work in high speed high power mode.

8.5.7 enum acmp_discrete_clock_source_t

Enumerator

kACMP_DiscreteClockSlow Slow clock (32kHz) is used as the discrete mode clock.
kACMP_DiscreteClockFast Fast clock (16-20MHz) is used as the discrete mode clock.

8.5.8 enum acmp_discrete_sample_time_t

These values configures the analog comparator sampling timing (speicified by the discrete mode clock period T which is selected by acmp_discrete_clock_source_t) in discrete mode.

Enumerator

kACMP_DiscreteSampleTimeAs1T The sampling time equals to 1xT.
kACMP_DiscreteSampleTimeAs2T The sampling time equals to 2xT.
kACMP_DiscreteSampleTimeAs4T The sampling time equals to 4xT.
kACMP_DiscreteSampleTimeAs8T The sampling time equals to 8xT.
kACMP_DiscreteSampleTimeAs16T The sampling time equals to 16xT.
kACMP_DiscreteSampleTimeAs32T The sampling time equals to 32xT.
kACMP_DiscreteSampleTimeAs64T The sampling time equals to 64xT.
kACMP_DiscreteSampleTimeAs256T The sampling time equals to 256xT.

8.5.9 enum acmp_discrete_phase_time_t

There are two phases for sampling input signals, phase 1 and phase 2.

Enumerator

kACMP_DiscretePhaseTimeAlt0 The phase x active in one sampling selection 0.
kACMP_DiscretePhaseTimeAlt1 The phase x active in one sampling selection 1.
kACMP_DiscretePhaseTimeAlt2 The phase x active in one sampling selection 2.
kACMP_DiscretePhaseTimeAlt3 The phase x active in one sampling selection 3.
kACMP_DiscretePhaseTimeAlt4 The phase x active in one sampling selection 4.
kACMP_DiscretePhaseTimeAlt5 The phase x active in one sampling selection 5.
kACMP_DiscretePhaseTimeAlt6 The phase x active in one sampling selection 6.
kACMP_DiscretePhaseTimeAlt7 The phase x active in one sampling selection 7.

8.6 Function Documentation

8.6.1 void ACMP_Init (CMP_Type * *base*, const acmp_config_t * *config*)

The default configuration can be got by calling ACMP_GetDefaultConfig().

Parameters

base	ACMP peripheral base address.
config	Pointer to ACMP configuration structure.

8.6.2 void ACMP_Deinit (CMP_Type * *base*)

Parameters

base	ACMP peripheral base address.

8.6.3 void ACMP_GetDefaultConfig (acmp_config_t * *config*)

This function initializes the user configuration structure to default value. The default value are:

Example:

```
config->enableHighSpeed = false;
config->enableInvertOutput = false;
config->useUnfilteredOutput = false;
config->enablePinOut = false;
config->enableHysteresisBothDirections = false;
config->hysteresisMode = kACMP_hysteresisMode0;
```

Parameters

config	Pointer to ACMP configuration structure.

8.6.4 void ACMP_Enable (CMP_Type * *base*, bool *enable*)

Parameters

base	ACMP peripheral base address.
enable	True to enable the ACMP.

8.6.5 void ACMP_EnableLinkToDAC (CMP_Type * *base*, bool *enable*)

When this bit is set, the DAC enable/disable is controlled by the bit CMP_C0[EN] instead of CMP_C1[DACEN].

Parameters

base	ACMP peripheral base address.
enable	Enable the feature or not.

8.6.6 void ACMP_SetChannelConfig (CMP_Type * *base*, const acmp_channel_config_t * *config*)

Note that the plus/minus mux's setting is only valid when the positive/negative port's input isn't from DAC but from channel mux.

Example:

```
acmp_channel_config_t configStruct = {0};
configStruct.positivePortInput = kACMP_PortInputFromDAC;
configStruct.negativePortInput = kACMP_PortInputFromMux;
configStruct.minusMuxInput = 1U;
ACMP_SetChannelConfig(CMP0, &configStruct);
```

Parameters

base	ACMP peripheral base address.
config	Pointer to channel configuration structure.

8.6.7 void ACMP_EnableDMA (CMP_Type * *base*, bool *enable*)

Parameters

base	ACMP peripheral base address.
enable	True to enable DMA.

8.6.8 void ACMP_EnableWindowMode (CMP_Type * *base*, bool *enable*)

Parameters

base	ACMP peripheral base address.
enable	True to enable window mode.

8.6.9 void ACMP_SetFilterConfig (CMP_Type * *base*, const acmp_filter_config_t * *config*)

The filter can be enabled when the filter count is bigger than 1, the filter period is greater than 0 and the sample clock is from divided bus clock or the filter is bigger than 1 and the sample clock is from external clock. Detailed usage can be got from the reference manual.

Example:

```
acmp_filter_config_t configStruct = {0};
configStruct.filterCount = 5U;
configStruct.filterPeriod = 200U;
configStruct.enableSample = false;
ACMP_SetFilterConfig(CMP0, &configStruct);
```

Parameters

base	ACMP peripheral base address.
config	Pointer to filter configuration structure.

8.6.10 void ACMP_SetDACConfig (CMP_Type * *base*, const acmp_dac_config_t * *config*)

Example:

```
acmp_dac_config_t configStruct = {0};
configStruct.referenceVoltageSource = kACMP_VrefSourceVin1;
configStruct.DACValue = 20U;
configStruct.enableOutput = false;
configStruct.workMode = kACMP_DACWorkLowSpeedMode;
ACMP_SetDACConfig(CMP0, &configStruct);
```

Parameters

base	ACMP peripheral base address.
config	Pointer to DAC configuration structure. "NULL" is for disabling the feature.

8.6.11 void ACMP_SetRoundRobinConfig (CMP_Type * *base*, const acmp_round_robin_config_t * *config*)

Example:

```
acmp_round_robin_config_t configStruct = {0};
configStruct.fixedPort = kACMP_FixedPlusPort;
configStruct.fixedChannelNumber = 3U;
configStruct.checkerChannelMask = 0xF7U;
configStruct.sampleClockCount = 0U;
configStruct.delayModulus = 0U;
ACMP_SetRoundRobinConfig(CMP0, &configStruct);
```

Parameters

base	ACMP peripheral base address.
config	Pointer to round robin mode configuration structure. "NULL" is for disabling the feature.

8.6.12 void ACMP_SetRoundRobinPreState (CMP_Type * *base*, uint32_t *mask*)

Note: The pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So get-round-robin-result can't return the same value as the value are set by pre-state.

Parameters

base	ACMP peripheral base address.
mask	Mask of round robin channel index. Available range is channel0:0x01 to channel7-:0x80.

8.6.13 static uint32_t ACMP_GetRoundRobinStatusFlags (CMP_Type * *base*) [inline], [static]

Parameters

base	ACMP peripheral base address.

Returns

Mask of channel input changed asserted flags. Available range is channel0:0x01 to channel7:0x80.

8.6.14 void ACMP_ClearRoundRobinStatusFlags (CMP_Type * *base,* uint32_t *mask*)

Parameters

base	ACMP peripheral base address.
mask	Mask of channel index. Available range is channel0:0x01 to channel7:0x80.

8.6.15 static uint32_t ACMP_GetRoundRobinResult (CMP_Type * *base*) [inline], [static]

Note that the set-pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So ACMP_GetRoundRobinResult() can't return the same value as the value are set by ACMP_SetRoundRobinPreState.

Parameters

base	ACMP peripheral base address.

Returns

Mask of round robin channel result. Available range is channel0:0x01 to channel7:0x80.

8.6.16 void ACMP_EnableInterrupts (CMP_Type * *base,* uint32_t *mask*)

Parameters

base	ACMP peripheral base address.
mask	Interrupts mask. See "_acmp_interrupt_enable".

8.6.17 void ACMP_DisableInterrupts (CMP_Type ∗ *base,* uint32_t *mask*)

Parameters

base	ACMP peripheral base address.
mask	Interrupts mask. See "_acmp_interrupt_enable".

8.6.18 uint32_t ACMP_GetStatusFlags (CMP_Type ∗ *base*)

Parameters

base	ACMP peripheral base address.

Returns

Status flags asserted mask. See "_acmp_status_flags".

8.6.19 void ACMP_ClearStatusFlags (CMP_Type ∗ *base,* uint32_t *mask*)

Parameters

base	ACMP peripheral base address.
mask	Status flags mask. See "_acmp_status_flags".

8.6.20 void ACMP_SetDiscreteModeConfig (CMP_Type ∗ *base,* const acmp_discrete_mode_config_t ∗ *config*)

Configure the discrete mode when supporting 3V domain with 1.8V core.

Parameters

base	ACMP peripheral base address.
config	Pointer to configuration structure. See "acmp_discrete_mode_config_t".

8.6.21 void ACMP_GetDefaultDiscreteModeConfig (acmp_discrete_mode_config_t * *config*)

Parameters

config	Pointer to configuration structure to be restored with the setting values.

Chapter 9
CACHE: CACHE Memory Controller

9.1 Overview

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

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

2L. The unified cache driver API. This level provides many APIs for unified cache driver APIs for combined L1 and L2 cache maintain operations. This is provided for SDK drivers (DMA, ENET, 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 local driver APIs.

9.2 Function groups

9.2.1 CACHE Operation

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

Data Structures

- struct cache64_config_t
  CACHE64 configuration structure. More...

Macros

- #define CACHE64_LINESIZE_BYTE (FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE)
  cache line size.
- #define CACHE64_REGION_NUM (3U)
  cache region number.
- #define CACHE64_REGION_ALIGNMENT (0x400U)
  cache region alignment.

Enumerations

- enum cache64_policy_t {
  kCACHE64_PolicyNonCacheable = 0,
  kCACHE64_PolicyWriteThrough = 1,
  kCACHE64_PolicyWriteBack = 2 }
  Level 2 cache controller way size.

Driver version

- #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  cache driver version 2.0.4.

cache control for cache64

- uint32_t CACHE64_GetInstance (CACHE64_POLSEL_Type ∗base)
  Returns an instance number given periphearl base address.
- uint32_t CACHE64_GetInstanceByAddr (uint32_t address)
  brief Returns an instance number given physical memory address.
- status_t CACHE64_Init (CACHE64_POLSEL_Type ∗base, const cache64_config_t ∗config)
  Initializes an CACHE64 instance with the user configuration structure.
- void CACHE64_GetDefaultConfig (cache64_config_t ∗config)
  Gets the default configuration structure.
- void CACHE64_EnableCache (CACHE64_CTRL_Type ∗base)
  Enables the cache.
- void CACHE64_DisableCache (CACHE64_CTRL_Type ∗base)
  Disables the cache.
- void CACHE64_InvalidateCache (CACHE64_CTRL_Type ∗base)
  Invalidates the cache.
- void CACHE64_InvalidateCacheByRange (uint32_t address, uint32_t size_byte)
  Invalidates cache by range.
- void CACHE64_CleanCache (CACHE64_CTRL_Type ∗base)
  Cleans the cache.
- void CACHE64_CleanCacheByRange (uint32_t address, uint32_t size_byte)
  Cleans cache by range.
- void CACHE64_CleanInvalidateCache (CACHE64_CTRL_Type ∗base)
  Cleans and invalidates the cache.
- void CACHE64_CleanInvalidateCacheByRange (uint32_t address, uint32_t size_byte)
  Cleans and invalidate cache by range.
- void CACHE64_EnableWriteBuffer (CACHE64_CTRL_Type ∗base, bool enable)
  Enables/disables the write buffer.

Unified Cache Control for all caches

- static void ICACHE_InvalidateByRange (uint32_t address, uint32_t size_byte)
  Invalidates instruction cache by range.
- static void DCACHE_InvalidateByRange (uint32_t address, uint32_t size_byte)
  Invalidates data cache by range.
- static void DCACHE_CleanByRange (uint32_t address, uint32_t size_byte)
  Clean data cache by range.
- static void DCACHE_CleanInvalidateByRange (uint32_t address, uint32_t size_byte)
  Cleans and Invalidates data cache by range.

9.3 Data Structure Documentation

9.3.1 struct cache64_config_t

Data Fields

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

Field Documentation

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

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

9.4 Macro Definition Documentation

9.4.1 #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

9.4.2 #define CACHE64_LINESIZE_BYTE (FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE)

9.4.3 #define CACHE64_REGION_NUM (3U)

9.4.4 #define CACHE64_REGION_ALIGNMENT (0x400U)

9.5 Enumeration Type Documentation

9.5.1 enum cache64_policy_t

Enumerator

kCACHE64_PolicyNonCacheable Non-cacheable.
kCACHE64_PolicyWriteThrough Write through.
kCACHE64_PolicyWriteBack Write back.

9.6 Function Documentation

9.6.1 uint32_t CACHE64_GetInstance (CACHE64_POLSEL_Type ∗ *base*)

Parameters

base	The peripheral base address.

Returns

CACHE64_POLSEL instance number starting from 0.

9.6.2 uint32_t CACHE64_GetInstanceByAddr (uint32_t *address*)

param address The physical memory address.

Returns

CACHE64_CTRL instance number starting from 0.

9.6.3 status_t CACHE64_Init (CACHE64_POLSEL_Type ∗ *base,* const cache64_config_t ∗ *config*)

This function configures the CACHE64 module with user-defined settings. Call the CACHE64_GetDefaultConfig() function to configure the configuration structure and get the default configuration.

Parameters

base	CACHE64_POLSEL peripheral base address.
config	Pointer to a user-defined configuration structure.

Return values

kStatus_Success	CACHE64 initialize succeed

9.6.4 void CACHE64_GetDefaultConfig (cache64_config_t ∗ *config*)

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

Parameters

config	Pointer to a configuration structure.

9.6.5 void CACHE64_EnableCache (CACHE64_CTRL_Type ∗ *base*)

Parameters

base	CACHE64_CTRL peripheral base address.

9.6.6 void CACHE64_DisableCache (CACHE64_CTRL_Type ∗ *base*)

Parameters

base	CACHE64_CTRL peripheral base address.

9.6.7 void CACHE64_InvalidateCache (CACHE64_CTRL_Type ∗ *base*)

Parameters

base	CACHE64_CTRL peripheral base address.

9.6.8 void CACHE64_InvalidateCacheByRange (uint32_t *address,* uint32_t *size_byte*)

Parameters

address	The physical address of cache.
size_byte	size of the memory to be invalidated.

Note

Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

9.6.9 void CACHE64_CleanCache (CACHE64_CTRL_Type * *base*)

Parameters

base	CACHE64_CTRL peripheral base address.

9.6.10 void CACHE64_CleanCacheByRange (uint32_t *address,* uint32_t *size_byte*)

Parameters

address	The physical address of cache.
size_byte	size of the memory to be cleaned.

Note

Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

9.6.11 void CACHE64_CleanInvalidateCache (CACHE64_CTRL_Type ∗ *base*)

Parameters

base	CACHE64_CTRL peripheral base address.

9.6.12 void CACHE64_CleanInvalidateCacheByRange (uint32_t *address,* uint32_t *size_byte*)

Parameters

address	The physical address of cache.
size_byte	size of the memory to be Cleaned and Invalidated.

Note

Address and size should be aligned to "CACHE64_LINESIZE_BYTE". The startAddr here will be forced to align to CACHE64_LINESIZE_BYTE if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

9.6.13 void CACHE64_EnableWriteBuffer (CACHE64_CTRL_Type * *base,* bool *enable*)

Parameters

base	CACHE64_CTRL peripheral base address.
enable	The enable or disable flag. true - enable the write buffer. false - disable the write buffer.

9.6.14 static void ICACHE_InvalidateByRange (uint32_t *address,* uint32_t *size_byte*) [inline], [static]

Parameters

address	The physical address.
size_byte	size of the memory to be invalidated.

Note

Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

9.6.15 static void DCACHE_InvalidateByRange (uint32_t *address,* uint32_t *size_byte*) [inline], [static]

Parameters

address	The physical address.
size_byte	size of the memory to be invalidated.

Note

Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

9.6.16 static void DCACHE_CleanByRange (uint32_t *address,* uint32_t *size_byte*) [inline], [static]

Parameters

address	The physical address.
size_byte	size of the memory to be cleaned.

Note

Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

9.6.17 static void DCACHE_CleanInvalidateByRange (uint32_t *address,* uint32_t *size_byte*) `[inline]`,`[static]`

Parameters

address	The physical address.
size_byte	size of the memory to be Cleaned and Invalidated.

Note

Address and size should be aligned to CACHE64_LINESIZE_BYTE due to the cache operation unit FSL_FEATURE_CACHE64_CTRL_LINESIZE_BYTE. The startAddr here will be forced to align to the cache line size if startAddr is not aligned. For the size_byte, application should make sure the alignment or make sure the right operation order if the size_byte is not aligned.

Chapter 10
CASPER: The Cryptographic Accelerator and Signal Processing Engine with RAM sharing

10.1 Overview

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

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

10.2 CASPER Driver Initialization and deinitialization

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

10.3 Comments about API usage in RTOS

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

10.4 Comments about API usage in interrupt handler

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

10.5 CASPER Driver Examples

10.5.1 Simple examples

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

Modules

- casper_driver
- casper_driver_pkha

10.6 casper_driver

10.6.1 Overview

Enumerations

- enum casper_operation_t { ,
  kCASPER_OpMul6464Sum,
  kCASPER_OpMul6464FullSum,
  kCASPER_OpMul6464Reduce,
  kCASPER_OpAdd64 = 0x08,
  kCASPER_OpSub64 = 0x09,
  kCASPER_OpDouble64 = 0x0A,
  kCASPER_OpXor64 = 0x0B,
  kCASPER_OpRSub64 = 0x0C,
  kCASPER_OpShiftLeft32,
  kCASPER_OpShiftRight32 = 0x11,
  kCASPER_OpCopy = 0x14,
  kCASPER_OpRemask = 0x15,
  kCASPER_OpFill = 0x16,
  kCASPER_OpZero = 0x17,
  kCASPER_OpCompare = 0x18,
  kCASPER_OpCompareFast = 0x19 }
  CASPER operation.
- enum casper_algo_t {
  kCASPER_ECC_P256 = 0x01,
  kCASPER_ECC_P384 = 0x02,
  kCASPER_ECC_P521 = 0x03 }
  Algorithm used for CASPER operation.

Functions

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

Driver version

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

10.6.2 Macro Definition Documentation

10.6.2.1 #define FSL_CASPER_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))

Version 2.2.3.

Current version: 2.2.3

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Bug fix KPSDK-24531 double_scalar_multiplication() result may be all zeroes for some specific input
- Version 2.0.2
  - Bug fix KPSDK-25015 CASPER_MEMCPY hard-fault on LPC55xx when both source and destination buffers are outside of CASPER_RAM
- Version 2.0.3
  - Bug fix KPSDK-28107 RSUB, FILL and ZERO operations not implemented in enum _casper-_operation.
- Version 2.0.4
  - For GCC compiler, enforce O1 optimize level, specifically to remove strict-aliasing option. This driver is very specific and requires -fno-strict-aliasing.
- Version 2.0.5
  - Fix sign-compare warning.
- Version 2.0.6
  - Fix IAR Pa082 warning.
- Version 2.0.7
  - Fix MISRA-C 2012 issue.
- Version 2.0.8
  - Add feature macro for CASPER_RAM_OFFSET.
- Version 2.0.9
  - Remove unused function Jac_oncurve().
  - Fix ECC384 build.
- Version 2.0.10
  - Fix MISRA-C 2012 issue.
- Version 2.1.0
  - Add ECC NIST P-521 elliptic curve.
- Version 2.2.0
  - Rework driver to support multiple curves at once.
- Version 2.2.1
  - Fix MISRA-C 2012 issue.
- Version 2.2.2
  - Enable hardware interleaving to RAMX0 and RAMX1 for CASPER by feature macro FSL_-FEATURE_CASPER_RAM_HW_INTERLEAVE
- Version 2.2.3
  - Fix MISRA-C 2012 issues.

10.6.3 Enumeration Type Documentation

10.6.3.1 enum casper_operation_t

Enumerator

kCASPER_OpMul6464Sum Walking 1 or more of J loop, doing r=a∗b using 64x64=128.
kCASPER_OpMul6464FullSum Walking 1 or more of J loop, doing c,r=r+a∗b using 64x64=128, but assume inner j loop.
kCASPER_OpMul6464Reduce Walking 1 or more of J loop, doing c,r=r+a∗b using 64x64=128, but sum all of w.
kCASPER_OpAdd64 Walking 1 or more of J loop, doing c,r[-1]=r+a∗b using 64x64=128, but skip 1st write.
kCASPER_OpSub64 Walking add with off_AB, and in/out off_RES doing c,r=r+a+c using 64+64=65.
kCASPER_OpDouble64 Walking subtract with off_AB, and in/out off_RES doing r=r-a using 64-64=64, with last borrow implicit if any.
kCASPER_OpXor64 Walking add to self with off_RES doing c,r=r+r+c using 64+64=65.
kCASPER_OpRSub64 Walking XOR with off_AB, and in/out off_RES doing r=r^a using 64^64=64.
kCASPER_OpShiftLeft32 Walking subtract with off_AB, and in/out off_RES using r=a-r.
kCASPER_OpShiftRight32 Walking shift left doing r1,r=(b∗D)|r1, where D is 2^amt and is loaded by app (off_CD not used)
kCASPER_OpCopy Walking shift right doing r,r1=(b∗D)|r1, where D is 2^(32-amt) and is loaded by app (off_CD not used) and off_RES starts at MSW.
kCASPER_OpRemask Copy from ABoff to resoff, 64b at a time.
kCASPER_OpFill Copy and mask from ABoff to resoff, 64b at a time.
kCASPER_OpZero Fill RESOFF using 64 bits at a time with value in A and B.
kCASPER_OpCompare Fill RESOFF using 64 bits at a time of 0s.
kCASPER_OpCompareFast Compare two arrays, running all the way to the end.

10.6.3.2 enum casper_algo_t

Enumerator

kCASPER_ECC_P256 ECC_P256.
kCASPER_ECC_P384 ECC_P384.
kCASPER_ECC_P521 ECC_P521.

10.6.4 Function Documentation

10.6.4.1 void CASPER_Init (CASPER_Type ∗ *base*)

Enable clock and disable reset for CASPER.

Parameters

base	CASPER base address

10.6.4.2 void CASPER_Deinit (CASPER_Type * *base*)

Disable clock and enable reset.

Parameters

base	CASPER base address

10.7 casper_driver_pkha

10.7.1 Overview

Functions

- void CASPER_ModExp (CASPER_Type ∗base, const uint8_t ∗signature, const uint8_t ∗pubN, size_t wordLen, uint32_t pubE, uint8_t ∗plaintext)
  Performs modular exponentiation - (A^E) mod N.
- void CASPER_ecc_init (casper_algo_t curve)
  Initialize prime modulus mod in Casper memory .
- void CASPER_ECC_SECP256R1_Mul (CASPER_Type ∗base, uint32_t resX[8], uint32_t resY[8], uint32_t X[8], uint32_t Y[8], uint32_t scalar[8])
  Performs ECC secp256r1 point single scalar multiplication.
- void CASPER_ECC_SECP256R1_MulAdd (CASPER_Type ∗base, uint32_t resX[8], uint32_t resY[8], uint32_t X1[8], uint32_t Y1[8], uint32_t scalar1[8], uint32_t X2[8], uint32_t Y2[8], uint32_t scalar2[8])
  Performs ECC secp256r1 point double scalar multiplication.
- void CASPER_ECC_SECP384R1_Mul (CASPER_Type ∗base, uint32_t resX[12], uint32_t resY[12], uint32_t X[12], uint32_t Y[12], uint32_t scalar[12])
  Performs ECC secp384r1 point single scalar multiplication.
- void CASPER_ECC_SECP384R1_MulAdd (CASPER_Type ∗base, uint32_t resX[12], uint32_t resY[12], uint32_t X1[12], uint32_t Y1[12], uint32_t scalar1[12], uint32_t X2[12], uint32_t Y2[12], uint32_t scalar2[12])
  Performs ECC secp384r1 point double scalar multiplication.
- void CASPER_ECC_SECP521R1_Mul (CASPER_Type ∗base, uint32_t resX[18], uint32_t resY[18], uint32_t X[18], uint32_t Y[18], uint32_t scalar[18])
  Performs ECC secp521r1 point single scalar multiplication.
- void CASPER_ECC_SECP521R1_MulAdd (CASPER_Type ∗base, uint32_t resX[18], uint32_t resY[18], uint32_t X1[18], uint32_t Y1[18], uint32_t scalar1[18], uint32_t X2[18], uint32_t Y2[18], uint32_t scalar2[18])
  Performs ECC secp521r1 point double scalar multiplication.

10.7.2 Function Documentation

10.7.2.1 void CASPER_ModExp (CASPER_Type ∗ *base,* const uint8_t ∗ *signature,* const uint8_t ∗ *pubN,* size_t *wordLen,* uint32_t *pubE,* uint8_t ∗ *plaintext*)

This function performs modular exponentiation.

Parameters

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

10.7.2.2 void CASPER_ecc_init (casper_algo_t *curve*)

Set the prime modulus mod in Casper memory and set N_wordlen according to selected algorithm.

Parameters

curve	elliptic curve algoritm

10.7.2.3 void CASPER_ECC_SECP256R1_Mul (CASPER_Type ∗ *base,* uint32_t *resX[8],* uint32_t *resY[8],* uint32_t *X[8],* uint32_t *Y[8],* uint32_t *scalar[8]*)

This function performs ECC secp256r1 point single scalar multiplication [resX; resY] = scalar ∗ [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate in normal form, little endian.
out	*resY*	Output Y affine coordinate in normal form, little endian.
	X	Input X affine coordinate in normal form, little endian.
	Y	Input Y affine coordinate in normal form, little endian.
	scalar	Input scalar integer, in normal form, little endian.

10.7.2.4 void CASPER_ECC_SECP256R1_MulAdd (CASPER_Type ∗ *base,* uint32_t *resX[8],* uint32_t *resY[8],* uint32_t *X1[8],* uint32_t *Y1[8],* uint32_t *scalar1[8],* uint32_t *X2[8],* uint32_t *Y2[8],* uint32_t *scalar2[8]*)

This function performs ECC secp256r1 point double scalar multiplication [resX; resY] = scalar1 ∗ [X1; Y1] + scalar2 ∗ [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All

arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate.
out	*resY*	Output Y affine coordinate.
	X1	Input X1 affine coordinate.
	Y1	Input Y1 affine coordinate.
	scalar1	Input scalar1 integer.
	X2	Input X2 affine coordinate.
	Y2	Input Y2 affine coordinate.
	scalar2	Input scalar2 integer.

10.7.2.5 void CASPER_ECC_SECP384R1_Mul (CASPER_Type ∗ *base,* uint32_t *resX[12],* uint32_t *resY[12],* uint32_t *X[12],* uint32_t *Y[12],* uint32_t *scalar[12]*)

This function performs ECC secp384r1 point single scalar multiplication [resX; resY] = scalar ∗ [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate in normal form, little endian.
out	*resY*	Output Y affine coordinate in normal form, little endian.
	X	Input X affine coordinate in normal form, little endian.
	Y	Input Y affine coordinate in normal form, little endian.
	scalar	Input scalar integer, in normal form, little endian.

10.7.2.6 void CASPER_ECC_SECP384R1_MulAdd (CASPER_Type ∗ *base,* uint32_t *resX[12],* uint32_t *resY[12],* uint32_t *X1[12],* uint32_t *Y1[12],* uint32_t *scalar1[12],* uint32_t *X2[12],* uint32_t *Y2[12],* uint32_t *scalar2[12]*)

This function performs ECC secp384r1 point double scalar multiplication [resX; resY] = scalar1 ∗ [X1; Y1] + scalar2 ∗ [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate.
out	*resY*	Output Y affine coordinate.
	X1	Input X1 affine coordinate.
	Y1	Input Y1 affine coordinate.
	scalar1	Input scalar1 integer.
	X2	Input X2 affine coordinate.
	Y2	Input Y2 affine coordinate.
	scalar2	Input scalar2 integer.

10.7.2.7 void CASPER_ECC_SECP521R1_Mul (CASPER_Type ∗ *base,* uint32_t *resX[18],* uint32_t *resY[18],* uint32_t *X[18],* uint32_t *Y[18],* uint32_t *scalar[18]*)

This function performs ECC secp521r1 point single scalar multiplication [resX; resY] = scalar ∗ [X; Y] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
out	*resX*	Output X affine coordinate in normal form, little endian.
out	*resY*	Output Y affine coordinate in normal form, little endian.
	X	Input X affine coordinate in normal form, little endian.
	Y	Input Y affine coordinate in normal form, little endian.
	scalar	Input scalar integer, in normal form, little endian.

10.7.2.8 void CASPER_ECC_SECP521R1_MulAdd (CASPER_Type ∗ *base,* uint32_t *resX[18],* uint32_t *resY[18],* uint32_t *X1[18],* uint32_t *Y1[18],* uint32_t *scalar1[18],* uint32_t *X2[18],* uint32_t *Y2[18],* uint32_t *scalar2[18]*)

This function performs ECC secp521r1 point double scalar multiplication [resX; resY] = scalar1 ∗ [X1; Y1] + scalar2 ∗ [X2; Y2] Coordinates are affine in normal form, little endian. Scalars are little endian. All arrays are little endian byte arrays, uint32_t type is used only to enforce the 32-bit alignment (0-mod-4 address).

Parameters

	base	CASPER base address
`out`	*resX*	Output X affine coordinate.
`out`	*resY*	Output Y affine coordinate.
	X1	Input X1 affine coordinate.
	Y1	Input Y1 affine coordinate.
	scalar1	Input scalar1 integer.
	X2	Input X2 affine coordinate.
	Y2	Input Y2 affine coordinate.
	scalar2	Input scalar2 integer.

Chapter 11
Common Driver

11.1 Overview

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

Macros

- #define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1
  Macro to use the default weak IRQ handler in drivers.
- #define MAKE_STATUS(group, code) ((((group)∗100L) + (code)))
  Construct a status code value from a group and code number.
- #define MAKE_VERSION(major, minor, bugfix) (((major) ∗ 65536L) + ((minor) ∗ 256L) + (bugfix))
  Construct the version number for drivers.
- #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U
  No debug console.
- #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U
  Debug console based on UART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U
  Debug console based on LPUART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U
  Debug console based on LPSCI.
- #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U
  Debug console based on USBCDC.
- #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U
  Debug console based on FLEXCOMM.
- #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U
  Debug console based on i.MX UART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U
  Debug console based on LPC_VUSART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U
  Debug console based on LPC_USART.
- #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U
  Debug console based on SWO.
- #define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U
  Debug console based on QSCI.
- #define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
  Computes the number of elements in an array.

Typedefs

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

Enumerations

- enum _status_groups {
  kStatusGroup_Generic = 0,
  kStatusGroup_FLASH = 1,
  kStatusGroup_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_USART = 57,
  kStatusGroup_DMIC = 58,
  kStatusGroup_SDIF = 59,

kStatusGroup_POWER_MANAGER = 159 }
Status group numbers.

- enum {
kStatus_Success = MAKE_STATUS(kStatusGroup_Generic, 0),
kStatus_Fail = MAKE_STATUS(kStatusGroup_Generic, 1),
kStatus_ReadOnly = MAKE_STATUS(kStatusGroup_Generic, 2),
kStatus_OutOfRange = MAKE_STATUS(kStatusGroup_Generic, 3),
kStatus_InvalidArgument = MAKE_STATUS(kStatusGroup_Generic, 4),
kStatus_Timeout = MAKE_STATUS(kStatusGroup_Generic, 5),
kStatus_NoTransferInProgress,
kStatus_Busy = MAKE_STATUS(kStatusGroup_Generic, 7),
kStatus_NoData }
Generic status return codes.

Functions

- void ∗ SDK_Malloc (size_t size, size_t alignbytes)
Allocate memory with given alignment and aligned size.
- void SDK_Free (void ∗ptr)
Free memory.
- void SDK_DelayAtLeastUs (uint32_t delayTime_us, uint32_t coreClock_Hz)
Delay at least for some time.

Driver version

- #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))
common driver version.

Min/max macros

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

UINT16_MAX/UINT32_MAX value

- #define **UINT16_MAX** ((uint16_t)-1)
- #define **UINT32_MAX** ((uint32_t)-1)

Suppress fallthrough warning macro

- #define **SUPPRESS_FALL_THROUGH_WARNING**()

11.2 Macro Definition Documentation

11.2.1 #define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1

11.2.2 #define MAKE_STATUS(*group, code*) ((((group)∗100L) + (code)))

11.2.3 #define MAKE_VERSION(*major, minor, bugfix*) (((major) ∗ 65536L) + ((minor) ∗ 256L) + (bugfix))

The driver version is a 32-bit number, for both 32-bit platforms(such as Cortex M) and 16-bit platforms(such as DSC).

```
| Unused    || Major Version || Minor Version ||  Bug Fix    |
31        25  24           17  16            9  8            0
```

11.2.4 #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))

11.2.5 #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

11.2.6 #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

11.2.7 #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

11.2.8 #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

11.2.9 #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

11.2.10 #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

11.2.11 #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

11.2.12 #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

11.2.13 #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

11.2.14 #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

11.2.15 #define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U

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

11.3 Typedef Documentation

11.3.1 typedef int32_t status_t

11.4 Enumeration Type Documentation

11.4.1 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 UART 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_TOUCH_PANEL Group number for touch panel status codes.
kStatusGroup_HAL_GPIO Group number for HAL GPIO status codes.
kStatusGroup_HAL_UART Group number for HAL UART status codes.
kStatusGroup_HAL_TIMER Group number for HAL TIMER status codes.
kStatusGroup_HAL_SPI Group number for HAL SPI status codes.
kStatusGroup_HAL_I2C Group number for HAL I2C status codes.
kStatusGroup_HAL_FLASH Group number for HAL FLASH status codes.
kStatusGroup_HAL_PWM Group number for HAL PWM status codes.
kStatusGroup_HAL_RNG Group number for HAL RNG status codes.

kStatusGroup_HAL_I2S Group number for HAL I2S status codes.
kStatusGroup_TIMERMANAGER Group number for TiMER MANAGER status codes.
kStatusGroup_SERIALMANAGER Group number for SERIAL MANAGER status codes.
kStatusGroup_LED Group number for LED status codes.
kStatusGroup_BUTTON Group number for BUTTON status codes.
kStatusGroup_EXTERN_EEPROM Group number for EXTERN EEPROM status codes.
kStatusGroup_SHELL Group number for SHELL status codes.
kStatusGroup_MEM_MANAGER Group number for MEM MANAGER status codes.
kStatusGroup_LIST Group number for List status codes.
kStatusGroup_OSA Group number for OSA status codes.
kStatusGroup_COMMON_TASK Group number for Common task status codes.
kStatusGroup_MSG Group number for messaging status codes.
kStatusGroup_SDK_OCOTP Group number for OCOTP status codes.
kStatusGroup_SDK_FLEXSPINOR Group number for FLEXSPINOR status codes.
kStatusGroup_CODEC Group number for codec status codes.
kStatusGroup_ASRC Group number for codec status ASRC.
kStatusGroup_OTFAD Group number for codec status codes.
kStatusGroup_SDIOSLV Group number for SDIOSLV status codes.
kStatusGroup_MECC Group number for MECC status codes.
kStatusGroup_ENET_QOS Group number for ENET_QOS status codes.
kStatusGroup_LOG Group number for LOG status codes.
kStatusGroup_I3CBUS Group number for I3CBUS status codes.
kStatusGroup_QSCI Group number for QSCI status codes.
kStatusGroup_SNT Group number for SNT status codes.
kStatusGroup_QUEUEDSPI Group number for QSPI status codes.
kStatusGroup_POWER_MANAGER Group number for POWER_MANAGER status codes.

11.4.2 anonymous enum

Enumerator

kStatus_Success Generic status for Success.
kStatus_Fail Generic status for Fail.
kStatus_ReadOnly Generic status for read only failure.
kStatus_OutOfRange Generic status for out of range access.
kStatus_InvalidArgument Generic status for invalid argument check.
kStatus_Timeout Generic status for timeout.
kStatus_NoTransferInProgress Generic status for no transfer in progress.
kStatus_Busy Generic status for module is busy.
kStatus_NoData Generic status for no data is found for the operation.

11.5 Function Documentation

11.5.1 void∗ SDK_Malloc (size_t *size,* size_t *alignbytes*)

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

Parameters

size	The length required to malloc.
alignbytes	The alignment size.

Return values

The	allocated memory.

11.5.2 void SDK_Free (void ∗ *ptr*)

Parameters

ptr	The memory to be release.

11.5.3 void SDK_DelayAtLeastUs (uint32_t *delayTime_us,* uint32_t *coreClock_Hz*)

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

Parameters

delayTime_us	Delay time in unit of microsecond.
coreClock_Hz	Core clock frequency with Hz.

Chapter 12
CTIMER: Standard counter/timers

12.1 Overview

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

12.2 Function groups

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

12.2.1 Initialization and deinitialization

The function CTIMER_Init() initializes the cTimer with specified configurations. The function CTIMER_GetDefaultConfig() gets the default configurations. The initialization function configures the counter/timer mode and input selection when running in counter mode.

The function CTIMER_Deinit() stops the timer and turns off the module clock.

12.2.2 PWM Operations

The function CTIMER_SetupPwm() sets up channels for PWM output. Each channel has its own duty cycle, however the same PWM period is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 0=inactive signal(0% duty cycle) and 100=always active signal (100% duty cycle).

The function CTIMER_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular channel.

12.2.3 Match Operation

The function CTIMER_SetupMatch() sets up channels for match operation. Each channel is configured with a match value: if the counter should stop on match, if counter should reset on match, and output pin action. The output signal can be cleared, set, or toggled on match.

12.2.4 Input capture operations

The function CTIMER_SetupCapture() sets up an channel for input capture. The user can specify the capture edge and if a interrupt should be generated when processing the input signal.

12.3 Typical use case

12.3.1 Match example

Set up a match channel to toggle output when a match occurs. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ctimer

12.3.2 PWM output example

Set up a channel for PWM output. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/ctimer

Files

- file fsl_ctimer.h

Data Structures

- struct ctimer_match_config_t
  Match configuration. More...
- struct ctimer_config_t
  Timer configuration structure. More...

Enumerations

- enum ctimer_capture_channel_t {
  kCTIMER_Capture_0 = 0U,
  kCTIMER_Capture_1,
  kCTIMER_Capture_2 }
  List of Timer capture channels.
- enum ctimer_capture_edge_t {
  kCTIMER_Capture_RiseEdge = 1U,
  kCTIMER_Capture_FallEdge = 2U,
  kCTIMER_Capture_BothEdge = 3U }
  List of capture edge options.
- enum ctimer_match_t {
  kCTIMER_Match_0 = 0U,
  kCTIMER_Match_1,
  kCTIMER_Match_2,
  kCTIMER_Match_3 }
  List of Timer match registers.
- enum ctimer_external_match_t {
  kCTIMER_External_Match_0 = (1U << 0),
  kCTIMER_External_Match_1 = (1U << 1),
  kCTIMER_External_Match_2 = (1U << 2),
  kCTIMER_External_Match_3 = (1U << 3) }

List of external match.

- enum ctimer_match_output_control_t {
  kCTIMER_Output_NoAction = 0U,
  kCTIMER_Output_Clear,
  kCTIMER_Output_Set,
  kCTIMER_Output_Toggle }
  List of output control options.
- enum ctimer_timer_mode_t
  List of Timer modes.
- enum ctimer_interrupt_enable_t {
  kCTIMER_Match0InterruptEnable = CTIMER_MCR_MR0I_MASK,
  kCTIMER_Match1InterruptEnable = CTIMER_MCR_MR1I_MASK,
  kCTIMER_Match2InterruptEnable = CTIMER_MCR_MR2I_MASK,
  kCTIMER_Match3InterruptEnable = CTIMER_MCR_MR3I_MASK,
  kCTIMER_Capture0InterruptEnable = CTIMER_CCR_CAP0I_MASK,
  kCTIMER_Capture1InterruptEnable = CTIMER_CCR_CAP1I_MASK,
  kCTIMER_Capture2InterruptEnable = CTIMER_CCR_CAP2I_MASK }
  List of Timer interrupts.
- enum ctimer_status_flags_t {
  kCTIMER_Match0Flag = CTIMER_IR_MR0INT_MASK,
  kCTIMER_Match1Flag = CTIMER_IR_MR1INT_MASK,
  kCTIMER_Match2Flag = CTIMER_IR_MR2INT_MASK,
  kCTIMER_Match3Flag = CTIMER_IR_MR3INT_MASK,
  kCTIMER_Capture0Flag = CTIMER_IR_CR0INT_MASK,
  kCTIMER_Capture1Flag = CTIMER_IR_CR1INT_MASK,
  kCTIMER_Capture2Flag = CTIMER_IR_CR2INT_MASK }
  List of Timer flags.
- enum ctimer_callback_type_t {
  kCTIMER_SingleCallback,
  kCTIMER_MultipleCallback }
  Callback type when registering for a callback.

Functions

- void CTIMER_SetupMatch (CTIMER_Type ∗base, ctimer_match_t matchChannel, const ctimer_-match_config_t ∗config)
  Setup the match register.
- uint32_t CTIMER_GetOutputMatchStatus (CTIMER_Type ∗base, uint32_t matchChannel)
  Get the status of output match.
- void CTIMER_SetupCapture (CTIMER_Type ∗base, ctimer_capture_channel_t capture, ctimer_-capture_edge_t edge, bool enableInt)
  Setup the capture.
- static uint32_t CTIMER_GetTimerCountValue (CTIMER_Type ∗base)
  Get the timer count value from TC register.
- void CTIMER_RegisterCallBack (CTIMER_Type ∗base, ctimer_callback_t ∗cb_func, ctimer_-callback_type_t cb_type)
  Register callback.
- static void CTIMER_Reset (CTIMER_Type ∗base)

Reset the counter.

Driver version

- #define FSL_CTIMER_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
  Version 2.2.1.

Initialization and deinitialization

- void CTIMER_Init (CTIMER_Type ∗base, const ctimer_config_t ∗config)
  Ungates the clock and configures the peripheral for basic operation.
- void CTIMER_Deinit (CTIMER_Type ∗base)
  Gates the timer clock.
- void CTIMER_GetDefaultConfig (ctimer_config_t ∗config)
  Fills in the timers configuration structure with the default settings.

PWM setup operations

- status_t CTIMER_SetupPwmPeriod (CTIMER_Type ∗base, const ctimer_match_t pwmPeriodChannel, ctimer_match_t matchChannel, uint32_t pwmPeriod, uint32_t pulsePeriod, bool enableInt)
  Configures the PWM signal parameters.
- status_t CTIMER_SetupPwm (CTIMER_Type ∗base, const ctimer_match_t pwmPeriodChannel, ctimer_match_t matchChannel, uint8_t dutyCyclePercent, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, bool enableInt)
  Configures the PWM signal parameters.
- static void CTIMER_UpdatePwmPulsePeriod (CTIMER_Type ∗base, ctimer_match_t matchChannel, uint32_t pulsePeriod)
  Updates the pulse period of an active PWM signal.
- void CTIMER_UpdatePwmDutycycle (CTIMER_Type ∗base, const ctimer_match_t pwmPeriodChannel, ctimer_match_t matchChannel, uint8_t dutyCyclePercent)
  Updates the duty cycle of an active PWM signal.

Interrupt Interface

- static void CTIMER_EnableInterrupts (CTIMER_Type ∗base, uint32_t mask)
  Enables the selected Timer interrupts.
- static void CTIMER_DisableInterrupts (CTIMER_Type ∗base, uint32_t mask)
  Disables the selected Timer interrupts.
- static uint32_t CTIMER_GetEnabledInterrupts (CTIMER_Type ∗base)
  Gets the enabled Timer interrupts.

Status Interface

- static uint32_t CTIMER_GetStatusFlags (CTIMER_Type ∗base)
  Gets the Timer status flags.
- static void CTIMER_ClearStatusFlags (CTIMER_Type ∗base, uint32_t mask)
  Clears the Timer status flags.

Counter Start and Stop

- static void CTIMER_StartTimer (CTIMER_Type *base)
  Starts the Timer counter.
- static void CTIMER_StopTimer (CTIMER_Type *base)
  Stops the Timer counter.

12.4 Data Structure Documentation

12.4.1 struct ctimer_match_config_t

This structure holds the configuration settings for each match register.

Data Fields

- uint32_t matchValue
  This is stored in the match register.
- bool enableCounterReset
  true: Match will reset the counter false: Match will not reser the counter
- bool enableCounterStop
  true: Match will stop the counter false: Match will not stop the counter
- ctimer_match_output_control_t outControl
  Action to be taken on a match on the EM bit/output.
- bool outPinInitState
  Initial value of the EM bit/output.
- bool enableInterrupt
  true: Generate interrupt upon match false: Do not generate interrupt on match

12.4.2 struct ctimer_config_t

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the CTIMER_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Data Fields

- ctimer_timer_mode_t mode
  Timer mode.
- ctimer_capture_channel_t input
  Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC.
- uint32_t prescale
  Prescale value.

12.5 Enumeration Type Documentation

12.5.1 enum ctimer_capture_channel_t

Enumerator

kCTIMER_Capture_0 Timer capture channel 0.
kCTIMER_Capture_1 Timer capture channel 1.
kCTIMER_Capture_2 Timer capture channel 2.

12.5.2 enum ctimer_capture_edge_t

Enumerator

kCTIMER_Capture_RiseEdge Capture on rising edge.
kCTIMER_Capture_FallEdge Capture on falling edge.
kCTIMER_Capture_BothEdge Capture on rising and falling edge.

12.5.3 enum ctimer_match_t

Enumerator

kCTIMER_Match_0 Timer match register 0.
kCTIMER_Match_1 Timer match register 1.
kCTIMER_Match_2 Timer match register 2.
kCTIMER_Match_3 Timer match register 3.

12.5.4 enum ctimer_external_match_t

Enumerator

kCTIMER_External_Match_0 External match 0.
kCTIMER_External_Match_1 External match 1.
kCTIMER_External_Match_2 External match 2.
kCTIMER_External_Match_3 External match 3.

12.5.5 enum ctimer_match_output_control_t

Enumerator

kCTIMER_Output_NoAction No action is taken.

kCTIMER_Output_Clear Clear the EM bit/output to 0.
kCTIMER_Output_Set Set the EM bit/output to 1.
kCTIMER_Output_Toggle Toggle the EM bit/output.

12.5.6 enum ctimer_interrupt_enable_t

Enumerator

kCTIMER_Match0InterruptEnable Match 0 interrupt.
kCTIMER_Match1InterruptEnable Match 1 interrupt.
kCTIMER_Match2InterruptEnable Match 2 interrupt.
kCTIMER_Match3InterruptEnable Match 3 interrupt.
kCTIMER_Capture0InterruptEnable Capture 0 interrupt.
kCTIMER_Capture1InterruptEnable Capture 1 interrupt.
kCTIMER_Capture2InterruptEnable Capture 2 interrupt.

12.5.7 enum ctimer_status_flags_t

Enumerator

kCTIMER_Match0Flag Match 0 interrupt flag.
kCTIMER_Match1Flag Match 1 interrupt flag.
kCTIMER_Match2Flag Match 2 interrupt flag.
kCTIMER_Match3Flag Match 3 interrupt flag.
kCTIMER_Capture0Flag Capture 0 interrupt flag.
kCTIMER_Capture1Flag Capture 1 interrupt flag.
kCTIMER_Capture2Flag Capture 2 interrupt flag.

12.5.8 enum ctimer_callback_type_t

When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Enumerator

kCTIMER_SingleCallback Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently
kCTIMER_MultipleCallback Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

12.6 Function Documentation

12.6.1 void CTIMER_Init (CTIMER_Type * *base,* const ctimer_config_t * *config*)

Note

This API should be called at the beginning of the application before using the driver.

Parameters

base	Ctimer peripheral base address
config	Pointer to the user configuration structure.

12.6.2 void CTIMER_Deinit (CTIMER_Type * *base*)

Parameters

base	Ctimer peripheral base address

12.6.3 void CTIMER_GetDefaultConfig (ctimer_config_t * *config*)

The default values are:

```
*   config->mode = kCTIMER_TimerMode;
*   config->input = kCTIMER_Capture_0;
*   config->prescale = 0;
*
```

Parameters

config	Pointer to the user configuration structure.

12.6.4 status_t CTIMER_SetupPwmPeriod (CTIMER_Type * *base,* const ctimer_match_t *pwmPeriodChannel,* ctimer_match_t *matchChannel,* uint32_t *pwmPeriod,* uint32_t *pulsePeriod,* bool *enableInt*)

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

Note

When setting PWM output from multiple output pins, all should use the same PWM period

Parameters

base	Ctimer peripheral base address
pwmPeriod-Channel	Specify the channel to control the PWM period
matchChannel	Match pin to be used to output the PWM signal
pwmPeriod	PWM period match value
pulsePeriod	Pulse width match value
enableInt	Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

12.6.5 status_t CTIMER_SetupPwm (CTIMER_Type * *base,* const ctimer_match_t *pwmPeriodChannel,* ctimer_match_t *matchChannel,* uint8_t *dutyCyclePercent,* uint32_t *pwmFreq_Hz,* uint32_t *srcClock_Hz,* bool *enableInt*)

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function can manually assign the specified channel to set the PWM cycle.

Note

When setting PWM output from multiple output pins, all should use the same PWM frequency. Please use CTIMER_SetupPwmPeriod to set up the PWM with high resolution.

Parameters

base	Ctimer peripheral base address
pwmPeriod-Channel	Specify the channel to control the PWM period

matchChannel	Match pin to be used to output the PWM signal
dutyCycle-Percent	PWM pulse width; the value should be between 0 to 100
pwmFreq_Hz	PWM signal frequency in Hz
srcClock_Hz	Timer counter clock in Hz
enableInt	Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated.

12.6.6 static void CTIMER_UpdatePwmPulsePeriod (CTIMER_Type ∗ *base,* ctimer_match_t *matchChannel,* uint32_t *pulsePeriod*) `[inline], [static]`

Parameters

base	Ctimer peripheral base address
matchChannel	Match pin to be used to output the PWM signal
pulsePeriod	New PWM pulse width match value

12.6.7 void CTIMER_UpdatePwmDutycycle (CTIMER_Type ∗ *base,* const ctimer_match_t *pwmPeriodChannel,* ctimer_match_t *matchChannel,* uint8_t *dutyCyclePercent*)

Note

Please use CTIMER_SetupPwmPeriod to update the PWM with high resolution. This function can manually assign the specified channel to set the PWM cycle.

Parameters

base	Ctimer peripheral base address
pwmPeriod-Channel	Specify the channel to control the PWM period

matchChannel	Match pin to be used to output the PWM signal
dutyCycle-Percent	New PWM pulse width; the value should be between 0 to 100

12.6.8 void CTIMER_SetupMatch (CTIMER_Type * *base*, ctimer_match_t *matchChannel*, const ctimer_match_config_t * *config*)

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

base	Ctimer peripheral base address
matchChannel	Match register to configure
config	Pointer to the match configuration structure

12.6.9 uint32_t CTIMER_GetOutputMatchStatus (CTIMER_Type * *base*, uint32_t *matchChannel*)

This function gets the status of output MAT, whether or not this output is connected to a pin. This status is driven to the MAT pins if the match function is selected via IOCON. 0 = LOW. 1 = HIGH.

Parameters

base	Ctimer peripheral base address
matchChannel	External match channel, user can obtain the status of multiple match channels at the same time by using the logic of "|" enumeration ctimer_external_match_t

Returns

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

12.6.10 void CTIMER_SetupCapture (CTIMER_Type * *base*, ctimer_capture-_channel_t *capture*, ctimer_capture_edge_t *edge*, bool *enableInt*)

Parameters

base	Ctimer peripheral base address
capture	Capture channel to configure
edge	Edge on the channel that will trigger a capture
enableInt	Flag to enable channel interrupts, if enabled then the registered call back is called upon capture

12.6.11 static uint32_t CTIMER_GetTimerCountValue (CTIMER_Type ∗ *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address.

Returns

return the timer count value.

12.6.12 void CTIMER_RegisterCallBack (CTIMER_Type ∗ *base,* ctimer_callback_t ∗ *cb_func,* ctimer_callback_type_t *cb_type*)

Parameters

base	Ctimer peripheral base address
cb_func	callback function
cb_type	callback function type, singular or multiple

12.6.13 static void CTIMER_EnableInterrupts (CTIMER_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	Ctimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration ctimer_interrupt_enable_t

12.6.14 static void CTIMER_DisableInterrupts (CTIMER_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	Ctimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration ctimer_interrupt_enable_t

12.6.15 static uint32_t CTIMER_GetEnabledInterrupts (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration ctimer_interrupt_enable_t

12.6.16 static uint32_t CTIMER_GetStatusFlags (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration ctimer_status_flags_t

12.6.17 static void CTIMER_ClearStatusFlags (CTIMER_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	Ctimer peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration ctimer_status_flags_t

12.6.18 static void CTIMER_StartTimer (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

12.6.19 static void CTIMER_StopTimer (CTIMER_Type * *base*) [inline], [static]

Parameters

base	Ctimer peripheral base address

12.6.20 static void CTIMER_Reset (CTIMER_Type * *base*) [inline], [static]

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

Parameters

base	Ctimer peripheral base address

Chapter 13
DMIC: Digital Microphone

13.1 Overview

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

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

13.2 Function groups

13.2.1 Initialization and deinitialization

This function group implements DMIC initialization and deinitialization API. DMIC_Init() function Enables the clock to the DMIC register interface. DMIC_Dinit() function Disables the clock to the DMIC register interface.

13.2.2 Configuration

This function group implements DMIC configration API. DMIC_ConfigIO()function configures the use of PDM (Pulse Density moulation) pins. DMIC_SetOperationMode()function configures the mode of operation either in DMA or in interrupt. DMIC_ConfigChannel() function configures the various property of a DMIC channel. DMIC_Use2fs()function configures the clock scaling used for PCM data output. DMIC_EnableChannnel() function enables a particualr DMIC channel. DMIC_FifoChannel() function configures FIFO settings for a DMIC channel.

13.2.3 DMIC Data and status

This function group implements the API to get data and status of DMIC FIFO. DMIC_FifoGetStatus() function gives the status of a DMIC FIFO. DMIC_ClearStatus() function clears the status of a DMIC FIFO. DMIC_FifoGetData() function gets data from a DMIC FIFO.

13.2.4 DMIC Interrupt Functions

DMIC_EnablebleIntCallback() enables the interrupt for the selected DMIC peripheral. DMIC_Disable-IntCallback() disables the interrupt for the selected DMIC peripheral.

13.2.5 DMIC HWVAD Functions

This function group implements the API for HWVAD. DMIC_SetGainNoiseEstHwvad() Sets the gain value for the noise estimator. DMIC_SetGainSignalEstHwvad() Sets the gain value for the signal estimator. DMIC_SetFilterCtrlHwvad() Sets the HWVAD filter cutoff frequency parameter. DMIC_SetInputGainHwvad() Sets the input gain of HWVAD. DMIC_CtrlClrIntrHwvad() Clears HWVAD internal interrupt flag. DMIC_FilterResetHwvad() Resets HWVAD filters. DMIC_GetNoiseEnvlpEst() Gets the value from output of the filter z7.

13.2.6 DMIC HWVAD Interrupt Functions

DMIC_HwvadEnableIntCallback() enables the HWVAD interrupt for the selected DMIC peripheral. DMIC_HwvadDisableIntCallback() disables the HWVAD interrupt for the selected DMIC peripheral.

13.3 Typical use case

13.3.1 DMIC DMA Configuration

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

13.3.2 DMIC use case

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

Modules

- DMIC DMA Driver
- DMIC Driver

13.4 DMIC Driver

13.4.1 Overview

Files

- file fsl_dmic.h

Data Structures

- struct dmic_channel_config_t
  DMIC Channel configuration structure. More...

Typedefs

- typedef void(∗ dmic_callback_t)(void)
  DMIC Callback function.
- typedef void(∗ dmic_hwvad_callback_t)(void)
  HWVAD Callback function.

Enumerations

- enum {
  kStatus_DMIC_Busy = MAKE_STATUS(kStatusGroup_DMIC, 0),
  kStatus_DMIC_Idle = MAKE_STATUS(kStatusGroup_DMIC, 1),
  kStatus_DMIC_OverRunError = MAKE_STATUS(kStatusGroup_DMIC, 2),
  kStatus_DMIC_UnderRunError = MAKE_STATUS(kStatusGroup_DMIC, 3) }
  _dmic_status DMIC transfer status.
- enum operation_mode_t {
  kDMIC_OperationModeInterrupt = 1U,
  kDMIC_OperationModeDma = 2U }
  DMIC different operation modes.
- enum stereo_side_t {
  kDMIC_Left = 0U,
  kDMIC_Right = 1U }
  DMIC left/right values.
- enum pdm_div_t {

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

DMIC Clock pre-divider values.

- enum compensation_t {
  kDMIC_CompValueZero = 0U,
  kDMIC_CompValueNegativePoint16 = 1U,
  kDMIC_CompValueNegativePoint15 = 2U,
  kDMIC_CompValueNegativePoint13 = 3U }

  Pre-emphasis Filter coefficient value for 2FS and 4FS modes.
- enum dc_removal_t {
  kDMIC_DcNoRemove = 0U,
  kDMIC_DcCut155 = 1U,
  kDMIC_DcCut78 = 2U,
  kDMIC_DcCut39 = 3U }

  DMIC DC filter control values.
- enum dmic_channel_t {
  kDMIC_Channel0 = 0U,
  kDMIC_Channel1 = 1U,
  kDMIC_Channel2 = 2U,
  kDMIC_Channel3 = 3U,
  kDMIC_Channel4 = 4U,
  kDMIC_Channel5 = 5U,
  kDMIC_Channel6 = 6U,
  kDMIC_Channel7 = 7U }

  DMIC Channel number.
- enum {
  kDMIC_EnableChannel0 = 1 << 0U,
  kDMIC_EnableChannel1 = 1 << 1U,
  kDMIC_EnableChannel2 = 1 << 2U,
  kDMIC_EnableChannel3 = 1 << 3U,
  kDMIC_EnableChannel4 = 1 << 4U,
  kDMIC_EnableChannel5 = 1 << 5U,
  kDMIC_EnableChannel6 = 1 << 6U,
  kDMIC_EnableChannel7 = 1 << 7U }

_dmic_channel_mask DMIC Channel mask.

- enum dmic_phy_sample_rate_t {
  kDMIC_PhyFullSpeed = 0U,
  kDMIC_PhyHalfSpeed = 1U }
  DMIC and decimator sample rates.

DMIC version

- #define FSL_DMIC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  DMIC driver version 2.3.0.

Initialization and deinitialization

- uint32_t DMIC_GetInstance (DMIC_Type ∗base)
  Get the DMIC instance from peripheral base address.
- void DMIC_Init (DMIC_Type ∗base)
  Turns DMIC Clock on.
- void DMIC_DeInit (DMIC_Type ∗base)
  Turns DMIC Clock off.
- void DMIC_SetOperationMode (DMIC_Type ∗base, operation_mode_t mode)
  Set DMIC operating mode.
- void DMIC_Use2fs (DMIC_Type ∗base, bool use2fs)
  Configure Clock scaling.

Channel configuration

- void DMIC_CfgChannelDc (DMIC_Type ∗base, dmic_channel_t channel, dc_removal_t dc_cut_-level, uint32_t post_dc_gain_reduce, bool saturate16bit)
  Configure DMIC channel.
- static void DMIC_EnableChannelSignExtend (DMIC_Type ∗base, dmic_channel_t channel, bool enable)
  Enbale channel sign extend which allows processing of 24bit audio data on 32bit machines.
- void DMIC_ConfigChannel (DMIC_Type ∗base, dmic_channel_t channel, stereo_side_t side, dmic-_channel_config_t ∗channel_config)
  Configure DMIC channel.
- void DMIC_EnableChannnel (DMIC_Type ∗base, uint32_t channelmask)
  Enable a particualr channel.
- void DMIC_FifoChannel (DMIC_Type ∗base, uint32_t channel, uint32_t trig_level, uint32_-t enable, uint32_t resetn)
  Configure fifo settings for DMIC channel.
- static void DMIC_EnableChannelInterrupt (DMIC_Type ∗base, dmic_channel_t channel, bool enable)
  Enable a particualr channel interrupt request.
- static void DMIC_EnableChannelDma (DMIC_Type ∗base, dmic_channel_t channel, bool enable)
  Enable a particualr channel dma request.
- static void DMIC_EnableChannelFifo (DMIC_Type ∗base, dmic_channel_t channel, bool enable)

Enable a particualr channel fifo.
- static void DMIC_DoFifoReset (DMIC_Type *base, dmic_channel_t channel)
  Channel fifo reset.
- static uint32_t DMIC_FifoGetStatus (DMIC_Type *base, uint32_t channel)
  Get FIFO status.
- static void DMIC_FifoClearStatus (DMIC_Type *base, uint32_t channel, uint32_t mask)
  Clear FIFO status.
- static uint32_t DMIC_FifoGetData (DMIC_Type *base, uint32_t channel)
  Get FIFO data.
- static uint32_t DMIC_FifoGetAddress (DMIC_Type *base, uint32_t channel)
  Get FIFO address.
- void DMIC_ResetChannelDecimator (DMIC_Type *base, uint32_t channelMask, bool reset)
  DMIC channel Decimator reset.
- static void DMIC_EnableChannelGlobalSync (DMIC_Type *base, uint32_t channelMask, uint32_t syncCounter)
  Enable DMIC channel global sync function.
- static void DMIC_DisableChannelGlobalSync (DMIC_Type *base, uint32_t channelMask)
  Disbale DMIC channel global sync function.

Register callback.

- void DMIC_EnableIntCallback (DMIC_Type *base, dmic_callback_t cb)
  Enable callback.
- void DMIC_DisableIntCallback (DMIC_Type *base, dmic_callback_t cb)
  Disable callback.

HWVAD

- static void DMIC_SetGainNoiseEstHwvad (DMIC_Type *base, uint32_t value)
  Sets the gain value for the noise estimator.
- static void DMIC_SetGainSignalEstHwvad (DMIC_Type *base, uint32_t value)
  Sets the gain value for the signal estimator.
- static void DMIC_SetFilterCtrlHwvad (DMIC_Type *base, uint32_t value)
  Sets the hwvad filter cutoff frequency parameter.
- static void DMIC_SetInputGainHwvad (DMIC_Type *base, uint32_t value)
  Sets the input gain of hwvad.
- static void DMIC_CtrlClrIntrHwvad (DMIC_Type *base, bool st10)
  Clears hwvad internal interrupt flag.
- static void DMIC_FilterResetHwvad (DMIC_Type *base, bool rstt)
  Resets hwvad filters.
- static uint16_t DMIC_GetNoiseEnvlpEst (DMIC_Type *base)
  Gets the value from output of the filter z7.
- void DMIC_HwvadEnableIntCallback (DMIC_Type *base, dmic_hwvad_callback_t vadcb)
  Enable hwvad callback.
- void DMIC_HwvadDisableIntCallback (DMIC_Type *base, dmic_hwvad_callback_t vadcb)
  Disable callback.

13.4.2 Data Structure Documentation

13.4.2.1 struct dmic_channel_config_t

Data Fields

- pdm_div_t divhfclk
  DMIC Clock pre-divider values.
- uint32_t osr
  oversampling rate(CIC decimation rate) for PCM
- int32_t gainshft
  4FS PCM data gain control
- compensation_t preac2coef
  Pre-emphasis Filter coefficient value for 2FS.
- compensation_t preac4coef
  Pre-emphasis Filter coefficient value for 4FS.
- dc_removal_t dc_cut_level
  DMIC DC filter control values.
- uint32_t post_dc_gain_reduce
  Fine gain adjustment in the form of a number of bits to downshift.
- dmic_phy_sample_rate_t sample_rate
  DMIC and decimator sample rates.
- bool saturate16bit
  Selects 16-bit saturation.
- bool enableSignExtend
  sign extend feature which allows processing of 24bit audio data on 32bit machine

Field Documentation

(1) dc_removal_t dmic_channel_config_t::dc_cut_level

(2) bool dmic_channel_config_t::saturate16bit

0 means results roll over if out range and do not saturate. 1 means if the result overflows, it saturates at 0xFFFF for positive overflow and 0x8000 for negative overflow.

13.4.3 Macro Definition Documentation

13.4.3.1 #define FSL_DMIC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

13.4.4 Typedef Documentation

13.4.4.1 typedef void(∗ dmic_callback_t)(void)

13.4.4.2 typedef void(∗ dmic_hwvad_callback_t)(void)

13.4.5 Enumeration Type Documentation

13.4.5.1 anonymous enum

Enumerator

kStatus_DMIC_Busy DMIC is busy.
kStatus_DMIC_Idle DMIC is idle.
kStatus_DMIC_OverRunError DMIC over run Error.
kStatus_DMIC_UnderRunError DMIC under run Error.

13.4.5.2 enum operation_mode_t

Enumerator

kDMIC_OperationModeInterrupt Interrupt mode.
kDMIC_OperationModeDma DMA mode.

13.4.5.3 enum stereo_side_t

Enumerator

kDMIC_Left Left Stereo channel.
kDMIC_Right Right Stereo channel.

13.4.5.4 enum pdm_div_t

Enumerator

kDMIC_PdmDiv1 DMIC pre-divider set in divide by 1.
kDMIC_PdmDiv2 DMIC pre-divider set in divide by 2.
kDMIC_PdmDiv3 DMIC pre-divider set in divide by 3.
kDMIC_PdmDiv4 DMIC pre-divider set in divide by 4.
kDMIC_PdmDiv6 DMIC pre-divider set in divide by 6.
kDMIC_PdmDiv8 DMIC pre-divider set in divide by 8.
kDMIC_PdmDiv12 DMIC pre-divider set in divide by 12.
kDMIC_PdmDiv16 DMIC pre-divider set in divide by 16.
kDMIC_PdmDiv24 DMIC pre-divider set in divide by 24.
kDMIC_PdmDiv32 DMIC pre-divider set in divide by 32.
kDMIC_PdmDiv48 DMIC pre-divider set in divide by 48.
kDMIC_PdmDiv64 DMIC pre-divider set in divide by 64.
kDMIC_PdmDiv96 DMIC pre-divider set in divide by 96.
kDMIC_PdmDiv128 DMIC pre-divider set in divide by 128.

13.4.5.5 enum compensation_t

Enumerator

kDMIC_CompValueZero Compensation 0.
kDMIC_CompValueNegativePoint16 Compensation -0.16.
kDMIC_CompValueNegativePoint15 Compensation -0.15.
kDMIC_CompValueNegativePoint13 Compensation -0.13.

13.4.5.6 enum dc_removal_t

Enumerator

kDMIC_DcNoRemove Flat response no filter.
kDMIC_DcCut155 Cut off Frequency is 155 Hz.
kDMIC_DcCut78 Cut off Frequency is 78 Hz.
kDMIC_DcCut39 Cut off Frequency is 39 Hz.

13.4.5.7 enum dmic_channel_t

Enumerator

kDMIC_Channel0 DMIC channel 0.
kDMIC_Channel1 DMIC channel 1.
kDMIC_Channel2 DMIC channel 2.
kDMIC_Channel3 DMIC channel 3.
kDMIC_Channel4 DMIC channel 4.
kDMIC_Channel5 DMIC channel 5.
kDMIC_Channel6 DMIC channel 6.
kDMIC_Channel7 DMIC channel 7.

13.4.5.8 anonymous enum

Enumerator

kDMIC_EnableChannel0 DMIC channel 0 mask.
kDMIC_EnableChannel1 DMIC channel 1 mask.
kDMIC_EnableChannel2 DMIC channel 2 mask.
kDMIC_EnableChannel3 DMIC channel 3 mask.
kDMIC_EnableChannel4 DMIC channel 4 mask.
kDMIC_EnableChannel5 DMIC channel 5 mask.
kDMIC_EnableChannel6 DMIC channel 6 mask.
kDMIC_EnableChannel7 DMIC channel 7 mask.

13.4.5.9 enum dmic_phy_sample_rate_t

Enumerator

kDMIC_PhyFullSpeed Decimator gets one sample per each chosen clock edge of PDM interface.
kDMIC_PhyHalfSpeed PDM clock to Microphone is halved, decimator receives each sample twice.

13.4.6 Function Documentation

13.4.6.1 uint32_t DMIC_GetInstance (DMIC_Type ∗ *base*)

Parameters

base	DMIC peripheral base address.

Returns

DMIC instance.

13.4.6.2 void DMIC_Init (DMIC_Type ∗ *base*)

Parameters

base	: DMIC base

Returns

Nothing

13.4.6.3 void DMIC_DeInit (DMIC_Type ∗ *base*)

Parameters

base	: DMIC base

Returns

Nothing

13.4.6.4 void DMIC_SetOperationMode (DMIC_Type ∗ *base,* operation_mode_t *mode*)

Deprecated Do not use this function. It has been superceded by DMIC_EnableChannelInterrupt, DMIC_EnableChannelDma.

Parameters

base	: The base address of DMIC interface
mode	: DMIC mode

Returns

Nothing

13.4.6.5 void DMIC_Use2fs (DMIC_Type * *base,* bool *use2fs*)

Parameters

base	: The base address of DMIC interface
use2fs	: clock scaling

Returns

Nothing

13.4.6.6 void DMIC_CfgChannelDc (DMIC_Type * *base,* dmic_channel_t *channel,* dc_removal_t *dc_cut_level,* uint32_t *post_dc_gain_reduce,* bool *saturate16bit*)

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel
dc_cut_level	: dc_removal_t, Cut off Frequency
post_dc_gain_-reduce	: Fine gain adjustment in the form of a number of bits to downshift.
saturate16bit	: If selects 16-bit saturation.

13.4.6.7 static void DMIC_EnableChannelSignExtend (DMIC_Type * *base,* dmic_channel_t *channel,* bool *enable*) `[inline], [static]`

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel
enable	: true is enable sign extend, false is disable sign extend

13.4.6.8 void DMIC_ConfigChannel (DMIC_Type * *base*, dmic_channel_t *channel*, stereo_side_t *side*, dmic_channel_config_t * *channel_config*)

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel
side	: stereo_side_t, choice of left or right
channel_config	: Channel configuration

Returns

Nothing

13.4.6.9 void DMIC_EnableChannnel (DMIC_Type * *base*, uint32_t *channelmask*)

Parameters

base	: The base address of DMIC interface
channelmask	reference _dmic_channel_mask

Returns

Nothing

13.4.6.10 void DMIC_FifoChannel (DMIC_Type * *base*, uint32_t *channel*, uint32_t *trig_level*, uint32_t *enable*, uint32_t *resetn*)

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel
trig_level	: FIFO trigger level
enable	: FIFO level
resetn	: FIFO reset

Returns

Nothing

13.4.6.11 static void DMIC_EnableChannelInterrupt (DMIC_Type * *base,* dmic_channel_t *channel,* bool *enable*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: Channel selection
enable	: true is enable, false is disable

13.4.6.12 static void DMIC_EnableChannelDma (DMIC_Type * *base,* dmic_channel_t *channel,* bool *enable*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: Channel selection
enable	: true is enable, false is disable

13.4.6.13 static void DMIC_EnableChannelFifo (DMIC_Type * *base,* dmic_channel_t *channel,* bool *enable*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: Channel selection
enable	: true is enable, false is disable

13.4.6.14 static void DMIC_DoFifoReset (DMIC_Type * *base,* dmic_channel_t *channel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: Channel selection

13.4.6.15 static uint32_t DMIC_FifoGetStatus (DMIC_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel

Returns

FIFO status

13.4.6.16 static void DMIC_FifoClearStatus (DMIC_Type * *base,* uint32_t *channel,* uint32_t *mask*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel

mask	: Bits to be cleared

Returns

FIFO status

13.4.6.17 static uint32_t DMIC_FifoGetData (DMIC_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel

Returns

FIFO data

13.4.6.18 static uint32_t DMIC_FifoGetAddress (DMIC_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channel	: DMIC channel

Returns

FIFO data

13.4.6.19 void DMIC_ResetChannelDecimator (DMIC_Type * *base,* uint32_t *channelMask,* bool *reset*)

Parameters

base	: The base address of DMIC interface
channelMask	: DMIC channel mask, reference _dmic_channel_mask
reset	: true is reset decimator, false is release decimator.

13.4.6.20 static void DMIC_EnableChannelGlobalSync (DMIC_Type * *base,* uint32_t *channelMask,* uint32_t *syncCounter*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channelMask	: DMIC channel mask, reference _dmic_channel_mask
syncCounter	:sync counter will trigger a pulse whenever count reaches CCOUNTVAL. If CCOUNTVAL is set to 0, there will be a pulse on every cycle

13.4.6.21 static void DMIC_DisableChannelGlobalSync (DMIC_Type * *base,* uint32_t *channelMask*) [inline], [static]

Parameters

base	: The base address of DMIC interface
channelMask	: DMIC channel mask, reference _dmic_channel_mask

13.4.6.22 void DMIC_EnableIntCallback (DMIC_Type * *base,* dmic_callback_t *cb*)

This function enables the interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

base	Base address of the DMIC peripheral.
cb	callback Pointer to store callback function.

Return values

None.	

13.4.6.23 void DMIC_DisableIntCallback (DMIC_Type * *base,* dmic_callback_t *cb*)

This function disables the interrupt for the selected DMIC peripheral.

Parameters

base	Base address of the DMIC peripheral.
cb	callback Pointer to store callback function..

Return values

None.	

13.4.6.24 static void DMIC_SetGainNoiseEstHwvad (DMIC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DMIC base pointer
value	gain value for the noise estimator.

Return values

None.	

13.4.6.25 static void DMIC_SetGainSignalEstHwvad (DMIC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DMIC base pointer
value	gain value for the signal estimator.

Return values

None.	

13.4.6.26 static void DMIC_SetFilterCtrlHwvad (DMIC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DMIC base pointer
value	cut off frequency value.

Return values

None.	

13.4.6.27 static void DMIC_SetInputGainHwvad (DMIC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DMIC base pointer
value	input gain value for hwvad.

Return values

None.	

13.4.6.28 static void DMIC_CtrlClrIntrHwvad (DMIC_Type * *base,* bool *st10*) [inline], [static]

Parameters

base	DMIC base pointer

st10	bit value.

Return values

None.	

13.4.6.29 static void DMIC_FilterResetHwvad (DMIC_Type * *base*, bool *rstt*) [inline], [static]

Parameters

base	DMIC base pointer
rstt	Reset bit value.

Return values

None.	

13.4.6.30 static uint16_t DMIC_GetNoiseEnvlpEst (DMIC_Type * *base*) [inline], [static]

Parameters

base	DMIC base pointer

Return values

output	of filter z7.

13.4.6.31 void DMIC_HwvadEnableIntCallback (DMIC_Type * *base*, dmic_hwvad_callback_t *vadcb*)

This function enables the hwvad interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

Parameters

base	Base address of the DMIC peripheral.
vadcb	callback Pointer to store callback function.

Return values

None.	

13.4.6.32 void DMIC_HwvadDisableIntCallback (DMIC_Type * *base*, dmic_hwvad_callback_t *vadcb*)

This function disables the hwvad interrupt for the selected DMIC peripheral.

Parameters

base	Base address of the DMIC peripheral.
vadcb	callback Pointer to store callback function..

Return values

None.	

13.5 DMIC DMA Driver

13.5.1 Overview

Files

- file fsl_dmic_dma.h

Data Structures

- struct dmic_transfer_t
  DMIC transfer structure. More...
- struct dmic_dma_handle_t
  DMIC DMA handle. More...

Typedefs

- typedef void(∗ dmic_dma_transfer_callback_t)(DMIC_Type ∗base, dmic_dma_handle_t ∗handle, status_t status, void ∗userData)
  DMIC transfer callback function.

DMIC DMA version

- #define FSL_DMIC_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  DMIC DMA driver version 2.3.0.

DMA transactional

- status_t DMIC_TransferCreateHandleDMA (DMIC_Type ∗base, dmic_dma_handle_t ∗handle, dmic_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗rxDmaHandle)
  Initializes the DMIC handle which is used in transactional functions.
- status_t DMIC_TransferReceiveDMA (DMIC_Type ∗base, dmic_dma_handle_t ∗handle, dmic_transfer_t ∗xfer, uint32_t channel)
  Receives data using DMA.
- void DMIC_TransferAbortReceiveDMA (DMIC_Type ∗base, dmic_dma_handle_t ∗handle)
  Aborts the received data using DMA.
- status_t DMIC_TransferGetReceiveCountDMA (DMIC_Type ∗base, dmic_dma_handle_t ∗handle, uint32_t ∗count)
  Get the number of bytes that have been received.
- void DMIC_InstallDMADescriptorMemory (dmic_dma_handle_t ∗handle, void ∗linkAddr, size_t linkNum)
  Install DMA descriptor memory.

13.5.2 Data Structure Documentation

13.5.2.1 struct dmic_transfer_t

Data Fields

- void * data
  The buffer of data to be transfer.
- uint8_t dataWidth
  DMIC support 16bit/32bit.
- size_t dataSize
  The byte count to be transfer.
- uint8_t dataAddrInterleaveSize
  destination address interleave size
- struct _dmic_transfer * linkTransfer
  use to support link transfer

Field Documentation

(1) void* dmic_transfer_t::data

(2) size_t dmic_transfer_t::dataSize

13.5.2.2 struct _dmic_dma_handle

Data Fields

- DMIC_Type * base
  DMIC peripheral base address.
- dma_handle_t * rxDmaHandle
  The DMA RX channel used.
- dmic_dma_transfer_callback_t callback
  Callback function.
- void * userData
  DMIC callback function parameter.
- size_t transferSize
  Size of the data to receive.
- volatile uint8_t state
  Internal state of DMIC DMA transfer.
- dma_descriptor_t * desLink
  descriptor pool pointer
- size_t linkNum
  number of descriptor in descriptors pool

Field Documentation

(1) DMIC_Type* dmic_dma_handle_t::base

(2) dma_handle_t* dmic_dma_handle_t::rxDmaHandle

(3) dmic_dma_transfer_callback_t dmic_dma_handle_t::callback

(4) void∗ dmic_dma_handle_t::userData

(5) size_t dmic_dma_handle_t::transferSize

13.5.3 Typedef Documentation

13.5.3.1 typedef void(∗ dmic_dma_transfer_callback_t)(DMIC_Type ∗base, dmic_dma_handle_t ∗handle, status_t status, void ∗userData)

13.5.4 Function Documentation

13.5.4.1 status_t DMIC_TransferCreateHandleDMA (DMIC_Type ∗ *base,* dmic_dma_handle_t ∗ *handle,* dmic_dma_transfer_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *rxDmaHandle*)

Parameters

base	DMIC peripheral base address.
handle	Pointer to dmic_dma_handle_t structure.
callback	Callback function.
userData	User data.
rxDmaHandle	User-requested DMA handle for RX DMA transfer.

13.5.4.2 status_t DMIC_TransferReceiveDMA (DMIC_Type ∗ *base,* dmic_dma_handle_t ∗ *handle,* dmic_transfer_t ∗ *xfer,* uint32_t *channel*)

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

base	USART peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	DMIC DMA transfer structure. See dmic_transfer_t.

channel	DMIC start channel number.

Return values

kStatus_Success	

13.5.4.3 void DMIC_TransferAbortReceiveDMA (DMIC_Type * *base,* dmic_dma_handle_t * *handle*)

This function aborts the received data using DMA.

Parameters

base	DMIC peripheral base address
handle	Pointer to dmic_dma_handle_t structure

13.5.4.4 status_t DMIC_TransferGetReceiveCountDMA (DMIC_Type * *base,* dmic_dma_handle_t * *handle,* uint32_t * *count*)

This function gets the number of bytes that have been received.

Parameters

base	DMIC peripheral base address.
handle	DMIC handle pointer.
count	Receive bytes count.

Return values

kStatus_NoTransferInProgress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter count;

13.5.4.5 void DMIC_InstallDMADescriptorMemory (dmic_dma_handle_t * *handle,* void * *linkAddr,* size_t *linkNum*)

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, it should be called after DMIC_TransferCreateHandleDMA. User should be take care about the address of DMA descriptor pool which

required align with 16BYTE at least.

Parameters

handle	Pointer to DMA channel transfer handle.
linkAddr	DMA link descriptor address.
linkNum	DMA link descriptor number.

Chapter 14
FLEXCOMM: FLEXCOMM Driver

14.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FLEXCOMM drivers for the FLEXCOMM module of MCUXpresso SDK devices.

Modules

- FLEXCOMM Driver

14.2 FLEXCOMM Driver

14.2.1 Overview

Typedefs

- typedef void(∗ flexcomm_irq_handler_t)(void ∗base, void ∗handle)
  Typedef for interrupt handler.

Enumerations

- enum FLEXCOMM_PERIPH_T {
  FLEXCOMM_PERIPH_NONE,
  FLEXCOMM_PERIPH_USART,
  FLEXCOMM_PERIPH_SPI,
  FLEXCOMM_PERIPH_I2C,
  FLEXCOMM_PERIPH_I2S_TX,
  FLEXCOMM_PERIPH_I2S_RX }
  FLEXCOMM peripheral modes.

Functions

- uint32_t FLEXCOMM_GetInstance (void ∗base)
  Returns instance number for FLEXCOMM module with given base address.
- status_t FLEXCOMM_Init (void ∗base, FLEXCOMM_PERIPH_T periph)
  Initializes FLEXCOMM and selects peripheral mode according to the second parameter.
- void FLEXCOMM_SetIRQHandler (void ∗base, flexcomm_irq_handler_t handler, void ∗flexcomm-Handle)
  Sets IRQ handler for given FLEXCOMM module.

Variables

- IRQn_Type const kFlexcommIrqs [ ]
  Array with IRQ number for each FLEXCOMM module.

Driver version

- #define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  FlexCOMM driver version 2.0.2.

14.2.2 Macro Definition Documentation

14.2.2.1 #define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

14.2.3 Typedef Documentation

14.2.3.1 typedef void(∗ flexcomm_irq_handler_t)(void ∗base, void ∗handle)

14.2.4 Enumeration Type Documentation

14.2.4.1 enum FLEXCOMM_PERIPH_T

Enumerator

FLEXCOMM_PERIPH_NONE No peripheral.
FLEXCOMM_PERIPH_USART USART peripheral.
FLEXCOMM_PERIPH_SPI SPI Peripheral.
FLEXCOMM_PERIPH_I2C I2C Peripheral.
FLEXCOMM_PERIPH_I2S_TX I2S TX Peripheral.
FLEXCOMM_PERIPH_I2S_RX I2S RX Peripheral.

14.2.5 Function Documentation

14.2.5.1 uint32_t FLEXCOMM_GetInstance (void ∗ *base*)

14.2.5.2 status_t FLEXCOMM_Init (void ∗ *base,* FLEXCOMM_PERIPH_T *periph*)

14.2.5.3 void FLEXCOMM_SetIRQHandler (void ∗ *base,* flexcomm_irq_handler_t *handler,* void ∗ *flexcommHandle*)

It is used by drivers register IRQ handler according to FLEXCOMM mode

14.2.6 Variable Documentation

14.2.6.1 IRQn_Type const kFlexcommIrqs[]

Chapter 15
I2C: Inter-Integrated Circuit Driver

15.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of MCUXpresso SDK devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions I2C_MasterTransferNonBlocking() set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

15.2 Typical use case

15.2.1 Master Operation in functional method

```
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

/* Send start and slave address. */
I2C_MasterStart(EXAMPLE_I2C_MASTER_BASEADDR, 7-bit slave address,
      kI2C_Write/kI2C_Read);

/* Wait address sent out. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR)) & kI2C_IntPendingFlag))
{

}

if(status & kI2C_ReceiveNakFlag)
{
    return kStatus_I2C_Nak;
```

```
}

result = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER_BASEADDR, txBuff, BUFFER_SIZE,
      kI2C_TransferDefaultFlag);

if(result)
{
    /* If error occours, send STOP. */
    I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
    return result;
}

while(!(I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR) & kI2C_IntPendingFlag))
{

}

/* Wait all data sent out, send STOP. */
I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
```

15.2.2 Master Operation in interrupt transactional method

```
i2c_master_handle_t g_m_handle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_handle_t *handle,
      status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

I2C_MasterTransferCreateHandle(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle,
      i2c_master_callback, NULL);
I2C_MasterTransferNonBlocking(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle, &
      masterXfer);

/*  Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

15.2.3 Master Operation in DMA transactional method

```
i2c_master_dma_handle_t g_m_dma_handle;
dma_handle_t dmaHandle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_dma_handle_t *handle,
      status_t status, void *userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

DMA_EnableChannel(EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);
DMA_CreateHandle(&dmaHandle, EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);

I2C_MasterTransferCreateHandleDMA(EXAMPLE_I2C_MASTER_BASEADDR, &
      g_m_dma_handle, i2c_master_callback, NULL, &dmaHandle);
I2C_MasterTransferDMA(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_dma_handle, &masterXfer);

/*  Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

15.2.4 Slave Operation in functional method

```
i2c_slave_config_t slaveConfig;
uint8_t status;
status_t result = kStatus_Success;

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
       mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;
I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

/* Wait address match. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_SLAVE_BASEADDR)) & kI2C_AddressMatchFlag))
{

}
```

```
/* Slave transmit, master reading from slave. */
if (status & kI2C_TransferDirectionFlag)
{
    result = I2C_SlaveWriteBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}
else
{
    I2C_SlaveReadBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}

return result;
```

15.2.5 Slave Operation in interrupt transactional method

```
i2c_slave_config_t slaveConfig;
i2c_slave_handle_t g_s_handle;
volatile bool g_SlaveCompletionFlag = false;

static void i2c_slave_callback(I2C_Type *base, i2c_slave_transfer_t *xfer, void *
      userData)
{
    switch (xfer->event)
    {
        /*  Transmit request */
        case kI2C_SlaveTransmitEvent:
            /*  Update information for transmit process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /*  Receive request */
        case kI2C_SlaveReceiveEvent:
            /*  Update information for received process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /*  Transfer done */
        case kI2C_SlaveCompletionEvent:
            g_SlaveCompletionFlag = true;
            break;

        default:
            g_SlaveCompletionFlag = true;
            break;
    }
}

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
       mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;

I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

I2C_SlaveTransferCreateHandle(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
      i2c_slave_callback, NULL);

I2C_SlaveTransferNonBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
      kI2C_SlaveCompletionEvent);

/*  Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
```

```
g_SlaveCompletionFlag = false;
```

Modules

- I2C CMSIS Driver
- I2C DMA Driver
- I2C Driver
- I2C FreeRTOS Driver
- I2C Master Driver
- I2C Slave Driver

15.3 I2C Driver

15.3.1 Overview

Files

- file fsl_i2c.h

Macros

- #define I2C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.
- #define I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK 1U /∗ Define to one means master ignores the last byte's nack and considers the transfer successful. ∗/
  Whether to ignore the nack signal of the last byte during master transmit.
- #define I2C_STAT_MSTCODE_IDLE (0U)
  Master Idle State Code.
- #define I2C_STAT_MSTCODE_RXREADY (1U)
  Master Receive Ready State Code.
- #define I2C_STAT_MSTCODE_TXREADY (2U)
  Master Transmit Ready State Code.
- #define I2C_STAT_MSTCODE_NACKADR (3U)
  Master NACK by slave on address State Code.
- #define I2C_STAT_MSTCODE_NACKDAT (4U)
  Master NACK by slave on data State Code.

Enumerations

- enum {
  kStatus_I2C_Busy = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 0),
  kStatus_I2C_Idle = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 1),
  kStatus_I2C_Nak,
  kStatus_I2C_InvalidParameter,
  kStatus_I2C_BitError = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 4),
  kStatus_I2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 5),
  kStatus_I2C_NoTransferInProgress,
  kStatus_I2C_DmaRequestFail = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 7),
  kStatus_I2C_StartStopError = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 8),
  kStatus_I2C_UnexpectedState = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 9),
  kStatus_I2C_Timeout,
  kStatus_I2C_Addr_Nak = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 11),
  kStatus_I2C_EventTimeout = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 12),
  kStatus_I2C_SclLowTimeout = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 13) }
  I2C status return codes.

- enum _i2c_status_flags {
  kI2C_MasterPendingFlag = I2C_STAT_MSTPENDING_MASK,
  kI2C_MasterArbitrationLostFlag,
  kI2C_MasterStartStopErrorFlag,
  kI2C_MasterIdleFlag = 1UL << 5U,
  kI2C_MasterRxReadyFlag = 1UL << I2C_STAT_MSTSTATE_SHIFT,
  kI2C_MasterTxReadyFlag = 1UL << (I2C_STAT_MSTSTATE_SHIFT + 1U),
  kI2C_MasterAddrNackFlag = 1UL << 7U,
  kI2C_MasterDataNackFlag = 1UL << (I2C_STAT_MSTSTATE_SHIFT + 2U),
  kI2C_SlavePendingFlag = I2C_STAT_SLVPENDING_MASK,
  kI2C_SlaveNotStretching = I2C_STAT_SLVNOTSTR_MASK,
  kI2C_SlaveSelected,
  kI2C_SaveDeselected = I2C_STAT_SLVDESEL_MASK,
  kI2C_SlaveAddressedFlag = 1UL << 22U,
  kI2C_SlaveReceiveFlag = 1UL << I2C_STAT_SLVSTATE_SHIFT,
  kI2C_SlaveTransmitFlag = 1UL << (I2C_STAT_SLVSTATE_SHIFT + 1U),
  kI2C_SlaveAddress0MatchFlag = 1UL << 20U,
  kI2C_SlaveAddress1MatchFlag = 1UL << I2C_STAT_SLVIDX_SHIFT,
  kI2C_SlaveAddress2MatchFlag = 1UL << (I2C_STAT_SLVIDX_SHIFT + 1U),
  kI2C_SlaveAddress3MatchFlag = 1UL << 21U,
  kI2C_MonitorReadyFlag = I2C_STAT_MONRDY_MASK,
  kI2C_MonitorOverflowFlag = I2C_STAT_MONOV_MASK,
  kI2C_MonitorActiveFlag = I2C_STAT_MONACTIVE_MASK,
  kI2C_MonitorIdleFlag = I2C_STAT_MONIDLE_MASK,
  kI2C_EventTimeoutFlag = I2C_STAT_EVENTTIMEOUT_MASK,
  kI2C_SclTimeoutFlag = I2C_STAT_SCLTIMEOUT_MASK }
  I2C status flags.
- enum _i2c_interrupt_enable {
  kI2C_MasterPendingInterruptEnable,
  kI2C_MasterArbitrationLostInterruptEnable,
  kI2C_MasterStartStopErrorInterruptEnable,
  kI2C_SlavePendingInterruptEnable = I2C_STAT_SLVPENDING_MASK,
  kI2C_SlaveNotStretchingInterruptEnable,
  kI2C_SlaveDeselectedInterruptEnable = I2C_STAT_SLVDESEL_MASK,
  kI2C_MonitorReadyInterruptEnable = I2C_STAT_MONRDY_MASK,
  kI2C_MonitorOverflowInterruptEnable = I2C_STAT_MONOV_MASK,
  kI2C_MonitorIdleInterruptEnable = I2C_STAT_MONIDLE_MASK,
  kI2C_EventTimeoutInterruptEnable = I2C_STAT_EVENTTIMEOUT_MASK,
  kI2C_SclTimeoutInterruptEnable = I2C_STAT_SCLTIMEOUT_MASK }
  I2C interrupt enable.

Driver version

- #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

I2C driver version.

15.3.2 Macro Definition Documentation

15.3.2.1 #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

15.3.2.2 #define I2C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/

15.3.2.3 #define I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK 1U /∗ Define to one means master ignores the last byte's nack and considers the transfer successful. ∗/

15.3.3 Enumeration Type Documentation

15.3.3.1 anonymous enum

Enumerator

kStatus_I2C_Busy The master is already performing a transfer.
kStatus_I2C_Idle The slave driver is idle.
kStatus_I2C_Nak The slave device sent a NAK in response to a byte.
kStatus_I2C_InvalidParameter Unable to proceed due to invalid parameter.
kStatus_I2C_BitError Transferred bit was not seen on the bus.
kStatus_I2C_ArbitrationLost Arbitration lost error.
kStatus_I2C_NoTransferInProgress Attempt to abort a transfer when one is not in progress.
kStatus_I2C_DmaRequestFail DMA request failed.
kStatus_I2C_StartStopError Start and stop error.
kStatus_I2C_UnexpectedState Unexpected state.
kStatus_I2C_Timeout Timeout when waiting for I2C master/slave pending status to set to continue transfer.
kStatus_I2C_Addr_Nak NAK received for Address.
kStatus_I2C_EventTimeout Timeout waiting for bus event.
kStatus_I2C_SclLowTimeout Timeout SCL signal remains low.

15.3.3.2 enum _i2c_status_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI2C_MasterPendingFlag The I2C module is waiting for software interaction. bit 0

kI2C_MasterArbitrationLostFlag The arbitration of the bus was lost. There was collision on the bus. bit 4
kI2C_MasterStartStopErrorFlag There was an error during start or stop phase of the transaction. bit 6
kI2C_MasterIdleFlag The I2C master idle status. bit 5
kI2C_MasterRxReadyFlag The I2C master rx ready status. bit 1
kI2C_MasterTxReadyFlag The I2C master tx ready status. bit 2
kI2C_MasterAddrNackFlag The I2C master address nack status. bit 7
kI2C_MasterDataNackFlag The I2C master data nack status. bit 3
kI2C_SlavePendingFlag The I2C module is waiting for software interaction. bit 8
kI2C_SlaveNotStretching Indicates whether the slave is currently stretching clock (0 = yes, 1 = no). bit 11
kI2C_SlaveSelected Indicates whether the slave is selected by an address match. bit 14
kI2C_SaveDeselected Indicates that slave was previously deselected (deselect event took place, w1c). bit 15
kI2C_SlaveAddressedFlag One of the I2C slave's 4 addresses is matched. bit 22
kI2C_SlaveReceiveFlag Slave receive data available. bit 9
kI2C_SlaveTransmitFlag Slave data can be transmitted. bit 10
kI2C_SlaveAddress0MatchFlag Slave address0 match. bit 20
kI2C_SlaveAddress1MatchFlag Slave address1 match. bit 12
kI2C_SlaveAddress2MatchFlag Slave address2 match. bit 13
kI2C_SlaveAddress3MatchFlag Slave address3 match. bit 21
kI2C_MonitorReadyFlag The I2C monitor ready interrupt. bit 16
kI2C_MonitorOverflowFlag The monitor data overrun interrupt. bit 17
kI2C_MonitorActiveFlag The monitor is active. bit 18
kI2C_MonitorIdleFlag The monitor idle interrupt. bit 19
kI2C_EventTimeoutFlag The bus event timeout interrupt. bit 24
kI2C_SclTimeoutFlag The SCL timeout interrupt. bit 25

15.3.3.3 enum _i2c_interrupt_enable

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI2C_MasterPendingInterruptEnable The I2C master communication pending interrupt.
kI2C_MasterArbitrationLostInterruptEnable The I2C master arbitration lost interrupt.
kI2C_MasterStartStopErrorInterruptEnable The I2C master start/stop timing error interrupt.
kI2C_SlavePendingInterruptEnable The I2C slave communication pending interrupt.
kI2C_SlaveNotStretchingInterruptEnable The I2C slave not streching interrupt, deep-sleep mode can be entered only when this interrupt occurs.
kI2C_SlaveDeselectedInterruptEnable The I2C slave deselection interrupt.
kI2C_MonitorReadyInterruptEnable The I2C monitor ready interrupt.

kI2C_MonitorOverflowInterruptEnable The monitor data overrun interrupt.
kI2C_MonitorIdleInterruptEnable The monitor idle interrupt.
kI2C_EventTimeoutInterruptEnable The bus event timeout interrupt.
kI2C_SclTimeoutInterruptEnable The SCL timeout interrupt.

15.4 I2C Master Driver

15.4.1 Overview

Data Structures

- struct i2c_master_config_t
  Structure with settings to initialize the I2C master module. More...
- struct i2c_master_transfer_t
  Non-blocking transfer descriptor structure. More...
- struct i2c_master_handle_t
  Driver handle for master non-blocking APIs. More...

Typedefs

- typedef void(∗ i2c_master_transfer_callback_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle, status_t completionStatus, void ∗userData)
  Master completion callback function pointer type.

Enumerations

- enum i2c_direction_t {
  kI2C_Write = 0U,
  kI2C_Read = 1U }
  Direction of master and slave transfers.
- enum _i2c_master_transfer_flags {
  kI2C_TransferDefaultFlag = 0x00U,
  kI2C_TransferNoStartFlag = 0x01U,
  kI2C_TransferRepeatedStartFlag = 0x02U,
  kI2C_TransferNoStopFlag = 0x04U }
  Transfer option flags.
- enum _i2c_transfer_states
  States for the state machine used by transactional APIs.

Initialization and deinitialization

- void I2C_MasterGetDefaultConfig (i2c_master_config_t ∗masterConfig)
  Provides a default configuration for the I2C master peripheral.
- void I2C_MasterInit (I2C_Type ∗base, const i2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes the I2C master peripheral.
- void I2C_MasterDeinit (I2C_Type ∗base)
  Deinitializes the I2C master peripheral.
- uint32_t I2C_GetInstance (I2C_Type ∗base)
  Returns an instance number given a base address.

- static void I2C_MasterReset (I2C_Type *base)
  Performs a software reset.
- static void I2C_MasterEnable (I2C_Type *base, bool enable)
  Enables or disables the I2C module as master.

Status

- uint32_t I2C_GetStatusFlags (I2C_Type *base)
  Gets the I2C status flags.
- static void I2C_ClearStatusFlags (I2C_Type *base, uint32_t statusMask)
  Clears the I2C status flag state.
- static void I2C_MasterClearStatusFlags (I2C_Type *base, uint32_t statusMask)
  Clears the I2C master status flag state.

Interrupts

- static void I2C_EnableInterrupts (I2C_Type *base, uint32_t interruptMask)
  Enables the I2C interrupt requests.
- static void I2C_DisableInterrupts (I2C_Type *base, uint32_t interruptMask)
  Disables the I2C interrupt requests.
- static uint32_t I2C_GetEnabledInterrupts (I2C_Type *base)
  Returns the set of currently enabled I2C interrupt requests.

Bus operations

- void I2C_MasterSetBaudRate (I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the I2C bus frequency for master transactions.
- void I2C_MasterSetTimeoutValue (I2C_Type *base, uint8_t timeout_Ms, uint32_t srcClock_Hz)
  Sets the I2C bus timeout value.
- static bool I2C_MasterGetBusIdleState (I2C_Type *base)
  Returns whether the bus is idle.
- status_t I2C_MasterStart (I2C_Type *base, uint8_t address, i2c_direction_t direction)
  Sends a START on the I2C bus.
- status_t I2C_MasterStop (I2C_Type *base)
  Sends a STOP signal on the I2C bus.
- static status_t I2C_MasterRepeatedStart (I2C_Type *base, uint8_t address, i2c_direction_t direction)
  Sends a REPEATED START on the I2C bus.
- status_t I2C_MasterWriteBlocking (I2C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
  Performs a polling send transfer on the I2C bus.
- status_t I2C_MasterReadBlocking (I2C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)
  Performs a polling receive transfer on the I2C bus.
- status_t I2C_MasterTransferBlocking (I2C_Type *base, i2c_master_transfer_t *xfer)
  Performs a master polling transfer on the I2C bus.

Non-blocking

- void I2C_MasterTransferCreateHandle (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_callback_t callback, void *userData)
  Creates a new handle for the I2C master non-blocking APIs.
- status_t I2C_MasterTransferNonBlocking (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_t *xfer)
  Performs a non-blocking transaction on the I2C bus.
- status_t I2C_MasterTransferGetCount (I2C_Type *base, i2c_master_handle_t *handle, size_t *count)
  Returns number of bytes transferred so far.
- status_t I2C_MasterTransferAbort (I2C_Type *base, i2c_master_handle_t *handle)
  Terminates a non-blocking I2C master transmission early.

IRQ handler

- void I2C_MasterTransferHandleIRQ (I2C_Type *base, i2c_master_handle_t *handle)
  Reusable routine to handle master interrupts.

15.4.2 Data Structure Documentation

15.4.2.1 struct i2c_master_config_t

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the I2C_MasterGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Data Fields

- bool enableMaster
  Whether to enable master mode.
- uint32_t baudRate_Bps
  Desired baud rate in bits per second.
- bool enableTimeout
  Enable internal timeout function.
- uint8_t timeout_Ms
  Event timeout and SCL low timeout value.

Field Documentation

(1) bool i2c_master_config_t::enableMaster

(2) uint32_t i2c_master_config_t::baudRate_Bps

(3) bool i2c_master_config_t::enableTimeout

(4) uint8_t i2c_master_config_t::timeout_Ms

15.4.2.2 struct _i2c_master_transfer

I2C master transfer typedef.

This structure is used to pass transaction parameters to the I2C_MasterTransferNonBlocking() API.

Data Fields

- uint32_t flags
  Bit mask of options for the transfer.
- uint8_t slaveAddress
  The 7-bit slave address.
- i2c_direction_t direction
  Either kI2C_Read or kI2C_Write.
- uint32_t subaddress
  Sub address.
- size_t subaddressSize
  Length of sub address to send in bytes.
- void * data
  Pointer to data to transfer.
- size_t dataSize
  Number of bytes to transfer.

Field Documentation

(1) uint32_t i2c_master_transfer_t::flags

See enumeration _i2c_master_transfer_flags for available options. Set to 0 or kI2C_TransferDefaultFlag for normal transfers.

(2) uint8_t i2c_master_transfer_t::slaveAddress

(3) i2c_direction_t i2c_master_transfer_t::direction

(4) uint32_t i2c_master_transfer_t::subaddress

Transferred MSB first.

(5) size_t i2c_master_transfer_t::subaddressSize

Maximum size is 4 bytes.

(6) void* i2c_master_transfer_t::data

(7) size_t i2c_master_transfer_t::dataSize

15.4.2.3 struct _i2c_master_handle

I2C master handle typedef.

Note

The contents of this structure are private and subject to change.

Data Fields

- uint8_t state
  Transfer state machine current state.
- uint32_t transferCount
  Indicates progress of the transfer.
- uint32_t remainingBytes
  Remaining byte count in current state.
- uint8_t * buf
  Buffer pointer for current state.
- bool checkAddrNack
  Whether to check the nack signal is detected during addressing.
- i2c_master_transfer_t transfer
  Copy of the current transfer info.
- i2c_master_transfer_callback_t completionCallback
  Callback function pointer.
- void * userData
  Application data passed to callback.

Field Documentation

(1) uint8_t i2c_master_handle_t::state

(2) uint32_t i2c_master_handle_t::remainingBytes

(3) uint8_t* i2c_master_handle_t::buf

(4) bool i2c_master_handle_t::checkAddrNack

(5) i2c_master_transfer_t i2c_master_handle_t::transfer

(6) i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback

(7) void* i2c_master_handle_t::userData

15.4.3 Typedef Documentation

15.4.3.1 typedef void(∗ i2c_master_transfer_callback_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle, status_t completionStatus, void ∗userData)

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to I2C_MasterTransferCreateHandle().

Parameters

base	The I2C peripheral base address.
completion-Status	Either kStatus_Success or an error code describing how the transfer completed.
userData	Arbitrary pointer-sized value passed from the application.

15.4.4 Enumeration Type Documentation

15.4.4.1 enum i2c_direction_t

Enumerator

kI2C_Write Master transmit.
kI2C_Read Master receive.

15.4.4.2 enum _i2c_master_transfer_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the _i2c_-master_transfer::flags field.

Enumerator

kI2C_TransferDefaultFlag Transfer starts with a start signal, stops with a stop signal.
kI2C_TransferNoStartFlag Don't send a start condition, address, and sub address.
kI2C_TransferRepeatedStartFlag Send a repeated start condition.
kI2C_TransferNoStopFlag Don't send a stop condition.

15.4.4.3 enum _i2c_transfer_states

15.4.5 Function Documentation

15.4.5.1 void I2C_MasterGetDefaultConfig (i2c_master_config_t ∗ *masterConfig*)

This function provides the following default configuration for the I2C master peripheral:

```
*  masterConfig->enableMaster            = true;
*  masterConfig->baudRate_Bps            = 100000U;
*  masterConfig->enableTimeout           = false;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with I2C_MasterInit().

Parameters

out	*masterConfig*	User provided configuration structure for default values. Refer to i2c_-master_config_t.

15.4.5.2 void I2C_MasterInit (I2C_Type * *base*, const i2c_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

base	The I2C peripheral base address.
masterConfig	User provided peripheral configuration. Use I2C_MasterGetDefaultConfig() to get a set of defaults that you can override.
srcClock_Hz	Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

15.4.5.3 void I2C_MasterDeinit (I2C_Type * *base*)

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

15.4.5.4 uint32_t I2C_GetInstance (I2C_Type * *base*)

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

base	The I2C peripheral base address.

Returns

I2C instance number starting from 0.

15.4.5.5 static void I2C_MasterReset (I2C_Type * *base*) [inline], [static]

Restores the I2C master peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

15.4.5.6 static void I2C_MasterEnable (I2C_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	The I2C peripheral base address.
enable	Pass true to enable or false to disable the specified I2C as master.

15.4.5.7 uint32_t I2C_GetStatusFlags (I2C_Type * *base*)

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I2C peripheral base address.

Returns

State of the status flags:
- 1: related status flag is set.
- 0: related status flag is not set.

See Also

_i2c_status_flags.

15.4.5.8 static void I2C_ClearStatusFlags (I2C_Type * *base*, uint32_t *statusMask*) [inline], [static]

Refer to kI2C_CommonAllClearStatusFlags, kI2C_MasterAllClearStatusFlags and kI2C_SlaveAllClear-StatusFlags to see the clearable flags. Attempts to clear other flags has no effect.

Parameters

base	The I2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of the members in kI2C_CommonAllClearStatusFlags, kI2C_MasterAllClearStatusFlags and kI2C_-SlaveAllClearStatusFlags. You may pass the result of a previous call to I2C_Get-StatusFlags().

See Also

_i2c_status_flags, _i2c_master_status_flags and _i2c_slave_status_flags.

15.4.5.9 static void I2C_MasterClearStatusFlags (I2C_Type * *base*, uint32_t *statusMask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by I2C_ClearStatusFlags The following status register flags can be cleared:

- kI2C_MasterArbitrationLostFlag
- kI2C_MasterStartStopErrorFlag

Attempts to clear other flags has no effect.

Parameters

base	The I2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i2c_-status_flags enumerators OR'd together. You may pass the result of a previous call to I2C_GetStatusFlags().

See Also

_i2c_status_flags.

15.4.5.10 static void I2C_EnableInterrupts (I2C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	The I2C peripheral base address.
interruptMask	Bit mask of interrupts to enable. See _i2c_interrupt_enable for the set of constants that should be OR'd together to form the bit mask.

15.4.5.11 static void I2C_DisableInterrupts (I2C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	The I2C peripheral base address.
interruptMask	Bit mask of interrupts to disable. See _i2c_interrupt_enable for the set of constants that should be OR'd together to form the bit mask.

15.4.5.12 static uint32_t I2C_GetEnabledInterrupts (I2C_Type * *base*) [inline], [static]

Parameters

base	The I2C peripheral base address.

Returns

A bitmask composed of _i2c_interrupt_enable enumerators OR'd together to indicate the set of enabled interrupts.

15.4.5.13 void I2C_MasterSetBaudRate (I2C_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

The I2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

base	The I2C peripheral base address.
srcClock_Hz	I2C functional clock frequency in Hertz.
baudRate_Bps	Requested bus frequency in bits per second.

15.4.5.14 void I2C_MasterSetTimeoutValue (I2C_Type * *base*, uint8_t *timeout_Ms*, uint32_t *srcClock_Hz*)

If the SCL signal remains low or bus does not have event longer than the timeout value, kI2C_SclTimeoutFlag or kI2C_EventTimeoutFlag is set. This can indicete the bus is held by slave or any fault occurs to the I2C module.

Parameters

base	The I2C peripheral base address.
timeout_Ms	Timeout value in millisecond.
srcClock_Hz	I2C functional clock frequency in Hertz.

15.4.5.15 static bool I2C_MasterGetBusIdleState (I2C_Type * *base*) [inline], [static]

Requires the master mode to be enabled.

Parameters

base	The I2C peripheral base address.

Return values

true	Bus is busy.
false	Bus is idle.

15.4.5.16 status_t I2C_MasterStart (I2C_Type * *base*, uint8_t *address*, i2c_direction_t *direction*)

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

base	I2C peripheral base pointer
address	7-bit slave device address.
direction	Master transfer directions(transmit/receive).

Return values

kStatus_Success	Successfully send the start signal.
kStatus_I2C_Busy	Current bus is busy.

15.4.5.17 status_t I2C_MasterStop (I2C_Type ∗ *base*)

Return values

kStatus_Success	Successfully send the stop signal.
kStatus_I2C_Timeout	Send stop signal failed, timeout.

15.4.5.18 static status_t I2C_MasterRepeatedStart (I2C_Type ∗ *base,* uint8_t *address,* i2c_direction_t *direction*) `[inline], [static]`

Parameters

base	I2C peripheral base pointer
address	7-bit slave device address.
direction	Master transfer directions(transmit/receive).

Return values

kStatus_Success	Successfully send the start signal.
kStatus_I2C_Busy	Current bus is busy but not occupied by current I2C master.

15.4.5.19 status_t I2C_MasterWriteBlocking (I2C_Type ∗ *base,* const void ∗ *txBuff,* size_t *txSize,* uint32_t *flags*)

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_I2C_Nak.

Parameters

base	The I2C peripheral base address.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.
flags	Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

kStatus_Success	Data was sent successfully.
kStatus_I2C_Busy	Another master is currently utilizing the bus.
kStatus_I2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_I2C_Arbitration-Lost	Arbitration lost error.

15.4.5.20 status_t I2C_MasterReadBlocking (I2C_Type * *base*, void * *rxBuff*, size_t *rxSize*, uint32_t *flags*)

Parameters

base	The I2C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.
flags	Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag

Return values

kStatus_Success	Data was received successfully.
kStatus_I2C_Busy	Another master is currently utilizing the bus.
kStatus_I2C_Nak	The slave device sent a NAK in response to a byte.
kStatus_I2C_Arbitration-Lost	Arbitration lost error.

15.4.5.21 status_t I2C_MasterTransferBlocking (I2C_Type * *base*, i2c_master_transfer_t * *xfer*)

Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

Parameters

base	I2C peripheral base address.
xfer	Pointer to the transfer structure.

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_Busy	Previous transmission still not finished.
kStatus_I2C_Timeout	Transfer error, wait signal timeout.
kStatus_I2C_Arbitration-Lost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.
kStataus_I2C_Addr_Nak	Transfer error, receive NAK during addressing.

15.4.5.22 void I2C_MasterTransferCreateHandle (I2C_Type ∗ *base,* i2c_master_handle_t ∗ *handle,* i2c_master_transfer_callback_t *callback,* void ∗ *userData*)

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I2C_MasterTransferAbort() API shall be called.

Parameters

	base	The I2C peripheral base address.
`out`	*handle*	Pointer to the I2C master driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

15.4.5.23 status_t I2C_MasterTransferNonBlocking (I2C_Type ∗ *base,* i2c_master_handle_t ∗ *handle,* i2c_master_transfer_t ∗ *xfer*)

Parameters

base	The I2C peripheral base address.
handle	Pointer to the I2C master driver handle.
xfer	The pointer to the transfer descriptor.

Return values

kStatus_Success	The transaction was started successfully.
kStatus_I2C_Busy	Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

15.4.5.24 status_t I2C_MasterTransferGetCount (I2C_Type ∗ *base*, i2c_master_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

	base	The I2C peripheral base address.
	handle	Pointer to the I2C master driver handle.
`out`	*count*	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_I2C_Busy	

15.4.5.25 status_t I2C_MasterTransferAbort (I2C_Type ∗ *base*, i2c_master_handle_t ∗ *handle*)

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

Parameters

base	The I2C peripheral base address.
handle	Pointer to the I2C master driver handle.

Return values

kStatus_Success	A transaction was successfully aborted.
kStatus_I2C_Timeout	Timeout during polling for flags.

15.4.5.26 void I2C_MasterTransferHandleIRQ (I2C_Type * *base*, i2c_master_handle_t * *handle*)

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

base	The I2C peripheral base address.
handle	Pointer to the I2C master driver handle.

15.5 I2C Slave Driver

15.5.1 Overview

Data Structures

- struct i2c_slave_address_t
  Data structure with 7-bit Slave address and Slave address disable. More...
- struct i2c_slave_config_t
  Structure with settings to initialize the I2C slave module. More...
- struct i2c_slave_transfer_t
  I2C slave transfer structure. More...
- struct i2c_slave_handle_t
  I2C slave handle structure. More...

Typedefs

- typedef void(∗ i2c_slave_transfer_callback_t)(I2C_Type ∗base, volatile i2c_slave_transfer_t ∗transfer, void ∗userData)
  Slave event callback function pointer type.
- typedef void(∗ flexcomm_i2c_master_irq_handler_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle)
  Typedef for master interrupt handler.
- typedef void(∗ flexcomm_i2c_slave_irq_handler_t)(I2C_Type ∗base, i2c_slave_handle_t ∗handle)
  Typedef for slave interrupt handler.

Enumerations

- enum i2c_slave_address_register_t {
  kI2C_SlaveAddressRegister0 = 0U,
  kI2C_SlaveAddressRegister1 = 1U,
  kI2C_SlaveAddressRegister2 = 2U,
  kI2C_SlaveAddressRegister3 = 3U }
  I2C slave address register.
- enum i2c_slave_address_qual_mode_t {
  kI2C_QualModeMask = 0U,
  kI2C_QualModeExtend }
  I2C slave address match options.
- enum i2c_slave_bus_speed_t
  I2C slave bus speed options.
- enum i2c_slave_transfer_event_t {
  kI2C_SlaveAddressMatchEvent = 0x01U,
  kI2C_SlaveTransmitEvent = 0x02U,
  kI2C_SlaveReceiveEvent = 0x04U,
  kI2C_SlaveCompletionEvent = 0x20U,
  kI2C_SlaveDeselectedEvent,

kI2C_SlaveAllEvents }
Set of events sent to the callback for non blocking slave transfers.
- enum i2c_slave_fsm_t
  I2C slave software finite state machine states.

Slave initialization and deinitialization

- void I2C_SlaveGetDefaultConfig (i2c_slave_config_t *slaveConfig)
  Provides a default configuration for the I2C slave peripheral.
- status_t I2C_SlaveInit (I2C_Type *base, const i2c_slave_config_t *slaveConfig, uint32_t srcClock_Hz)
  Initializes the I2C slave peripheral.
- void I2C_SlaveSetAddress (I2C_Type *base, i2c_slave_address_register_t addressRegister, uint8_t address, bool addressDisable)
  Configures Slave Address n register.
- void I2C_SlaveDeinit (I2C_Type *base)
  Deinitializes the I2C slave peripheral.
- static void I2C_SlaveEnable (I2C_Type *base, bool enable)
  Enables or disables the I2C module as slave.

Slave status

- static void I2C_SlaveClearStatusFlags (I2C_Type *base, uint32_t statusMask)
  Clears the I2C status flag state.

Slave bus operations

- status_t I2C_SlaveWriteBlocking (I2C_Type *base, const uint8_t *txBuff, size_t txSize)
  Performs a polling send transfer on the I2C bus.
- status_t I2C_SlaveReadBlocking (I2C_Type *base, uint8_t *rxBuff, size_t rxSize)
  Performs a polling receive transfer on the I2C bus.

Slave non-blocking

- void I2C_SlaveTransferCreateHandle (I2C_Type *base, i2c_slave_handle_t *handle, i2c_slave_transfer_callback_t callback, void *userData)
  Creates a new handle for the I2C slave non-blocking APIs.
- status_t I2C_SlaveTransferNonBlocking (I2C_Type *base, i2c_slave_handle_t *handle, uint32_t eventMask)
  Starts accepting slave transfers.
- status_t I2C_SlaveSetSendBuffer (I2C_Type *base, volatile i2c_slave_transfer_t *transfer, const void *txData, size_t txSize, uint32_t eventMask)
  Starts accepting master read from slave requests.
- status_t I2C_SlaveSetReceiveBuffer (I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *rxData, size_t rxSize, uint32_t eventMask)

Starts accepting master write to slave requests.

- static uint32_t I2C_SlaveGetReceivedAddress (I2C_Type *base, volatile i2c_slave_transfer_t *transfer)
  Returns the slave address sent by the I2C master.
- void I2C_SlaveTransferAbort (I2C_Type *base, i2c_slave_handle_t *handle)
  Aborts the slave non-blocking transfers.
- status_t I2C_SlaveTransferGetCount (I2C_Type *base, i2c_slave_handle_t *handle, size_t *count)
  Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.

Slave IRQ handler

- void I2C_SlaveTransferHandleIRQ (I2C_Type *base, i2c_slave_handle_t *handle)
  Reusable routine to handle slave interrupts.

15.5.2 Data Structure Documentation

15.5.2.1 struct i2c_slave_address_t

Data Fields

- uint8_t address
  7-bit Slave address SLVADR.
- bool addressDisable
  Slave address disable SADISABLE.

Field Documentation

(1) uint8_t i2c_slave_address_t::address

(2) bool i2c_slave_address_t::addressDisable

15.5.2.2 struct i2c_slave_config_t

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the I2C_SlaveGetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Data Fields

- i2c_slave_address_t address0
  Slave's 7-bit address and disable.
- i2c_slave_address_t address1
  Alternate slave 7-bit address and disable.
- i2c_slave_address_t address2
  Alternate slave 7-bit address and disable.

- i2c_slave_address_t address3
  Alternate slave 7-bit address and disable.
- i2c_slave_address_qual_mode_t qualMode
  Qualify mode for slave address 0.
- uint8_t qualAddress
  Slave address qualifier for address 0.
- i2c_slave_bus_speed_t busSpeed
  Slave bus speed mode.
- bool enableSlave
  Enable slave mode.

Field Documentation

(1) i2c_slave_address_t i2c_slave_config_t::address0

(2) i2c_slave_address_t i2c_slave_config_t::address1

(3) i2c_slave_address_t i2c_slave_config_t::address2

(4) i2c_slave_address_t i2c_slave_config_t::address3

(5) i2c_slave_address_qual_mode_t i2c_slave_config_t::qualMode

(6) uint8_t i2c_slave_config_t::qualAddress

(7) i2c_slave_bus_speed_t i2c_slave_config_t::busSpeed

If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The busSpeed value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the busSpeed mode is unknown at compile time, use the longest data setup time kI2C_SlaveStandardMode (250 ns)

(8) bool i2c_slave_config_t::enableSlave

15.5.2.3 struct i2c_slave_transfer_t

Data Fields

- i2c_slave_handle_t * handle
  Pointer to handle that contains this transfer.
- i2c_slave_transfer_event_t event
  Reason the callback is being invoked.
- uint8_t receivedAddress
  Matching address send by master.
- uint32_t eventMask
  Mask of enabled events.
- uint8_t * rxData
  Transfer buffer for receive data.

- const uint8_t * txData
  Transfer buffer for transmit data.
- size_t txSize
  Transfer size.
- size_t rxSize
  Transfer size.
- size_t transferredCount
  Number of bytes transferred during this transfer.
- status_t completionStatus
  Success or error code describing how the transfer completed.

Field Documentation

(1) i2c_slave_handle_t* i2c_slave_transfer_t::handle

(2) i2c_slave_transfer_event_t i2c_slave_transfer_t::event

(3) uint8_t i2c_slave_transfer_t::receivedAddress

7-bits plus R/nW bit0

(4) uint32_t i2c_slave_transfer_t::eventMask

(5) size_t i2c_slave_transfer_t::transferredCount

(6) status_t i2c_slave_transfer_t::completionStatus

Only applies for kI2C_SlaveCompletionEvent.

15.5.2.4 struct _i2c_slave_handle

I2C slave handle typedef.

Note

The contents of this structure are private and subject to change.

Data Fields

- volatile i2c_slave_transfer_t transfer
  I2C slave transfer.
- volatile bool isBusy
  Whether transfer is busy.
- volatile i2c_slave_fsm_t slaveFsm
  slave transfer state machine.
- i2c_slave_transfer_callback_t callback
  Callback function called at transfer event.
- void * userData
  Callback parameter passed to callback.

Field Documentation

(1) volatile i2c_slave_transfer_t i2c_slave_handle_t::transfer

(2) volatile bool i2c_slave_handle_t::isBusy

(3) volatile i2c_slave_fsm_t i2c_slave_handle_t::slaveFsm

(4) i2c_slave_transfer_callback_t i2c_slave_handle_t::callback

(5) void∗ i2c_slave_handle_t::userData

15.5.3 Typedef Documentation

15.5.3.1 typedef void(∗ i2c_slave_transfer_callback_t)(I2C_Type ∗base, volatile i2c_slave_transfer_t ∗transfer, void ∗userData)

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I2C_-SlaveSetCallback() function after you have created a handle.

Parameters

base	Base address for the I2C instance on which the event occurred.
transfer	Pointer to transfer descriptor containing values passed to and/or from the callback.
userData	Arbitrary pointer-sized value passed from the application.

15.5.3.2 typedef void(∗ flexcomm_i2c_master_irq_handler_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle)

15.5.3.3 typedef void(∗ flexcomm_i2c_slave_irq_handler_t)(I2C_Type ∗base, i2c_slave_handle_t ∗handle)

15.5.4 Enumeration Type Documentation

15.5.4.1 enum i2c_slave_address_register_t

Enumerator

kI2C_SlaveAddressRegister0	Slave Address 0 register.
kI2C_SlaveAddressRegister1	Slave Address 1 register.
kI2C_SlaveAddressRegister2	Slave Address 2 register.
kI2C_SlaveAddressRegister3	Slave Address 3 register.

15.5.4.2 enum i2c_slave_address_qual_mode_t

Enumerator

kI2C_QualModeMask The SLVQUAL0 field (qualAddress) is used as a logical mask for matching address0.

kI2C_QualModeExtend The SLVQUAL0 (qualAddress) field is used to extend address 0 matching in a range of addresses.

15.5.4.3 enum i2c_slave_bus_speed_t

15.5.4.4 enum i2c_slave_transfer_event_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I2C_SlaveTransferNonBlocking() in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

kI2C_SlaveAddressMatchEvent Received the slave address after a start or repeated start.

kI2C_SlaveTransmitEvent Callback is requested to provide data to transmit (slave-transmitter role).

kI2C_SlaveReceiveEvent Callback is requested to provide a buffer in which to place received data (slave-receiver role).

kI2C_SlaveCompletionEvent All data in the active transfer have been consumed.

kI2C_SlaveDeselectedEvent The slave function has become deselected (SLVSEL flag changing from 1 to 0.

kI2C_SlaveAllEvents Bit mask of all available events.

15.5.5 Function Documentation

15.5.5.1 void I2C_SlaveGetDefaultConfig (i2c_slave_config_t * *slaveConfig*)

This function provides the following default configuration for the I2C slave peripheral:

```
*  slaveConfig->enableSlave = true;
*  slaveConfig->address0.disable = false;
*  slaveConfig->address0.address = 0u;
*  slaveConfig->address1.disable = true;
*  slaveConfig->address2.disable = true;
*  slaveConfig->address3.disable = true;
*  slaveConfig->busSpeed = kI2C_SlaveStandardMode;
*
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with I2C_SlaveInit(). Be sure to override at least the *address0.address* member of the configuration structure with the desired slave address.

Parameters

out	*slaveConfig*	User provided configuration structure that is set to default values. Refer to i2c_slave_config_t.

15.5.5.2 status_t I2C_SlaveInit (I2C_Type * *base*, const i2c_slave_config_t * *slaveConfig*, uint32_t *srcClock_Hz*)

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

Parameters

base	The I2C peripheral base address.
slaveConfig	User provided peripheral configuration. Use I2C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
srcClock_Hz	Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock.

15.5.5.3 void I2C_SlaveSetAddress (I2C_Type * *base*, i2c_slave_address_register_t *addressRegister*, uint8_t *address*, bool *addressDisable*)

This function writes new value to Slave Address register.

Parameters

base	The I2C peripheral base address.
addressRegister	The module supports multiple address registers. The parameter determines which one shall be changed.
address	The slave address to be stored to the address register for matching.
addressDisable	Disable matching of the specified address register.

15.5.5.4 void I2C_SlaveDeinit (I2C_Type * *base*)

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The I2C peripheral base address.

15.5.5.5 static void I2C_SlaveEnable (I2C_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	The I2C peripheral base address.
enable	True to enable or flase to disable.

15.5.5.6 static void I2C_SlaveClearStatusFlags (I2C_Type * *base,* uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

Parameters

base	The I2C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i2c_-slave_flags enumerators OR'd together. You may pass the result of a previous call to I2C_SlaveGetStatusFlags().

See Also

_i2c_slave_flags.

15.5.5.7 status_t I2C_SlaveWriteBlocking (I2C_Type * *base,* const uint8_t * *txBuff,* size_t *txSize*)

The function executes blocking address phase and blocking data phase.

Parameters

base	The I2C peripheral base address.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.

Returns

kStatus_Success Data has been sent.
kStatus_Fail Unexpected slave state (master data write while master read from slave is expected).

15.5.5.8 status_t I2C_SlaveReadBlocking (I2C_Type * *base*, uint8_t * *rxBuff*, size_t *rxSize*)

The function executes blocking address phase and blocking data phase.

Parameters

base	The I2C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.

Returns

kStatus_Success Data has been received.
kStatus_Fail Unexpected slave state (master data read while master write to slave is expected).

15.5.5.9 void I2C_SlaveTransferCreateHandle (I2C_Type * *base*, i2c_slave_handle_t * *handle*, i2c_slave_transfer_callback_t *callback*, void * *userData*)

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the I2C_SlaveTransferAbort() API shall be called.

Parameters

	base	The I2C peripheral base address.
out	*handle*	Pointer to the I2C slave driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

15.5.5.10 status_t I2C_SlaveTransferNonBlocking (I2C_Type * *base*, i2c_slave_handle_t * *handle*, uint32_t *eventMask*)

Call this API after calling I2C_SlaveInit() and I2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to I2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

If no slave Tx transfer is busy, a master read from slave request invokes kI2C_SlaveTransmitEvent callback. If no slave Rx transfer is busy, a master write to slave request invokes kI2C_SlaveReceiveEvent callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
handle	Pointer to i2c_slave_handle_t structure which stores the transfer state.
eventMask	Bit mask formed by OR'ing together i2c_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 kI2C_SlaveAllEvents to enable all events.

Return values

kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

15.5.5.11 status_t I2C_SlaveSetSendBuffer (I2C_Type * *base*, volatile i2c_slave_transfer_t * *transfer*, const void * *txData*, size_t *txSize*, uint32_t *eventMask*)

The function can be called in response to kI2C_SlaveTransmitEvent callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
transfer	Pointer to i2c_slave_transfer_t structure.
txData	Pointer to data to send to master.
txSize	Size of txData in bytes.
eventMask	Bit mask formed by OR'ing together i2c_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 kI2C_SlaveAllEvents to enable all events.

Return values

kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

15.5.5.12 status_t I2C_SlaveSetReceiveBuffer (I2C_Type * *base*, volatile i2c_slave_transfer_t * *transfer*, void * *rxData*, size_t *rxSize*, uint32_t *eventMask*)

The function can be called in response to kI2C_SlaveReceiveEvent callback to start a new slave Rx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and kI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
transfer	Pointer to i2c_slave_transfer_t structure.
rxData	Pointer to data to store data from master.
rxSize	Size of rxData in bytes.
eventMask	Bit mask formed by OR'ing together i2c_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 kI2C_SlaveAllEvents to enable all events.

Return values

kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

15.5.5.13 static uint32_t I2C_SlaveGetReceivedAddress (I2C_Type * *base*, volatile i2c_slave_transfer_t * *transfer*) [inline], [static]

This function should only be called from the address match event callback kI2C_SlaveAddressMatch-Event.

Parameters

base	The I2C peripheral base address.
transfer	The I2C slave transfer.

Returns

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

15.5.5.14 void I2C_SlaveTransferAbort (I2C_Type * *base*, i2c_slave_handle_t * *handle*)

Note

This API could be called at any time to stop slave for handling the bus events.

Parameters

base	The I2C peripheral base address.
handle	Pointer to i2c_slave_handle_t structure which stores the transfer state.

Return values

kStatus_Success	
kStatus_I2C_Idle	

15.5.5.15 status_t I2C_SlaveTransferGetCount (I2C_Type * *base*, i2c_slave_handle_t * *handle*, size_t * *count*)

Parameters

base	I2C base pointer.
handle	pointer to i2c_slave_handle_t structure.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

15.5.5.16 void I2C_SlaveTransferHandleIRQ (I2C_Type * *base*, i2c_slave_handle_t * *handle*)

Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

Parameters

base	The I2C peripheral base address.

handle	Pointer to i2c_slave_handle_t structure which stores the transfer state.

15.6 I2C DMA Driver

15.6.1 Overview

Data Structures

- struct i2c_master_dma_handle_t
  I2C master dma transfer structure. More...

Macros

- #define I2C_MAX_DMA_TRANSFER_COUNT 1024
  Maximum lenght of single DMA transfer (determined by capability of the DMA engine)

Typedefs

- typedef void(∗ i2c_master_dma_transfer_callback_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, status_t status, void ∗userData)
  I2C master dma transfer callback typedef.
- typedef void(∗ flexcomm_i2c_dma_master_irq_handler_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle)
  Typedef for master dma handler.

Driver version

- #define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  I2C DMA driver version.

I2C Block DMA Transfer Operation

- void I2C_MasterTransferCreateHandleDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, i2c_master_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗dmaHandle)
  Init the I2C handle which is used in transactional functions.
- status_t I2C_MasterTransferDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, i2c_master_transfer_t ∗xfer)
  Performs a master dma non-blocking transfer on the I2C bus.
- status_t I2C_MasterTransferGetCountDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, size_t ∗count)
  Get master transfer status during a dma non-blocking transfer.
- void I2C_MasterTransferAbortDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle)
  Abort a master dma non-blocking transfer in a early time.

15.6.2 Data Structure Documentation

15.6.2.1 struct _i2c_master_dma_handle

I2C master dma handle typedef.

Data Fields

- uint8_t state
  Transfer state machine current state.
- uint32_t transferCount
  Indicates progress of the transfer.
- uint32_t remainingBytesDMA
  Remaining byte count to be transferred using DMA.
- uint8_t * buf
  Buffer pointer for current state.
- bool checkAddrNack
  Whether to check the nack signal is detected during addressing.
- dma_handle_t * dmaHandle
  The DMA handler used.
- i2c_master_transfer_t transfer
  Copy of the current transfer info.
- i2c_master_dma_transfer_callback_t completionCallback
  Callback function called after dma transfer finished.
- void * userData
  Callback parameter passed to callback function.

Field Documentation

(1) uint8_t i2c_master_dma_handle_t::state

(2) uint32_t i2c_master_dma_handle_t::remainingBytesDMA

(3) uint8_t* i2c_master_dma_handle_t::buf

(4) bool i2c_master_dma_handle_t::checkAddrNack

(5) dma_handle_t* i2c_master_dma_handle_t::dmaHandle

(6) i2c_master_transfer_t i2c_master_dma_handle_t::transfer

(7) i2c_master_dma_transfer_callback_t i2c_master_dma_handle_t::completionCallback

(8) void* i2c_master_dma_handle_t::userData

15.6.3 Macro Definition Documentation

15.6.3.1 #define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

15.6.4 Typedef Documentation

15.6.4.1 typedef void(∗ i2c_master_dma_transfer_callback_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, status_t status, void ∗userData)

15.6.4.2 typedef void(∗ flexcomm_i2c_dma_master_irq_handler_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle)

15.6.5 Function Documentation

15.6.5.1 void I2C_MasterTransferCreateHandleDMA (I2C_Type ∗ *base,* i2c_master_dma_handle_t ∗ *handle,* i2c_master_dma_transfer_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *dmaHandle*)

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure
callback	pointer to user callback function
userData	user param passed to the callback function
dmaHandle	DMA handle pointer

15.6.5.2 status_t I2C_MasterTransferDMA (I2C_Type ∗ *base,* i2c_master_dma_handle_t ∗ *handle,* i2c_master_transfer_t ∗ *xfer*)

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure
xfer	pointer to transfer structure of i2c_master_transfer_t

Return values

kStatus_Success	Sucessully complete the data transmission.
kStatus_I2C_Busy	Previous transmission still not finished.

kStatus_I2C_Timeout	Transfer error, wait signal timeout.
kStatus_I2C_Arbitration-Lost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive Nak during transfer.

15.6.5.3 status_t I2C_MasterTransferGetCountDMA (I2C_Type * *base,* i2c_master_dma_handle_t * *handle,* size_t * *count*)

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure
count	Number of bytes transferred so far by the non-blocking transaction.

15.6.5.4 void I2C_MasterTransferAbortDMA (I2C_Type * *base,* i2c_master_dma_handle_t * *handle*)

Parameters

base	I2C peripheral base address
handle	pointer to i2c_master_dma_handle_t structure

15.7 I2C FreeRTOS Driver

15.7.1 Overview

Data Structures

- struct i2c_rtos_handle_t
  I2C FreeRTOS handle. More...

Driver version

- #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 8))
  I2C FreeRTOS driver version 2.0.8.

I2C RTOS Operation

- status_t I2C_RTOS_Init (i2c_rtos_handle_t ∗handle, I2C_Type ∗base, const i2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes I2C.
- status_t I2C_RTOS_Deinit (i2c_rtos_handle_t ∗handle)
  Deinitializes the I2C.
- status_t I2C_RTOS_Transfer (i2c_rtos_handle_t ∗handle, i2c_master_transfer_t ∗transfer)
  Performs I2C transfer.

15.7.2 Data Structure Documentation

15.7.2.1 struct i2c_rtos_handle_t

Data Fields

- I2C_Type ∗ base
  I2C base address.
- i2c_master_handle_t drv_handle
  A handle of the underlying driver, treated as opaque by the RTOS layer.
- status_t async_status
  Transactional state of the underlying driver.
- SemaphoreHandle_t mutex
  A mutex to lock the handle during a transfer.
- SemaphoreHandle_t semaphore
  A semaphore to notify and unblock task when the transfer ends.

15.7.3 Macro Definition Documentation

15.7.3.1 #define FSL_I2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 8))

15.7.4 Function Documentation

15.7.4.1 status_t I2C_RTOS_Init (i2c_rtos_handle_t * *handle*, I2C_Type * *base*, const i2c_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

This function initializes the I2C module and the related RTOS context.

Parameters

handle	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the I2C instance to initialize.
masterConfig	Configuration structure to set-up I2C in master mode.
srcClock_Hz	Frequency of input clock of the I2C module.

Returns

status of the operation.

15.7.4.2 status_t I2C_RTOS_Deinit (i2c_rtos_handle_t * *handle*)

This function deinitializes the I2C module and the related RTOS context.

Parameters

handle	The RTOS I2C handle.

15.7.4.3 status_t I2C_RTOS_Transfer (i2c_rtos_handle_t * *handle*, i2c_master_transfer_t * *transfer*)

This function performs an I2C transfer according to data given in the transfer structure.

Parameters

handle	The RTOS I2C handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

15.8 I2C CMSIS Driver

This section describes the programming interface of the I2C Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage methord see http://www.keil.-com/pack/doc/cmsis/Driver/html/index.html.

The I2C CMSIS driver includes transactional APIs.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code accessing the hardware registers.

15.8.1 I2C CMSIS Driver

15.8.1.1 Master Operation in interrupt transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}
/*Init I2C MASTER*/
EXAMPLE_I2C_MASTER.Initialize(I2C_MasterSignalEvent_t);

EXAMPLE_I2C_MASTER.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
EXAMPLE_I2C_MASTER.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transmit*/
EXAMPLE_I2C_MASTER.MasterTransmit(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/*  Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

15.8.1.2 Master Operation in DMA transactional method

```
void I2C_MasterSignalEvent_t(uint32_t event)
{
    /*  Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Init DMA*/
DMA_Init(EXAMPLE_DMA);
```

```
/*Init I2C MASTER*/
EXAMPLE_I2C_MASTER.Initialize(I2C_MasterSignalEvent_t);

EXAMPLE_I2C_MASTER.PowerControl(ARM_POWER_FULL);

/*config transmit speed*/
EXAMPLE_I2C_MASTER.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_STANDARD);

/*start transfer*/
EXAMPLE_I2C_MASTER.MasterReceive(I2C_MASTER_SLAVE_ADDR, g_master_buff, I2C_DATA_LENGTH, false);

/*  Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

15.8.1.3 Slave Operation in interrupt transactional method

```
void I2C_SlaveSignalEvent_t(uint32_t event)
{
    /*  Transfer done */
    if (event == ARM_I2C_EVENT_TRANSFER_DONE)
    {
        g_SlaveCompletionFlag = true;
    }
}

/*Init I2C SLAVE*/
EXAMPLE_I2C_SLAVE.Initialize(I2C_SlaveSignalEvent_t);

EXAMPLE_I2C_SLAVE.PowerControl(ARM_POWER_FULL);

/*config slave addr*/
EXAMPLE_I2C_SLAVE.Control(ARM_I2C_OWN_ADDRESS, I2C_MASTER_SLAVE_ADDR);

/*start transfer*/
EXAMPLE_I2C_SLAVE.SlaveReceive(g_slave_buff, I2C_DATA_LENGTH);

/*  Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
g_SlaveCompletionFlag = false;
```

Chapter 16
I2S: I2S Driver

16.1 Overview

The MCUXpresso SDK provides the peripheral driver for the I2S function of FLEXCOMM module of MCUXpresso SDK devices.

The I2S module is used to transmit or receive digital audio data. Only transmit or receive is enabled at one time in one module.

Driver currently supports one (primary) channel pair per one I2S enabled FLEXCOMM module only.

16.2 I2S Driver Initialization and Configuration

I2S_TxInit() and I2S_RxInit() functions ungate the clock for the FLEXCOMM module, assign I2S function to FLEXCOMM module and configure audio data format and other I2S operational settings. I2S_TxInit() is used when I2S should transmit data, I2S_RxInit() when it should receive data.

I2S_TxGetDefaultConfig() and I2S_RxGetDefaultConfig() functions can be used to set the module configuration structure with default values for transmit and receive function, respectively.

I2S_Deinit() function resets the FLEXCOMM module.

I2S_TxTransferCreateHandle() function creates transactional handle for transmit in interrupt mode.

I2S_RxTransferCreateHandle() function creates transactional handle for receive in interrupt mode.

I2S_TxTransferCreateHandleDMA() function creates transactional handle for transmit in DMA mode.

I2S_RxTransferCreateHandleDMA() function creates transactional handle for receive in DMA mode.

16.3 I2S Transmit Data

I2S_TxTransferNonBlocking() function is used to add data buffer to transmit in interrupt mode. It also begins transmission if not transmitting yet.

I2S_RxTransferNonBlocking() function is used to add data buffer to receive data into in interrupt mode. It also begins reception if not receiving yet.

I2S_TxTransferSendDMA() function is used to add data buffer to transmit in DMA mode. It also begins transmission if not transmitting yet.

I2S_RxTransferReceiveDMA() function is used to add data buffer to receive data into in DMA mode. It also begins reception if not receiving yet.

The transfer of data will be stopped automatically when all data buffers queued using the above functions will be processed and no new data buffer is enqueued meanwhile. If the above functions are not called frequently enough, I2S stop followed by restart may keep occurring resulting in drops audio stream.

16.4 I2S Interrupt related functions

I2S_EnableInterrupts() function is used to enable interrupts in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

I2S_DisableInterrupts() function is used to disable interrupts in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

I2S_GetEnabledInterrupts() function returns interrupts enabled in FIFO interrupt register. Regular use cases do not require this function to be called from application code.

I2S_TxHandleIRQ() and I2S_RxHandleIRQ() functions are called from ISR which is invoked when actual FIFO level decreases to configured watermark value.

I2S_DMACallback() function is called from ISR which is invoked when DMA transfer (actual descriptor) finishes.

16.5 I2S Other functions

I2S_Enable() function enables I2S function in FLEXCOMM module. Regular use cases do not require this function to be called from application code.

I2S_Disable() function disables I2S function in FLEXCOMM module. Regular use cases do not require this function to be called from application code.

I2S_TransferGetErrorCount() function returns the number of FIFO underruns or overruns in interrupt mode.

I2S_TransferGetCount() function returns the number of bytes transferred in interrupt mode.

I2S_TxTransferAbort() function aborts trasmit operation in interrupt mode.

I2S_RxTransferAbort() function aborts receive operation in interrupt mode.

I2S_TransferAbortDMA() function aborts transmit or receive operation in DMA mode.

16.6 I2S Data formats

16.6.1 DMA mode

Length of buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes. Data are put into or taken from FIFO unaltered in DMA mode so buffer has to be prepared according to following information. If oneChannel is enabled, then the buffer should contain valid data only, that is to say, audio data should be put continuously in the buffer Take 8 bit data as example:

```
LSB
L07
L06
L05
L04
L03
```

L02

L01

L00

L07

L06

L05

L04

L03

L02

L01

L00

L07

L06

L05

L04

L03

L02

L01

L00

L07

L06

L05

L04

L03

L02

L01

L00

If i2s_config_t.dataLength (channel bit width) is between 4 and 16, every word in buffer should contain data for left and right channels.

MSB																													**LSB**
R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Rnn - right channel bit nn

Lnn - left channel bit nn

Note that for example if i2s_config_t.dataLength = 7, bits on positions R07-R15 and L07-L15 are ignored (buffer "wastes space").

If i2s_config_t.dataLength (channel bit width) is between 17 and 24 and i2s_config_t.pack48 = false:

Even words (counting from zero):

MSB																																LSB
								L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Odd words (counting from zero):

MSB																																LSB
								R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00

If i2s_config_t.dataLength (channel bit width) is between 17 and 24 and i2s_config_t.pack48 = true:

Even words (counting from zero):

MSB																																LSB
R07	R06	R05	R04	R03	R02	R01	R00	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Odd words (counting from zero):

MSB																																LSB
																R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08

If i2s_config_t.dataLength (channel bit width) is between 25 and 32:

Even words (counting from zero):

MSB																																LSB
L31	L30	L29	L28	L27	L26	L25	L24	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Odd words (counting from zero):

MSB																																LSB
R31	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00

16.6.2 Interrupt mode

If i2s_config_t.dataLength (channel bit width) is 4:

Buffer does not need to be aligned (buffer is read / written by single bytes, each byte contain left and right channel):

MSB							**LSB**
R03	R02	R01	R00	L03	L02	L01	L00

If i2s_config_t.dataLength (channel bit width) is between 5 and 8:

Length of buffer for transmit or receive has to be multiply of 2 bytes. Buffer address has to be aligned to 2-bytes.

MSB															**LSB**
R07	R06	R05	R04	R03	R02	R01	R00	L07	L06	L05	L04	L03	L02	L01	L00

If i2s_config_t.dataLength (channel bit width) is between 9 and 16:

Length of buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes.

MSB																													**LSB**
R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

If i2s_config_t.dataLength (channel bit width) is between 17 and 24 and i2s_config_t.pack48 = false(mono channel audio data is supported):

Length of buffer for transmit or receive has to be multiply of 6 bytes.

MSB																																									
R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	

If i2s_config_t.dataLength (channel bit width) is between 17 and 24 and i2s_config_t.pack48 = true:

Length of buffer for transmit or receive has to be multiply of 6 bytes. Buffer address has to be aligned to 4-bytes.

MSB																																									
R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	

If i2s_config_t.dataLength (channel bit width) is between 25 and 32 and i2s_config_t.oneChannel = false:

Buffer for transmit or receive has to be multiply of 8 bytes. Buffer address has to be aligned to 4-bytes.

Even words (counting from zero):

MSB																																LSB
L31	L30	L29	L28	L27	L26	L25	L24	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

Odd words (counting from zero):

MSB																																LSB
R31	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R09	R08	R07	R06	R05	R04	R03	R02	R01	R00

If i2s_config_t.dataLength (channel bit width) is between 25 and 32 and i2s_config_t.oneChannel = true:

Buffer for transmit or receive has to be multiply of 4 bytes. Buffer address has to be aligned to 4-bytes.

MSB																																LSB
L31	L30	L29	L28	L27	L26	L25	L24	L23	L22	L21	L20	L19	L18	L17	L16	L15	L14	L13	L12	L11	L10	L09	L08	L07	L06	L05	L04	L03	L02	L01	L00

16.7 I2S Driver Examples

16.7.1 Interrupt mode examples

Transmit example

```
void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_handle_t handle;

    I2S_TxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_TxInit(I2S0, &config);

    I2S_TxTransferCreateHandle(I2S0, &handle, TxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_TxTransferNonBlocking(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
       finishes */
    I2S_TxTransferNonBlocking(I2S0, &handle, someTransfer);
}

void TxCallback(I2S_Type *base, i2s_handle_t *handle, status_t completionStatus, void *userData)
{
    i2s_tranfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
```

```
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_TxTransferNonBlocking(base, handle, transfer);
    }
}
```

Receive example

```
void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_handle_t handle;

    I2S_RxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_RxInit(I2S0, &config);

    I2S_RxTransferCreateHandle(I2S0, &handle, RxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_RxTransferNonBlocking(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
       finishes */
    I2S_RxTransferNonBlocking(I2S0, &handle, someTransfer);
}

void RxCallback(I2S_Type *base, i2s_handle_t *handle, status_t completionStatus, void *userData)
{
    i2s_tranfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_RxTransferNonBlocking(base, handle, transfer);
    }
}
```

16.7.2 DMA mode examples

Transmit example

```
void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_dma_handle_t handle;

    I2S_TxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_TxInit(I2S0, &config);

    I2S_TxTransferCreateHandleDMA(I2S0, &handle, TxCallback, NULL);
```

```
    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_TxTransferNonBlockingDMA(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
       finishes */
    I2S_TxTransferNonBlockingDMA(I2S0, &handle, someTransfer);
}

void TxCallback(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData
      )
{
    i2s_tranfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_TxTransferNonBlockingDMA(base, handle, transfer);
    }
}
```

Receive example

```
void StartTransfer(void)
{
    i2s_config_t config;
    i2s_transfer_t transfer;
    i2s_dma_handle_t handle;

    I2S_RxGetDefaultConfig(&config);
    config.masterSlave = kI2S_MasterSlaveNormalMaster;
    config.divider = 32; /* clock frequency/audio sample frequency/channels/channel bit depth */
    I2S_RxInit(I2S0, &config);

    I2S_RxTransferCreateHandleDMA(I2S0, &handle, RxCallback, NULL);

    transfer.data = buffer;
    transfer.dataSize = sizeof(buffer);
    I2S_RxTransferNonBlockingDMA(I2S0, &handle, transfer);

    /* Enqueue next buffer right away so there is no drop in audio data stream when the first buffer
       finishes */
    I2S_RxTransferNonBlockingDMA(I2S0, &handle, someTransfer);
}

void RxCallback(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData
      )
{
    i2s_tranfer_t transfer;

    if (completionStatus == kStatus_I2S_BufferComplete)
    {
        /* Enqueue next buffer */
        transfer.data = buffer;
        transfer.dataSize = sizeof(buffer);
        I2S_RxTransferNonBlockingDMA(base, handle, transfer);
    }
}
```

Modules

- I2S Driver

16.8 I2S Driver

16.8.1 Overview

Files

- file fsl_i2s.h

Data Structures

- struct i2s_config_t
  I2S configuration structure. More...
- struct i2s_transfer_t
  Buffer to transfer from or receive audio data into. More...
- struct i2s_handle_t
  Members not to be accessed / modified outside of the driver. More...

Macros

- #define I2S_NUM_BUFFERS (4U)
  Number of buffers .

Typedefs

- typedef void(∗ i2s_transfer_callback_t)(I2S_Type ∗base, i2s_handle_t ∗handle, status_t completionStatus, void ∗userData)
  Callback function invoked from transactional API on completion of a single buffer transfer.

Enumerations

- enum {
  kStatus_I2S_BufferComplete,
  kStatus_I2S_Done = MAKE_STATUS(kStatusGroup_I2S, 1),
  kStatus_I2S_Busy }
  _i2s_status I2S status codes.
- enum i2s_flags_t {
  kI2S_TxErrorFlag = I2S_FIFOINTENSET_TXERR_MASK,
  kI2S_TxLevelFlag = I2S_FIFOINTENSET_TXLVL_MASK,
  kI2S_RxErrorFlag = I2S_FIFOINTENSET_RXERR_MASK,
  kI2S_RxLevelFlag = I2S_FIFOINTENSET_RXLVL_MASK }
  I2S flags.

- enum i2s_master_slave_t {
  kI2S_MasterSlaveNormalSlave = 0x0,
  kI2S_MasterSlaveWsSyncMaster = 0x1,
  kI2S_MasterSlaveExtSckMaster = 0x2,
  kI2S_MasterSlaveNormalMaster = 0x3 }
  Master / slave mode.
- enum i2s_mode_t {
  kI2S_ModeI2sClassic = 0x0,
  kI2S_ModeDspWs50 = 0x1,
  kI2S_ModeDspWsShort = 0x2,
  kI2S_ModeDspWsLong = 0x3 }
  I2S mode.
- enum {
  kI2S_SecondaryChannel1 = 0U,
  kI2S_SecondaryChannel2 = 1U,
  kI2S_SecondaryChannel3 = 2U }
  _i2s_secondary_channel I2S secondary channel.

Driver version

- #define FSL_I2S_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))
  I2S driver version 2.2.2.

Initialization and deinitialization

- void I2S_TxInit (I2S_Type ∗base, const i2s_config_t ∗config)
  Initializes the FLEXCOMM peripheral for I2S transmit functionality.
- void I2S_RxInit (I2S_Type ∗base, const i2s_config_t ∗config)
  Initializes the FLEXCOMM peripheral for I2S receive functionality.
- void I2S_TxGetDefaultConfig (i2s_config_t ∗config)
  Sets the I2S Tx configuration structure to default values.
- void I2S_RxGetDefaultConfig (i2s_config_t ∗config)
  Sets the I2S Rx configuration structure to default values.
- void I2S_Deinit (I2S_Type ∗base)
  De-initializes the I2S peripheral.
- void I2S_SetBitClockRate (I2S_Type ∗base, uint32_t sourceClockHz, uint32_t sampleRate, uint32_t bitWidth, uint32_t channelNumbers)
  Transmitter/Receiver bit clock rate configurations.

Non-blocking API

- void I2S_TxTransferCreateHandle (I2S_Type ∗base, i2s_handle_t ∗handle, i2s_transfer_callback_t callback, void ∗userData)
  Initializes handle for transfer of audio data.

- status_t I2S_TxTransferNonBlocking (I2S_Type *base, i2s_handle_t *handle, i2s_transfer_t transfer)
  Begins or queue sending of the given data.
- void I2S_TxTransferAbort (I2S_Type *base, i2s_handle_t *handle)
  Aborts sending of data.
- void I2S_RxTransferCreateHandle (I2S_Type *base, i2s_handle_t *handle, i2s_transfer_callback_t callback, void *userData)
  Initializes handle for reception of audio data.
- status_t I2S_RxTransferNonBlocking (I2S_Type *base, i2s_handle_t *handle, i2s_transfer_t transfer)
  Begins or queue reception of data into given buffer.
- void I2S_RxTransferAbort (I2S_Type *base, i2s_handle_t *handle)
  Aborts receiving of data.
- status_t I2S_TransferGetCount (I2S_Type *base, i2s_handle_t *handle, size_t *count)
  Returns number of bytes transferred so far.
- status_t I2S_TransferGetErrorCount (I2S_Type *base, i2s_handle_t *handle, size_t *count)
  Returns number of buffer underruns or overruns.

Enable / disable

- static void I2S_Enable (I2S_Type *base)
  Enables I2S operation.
- void I2S_EnableSecondaryChannel (I2S_Type *base, uint32_t channel, bool oneChannel, uint32_t position)
  Enables I2S secondary channel.
- static void I2S_DisableSecondaryChannel (I2S_Type *base, uint32_t channel)
  Disables I2S secondary channel.
- static void I2S_Disable (I2S_Type *base)
  Disables I2S operation.

Interrupts

- static void I2S_EnableInterrupts (I2S_Type *base, uint32_t interruptMask)
  Enables I2S FIFO interrupts.
- static void I2S_DisableInterrupts (I2S_Type *base, uint32_t interruptMask)
  Disables I2S FIFO interrupts.
- static uint32_t I2S_GetEnabledInterrupts (I2S_Type *base)
  Returns the set of currently enabled I2S FIFO interrupts.
- status_t I2S_EmptyTxFifo (I2S_Type *base)
  Flush the valid data in TX fifo.
- void I2S_TxHandleIRQ (I2S_Type *base, i2s_handle_t *handle)
  Invoked from interrupt handler when transmit FIFO level decreases.
- void I2S_RxHandleIRQ (I2S_Type *base, i2s_handle_t *handle)
  Invoked from interrupt handler when receive FIFO level decreases.

16.8.2 Data Structure Documentation

16.8.2.1 struct i2s_config_t

Data Fields

- i2s_master_slave_t masterSlave
  Master / slave configuration.
- i2s_mode_t mode
  I2S mode.
- bool rightLow
  Right channel data in low portion of FIFO.
- bool leftJust
  Left justify data in FIFO.
- bool pdmData
  Data source is the D-Mic subsystem.
- bool sckPol
  SCK polarity.
- bool wsPol
  WS polarity.
- uint16_t divider
  Flexcomm function clock divider (1 - 4096)
- bool oneChannel
  true mono, false stereo
- uint8_t dataLength
  Data length (4 - 32)
- uint16_t frameLength
  Frame width (4 - 512)
- uint16_t position
  Data position in the frame.
- uint8_t watermark
  FIFO trigger level.
- bool txEmptyZero
  Transmit zero when buffer becomes empty or last item.
- bool pack48
  Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)

16.8.2.2 struct i2s_transfer_t

Data Fields

- uint8_t ∗ data
  Pointer to data buffer.
- size_t dataSize
  Buffer size in bytes.

Field Documentation

(1) uint8_t∗ i2s_transfer_t::data

(2) size_t i2s_transfer_t::dataSize

16.8.2.3 struct _i2s_handle

Transactional state of the intialized transfer or receive I2S operation.

Data Fields

- volatile uint32_t state
  State of transfer.
- i2s_transfer_callback_t completionCallback
  Callback function pointer.
- void ∗ userData
  Application data passed to callback.
- bool oneChannel
  true mono, false stereo
- uint8_t dataLength
  Data length (4 - 32)
- bool pack48
  Packing format for 48-bit data (false - 24 bit values, true - alternating 32-bit and 16-bit values)
- uint8_t watermark
  FIFO trigger level.
- bool useFifo48H
  When dataLength 17-24: true use FIFOWR48H, false use FIFOWR.
- volatile i2s_transfer_t i2sQueue [I2S_NUM_BUFFERS]
  Transfer queue storing transfer buffers.
- volatile uint8_t queueUser
  Queue index where user's next transfer will be stored.
- volatile uint8_t queueDriver
  Queue index of buffer actually used by the driver.
- volatile uint32_t errorCount
  Number of buffer underruns/overruns.
- volatile uint32_t transferCount
  Number of bytes transferred.

16.8.3 Macro Definition Documentation

16.8.3.1 #define FSL_I2S_DRIVER_VERSION (MAKE_VERSION(2, 2, 2))

16.8.3.2 #define I2S_NUM_BUFFERS (4U)

16.8.4 Typedef Documentation

16.8.4.1 typedef void(∗ i2s_transfer_callback_t)(I2S_Type ∗base, i2s_handle_t ∗handle, status_t completionStatus, void ∗userData)

Parameters

base	I2S base pointer.
handle	pointer to I2S transaction.
completion-Status	status of the transaction.
userData	optional pointer to user arguments data.

16.8.5 Enumeration Type Documentation

16.8.5.1 anonymous enum

Enumerator

kStatus_I2S_BufferComplete Transfer from/into a single buffer has completed.
kStatus_I2S_Done All buffers transfers have completed.
kStatus_I2S_Busy Already performing a transfer and cannot queue another buffer.

16.8.5.2 enum i2s_flags_t

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI2S_TxErrorFlag TX error interrupt.
kI2S_TxLevelFlag TX level interrupt.
kI2S_RxErrorFlag RX error interrupt.
kI2S_RxLevelFlag RX level interrupt.

16.8.5.3 enum i2s_master_slave_t

Enumerator

kI2S_MasterSlaveNormalSlave Normal slave.
kI2S_MasterSlaveWsSyncMaster WS synchronized master.
kI2S_MasterSlaveExtSckMaster Master using existing SCK.
kI2S_MasterSlaveNormalMaster Normal master.

16.8.5.4 enum i2s_mode_t

Enumerator

kI2S_ModeI2sClassic I2S classic mode.
kI2S_ModeDspWs50 DSP mode, WS having 50% duty cycle.
kI2S_ModeDspWsShort DSP mode, WS having one clock long pulse.
kI2S_ModeDspWsLong DSP mode, WS having one data slot long pulse.

16.8.5.5 anonymous enum

Enumerator

kI2S_SecondaryChannel1 secondary channel 1
kI2S_SecondaryChannel2 secondary channel 2
kI2S_SecondaryChannel3 secondary channel 3

16.8.6 Function Documentation

16.8.6.1 void I2S_TxInit (I2S_Type ∗ *base*, const i2s_config_t ∗ *config*)

Ungates the FLEXCOMM clock and configures the module for I2S transmission using a configuration structure. The configuration structure can be custom filled or set with default values by I2S_TxGetDefaultConfig().

Note

This API should be called at the beginning of the application to use the I2S driver.

Parameters

base	I2S base pointer.
config	pointer to I2S configuration structure.

16.8.6.2 void I2S_RxInit (I2S_Type ∗ *base*, const i2s_config_t ∗ *config*)

Ungates the FLEXCOMM clock and configures the module for I2S receive using a configuration structure. The configuration structure can be custom filled or set with default values by I2S_RxGetDefaultConfig().

Note

This API should be called at the beginning of the application to use the I2S driver.

Parameters

base	I2S base pointer.
config	pointer to I2S configuration structure.

16.8.6.3 void I2S_TxGetDefaultConfig (i2s_config_t * *config*)

This API initializes the configuration structure for use in I2S_TxInit(). The initialized structure can remain unchanged in I2S_TxInit(), or it can be modified before calling I2S_TxInit(). Example:

```
i2s_config_t config;
I2S_TxGetDefaultConfig(&config);
```

Default values:

```
*   config->masterSlave = kI2S_MasterSlaveNormalMaster;
*   config->mode = kI2S_ModeI2sClassic;
*   config->rightLow = false;
*   config->leftJust = false;
*   config->pdmData = false;
*   config->sckPol = false;
*   config->wsPol = false;
*   config->divider = 1;
*   config->oneChannel = false;
*   config->dataLength = 16;
*   config->frameLength = 32;
*   config->position = 0;
*   config->watermark = 4;
*   config->txEmptyZero = true;
*   config->pack48 = false;
*
```

Parameters

config	pointer to I2S configuration structure.

16.8.6.4 void I2S_RxGetDefaultConfig (i2s_config_t * *config*)

This API initializes the configuration structure for use in I2S_RxInit(). The initialized structure can remain unchanged in I2S_RxInit(), or it can be modified before calling I2S_RxInit(). Example:

```
i2s_config_t config;
I2S_RxGetDefaultConfig(&config);
```

Default values:

```
*   config->masterSlave = kI2S_MasterSlaveNormalSlave;
*   config->mode = kI2S_ModeI2sClassic;
*   config->rightLow = false;
*   config->leftJust = false;
*   config->pdmData = false;
*   config->sckPol = false;
*   config->wsPol = false;
*   config->divider = 1;
*   config->oneChannel = false;
*   config->dataLength = 16;
*   config->frameLength = 32;
*   config->position = 0;
*   config->watermark = 4;
*   config->txEmptyZero = false;
*   config->pack48 = false;
*
```

Parameters

config	pointer to I2S configuration structure.

16.8.6.5 void I2S_Deinit (I2S_Type ∗ *base*)

This API gates the FLEXCOMM clock. The I2S module can't operate unless I2S_TxInit or I2S_RxInit is called to enable the clock.

Parameters

base	I2S base pointer.

16.8.6.6 void I2S_SetBitClockRate (I2S_Type ∗ *base*, uint32_t *sourceClockHz*, uint32_t *sampleRate*, uint32_t *bitWidth*, uint32_t *channelNumbers*)

Parameters

base	SAI base pointer.
sourceClockHz	bit clock source frequency.
sampleRate	audio data sample rate.
bitWidth	audio data bitWidth.
channel-Numbers	audio channel numbers.

16.8.6.7 void I2S_TxTransferCreateHandle (I2S_Type ∗ *base*, i2s_handle_t ∗ *handle*, i2s_transfer_callback_t *callback*, void ∗ *userData*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
callback	function to be called back when transfer is done or fails.
userData	pointer to data passed to callback.

16.8.6.8 status_t I2S_TxTransferNonBlocking (I2S_Type * *base*, i2s_handle_t * *handle*, i2s_transfer_t *transfer*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
transfer	data buffer.

Return values

kStatus_Success	
kStatus_I2S_Busy	if all queue slots are occupied with unsent buffers.

16.8.6.9 void I2S_TxTransferAbort (I2S_Type * *base*, i2s_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

16.8.6.10 void I2S_RxTransferCreateHandle (I2S_Type * *base*, i2s_handle_t * *handle*, i2s_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
callback	function to be called back when transfer is done or fails.
userData	pointer to data passed to callback.

16.8.6.11 status_t I2S_RxTransferNonBlocking (I2S_Type * *base*, i2s_handle_t * *handle*, i2s_transfer_t *transfer*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
transfer	data buffer.

Return values

kStatus_Success	
kStatus_I2S_Busy	if all queue slots are occupied with buffers which are not full.

16.8.6.12 void I2S_RxTransferAbort (I2S_Type * *base*, i2s_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

16.8.6.13 status_t I2S_TransferGetCount (I2S_Type * *base*, i2s_handle_t * *handle*, size_t * *count*)

Parameters

	base	I2S base pointer.
	handle	pointer to handle structure.
`out`	*count*	number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_NoTransferInProgress	there is no non-blocking transaction currently in progress.

16.8.6.14 status_t I2S_TransferGetErrorCount (I2S_Type * *base*, i2s_handle_t * *handle*, size_t * *count*)

Parameters

	base	I2S base pointer.
	handle	pointer to handle structure.
`out`	*count*	number of transmit errors encountered so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_NoTransferInProgress	there is no non-blocking transaction currently in progress.

16.8.6.15 static void I2S_Enable (I2S_Type * *base*) `[inline], [static]`

Parameters

base	I2S base pointer.

16.8.6.16 void I2S_EnableSecondaryChannel (I2S_Type * *base*, uint32_t *channel*, bool *oneChannel*, uint32_t *position*)

Parameters

base	I2S base pointer.
channel	seondary channel channel number, reference _i2s_secondary_channel.
oneChannel	true is treated as single channel, functionality left channel for this pair.
position	define the location within the frame of the data, should not bigger than 0x1FFU.

16.8.6.17 static void I2S_DisableSecondaryChannel (I2S_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	I2S base pointer.
channel	seondary channel channel number, reference _i2s_secondary_channel.

16.8.6.18 static void I2S_Disable (I2S_Type * *base*) [inline], [static]

Parameters

base	I2S base pointer.

16.8.6.19 static void I2S_EnableInterrupts (I2S_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	I2S base pointer.
interruptMask	bit mask of interrupts to enable. See i2s_flags_t for the set of constants that should be OR'd together to form the bit mask.

16.8.6.20 static void I2S_DisableInterrupts (I2S_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Parameters

base	I2S base pointer.
interruptMask	bit mask of interrupts to enable. See i2s_flags_t for the set of constants that should be OR'd together to form the bit mask.

16.8.6.21 static uint32_t I2S_GetEnabledInterrupts (I2S_Type * *base*) [inline], [static]

Parameters

base	I2S base pointer.

Returns

A bitmask composed of i2s_flags_t enumerators OR'd together to indicate the set of enabled interrupts.

16.8.6.22 status_t I2S_EmptyTxFifo (I2S_Type * *base*)

Parameters

base	I2S base pointer.

Returns

kStatus_Fail empty TX fifo failed, kStatus_Success empty tx fifo success.

16.8.6.23 void I2S_TxHandleIRQ (I2S_Type * *base,* i2s_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

16.8.6.24 void I2S_RxHandleIRQ (I2S_Type * *base,* i2s_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

Chapter 17
SPI: Serial Peripheral Interface Driver

17.1 Overview

SPI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the spi_handle_t as the first parameter. Initialize the handle by calling the SPI_MasterTransferCreateHandle() or SPI_SlaveTransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions SPI_MasterTransferNonBlocking() and SPI_SlaveTransferNonBlocking() set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_SPI_Idle status.

17.2 Typical use case

17.2.1 SPI master transfer using an interrupt method

```
#define BUFFER_LEN (64)
spi_master_handle_t spiHandle;
spi_master_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished = false;

const uint8_t sendData[BUFFER_LEN] = [......];
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_master_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    SPI_MasterGetDefaultConfig(&masterConfig);

    SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);
    SPI_MasterTransferCreateHandle(SPI0, &spiHandle, SPI_UserCallback, NULL);

    // Prepare to send.
```

```
    xfer.txData = sendData;
    xfer.rxData = receiveBuff;
    xfer.dataSize = sizeof(sendData);

    // Send out.
    SPI_MasterTransferNonBlocking(SPI0, &spiHandle, &xfer);

    // Wait send finished.
    while (!isFinished)
    {
    }

    // ...
}
```

17.2.2 SPI Send/receive using a DMA method

```
#define BUFFER_LEN (64)
spi_dma_handle_t spiHandle;
dma_handle_t g_spiTxDmaHandle;
dma_handle_t g_spiRxDmaHandle;
spi_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished;

uint8_t sendData[BUFFER_LEN] = ...;
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)
{
    isFinished = true;
}

void main(void)
{
    //...

    // Initialize DMA peripheral
    DMA_Init(DMA0);

    // Initialize SPI peripheral
    SPI_MasterGetDefaultConfig(&masterConfig);
    masterConfig.sselNum = SPI_SSEL;
    SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);

    // Enable DMA channels connected to SPI0 Tx/SPI0 Rx request lines
    DMA_EnableChannel(SPI0, SPI_MASTER_TX_CHANNEL);
    DMA_EnableChannel(SPI0, SPI_MASTER_RX_CHANNEL);

    // Set DMA channels priority
    DMA_SetChannelPriority(SPI0, SPI_MASTER_TX_CHANNEL,
      kDMA_ChannelPriority3);
    DMA_SetChannelPriority(SPI0, SPI_MASTER_RX_CHANNEL,
      kDMA_ChannelPriority2);

    // Creates the DMA handle.
    DMA_CreateHandle(&masterTxHandle, SPI0, SPI_MASTER_TX_CHANNEL);
    DMA_CreateHandle(&masterRxHandle, SPI0, SPI_MASTER_RX_CHANNEL);

    // Create SPI DMA handle
    SPI_MasterTransferCreateHandleDMA(SPI0, spiHandle, SPI_UserCallback,
      NULL, &g_spiTxDmaHandle, &g_spiRxDmaHandle);

    // Prepares to send.
    xfer.txData = sendData;
```

```
    xfer.rxData = receiveBuff;
    xfer.dataSize = sizeof(sendData);

    // Sends out.
    SPI_MasterTransferDMA(SPI0, &spiHandle, &xfer);

    // Waits for send to complete.
    while (!isFinished)
    {
    }

    // ...
}
```

Modules

- SPI CMSIS driver
- SPI DMA Driver
- SPI Driver
- SPI FreeRTOS driver

17.3 SPI Driver

17.3.1 Overview

This section describes the programming interface of the SPI DMA driver.

Files

- file fsl_spi.h

Data Structures

- struct spi_delay_config_t
  SPI delay time configure structure. More...
- struct spi_master_config_t
  SPI master user configure structure. More...
- struct spi_slave_config_t
  SPI slave user configure structure. More...
- struct spi_transfer_t
  SPI transfer structure. More...
- struct spi_half_duplex_transfer_t
  SPI half-duplex(master only) transfer structure. More...
- struct spi_config_t
  Internal configuration structure used in 'spi' and 'spi_dma' driver. More...
- struct spi_master_handle_t
  SPI transfer handle structure. More...

Macros

- #define SPI_DUMMYDATA (0xFFU)
  SPI dummy transfer data, the data is sent while txBuff is NULL.
- #define SPI_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef spi_master_handle_t spi_slave_handle_t
  Slave handle type.
- typedef void(∗ spi_master_callback_t)(SPI_Type ∗base, spi_master_handle_t ∗handle, status_t status, void ∗userData)
  SPI master callback for finished transmit.
- typedef void(∗ spi_slave_callback_t)(SPI_Type ∗base, spi_slave_handle_t ∗handle, status_t status, void ∗userData)
  SPI slave callback for finished transmit.

- typedef void(∗ flexcomm_spi_master_irq_handler_t)(SPI_Type ∗base, spi_master_handle_-t ∗handle)
  Typedef for master interrupt handler.
- typedef void(∗ flexcomm_spi_slave_irq_handler_t)(SPI_Type ∗base, spi_slave_handle_t ∗handle)
  Typedef for slave interrupt handler.

Enumerations

- enum spi_xfer_option_t {
  kSPI_FrameDelay = (SPI_FIFOWR_EOF_MASK),
  kSPI_FrameAssert = (SPI_FIFOWR_EOT_MASK) }
  SPI transfer option.
- enum spi_shift_direction_t {
  kSPI_MsbFirst = 0U,
  kSPI_LsbFirst = 1U }
  SPI data shifter direction options.
- enum spi_clock_polarity_t {
  kSPI_ClockPolarityActiveHigh = 0x0U,
  kSPI_ClockPolarityActiveLow }
  SPI clock polarity configuration.
- enum spi_clock_phase_t {
  kSPI_ClockPhaseFirstEdge = 0x0U,
  kSPI_ClockPhaseSecondEdge }
  SPI clock phase configuration.
- enum spi_txfifo_watermark_t {
  kSPI_TxFifo0 = 0,
  kSPI_TxFifo1 = 1,
  kSPI_TxFifo2 = 2,
  kSPI_TxFifo3 = 3,
  kSPI_TxFifo4 = 4,
  kSPI_TxFifo5 = 5,
  kSPI_TxFifo6 = 6,
  kSPI_TxFifo7 = 7 }
  txFIFO watermark values
- enum spi_rxfifo_watermark_t {
  kSPI_RxFifo1 = 0,
  kSPI_RxFifo2 = 1,
  kSPI_RxFifo3 = 2,
  kSPI_RxFifo4 = 3,
  kSPI_RxFifo5 = 4,
  kSPI_RxFifo6 = 5,
  kSPI_RxFifo7 = 6,
  kSPI_RxFifo8 = 7 }
  rxFIFO watermark values
- enum spi_data_width_t {

kSPI_Data4Bits = 3,
kSPI_Data5Bits = 4,
kSPI_Data6Bits = 5,
kSPI_Data7Bits = 6,
kSPI_Data8Bits = 7,
kSPI_Data9Bits = 8,
kSPI_Data10Bits = 9,
kSPI_Data11Bits = 10,
kSPI_Data12Bits = 11,
kSPI_Data13Bits = 12,
kSPI_Data14Bits = 13,
kSPI_Data15Bits = 14,
kSPI_Data16Bits = 15 }

Transfer data width.

- enum spi_ssel_t {
  kSPI_Ssel0 = 0,
  kSPI_Ssel1 = 1,
  kSPI_Ssel2 = 2,
  kSPI_Ssel3 = 3 }

  Slave select.
- enum spi_spol_t

  ssel polarity
- enum {
  kStatus_SPI_Busy = MAKE_STATUS(kStatusGroup_LPC_SPI, 0),
  kStatus_SPI_Idle = MAKE_STATUS(kStatusGroup_LPC_SPI, 1),
  kStatus_SPI_Error = MAKE_STATUS(kStatusGroup_LPC_SPI, 2),
  kStatus_SPI_BaudrateNotSupport,
  kStatus_SPI_Timeout = MAKE_STATUS(kStatusGroup_LPC_SPI, 4) }

  SPI transfer status.
- enum _spi_interrupt_enable {
  kSPI_RxLvlIrq = SPI_FIFOINTENSET_RXLVL_MASK,
  kSPI_TxLvlIrq = SPI_FIFOINTENSET_TXLVL_MASK }

  SPI interrupt sources.
- enum _spi_statusflags {
  kSPI_TxEmptyFlag = SPI_FIFOSTAT_TXEMPTY_MASK,
  kSPI_TxNotFullFlag = SPI_FIFOSTAT_TXNOTFULL_MASK,
  kSPI_RxNotEmptyFlag = SPI_FIFOSTAT_RXNOTEMPTY_MASK,
  kSPI_RxFullFlag = SPI_FIFOSTAT_RXFULL_MASK }

  SPI status flags.

Variables

- volatile uint8_t s_dummyData [ ]

  Global variable for dummy data value setting.

Driver version

- #define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
  SPI driver version.

17.3.2 Data Structure Documentation

17.3.2.1 struct spi_delay_config_t

Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maxinun value of these delay time is 15.

Data Fields

- uint8_t preDelay
  Delay between SSEL assertion and the beginning of transfer.
- uint8_t postDelay
  Delay between the end of transfer and SSEL deassertion.
- uint8_t frameDelay
  Delay between frame to frame.
- uint8_t transferDelay
  Delay between transfer to transfer.

Field Documentation

(1) uint8_t spi_delay_config_t::preDelay

(2) uint8_t spi_delay_config_t::postDelay

(3) uint8_t spi_delay_config_t::frameDelay

(4) uint8_t spi_delay_config_t::transferDelay

17.3.2.2 struct spi_master_config_t

Data Fields

- bool enableLoopback
  Enable loopback for test purpose.
- bool enableMaster
  Enable SPI at initialization time.
- spi_clock_polarity_t polarity
  Clock polarity.
- spi_clock_phase_t phase
  Clock phase.
- spi_shift_direction_t direction
  MSB or LSB.
- uint32_t baudRate_Bps

Baud Rate for SPI in Hz.
- spi_data_width_t dataWidth
  Width of the data.
- spi_ssel_t sselNum
  Slave select number.
- spi_spol_t sselPol
  Configure active CS polarity.
- uint8_t txWatermark
  txFIFO watermark
- uint8_t rxWatermark
  rxFIFO watermark
- spi_delay_config_t delayConfig
  Delay configuration.

Field Documentation

(1) spi_delay_config_t spi_master_config_t::delayConfig

17.3.2.3 struct spi_slave_config_t

Data Fields

- bool enableSlave
  Enable SPI at initialization time.
- spi_clock_polarity_t polarity
  Clock polarity.
- spi_clock_phase_t phase
  Clock phase.
- spi_shift_direction_t direction
  MSB or LSB.
- spi_data_width_t dataWidth
  Width of the data.
- spi_spol_t sselPol
  Configure active CS polarity.
- uint8_t txWatermark
  txFIFO watermark
- uint8_t rxWatermark
  rxFIFO watermark

17.3.2.4 struct spi_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- uint32_t configFlags
  Additional option to control transfer, spi_xfer_option_t.
- size_t dataSize

Transfer bytes.

Field Documentation

(1) uint32_t spi_transfer_t::configFlags

17.3.2.5 struct spi_half_duplex_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- size_t txDataSize
  Transfer bytes for transmit.
- size_t rxDataSize
  Transfer bytes.
- uint32_t configFlags
  Transfer configuration flags, spi_xfer_option_t.
- bool isPcsAssertInTransfer
  If PCS pin keep assert between transmit and receive.
- bool isTransmitFirst
  True for transmit first and false for receive first.

Field Documentation

(1) uint32_t spi_half_duplex_transfer_t::configFlags

(2) bool spi_half_duplex_transfer_t::isPcsAssertInTransfer

true for assert and false for deassert.

(3) bool spi_half_duplex_transfer_t::isTransmitFirst

17.3.2.6 struct spi_config_t

17.3.2.7 struct _spi_master_handle

Master handle type.

Data Fields

- uint8_t *volatile txData
  Transfer buffer.
- uint8_t *volatile rxData
  Receive buffer.
- volatile size_t txRemainingBytes
  Number of data to be transmitted [in bytes].
- volatile size_t rxRemainingBytes

Number of data to be received [in bytes].
- volatile int8_t toReceiveCount
  The number of data expected to receive in data width.
- size_t totalByteCount
  A number of transfer bytes.
- volatile uint32_t state
  SPI internal state.
- spi_master_callback_t callback
  SPI callback.
- void ∗ userData
  Callback parameter.
- uint8_t dataWidth
  Width of the data [Valid values: 1 to 16].
- uint8_t sselNum
  Slave select number to be asserted when transferring data [Valid values: 0 to 3].
- uint32_t configFlags
  Additional option to control transfer.
- uint8_t txWatermark
  txFIFO watermark
- uint8_t rxWatermark
  rxFIFO watermark

Field Documentation

(1) volatile int8_t spi_master_handle_t::toReceiveCount

Since the received count and sent count should be the same to complete the transfer, if the sent count is x and the received count is y, toReceiveCount is x-y.

17.3.3 Macro Definition Documentation

17.3.3.1 #define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

17.3.3.2 #define SPI_DUMMYDATA (0xFFU)

17.3.3.3 #define SPI_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/

17.3.4 Typedef Documentation

17.3.4.1 typedef void(∗ flexcomm_spi_master_irq_handler_t)(SPI_Type ∗base, spi_master_handle_t ∗handle)

17.3.4.2 typedef void(∗ flexcomm_spi_slave_irq_handler_t)(SPI_Type ∗base, spi_slave_handle_t ∗handle)

17.3.5 Enumeration Type Documentation

17.3.5.1 enum spi_xfer_option_t

Enumerator

kSPI_FrameDelay A delay may be inserted, defined in the DLY register.
kSPI_FrameAssert SSEL will be deasserted at the end of a transfer.

17.3.5.2 enum spi_shift_direction_t

Enumerator

kSPI_MsbFirst Data transfers start with most significant bit.
kSPI_LsbFirst Data transfers start with least significant bit.

17.3.5.3 enum spi_clock_polarity_t

Enumerator

kSPI_ClockPolarityActiveHigh Active-high SPI clock (idles low).
kSPI_ClockPolarityActiveLow Active-low SPI clock (idles high).

17.3.5.4 enum spi_clock_phase_t

Enumerator

kSPI_ClockPhaseFirstEdge First edge on SCK occurs at the middle of the first cycle of a data transfer.
kSPI_ClockPhaseSecondEdge First edge on SCK occurs at the start of the first cycle of a data transfer.

17.3.5.5 enum spi_txfifo_watermark_t

Enumerator

kSPI_TxFifo0 SPI tx watermark is empty.
kSPI_TxFifo1 SPI tx watermark at 1 item.
kSPI_TxFifo2 SPI tx watermark at 2 items.
kSPI_TxFifo3 SPI tx watermark at 3 items.
kSPI_TxFifo4 SPI tx watermark at 4 items.
kSPI_TxFifo5 SPI tx watermark at 5 items.
kSPI_TxFifo6 SPI tx watermark at 6 items.
kSPI_TxFifo7 SPI tx watermark at 7 items.

17.3.5.6 enum spi_rxfifo_watermark_t

Enumerator

kSPI_RxFifo1 SPI rx watermark at 1 item.
kSPI_RxFifo2 SPI rx watermark at 2 items.
kSPI_RxFifo3 SPI rx watermark at 3 items.
kSPI_RxFifo4 SPI rx watermark at 4 items.
kSPI_RxFifo5 SPI rx watermark at 5 items.
kSPI_RxFifo6 SPI rx watermark at 6 items.
kSPI_RxFifo7 SPI rx watermark at 7 items.
kSPI_RxFifo8 SPI rx watermark at 8 items.

17.3.5.7 enum spi_data_width_t

Enumerator

kSPI_Data4Bits 4 bits data width
kSPI_Data5Bits 5 bits data width
kSPI_Data6Bits 6 bits data width
kSPI_Data7Bits 7 bits data width
kSPI_Data8Bits 8 bits data width
kSPI_Data9Bits 9 bits data width
kSPI_Data10Bits 10 bits data width
kSPI_Data11Bits 11 bits data width
kSPI_Data12Bits 12 bits data width
kSPI_Data13Bits 13 bits data width
kSPI_Data14Bits 14 bits data width
kSPI_Data15Bits 15 bits data width
kSPI_Data16Bits 16 bits data width

17.3.5.8 enum spi_ssel_t

Enumerator

kSPI_Ssel0 Slave select 0.
kSPI_Ssel1 Slave select 1.
kSPI_Ssel2 Slave select 2.
kSPI_Ssel3 Slave select 3.

17.3.5.9 anonymous enum

Enumerator

kStatus_SPI_Busy SPI bus is busy.

kStatus_SPI_Idle SPI is idle.
kStatus_SPI_Error SPI error.
kStatus_SPI_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_SPI_Timeout SPI timeout polling status flags.

17.3.5.10 enum _spi_interrupt_enable

Enumerator

kSPI_RxLvlIrq Rx level interrupt.
kSPI_TxLvlIrq Tx level interrupt.

17.3.5.11 enum _spi_statusflags

Enumerator

kSPI_TxEmptyFlag txFifo is empty
kSPI_TxNotFullFlag txFifo is not full
kSPI_RxNotEmptyFlag rxFIFO is not empty
kSPI_RxFullFlag rxFIFO is full

17.3.6 Variable Documentation

17.3.6.1 volatile uint8_t s_dummyData[]

17.4 SPI DMA Driver

17.4.1 Overview

This section describes the programming interface of the SPI DMA driver.

Files

- file fsl_spi_dma.h

Data Structures

- struct spi_dma_handle_t
  SPI DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ spi_dma_callback_t)(SPI_Type ∗base, spi_dma_handle_t ∗handle, status_t status, void ∗userData)
  SPI DMA callback called at the end of transfer.

Driver version

- #define FSL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
  SPI DMA driver version 2.1.1.

DMA Transactional

- status_t SPI_MasterTransferCreateHandleDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  Initialize the SPI master DMA handle.
- status_t SPI_MasterTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_transfer_t ∗xfer)
  Perform a non-blocking SPI transfer using DMA.
- status_t SPI_MasterHalfDuplexTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_half_duplex_transfer_t ∗xfer)
  Transfers a block of data using a DMA method.
- static status_t SPI_SlaveTransferCreateHandleDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txHandle, dma_handle_t ∗rxHandle)
  Initialize the SPI slave DMA handle.
- static status_t SPI_SlaveTransferDMA (SPI_Type ∗base, spi_dma_handle_t ∗handle, spi_transfer_t ∗xfer)
  Perform a non-blocking SPI transfer using DMA.

- void SPI_MasterTransferAbortDMA (SPI_Type *base, spi_dma_handle_t *handle)
  Abort a SPI transfer using DMA.
- status_t SPI_MasterTransferGetCountDMA (SPI_Type *base, spi_dma_handle_t *handle, size_t *count)
  Gets the master DMA transfer remaining bytes.
- static void SPI_SlaveTransferAbortDMA (SPI_Type *base, spi_dma_handle_t *handle)
  Abort a SPI transfer using DMA.
- static status_t SPI_SlaveTransferGetCountDMA (SPI_Type *base, spi_dma_handle_t *handle, size_t *count)
  Gets the slave DMA transfer remaining bytes.

17.4.2 Data Structure Documentation

17.4.2.1 struct _spi_dma_handle

Data Fields

- volatile bool txInProgress
  Send transfer finished.
- volatile bool rxInProgress
  Receive transfer finished.
- dma_handle_t * txHandle
  DMA handler for SPI send.
- dma_handle_t * rxHandle
  DMA handler for SPI receive.
- uint8_t bytesPerFrame
  Bytes in a frame for SPI transfer.
- spi_dma_callback_t callback
  Callback for SPI DMA transfer.
- void * userData
  User Data for SPI DMA callback.
- uint32_t state
  Internal state of SPI DMA transfer.
- size_t transferSize
  Bytes need to be transfer.

17.4.3 Macro Definition Documentation

17.4.3.1 #define FSL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

17.4.4 Typedef Documentation

17.4.4.1 typedef void(* spi_dma_callback_t)(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)

17.4.5 Function Documentation

17.4.5.1 status_t SPI_MasterTransferCreateHandleDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *txHandle*, dma_handle_t * *rxHandle*)

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
rxHandle	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

17.4.5.2 status_t SPI_MasterTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_transfer_t * *xfer*)

Note

This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
xfer	Pointer to dma transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.

kStatus_SPI_Busy	SPI is not idle, is running another transfer.

17.4.5.3 status_t SPI_MasterHalfDuplexTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_half_duplex_transfer_t * *xfer*)

This function using polling way to do the first half transimission and using DMA way to do the srcond half transimission, the transfer mechanism is half-duplex. When do the second half transimission, code will return right away. When all data is transferred, the callback function is called.

Parameters

base	SPI base pointer
handle	A pointer to the spi_master_dma_handle_t structure which stores the transfer state.
xfer	A pointer to the spi_half_duplex_transfer_t structure.

Returns

status of status_t.

17.4.5.4 static status_t SPI_SlaveTransferCreateHandleDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *txHandle*, dma_handle_t * *rxHandle*) [inline], [static]

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

base	SPI peripheral base address.
handle	SPI handle pointer.
callback	User callback function called at the end of a transfer.
userData	User data for callback.
txHandle	DMA handle pointer for SPI Tx, the handle shall be static allocated by users.
rxHandle	DMA handle pointer for SPI Rx, the handle shall be static allocated by users.

17.4.5.5 static status_t SPI_SlaveTransferDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, spi_transfer_t * *xfer*) [inline], [static]

Note

This interface returned immediately after transfer initiates, users should call SPI_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.
xfer	Pointer to dma transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_SPI_Busy	SPI is not idle, is running another transfer.

17.4.5.6 void SPI_MasterTransferAbortDMA (SPI_Type * *base,* spi_dma_handle_t * *handle*)

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.

17.4.5.7 status_t SPI_MasterTransferGetCountDMA (SPI_Type * *base,* spi_dma_handle_t * *handle,* size_t * *count*)

This function gets the master DMA transfer remaining bytes.

Parameters

base	SPI peripheral base address.
handle	A pointer to the spi_dma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

17.4.5.8 static void SPI_SlaveTransferAbortDMA (SPI_Type * *base,* spi_dma_handle_t * *handle*) [inline], [static]

Parameters

base	SPI peripheral base address.
handle	SPI DMA handle pointer.

17.4.5.9 static status_t SPI_SlaveTransferGetCountDMA (SPI_Type * *base*, spi_dma_handle_t * *handle*, size_t * *count*) [inline], [static]

This function gets the slave DMA transfer remaining bytes.

Parameters

base	SPI peripheral base address.
handle	A pointer to the spi_dma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

17.5 SPI FreeRTOS driver

17.5.1 Overview

This section describes the programming interface of the SPI FreeRTOS driver.

Files

- file fsl_spi_freertos.h

Data Structures

- struct spi_rtos_handle_t
  SPI FreeRTOS handle. More...

Driver version

- #define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  SPI FreeRTOS driver version 2.1.0.

SPI RTOS Operation

- status_t SPI_RTOS_Init (spi_rtos_handle_t *handle, SPI_Type *base, const spi_master_config_t *masterConfig, uint32_t srcClock_Hz)
  Initializes SPI.
- status_t SPI_RTOS_Deinit (spi_rtos_handle_t *handle)
  Deinitializes the SPI.
- status_t SPI_RTOS_Transfer (spi_rtos_handle_t *handle, spi_transfer_t *transfer)
  Performs SPI transfer.

17.5.2 Data Structure Documentation

17.5.2.1 struct spi_rtos_handle_t

Data Fields

- SPI_Type * base
  SPI base address.
- spi_master_handle_t drv_handle
  Handle of the underlying driver, treated as opaque by the RTOS layer.
- SemaphoreHandle_t mutex
  Mutex to lock the handle during a trasfer.
- SemaphoreHandle_t event

Semaphore to notify and unblock task when transfer ends.

17.5.3 Macro Definition Documentation

17.5.3.1 #define FSL_SPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

17.5.4 Function Documentation

17.5.4.1 status_t SPI_RTOS_Init (spi_rtos_handle_t * *handle,* SPI_Type * *base,* const spi_master_config_t * *masterConfig,* uint32_t *srcClock_Hz*)

This function initializes the SPI module and related RTOS context.

Parameters

handle	The RTOS SPI handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the SPI instance to initialize.
masterConfig	Configuration structure to set-up SPI in master mode.
srcClock_Hz	Frequency of input clock of the SPI module.

Returns

status of the operation.

17.5.4.2 status_t SPI_RTOS_Deinit (spi_rtos_handle_t * *handle*)

This function deinitializes the SPI module and related RTOS context.

Parameters

handle	The RTOS SPI handle.

17.5.4.3 status_t SPI_RTOS_Transfer (spi_rtos_handle_t * *handle,* spi_transfer_t * *transfer*)

This function performs an SPI transfer according to data given in the transfer structure.

Parameters

handle	The RTOS SPI handle.
transfer	Structure specifying the transfer parameters.

Returns

status of the operation.

17.6 SPI CMSIS driver

This section describes the programming interface of the SPI Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage methord see http://www.keil.-com/pack/doc/cmsis/Driver/html/index.html.

17.6.1 Function groups

17.6.1.1 SPI CMSIS GetVersion Operation

This function group will return the SPI CMSIS Driver version to user.

17.6.1.2 SPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

17.6.1.3 SPI 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 slected 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.

17.6.1.4 SPI CMSIS Transfer Operation

This function group controls the transfer, master send/receive data, and slave send/receive data.

17.6.1.5 SPI CMSIS Status Operation

This function group gets the SPI transfer status.

17.6.1.6 SPI CMSIS Control Operation

This function can configure instance as master mode or slave mode, set baudrate for master mode transfer, get current baudrate of master mode transfer,set transfer data bits and other control command.

17.6.2 Typical use case

17.6.2.1 Master Operation

```
/* Variables */
uint8_t masterRxData[TRANSFER_SIZE] = {0U};
uint8_t masterTxData[TRANSFER_SIZE] = {0U};

/*SPI master init*/
DRIVER_MASTER_SPI.Initialize(SPI_MasterSignalEvent_t);
DRIVER_MASTER_SPI.PowerControl(ARM_POWER_FULL);
DRIVER_MASTER_SPI.Control(ARM_SPI_MODE_MASTER, TRANSFER_BAUDRATE);

/* Start master transfer */
DRIVER_MASTER_SPI.Transfer(masterTxData, masterRxData, TRANSFER_SIZE);

/* Master power off */
DRIVER_MASTER_SPI.PowerControl(ARM_POWER_OFF);

/* Master uninitialize */
DRIVER_MASTER_SPI.Uninitialize();
```

17.6.2.2 Slave Operation

```
/* Variables */
uint8_t slaveRxData[TRANSFER_SIZE] = {0U};
uint8_t slaveTxData[TRANSFER_SIZE] = {0U};

/*SPI slave init*/
DRIVER_SLAVE_SPI.Initialize(SPI_SlaveSignalEvent_t);
DRIVER_SLAVE_SPI.PowerControl(ARM_POWER_FULL);
DRIVER_SLAVE_SPI.Control(ARM_SPI_MODE_SLAVE, false);

/* Start slave transfer */
DRIVER_SLAVE_SPI.Transfer(slaveTxData, slaveRxData, TRANSFER_SIZE);

/* slave power off */
DRIVER_SLAVE_SPI.PowerControl(ARM_POWER_OFF);

/* slave uninitialize */
DRIVER_SLAVE_SPI.Uninitialize();
```

Chapter 18
USART: Universal Synchronous/Asynchronous Receiver/-Transmitter Driver

18.1 Overview

The MCUXpresso SDK provides a peripheral UART driver for the Universal Synchronous Receiver/-Transmitter (USART) module of MCUXpresso SDK devices. Driver does not support synchronous mode.

The USART driver includes two parts: functional APIs and transactional APIs.

Functional APIs are used for USART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the USART peripheral and know how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. USART functional operation groups provide the functional APIs set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the usart_handle_t as the second parameter. Initialize the handle by calling the USART_TransferCreateHandle() API.

Transactional APIs support asynchronous transfer, which means that the functions USART_TransferSendNonBlocking() and USART_TransferReceiveNonBlocking() set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_USART_TxIdle and kStatus_USART_RxIdle.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the USART_TransferCreateHandle(). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The USART_TransferReceiveNonBlocking() function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the kStatus_USART_RxIdle.

If the receive ring buffer is full, the upper layer is informed through a callback with the kStatus_USART_RxRingBufferOverrun. In the callback function, the upper layer reads data out from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code:

```
USART_TransferCreateHandle(USART0, &handle, USART_UserCallback, NULL);
```

In this example, the buffer size is 32, but only 31 bytes are used for saving data.

18.2 Typical use case

18.2.1 USART Send/receive using a polling method

```
uint8_t ch;
USART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableTx = true;
user_config.enableRx = true;

USART_Init(USART1,&user_config,120000000U);

while(1)
{
    USART_ReadBlocking(USART1, &ch, 1);
    USART_WriteBlocking(USART1, &ch, 1);
}
```

18.2.2 USART Send/receive using an interrupt method

```
usart_handle_t g_usartHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);
    USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);

    // Prepare to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData);
    txFinished = false;

    // Send out.
    USART_TransferSendNonBlocking(USART1, &g_usartHandle, &sendXfer);
```

```
    // Wait send finished.
    while (!txFinished)
    {
    }

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
    rxFinished = false;

    // Receive.
    USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer,
      NULL);

    // Wait receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

18.2.3 USART Receive using the ringbuffer feature

```
#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE     32

usart_handle_t g_usartHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);
    USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);
    USART_TransferStartRingBuffer(USART1, &g_usartHandle, ringBuffer,
      RING_BUFFER_SIZE);
    // Now the RX is working in background, receive in to ring buffer.

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
```

```
    rxFinished = false;

    // Receive.
    USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer);

    if (bytesRead = RX_DATA_SIZE) /* Have read enough data. */
    {
        ;
    }
    else
    {
        if (bytesRead) /* Received some data, process first. */
        {
            ;
        }

        // Wait receive finished.
        while (!rxFinished)
        {
        }
    }

    // ...
}
```

18.2.4 USART Send/Receive using the DMA method

```
usart_handle_t g_usartHandle;
dma_handle_t g_usartTxDmaHandle;
dma_handle_t g_usartRxDmaHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_USART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_USART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    USART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    USART_Init(USART1, &user_config, 120000000U);

    // Set up the DMA
```

```
    DMA_Init(DMA0);
    DMA_EnableChannel(DMA0, USART_TX_DMA_CHANNEL);
    DMA_EnableChannel(DMA0, USART_RX_DMA_CHANNEL);

    DMA_CreateHandle(&g_usartTxDmaHandle, DMA0, USART_TX_DMA_CHANNEL);
    DMA_CreateHandle(&g_usartRxDmaHandle, DMA0, USART_RX_DMA_CHANNEL);

    USART_TransferCreateHandleDMA(USART1, &g_usartHandle, USART_UserCallback,
      NULL, &g_usartTxDmaHandle, &g_usartRxDmaHandle);

    // Prepare to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData);
    txFinished = false;

    // Send out.
    USART_TransferSendDMA(USART1, &g_usartHandle, &sendXfer);

    // Wait send finished.
    while (!txFinished)
    {
    }

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData);
    rxFinished = false;

    // Receive.
    USART_TransferReceiveDMA(USART1, &g_usartHandle, &receiveXfer);

    // Wait receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

Modules

- USART CMSIS Driver
- USART DMA Driver
- USART Driver
- USART FreeRTOS Driver

18.3 USART Driver

18.3.1 Overview

Data Structures

- struct usart_config_t
  USART configuration structure. More...
- struct usart_transfer_t
  USART transfer structure. More...
- struct usart_handle_t
  USART handle structure. More...

Macros

- #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef void(∗ usart_transfer_callback_t)(USART_Type ∗base, usart_handle_t ∗handle, status_t status, void ∗userData)
  USART transfer callback function.
- typedef void(∗ flexcomm_usart_irq_handler_t)(USART_Type ∗base, usart_handle_t ∗handle)
  Typedef for usart interrupt handler.

Enumerations

- enum {
  kStatus_USART_TxBusy = MAKE_STATUS(kStatusGroup_LPC_USART, 0),
  kStatus_USART_RxBusy = MAKE_STATUS(kStatusGroup_LPC_USART, 1),
  kStatus_USART_TxIdle = MAKE_STATUS(kStatusGroup_LPC_USART, 2),
  kStatus_USART_RxIdle = MAKE_STATUS(kStatusGroup_LPC_USART, 3),
  kStatus_USART_TxError = MAKE_STATUS(kStatusGroup_LPC_USART, 7),
  kStatus_USART_RxError = MAKE_STATUS(kStatusGroup_LPC_USART, 9),
  kStatus_USART_RxRingBufferOverrun = MAKE_STATUS(kStatusGroup_LPC_USART, 8),
  kStatus_USART_NoiseError = MAKE_STATUS(kStatusGroup_LPC_USART, 10),
  kStatus_USART_FramingError = MAKE_STATUS(kStatusGroup_LPC_USART, 11),
  kStatus_USART_ParityError = MAKE_STATUS(kStatusGroup_LPC_USART, 12),
  kStatus_USART_BaudrateNotSupport,
  kStatus_USART_Timeout = MAKE_STATUS(kStatusGroup_LPC_USART, 14) }
  Error codes for the USART driver.

- enum usart_sync_mode_t {
  kUSART_SyncModeDisabled = 0x0U,
  kUSART_SyncModeSlave = 0x2U,
  kUSART_SyncModeMaster = 0x3U }
  USART synchronous mode.
- enum usart_parity_mode_t {
  kUSART_ParityDisabled = 0x0U,
  kUSART_ParityEven = 0x2U,
  kUSART_ParityOdd = 0x3U }
  USART parity mode.
- enum usart_stop_bit_count_t {
  kUSART_OneStopBit = 0U,
  kUSART_TwoStopBit = 1U }
  USART stop bit count.
- enum usart_data_len_t {
  kUSART_7BitsPerChar = 0U,
  kUSART_8BitsPerChar = 1U }
  USART data size.
- enum usart_clock_polarity_t {
  kUSART_RxSampleOnFallingEdge = 0x0U,
  kUSART_RxSampleOnRisingEdge = 0x1U }
  USART clock polarity configuration, used in sync mode.
- enum usart_txfifo_watermark_t {
  kUSART_TxFifo0 = 0,
  kUSART_TxFifo1 = 1,
  kUSART_TxFifo2 = 2,
  kUSART_TxFifo3 = 3,
  kUSART_TxFifo4 = 4,
  kUSART_TxFifo5 = 5,
  kUSART_TxFifo6 = 6,
  kUSART_TxFifo7 = 7 }
  txFIFO watermark values
- enum usart_rxfifo_watermark_t {
  kUSART_RxFifo1 = 0,
  kUSART_RxFifo2 = 1,
  kUSART_RxFifo3 = 2,
  kUSART_RxFifo4 = 3,
  kUSART_RxFifo5 = 4,
  kUSART_RxFifo6 = 5,
  kUSART_RxFifo7 = 6,
  kUSART_RxFifo8 = 7 }
  rxFIFO watermark values
- enum _usart_interrupt_enable
  USART interrupt configuration structure, default settings all disabled.
- enum _usart_flags {

kUSART_TxError = (USART_FIFOSTAT_TXERR_MASK),
kUSART_RxError = (USART_FIFOSTAT_RXERR_MASK),
kUSART_TxFifoEmptyFlag = (USART_FIFOSTAT_TXEMPTY_MASK),
kUSART_TxFifoNotFullFlag = (USART_FIFOSTAT_TXNOTFULL_MASK),
kUSART_RxFifoNotEmptyFlag = (USART_FIFOSTAT_RXNOTEMPTY_MASK),
kUSART_RxFifoFullFlag = (USART_FIFOSTAT_RXFULL_MASK) }

USART status flags.

Functions

- uint32_t USART_GetInstance (USART_Type *base)
  Returns instance number for USART peripheral base address.

Driver version

- #define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  USART driver version.

Initialization and deinitialization

- status_t USART_Init (USART_Type *base, const usart_config_t *config, uint32_t srcClock_Hz)
  Initializes a USART instance with user configuration structure and peripheral clock.
- void USART_Deinit (USART_Type *base)
  Deinitializes a USART instance.
- void USART_GetDefaultConfig (usart_config_t *config)
  Gets the default configuration structure.
- status_t USART_SetBaudRate (USART_Type *base, uint32_t baudrate_Bps, uint32_t srcClock_-Hz)
  Sets the USART instance baud rate.
- status_t USART_Enable32kMode (USART_Type *base, uint32_t baudRate_Bps, bool enable-Mode32k, uint32_t srcClock_Hz)
  Enable 32 kHz mode which USART uses clock from the RTC oscillator as the clock source.
- void USART_Enable9bitMode (USART_Type *base, bool enable)
  Enable 9-bit data mode for USART.
- static void USART_SetMatchAddress (USART_Type *base, uint8_t address)
  Set the USART slave address.
- static void USART_EnableMatchAddress (USART_Type *base, bool match)
  Enable the USART match address feature.

Status

- static uint32_t USART_GetStatusFlags (USART_Type *base)
  Get USART status flags.
- static void USART_ClearStatusFlags (USART_Type *base, uint32_t mask)
  Clear USART status flags.

Interrupts

- static void USART_EnableInterrupts (USART_Type *base, uint32_t mask)
  Enables USART interrupts according to the provided mask.
- static void USART_DisableInterrupts (USART_Type *base, uint32_t mask)
  Disables USART interrupts according to a provided mask.
- static uint32_t USART_GetEnabledInterrupts (USART_Type *base)
  Returns enabled USART interrupts.
- static void USART_EnableTxDMA (USART_Type *base, bool enable)
  Enable DMA for Tx.
- static void USART_EnableRxDMA (USART_Type *base, bool enable)
  Enable DMA for Rx.
- static void USART_EnableCTS (USART_Type *base, bool enable)
  Enable CTS.
- static void USART_EnableContinuousSCLK (USART_Type *base, bool enable)
  Continuous Clock generation.
- static void USART_EnableAutoClearSCLK (USART_Type *base, bool enable)
  Enable Continuous Clock generation bit auto clear.
- static void USART_SetRxFifoWatermark (USART_Type *base, uint8_t water)
  Sets the rx FIFO watermark.
- static void USART_SetTxFifoWatermark (USART_Type *base, uint8_t water)
  Sets the tx FIFO watermark.

Bus Operations

- static void USART_WriteByte (USART_Type *base, uint8_t data)
  Writes to the FIFOWR register.
- static uint8_t USART_ReadByte (USART_Type *base)
  Reads the FIFORD register directly.
- static uint8_t USART_GetRxFifoCount (USART_Type *base)
  Gets the rx FIFO data count.
- static uint8_t USART_GetTxFifoCount (USART_Type *base)
  Gets the tx FIFO data count.
- void USART_SendAddress (USART_Type *base, uint8_t address)
  Transmit an address frame in 9-bit data mode.
- status_t USART_WriteBlocking (USART_Type *base, const uint8_t *data, size_t length)
  Writes to the TX register using a blocking method.
- status_t USART_ReadBlocking (USART_Type *base, uint8_t *data, size_t length)
  Read RX data register using a blocking method.

Transactional

- status_t USART_TransferCreateHandle (USART_Type *base, usart_handle_t *handle, usart_transfer_callback_t callback, void *userData)
  Initializes the USART handle.
- status_t USART_TransferSendNonBlocking (USART_Type *base, usart_handle_t *handle, usart_transfer_t *xfer)
  Transmits a buffer of data using the interrupt method.

- void USART_TransferStartRingBuffer (USART_Type *base, usart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
  Sets up the RX ring buffer.
- void USART_TransferStopRingBuffer (USART_Type *base, usart_handle_t *handle)
  Aborts the background transfer and uninstalls the ring buffer.
- size_t USART_TransferGetRxRingBufferLength (usart_handle_t *handle)
  Get the length of received data in RX ring buffer.
- void USART_TransferAbortSend (USART_Type *base, usart_handle_t *handle)
  Aborts the interrupt-driven data transmit.
- status_t USART_TransferGetSendCount (USART_Type *base, usart_handle_t *handle, uint32_t *count)
  Get the number of bytes that have been sent out to bus.
- status_t USART_TransferReceiveNonBlocking (USART_Type *base, usart_handle_t *handle, usart_transfer_t *xfer, size_t *receivedBytes)
  Receives a buffer of data using an interrupt method.
- void USART_TransferAbortReceive (USART_Type *base, usart_handle_t *handle)
  Aborts the interrupt-driven data receiving.
- status_t USART_TransferGetReceiveCount (USART_Type *base, usart_handle_t *handle, uint32_t *count)
  Get the number of bytes that have been received.
- void USART_TransferHandleIRQ (USART_Type *base, usart_handle_t *handle)
  USART IRQ handle function.

18.3.2 Data Structure Documentation

18.3.2.1 struct usart_config_t

Data Fields

- uint32_t baudRate_Bps
  USART baud rate.
- usart_parity_mode_t parityMode
  Parity mode, disabled (default), even, odd.
- usart_stop_bit_count_t stopBitCount
  Number of stop bits, 1 stop bit (default) or 2 stop bits.
- usart_data_len_t bitCountPerChar
  Data length - 7 bit, 8 bit.
- bool loopback
  Enable peripheral loopback.
- bool enableRx
  Enable RX.
- bool enableTx
  Enable TX.
- bool enableContinuousSCLK
  USART continuous Clock generation enable in synchronous master mode.
- bool enableMode32k
  USART uses 32 kHz clock from the RTC oscillator as the clock source.
- bool enableHardwareFlowControl
  Enable hardware control RTS/CTS.

- usart_txfifo_watermark_t txWatermark
  txFIFO watermark
- usart_rxfifo_watermark_t rxWatermark
  rxFIFO watermark
- usart_sync_mode_t syncMode
  Transfer mode select - asynchronous, synchronous master, synchronous slave.
- usart_clock_polarity_t clockPolarity
  Selects the clock polarity and sampling edge in synchronous mode.

Field Documentation

(1) bool usart_config_t::enableContinuousSCLK

(2) bool usart_config_t::enableMode32k

(3) usart_sync_mode_t usart_config_t::syncMode

(4) usart_clock_polarity_t usart_config_t::clockPolarity

18.3.2.2 struct usart_transfer_t

Data Fields

- size_t dataSize
  The byte count to be transfer.
- uint8_t ∗ data
  The buffer of data to be transfer.
- uint8_t ∗ rxData
  The buffer to receive data.
- const uint8_t ∗ txData
  The buffer of data to be sent.

Field Documentation

(1) uint8_t∗ usart_transfer_t::data

(2) uint8_t∗ usart_transfer_t::rxData

(3) const uint8_t∗ usart_transfer_t::txData

(4) size_t usart_transfer_t::dataSize

18.3.2.3 struct _usart_handle

Data Fields

- const uint8_t ∗volatile txData
  Address of remaining data to send.
- volatile size_t txDataSize
  Size of the remaining data to send.
- size_t txDataSizeAll

Size of the data to send out.
- uint8_t *volatile rxData
  Address of remaining data to receive.
- volatile size_t rxDataSize
  Size of the remaining data to receive.
- size_t rxDataSizeAll
  Size of the data to receive.
- uint8_t * rxRingBuffer
  Start address of the receiver ring buffer.
- size_t rxRingBufferSize
  Size of the ring buffer.
- volatile uint16_t rxRingBufferHead
  Index for the driver to store received data into ring buffer.
- volatile uint16_t rxRingBufferTail
  Index for the user to get data from the ring buffer.
- usart_transfer_callback_t callback
  Callback function.
- void * userData
  USART callback function parameter.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.
- uint8_t txWatermark
  txFIFO watermark
- uint8_t rxWatermark
  rxFIFO watermark

Field Documentation

(1) const uint8_t* volatile usart_handle_t::txData

(2) volatile size_t usart_handle_t::txDataSize

(3) size_t usart_handle_t::txDataSizeAll

(4) uint8_t* volatile usart_handle_t::rxData

(5) volatile size_t usart_handle_t::rxDataSize

(6) size_t usart_handle_t::rxDataSizeAll

(7) uint8_t* usart_handle_t::rxRingBuffer

(8) size_t usart_handle_t::rxRingBufferSize

(9) volatile uint16_t usart_handle_t::rxRingBufferHead

(10) volatile uint16_t usart_handle_t::rxRingBufferTail

(11) usart_transfer_callback_t usart_handle_t::callback

(12) void∗ usart_handle_t::userData

(13) volatile uint8_t usart_handle_t::txState

18.3.3 Macro Definition Documentation

18.3.3.1 #define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

18.3.3.2 #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/

18.3.4 Typedef Documentation

18.3.4.1 typedef void(∗ usart_transfer_callback_t)(USART_Type ∗base, usart_handle_t ∗handle, status_t status, void ∗userData)

18.3.4.2 typedef void(∗ flexcomm_usart_irq_handler_t)(USART_Type ∗base, usart_handle_t ∗handle)

18.3.5 Enumeration Type Documentation

18.3.5.1 anonymous enum

Enumerator

kStatus_USART_TxBusy Transmitter is busy.
kStatus_USART_RxBusy Receiver is busy.
kStatus_USART_TxIdle USART transmitter is idle.
kStatus_USART_RxIdle USART receiver is idle.
kStatus_USART_TxError Error happens on txFIFO.
kStatus_USART_RxError Error happens on rxFIFO.
kStatus_USART_RxRingBufferOverrun Error happens on rx ring buffer.
kStatus_USART_NoiseError USART noise error.
kStatus_USART_FramingError USART framing error.
kStatus_USART_ParityError USART parity error.
kStatus_USART_BaudrateNotSupport Baudrate is not support in current clock source.
kStatus_USART_Timeout USART time out.

18.3.5.2 enum usart_sync_mode_t

Enumerator

kUSART_SyncModeDisabled Asynchronous mode.
kUSART_SyncModeSlave Synchronous slave mode.

kUSART_SyncModeMaster Synchronous master mode.

18.3.5.3 enum usart_parity_mode_t

Enumerator

kUSART_ParityDisabled Parity disabled.
kUSART_ParityEven Parity enabled, type even, bit setting: PE|PT = 10.
kUSART_ParityOdd Parity enabled, type odd, bit setting: PE|PT = 11.

18.3.5.4 enum usart_stop_bit_count_t

Enumerator

kUSART_OneStopBit One stop bit.
kUSART_TwoStopBit Two stop bits.

18.3.5.5 enum usart_data_len_t

Enumerator

kUSART_7BitsPerChar Seven bit mode.
kUSART_8BitsPerChar Eight bit mode.

18.3.5.6 enum usart_clock_polarity_t

Enumerator

kUSART_RxSampleOnFallingEdge Un_RXD is sampled on the falling edge of SCLK.
kUSART_RxSampleOnRisingEdge Un_RXD is sampled on the rising edge of SCLK.

18.3.5.7 enum usart_txfifo_watermark_t

Enumerator

kUSART_TxFifo0 USART tx watermark is empty.
kUSART_TxFifo1 USART tx watermark at 1 item.
kUSART_TxFifo2 USART tx watermark at 2 items.
kUSART_TxFifo3 USART tx watermark at 3 items.
kUSART_TxFifo4 USART tx watermark at 4 items.
kUSART_TxFifo5 USART tx watermark at 5 items.
kUSART_TxFifo6 USART tx watermark at 6 items.
kUSART_TxFifo7 USART tx watermark at 7 items.

18.3.5.8 enum usart_rxfifo_watermark_t

Enumerator

kUSART_RxFifo1 USART rx watermark at 1 item.
kUSART_RxFifo2 USART rx watermark at 2 items.
kUSART_RxFifo3 USART rx watermark at 3 items.
kUSART_RxFifo4 USART rx watermark at 4 items.
kUSART_RxFifo5 USART rx watermark at 5 items.
kUSART_RxFifo6 USART rx watermark at 6 items.
kUSART_RxFifo7 USART rx watermark at 7 items.
kUSART_RxFifo8 USART rx watermark at 8 items.

18.3.5.9 enum _usart_flags

This provides constants for the USART status flags for use in the USART functions.

Enumerator

kUSART_TxError TEERR bit, sets if TX buffer is error.
kUSART_RxError RXERR bit, sets if RX buffer is error.
kUSART_TxFifoEmptyFlag TXEMPTY bit, sets if TX buffer is empty.
kUSART_TxFifoNotFullFlag TXNOTFULL bit, sets if TX buffer is not full.
kUSART_RxFifoNotEmptyFlag RXNOEMPTY bit, sets if RX buffer is not empty.
kUSART_RxFifoFullFlag RXFULL bit, sets if RX buffer is full.

18.3.6 Function Documentation

18.3.6.1 uint32_t USART_GetInstance (USART_Type * *base*)

18.3.6.2 status_t USART_Init (USART_Type * *base*, const usart_config_t * *config*, uint32_t *srcClock_Hz*)

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the USART_GetDefaultConfig() function. Example below shows how to use this API to configure USART.

```
*  usart_config_t usartConfig;
*  usartConfig.baudRate_Bps = 115200U;
*  usartConfig.parityMode = kUSART_ParityDisabled;
*  usartConfig.stopBitCount = kUSART_OneStopBit;
*  USART_Init(USART1, &usartConfig, 20000000U);
*
```

Parameters

base	USART peripheral base address.
config	Pointer to user-defined configuration structure.
srcClock_Hz	USART clock source frequency in HZ.

Return values

kStatus_USART_-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_InvalidArgument	USART base address is not valid
kStatus_Success	Status USART initialize succeed

18.3.6.3 void USART_Deinit (USART_Type ∗ *base*)

This function waits for TX complete, disables TX and RX, and disables the USART clock.

Parameters

base	USART peripheral base address.

18.3.6.4 void USART_GetDefaultConfig (usart_config_t ∗ *config*)

This function initializes the USART configuration structure to a default value. The default values are: usartConfig->baudRate_Bps = 115200U; usartConfig->parityMode = kUSART_ParityDisabled; usartConfig->stopBitCount = kUSART_OneStopBit; usartConfig->bitCountPerChar = kUSART_8BitsPerChar; usartConfig->loopback = false; usartConfig->enableTx = false; usartConfig->enableRx = false;

Parameters

config	Pointer to configuration structure.

18.3.6.5 status_t USART_SetBaudRate (USART_Type ∗ *base,* uint32_t *baudrate_Bps,* uint32_t *srcClock_Hz*)

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the USART_Init.

```
* USART_SetBaudRate(USART1, 115200U, 20000000U);
*
```

Parameters

base	USART peripheral base address.
baudrate_Bps	USART baudrate to be set.
srcClock_Hz	USART clock source frequency in HZ.

Return values

kStatus_USART_-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	Set baudrate succeed.
kStatus_InvalidArgument	One or more arguments are invalid.

18.3.6.6 status_t USART_Enable32kMode (USART_Type * *base*, uint32_t *baudRate_Bps*, bool *enableMode32k*, uint32_t *srcClock_Hz*)

Please note that in order to use a 32 kHz clock to operate USART properly, the RTC oscillator and its 32 kHz output must be manully enabled by user, by calling RTC_Init and setting SYSCON_RTCOSCCTRL-_EN bit to 1. And in 32kHz clocking mode the USART can only work at 9600 baudrate or at the baudrate that 9600 can evenly divide, eg: 4800, 3200.

Parameters

base	USART peripheral base address.
baudRate_Bps	USART baudrate to be set..
enable-Mode32k	true is 32k mode, false is normal mode.
srcClock_Hz	USART clock source frequency in HZ.

Return values

kStatus_USART_-BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	Set baudrate succeed.
kStatus_InvalidArgument	One or more arguments are invalid.

18.3.6.7 void USART_Enable9bitMode (USART_Type * *base*, bool *enable*)

This function set the 9-bit mode for USART module. The 9th bit is not used for parity thus can be modified by user.

Parameters

base	USART peripheral base address.
enable	true to enable, false to disable.

18.3.6.8 static void USART_SetMatchAddress (USART_Type * *base,* uint8_t *address*) [inline], [static]

This function configures the address for USART module that works as slave in 9-bit data mode. When the address detection is enabled, the frame it receices with MSB being 1 is considered as an address frame, otherwise it is considered as data frame. Once the address frame matches slave's own addresses, this slave is addressed. This address frame and its following data frames are stored in the receive buffer, otherwise the frames will be discarded. To un-address a slave, just send an address frame with unmatched address.

Note

Any USART instance joined in the multi-slave system can work as slave. The position of the address mark is the same as the parity bit when parity is enabled for 8 bit and 9 bit data formats.

Parameters

base	USART peripheral base address.
address	USART slave address.

18.3.6.9 static void USART_EnableMatchAddress (USART_Type * *base,* bool *match*) [inline], [static]

Parameters

base	USART peripheral base address.
match	true to enable match address, false to disable.

18.3.6.10 static uint32_t USART_GetStatusFlags (USART_Type * *base*) [inline], [static]

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators _usart_flags. To check a specific status, compare the return value with enumerators in _usart_flags. For example, to check whether the TX is empty:

```
*     if (kUSART_TxFifoNotFullFlag &
```

```
*     USART_GetStatusFlags(USART1))
*     {
*         ...
*     }
*
```

Parameters

base	USART peripheral base address.

Returns

USART status flags which are ORed by the enumerators in the _usart_flags.

18.3.6.11 static void USART_ClearStatusFlags (USART_Type * *base*, uint32_t *mask*) [inline], [static]

This function clear supported USART status flags Flags that can be cleared or set are: kUSART_TxError kUSART_RxError For example:

```
*     USART_ClearStatusFlags(USART1, kUSART_TxError |
      kUSART_RxError)
*
```

Parameters

base	USART peripheral base address.
mask	status flags to be cleared.

18.3.6.12 static void USART_EnableInterrupts (USART_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See _usart_interrupt_enable. For example, to enable TX empty interrupt and RX full interrupt:

```
*     USART_EnableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
      kUSART_RxLevelInterruptEnable);
*
```

Parameters

base	USART peripheral base address.
mask	The interrupts to enable. Logical OR of _usart_interrupt_enable.

18.3.6.13 static void USART_DisableInterrupts (USART_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See _usart_interrupt_enable. This example shows how to disable the TX empty interrupt and RX full interrupt:

```
*     USART_DisableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
      kUSART_RxLevelInterruptEnable);
*
```

Parameters

base	USART peripheral base address.
mask	The interrupts to disable. Logical OR of _usart_interrupt_enable.

18.3.6.14 static uint32_t USART_GetEnabledInterrupts (USART_Type * *base*) [inline], [static]

This function returns the enabled USART interrupts.

Parameters

base	USART peripheral base address.

18.3.6.15 static void USART_EnableCTS (USART_Type * *base*, bool *enable*) [inline], [static]

This function will determine whether CTS is used for flow control.

Parameters

base	USART peripheral base address.
enable	Enable CTS or not, true for enable and false for disable.

18.3.6.16 static void USART_EnableContinuousSCLK (USART_Type * *base,* bool *enable*) [inline], [static]

By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this funciton, SCLK will run continuously in synchronous mode, allowing characters to be received on Un_-RxD independently from transmission on Un_TXD).

Parameters

base	USART peripheral base address.
enable	Enable Continuous Clock generation mode or not, true for enable and false for disable.

18.3.6.17 static void USART_EnableAutoClearSCLK (USART_Type * *base,* bool *enable*) [inline], [static]

While enable this cuntion, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

Parameters

base	USART peripheral base address.
enable	Enable auto clear or not, true for enable and false for disable.

18.3.6.18 static void USART_SetRxFifoWatermark (USART_Type * *base,* uint8_t *water*) [inline], [static]

Parameters

base	USART peripheral base address.
water	Rx FIFO watermark.

18.3.6.19 static void USART_SetTxFifoWatermark (USART_Type * *base,* uint8_t *water*) [inline], [static]

Parameters

base	USART peripheral base address.
water	Tx FIFO watermark.

18.3.6.20 static void USART_WriteByte (USART_Type * *base,* uint8_t *data*) [inline], [static]

This function writes data to the txFIFO directly. The upper layer must ensure that txFIFO has space for data to write before calling this function.

Parameters

base	USART peripheral base address.
data	The byte to write.

18.3.6.21 static uint8_t USART_ReadByte (USART_Type * *base*) [inline], [static]

This function reads data from the rxFIFO directly. The upper layer must ensure that the rxFIFO is not empty before calling this function.

Parameters

base	USART peripheral base address.

Returns

The byte read from USART data register.

18.3.6.22 static uint8_t USART_GetRxFifoCount (USART_Type * *base*) [inline], [static]

Parameters

base	USART peripheral base address.

Returns

rx FIFO data count.

18.3.6.23 static uint8_t USART_GetTxFifoCount (USART_Type * *base*) [inline], [static]

Parameters

base	USART peripheral base address.

Returns

tx FIFO data count.

18.3.6.24 void USART_SendAddress (USART_Type * *base*, uint8_t *address*)

Parameters

base	USART peripheral base address.
address	USART slave address.

18.3.6.25 status_t USART_WriteBlocking (USART_Type * *base*, const uint8_t * *data*, size_t *length*)

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

Parameters

base	USART peripheral base address.
data	Start address of the data to write.
length	Size of the data to write.

Return values

kStatus_USART_Timeout	Transmission timed out and was aborted.
kStatus_InvalidArgument	Invalid argument.
kStatus_Success	Successfully wrote all data.

18.3.6.26 status_t USART_ReadBlocking (USART_Type * *base*, uint8_t * *data*, size_t *length*)

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data and read data from the TX register.

Parameters

base	USART peripheral base address.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_USART_-FramingError	Receiver overrun happened while receiving data.
kStatus_USART_Parity-Error	Noise error happened while receiving data.
kStatus_USART_Noise-Error	Framing error happened while receiving data.
kStatus_USART_RxError	Overflow or underflow rxFIFO happened.
kStatus_USART_Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully received all data.

18.3.6.27 status_t USART_TransferCreateHandle (USART_Type * *base*, usart_handle_t * *handle*, usart_transfer_callback_t *callback*, void * *userData*)

This function initializes the USART handle which can be used for other USART transactional APIs. Usually, for a specified USART instance, call this API once to get the initialized handle.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
callback	The callback function.
userData	The parameter of the callback function.

18.3.6.28 status_t USART_TransferSendNonBlocking (USART_Type * *base*, usart_handle_t * *handle*, usart_transfer_t * *xfer*)

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the IRQ handler, the USART driver calls the callback function and passes the kStatus_USART_TxIdle as status parameter.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
xfer	USART transfer structure. See usart_transfer_t.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_USART_TxBusy	Previous transmission still not finished, data not all written to TX register yet.
kStatus_InvalidArgument	Invalid argument.

18.3.6.29 void USART_TransferStartRingBuffer (USART_Type * *base*, usart_handle_t * *handle*, uint8_t * *ringBuffer*, size_t *ringBufferSize*)

This function sets up the RX ring buffer to a specific USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the USART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
ringBuffer	Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer.
ringBufferSize	size of the ring buffer.

18.3.6.30 void USART_TransferStopRingBuffer (USART_Type * *base*, usart_handle_t * *handle*)

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

18.3.6.31 size_t USART_TransferGetRxRingBufferLength (usart_handle_t * *handle*)

Parameters

handle	USART handle pointer.

Returns

Length of received data in RX ring buffer.

18.3.6.32 void USART_TransferAbortSend (USART_Type * *base,* usart_handle_t * *handle*)

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are still not sent out.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

18.3.6.33 status_t USART_TransferGetSendCount (USART_Type * *base,* usart_handle_t * *handle,* uint32_t * *count*)

This function gets the number of bytes that have been sent out to bus by interrupt method.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferInProgress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

18.3.6.34 status_t USART_TransferReceiveNonBlocking (USART_Type * *base*, usart_handle_t * *handle*, usart_transfer_t * *xfer*, size_t * *receivedBytes*)

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter kStatus_USART_RxIdle. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the xfer->data and this function returns with the parameter `receivedBytes` set to 5. For the left 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the USART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the xfer->data. When all data is received, the upper layer is notified.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
xfer	USART transfer structure, see usart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into transmit queue.
kStatus_USART_RxBusy	Previous receive request is not finished.
kStatus_InvalidArgument	Invalid argument.

18.3.6.35 void USART_TransferAbortReceive (USART_Type * *base,* usart_handle_t * *handle*)

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

18.3.6.36 status_t USART_TransferGetReceiveCount (USART_Type * *base*, usart_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been received.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
count	Receive bytes count.

Return values

kStatus_NoTransferInProgress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

18.3.6.37 void USART_TransferHandleIRQ (USART_Type * *base*, usart_handle_t * *handle*)

This function handles the USART transmit and receive IRQ request.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.

18.4 USART DMA Driver

18.4.1 Overview

Files

- file fsl_usart_dma.h

Data Structures

- struct usart_dma_handle_t
  UART DMA handle. More...

Typedefs

- typedef void(∗ usart_dma_transfer_callback_t)(USART_Type ∗base, usart_dma_handle_t ∗handle, status_t status, void ∗userData)
  UART transfer callback function.

Driver version

- #define FSL_USART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  USART dma driver version.

DMA transactional

- status_t USART_TransferCreateHandleDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  Initializes the USART handle which is used in transactional functions.
- status_t USART_TransferSendDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_transfer_t ∗xfer)
  Sends data using DMA.
- status_t USART_TransferReceiveDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, usart_transfer_t ∗xfer)
  Receives data using DMA.
- void USART_TransferAbortSendDMA (USART_Type ∗base, usart_dma_handle_t ∗handle)
  Aborts the sent data using DMA.
- void USART_TransferAbortReceiveDMA (USART_Type ∗base, usart_dma_handle_t ∗handle)
  Aborts the received data using DMA.
- status_t USART_TransferGetReceiveCountDMA (USART_Type ∗base, usart_dma_handle_t ∗handle, uint32_t ∗count)
  Get the number of bytes that have been received.

- status_t USART_TransferGetSendCountDMA (USART_Type *base, usart_dma_handle_t *handle, uint32_t *count)
  Get the number of bytes that have been sent.

18.4.2 Data Structure Documentation

18.4.2.1 struct _usart_dma_handle

Data Fields

- USART_Type * base
  UART peripheral base address.
- usart_dma_transfer_callback_t callback
  Callback function.
- void * userData
  UART callback function parameter.
- size_t rxDataSizeAll
  Size of the data to receive.
- size_t txDataSizeAll
  Size of the data to send out.
- dma_handle_t * txDmaHandle
  The DMA TX channel used.
- dma_handle_t * rxDmaHandle
  The DMA RX channel used.
- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

Field Documentation

(1) USART_Type* usart_dma_handle_t::base

(2) usart_dma_transfer_callback_t usart_dma_handle_t::callback

(3) void* usart_dma_handle_t::userData

(4) size_t usart_dma_handle_t::rxDataSizeAll

(5) size_t usart_dma_handle_t::txDataSizeAll

(6) dma_handle_t* usart_dma_handle_t::txDmaHandle

(7) dma_handle_t* usart_dma_handle_t::rxDmaHandle

(8) volatile uint8_t usart_dma_handle_t::txState

18.4.3 Macro Definition Documentation

18.4.3.1 #define FSL_USART_DMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

18.4.4 Typedef Documentation

18.4.4.1 typedef void(∗ usart_dma_transfer_callback_t)(USART_Type ∗base, usart_dma_handle_t ∗handle, status_t status, void ∗userData)

18.4.5 Function Documentation

18.4.5.1 status_t USART_TransferCreateHandleDMA (USART_Type ∗ *base,* usart_dma_handle_t ∗ *handle,* usart_dma_transfer_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *txDmaHandle,* dma_handle_t ∗ *rxDmaHandle*)

Parameters

base	USART peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
callback	Callback function.
userData	User data.
txDmaHandle	User-requested DMA handle for TX DMA transfer.
rxDmaHandle	User-requested DMA handle for RX DMA transfer.

18.4.5.2 status_t USART_TransferSendDMA (USART_Type ∗ *base,* usart_dma_handle_t ∗ *handle,* usart_transfer_t ∗ *xfer*)

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
xfer	USART DMA transfer structure. See usart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_USART_TxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

18.4.5.3 status_t USART_TransferReceiveDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, usart_transfer_t * *xfer*)

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

base	USART peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	USART DMA transfer structure. See usart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_USART_RxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

18.4.5.4 void USART_TransferAbortSendDMA (USART_Type * *base*, usart_dma_handle_t * *handle*)

This function aborts send data using DMA.

Parameters

base	USART peripheral base address
handle	Pointer to usart_dma_handle_t structure

18.4.5.5 void USART_TransferAbortReceiveDMA (USART_Type * *base*, usart_dma_handle_t * *handle*)

This function aborts the received data using DMA.

Parameters

base	USART peripheral base address
handle	Pointer to usart_dma_handle_t structure

18.4.5.6 status_t USART_TransferGetReceiveCountDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been received.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
count	Receive bytes count.

Return values

kStatus_NoTransferIn-Progress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

18.4.5.7 status_t USART_TransferGetSendCountDMA (USART_Type * *base*, usart_dma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been sent.

Parameters

base	USART peripheral base address.
handle	USART handle pointer.
count	Sent bytes count.

Return values

kStatus_NoTransferIn-Progress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

18.5 USART FreeRTOS Driver

18.5.1 Overview

Files

- file fsl_usart_freertos.h

Data Structures

- struct rtos_usart_config
  FLEX USART configuration structure. More...
- struct usart_rtos_handle_t
  FLEX USART FreeRTOS handle. More...

Driver version

- #define FSL_USART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))
  USART FreeRTOS driver version.

USART RTOS Operation

- int USART_RTOS_Init (usart_rtos_handle_t *handle, usart_handle_t *t_handle, const struct rtos_usart_config *cfg)
  Initializes a USART instance for operation in RTOS.
- int USART_RTOS_Deinit (usart_rtos_handle_t *handle)
  Deinitializes a USART instance for operation.

USART transactional Operation

- int USART_RTOS_Send (usart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length)
  Sends data in the background.
- int USART_RTOS_Receive (usart_rtos_handle_t *handle, uint8_t *buffer, uint32_t length, size_t *received)
  Receives data.

18.5.2 Data Structure Documentation

18.5.2.1 struct rtos_usart_config

Data Fields

- USART_Type * base

USART base address.
- uint32_t srcclk
  USART source clock in Hz.
- uint32_t baudrate
  Desired communication speed.
- usart_parity_mode_t parity
  Parity setting.
- usart_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.

18.5.2.2 struct usart_rtos_handle_t

Data Fields

- USART_Type ∗ base
  USART base address.
- usart_transfer_t txTransfer
  TX transfer structure.
- usart_transfer_t rxTransfer
  RX transfer structure.
- SemaphoreHandle_t rxSemaphore
  RX semaphore for resource sharing.
- SemaphoreHandle_t txSemaphore
  TX semaphore for resource sharing.
- EventGroupHandle_t rxEvent
  RX completion event.
- EventGroupHandle_t txEvent
  TX completion event.
- void ∗ t_state
  Transactional state of the underlying driver.

18.5.3 Macro Definition Documentation

18.5.3.1 #define FSL_USART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 6, 0))

18.5.4 Function Documentation

18.5.4.1 int USART_RTOS_Init (usart_rtos_handle_t ∗ *handle,* usart_handle_t ∗ *t_handle,* const struct rtos_usart_config ∗ *cfg*)

Parameters

handle	The RTOS USART handle, the pointer to allocated space for RTOS context.
t_handle	The pointer to allocated space where to store transactional layer internal state.
cfg	The pointer to the parameters required to configure the USART after initialization.

Returns

0 succeed, others fail.

18.5.4.2 int USART_RTOS_Deinit (usart_rtos_handle_t ∗ *handle*)

This function deinitializes the USART module, sets all register values to reset value, and releases the resources.

Parameters

handle	The RTOS USART handle.

18.5.4.3 int USART_RTOS_Send (usart_rtos_handle_t ∗ *handle,* uint8_t ∗ *buffer,* uint32_t *length*)

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

Parameters

handle	The RTOS USART handle.
buffer	The pointer to buffer to send.
length	The number of bytes to send.

18.5.4.4 int USART_RTOS_Receive (usart_rtos_handle_t ∗ *handle,* uint8_t ∗ *buffer,* uint32_t *length,* size_t ∗ *received*)

This function receives data from USART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

Parameters

handle	The RTOS USART handle.
buffer	The pointer to buffer where to write received data.
length	The number of bytes to receive.
received	The pointer to a variable of size_t where the number of received data is filled.

18.6 USART CMSIS Driver

This section describes the programming interface of the USART Cortex Microcontroller Software Interface Standard (CMSIS) driver. And this driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage methord see http://www.keil.-com/pack/doc/cmsis/Driver/html/index.html.

The USART driver includes transactional APIs.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements please write custom code.

18.6.1 USART Send Methods

18.6.1.1 USART Send/receive using an interrupt method

```
/* USART  callback */
void USART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txOnGoing = false;
    }
}
Driver_USART0.Initialize(USART_Callback);
Driver_USART0.PowerControl(ARM_POWER_FULL);
/* Send g_tipString out. */
txOnGoing = true;
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

18.6.1.2 USART Send/Receive using the DMA method

```
/* USART  callback */
void USART_Callback(uint32_t event)
{
    if (event == ARM_USART_EVENT_SEND_COMPLETE)
    {
        txOnGoing = false;
    }

}

Driver_USART0.Initialize(USART_Callback);
DMA_Init(DMA0);
Driver_USART0.PowerControl(ARM_POWER_FULL);

/* Send g_tipString out. */
txOnGoing = true;
```

```
Driver_USART0.Send(g_tipString, sizeof(g_tipString) - 1);

/* Wait send finished */
while (txOnGoing)
{
}
```

Chapter 19
FlexIO: FlexIO Driver

19.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FlexIO drivers for the FlexIO module of MCUXpresso SDK devices.

Modules

- FlexIO Driver
- FlexIO I2C Master Driver
- FlexIO MCU Interface LCD Driver
- FlexIO SPI Driver
- FlexIO UART Driver

19.2 FlexIO Driver

19.2.1 Overview

Data Structures

- struct flexio_config_t
  Define FlexIO user configuration structure. More...
- struct flexio_timer_config_t
  Define FlexIO timer configuration structure. More...
- struct flexio_shifter_config_t
  Define FlexIO shifter configuration structure. More...

Macros

- #define FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x) ((uint32_t)(x) << 1U)
  Calculate FlexIO timer trigger.

Typedefs

- typedef void(∗ flexio_isr_t)(void ∗base, void ∗handle)
  typedef for FlexIO simulated driver interrupt handler.

Enumerations

- enum flexio_timer_trigger_polarity_t {
  kFLEXIO_TimerTriggerPolarityActiveHigh = 0x0U,
  kFLEXIO_TimerTriggerPolarityActiveLow = 0x1U }
  Define time of timer trigger polarity.
- enum flexio_timer_trigger_source_t {
  kFLEXIO_TimerTriggerSourceExternal = 0x0U,
  kFLEXIO_TimerTriggerSourceInternal = 0x1U }
  Define type of timer trigger source.
- enum flexio_pin_config_t {
  kFLEXIO_PinConfigOutputDisabled = 0x0U,
  kFLEXIO_PinConfigOpenDrainOrBidirection = 0x1U,
  kFLEXIO_PinConfigBidirectionOutputData = 0x2U,
  kFLEXIO_PinConfigOutput = 0x3U }
  Define type of timer/shifter pin configuration.
- enum flexio_pin_polarity_t {
  kFLEXIO_PinActiveHigh = 0x0U,
  kFLEXIO_PinActiveLow = 0x1U }
  Definition of pin polarity.

- enum flexio_timer_mode_t {
  kFLEXIO_TimerModeDisabled = 0x0U,
  kFLEXIO_TimerModeDual8BitBaudBit = 0x1U,
  kFLEXIO_TimerModeDual8BitPWM = 0x2U,
  kFLEXIO_TimerModeSingle16Bit = 0x3U }
  Define type of timer work mode.
- enum flexio_timer_output_t {
  kFLEXIO_TimerOutputOneNotAffectedByReset = 0x0U,
  kFLEXIO_TimerOutputZeroNotAffectedByReset = 0x1U,
  kFLEXIO_TimerOutputOneAffectedByReset = 0x2U,
  kFLEXIO_TimerOutputZeroAffectedByReset = 0x3U }
  Define type of timer initial output or timer reset condition.
- enum flexio_timer_decrement_source_t {
  kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput = 0x0U,
  kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput = 0x1U,
  kFLEXIO_TimerDecSrcOnPinInputShiftPinInput = 0x2U,
  kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput = 0x3U }
  Define type of timer decrement.
- enum flexio_timer_reset_condition_t {
  kFLEXIO_TimerResetNever = 0x0U,
  kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput = 0x2U,
  kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput = 0x3U,
  kFLEXIO_TimerResetOnTimerPinRisingEdge = 0x4U,
  kFLEXIO_TimerResetOnTimerTriggerRisingEdge = 0x6U,
  kFLEXIO_TimerResetOnTimerTriggerBothEdge = 0x7U }
  Define type of timer reset condition.
- enum flexio_timer_disable_condition_t {
  kFLEXIO_TimerDisableNever = 0x0U,
  kFLEXIO_TimerDisableOnPreTimerDisable = 0x1U,
  kFLEXIO_TimerDisableOnTimerCompare = 0x2U,
  kFLEXIO_TimerDisableOnTimerCompareTriggerLow = 0x3U,
  kFLEXIO_TimerDisableOnPinBothEdge = 0x4U,
  kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh = 0x5U,
  kFLEXIO_TimerDisableOnTriggerFallingEdge = 0x6U }
  Define type of timer disable condition.
- enum flexio_timer_enable_condition_t {
  kFLEXIO_TimerEnabledAlways = 0x0U,
  kFLEXIO_TimerEnableOnPrevTimerEnable = 0x1U,
  kFLEXIO_TimerEnableOnTriggerHigh = 0x2U,
  kFLEXIO_TimerEnableOnTriggerHighPinHigh = 0x3U,
  kFLEXIO_TimerEnableOnPinRisingEdge = 0x4U,
  kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh = 0x5U,
  kFLEXIO_TimerEnableOnTriggerRisingEdge = 0x6U,
  kFLEXIO_TimerEnableOnTriggerBothEdge = 0x7U }
  Define type of timer enable condition.
- enum flexio_timer_stop_bit_condition_t {

kFLEXIO_TimerStopBitDisabled = 0x0U,
kFLEXIO_TimerStopBitEnableOnTimerCompare = 0x1U,
kFLEXIO_TimerStopBitEnableOnTimerDisable = 0x2U,
kFLEXIO_TimerStopBitEnableOnTimerCompareDisable = 0x3U }

Define type of timer stop bit generate condition.

- enum flexio_timer_start_bit_condition_t {
  kFLEXIO_TimerStartBitDisabled = 0x0U,
  kFLEXIO_TimerStartBitEnabled = 0x1U }

  Define type of timer start bit generate condition.
- enum flexio_shifter_timer_polarity_t {
  kFLEXIO_ShifterTimerPolarityOnPositive = 0x0U,
  kFLEXIO_ShifterTimerPolarityOnNegitive = 0x1U }

  Define type of timer polarity for shifter control.
- enum flexio_shifter_mode_t {
  kFLEXIO_ShifterDisabled = 0x0U,
  kFLEXIO_ShifterModeReceive = 0x1U,
  kFLEXIO_ShifterModeTransmit = 0x2U,
  kFLEXIO_ShifterModeMatchStore = 0x4U,
  kFLEXIO_ShifterModeMatchContinuous = 0x5U,
  kFLEXIO_ShifterModeState = 0x6U,
  kFLEXIO_ShifterModeLogic = 0x7U }

  Define type of shifter working mode.
- enum flexio_shifter_input_source_t {
  kFLEXIO_ShifterInputFromPin = 0x0U,
  kFLEXIO_ShifterInputFromNextShifterOutput = 0x1U }

  Define type of shifter input source.
- enum flexio_shifter_stop_bit_t {
  kFLEXIO_ShifterStopBitDisable = 0x0U,
  kFLEXIO_ShifterStopBitLow = 0x2U,
  kFLEXIO_ShifterStopBitHigh = 0x3U }

  Define of STOP bit configuration.
- enum flexio_shifter_start_bit_t {
  kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable = 0x0U,
  kFLEXIO_ShifterStartBitDisabledLoadDataOnShift = 0x1U,
  kFLEXIO_ShifterStartBitLow = 0x2U,
  kFLEXIO_ShifterStartBitHigh = 0x3U }

  Define type of START bit configuration.
- enum flexio_shifter_buffer_type_t {
  kFLEXIO_ShifterBuffer = 0x0U,
  kFLEXIO_ShifterBufferBitSwapped = 0x1U,
  kFLEXIO_ShifterBufferByteSwapped = 0x2U,
  kFLEXIO_ShifterBufferBitByteSwapped = 0x3U,
  kFLEXIO_ShifterBufferNibbleByteSwapped = 0x4U,
  kFLEXIO_ShifterBufferHalfWordSwapped = 0x5U,
  kFLEXIO_ShifterBufferNibbleSwapped = 0x6U }

  Define FlexIO shifter buffer type.

Variables

- FLEXIO_Type *const s_flexioBases []
  Pointers to flexio bases for each instance.
- const clock_ip_name_t s_flexioClocks []
  Pointers to flexio clocks for each instance.

Driver version

- #define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  FlexIO driver version.

FlexIO Initialization and De-initialization

- void FLEXIO_GetDefaultConfig (flexio_config_t *userConfig)
  Gets the default configuration to configure the FlexIO module.
- void FLEXIO_Init (FLEXIO_Type *base, const flexio_config_t *userConfig)
  Configures the FlexIO with a FlexIO configuration.
- void FLEXIO_Deinit (FLEXIO_Type *base)
  Gates the FlexIO clock.
- uint32_t FLEXIO_GetInstance (FLEXIO_Type *base)
  Get instance number for FLEXIO module.

FlexIO Basic Operation

- void FLEXIO_Reset (FLEXIO_Type *base)
  Resets the FlexIO module.
- static void FLEXIO_Enable (FLEXIO_Type *base, bool enable)
  Enables the FlexIO module operation.
- static uint32_t FLEXIO_ReadPinInput (FLEXIO_Type *base)
  Reads the input data on each of the FlexIO pins.
- static uint8_t FLEXIO_GetShifterState (FLEXIO_Type *base)
  Gets the current state pointer for state mode use.
- void FLEXIO_SetShifterConfig (FLEXIO_Type *base, uint8_t index, const flexio_shifter_config_t *shifterConfig)
  Configures the shifter with the shifter configuration.
- void FLEXIO_SetTimerConfig (FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t *timerConfig)
  Configures the timer with the timer configuration.

FlexIO Interrupt Operation

- static void FLEXIO_EnableShifterStatusInterrupts (FLEXIO_Type *base, uint32_t mask)
  Enables the shifter status interrupt.
- static void FLEXIO_DisableShifterStatusInterrupts (FLEXIO_Type *base, uint32_t mask)

Disables the shifter status interrupt.
- static void FLEXIO_EnableShifterErrorInterrupts (FLEXIO_Type ∗base, uint32_t mask)
  Enables the shifter error interrupt.
- static void FLEXIO_DisableShifterErrorInterrupts (FLEXIO_Type ∗base, uint32_t mask)
  Disables the shifter error interrupt.
- static void FLEXIO_EnableTimerStatusInterrupts (FLEXIO_Type ∗base, uint32_t mask)
  Enables the timer status interrupt.
- static void FLEXIO_DisableTimerStatusInterrupts (FLEXIO_Type ∗base, uint32_t mask)
  Disables the timer status interrupt.

FlexIO Status Operation

- static uint32_t FLEXIO_GetShifterStatusFlags (FLEXIO_Type ∗base)
  Gets the shifter status flags.
- static void FLEXIO_ClearShifterStatusFlags (FLEXIO_Type ∗base, uint32_t mask)
  Clears the shifter status flags.
- static uint32_t FLEXIO_GetShifterErrorFlags (FLEXIO_Type ∗base)
  Gets the shifter error flags.
- static void FLEXIO_ClearShifterErrorFlags (FLEXIO_Type ∗base, uint32_t mask)
  Clears the shifter error flags.
- static uint32_t FLEXIO_GetTimerStatusFlags (FLEXIO_Type ∗base)
  Gets the timer status flags.
- static void FLEXIO_ClearTimerStatusFlags (FLEXIO_Type ∗base, uint32_t mask)
  Clears the timer status flags.

FlexIO DMA Operation

- static void FLEXIO_EnableShifterStatusDMA (FLEXIO_Type ∗base, uint32_t mask, bool enable)
  Enables/disables the shifter status DMA.
- uint32_t FLEXIO_GetShifterBufferAddress (FLEXIO_Type ∗base, flexio_shifter_buffer_type_-t type, uint8_t index)
  Gets the shifter buffer address for the DMA transfer usage.
- status_t FLEXIO_RegisterHandleIRQ (void ∗base, void ∗handle, flexio_isr_t isr)
  Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.
- status_t FLEXIO_UnregisterHandleIRQ (void ∗base)
  Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

19.2.2 Data Structure Documentation

19.2.2.1 struct flexio_config_t

Data Fields

- bool enableFlexio
  Enable/disable FlexIO module.
- bool enableInDoze

Enable/disable FlexIO operation in doze mode.
- bool enableInDebug
  Enable/disable FlexIO operation in debug mode.
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers, fast access requires` *the FlexIO clock to be at least twice the frequency of the bus clock.*

Field Documentation

(1) bool flexio_config_t::enableFastAccess

19.2.2.2 struct flexio_timer_config_t

Data Fields

- uint32_t triggerSelect
  The internal trigger selection number using MACROs.
- flexio_timer_trigger_polarity_t triggerPolarity
  Trigger Polarity.
- flexio_timer_trigger_source_t triggerSource
  Trigger Source, internal (see 'trgsel') or external.
- flexio_pin_config_t pinConfig
  Timer Pin Configuration.
- uint32_t pinSelect
  Timer Pin number Select.
- flexio_pin_polarity_t pinPolarity
  Timer Pin Polarity.
- flexio_timer_mode_t timerMode
  Timer work Mode.
- flexio_timer_output_t timerOutput
  `Configures the initial state of the Timer Output and` *whether it is affected by the Timer reset.*
- flexio_timer_decrement_source_t timerDecrement
  `Configures the source of the Timer decrement and the` *source of the Shift clock.*
- flexio_timer_reset_condition_t timerReset
  `Configures the condition that causes the timer counter` *(and optionally the timer output) to be reset.*
- flexio_timer_disable_condition_t timerDisable
  `Configures the condition that causes the Timer to be` *disabled and stop decrementing.*
- flexio_timer_enable_condition_t timerEnable
  `Configures the condition that causes the Timer to be` *enabled and start decrementing.*
- flexio_timer_stop_bit_condition_t timerStop
  Timer STOP Bit generation.
- flexio_timer_start_bit_condition_t timerStart
  Timer STRAT Bit generation.
- uint32_t timerCompare
  Value for Timer Compare N Register.

Field Documentation

(1) uint32_t flexio_timer_config_t::triggerSelect

(2) flexio_timer_trigger_polarity_t flexio_timer_config_t::triggerPolarity

(3) flexio_timer_trigger_source_t flexio_timer_config_t::triggerSource

(4) flexio_pin_config_t flexio_timer_config_t::pinConfig

(5) uint32_t flexio_timer_config_t::pinSelect

(6) flexio_pin_polarity_t flexio_timer_config_t::pinPolarity

(7) flexio_timer_mode_t flexio_timer_config_t::timerMode

(8) flexio_timer_output_t flexio_timer_config_t::timerOutput

(9) flexio_timer_decrement_source_t flexio_timer_config_t::timerDecrement

(10) flexio_timer_reset_condition_t flexio_timer_config_t::timerReset

(11) flexio_timer_disable_condition_t flexio_timer_config_t::timerDisable

(12) flexio_timer_enable_condition_t flexio_timer_config_t::timerEnable

(13) flexio_timer_stop_bit_condition_t flexio_timer_config_t::timerStop

(14) flexio_timer_start_bit_condition_t flexio_timer_config_t::timerStart

(15) uint32_t flexio_timer_config_t::timerCompare

19.2.2.3 struct flexio_shifter_config_t

Data Fields

- uint32_t timerSelect
  `Selects which Timer is used for controlling the`
  logic/shift register and generating the Shift clock.
- flexio_shifter_timer_polarity_t timerPolarity
  Timer Polarity.
- flexio_pin_config_t pinConfig
  Shifter Pin Configuration.
- uint32_t pinSelect
  Shifter Pin number Select.
- flexio_pin_polarity_t pinPolarity
  Shifter Pin Polarity.
- flexio_shifter_mode_t shifterMode
  Configures the mode of the Shifter.
- uint32_t parallelWidth
  Configures the parallel width when using parallel mode.

- flexio_shifter_input_source_t inputSource
  Selects the input source for the shifter.
- flexio_shifter_stop_bit_t shifterStop
  Shifter STOP bit.
- flexio_shifter_start_bit_t shifterStart
  Shifter START bit.

Field Documentation

(1) uint32_t flexio_shifter_config_t::timerSelect

(2) flexio_shifter_timer_polarity_t flexio_shifter_config_t::timerPolarity

(3) flexio_pin_config_t flexio_shifter_config_t::pinConfig

(4) uint32_t flexio_shifter_config_t::pinSelect

(5) flexio_pin_polarity_t flexio_shifter_config_t::pinPolarity

(6) flexio_shifter_mode_t flexio_shifter_config_t::shifterMode

(7) uint32_t flexio_shifter_config_t::parallelWidth

(8) flexio_shifter_input_source_t flexio_shifter_config_t::inputSource

(9) flexio_shifter_stop_bit_t flexio_shifter_config_t::shifterStop

(10) flexio_shifter_start_bit_t flexio_shifter_config_t::shifterStart

19.2.3 Macro Definition Documentation

19.2.3.1 #define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

19.2.3.2 #define FLEXIO_TIMER_TRIGGER_SEL_PININPUT(*x*) ((uint32_t)(x) << 1U)

19.2.4 Typedef Documentation

19.2.4.1 typedef void(* flexio_isr_t)(void *base, void *handle)

19.2.5 Enumeration Type Documentation

19.2.5.1 enum flexio_timer_trigger_polarity_t

Enumerator

kFLEXIO_TimerTriggerPolarityActiveHigh Active high.
kFLEXIO_TimerTriggerPolarityActiveLow Active low.

19.2.5.2 enum flexio_timer_trigger_source_t

Enumerator

kFLEXIO_TimerTriggerSourceExternal External trigger selected.
kFLEXIO_TimerTriggerSourceInternal Internal trigger selected.

19.2.5.3 enum flexio_pin_config_t

Enumerator

kFLEXIO_PinConfigOutputDisabled Pin output disabled.
kFLEXIO_PinConfigOpenDrainOrBidirection Pin open drain or bidirectional output enable.
kFLEXIO_PinConfigBidirectionOutputData Pin bidirectional output data.
kFLEXIO_PinConfigOutput Pin output.

19.2.5.4 enum flexio_pin_polarity_t

Enumerator

kFLEXIO_PinActiveHigh Active high.
kFLEXIO_PinActiveLow Active low.

19.2.5.5 enum flexio_timer_mode_t

Enumerator

kFLEXIO_TimerModeDisabled Timer Disabled.
kFLEXIO_TimerModeDual8BitBaudBit Dual 8-bit counters baud/bit mode.
kFLEXIO_TimerModeDual8BitPWM Dual 8-bit counters PWM mode.
kFLEXIO_TimerModeSingle16Bit Single 16-bit counter mode.

19.2.5.6 enum flexio_timer_output_t

Enumerator

kFLEXIO_TimerOutputOneNotAffectedByReset Logic one when enabled and is not affected by timer reset.
kFLEXIO_TimerOutputZeroNotAffectedByReset Logic zero when enabled and is not affected by timer reset.
kFLEXIO_TimerOutputOneAffectedByReset Logic one when enabled and on timer reset.
kFLEXIO_TimerOutputZeroAffectedByReset Logic zero when enabled and on timer reset.

19.2.5.7 enum flexio_timer_decrement_source_t

Enumerator

kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput Decrement counter on FlexIO clock, Shift clock equals Timer output.

kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput Decrement counter on Trigger input (both edges), Shift clock equals Timer output.

kFLEXIO_TimerDecSrcOnPinInputShiftPinInput Decrement counter on Pin input (both edges), Shift clock equals Pin input.

kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

19.2.5.8 enum flexio_timer_reset_condition_t

Enumerator

kFLEXIO_TimerResetNever Timer never reset.

kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput Timer reset on Timer Pin equal to Timer Output.

kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput Timer reset on Timer Trigger equal to Timer Output.

kFLEXIO_TimerResetOnTimerPinRisingEdge Timer reset on Timer Pin rising edge.

kFLEXIO_TimerResetOnTimerTriggerRisingEdge Timer reset on Trigger rising edge.

kFLEXIO_TimerResetOnTimerTriggerBothEdge Timer reset on Trigger rising or falling edge.

19.2.5.9 enum flexio_timer_disable_condition_t

Enumerator

kFLEXIO_TimerDisableNever Timer never disabled.

kFLEXIO_TimerDisableOnPreTimerDisable Timer disabled on Timer N-1 disable.

kFLEXIO_TimerDisableOnTimerCompare Timer disabled on Timer compare.

kFLEXIO_TimerDisableOnTimerCompareTriggerLow Timer disabled on Timer compare and Trigger Low.

kFLEXIO_TimerDisableOnPinBothEdge Timer disabled on Pin rising or falling edge.

kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh Timer disabled on Pin rising or falling edge provided Trigger is high.

kFLEXIO_TimerDisableOnTriggerFallingEdge Timer disabled on Trigger falling edge.

19.2.5.10 enum flexio_timer_enable_condition_t

Enumerator

kFLEXIO_TimerEnabledAlways Timer always enabled.

kFLEXIO_TimerEnableOnPrevTimerEnable Timer enabled on Timer N-1 enable.
kFLEXIO_TimerEnableOnTriggerHigh Timer enabled on Trigger high.
kFLEXIO_TimerEnableOnTriggerHighPinHigh Timer enabled on Trigger high and Pin high.
kFLEXIO_TimerEnableOnPinRisingEdge Timer enabled on Pin rising edge.
kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh Timer enabled on Pin rising edge and Trigger high.
kFLEXIO_TimerEnableOnTriggerRisingEdge Timer enabled on Trigger rising edge.
kFLEXIO_TimerEnableOnTriggerBothEdge Timer enabled on Trigger rising or falling edge.

19.2.5.11 enum flexio_timer_stop_bit_condition_t

Enumerator

kFLEXIO_TimerStopBitDisabled Stop bit disabled.
kFLEXIO_TimerStopBitEnableOnTimerCompare Stop bit is enabled on timer compare.
kFLEXIO_TimerStopBitEnableOnTimerDisable Stop bit is enabled on timer disable.
kFLEXIO_TimerStopBitEnableOnTimerCompareDisable Stop bit is enabled on timer compare and timer disable.

19.2.5.12 enum flexio_timer_start_bit_condition_t

Enumerator

kFLEXIO_TimerStartBitDisabled Start bit disabled.
kFLEXIO_TimerStartBitEnabled Start bit enabled.

19.2.5.13 enum flexio_shifter_timer_polarity_t

Enumerator

kFLEXIO_ShifterTimerPolarityOnPositive Shift on positive edge of shift clock.
kFLEXIO_ShifterTimerPolarityOnNegitive Shift on negative edge of shift clock.

19.2.5.14 enum flexio_shifter_mode_t

Enumerator

kFLEXIO_ShifterDisabled Shifter is disabled.
kFLEXIO_ShifterModeReceive Receive mode.
kFLEXIO_ShifterModeTransmit Transmit mode.
kFLEXIO_ShifterModeMatchStore Match store mode.
kFLEXIO_ShifterModeMatchContinuous Match continuous mode.

kFLEXIO_ShifterModeState SHIFTBUF contents are used for storing programmable state attributes.
kFLEXIO_ShifterModeLogic SHIFTBUF contents are used for implementing programmable logic look up table.

19.2.5.15 enum flexio_shifter_input_source_t

Enumerator

kFLEXIO_ShifterInputFromPin Shifter input from pin.
kFLEXIO_ShifterInputFromNextShifterOutput Shifter input from Shifter N+1.

19.2.5.16 enum flexio_shifter_stop_bit_t

Enumerator

kFLEXIO_ShifterStopBitDisable Disable shifter stop bit.
kFLEXIO_ShifterStopBitLow Set shifter stop bit to logic low level.
kFLEXIO_ShifterStopBitHigh Set shifter stop bit to logic high level.

19.2.5.17 enum flexio_shifter_start_bit_t

Enumerator

kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable Disable shifter start bit, transmitter loads data on enable.
kFLEXIO_ShifterStartBitDisabledLoadDataOnShift Disable shifter start bit, transmitter loads data on first shift.
kFLEXIO_ShifterStartBitLow Set shifter start bit to logic low level.
kFLEXIO_ShifterStartBitHigh Set shifter start bit to logic high level.

19.2.5.18 enum flexio_shifter_buffer_type_t

Enumerator

kFLEXIO_ShifterBuffer Shifter Buffer N Register.
kFLEXIO_ShifterBufferBitSwapped Shifter Buffer N Bit Byte Swapped Register.
kFLEXIO_ShifterBufferByteSwapped Shifter Buffer N Byte Swapped Register.
kFLEXIO_ShifterBufferBitByteSwapped Shifter Buffer N Bit Swapped Register.
kFLEXIO_ShifterBufferNibbleByteSwapped Shifter Buffer N Nibble Byte Swapped Register.
kFLEXIO_ShifterBufferHalfWordSwapped Shifter Buffer N Half Word Swapped Register.
kFLEXIO_ShifterBufferNibbleSwapped Shifter Buffer N Nibble Swapped Register.

19.2.6 Function Documentation

19.2.6.1 void FLEXIO_GetDefaultConfig (flexio_config_t * *userConfig*)

The configuration can used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

userConfig	pointer to flexio_config_t structure

19.2.6.2 void FLEXIO_Init (FLEXIO_Type * *base*, const flexio_config_t * *userConfig*)

The configuration structure can be filled by the user or be set with default values by FLEXIO_GetDefaultConfig().

Example

```
flexio_config_t config = {
.enableFlexio = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

base	FlexIO peripheral base address
userConfig	pointer to flexio_config_t structure

19.2.6.3 void FLEXIO_Deinit (FLEXIO_Type * *base*)

Call this API to stop the FlexIO clock.

Note

After calling this API, call the FLEXO_Init to use the FlexIO module.

Parameters

base	FlexIO peripheral base address

19.2.6.4 uint32_t FLEXIO_GetInstance (FLEXIO_Type ∗ *base*)

Parameters

base	FLEXIO peripheral base address.

19.2.6.5 void FLEXIO_Reset (FLEXIO_Type ∗ *base*)

Parameters

base	FlexIO peripheral base address

19.2.6.6 static void FLEXIO_Enable (FLEXIO_Type ∗ *base,* bool *enable*) `[inline], [static]`

Parameters

base	FlexIO peripheral base address
enable	true to enable, false to disable.

19.2.6.7 static uint32_t FLEXIO_ReadPinInput (FLEXIO_Type ∗ *base*) `[inline], [static]`

Parameters

base	FlexIO peripheral base address

Returns

FlexIO pin input data

19.2.6.8 static uint8_t FLEXIO_GetShifterState (FLEXIO_Type ∗ *base*) `[inline], [static]`

Parameters

base	FlexIO peripheral base address

Returns

current State pointer

19.2.6.9 void FLEXIO_SetShifterConfig (FLEXIO_Type * *base*, uint8_t *index*, const flexio_shifter_config_t * *shifterConfig*)

The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
.timerSelect = 0,
.timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinPolarity = kFLEXIO_PinActiveLow,
.shifterMode = kFLEXIO_ShifterModeTransmit,
.inputSource = kFLEXIO_ShifterInputFromPin,
.shifterStop = kFLEXIO_ShifterStopBitHigh,
.shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

base	FlexIO peripheral base address
index	Shifter index
shifterConfig	Pointer to flexio_shifter_config_t structure

19.2.6.10 void FLEXIO_SetTimerConfig (FLEXIO_Type * *base*, uint8_t *index*, const flexio_timer_config_t * *timerConfig*)

The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
.triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFTnSTAT(0),
.triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
.triggerSource = kFLEXIO_TimerTriggerSourceInternal,
```

```
.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
      ,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);
```

Parameters

base	FlexIO peripheral base address
index	Timer index
timerConfig	Pointer to the flexio_timer_config_t structure

19.2.6.11 static void FLEXIO_EnableShifterStatusInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt generates when the corresponding SSF is set.

Parameters

base	FlexIO peripheral base address
mask	The shifter status mask which can be calculated by (1 << shifter index)

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

19.2.6.12 static void FLEXIO_DisableShifterStatusInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt won't generate when the corresponding SSF is set.

Parameters

base	FlexIO peripheral base address
mask	The shifter status mask which can be calculated by (1 << shifter index)

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

19.2.6.13 static void FLEXIO_EnableShifterErrorInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt generates when the corresponding SEF is set.

Parameters

base	FlexIO peripheral base address
mask	The shifter error mask which can be calculated by (1 << shifter index)

Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

19.2.6.14 static void FLEXIO_DisableShifterErrorInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt won't generate when the corresponding SEF is set.

Parameters

base	FlexIO peripheral base address
mask	The shifter error mask which can be calculated by (1 << shifter index)

Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

19.2.6.15 static void FLEXIO_EnableTimerStatusInterrupts (FLEXIO_Type * *base,* uint32_t *mask*) [inline], [static]

The interrupt generates when the corresponding SSF is set.

Parameters

base	FlexIO peripheral base address
mask	The timer status mask which can be calculated by (1 << timer index)

Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 << timer index0) | (1 << timer index1))

19.2.6.16 static void FLEXIO_DisableTimerStatusInterrupts (FLEXIO_Type * *base,* uint32_t *mask*) [inline], [static]

The interrupt won't generate when the corresponding SSF is set.

Parameters

base	FlexIO peripheral base address
mask	The timer status mask which can be calculated by (1 << timer index)

Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using ((1 << timer index0) | (1 << timer index1))

19.2.6.17 static uint32_t FLEXIO_GetShifterStatusFlags (FLEXIO_Type * *base*) [inline], [static]

Parameters

base	FlexIO peripheral base address

Returns

Shifter status flags

19.2.6.18 static void FLEXIO_ClearShifterStatusFlags (FLEXIO_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	FlexIO peripheral base address
mask	The shifter status mask which can be calculated by (1 << shifter index)

Note

For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

19.2.6.19 static uint32_t FLEXIO_GetShifterErrorFlags (FLEXIO_Type * *base*) [inline], [static]

Parameters

base	FlexIO peripheral base address

Returns

Shifter error flags

19.2.6.20 static void FLEXIO_ClearShifterErrorFlags (FLEXIO_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	FlexIO peripheral base address
mask	The shifter error mask which can be calculated by (1 << shifter index)

Note

For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

19.2.6.21 static uint32_t FLEXIO_GetTimerStatusFlags (FLEXIO_Type * *base*) [inline], [static]

Parameters

base	FlexIO peripheral base address

Returns

Timer status flags

19.2.6.22 static void FLEXIO_ClearTimerStatusFlags (FLEXIO_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	FlexIO peripheral base address
mask	The timer status mask which can be calculated by (1 << timer index)

Note

For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using ((1 << timer index0) | (1 << timer index1))

19.2.6.23 static void FLEXIO_EnableShifterStatusDMA (FLEXIO_Type * *base,* uint32_t *mask,* bool *enable*) [inline], [static]

The DMA request generates when the corresponding SSF is set.

Note

For multiple shifter status DMA enables, for example, calculate the mask by using ((1 << shifter index0) | (1 << shifter index1))

Parameters

base	FlexIO peripheral base address
mask	The shifter status mask which can be calculated by (1 << shifter index)

enable	True to enable, false to disable.

19.2.6.24 uint32_t FLEXIO_GetShifterBufferAddress (FLEXIO_Type * *base*, flexio_shifter_buffer_type_t *type*, uint8_t *index*)

Parameters

base	FlexIO peripheral base address
type	Shifter type of flexio_shifter_buffer_type_t
index	Shifter index

Returns

Corresponding shifter buffer index

19.2.6.25 status_t FLEXIO_RegisterHandleIRQ (void * *base*, void * *handle*, flexio_isr_t *isr*)

Parameters

base	Pointer to the FlexIO simulated peripheral type.
handle	Pointer to the handler for FlexIO simulated peripheral.
isr	FlexIO simulated peripheral interrupt handler.

Return values

kStatus_Success	Successfully create the handle.
kStatus_OutOfRange	The FlexIO type/handle/ISR table out of range.

19.2.6.26 status_t FLEXIO_UnregisterHandleIRQ (void * *base*)

Parameters

base	Pointer to the FlexIO simulated peripheral type.

Return values

kStatus_Success	Successfully create the handle.
kStatus_OutOfRange	The FlexIO type/handle/ISR table out of range.

19.2.7 Variable Documentation

19.2.7.1 FLEXIO_Type∗ const s_flexioBases[]

19.2.7.2 const clock_ip_name_t s_flexioClocks[]

19.3 FlexIO I2C Master Driver

19.3.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2C master function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO I2C master driver includes functional APIs and transactional APIs.

Functional APIs target low level APIs. Functional APIs can be used for the FlexIO I2C master initialization/configuration/operation for the optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO I2C master peripheral and how to organize functional APIs to meet the application requirements. The FlexIO I2C master functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support an asynchronous transfer. This means that the functions FLEXIO_I2C_-MasterTransferNonBlocking() set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_Success status.

19.3.2 Typical use case

19.3.2.1 FlexIO I2C master transfer using an interrupt method

```
flexio_i2c_master_handle_t g_m_handle;
flexio_i2c_master_config_t masterConfig;
flexio_i2c_master_transfer_t masterXfer;
volatile bool completionFlag = false;
const uint8_t sendData[] = [......];
FLEXIO_I2C_Type i2cDev;

void FLEXIO_I2C_MasterCallback(FLEXIO_I2C_Type *base, status_t status, void *
      userData)
{
    userData = userData;

    if (kStatus_Success == status)
    {
        completionFlag = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2C_MasterGetDefaultConfig(&masterConfig);

    FLEXIO_I2C_MasterInit(&i2cDev, &user_config);
    FLEXIO_I2C_MasterTransferCreateHandle(&i2cDev, &g_m_handle,
      FLEXIO_I2C_MasterCallback, NULL);
```

```
    // Prepares to send.
    masterXfer.slaveAddress = g_accel_address[0];
    masterXfer.direction = kI2C_Read;
    masterXfer.subaddress = &who_am_i_reg;
    masterXfer.subaddressSize = 1;
    masterXfer.data = &who_am_i_value;
    masterXfer.dataSize = 1;
    masterXfer.flags = kI2C_TransferDefaultFlag;

    // Sends out.
    FLEXIO_I2C_MasterTransferNonBlocking(&i2cDev, &g_m_handle, &
      masterXfer);

    // Wait for sending is complete.
    while (!completionFlag)
    {
    }

    // ...
}
```

Data Structures

- struct FLEXIO_I2C_Type
  Define FlexIO I2C master access structure typedef. More...
- struct flexio_i2c_master_config_t
  Define FlexIO I2C master user configuration structure. More...
- struct flexio_i2c_master_transfer_t
  Define FlexIO I2C master transfer structure. More...
- struct flexio_i2c_master_handle_t
  Define FlexIO I2C master handle structure. More...

Macros

- #define I2C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef void(∗ flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type ∗base, flexio_i2c_-master_handle_t ∗handle, status_t status, void ∗userData)
  FlexIO I2C master transfer callback typedef.

Enumerations

- enum {
  kStatus_FLEXIO_I2C_Busy = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 0),
  kStatus_FLEXIO_I2C_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 1),
  kStatus_FLEXIO_I2C_Nak = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 2),

kStatus_FLEXIO_I2C_Timeout = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 3) }
FlexIO I2C transfer status.

- enum _flexio_i2c_master_interrupt {
  kFLEXIO_I2C_TxEmptyInterruptEnable = 0x1U,
  kFLEXIO_I2C_RxFullInterruptEnable = 0x2U }
  Define FlexIO I2C master interrupt mask.
- enum _flexio_i2c_master_status_flags {
  kFLEXIO_I2C_TxEmptyFlag = 0x1U,
  kFLEXIO_I2C_RxFullFlag = 0x2U,
  kFLEXIO_I2C_ReceiveNakFlag = 0x4U }
  Define FlexIO I2C master status mask.
- enum flexio_i2c_direction_t {
  kFLEXIO_I2C_Write = 0x0U,
  kFLEXIO_I2C_Read = 0x1U }
  Direction of master transfer.

Driver version

- #define **FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION** (MAKE_VERSION(2, 4, 0))

Initialization and deinitialization

- status_t FLEXIO_I2C_CheckForBusyBus (FLEXIO_I2C_Type *base)
  Make sure the bus isn't already pulled down.
- status_t FLEXIO_I2C_MasterInit (FLEXIO_I2C_Type *base, flexio_i2c_master_config_t *masterConfig, uint32_t srcClock_Hz)
  Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.
- void FLEXIO_I2C_MasterDeinit (FLEXIO_I2C_Type *base)
  De-initializes the FlexIO I2C master peripheral.
- void FLEXIO_I2C_MasterGetDefaultConfig (flexio_i2c_master_config_t *masterConfig)
  Gets the default configuration to configure the FlexIO module.
- static void FLEXIO_I2C_MasterEnable (FLEXIO_I2C_Type *base, bool enable)
  Enables/disables the FlexIO module operation.

Status

- uint32_t FLEXIO_I2C_MasterGetStatusFlags (FLEXIO_I2C_Type *base)
  Gets the FlexIO I2C master status flags.
- void FLEXIO_I2C_MasterClearStatusFlags (FLEXIO_I2C_Type *base, uint32_t mask)
  Clears the FlexIO I2C master status flags.

Interrupts

- void FLEXIO_I2C_MasterEnableInterrupts (FLEXIO_I2C_Type *base, uint32_t mask)

Enables the FlexIO i2c master interrupt requests.

- void FLEXIO_I2C_MasterDisableInterrupts (FLEXIO_I2C_Type *base, uint32_t mask)
  Disables the FlexIO I2C master interrupt requests.

Bus Operations

- void FLEXIO_I2C_MasterSetBaudRate (FLEXIO_I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the FlexIO I2C master transfer baudrate.
- void FLEXIO_I2C_MasterStart (FLEXIO_I2C_Type *base, uint8_t address, flexio_i2c_direction_t direction)
  Sends START + 7-bit address to the bus.
- void FLEXIO_I2C_MasterStop (FLEXIO_I2C_Type *base)
  Sends the stop signal on the bus.
- void FLEXIO_I2C_MasterRepeatedStart (FLEXIO_I2C_Type *base)
  Sends the repeated start signal on the bus.
- void FLEXIO_I2C_MasterAbortStop (FLEXIO_I2C_Type *base)
  Sends the stop signal when transfer is still on-going.
- void FLEXIO_I2C_MasterEnableAck (FLEXIO_I2C_Type *base, bool enable)
  Configures the sent ACK/NAK for the following byte.
- status_t FLEXIO_I2C_MasterSetTransferCount (FLEXIO_I2C_Type *base, uint16_t count)
  Sets the number of bytes to be transferred from a start signal to a stop signal.
- static void FLEXIO_I2C_MasterWriteByte (FLEXIO_I2C_Type *base, uint32_t data)
  Writes one byte of data to the I2C bus.
- static uint8_t FLEXIO_I2C_MasterReadByte (FLEXIO_I2C_Type *base)
  Reads one byte of data from the I2C bus.
- status_t FLEXIO_I2C_MasterWriteBlocking (FLEXIO_I2C_Type *base, const uint8_t *txBuff, uint8_t txSize)
  Sends a buffer of data in bytes.
- status_t FLEXIO_I2C_MasterReadBlocking (FLEXIO_I2C_Type *base, uint8_t *rxBuff, uint8_t rxSize)
  Receives a buffer of bytes.
- status_t FLEXIO_I2C_MasterTransferBlocking (FLEXIO_I2C_Type *base, flexio_i2c_master_-transfer_t *xfer)
  Performs a master polling transfer on the I2C bus.

Transactional

- status_t FLEXIO_I2C_MasterTransferCreateHandle (FLEXIO_I2C_Type *base, flexio_i2c_-master_handle_t *handle, flexio_i2c_master_transfer_callback_t callback, void *userData)
  Initializes the I2C handle which is used in transactional functions.
- status_t FLEXIO_I2C_MasterTransferNonBlocking (FLEXIO_I2C_Type *base, flexio_i2c_-master_handle_t *handle, flexio_i2c_master_transfer_t *xfer)
  Performs a master interrupt non-blocking transfer on the I2C bus.
- status_t FLEXIO_I2C_MasterTransferGetCount (FLEXIO_I2C_Type *base, flexio_i2c_master_-handle_t *handle, size_t *count)
  Gets the master transfer status during a interrupt non-blocking transfer.

- void FLEXIO_I2C_MasterTransferAbort (FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle)
  Aborts an interrupt non-blocking transfer early.
- void FLEXIO_I2C_MasterTransferHandleIRQ (void *i2cType, void *i2cHandle)
  Master interrupt handler.

19.3.3 Data Structure Documentation

19.3.3.1 struct FLEXIO_I2C_Type

Data Fields

- FLEXIO_Type * flexioBase
  FlexIO base pointer.
- uint8_t SDAPinIndex
  Pin select for I2C SDA.
- uint8_t SCLPinIndex
  Pin select for I2C SCL.
- uint8_t shifterIndex [2]
  Shifter index used in FlexIO I2C.
- uint8_t timerIndex [3]
  Timer index used in FlexIO I2C.
- uint32_t baudrate
  Master transfer baudrate, used to calculate delay time.

Field Documentation

(1) FLEXIO_Type* FLEXIO_I2C_Type::flexioBase

(2) uint8_t FLEXIO_I2C_Type::SDAPinIndex

(3) uint8_t FLEXIO_I2C_Type::SCLPinIndex

(4) uint8_t FLEXIO_I2C_Type::shifterIndex[2]

(5) uint8_t FLEXIO_I2C_Type::timerIndex[3]

(6) uint32_t FLEXIO_I2C_Type::baudrate

19.3.3.2 struct flexio_i2c_master_config_t

Data Fields

- bool enableMaster
  Enables the FlexIO I2C peripheral at initialization time.
- bool enableInDoze
  Enable/disable FlexIO operation in doze mode.
- bool enableInDebug
  Enable/disable FlexIO operation in debug mode.

- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers, fast access requires` *the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32_t baudRate_Bps
  Baud rate in Bps.

Field Documentation

(1) bool flexio_i2c_master_config_t::enableMaster

(2) bool flexio_i2c_master_config_t::enableInDoze

(3) bool flexio_i2c_master_config_t::enableInDebug

(4) bool flexio_i2c_master_config_t::enableFastAccess

(5) uint32_t flexio_i2c_master_config_t::baudRate_Bps

19.3.3.3 struct flexio_i2c_master_transfer_t

Data Fields

- uint32_t flags
  Transfer flag which controls the transfer, reserved for FlexIO I2C.
- uint8_t slaveAddress
  7-bit slave address.
- flexio_i2c_direction_t direction
  Transfer direction, read or write.
- uint32_t subaddress
  Sub address.
- uint8_t subaddressSize
  Size of command buffer.
- uint8_t volatile * data
  Transfer buffer.
- volatile size_t dataSize
  Transfer size.

Field Documentation

(1) uint32_t flexio_i2c_master_transfer_t::flags

(2) uint8_t flexio_i2c_master_transfer_t::slaveAddress

(3) flexio_i2c_direction_t flexio_i2c_master_transfer_t::direction

(4) uint32_t flexio_i2c_master_transfer_t::subaddress

Transferred MSB first.

(5) uint8_t flexio_i2c_master_transfer_t::subaddressSize

(6) uint8_t volatile∗ flexio_i2c_master_transfer_t::data

(7) volatile size_t flexio_i2c_master_transfer_t::dataSize

19.3.3.4 struct _flexio_i2c_master_handle

FlexIO I2C master handle typedef.

Data Fields

- flexio_i2c_master_transfer_t transfer
  FlexIO I2C master transfer copy.
- size_t transferSize
  Total bytes to be transferred.
- uint8_t state
  Transfer state maintained during transfer.
- flexio_i2c_master_transfer_callback_t completionCallback
  Callback function called at transfer event.
- void ∗ userData
  Callback parameter passed to callback function.
- bool needRestart
  Whether master needs to send re-start signal.

Field Documentation

(1) flexio_i2c_master_transfer_t flexio_i2c_master_handle_t::transfer

(2) size_t flexio_i2c_master_handle_t::transferSize

(3) uint8_t flexio_i2c_master_handle_t::state

(4) flexio_i2c_master_transfer_callback_t flexio_i2c_master_handle_t::completionCallback

Callback function called at transfer event.

(5) void∗ flexio_i2c_master_handle_t::userData

(6) bool flexio_i2c_master_handle_t::needRestart

19.3.4 Macro Definition Documentation

19.3.4.1 #define I2C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/

19.3.5 Typedef Documentation

19.3.5.1 typedef void(∗ flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type ∗base, flexio_i2c_master_handle_t ∗handle, status_t status, void ∗userData)

19.3.6 Enumeration Type Documentation

19.3.6.1 anonymous enum

Enumerator

kStatus_FLEXIO_I2C_Busy I2C is busy doing transfer.
kStatus_FLEXIO_I2C_Idle I2C is busy doing transfer.
kStatus_FLEXIO_I2C_Nak NAK received during transfer.
kStatus_FLEXIO_I2C_Timeout Timeout polling status flags.

19.3.6.2 enum _flexio_i2c_master_interrupt

Enumerator

kFLEXIO_I2C_TxEmptyInterruptEnable Tx buffer empty interrupt enable.
kFLEXIO_I2C_RxFullInterruptEnable Rx buffer full interrupt enable.

19.3.6.3 enum _flexio_i2c_master_status_flags

Enumerator

kFLEXIO_I2C_TxEmptyFlag Tx shifter empty flag.
kFLEXIO_I2C_RxFullFlag Rx shifter full/Transfer complete flag.
kFLEXIO_I2C_ReceiveNakFlag Receive NAK flag.

19.3.6.4 enum flexio_i2c_direction_t

Enumerator

kFLEXIO_I2C_Write Master send to slave.
kFLEXIO_I2C_Read Master receive from slave.

19.3.7 Function Documentation

19.3.7.1 status_t FLEXIO_I2C_CheckForBusyBus (FLEXIO_I2C_Type ∗ *base*)

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

Parameters

base	Pointer to FLEXIO_I2C_Type structure..

Return values

kStatus_Success	
kStatus_FLEXIO_I2C_-Busy	

19.3.7.2 status_t FLEXIO_I2C_MasterInit (FLEXIO_I2C_Type ∗ *base*, flexio_i2c_master_config_t ∗ *masterConfig*, uint32_t *srcClock_Hz*)

Example

```
FLEXIO_I2C_Type base = {
.flexioBase = FLEXIO,
.SDAPinIndex = 0,
.SCLPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
masterConfig	Pointer to flexio_i2c_master_config_t structure.
srcClock_Hz	FlexIO source clock in Hz.

Return values

kStatus_Success	Initialization successful
kStatus_InvalidArgument	The source clock exceed upper range limitation

19.3.7.3 void FLEXIO_I2C_MasterDeinit (FLEXIO_I2C_Type ∗ *base*)

Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO_I2C_MasterInit is called.

Parameters

base	pointer to FLEXIO_I2C_Type structure.

19.3.7.4 void FLEXIO_I2C_MasterGetDefaultConfig (flexio_i2c_master_config_t * *masterConfig*)

The configuration can be used directly for calling the FLEXIO_I2C_MasterInit().

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

Parameters

masterConfig	Pointer to flexio_i2c_master_config_t structure.

19.3.7.5 static void FLEXIO_I2C_MasterEnable (FLEXIO_I2C_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
enable	Pass true to enable module, false does not have any effect.

19.3.7.6 uint32_t FLEXIO_I2C_MasterGetStatusFlags (FLEXIO_I2C_Type * *base*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure

Returns

Status flag, use status flag to AND _flexio_i2c_master_status_flags can get the related status.

19.3.7.7 void FLEXIO_I2C_MasterClearStatusFlags (FLEXIO_I2C_Type * *base*, uint32_t *mask*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
mask	Status flag. The parameter can be any combination of the following values: • kFLEXIO_I2C_RxFullFlag • kFLEXIO_I2C_ReceiveNakFlag

19.3.7.8 void FLEXIO_I2C_MasterEnableInterrupts (FLEXIO_I2C_Type * *base*, uint32_t *mask*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
mask	Interrupt source. Currently only one interrupt request source: • kFLEXIO_I2C_TransferCompleteInterruptEnable

19.3.7.9 void FLEXIO_I2C_MasterDisableInterrupts (FLEXIO_I2C_Type * *base*, uint32_t *mask*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
mask	Interrupt source.

19.3.7.10 void FLEXIO_I2C_MasterSetBaudRate (FLEXIO_I2C_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure
baudRate_Bps	the baud rate value in HZ

srcClock_Hz	source clock in HZ

19.3.7.11 void FLEXIO_I2C_MasterStart (FLEXIO_I2C_Type * *base*, uint8_t *address*, flexio_i2c_direction_t *direction*)

Note

This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO_I2C_-RxFullFlag status is asserted before calling this API.

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
address	7-bit address.
direction	transfer direction. This parameter is one of the values in flexio_i2c_direction_t: • kFLEXIO_I2C_Write: Transmit • kFLEXIO_I2C_Read: Receive

19.3.7.12 void FLEXIO_I2C_MasterStop (FLEXIO_I2C_Type * *base*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.

19.3.7.13 void FLEXIO_I2C_MasterRepeatedStart (FLEXIO_I2C_Type * *base*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.

19.3.7.14 void FLEXIO_I2C_MasterAbortStop (FLEXIO_I2C_Type * *base*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.

19.3.7.15 void FLEXIO_I2C_MasterEnableAck (FLEXIO_I2C_Type * *base*, bool *enable*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
enable	True to configure send ACK, false configure to send NAK.

19.3.7.16 status_t FLEXIO_I2C_MasterSetTransferCount (FLEXIO_I2C_Type * *base*, uint16_t *count*)

Note

Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
count	Number of bytes need to be transferred from a start signal to a re-start/stop signal

Return values

kStatus_Success	Successfully configured the count.
kStatus_InvalidArgument	Input argument is invalid.

19.3.7.17 static void FLEXIO_I2C_MasterWriteByte (FLEXIO_I2C_Type * *base*, uint32_t *data*) `[inline], [static]`

Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
data	a byte of data.

19.3.7.18 static uint8_t FLEXIO_I2C_MasterReadByte (FLEXIO_I2C_Type ∗ *base*) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

Parameters

base	Pointer to FLEXIO_I2C_Type structure.

Returns

data byte read.

19.3.7.19 status_t FLEXIO_I2C_MasterWriteBlocking (FLEXIO_I2C_Type ∗ *base,* const uint8_t ∗ *txBuff,* uint8_t *txSize*)

Note

This function blocks via polling until all bytes have been sent.

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
txBuff	The data bytes to send.
txSize	The number of data bytes to send.

Return values

kStatus_Success	Successfully write data.
kStatus_FLEXIO_I2C_-Nak	Receive NAK during writing data.
kStatus_FLEXIO_I2C_-Timeout	Timeout polling status flags.

19.3.7.20 status_t FLEXIO_I2C_MasterReadBlocking (FLEXIO_I2C_Type * *base*, uint8_t * *rxBuff*, uint8_t *rxSize*)

Note

This function blocks via polling until all bytes have been received.

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
rxBuff	The buffer to store the received bytes.
rxSize	The number of data bytes to be received.

Return values

kStatus_Success	Successfully read data.
kStatus_FLEXIO_I2C_-Timeout	Timeout polling status flags.

19.3.7.21 status_t FLEXIO_I2C_MasterTransferBlocking (FLEXIO_I2C_Type * *base*, flexio_i2c_master_transfer_t * *xfer*)

Note

The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

base	pointer to FLEXIO_I2C_Type structure.
xfer	pointer to flexio_i2c_master_transfer_t structure.

Returns

status of status_t.

19.3.7.22 status_t FLEXIO_I2C_MasterTransferCreateHandle (FLEXIO_I2C_Type * *base*, flexio_i2c_master_handle_t * *handle*, flexio_i2c_master_transfer_callback_t *callback*, void * *userData*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
handle	Pointer to flexio_i2c_master_handle_t structure to store the transfer state.
callback	Pointer to user callback function.
userData	User param passed to the callback function.

Return values

kStatus_Success	Successfully create the handle.
kStatus_OutOfRange	The FlexIO type/handle/isr table out of range.

19.3.7.23 status_t FLEXIO_I2C_MasterTransferNonBlocking (FLEXIO_I2C_Type ∗ *base*, flexio_i2c_master_handle_t ∗ *handle*, flexio_i2c_master_transfer_t ∗ *xfer*)

Note

The API returns immediately after the transfer initiates. Call FLEXIO_I2C_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_FLEXIO_I2C_Busy, the transfer is finished.

Parameters

base	Pointer to FLEXIO_I2C_Type structure
handle	Pointer to flexio_i2c_master_handle_t structure which stores the transfer state
xfer	pointer to flexio_i2c_master_transfer_t structure

Return values

kStatus_Success	Successfully start a transfer.
kStatus_FLEXIO_I2C_Busy	FlexIO I2C is not idle, is running another transfer.

19.3.7.24 status_t FLEXIO_I2C_MasterTransferGetCount (FLEXIO_I2C_Type ∗ *base*, flexio_i2c_master_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	Pointer to FLEXIO_I2C_Type structure.
handle	Pointer to flexio_i2c_master_handle_t structure which stores the transfer state.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.
kStatus_Success	Successfully return the count.

19.3.7.25 void FLEXIO_I2C_MasterTransferAbort (FLEXIO_I2C_Type ∗ *base*, flexio_i2c_master_handle_t ∗ *handle*)

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

base	Pointer to FLEXIO_I2C_Type structure
handle	Pointer to flexio_i2c_master_handle_t structure which stores the transfer state

19.3.7.26 void FLEXIO_I2C_MasterTransferHandleIRQ (void ∗ *i2cType*, void ∗ *i2cHandle*)

Parameters

i2cType	Pointer to FLEXIO_I2C_Type structure
i2cHandle	Pointer to flexio_i2c_master_transfer_t structure

19.4 FlexIO MCU Interface LCD Driver

19.4.1 Overview

The MCUXpresso SDK provides a peripheral driver for LCD (8080 or 6800 interface) function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO LCD driver supports both 8-bit and 16-bit data bus, 8080 and 6800 interface. User could change the macro FLEXIO_MCULCD_DATA_BUS_WIDTH to choose 8-bit data bus or 16-bit data bus.

The FlexIO LCD driver supports three kinds of data transfer:

1. Send a data array. For example, send the LCD image data to the LCD controller.
2. Send a value many times. For example, send 0 many times to clean the LCD screen.
3. Read data into a data array. For example, read image from LCD controller.

The FlexIO LCD driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FlexIO LCD initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO LCD peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FlexIO LCD functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code.

Transactional APIs support asynchronous transfer. This means that the function FLEXIO_MCULCD_-TransferNonBlocking sets up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_FLEXIO_MCULCD_Idle status.

19.4.2 Typical use case

19.4.2.1 FlexIO LCD send/receive using functional APIs

This example shows how to send command, or write and read data using the functional APIs. The data bus is 16-bit.

```
uint16_t dataToSend[] = { ... };
uint16_t dataToReceive[] = { ... };

FLEXIO_MCULCD_Type flexioLcdDev;
flexio_MCULCD_transfer_t xfer;
flexio_MCULCD_config_t config;

FLEXIO_MCULCD_GetDefaultConfig(&config);
FLEXIO_MCULCD_Init(&flexioLcdDev, &config, 120000000);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
```

```
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command1);
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command1;
xfer.dataCount = 0; // Only send command, no data transfer.
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteDataArrayBlocking(&flexioLcdDev, dataToSend, sizeof(
      dataToSend));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
xfer.dataCount = sizeof(dataToSend);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteSameValueBlocking(&flexioLcdDev, value, 1000); //
       Send value 1000 times
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
xfer.dataAddrOrSameValue = value;
xfer.dataCount = 1000;
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command3);
FLEXIO_MCULCD_ReadDataArrayBlocking(&flexioLcdDev, dataToReceive, sizeof
      (dataToReceive));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command3;
xfer.mode = kFLEXIO_MCULCD_ReadArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
xfer.dataCount = sizeof(dataToReceive);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);
```

19.4.2.2 FlexIO LCD send/receive using interrupt transactional APIs

```
flexio_MCULCD_handle_t handle;
volatile bool completeFlag = false;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_handle_t *handle,
      status_t status, void *userData)
{
    if (kStatus_FLEXIO_MCULCD_Idle == status)
    {
        completeFlag = true;
    }
}

void main(void)
```

```
{
    // Init the FlexIO LCD driver.
    FLEXIO_MCULCD_Init(...);

    // Create the transactional handle.
    FLEXIO_MCULCD_TransferCreateHandle(&flexioLcdDev, &handle,
      flexioLcdCallback, NULL);

    xfer.command = command1;
    xfer.dataCount = 0; // Only send command, no data transfer.
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &xfer);

    // When only send method, it is not necessary to wait for the callback,
    // because the command is sent using a blocking method internally. The
    // command has been sent out after the function FLEXIO_MCULCD_TransferNonBlocking
    // returns.
    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
    xfer.dataCount = sizeof(dataToSend);
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
    xfer.dataAddrOrSameValue = value;
    xfer.dataCount = 1000;
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }

    xfer.command = command3;
    xfer.mode = kFLEXIO_MCULCD_ReadArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
    xfer.dataCount = sizeof(dataToReceive);
    completeFlag = false;
    FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

    while (!completeFlag)
    {
    }
}
```

Modules

- FlexIO DMA MCU Interface LCD Driver
- FlexIO SMARTDMA MCU Interface LCD Driver

Data Structures

- struct FLEXIO_MCULCD_Type
  Define FlexIO MCULCD access structure typedef. More...
- struct flexio_mculcd_config_t
  Define FlexIO MCULCD configuration structure. More...
- struct flexio_mculcd_transfer_t
  Define FlexIO MCULCD transfer structure. More...
- struct flexio_mculcd_handle_t
  Define FlexIO MCULCD handle structure. More...

Macros

- #define FLEXIO_MCULCD_WAIT_COMPLETE_TIME 512
  The delay time to wait for FLEXIO transmit complete.
- #define FLEXIO_MCULCD_DATA_BUS_WIDTH 16UL
  The data bus width, must be 8 or 16.

Typedefs

- typedef void(∗ flexio_mculcd_pin_func_t)(bool set)
  Function to set or clear the CS and RS pin.
- typedef void(∗ flexio_mculcd_transfer_callback_t)(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_handle_t ∗handle, status_t status, void ∗userData)
  FlexIO MCULCD callback for finished transfer.

Enumerations

- enum {
  kStatus_FLEXIO_MCULCD_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 0),
  kStatus_FLEXIO_MCULCD_Busy = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 1),
  kStatus_FLEXIO_MCULCD_Error = MAKE_STATUS(kStatusGroup_FLEXIO_MCULCD, 2) }
  FlexIO LCD transfer status.
- enum flexio_mculcd_pixel_format_t {
  kFLEXIO_MCULCD_RGB565 = 0,
  kFLEXIO_MCULCD_BGR565,
  kFLEXIO_MCULCD_RGB888,
  kFLEXIO_MCULCD_BGR888 }
  Define FlexIO MCULCD pixel format.
- enum flexio_mculcd_bus_t {
  kFLEXIO_MCULCD_8080,
  kFLEXIO_MCULCD_6800 }
  Define FlexIO MCULCD bus type.
- enum _flexio_mculcd_interrupt_enable {
  kFLEXIO_MCULCD_TxEmptyInterruptEnable = (1U << 0U),

kFLEXIO_MCULCD_RxFullInterruptEnable = (1U << 1U) }
Define FlexIO MCULCD interrupt mask.

- enum _flexio_mculcd_status_flags {
  kFLEXIO_MCULCD_TxEmptyFlag = (1U << 0U),
  kFLEXIO_MCULCD_RxFullFlag = (1U << 1U) }
  Define FlexIO MCULCD status mask.
- enum _flexio_mculcd_dma_enable {
  kFLEXIO_MCULCD_TxDmaEnable = 0x1U,
  kFLEXIO_MCULCD_RxDmaEnable = 0x2U }
  Define FlexIO MCULCD DMA mask.
- enum flexio_mculcd_transfer_mode_t {
  kFLEXIO_MCULCD_ReadArray,
  kFLEXIO_MCULCD_WriteArray,
  kFLEXIO_MCULCD_WriteSameValue }
  Transfer mode.

Driver version

- #define FSL_FLEXIO_MCULCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))
  FlexIO MCULCD driver version.

FlexIO MCULCD Configuration

- status_t FLEXIO_MCULCD_Init (FLEXIO_MCULCD_Type *base, flexio_mculcd_config_t *config, uint32_t srcClock_Hz)
  Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO MCULCD hardware, and configures the FlexIO MCULCD with FlexIO MCULCD configuration.
- void FLEXIO_MCULCD_Deinit (FLEXIO_MCULCD_Type *base)
  Resets the FLEXIO_MCULCD timer and shifter configuration.
- void FLEXIO_MCULCD_GetDefaultConfig (flexio_mculcd_config_t *config)
  Gets the default configuration to configure the FlexIO MCULCD.

Status

- uint32_t FLEXIO_MCULCD_GetStatusFlags (FLEXIO_MCULCD_Type *base)
  Gets FlexIO MCULCD status flags.
- void FLEXIO_MCULCD_ClearStatusFlags (FLEXIO_MCULCD_Type *base, uint32_t mask)
  Clears FlexIO MCULCD status flags.

Interrupts

- void FLEXIO_MCULCD_EnableInterrupts (FLEXIO_MCULCD_Type *base, uint32_t mask)
  Enables the FlexIO MCULCD interrupt.
- void FLEXIO_MCULCD_DisableInterrupts (FLEXIO_MCULCD_Type *base, uint32_t mask)

Disables the FlexIO MCULCD interrupt.

DMA Control

- static void FLEXIO_MCULCD_EnableTxDMA (FLEXIO_MCULCD_Type ∗base, bool enable)
  Enables/disables the FlexIO MCULCD transmit DMA.
- static void FLEXIO_MCULCD_EnableRxDMA (FLEXIO_MCULCD_Type ∗base, bool enable)
  Enables/disables the FlexIO MCULCD receive DMA.
- static uint32_t FLEXIO_MCULCD_GetTxDataRegisterAddress (FLEXIO_MCULCD_Type ∗base)
  Gets the FlexIO MCULCD transmit data register address.
- static uint32_t FLEXIO_MCULCD_GetRxDataRegisterAddress (FLEXIO_MCULCD_Type ∗base)
  Gets the FlexIO MCULCD receive data register address.

Bus Operations

- status_t FLEXIO_MCULCD_SetBaudRate (FLEXIO_MCULCD_Type ∗base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Set desired baud rate.
- void FLEXIO_MCULCD_SetSingleBeatWriteConfig (FLEXIO_MCULCD_Type ∗base)
  Configures the FLEXIO MCULCD to multiple beats write mode.
- void FLEXIO_MCULCD_ClearSingleBeatWriteConfig (FLEXIO_MCULCD_Type ∗base)
  Clear the FLEXIO MCULCD multiple beats write mode configuration.
- void FLEXIO_MCULCD_SetSingleBeatReadConfig (FLEXIO_MCULCD_Type ∗base)
  Configures the FLEXIO MCULCD to multiple beats read mode.
- void FLEXIO_MCULCD_ClearSingleBeatReadConfig (FLEXIO_MCULCD_Type ∗base)
  Clear the FLEXIO MCULCD multiple beats read mode configuration.
- void FLEXIO_MCULCD_SetMultiBeatsWriteConfig (FLEXIO_MCULCD_Type ∗base)
  Configures the FLEXIO MCULCD to multiple beats write mode.
- void FLEXIO_MCULCD_ClearMultiBeatsWriteConfig (FLEXIO_MCULCD_Type ∗base)
  Clear the FLEXIO MCULCD multiple beats write mode configuration.
- void FLEXIO_MCULCD_SetMultiBeatsReadConfig (FLEXIO_MCULCD_Type ∗base)
  Configures the FLEXIO MCULCD to multiple beats read mode.
- void FLEXIO_MCULCD_ClearMultiBeatsReadConfig (FLEXIO_MCULCD_Type ∗base)
  Clear the FLEXIO MCULCD multiple beats read mode configuration.
- static void FLEXIO_MCULCD_Enable (FLEXIO_MCULCD_Type ∗base, bool enable)
  Enables/disables the FlexIO MCULCD module operation.
- uint32_t FLEXIO_MCULCD_ReadData (FLEXIO_MCULCD_Type ∗base)
  Read data from the FLEXIO MCULCD RX shifter buffer.
- static void FLEXIO_MCULCD_WriteData (FLEXIO_MCULCD_Type ∗base, uint32_t data)
  Write data into the FLEXIO MCULCD TX shifter buffer.
- static void FLEXIO_MCULCD_StartTransfer (FLEXIO_MCULCD_Type ∗base)
  Assert the nCS to start transfer.
- static void FLEXIO_MCULCD_StopTransfer (FLEXIO_MCULCD_Type ∗base)
  De-assert the nCS to stop transfer.
- void FLEXIO_MCULCD_WaitTransmitComplete (void)
  Wait for transmit data send out finished.

- void FLEXIO_MCULCD_WriteCommandBlocking (FLEXIO_MCULCD_Type *base, uint32_t command)
  Send command in blocking way.
- void FLEXIO_MCULCD_WriteDataArrayBlocking (FLEXIO_MCULCD_Type *base, const void *data, size_t size)
  Send data array in blocking way.
- void FLEXIO_MCULCD_ReadDataArrayBlocking (FLEXIO_MCULCD_Type *base, void *data, size_t size)
  Read data into array in blocking way.
- void FLEXIO_MCULCD_WriteSameValueBlocking (FLEXIO_MCULCD_Type *base, uint32_t sameValue, size_t size)
  Send the same value many times in blocking way.
- void FLEXIO_MCULCD_TransferBlocking (FLEXIO_MCULCD_Type *base, flexio_mculcd_transfer_t *xfer)
  Performs a polling transfer.

Transactional

- status_t FLEXIO_MCULCD_TransferCreateHandle (FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, flexio_mculcd_transfer_callback_t callback, void *userData)
  Initializes the FlexIO MCULCD handle, which is used in transactional functions.
- status_t FLEXIO_MCULCD_TransferNonBlocking (FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, flexio_mculcd_transfer_t *xfer)
  Transfer data using IRQ.
- void FLEXIO_MCULCD_TransferAbort (FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle)
  Aborts the data transfer, which used IRQ.
- status_t FLEXIO_MCULCD_TransferGetCount (FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, size_t *count)
  Gets the data transfer status which used IRQ.
- void FLEXIO_MCULCD_TransferHandleIRQ (void *base, void *handle)
  FlexIO MCULCD IRQ handler function.

19.4.3 Data Structure Documentation

19.4.3.1 struct FLEXIO_MCULCD_Type

Data Fields

- FLEXIO_Type * flexioBase
  FlexIO base pointer.
- flexio_mculcd_bus_t busType
  The bus type, 8080 or 6800.
- uint8_t dataPinStartIndex
  Start index of the data pin, the FlexIO pin dataPinStartIndex to (dataPinStartIndex + FLEXIO_MCULCD_DATA_BUS_WIDTH -1) will be used for data transfer.

- uint8_t ENWRPinIndex
  Pin select for WR(8080 mode), EN(6800 mode).
- uint8_t RDPinIndex
  Pin select for RD(8080 mode), not used in 6800 mode.
- uint8_t txShifterStartIndex
  Start index of shifters used for data write, it must be 0 or 4.
- uint8_t txShifterEndIndex
  End index of shifters used for data write.
- uint8_t rxShifterStartIndex
  Start index of shifters used for data read.
- uint8_t rxShifterEndIndex
  End index of shifters used for data read, it must be 3 or 7.
- uint8_t timerIndex
  Timer index used in FlexIO MCULCD.
- flexio_mculcd_pin_func_t setCSPin
  Function to set or clear the CS pin.
- flexio_mculcd_pin_func_t setRSPin
  Function to set or clear the RS pin.
- flexio_mculcd_pin_func_t setRDWRPin
  Function to set or clear the RD/WR pin, only used in 6800 mode.

Field Documentation

(1) FLEXIO_Type* FLEXIO_MCULCD_Type::flexioBase

(2) flexio_mculcd_bus_t FLEXIO_MCULCD_Type::busType

(3) uint8_t FLEXIO_MCULCD_Type::dataPinStartIndex

Only support data bus width 8 and 16.

(4) uint8_t FLEXIO_MCULCD_Type::ENWRPinIndex

(5) uint8_t FLEXIO_MCULCD_Type::RDPinIndex

(6) uint8_t FLEXIO_MCULCD_Type::txShifterStartIndex

(7) uint8_t FLEXIO_MCULCD_Type::txShifterEndIndex

(8) uint8_t FLEXIO_MCULCD_Type::rxShifterStartIndex

(9) uint8_t FLEXIO_MCULCD_Type::rxShifterEndIndex

(10) uint8_t FLEXIO_MCULCD_Type::timerIndex

(11) flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setCSPin

(12) flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setRSPin

(13) flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setRDWRPin

19.4.3.2 struct flexio_mculcd_config_t

Data Fields

- bool enable
  Enable/disable FlexIO MCULCD after configuration.
- bool enableInDoze
  Enable/disable FlexIO operation in doze mode.
- bool enableInDebug
  Enable/disable FlexIO operation in debug mode.
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.
- uint32_t baudRate_Bps
  Baud rate in Bps.

Field Documentation

(1) bool flexio_mculcd_config_t::enable

(2) bool flexio_mculcd_config_t::enableInDoze

(3) bool flexio_mculcd_config_t::enableInDebug

(4) bool flexio_mculcd_config_t::enableFastAccess

(5) uint32_t flexio_mculcd_config_t::baudRate_Bps

19.4.3.3 struct flexio_mculcd_transfer_t

Data Fields

- uint32_t command
  Command to send.
- flexio_mculcd_transfer_mode_t mode
  Transfer mode.
- uint32_t dataAddrOrSameValue
  When sending the same value for many times, this is the value to send.
- size_t dataSize
  How many bytes to transfer.

Field Documentation

(1) uint32_t flexio_mculcd_transfer_t::command

(2) flexio_mculcd_transfer_mode_t flexio_mculcd_transfer_t::mode

(3) uint32_t flexio_mculcd_transfer_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

(4) size_t flexio_mculcd_transfer_t::dataSize

19.4.3.4 struct _flexio_mculcd_handle

typedef for flexio_mculcd_handle_t in advance.

Data Fields

- uint32_t dataAddrOrSameValue
  When sending the same value for many times, this is the value to send.
- size_t dataCount
  Total count to be transferred.
- volatile size_t remainingCount
  Remaining count to transfer.
- volatile uint32_t state
  FlexIO MCULCD internal state.
- flexio_mculcd_transfer_callback_t completionCallback
  FlexIO MCULCD transfer completed callback.
- void ∗ userData
  Callback parameter.

Field Documentation

(1) uint32_t flexio_mculcd_handle_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

(2) size_t flexio_mculcd_handle_t::dataCount

(3) volatile size_t flexio_mculcd_handle_t::remainingCount

(4) volatile uint32_t flexio_mculcd_handle_t::state

(5) flexio_mculcd_transfer_callback_t flexio_mculcd_handle_t::completionCallback

(6) void∗ flexio_mculcd_handle_t::userData

19.4.4 Macro Definition Documentation

19.4.4.1 #define FSL_FLEXIO_MCULCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))

19.4.4.2 #define FLEXIO_MCULCD_WAIT_COMPLETE_TIME 512

Currently there is no method to detect whether the data has been sent out from the shifter, so the driver use a software delay for this. When the data is written to shifter buffer, the driver call the delay function to wait for the data shift out. If this value is too small, then the last few bytes might be lost when writing data using interrupt method or DMA method.

19.4.5 Typedef Documentation

19.4.5.1 typedef void(∗ flexio_mculcd_pin_func_t)(bool set)

19.4.5.2 typedef void(∗ flexio_mculcd_transfer_callback_t)(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_handle_t ∗handle, status_t status, void ∗userData)

When transfer finished, the callback function is called and returns the `status` as kStatus_FLEXIO_MCULCD_Idle.

19.4.6 Enumeration Type Documentation

19.4.6.1 anonymous enum

Enumerator

kStatus_FLEXIO_MCULCD_Idle FlexIO LCD is idle.
kStatus_FLEXIO_MCULCD_Busy FlexIO LCD is busy.
kStatus_FLEXIO_MCULCD_Error FlexIO LCD error occurred.

19.4.6.2 enum flexio_mculcd_pixel_format_t

Enumerator

kFLEXIO_MCULCD_RGB565 RGB565, 16-bit.
kFLEXIO_MCULCD_BGR565 BGR565, 16-bit.
kFLEXIO_MCULCD_RGB888 RGB888, 24-bit.
kFLEXIO_MCULCD_BGR888 BGR888, 24-bit.

19.4.6.3 enum flexio_mculcd_bus_t

Enumerator

kFLEXIO_MCULCD_8080 Using Intel 8080 bus.
kFLEXIO_MCULCD_6800 Using Motorola 6800 bus.

19.4.6.4 enum _flexio_mculcd_interrupt_enable

Enumerator

kFLEXIO_MCULCD_TxEmptyInterruptEnable Transmit buffer empty interrupt enable.
kFLEXIO_MCULCD_RxFullInterruptEnable Receive buffer full interrupt enable.

19.4.6.5 enum _flexio_mculcd_status_flags

Enumerator

kFLEXIO_MCULCD_TxEmptyFlag Transmit buffer empty flag.
kFLEXIO_MCULCD_RxFullFlag Receive buffer full flag.

19.4.6.6 enum _flexio_mculcd_dma_enable

Enumerator

kFLEXIO_MCULCD_TxDmaEnable Tx DMA request source.
kFLEXIO_MCULCD_RxDmaEnable Rx DMA request source.

19.4.6.7 enum flexio_mculcd_transfer_mode_t

Enumerator

kFLEXIO_MCULCD_ReadArray Read data into an array.
kFLEXIO_MCULCD_WriteArray Write data from an array.
kFLEXIO_MCULCD_WriteSameValue Write the same value many times.

19.4.7 Function Documentation

19.4.7.1 status_t FLEXIO_MCULCD_Init (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_config_t * *config*, uint32_t *srcClock_Hz*)

The configuration structure can be filled by the user, or be set with default values by the FLEXIO_MCULCD_GetDefaultConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
config	Pointer to the flexio_mculcd_config_t structure.
srcClock_Hz	FlexIO source clock in Hz.

Return values

kStatus_Success	Initialization success.
kStatus_InvalidArgument	Initialization failed because of invalid argument.

19.4.7.2 void FLEXIO_MCULCD_Deinit (FLEXIO_MCULCD_Type * *base*)

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

19.4.7.3 void FLEXIO_MCULCD_GetDefaultConfig (flexio_mculcd_config_t * *config*)

The default configuration value is:

```
*  config->enable = true;
*  config->enableInDoze = false;
*  config->enableInDebug = true;
*  config->enableFastAccess = true;
*  config->baudRate_Bps = 96000000U;
*
```

Parameters

config	Pointer to the flexio_mculcd_config_t structure.

19.4.7.4 uint32_t FLEXIO_MCULCD_GetStatusFlags (FLEXIO_MCULCD_Type * *base*)

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.

Returns

status flag; OR'ed value or the _flexio_mculcd_status_flags.

Note

Don't use this function with DMA APIs.

19.4.7.5 void FLEXIO_MCULCD_ClearStatusFlags (FLEXIO_MCULCD_Type * *base*, uint32_t *mask*)

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
mask	Status to clear, it is the OR'ed value of _flexio_mculcd_status_flags.

Note

Don't use this function with DMA APIs.

19.4.7.6 void FLEXIO_MCULCD_EnableInterrupts (FLEXIO_MCULCD_Type * *base*, uint32_t *mask*)

This function enables the FlexIO MCULCD interrupt.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
mask	Interrupts to enable, it is the OR'ed value of _flexio_mculcd_interrupt_enable.

19.4.7.7 void FLEXIO_MCULCD_DisableInterrupts (FLEXIO_MCULCD_Type * *base*, uint32_t *mask*)

This function disables the FlexIO MCULCD interrupt.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
mask	Interrupts to disable, it is the OR'ed value of _flexio_mculcd_interrupt_enable.

19.4.7.8 static void FLEXIO_MCULCD_EnableTxDMA (FLEXIO_MCULCD_Type * *base*, bool *enable*) `[inline], [static]`

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.

enable	True means enable DMA, false means disable DMA.

19.4.7.9 static void FLEXIO_MCULCD_EnableRxDMA (FLEXIO_MCULCD_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
enable	True means enable DMA, false means disable DMA.

19.4.7.10 static uint32_t FLEXIO_MCULCD_GetTxDataRegisterAddress (FLEXIO_MCULCD_Type * *base*) [inline], [static]

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.

Returns

FlexIO MCULCD transmit data register address.

19.4.7.11 static uint32_t FLEXIO_MCULCD_GetRxDataRegisterAddress (FLEXIO_MCULCD_Type * *base*) [inline], [static]

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.

Returns

FlexIO MCULCD receive data register address.

19.4.7.12 status_t FLEXIO_MCULCD_SetBaudRate (FLEXIO_MCULCD_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
baudRate_Bps	Desired baud rate.
srcClock_Hz	FLEXIO clock frequency in Hz.

Return values

kStatus_Success	Set successfully.
kStatus_InvalidArgument	Could not set the baud rate.

19.4.7.13 void FLEXIO_MCULCD_SetSingleBeatWriteConfig (FLEXIO_MCULCD_Type * *base*)

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by FLEXIO_MCULCD_ClearSingleBeatWriteConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

Note

This is an internal used function, upper layer should not use.

19.4.7.14 void FLEXIO_MCULCD_ClearSingleBeatWriteConfig (FLEXIO_MCULCD_Type * *base*)

Clear the write configuration set by FLEXIO_MCULCD_SetSingleBeatWriteConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

Note

This is an internal used function, upper layer should not use.

19.4.7.15 void FLEXIO_MCULCD_SetSingleBeatReadConfig (FLEXIO_MCULCD_Type * *base*)

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO_MCULCD-_ClearSingleBeatReadConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

Note

This is an internal used function, upper layer should not use.

19.4.7.16 void FLEXIO_MCULCD_ClearSingleBeatReadConfig (FLEXIO_MCULCD_-Type * *base*)

Clear the read configuration set by FLEXIO_MCULCD_SetSingleBeatReadConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

Note

This is an internal used function, upper layer should not use.

19.4.7.17 void FLEXIO_MCULCD_SetMultiBeatsWriteConfig (FLEXIO_MCULCD_Type * *base*)

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by FLEXIO_MCULC-D_ClearMultBeatsWriteConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

Note

This is an internal used function, upper layer should not use.

19.4.7.18 void FLEXIO_MCULCD_ClearMultiBeatsWriteConfig (FLEXIO_MCULCD_Type * *base*)

Clear the write configuration set by FLEXIO_MCULCD_SetMultBeatsWriteConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

Note

This is an internal used function, upper layer should not use.

19.4.7.19 void FLEXIO_MCULCD_SetMultiBeatsReadConfig (FLEXIO_MCULCD_Type * *base*)

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO_MCULCD-_ClearMultBeatsReadConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

Note

This is an internal used function, upper layer should not use.

19.4.7.20 void FLEXIO_MCULCD_ClearMultiBeatsReadConfig (FLEXIO_MCULCD_-Type * *base*)

Clear the read configuration set by FLEXIO_MCULCD_SetMultBeatsReadConfig.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.

Note

This is an internal used function, upper layer should not use.

19.4.7.21 static void FLEXIO_MCULCD_Enable (FLEXIO_MCULCD_Type * *base*, bool *enable*) `[inline], [static]`

Parameters

base	Pointer to the FLEXIO_MCULCD_Type.
enable	True to enable, false does not have any effect.

19.4.7.22 uint32_t FLEXIO_MCULCD_ReadData (FLEXIO_MCULCD_Type * *base*)

Read data from the RX shift buffer directly, it does no check whether the buffer is empty or not.

If the data bus width is 8-bit:

```
* uint8_t value;
* value = (uint8_t)FLEXIO_MCULCD_ReadData(base);
*
```

If the data bus width is 16-bit:

```
* uint16_t value;
* value = (uint16_t)FLEXIO_MCULCD_ReadData(base);
*
```

Note

This function returns the RX shifter buffer value (32-bit) directly. The return value should be converted according to data bus width.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.

Returns

The data read out.

Note

Don't use this function with DMA APIs.

19.4.7.23 static void FLEXIO_MCULCD_WriteData (FLEXIO_MCULCD_Type * *base*, uint32_t *data*) [inline], [static]

Write data into the TX shift buffer directly, it does no check whether the buffer is full or not.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
data	The data to write.

Note

Don't use this function with DMA APIs.

19.4.7.24 static void FLEXIO_MCULCD_StartTransfer (FLEXIO_MCULCD_Type * *base*) [inline], [static]

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.

19.4.7.25 static void FLEXIO_MCULCD_StopTransfer (FLEXIO_MCULCD_Type * *base*) [inline], [static]

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.

19.4.7.26 void FLEXIO_MCULCD_WaitTransmitComplete (void)

Currently there is no effective method to wait for the data send out from the shiter, so here use a while loop to wait.

Note

This is an internal used function.

19.4.7.27 void FLEXIO_MCULCD_WriteCommandBlocking (FLEXIO_MCULCD_Type * *base*, uint32_t *command*)

This function sends the command and returns when the command has been sent out.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
command	The command to send.

19.4.7.28 void FLEXIO_MCULCD_WriteDataArrayBlocking (FLEXIO_MCULCD_Type * *base*, const void * *data*, size_t *size*)

This function sends the data array and returns when the data sent out.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
data	The data array to send.
size	How many bytes to write.

19.4.7.29 void FLEXIO_MCULCD_ReadDataArrayBlocking (FLEXIO_MCULCD_Type * *base*, void * *data*, size_t *size*)

This function reads the data into array and returns when the data read finished.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
data	The array to save the data.
size	How many bytes to read.

19.4.7.30 void FLEXIO_MCULCD_WriteSameValueBlocking (FLEXIO_MCULCD_Type * *base*, uint32_t *sameValue*, size_t *size*)

This function sends the same value many times. It could be used to clear the LCD screen. If the data bus width is 8, this function will send LSB 8 bits of `sameValue` for `size` times. If the data bus is 16, this function will send LSB 16 bits of `sameValue` for `size` / 2 times.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
sameValue	The same value to send.
size	How many bytes to send.

19.4.7.31 void FLEXIO_MCULCD_TransferBlocking (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_transfer_t * *xfer*)

Note

The API does not return until the transfer finished.

Parameters

base	pointer to FLEXIO_MCULCD_Type structure.
xfer	pointer to flexio_mculcd_transfer_t structure.

19.4.7.32 status_t FLEXIO_MCULCD_TransferCreateHandle (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, flexio_mculcd_transfer_callback_t *callback*, void * *userData*)

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
handle	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.
callback	The callback function.
userData	The parameter of the callback function.

Return values

kStatus_Success	Successfully create the handle.
kStatus_OutOfRange	The FlexIO type/handle/ISR table out of range.

19.4.7.33 status_t FLEXIO_MCULCD_TransferNonBlocking (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer*)

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
handle	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.
xfer	FlexIO MCULCD transfer structure. See flexio_mculcd_transfer_t.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_FLEXIO_MCULCD_Busy	MCULCD is busy with another transfer.

19.4.7.34 void FLEXIO_MCULCD_TransferAbort (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*)

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
handle	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.

19.4.7.35 status_t FLEXIO_MCULCD_TransferGetCount (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, size_t * *count*)

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
handle	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.
count	How many bytes transferred so far by the non-blocking transaction.

Return values

kStatus_Success	Get the transferred count Successfully.
kStatus_NoTransferInProgress	No transfer in process.

19.4.7.36 void FLEXIO_MCULCD_TransferHandleIRQ (void * *base*, void * *handle*)

Parameters

base	Pointer to the FLEXIO_MCULCD_Type structure.
handle	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.

19.4.8 FlexIO DMA MCU Interface LCD Driver

19.4.8.1 Overview

Data Structures

- struct flexio_mculcd_dma_handle_t
  FlexIO MCULCD DMA transfer handle, users should not touch the content of the handle. More...

Macros

- #define FSL_FLEXIO_MCULCD_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  FlexIO MCULCD DMA driver version 2.0.4.

Typedefs

- typedef void(∗ flexio_mculcd_dma_transfer_callback_t)(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, status_t status, void ∗userData)
  FlexIO MCULCD master callback for transfer complete.

DMA Transactional

- status_t FLEXIO_MCULCD_TransferCreateHandleDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, flexio_mculcd_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  Initializes the FLEXO MCULCD master DMA handle.
- status_t FLEXIO_MCULCD_TransferDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, flexio_mculcd_transfer_t ∗xfer)
  Performs a non-blocking FlexIO MCULCD transfer using DMA.
- void FLEXIO_MCULCD_TransferAbortDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle)
  Aborts a FlexIO MCULCD transfer using DMA.
- status_t FLEXIO_MCULCD_TransferGetCountDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, size_t ∗count)
  Gets the remaining bytes for FlexIO MCULCD DMA transfer.

19.4.8.2 Data Structure Documentation

19.4.8.2.1 struct _flexio_mculcd_dma_handle

typedef for flexio_mculcd_dma_handle_t in advance.

Data Fields

- FLEXIO_MCULCD_Type ∗ base

Pointer to the FLEXIO_MCULCD_Type.

- uint8_t txShifterNum
  Number of shifters used for TX.
- uint8_t rxShifterNum
  Number of shifters used for RX.
- dma_trigger_burst_t txTriggerBurst
  Trigger burst setting for TX.
- dma_trigger_burst_t rxTriggerBurst
  Trigger burst setting for RX.
- uint32_t burstBytes
  DMA transfer bytes per burst.
- uint32_t dataAddrOrSameValue
  When sending the same value for many times, this is the value to send.
- size_t dataCount
  Total count to be transferred.
- volatile size_t remainingCount
  Remaining count still not transfered.
- volatile uint32_t state
  FlexIO MCULCD driver internal state.
- dma_handle_t * txDmaHandle
  DMA handle for MCULCD TX.
- dma_handle_t * rxDmaHandle
  DMA handle for MCULCD RX.
- flexio_mculcd_dma_transfer_callback_t completionCallback
  Callback for MCULCD DMA transfer.
- void * userData
  User Data for MCULCD DMA callback.

Field Documentation

(1) FLEXIO_MCULCD_Type* flexio_mculcd_dma_handle_t::base

(2) uint8_t flexio_mculcd_dma_handle_t::txShifterNum

(3) uint8_t flexio_mculcd_dma_handle_t::rxShifterNum

(4) dma_trigger_burst_t flexio_mculcd_dma_handle_t::txTriggerBurst

(5) dma_trigger_burst_t flexio_mculcd_dma_handle_t::rxTriggerBurst

(6) uint32_t flexio_mculcd_dma_handle_t::burstBytes

(7) uint32_t flexio_mculcd_dma_handle_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

(8) size_t flexio_mculcd_dma_handle_t::dataCount

(9) volatile size_t flexio_mculcd_dma_handle_t::remainingCount

(10) volatile uint32_t flexio_mculcd_dma_handle_t::state

19.4.8.3 Macro Definition Documentation

19.4.8.3.1 #define FSL_FLEXIO_MCULCD_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

19.4.8.4 Typedef Documentation

19.4.8.4.1 typedef void(∗ flexio_mculcd_dma_transfer_callback_t)(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_dma_handle_t ∗handle, status_t status, void ∗userData)

When transfer finished, the callback function is called and returns the `status` as kStatus_FLEXIO_MCULCD_Idle.

19.4.8.5 Function Documentation

19.4.8.5.1 status_t FLEXIO_MCULCD_TransferCreateHandleDMA (FLEXIO_MCULCD_Type ∗ *base,* flexio_mculcd_dma_handle_t ∗ *handle,* flexio_mculcd_dma_transfer_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *txDmaHandle,* dma_handle_t ∗ *rxDmaHandle*)

This function initializes the FLEXO MCULCD master DMA handle which can be used for other FLEXO MCULCD transactional APIs. For a specified FLEXO MCULCD instance, call this API once to get the initialized handle.

Parameters

base	Pointer to FLEXIO_MCULCD_Type structure.
handle	Pointer to flexio_mculcd_dma_handle_t structure to store the transfer state.
callback	MCULCD transfer complete callback, NULL means no callback.
userData	callback function parameter.
txDmaHandle	User requested DMA handle for FlexIO MCULCD DMA TX, the DMA request source of this handle should be the first of TX shifters.
rxDmaHandle	User requested DMA handle for FlexIO MCULCD DMA RX, the DMA request source of this handle should be the last of RX shifters.

Return values

kStatus_Success	Successfully create the handle.

19.4.8.5.2 status_t FLEXIO_MCULCD_TransferDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_dma_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer*)

This function returns immediately after transfer initiates. To check whether the transfer is completed, user could:

1. Use the transfer completed callback;
2. Polling function FLEXIO_MCULCD_GetTransferCountDMA

Parameters

base	pointer to FLEXIO_MCULCD_Type structure.
handle	pointer to flexio_mculcd_dma_handle_t structure to store the transfer state.
xfer	Pointer to FlexIO MCULCD transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_FLEXIO_MCUL-CD_Busy	FlexIO MCULCD is not idle, it is running another transfer.

19.4.8.5.3 void FLEXIO_MCULCD_TransferAbortDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_dma_handle_t * *handle*)

Parameters

base	pointer to FLEXIO_MCULCD_Type structure.
handle	FlexIO MCULCD DMA handle pointer.

19.4.8.5.4 status_t FLEXIO_MCULCD_TransferGetCountDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_dma_handle_t * *handle*, size_t * *count*)

Parameters

base	pointer to FLEXIO_MCULCD_Type structure.

handle	FlexIO MCULCD DMA handle pointer.
count	Number of count transferred so far by the DMA transaction.

Return values

kStatus_Success	Get the transferred count Successfully.
kStatus_NoTransferIn-Progress	No transfer in process.

19.4.9 FlexIO SMARTDMA MCU Interface LCD Driver

19.4.9.1 Overview

Data Structures

- struct flexio_mculcd_smartdma_handle_t
  FlexIO MCULCD SMARTDMA transfer handle, users should not touch the content of the handle. More...
- struct flexio_mculcd_smartdma_config_t
  FlexIO MCULCD SMARTDMA configuration. More...

Macros

- #define FSL_FLEXIO_MCULCD_SMARTDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  FlexIO MCULCD SMARTDMA driver version.
- #define FLEXIO_MCULCD_SMARTDMA_TX_LEN_ALIGN 64U
  SMARTDMA transfer size should be multiple of 64 bytes.
- #define FLEXIO_MCULCD_SMARTDMA_TX_ADDR_ALIGN 4U
  SMARTDMA transfer memory address should be 4 byte aligned.

Typedefs

- typedef void(∗ flexio_mculcd_smartdma_transfer_callback_t)(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_smartdma_handle_t ∗handle, status_t status, void ∗userData)
  FlexIO MCULCD master callback for transfer complete.

SMARTDMA Transactional

- status_t FLEXIO_MCULCD_TransferCreateHandleSMARTDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_smartdma_handle_t ∗handle, const flexio_mculcd_smartdma_config_t ∗config, flexio_mculcd_smartdma_transfer_callback_t callback, void ∗userData)
  Initializes the FLEXO MCULCD master SMARTDMA handle.
- status_t FLEXIO_MCULCD_TransferSMARTDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_smartdma_handle_t ∗handle, flexio_mculcd_transfer_t ∗xfer)
  Performs a non-blocking FlexIO MCULCD transfer using SMARTDMA.
- void FLEXIO_MCULCD_TransferAbortSMARTDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_smartdma_handle_t ∗handle)
  Aborts a FlexIO MCULCD transfer using SMARTDMA.
- status_t FLEXIO_MCULCD_TransferGetCountSMARTDMA (FLEXIO_MCULCD_Type ∗base, flexio_mculcd_smartdma_handle_t ∗handle, size_t ∗count)
  Gets the remaining bytes for FlexIO MCULCD SMARTDMA transfer.

19.4.9.2 Data Structure Documentation

19.4.9.2.1 struct _flexio_mculcd_smartdma_handle

typedef for flexio_mculcd_smartdma_handle_t in advance.

Data Fields

- FLEXIO_MCULCD_Type * base
  Pointer to the FLEXIO_MCULCD_Type.
- size_t dataCount
  Total count to be transferred.
- uint32_t dataAddrOrSameValue
  When sending the same value for many times, this is the value to send.
- size_t dataCountUsingEzh
  Data transfered using SMARTDMA.
- volatile size_t remainingCount
  Remaining count to transfer.
- volatile uint32_t state
  FlexIO MCULCD driver internal state.
- uint8_t smartdmaApi
  The SMARTDMA API used during transfer.
- bool needColorConvert
  Need color convert or not.
- uint8_t blockingXferBuffer [FLEXIO_MCULCD_SMARTDMA_TX_LEN_ALIGN *3/2]
  Used for blocking method color space convet.
- flexio_mculcd_smartdma_transfer_callback_t completionCallback
  Callback for MCULCD SMARTDMA transfer.
- void * userData
  User Data for MCULCD SMARTDMA callback.
- smartdma_flexio_mculcd_param_t smartdmaParam
  SMARTDMA function parameters.
- uint32_t smartdmaStack [1]
  SMARTDMA function stack.

Field Documentation

(1) FLEXIO_MCULCD_Type* flexio_mculcd_smartdma_handle_t::base

(2) size_t flexio_mculcd_smartdma_handle_t::dataCount

(3) uint32_t flexio_mculcd_smartdma_handle_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

(4) size_t flexio_mculcd_smartdma_handle_t::dataCountUsingEzh

(5) volatile size_t flexio_mculcd_smartdma_handle_t::remainingCount

(6) volatile uint32_t flexio_mculcd_smartdma_handle_t::state

(7) uint8_t flexio_mculcd_smartdma_handle_t::smartdmaApi

(8) bool flexio_mculcd_smartdma_handle_t::needColorConvert

(9) uint8_t flexio_mculcd_smartdma_handle_t::blockingXferBuffer[FLEXIO_MCULCD_SMARTDMA_TX_LEN_ALIGN ∗3/2]

(10) smartdma_flexio_mculcd_param_t flexio_mculcd_smartdma_handle_t::smartdmaParam

(11) uint32_t flexio_mculcd_smartdma_handle_t::smartdmaStack[1]

19.4.9.2.2 struct flexio_mculcd_smartdma_config_t

Data Fields

- flexio_mculcd_pixel_format_t inputPixelFormat
  The pixel format in the frame buffer.
- flexio_mculcd_pixel_format_t outputPixelFormat
  The pixel format on the 8080/68k bus.

Field Documentation

(1) flexio_mculcd_pixel_format_t flexio_mculcd_smartdma_config_t::inputPixelFormat

(2) flexio_mculcd_pixel_format_t flexio_mculcd_smartdma_config_t::outputPixelFormat

19.4.9.3 Macro Definition Documentation

19.4.9.3.1 #define FSL_FLEXIO_MCULCD_SMARTDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

19.4.9.3.2 #define FLEXIO_MCULCD_SMARTDMA_TX_LEN_ALIGN 64U

19.4.9.3.3 #define FLEXIO_MCULCD_SMARTDMA_TX_ADDR_ALIGN 4U

19.4.9.4 Typedef Documentation

19.4.9.4.1 typedef void(∗ flexio_mculcd_smartdma_transfer_callback_t)(FLEXIO_MCULCD_Type ∗base, flexio_mculcd_smartdma_handle_t ∗handle, status_t status, void ∗userData)

When transfer finished, the callback function is called and returns the `status` as kStatus_FLEXIO_MCULCD_Idle.

19.4.9.5 Function Documentation

19.4.9.5.1 status_t FLEXIO_MCULCD_TransferCreateHandleSMARTDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_smartdma_handle_t * *handle*, const flexio_mculcd_smartdma_config_t * *config*, flexio_mculcd_smartdma_transfer_callback_t *callback*, void * *userData*)

This function initializes the FLEXO MCULCD master SMARTDMA handle which can be used for other FLEXO MCULCD transactional APIs. For a specified FLEXO MCULCD instance, call this API once to get the initialized handle.

Parameters

base	Pointer to FLEXIO_MCULCD_Type structure.
handle	Pointer to flexio_mculcd_smartdma_handle_t structure to store the transfer state.
config	Pointer to the configuration.
callback	MCULCD transfer complete callback, NULL means no callback.
userData	callback function parameter.

Return values

kStatus_Success	Successfully create the handle.

19.4.9.5.2 status_t FLEXIO_MCULCD_TransferSMARTDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_smartdma_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer*)

This function returns immediately after transfer initiates. Use the callback function to check whether the transfer is completed.

Parameters

base	pointer to FLEXIO_MCULCD_Type structure.
handle	pointer to flexio_mculcd_smartdma_handle_t structure to store the transfer state.
xfer	Pointer to FlexIO MCULCD transfer structure.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.

kStatus_FLEXIO_MCULCD_Busy	FlexIO MCULCD is not idle, it is running another transfer.

19.4.9.5.3 void FLEXIO_MCULCD_TransferAbortSMARTDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_smartdma_handle_t * *handle*)

Parameters

base	pointer to FLEXIO_MCULCD_Type structure.
handle	FlexIO MCULCD SMARTDMA handle pointer.

19.4.9.5.4 status_t FLEXIO_MCULCD_TransferGetCountSMARTDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_smartdma_handle_t * *handle*, size_t * *count*)

Parameters

base	pointer to FLEXIO_MCULCD_Type structure.
handle	FlexIO MCULCD SMARTDMA handle pointer.
count	Number of count transferred so far by the SMARTDMA transaction.

Return values

kStatus_Success	Get the transferred count Successfully.
kStatus_NoTransferInProgress	No transfer in process.

19.5 FlexIO SPI Driver

19.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for an SPI function using the Flexible I/O module of MCUXpresso SDK devices.

FlexIO SPI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for FlexIO SPI initialization/configuration/operati for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the FLEXIO_SPI_Type ∗base as the first parameter. FlexIO SPI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the flexio_spi_master_handle_t/flexio_spi_slave_handle_t as the second parameter. Initialize the handle by calling the FLEXIO_SPI_MasterTransferCreateHandle() or FLEXIO_SPI_SlaveTransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions FLEXIO_SPI_MasterTransferNonBlocking()/FLEXIO_SPI_SlaveTransferNonBlocking() set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the kStatus_FLEXIO_SPI_Idle status.

Note that the FlexIO SPI slave driver only supports discontinuous PCS access, which is a limitation. The FlexIO SPI slave driver can support continuous PCS, but the slave cannot adapt discontinuous and continuous PCS automatically. Users can change the timer disable mode in FLEXIO_SPI_SlaveInit manually, from kFLEXIO_TimerDisableOnTimerCompare to kFLEXIO_TimerDisableNever to enable a discontinuous PCS access. Only CPHA = 0 is supported.

19.5.2 Typical use case

19.5.2.1 FlexIO SPI send/receive using an interrupt method

```
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];

void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle
      , status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_SPI_Idle == status)
    {
        txFinished = true;
    }
```

```
}

void main(void)
{
    //...
    flexio_spi_transfer_t xfer = {0};
    flexio_spi_master_config_t userConfig;

    FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);
    userConfig.baudRate_Bps = 500000U;

    spiDev.flexioBase = BOARD_FLEXIO_BASE;
    spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
    spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
    spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
    spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
    spiDev.shifterIndex[0] = 0U;
    spiDev.shifterIndex[1] = 1U;
    spiDev.timerIndex[0] = 0U;
    spiDev.timerIndex[1] = 1U;

    FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

    xfer.txData = srcBuff;
    xfer.rxData = destBuff;
    xfer.dataSize = BUFFER_SIZE;
    xfer.flags = kFLEXIO_SPI_8bitMsb;
    FLEXIO_SPI_MasterTransferCreateHandle(&spiDev, &g_spiHandle,
      FLEXIO_SPI_MasterUserCallback, NULL);
    FLEXIO_SPI_MasterTransferNonBlocking(&spiDev, &g_spiHandle, &xfer);

    // Send finished.
    while (!txFinished)
    {
    }

        // ...
}
```

19.5.2.2 FlexIO_SPI Send/Receive in DMA way

```
    dma_handle_t g_spiTxDmaHandle;
    dma_handle_t g_spiRxDmaHandle;
    flexio_spi_master_handle_t g_spiHandle;
    FLEXIO_SPI_Type spiDev;
    volatile bool txFinished;
    static uint8_t srcBuff[BUFFER_SIZE];
    static uint8_t destBuff[BUFFER_SIZE];
    void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_dma_handle_t
       *handle, status_t status, void *userData)
    {
        userData = userData;

        if (kStatus_FLEXIO_SPI_Idle == status)
        {
            txFinished = true;
        }
    }

    void main(void)
    {
        flexio_spi_transfer_t xfer = {0};
        flexio_spi_master_config_t userConfig;

        FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);
```

```
        userConfig.baudRate_Bps = 500000U;

        spiDev.flexioBase = BOARD_FLEXIO_BASE;
        spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
        spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
        spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
        spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
        spiDev.shifterIndex[0] = 0U;
        spiDev.shifterIndex[1] = 1U;
        spiDev.timerIndex[0] = 0U;
        spiDev.timerIndex[1] = 1U;

        /* Init DMAMUX. */
        DMAMUX_Init(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR)

        /* Init the DMA/EDMA module */
#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
        DMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR);
        DMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
      FLEXIO_SPI_TX_DMA_CHANNEL);
        DMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
      FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
        edma_config_t edmaConfig;

        EDMA_GetDefaultConfig(&edmaConfig);
        EDMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, &edmaConfig);
        EDMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL);
        EDMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

        dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
      shifterIndex[0]);
        dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
      shifterIndex[1]);

        /* Requests  DMA channels for transmit and receive. */
        DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL, (
      dma_request_source_t)dma_request_source_tx);
        DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL, (
      dma_request_source_t)dma_request_source_rx);
        DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL);
        DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL);

        FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

        /* Initializes the buffer. */
        for (i = 0; i < BUFFER_SIZE; i++)
        {
            srcBuff[i] = i;
        }

        /* Sends to the slave. */
        xfer.txData = srcBuff;
        xfer.rxData = destBuff;
        xfer.dataSize = BUFFER_SIZE;
        xfer.flags = kFLEXIO_SPI_8bitMsb;
        FLEXIO_SPI_MasterTransferCreateHandleDMA(&spiDev, &g_spiHandle, FLEXIO_SPI_MasterUserCallback, NULL
      , &g_spiTxDmaHandle, &g_spiRxDmaHandle);
        FLEXIO_SPI_MasterTransferDMA(&spiDev, &g_spiHandle, &xfer);

        // Send finished.
        while (!txFinished)
        {
        }
```

```
    // ...
}
```

Data Structures

- struct FLEXIO_SPI_Type
  Define FlexIO SPI access structure typedef. More...
- struct flexio_spi_master_config_t
  Define FlexIO SPI master configuration structure. More...
- struct flexio_spi_slave_config_t
  Define FlexIO SPI slave configuration structure. More...
- struct flexio_spi_transfer_t
  Define FlexIO SPI transfer structure. More...
- struct flexio_spi_master_handle_t
  Define FlexIO SPI handle structure. More...

Macros

- #define FLEXIO_SPI_DUMMYDATA (0xFFFFU)
  FlexIO SPI dummy transfer data, the data is sent while txData is NULL.
- #define SPI_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Retry times for waiting flag.

Typedefs

- typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t
  Slave handle is the same with master handle.
- typedef void(∗ flexio_spi_master_transfer_callback_t)(FLEXIO_SPI_Type ∗base, flexio_spi_master_handle_t ∗handle, status_t status, void ∗userData)
  FlexIO SPI master callback for finished transmit.
- typedef void(∗ flexio_spi_slave_transfer_callback_t)(FLEXIO_SPI_Type ∗base, flexio_spi_slave_handle_t ∗handle, status_t status, void ∗userData)
  FlexIO SPI slave callback for finished transmit.

Enumerations

- enum {
  kStatus_FLEXIO_SPI_Busy = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 1),
  kStatus_FLEXIO_SPI_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 2),
  kStatus_FLEXIO_SPI_Error = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 3),
  kStatus_FLEXIO_SPI_Timeout }
  Error codes for the FlexIO SPI driver.

- enum flexio_spi_clock_phase_t {
  kFLEXIO_SPI_ClockPhaseFirstEdge = 0x0U,
  kFLEXIO_SPI_ClockPhaseSecondEdge = 0x1U }
  FlexIO SPI clock phase configuration.
- enum flexio_spi_shift_direction_t {
  kFLEXIO_SPI_MsbFirst = 0,
  kFLEXIO_SPI_LsbFirst = 1 }
  FlexIO SPI data shifter direction options.
- enum flexio_spi_data_bitcount_mode_t {
  kFLEXIO_SPI_8BitMode = 0x08U,
  kFLEXIO_SPI_16BitMode = 0x10U }
  FlexIO SPI data length mode options.
- enum _flexio_spi_interrupt_enable {
  kFLEXIO_SPI_TxEmptyInterruptEnable = 0x1U,
  kFLEXIO_SPI_RxFullInterruptEnable = 0x2U }
  Define FlexIO SPI interrupt mask.
- enum _flexio_spi_status_flags {
  kFLEXIO_SPI_TxBufferEmptyFlag = 0x1U,
  kFLEXIO_SPI_RxBufferFullFlag = 0x2U }
  Define FlexIO SPI status mask.
- enum _flexio_spi_dma_enable {
  kFLEXIO_SPI_TxDmaEnable = 0x1U,
  kFLEXIO_SPI_RxDmaEnable = 0x2U,
  kFLEXIO_SPI_DmaAllEnable = 0x3U }
  Define FlexIO SPI DMA mask.
- enum _flexio_spi_transfer_flags {
  kFLEXIO_SPI_8bitMsb = 0x1U,
  kFLEXIO_SPI_8bitLsb = 0x2U,
  kFLEXIO_SPI_16bitMsb = 0x9U,
  kFLEXIO_SPI_16bitLsb = 0xaU }
  Define FlexIO SPI transfer flags.

Driver version

- #define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
  FlexIO SPI driver version 2.2.1.

FlexIO SPI Configuration

- void FLEXIO_SPI_MasterInit (FLEXIO_SPI_Type *base, flexio_spi_master_config_t *masterConfig, uint32_t srcClock_Hz)
  Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration.
- void FLEXIO_SPI_MasterDeinit (FLEXIO_SPI_Type *base)
  Resets the FlexIO SPI timer and shifter config.
- void FLEXIO_SPI_MasterGetDefaultConfig (flexio_spi_master_config_t *masterConfig)

Gets the default configuration to configure the FlexIO SPI master.

- void FLEXIO_SPI_SlaveInit (FLEXIO_SPI_Type *base, flexio_spi_slave_config_t *slaveConfig)
  Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration.
- void FLEXIO_SPI_SlaveDeinit (FLEXIO_SPI_Type *base)
  Gates the FlexIO clock.
- void FLEXIO_SPI_SlaveGetDefaultConfig (flexio_spi_slave_config_t *slaveConfig)
  Gets the default configuration to configure the FlexIO SPI slave.

Status

- uint32_t FLEXIO_SPI_GetStatusFlags (FLEXIO_SPI_Type *base)
  Gets FlexIO SPI status flags.
- void FLEXIO_SPI_ClearStatusFlags (FLEXIO_SPI_Type *base, uint32_t mask)
  Clears FlexIO SPI status flags.

Interrupts

- void FLEXIO_SPI_EnableInterrupts (FLEXIO_SPI_Type *base, uint32_t mask)
  Enables the FlexIO SPI interrupt.
- void FLEXIO_SPI_DisableInterrupts (FLEXIO_SPI_Type *base, uint32_t mask)
  Disables the FlexIO SPI interrupt.

DMA Control

- void FLEXIO_SPI_EnableDMA (FLEXIO_SPI_Type *base, uint32_t mask, bool enable)
  Enables/disables the FlexIO SPI transmit DMA.
- static uint32_t FLEXIO_SPI_GetTxDataRegisterAddress (FLEXIO_SPI_Type *base, flexio_spi_-shift_direction_t direction)
  Gets the FlexIO SPI transmit data register address for MSB first transfer.
- static uint32_t FLEXIO_SPI_GetRxDataRegisterAddress (FLEXIO_SPI_Type *base, flexio_spi_-shift_direction_t direction)
  Gets the FlexIO SPI receive data register address for the MSB first transfer.

Bus Operations

- static void FLEXIO_SPI_Enable (FLEXIO_SPI_Type *base, bool enable)
  Enables/disables the FlexIO SPI module operation.
- void FLEXIO_SPI_MasterSetBaudRate (FLEXIO_SPI_Type *base, uint32_t baudRate_Bps, uint32_t srcClockHz)
  Sets baud rate for the FlexIO SPI transfer, which is only used for the master.
- static void FLEXIO_SPI_WriteData (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction, uint16_t data)
  Writes one byte of data, which is sent using the MSB method.

- static uint16_t FLEXIO_SPI_ReadData (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction)
  Reads 8 bit/16 bit data.
- status_t FLEXIO_SPI_WriteBlocking (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction, const uint8_t *buffer, size_t size)
  Sends a buffer of data bytes.
- status_t FLEXIO_SPI_ReadBlocking (FLEXIO_SPI_Type *base, flexio_spi_shift_direction_t direction, uint8_t *buffer, size_t size)
  Receives a buffer of bytes.
- status_t FLEXIO_SPI_MasterTransferBlocking (FLEXIO_SPI_Type *base, flexio_spi_transfer_t *xfer)
  Receives a buffer of bytes.

Transactional

- status_t FLEXIO_SPI_MasterTransferCreateHandle (FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle, flexio_spi_master_transfer_callback_t callback, void *userData)
  Initializes the FlexIO SPI Master handle, which is used in transactional functions.
- status_t FLEXIO_SPI_MasterTransferNonBlocking (FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle, flexio_spi_transfer_t *xfer)
  Master transfer data using IRQ.
- void FLEXIO_SPI_MasterTransferAbort (FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle)
  Aborts the master data transfer, which used IRQ.
- status_t FLEXIO_SPI_MasterTransferGetCount (FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle, size_t *count)
  Gets the data transfer status which used IRQ.
- void FLEXIO_SPI_MasterTransferHandleIRQ (void *spiType, void *spiHandle)
  FlexIO SPI master IRQ handler function.
- status_t FLEXIO_SPI_SlaveTransferCreateHandle (FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle, flexio_spi_slave_transfer_callback_t callback, void *userData)
  Initializes the FlexIO SPI Slave handle, which is used in transactional functions.
- status_t FLEXIO_SPI_SlaveTransferNonBlocking (FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle, flexio_spi_transfer_t *xfer)
  Slave transfer data using IRQ.
- static void FLEXIO_SPI_SlaveTransferAbort (FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle)
  Aborts the slave data transfer which used IRQ, share same API with master.
- static status_t FLEXIO_SPI_SlaveTransferGetCount (FLEXIO_SPI_Type *base, flexio_spi_slave_handle_t *handle, size_t *count)
  Gets the data transfer status which used IRQ, share same API with master.
- void FLEXIO_SPI_SlaveTransferHandleIRQ (void *spiType, void *spiHandle)
  FlexIO SPI slave IRQ handler function.

19.5.3 Data Structure Documentation

19.5.3.1 struct FLEXIO_SPI_Type

Data Fields

- FLEXIO_Type * flexioBase
  FlexIO base pointer.
- uint8_t SDOPinIndex
  Pin select for data output.
- uint8_t SDIPinIndex
  Pin select for data input.
- uint8_t SCKPinIndex
  Pin select for clock.
- uint8_t CSnPinIndex
  Pin select for enable.
- uint8_t shifterIndex [2]
  Shifter index used in FlexIO SPI.
- uint8_t timerIndex [2]
  Timer index used in FlexIO SPI.

Field Documentation

(1) FLEXIO_Type* FLEXIO_SPI_Type::flexioBase

(2) uint8_t FLEXIO_SPI_Type::SDOPinIndex

(3) uint8_t FLEXIO_SPI_Type::SDIPinIndex

(4) uint8_t FLEXIO_SPI_Type::SCKPinIndex

(5) uint8_t FLEXIO_SPI_Type::CSnPinIndex

(6) uint8_t FLEXIO_SPI_Type::shifterIndex[2]

(7) uint8_t FLEXIO_SPI_Type::timerIndex[2]

19.5.3.2 struct flexio_spi_master_config_t

Data Fields

- bool enableMaster
  Enable/disable FlexIO SPI master after configuration.
- bool enableInDoze
  Enable/disable FlexIO operation in doze mode.
- bool enableInDebug
  Enable/disable FlexIO operation in debug mode.
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.
- uint32_t baudRate_Bps
  Baud rate in Bps.
- flexio_spi_clock_phase_t phase

Clock phase.
- flexio_spi_data_bitcount_mode_t dataMode
  8bit or 16bit mode.

Field Documentation

(1) bool flexio_spi_master_config_t::enableMaster

(2) bool flexio_spi_master_config_t::enableInDoze

(3) bool flexio_spi_master_config_t::enableInDebug

(4) bool flexio_spi_master_config_t::enableFastAccess

(5) uint32_t flexio_spi_master_config_t::baudRate_Bps

(6) flexio_spi_clock_phase_t flexio_spi_master_config_t::phase

(7) flexio_spi_data_bitcount_mode_t flexio_spi_master_config_t::dataMode

19.5.3.3 struct flexio_spi_slave_config_t

Data Fields

- bool enableSlave
  Enable/disable FlexIO SPI slave after configuration.
- bool enableInDoze
  Enable/disable FlexIO operation in doze mode.
- bool enableInDebug
  Enable/disable FlexIO operation in debug mode.
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.
- flexio_spi_clock_phase_t phase
  Clock phase.
- flexio_spi_data_bitcount_mode_t dataMode
  8bit or 16bit mode.

Field Documentation

(1) bool flexio_spi_slave_config_t::enableSlave

(2) bool flexio_spi_slave_config_t::enableInDoze

(3) bool flexio_spi_slave_config_t::enableInDebug

(4) bool flexio_spi_slave_config_t::enableFastAccess

(5) flexio_spi_clock_phase_t flexio_spi_slave_config_t::phase

(6) flexio_spi_data_bitcount_mode_t flexio_spi_slave_config_t::dataMode

19.5.3.4 struct flexio_spi_transfer_t

Data Fields

- uint8_t * txData
  Send buffer.
- uint8_t * rxData
  Receive buffer.
- size_t dataSize
  Transfer bytes.
- uint8_t flags
  FlexIO SPI control flag, MSB first or LSB first.

Field Documentation

(1) uint8_t* flexio_spi_transfer_t::txData

(2) uint8_t* flexio_spi_transfer_t::rxData

(3) size_t flexio_spi_transfer_t::dataSize

(4) uint8_t flexio_spi_transfer_t::flags

19.5.3.5 struct _flexio_spi_master_handle

typedef for flexio_spi_master_handle_t in advance.

Data Fields

- uint8_t * txData
  Transfer buffer.
- uint8_t * rxData
  Receive buffer.
- size_t transferSize
  Total bytes to be transferred.
- volatile size_t txRemainingBytes
  Send data remaining in bytes.
- volatile size_t rxRemainingBytes
  Receive data remaining in bytes.
- volatile uint32_t state
  FlexIO SPI internal state.
- uint8_t bytePerFrame
  SPI mode, 2bytes or 1byte in a frame.
- flexio_spi_shift_direction_t direction
  Shift direction.
- flexio_spi_master_transfer_callback_t callback
  FlexIO SPI callback.
- void * userData
  Callback parameter.

Field Documentation

(1) uint8_t* flexio_spi_master_handle_t::txData

(2) uint8_t* flexio_spi_master_handle_t::rxData

(3) size_t flexio_spi_master_handle_t::transferSize

(4) volatile size_t flexio_spi_master_handle_t::txRemainingBytes

(5) volatile size_t flexio_spi_master_handle_t::rxRemainingBytes

(6) volatile uint32_t flexio_spi_master_handle_t::state

(7) flexio_spi_shift_direction_t flexio_spi_master_handle_t::direction

(8) flexio_spi_master_transfer_callback_t flexio_spi_master_handle_t::callback

(9) void* flexio_spi_master_handle_t::userData

19.5.4 Macro Definition Documentation

19.5.4.1 #define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

19.5.4.2 #define FLEXIO_SPI_DUMMYDATA (0xFFFFU)

19.5.4.3 #define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */

19.5.5 Typedef Documentation

19.5.5.1 typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t

19.5.6 Enumeration Type Documentation

19.5.6.1 anonymous enum

Enumerator

kStatus_FLEXIO_SPI_Busy FlexIO SPI is busy.
kStatus_FLEXIO_SPI_Idle SPI is idle.
kStatus_FLEXIO_SPI_Error FlexIO SPI error.
kStatus_FLEXIO_SPI_Timeout FlexIO SPI timeout polling status flags.

19.5.6.2 enum flexio_spi_clock_phase_t

Enumerator

kFLEXIO_SPI_ClockPhaseFirstEdge First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

kFLEXIO_SPI_ClockPhaseSecondEdge First edge on SPSCK occurs at the start of the first cycle of a data transfer.

19.5.6.3 enum flexio_spi_shift_direction_t

Enumerator

kFLEXIO_SPI_MsbFirst Data transfers start with most significant bit.

kFLEXIO_SPI_LsbFirst Data transfers start with least significant bit.

19.5.6.4 enum flexio_spi_data_bitcount_mode_t

Enumerator

kFLEXIO_SPI_8BitMode 8-bit data transmission mode.

kFLEXIO_SPI_16BitMode 16-bit data transmission mode.

19.5.6.5 enum _flexio_spi_interrupt_enable

Enumerator

kFLEXIO_SPI_TxEmptyInterruptEnable Transmit buffer empty interrupt enable.

kFLEXIO_SPI_RxFullInterruptEnable Receive buffer full interrupt enable.

19.5.6.6 enum _flexio_spi_status_flags

Enumerator

kFLEXIO_SPI_TxBufferEmptyFlag Transmit buffer empty flag.

kFLEXIO_SPI_RxBufferFullFlag Receive buffer full flag.

19.5.6.7 enum _flexio_spi_dma_enable

Enumerator

kFLEXIO_SPI_TxDmaEnable Tx DMA request source.

kFLEXIO_SPI_RxDmaEnable Rx DMA request source.

kFLEXIO_SPI_DmaAllEnable All DMA request source.

19.5.6.8 enum _flexio_spi_transfer_flags

Enumerator

kFLEXIO_SPI_8bitMsb FlexIO SPI 8-bit MSB first.
kFLEXIO_SPI_8bitLsb FlexIO SPI 8-bit LSB first.
kFLEXIO_SPI_16bitMsb FlexIO SPI 16-bit MSB first.
kFLEXIO_SPI_16bitLsb FlexIO SPI 16-bit LSB first.

19.5.7 Function Documentation

19.5.7.1 void FLEXIO_SPI_MasterInit (FLEXIO_SPI_Type ∗ *base,* flexio_spi_master_config_t ∗ *masterConfig,* uint32_t *srcClock_Hz*)

The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_-MasterGetDefaultConfig().

Note

1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO S-PI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2∗2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)∗2=8.

Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
.enableMaster = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 500000,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);
```

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
masterConfig	Pointer to the flexio_spi_master_config_t structure.
srcClock_Hz	FlexIO source clock in Hz.

19.5.7.2 void FLEXIO_SPI_MasterDeinit (FLEXIO_SPI_Type ∗ *base*)

Parameters

base	Pointer to the FLEXIO_SPI_Type.

19.5.7.3 void FLEXIO_SPI_MasterGetDefaultConfig (flexio_spi_master_config_t ∗ *masterConfig*)

The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```
flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);
```

Parameters

masterConfig	Pointer to the flexio_spi_master_config_t structure.

19.5.7.4 void FLEXIO_SPI_SlaveInit (FLEXIO_SPI_Type ∗ *base,* flexio_spi_slave_config_t ∗ *slaveConfig*)

The configuration structure can be filled by the user, or be set with default values by the FLEXIO_SPI_-SlaveGetDefaultConfig().

Note

1.Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2.-FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 3∗2=6. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)∗2=8. Example

```
FLEXIO_SPI_Type spiDev = {
.flexioBase = FLEXIO,
.SDOPinIndex = 0,
```

```
.SDIPinIndex = 1,
.SCKPinIndex = 2,
.CSnPinIndex = 3,
.shifterIndex = {0,1},
.timerIndex = {0}
};
flexio_spi_slave_config_t config = {
.enableSlave = true,
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
.direction = kFLEXIO_SPI_MsbFirst,
.dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
slaveConfig	Pointer to the flexio_spi_slave_config_t structure.

19.5.7.5 void FLEXIO_SPI_SlaveDeinit (FLEXIO_SPI_Type ∗ *base*)

Parameters

base	Pointer to the FLEXIO_SPI_Type.

19.5.7.6 void FLEXIO_SPI_SlaveGetDefaultConfig (flexio_spi_slave_config_t ∗ *slaveConfig*)

The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

slaveConfig	Pointer to the flexio_spi_slave_config_t structure.

19.5.7.7 uint32_t FLEXIO_SPI_GetStatusFlags (FLEXIO_SPI_Type ∗ *base*)

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.

Returns

status flag; Use the status flag to AND the following flag mask and get the status.
- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

19.5.7.8 void FLEXIO_SPI_ClearStatusFlags (FLEXIO_SPI_Type * *base*, uint32_t *mask*)

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
mask	status flag The parameter can be any combination of the following values: • kFLEXIO_SPI_TxEmptyFlag • kFLEXIO_SPI_RxEmptyFlag

19.5.7.9 void FLEXIO_SPI_EnableInterrupts (FLEXIO_SPI_Type * *base*, uint32_t *mask*)

This function enables the FlexIO SPI interrupt.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
mask	interrupt source. The parameter can be any combination of the following values: • kFLEXIO_SPI_RxFullInterruptEnable • kFLEXIO_SPI_TxEmptyInterruptEnable

19.5.7.10 void FLEXIO_SPI_DisableInterrupts (FLEXIO_SPI_Type * *base*, uint32_t *mask*)

This function disables the FlexIO SPI interrupt.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
mask	interrupt source The parameter can be any combination of the following values: • kFLEXIO_SPI_RxFullInterruptEnable • kFLEXIO_SPI_TxEmptyInterruptEnable

19.5.7.11 void FLEXIO_SPI_EnableDMA (FLEXIO_SPI_Type * *base*, uint32_t *mask*, bool *enable*)

This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_-TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
mask	SPI DMA source.
enable	True means enable DMA, false means disable DMA.

19.5.7.12 static uint32_t FLEXIO_SPI_GetTxDataRegisterAddress (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*) `[inline], [static]`

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
direction	Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

19.5.7.13 static uint32_t FLEXIO_SPI_GetRxDataRegisterAddress (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*) `[inline], [static]`

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
direction	Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

19.5.7.14 static void FLEXIO_SPI_Enable (FLEXIO_SPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	Pointer to the FLEXIO_SPI_Type.
enable	True to enable, false does not have any effect.

19.5.7.15 void FLEXIO_SPI_MasterSetBaudRate (FLEXIO_SPI_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClockHz*)

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
baudRate_Bps	Baud Rate needed in Hz.
srcClockHz	SPI source clock frequency in Hz.

19.5.7.16 static void FLEXIO_SPI_WriteData (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*, uint16_t *data*) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
direction	Shift direction of MSB first or LSB first.
data	8 bit/16 bit data.

19.5.7.17 static uint16_t FLEXIO_SPI_ReadData (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
direction	Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

19.5.7.18 status_t FLEXIO_SPI_WriteBlocking (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*, const uint8_t * *buffer*, size_t *size*)

Note

This function blocks using the polling method until all bytes have been sent.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
direction	Shift direction of MSB first or LSB first.
buffer	The data bytes to send.

size	The number of data bytes to send.

Return values

kStatus_Success	Successfully create the handle.
kStatus_FLEXIO_SPI_-Timeout	The transfer timed out and was aborted.

19.5.7.19 status_t FLEXIO_SPI_ReadBlocking (FLEXIO_SPI_Type ∗ *base*, flexio_spi_shift_direction_t *direction*, uint8_t ∗ *buffer*, size_t *size*)

Note

This function blocks using the polling method until all bytes have been received.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
direction	Shift direction of MSB first or LSB first.
buffer	The buffer to store the received bytes.
size	The number of data bytes to be received.
direction	Shift direction of MSB first or LSB first.

Return values

kStatus_Success	Successfully create the handle.
kStatus_FLEXIO_SPI_-Timeout	The transfer timed out and was aborted.

19.5.7.20 status_t FLEXIO_SPI_MasterTransferBlocking (FLEXIO_SPI_Type ∗ *base*, flexio_spi_transfer_t ∗ *xfer*)

Note

This function blocks via polling until all bytes have been received.

Parameters

base	pointer to FLEXIO_SPI_Type structure
xfer	FlexIO SPI transfer structure, see flexio_spi_transfer_t.

Return values

kStatus_Success	Successfully create the handle.
kStatus_FLEXIO_SPI_-Timeout	The transfer timed out and was aborted.

19.5.7.21 status_t FLEXIO_SPI_MasterTransferCreateHandle (FLEXIO_SPI_Type ∗ *base*, flexio_spi_master_handle_t ∗ *handle*, flexio_spi_master_transfer_callback_t *callback*, void ∗ *userData*)

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
handle	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
callback	The callback function.
userData	The parameter of the callback function.

Return values

kStatus_Success	Successfully create the handle.
kStatus_OutOfRange	The FlexIO type/handle/ISR table out of range.

19.5.7.22 status_t FLEXIO_SPI_MasterTransferNonBlocking (FLEXIO_SPI_Type ∗ *base*, flexio_spi_master_handle_t ∗ *handle*, flexio_spi_transfer_t ∗ *xfer*)

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
handle	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
xfer	FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_FLEXIO_SPI_-Busy	SPI is not idle, is running another transfer.

19.5.7.23 void FLEXIO_SPI_MasterTransferAbort (FLEXIO_SPI_Type ∗ *base*, flexio_spi_master_handle_t ∗ *handle*)

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
handle	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

19.5.7.24 status_t FLEXIO_SPI_MasterTransferGetCount (FLEXIO_SPI_Type ∗ *base*, flexio_spi_master_handle_t ∗ *handle*, size_t ∗ *count*)

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
handle	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

19.5.7.25 void FLEXIO_SPI_MasterTransferHandleIRQ (void ∗ *spiType*, void ∗ *spiHandle*)

Parameters

spiType	Pointer to the FLEXIO_SPI_Type structure.
spiHandle	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

19.5.7.26 status_t FLEXIO_SPI_SlaveTransferCreateHandle (FLEXIO_SPI_Type * *base*, flexio_spi_slave_handle_t * *handle*, flexio_spi_slave_transfer_callback_t *callback*, void * *userData*)

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
handle	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
callback	The callback function.
userData	The parameter of the callback function.

Return values

kStatus_Success	Successfully create the handle.
kStatus_OutOfRange	The FlexIO type/handle/ISR table out of range.

19.5.7.27 status_t FLEXIO_SPI_SlaveTransferNonBlocking (FLEXIO_SPI_Type ∗ *base*, flexio_spi_slave_handle_t ∗ *handle*, flexio_spi_transfer_t ∗ *xfer*)

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

handle	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
base	Pointer to the FLEXIO_SPI_Type structure.
xfer	FlexIO SPI transfer structure. See flexio_spi_transfer_t.

Return values

kStatus_Success	Successfully start a transfer.
kStatus_InvalidArgument	Input argument is invalid.
kStatus_FLEXIO_SPI_-Busy	SPI is not idle; it is running another transfer.

19.5.7.28 static void FLEXIO_SPI_SlaveTransferAbort (FLEXIO_SPI_Type ∗ *base*, flexio_spi_slave_handle_t ∗ *handle*) `[inline], [static]`

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
handle	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

19.5.7.29 static status_t FLEXIO_SPI_SlaveTransferGetCount (FLEXIO_SPI_Type * *base*, flexio_spi_slave_handle_t * *handle*, size_t * *count*) [inline], [static]

Parameters

base	Pointer to the FLEXIO_SPI_Type structure.
handle	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

19.5.7.30 void FLEXIO_SPI_SlaveTransferHandleIRQ (void * *spiType*, void * *spiHandle*)

Parameters

spiType	Pointer to the FLEXIO_SPI_Type structure.
spiHandle	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

19.6 FlexIO UART Driver

19.6.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) function using the Flexible I/O.

FlexIO UART driver includes functional APIs and transactional APIs. Functional APIs target low-level APIs. Functional APIs can be used for the FlexIO UART initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO UART peripheral and how to organize functional APIs to meet the application requirements. All functional API use the FLEXIO_UART_Type ∗ as the first parameter. FlexIO UART functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the flexio_uart_handle_t as the second parameter. Initialize the handle by calling the FLEXIO_UART_TransferCreateHandle() API.

Transactional APIs support asynchronous transfer. This means that the functions FLEXIO_UART_SendNonBlocking() and FLEXIO_UART_ReceiveNonBlocking() set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the kStatus_FLEXIO_UART_TxIdle and kStatus_FLEXIO_UART_RxIdle status.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size through calling the FLEXIO_UART_InstallRingBuffer(). When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The function FLEXIO_UART_ReceiveNonBlocking() first gets data the from the ring buffer. If ring buffer does not have enough data, the function returns the data to the ring buffer and saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the statuskStatus_FLEXIO_UART_RxIdle status.

If the receive ring buffer is full, the upper layer is informed through a callback with status kStatus_FLEXIO_UART_RxRingBufferOverrun. In the callback function, the upper layer reads data from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when calling the FLEXIO_UART_InstallRingBuffer. Note that one byte is reserved for the ring buffer maintenance. Create a handle as follows.

```
FLEXIO_UART_InstallRingBuffer(&uartDev, &handle, &ringBuffer, 32);
```

In this example, the buffer size is 32. However, only 31 bytes are used for saving data.

19.6.2 Typical use case

19.6.2.1 FlexIO UART send/receive using a polling method

```
uint8_t ch;
```

```
FLEXIO_UART_Type uartDev;
status_t result = kStatus_Success;
flexio_uart_user_config user_config;
FLEXIO_UART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableUart = true;

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
//Check if configuration is correct.
if(result != kStatus_Success)
{
    return;
}
FLEXIO_UART_WriteBlocking(&uartDev, txbuff, sizeof(txbuff));

while(1)
{
    FLEXIO_UART_ReadBlocking(&uartDev, &ch, 1);
    FLEXIO_UART_WriteBlocking(&uartDev, &ch, 1);
}
```

19.6.2.2 FlexIO UART send/receive using an interrupt method

```
FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
      status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;
```

```
    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;

    result = FLEXIO_UART_Init(&uartDev, &user_config, 120000000U);
    //Check if configuration is correct.
    if(result != kStatus_Success)
    {
        return;
    }

    FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
      FLEXIO_UART_UserCallback, NULL);

    // Prepares to send.
    sendXfer.data = sendData;
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Sends out.
    FLEXIO_UART_SendNonBlocking(&uartDev, &g_uartHandle, &sendXfer);

    // Send finished.
    while (!txFinished)
    {
    }

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, NULL);

    // Receive finished.
    while (!rxFinished)
    {
    }

    // ...
}
```

19.6.2.3 FlexIO UART receive using the ringbuffer feature

```
#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE     32

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
      status_t status, void *userData)
{
```

```
    userData = userData;

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;

    result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
    //Check if configuration is correct.
    if(result != kStatus_Success)
    {
        return;
    }

    FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
      FLEXIO_UART_UserCallback, NULL);
    FLEXIO_UART_InstallRingBuffer(&uartDev, &g_uartHandle, ringBuffer, RING_BUFFER_SIZE);

    // Receive is working in the background to the ring buffer.

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = RX_DATA_SIZE;
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, &bytesRead);

    if (bytesRead = RX_DATA_SIZE) /* Have read enough data. */
    {
        ;
    }
    else
    {
        if (bytesRead) /* Received some data, process first. */
        {
            ;
        }

        // Receive finished.
        while (!rxFinished)
        {
        }
    }

    // ...
}
```

19.6.2.4 FlexIO UART send/receive using a DMA method

```
    FLEXIO_UART_Type uartDev;
    flexio_uart_handle_t g_uartHandle;
    dma_handle_t g_uartTxDmaHandle;
    dma_handle_t g_uartRxDmaHandle;
    flexio_uart_config_t user_config;
    flexio_uart_transfer_t sendXfer;
    flexio_uart_transfer_t receiveXfer;
    volatile bool txFinished;
    volatile bool rxFinished;
    uint8_t sendData[] = ['H', 'e', 'l', 'l', 'o'];
    uint8_t receiveData[32];

    void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
      status_t status, void *userData)
    {
        userData = userData;

        if (kStatus_FLEXIO_UART_TxIdle == status)
        {
            txFinished = true;
        }

        if (kStatus_FLEXIO_UART_RxIdle == status)
        {
            rxFinished = true;
        }
    }

    void main(void)
    {
        //...

        FLEXIO_UART_GetDefaultConfig(&user_config);
        user_config.baudRate_Bps = 115200U;
        user_config.enableUart = true;

        uartDev.flexioBase = BOARD_FLEXIO_BASE;
        uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
        uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
        uartDev.shifterIndex[0] = 0U;
        uartDev.shifterIndex[1] = 1U;
        uartDev.timerIndex[0] = 0U;
        uartDev.timerIndex[1] = 1U;
        result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
        //Check if configuration is correct.
        if(result != kStatus_Success)
        {
            return;
        }

        /* Init DMAMUX. */
        DMAMUX_Init(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR)

        /* Init the DMA/EDMA module */
#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
        DMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR);
        DMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
      FLEXIO_UART_TX_DMA_CHANNEL);
        DMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
      FLEXIO_UART_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
        edma_config_t edmaConfig;
```

```
        EDMA_GetDefaultConfig(&edmaConfig);
        EDMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR, &edmaConfig);
        EDMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL)
      ;
        EDMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL)
      ;
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

        dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
      shifterIndex[0]);
        dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
      shifterIndex[1]);

        /* Requests  DMA channels for transmit and receive. */
        DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL, (
      dma_request_source_t)dma_request_source_tx);
        DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL, (
      dma_request_source_t)dma_request_source_rx);
        DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL);
        DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL);

        FLEXIO_UART_TransferCreateHandleDMA(&uartDev, &g_uartHandle, FLEXIO_UART_UserCallback, NULL, &
      g_uartTxDmaHandle, &g_uartRxDmaHandle);

        // Prepares to send.
        sendXfer.data = sendData
        sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
        txFinished = false;

        // Sends out.
        FLEXIO_UART_SendDMA(&uartDev, &g_uartHandle, &sendXfer);

        // Send finished.
        while (!txFinished)
        {
        }

        // Prepares to receive.
        receiveXfer.data = receiveData;
        receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
        rxFinished = false;

        // Receives.
        FLEXIO_UART_ReceiveDMA(&uartDev, &g_uartHandle, &receiveXfer, NULL);

        // Receive finished.
        while (!rxFinished)
        {
        }

        // ...
    }
```

Data Structures

- struct FLEXIO_UART_Type
  Define FlexIO UART access structure typedef. More...
- struct flexio_uart_config_t
  Define FlexIO UART user configuration structure. More...
- struct flexio_uart_transfer_t
  Define FlexIO UART transfer structure. More...
- struct flexio_uart_handle_t

Define FLEXIO UART 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(∗ flexio_uart_transfer_callback_t)(FLEXIO_UART_Type ∗base, flexio_uart_handle_t ∗handle, status_t status, void ∗userData)
  FlexIO UART transfer callback function.

Enumerations

- enum {
  kStatus_FLEXIO_UART_TxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 0),
  kStatus_FLEXIO_UART_RxBusy = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 1),
  kStatus_FLEXIO_UART_TxIdle = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 2),
  kStatus_FLEXIO_UART_RxIdle = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 3),
  kStatus_FLEXIO_UART_ERROR = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 4),
  kStatus_FLEXIO_UART_RxRingBufferOverrun,
  kStatus_FLEXIO_UART_RxHardwareOverrun = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 6),
  kStatus_FLEXIO_UART_Timeout = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 7),
  kStatus_FLEXIO_UART_BaudrateNotSupport }
  Error codes for the UART driver.
- enum flexio_uart_bit_count_per_char_t {
  kFLEXIO_UART_7BitsPerChar = 7U,
  kFLEXIO_UART_8BitsPerChar = 8U,
  kFLEXIO_UART_9BitsPerChar = 9U }
  FlexIO UART bit count per char.
- enum _flexio_uart_interrupt_enable {
  kFLEXIO_UART_TxDataRegEmptyInterruptEnable = 0x1U,
  kFLEXIO_UART_RxDataRegFullInterruptEnable = 0x2U }
  Define FlexIO UART interrupt mask.
- enum _flexio_uart_status_flags {
  kFLEXIO_UART_TxDataRegEmptyFlag = 0x1U,
  kFLEXIO_UART_RxDataRegFullFlag = 0x2U,
  kFLEXIO_UART_RxOverRunFlag = 0x4U }
  Define FlexIO UART status mask.

Driver version

- #define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  FlexIO UART driver version.

Initialization and deinitialization

- status_t FLEXIO_UART_Init (FLEXIO_UART_Type *base, const flexio_uart_config_t *user-Config, uint32_t srcClock_Hz)
  Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration.
- void FLEXIO_UART_Deinit (FLEXIO_UART_Type *base)
  Resets the FlexIO UART shifter and timer config.
- void FLEXIO_UART_GetDefaultConfig (flexio_uart_config_t *userConfig)
  Gets the default configuration to configure the FlexIO UART.

Status

- uint32_t FLEXIO_UART_GetStatusFlags (FLEXIO_UART_Type *base)
  Gets the FlexIO UART status flags.
- void FLEXIO_UART_ClearStatusFlags (FLEXIO_UART_Type *base, uint32_t mask)
  Gets the FlexIO UART status flags.

Interrupts

- void FLEXIO_UART_EnableInterrupts (FLEXIO_UART_Type *base, uint32_t mask)
  Enables the FlexIO UART interrupt.
- void FLEXIO_UART_DisableInterrupts (FLEXIO_UART_Type *base, uint32_t mask)
  Disables the FlexIO UART interrupt.

DMA Control

- static uint32_t FLEXIO_UART_GetTxDataRegisterAddress (FLEXIO_UART_Type *base)
  Gets the FlexIO UARt transmit data register address.
- static uint32_t FLEXIO_UART_GetRxDataRegisterAddress (FLEXIO_UART_Type *base)
  Gets the FlexIO UART receive data register address.
- static void FLEXIO_UART_EnableTxDMA (FLEXIO_UART_Type *base, bool enable)
  Enables/disables the FlexIO UART transmit DMA.
- static void FLEXIO_UART_EnableRxDMA (FLEXIO_UART_Type *base, bool enable)
  Enables/disables the FlexIO UART receive DMA.

Bus Operations

- static void FLEXIO_UART_Enable (FLEXIO_UART_Type *base, bool enable)

Enables/disables the FlexIO UART module operation.

- static void FLEXIO_UART_WriteByte (FLEXIO_UART_Type *base, const uint8_t *buffer)
  Writes one byte of data.
- static void FLEXIO_UART_ReadByte (FLEXIO_UART_Type *base, uint8_t *buffer)
  Reads one byte of data.
- status_t FLEXIO_UART_WriteBlocking (FLEXIO_UART_Type *base, const uint8_t *txData, size_t txSize)
  Sends a buffer of data bytes.
- status_t FLEXIO_UART_ReadBlocking (FLEXIO_UART_Type *base, uint8_t *rxData, size_t rxSize)
  Receives a buffer of bytes.

Transactional

- status_t FLEXIO_UART_TransferCreateHandle (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_callback_t callback, void *userData)
  Initializes the UART handle.
- void FLEXIO_UART_TransferStartRingBuffer (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
  Sets up the RX ring buffer.
- void FLEXIO_UART_TransferStopRingBuffer (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
  Aborts the background transfer and uninstalls the ring buffer.
- status_t FLEXIO_UART_TransferSendNonBlocking (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer)
  Transmits a buffer of data using the interrupt method.
- void FLEXIO_UART_TransferAbortSend (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
  Aborts the interrupt-driven data transmit.
- status_t FLEXIO_UART_TransferGetSendCount (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
  Gets the number of bytes sent.
- status_t FLEXIO_UART_TransferReceiveNonBlocking (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer, size_t *receivedBytes)
  Receives a buffer of data using the interrupt method.
- void FLEXIO_UART_TransferAbortReceive (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle)
  Aborts the receive data which was using IRQ.
- status_t FLEXIO_UART_TransferGetReceiveCount (FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, size_t *count)
  Gets the number of bytes received.
- void FLEXIO_UART_TransferHandleIRQ (void *uartType, void *uartHandle)
  FlexIO UART IRQ handler function.

19.6.3 Data Structure Documentation

19.6.3.1 struct FLEXIO_UART_Type

Data Fields

- FLEXIO_Type * flexioBase
  FlexIO base pointer.
- uint8_t TxPinIndex
  Pin select for UART_Tx.
- uint8_t RxPinIndex
  Pin select for UART_Rx.
- uint8_t shifterIndex [2]
  Shifter index used in FlexIO UART.
- uint8_t timerIndex [2]
  Timer index used in FlexIO UART.

Field Documentation

(1) FLEXIO_Type* FLEXIO_UART_Type::flexioBase

(2) uint8_t FLEXIO_UART_Type::TxPinIndex

(3) uint8_t FLEXIO_UART_Type::RxPinIndex

(4) uint8_t FLEXIO_UART_Type::shifterIndex[2]

(5) uint8_t FLEXIO_UART_Type::timerIndex[2]

19.6.3.2 struct flexio_uart_config_t

Data Fields

- bool enableUart
  Enable/disable FlexIO UART TX & RX.
- bool enableInDoze
  Enable/disable FlexIO operation in doze mode.
- bool enableInDebug
  Enable/disable FlexIO operation in debug mode.
- bool enableFastAccess
  `Enable/disable fast access to FlexIO registers,`
  fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.
- uint32_t baudRate_Bps
  Baud rate in Bps.
- flexio_uart_bit_count_per_char_t bitCountPerChar
  number of bits, 7/8/9 -bit

Field Documentation

(1) bool flexio_uart_config_t::enableUart

(2) bool flexio_uart_config_t::enableFastAccess

(3) uint32_t flexio_uart_config_t::baudRate_Bps

19.6.3.3 struct flexio_uart_transfer_t

Data Fields

- size_t dataSize
  Transfer size.
- uint8_t ∗ data
  The buffer of data to be transfer.
- uint8_t ∗ rxData
  The buffer to receive data.
- const uint8_t ∗ txData
  The buffer of data to be sent.

Field Documentation

(1) uint8_t∗ flexio_uart_transfer_t::data

(2) uint8_t∗ flexio_uart_transfer_t::rxData

(3) const uint8_t∗ flexio_uart_transfer_t::txData

19.6.3.4 struct _flexio_uart_handle

Data Fields

- const uint8_t ∗volatile txData
  Address of remaining data to send.
- volatile size_t txDataSize
  Size of the remaining data to send.
- uint8_t ∗volatile rxData
  Address of remaining data to receive.
- volatile size_t rxDataSize
  Size of the remaining data to receive.
- size_t txDataSizeAll
  Total bytes to be sent.
- size_t rxDataSizeAll
  Total bytes to be received.
- 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.
- flexio_uart_transfer_callback_t callback
  Callback function.
- void ∗ userData
  UART callback function parameter.

- volatile uint8_t txState
  TX transfer state.
- volatile uint8_t rxState
  RX transfer state.

Field Documentation

(1) const uint8_t∗ volatile flexio_uart_handle_t::txData

(2) volatile size_t flexio_uart_handle_t::txDataSize

(3) uint8_t∗ volatile flexio_uart_handle_t::rxData

(4) volatile size_t flexio_uart_handle_t::rxDataSize

(5) size_t flexio_uart_handle_t::txDataSizeAll

(6) size_t flexio_uart_handle_t::rxDataSizeAll

(7) uint8_t∗ flexio_uart_handle_t::rxRingBuffer

(8) size_t flexio_uart_handle_t::rxRingBufferSize

(9) volatile uint16_t flexio_uart_handle_t::rxRingBufferHead

(10) volatile uint16_t flexio_uart_handle_t::rxRingBufferTail

(11) flexio_uart_transfer_callback_t flexio_uart_handle_t::callback

(12) void∗ flexio_uart_handle_t::userData

(13) volatile uint8_t flexio_uart_handle_t::txState

19.6.4 Macro Definition Documentation

19.6.4.1 #define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))

19.6.4.2 #define UART_RETRY_TIMES 0U /∗ Defining to zero means to keep waiting for the flag until it is assert/deassert. ∗/

19.6.5 Typedef Documentation

19.6.5.1 typedef void(∗ flexio_uart_transfer_callback_t)(FLEXIO_UART_Type ∗base, flexio_uart_handle_t ∗handle, status_t status, void ∗userData)

19.6.6 Enumeration Type Documentation

19.6.6.1 anonymous enum

Enumerator

kStatus_FLEXIO_UART_TxBusy Transmitter is busy.
kStatus_FLEXIO_UART_RxBusy Receiver is busy.
kStatus_FLEXIO_UART_TxIdle UART transmitter is idle.
kStatus_FLEXIO_UART_RxIdle UART receiver is idle.
kStatus_FLEXIO_UART_ERROR ERROR happens on UART.
kStatus_FLEXIO_UART_RxRingBufferOverrun UART RX software ring buffer overrun.
kStatus_FLEXIO_UART_RxHardwareOverrun UART RX receiver overrun.
kStatus_FLEXIO_UART_Timeout UART times out.
kStatus_FLEXIO_UART_BaudrateNotSupport Baudrate is not supported in current clock source.

19.6.6.2 enum flexio_uart_bit_count_per_char_t

Enumerator

kFLEXIO_UART_7BitsPerChar 7-bit data characters
kFLEXIO_UART_8BitsPerChar 8-bit data characters
kFLEXIO_UART_9BitsPerChar 9-bit data characters

19.6.6.3 enum _flexio_uart_interrupt_enable

Enumerator

kFLEXIO_UART_TxDataRegEmptyInterruptEnable Transmit buffer empty interrupt enable.
kFLEXIO_UART_RxDataRegFullInterruptEnable Receive buffer full interrupt enable.

19.6.6.4 enum _flexio_uart_status_flags

Enumerator

kFLEXIO_UART_TxDataRegEmptyFlag Transmit buffer empty flag.
kFLEXIO_UART_RxDataRegFullFlag Receive buffer full flag.
kFLEXIO_UART_RxOverRunFlag Receive buffer over run flag.

19.6.7 Function Documentation

19.6.7.1 status_t FLEXIO_UART_Init (FLEXIO_UART_Type * *base*, const flexio_uart_config_t * *userConfig*, uint32_t *srcClock_Hz*)

The configuration structure can be filled by the user or be set with default values by FLEXIO_UART_-GetDefaultConfig().

Example

```
FLEXIO_UART_Type base = {
.flexioBase = FLEXIO,
.TxPinIndex = 0,
.RxPinIndex = 1,
.shifterIndex = {0,1},
.timerIndex = {0,1}
};
flexio_uart_config_t config = {
.enableInDoze = false,
.enableInDebug = true,
.enableFastAccess = false,
.baudRate_Bps = 115200U,
.bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
userConfig	Pointer to the flexio_uart_config_t structure.
srcClock_Hz	FlexIO source clock in Hz.

Return values

kStatus_Success	Configuration success.
kStatus_FLEXIO_UART-_BaudrateNotSupport	Baudrate is not supported for current clock source frequency.

19.6.7.2 void FLEXIO_UART_Deinit (FLEXIO_UART_Type * *base*)

Note

After calling this API, call the FLEXO_UART_Init to use the FlexIO UART module.

Parameters

base	Pointer to FLEXIO_UART_Type structure

19.6.7.3 void FLEXIO_UART_GetDefaultConfig (flexio_uart_config_t ∗ *userConfig*)

The configuration can be used directly for calling the FLEXIO_UART_Init(). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

userConfig	Pointer to the flexio_uart_config_t structure.

19.6.7.4 uint32_t FLEXIO_UART_GetStatusFlags (FLEXIO_UART_Type ∗ *base*)

Parameters

base	Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART status flags.

19.6.7.5 void FLEXIO_UART_ClearStatusFlags (FLEXIO_UART_Type ∗ *base*, uint32_t *mask*)

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
mask	Status flag. The parameter can be any combination of the following values: • kFLEXIO_UART_TxDataRegEmptyFlag • kFLEXIO_UART_RxEmptyFlag • kFLEXIO_UART_RxOverRunFlag

19.6.7.6 void FLEXIO_UART_EnableInterrupts (FLEXIO_UART_Type ∗ *base*, uint32_t *mask*)

This function enables the FlexIO UART interrupt.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
mask	Interrupt source.

19.6.7.7 void FLEXIO_UART_DisableInterrupts (FLEXIO_UART_Type * *base*, uint32_t *mask*)

This function disables the FlexIO UART interrupt.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
mask	Interrupt source.

19.6.7.8 static uint32_t FLEXIO_UART_GetTxDataRegisterAddress (FLEXIO_UART_Type * *base*) [inline], [static]

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART transmit data register address.

19.6.7.9 static uint32_t FLEXIO_UART_GetRxDataRegisterAddress (FLEXIO_UART_Type * *base*) [inline], [static]

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.

Returns

FlexIO UART receive data register address.

19.6.7.10 static void FLEXIO_UART_EnableTxDMA (FLEXIO_UART_Type * *base,* bool *enable*) [inline], [static]

This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
enable	True to enable, false to disable.

19.6.7.11 static void FLEXIO_UART_EnableRxDMA (FLEXIO_UART_Type * *base*, bool *enable*) [inline], [static]

This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_-RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
enable	True to enable, false to disable.

19.6.7.12 static void FLEXIO_UART_Enable (FLEXIO_UART_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	Pointer to the FLEXIO_UART_Type.
enable	True to enable, false does not have any effect.

19.6.7.13 static void FLEXIO_UART_WriteByte (FLEXIO_UART_Type * *base*, const uint8_t * *buffer*) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.

buffer	The data bytes to send.

19.6.7.14 static void FLEXIO_UART_ReadByte (FLEXIO_UART_Type * *base*, uint8_t * *buffer*) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
buffer	The buffer to store the received bytes.

19.6.7.15 status_t FLEXIO_UART_WriteBlocking (FLEXIO_UART_Type * *base*, const uint8_t * *txData*, size_t *txSize*)

Note

This function blocks using the polling method until all bytes have been sent.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
txData	The data bytes to send.
txSize	The number of data bytes to send.

Return values

kStatus_FLEXIO_UART-_Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully wrote all data.

19.6.7.16 status_t FLEXIO_UART_ReadBlocking (FLEXIO_UART_Type * *base*, uint8_t * *rxData*, size_t *rxSize*)

Note

This function blocks using the polling method until all bytes have been received.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
rxData	The buffer to store the received bytes.
rxSize	The number of data bytes to be received.

Return values

kStatus_FLEXIO_UART-_Timeout	Transmission timed out and was aborted.
kStatus_Success	Successfully received all data.

19.6.7.17 status_t FLEXIO_UART_TransferCreateHandle (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_callback_t *callback*, void * *userData*)

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the FLEXIO_UART_TransferReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as `ringBuffer`.

Parameters

base	to FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.
callback	The callback function.
userData	The parameter of the callback function.

Return values

kStatus_Success	Successfully create the handle.
kStatus_OutOfRange	The FlexIO type/handle/ISR table out of range.

19.6.7.18 void FLEXIO_UART_TransferStartRingBuffer (FLEXIO_UART_Type * *base*, flexio_uart_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_ReceiveNonBlocking() API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, only 31 bytes are used for saving data.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.
ringBuffer	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
ringBufferSize	Size of the ring buffer.

19.6.7.19 void FLEXIO_UART_TransferStopRingBuffer (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*)

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.

19.6.7.20 status_t FLEXIO_UART_TransferSendNonBlocking (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_t * *xfer*)

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in ISR, the FlexIO UART driver calls the callback function and passes the kStatus_FLEXIO_UART_TxIdle as status parameter.

Note

The kStatus_FLEXIO_UART_TxIdle is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.
xfer	FlexIO UART transfer structure. See flexio_uart_transfer_t.

Return values

kStatus_Success	Successfully starts the data transmission.
kStatus_UART_TxBusy	Previous transmission still not finished, data not written to the TX register.

19.6.7.21 void FLEXIO_UART_TransferAbortSend (FLEXIO_UART_Type ∗ *base*, flexio_uart_handle_t ∗ *handle*)

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.

19.6.7.22 status_t FLEXIO_UART_TransferGetSendCount (FLEXIO_UART_Type ∗ *base*, flexio_uart_handle_t ∗ *handle*, size_t ∗ *count*)

This function gets the number of bytes sent driven by interrupt.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.
count	Number of bytes sent so far by the non-blocking transaction.

Return values

kStatus_NoTransferIn-Progress	transfer has finished or no transfer in progress.
kStatus_Success	Successfully return the count.

19.6.7.23 status_t FLEXIO_UART_TransferReceiveNonBlocking (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_t * *xfer*, size_t * *receivedBytes*)

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter `receivedBytes` shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter kStatus_-UART_RxIdle. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to xfer->data. This function returns with the parameter `receivedBytes` set to 5. For the last 5 bytes, newly arrived data is saved from the xfer->data[5]. When 5 bytes are received, the UART driver notifies 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

base	Pointer to the FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.
xfer	UART transfer structure. See flexio_uart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into the transmit queue.
kStatus_FLEXIO_UART-_RxBusy	Previous receive request is not finished.

19.6.7.24 void FLEXIO_UART_TransferAbortReceive (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*)

This function aborts the receive data which was using IRQ.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.

19.6.7.25 status_t FLEXIO_UART_TransferGetReceiveCount (FLEXIO_UART_Type * *base,* flexio_uart_handle_t * *handle,* size_t * *count*)

This function gets the number of bytes received driven by interrupt.

Parameters

base	Pointer to the FLEXIO_UART_Type structure.
handle	Pointer to the flexio_uart_handle_t structure to store the transfer state.
count	Number of bytes received so far by the non-blocking transaction.

Return values

kStatus_NoTransferIn-Progress	transfer has finished or no transfer in progress.
kStatus_Success	Successfully return the count.

19.6.7.26 void FLEXIO_UART_TransferHandleIRQ (void ∗ *uartType,* void ∗ *uartHandle*)

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

uartType	Pointer to the FLEXIO_UART_Type structure.
uartHandle	Pointer to the flexio_uart_handle_t structure to store the transfer state.

Chapter 20
FLEXSPI: Flexible Serial Peripheral Interface Driver

20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flexible Serial Peripheral Interface (FLEXSPI) module of MCUXpresso SDK/i.MX devices.

FLEXSPI driver includes functional APIs and interrupt/EDMA non-blocking transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for FLEXSPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FLEXSPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FLEXSPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the flexspi_handle_t/flexspi_edma_handle_t as the second parameter. Initialize the handle for interrupt non-blocking transfer by calling the FLEXSPI_TransferCreateHandle API. Initialize the handle for interrupt non-blocking transfer by calling the FLEXSPI_TransferCreateHandleEDMA API.

Transactional APIs support asynchronous transfer. This means that the functions FLEXSPI_TransferNonBlocking() and FLEXSPI_TransferEDMA() set up data transfer. When the transfer completes, the upper layer is notified through a callback function with the kStatus_FLEXSPI_Idle status.

Modules

- FLEXSPI DMA Driver

Data Structures

- struct flexspi_config_t
  FLEXSPI configuration structure. More...
- struct flexspi_device_config_t
  External device configuration items. More...
- struct flexspi_transfer_t
  Transfer structure for FLEXSPI. More...
- struct flexspi_handle_t
  Transfer handle structure for FLEXSPI. More...

Macros

- #define FLEXSPI_LUT_SEQ(cmd0, pad0, op0, cmd1, pad1, op1)
  Formula to form FLEXSPI instructions in LUT table.

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 {

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, 3, 5))

  FLEXSPI driver version 2.3.5.

Initialization and deinitialization

- uint32_t FLEXSPI_GetInstance (FLEXSPI_Type *base)

  Get the instance number for FLEXSPI.
- status_t FLEXSPI_CheckAndClearError (FLEXSPI_Type *base, uint32_t status)

  Check and clear IP command execution errors.
- void FLEXSPI_Init (FLEXSPI_Type *base, const flexspi_config_t *config)

  Initializes the FLEXSPI module and internal state.
- void FLEXSPI_GetDefaultConfig (flexspi_config_t *config)

  Gets default settings for FLEXSPI.
- void FLEXSPI_Deinit (FLEXSPI_Type *base)

  Deinitializes the FLEXSPI module.

- void FLEXSPI_UpdateDllValue (FLEXSPI_Type *base, flexspi_device_config_t *config, flexspi_port_t port)
  Update FLEXSPI DLL value depending on currently flexspi root clock.
- void FLEXSPI_SetFlashConfig (FLEXSPI_Type *base, flexspi_device_config_t *config, flexspi_port_t port)
  Configures the connected device parameter.
- static void FLEXSPI_SoftwareReset (FLEXSPI_Type *base)
  Software reset for the FLEXSPI logic.
- static void FLEXSPI_Enable (FLEXSPI_Type *base, bool enable)
  Enables or disables the FLEXSPI module.

Interrupts

- static void FLEXSPI_EnableInterrupts (FLEXSPI_Type *base, uint32_t mask)
  Enables the FLEXSPI interrupts.
- static void FLEXSPI_DisableInterrupts (FLEXSPI_Type *base, uint32_t mask)
  Disable the FLEXSPI interrupts.

DMA control

- static void FLEXSPI_EnableTxDMA (FLEXSPI_Type *base, bool enable)
  Enables or disables FLEXSPI IP Tx FIFO DMA requests.
- static void FLEXSPI_EnableRxDMA (FLEXSPI_Type *base, bool enable)
  Enables or disables FLEXSPI IP Rx FIFO DMA requests.
- static uint32_t FLEXSPI_GetTxFifoAddress (FLEXSPI_Type *base)
  Gets FLEXSPI IP tx fifo address for DMA transfer.
- static uint32_t FLEXSPI_GetRxFifoAddress (FLEXSPI_Type *base)
  Gets FLEXSPI IP rx fifo address for DMA transfer.

FIFO control

- static void FLEXSPI_ResetFifos (FLEXSPI_Type *base, bool txFifo, bool rxFifo)
  Clears the FLEXSPI IP FIFO logic.
- static void FLEXSPI_GetFifoCounts (FLEXSPI_Type *base, size_t *txCount, size_t *rxCount)
  Gets the valid data entries in the FLEXSPI FIFOs.

Status

- static uint32_t FLEXSPI_GetInterruptStatusFlags (FLEXSPI_Type *base)
  Get the FLEXSPI interrupt status flags.
- static void FLEXSPI_ClearInterruptStatusFlags (FLEXSPI_Type *base, uint32_t mask)
  Get the FLEXSPI interrupt status flags.
- static void FLEXSPI_GetDataLearningPhase (FLEXSPI_Type *base, uint8_t *portAPhase, uint8_t *portBPhase)
  Gets the sampling clock phase selection after Data Learning.
- static flexspi_arb_command_source_t FLEXSPI_GetArbitratorCommandSource (FLEXSPI_Type *base)
  Gets the trigger source of current command sequence granted by arbitrator.
- static flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode (FLEXSPI_Type *base, uint8_t *index)

Gets the error code when IP command error detected.
- static flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode (FLEXSPI_Type *base, uint8_t *index)
  Gets the error code when AHB command error detected.
- static bool FLEXSPI_GetBusIdleStatus (FLEXSPI_Type *base)
  Returns whether the bus is idle.

Bus Operations

- void FLEXSPI_UpdateRxSampleClock (FLEXSPI_Type *base, flexspi_read_sample_clock_t clockSource)
  Update read sample clock source.
- 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.

20.2 Data Structure Documentation

20.2.1 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 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 ahbGrantTimeoutCyle∗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.
- uint8_t ahbGrantTimeoutCycle
  `Timeout wait cycle for AHB command grant,`
  timeout after ahbGrantTimeoutCyle∗1024 AHB clock cycles.
- uint16_t ahbBusTimeoutCycle
  `Timeout wait cycle for AHB read/write access,`
  timeout after ahbBusTimeoutCycle∗1024 AHB clock cycles.
- uint8_t resumeWaitCycle
  `Wait cycle for idle state before suspended command sequence`
  resume, timeout after ahbBusTimeoutCycle AHB clock cycles.
- flexspi_ahbBuffer_config_t buffer [FSL_FEATURE_FLEXSPI_AHB_BUFFER_COUNT]
  AHB buffer size.
- bool enableClearAHBBufferOpt
  `Enable/disable automatically clean AHB RX Buffer and TX Buffer`
  when FLEXSPI returns STOP mode ACK.
- bool enableReadAddressOpt
  Enable/disable remove AHB read burst start address alignment limitation.
- bool enableAHBPrefetch
  `Enable/disable AHB read prefetch feature, when enabled, FLEXSPI`
  will fetch more data than current AHB burst.
- bool enableAHBBufferable
  `Enable/disable AHB bufferable write access support, when enabled,`
  FLEXSPI return before waiting for command execution finished.

- bool enableAHBCachable
  Enable AHB bus cachable read access support.

Field Documentation

(1) flexspi_read_sample_clock_t flexspi_config_t::rxSampleClock

(2) bool flexspi_config_t::enableSckFreeRunning

(3) bool flexspi_config_t::enableDoze

(4) bool flexspi_config_t::enableHalfSpeedAccess

(5) bool flexspi_config_t::enableSckBDiffOpt

(6) bool flexspi_config_t::enableSameConfigForAll

(7) uint16_t flexspi_config_t::seqTimeoutCycle

(8) uint8_t flexspi_config_t::ipGrantTimeoutCycle

(9) uint8_t flexspi_config_t::txWatermark

(10) uint8_t flexspi_config_t::rxWatermark

(11) uint8_t flexspi_config_t::ahbGrantTimeoutCycle

(12) uint16_t flexspi_config_t::ahbBusTimeoutCycle

(13) uint8_t flexspi_config_t::resumeWaitCycle

(14) flexspi_ahbBuffer_config_t flexspi_config_t::buffer[FSL_FEATURE_FLEXSPI_AHB_BUFFE-R_COUNT]

(15) bool flexspi_config_t::enableClearAHBBufferOpt

(16) bool flexspi_config_t::enableReadAddressOpt

when enable, there is no AHB read burst start address alignment limitation.

(17) bool flexspi_config_t::enableAHBPrefetch

(18) bool flexspi_config_t::enableAHBBufferable

(19) bool flexspi_config_t::enableAHBCachable

20.2.2 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.*

Field Documentation

(1) uint32_t flexspi_device_config_t::flexspiRootClk

(2) bool flexspi_device_config_t::isSck2Enabled

(3) uint32_t flexspi_device_config_t::flashSize

(4) flexspi_cs_interval_cycle_unit_t flexspi_device_config_t::CSIntervalUnit

(5) uint16_t flexspi_device_config_t::CSInterval

(6) uint8_t flexspi_device_config_t::CSHoldTime

(7) uint8_t flexspi_device_config_t::CSSetupTime

(8) uint8_t flexspi_device_config_t::dataValidTime

(9) uint8_t flexspi_device_config_t::columnspace

(10) bool flexspi_device_config_t::enableWordAddress

(11) uint8_t flexspi_device_config_t::AWRSeqIndex

(12) uint8_t flexspi_device_config_t::AWRSeqNumber

(13) uint8_t flexspi_device_config_t::ARDSeqIndex

(14) uint8_t flexspi_device_config_t::ARDSeqNumber

(15) flexspi_ahb_write_wait_unit_t flexspi_device_config_t::AHBWriteWaitUnit

(16) uint16_t flexspi_device_config_t::AHBWriteWaitInterval

(17) bool flexspi_device_config_t::enableWriteMask

20.2.3 struct flexspi_transfer_t

Data Fields

- uint32_t deviceAddress
  Operation device address.
- flexspi_port_t port
  Operation port.
- flexspi_command_type_t cmdType
  Execution command type.
- uint8_t seqIndex
  Sequence ID for command.
- uint8_t SeqNumber
  Sequence number for command.
- uint32_t * data
  Data buffer.
- size_t dataSize
  Data size in bytes.

Field Documentation

(1) uint32_t flexspi_transfer_t::deviceAddress

(2) flexspi_port_t flexspi_transfer_t::port

(3) flexspi_command_type_t flexspi_transfer_t::cmdType

(4) uint8_t flexspi_transfer_t::seqIndex

(5) uint8_t flexspi_transfer_t::SeqNumber

(6) uint32_t* flexspi_transfer_t::data

(7) size_t flexspi_transfer_t::dataSize

20.2.4 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.

Field Documentation

(1) uint32_t* flexspi_handle_t::data

(2) size_t flexspi_handle_t::dataSize

(3) size_t flexspi_handle_t::transferTotalSize

(4) void* flexspi_handle_t::userData

20.3 Macro Definition Documentation

20.3.1 #define FSL_FLEXSPI_DRIVER_VERSION (MAKE_VERSION(2, 3, 5))

20.3.2 #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))
```

20.4 Typedef Documentation

20.4.1 typedef void(∗ flexspi_transfer_callback_t)(FLEXSPI_Type ∗base, flexspi_handle_t ∗handle, status_t status, void ∗userData)

20.5 Enumeration Type Documentation

20.5.1 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.

20.5.2 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.

20.5.3 enum flexspi_pad_t

Enumerator

kFLEXSPI_1PAD Transmit command/address and transmit/receive data only through DATA0/DATA1.
kFLEXSPI_2PAD Transmit command/address and transmit/receive data only through DATA[1:0].

kFLEXSPI_4PAD Transmit command/address and transmit/receive data only through DATA[3:0].

kFLEXSPI_8PAD Transmit command/address and transmit/receive data only through DATA[7:0].

20.5.4 enum flexspi_flags_t

Enumerator

kFLEXSPI_SequenceExecutionTimeoutFlag Sequence execution timeout.
kFLEXSPI_AhbBusTimeoutFlag AHB Bus timeout.
kFLEXSPI_SckStoppedBecauseTxEmptyFlag SCK is stopped during command sequence because Async TX FIFO empty.
kFLEXSPI_SckStoppedBecauseRxFullFlag SCK is stopped during command sequence because Async RX FIFO full.
kFLEXSPI_DataLearningFailedFlag Data learning failed.
kFLEXSPI_IpTxFifoWatermarkEmptyFlag IP TX FIFO WaterMark empty.
kFLEXSPI_IpRxFifoWatermarkAvailableFlag IP RX FIFO WaterMark available.
kFLEXSPI_AhbCommandSequenceErrorFlag AHB triggered Command Sequences Error.
kFLEXSPI_IpCommandSequenceErrorFlag IP triggered Command Sequences Error.
kFLEXSPI_AhbCommandGrantTimeoutFlag AHB triggered Command Sequences Grant Timeout.

kFLEXSPI_IpCommandGrantTimeoutFlag IP triggered Command Sequences Grant Timeout.
kFLEXSPI_IpCommandExecutionDoneFlag IP triggered Command Sequences Execution finished.
kFLEXSPI_AllInterruptFlags All flags.

20.5.5 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.

20.5.6 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.

20.5.7 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.

20.5.8 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_RWDS_SDR used in DDR sequence.
kFLEXSPI_IpCmdErrorDdrDummyInSdrSequence Instruction DUMMY_DDR/DUMMY_RWDS_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.

20.5.9 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_RWDS_SDR used in DDR sequence.
kFLEXSPI_AhbCmdErrorDdrDummyInSdrSequence Instruction DUMMY_DDR/DUMMY_RWDS_DDR used in SDR sequence.
kFLEXSPI_AhbCmdSequenceExecutionTimeout Sequence execution timeout.

20.5.10 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.

20.5.11 enum flexspi_arb_command_source_t

20.5.12 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.

20.6 Function Documentation

20.6.1 uint32_t FLEXSPI_GetInstance (FLEXSPI_Type ∗ *base*)

Parameters

base	FLEXSPI base pointer.

20.6.2 status_t FLEXSPI_CheckAndClearError (FLEXSPI_Type ∗ *base,* uint32_t *status*)

Parameters

base	FLEXSPI base pointer.
status	interrupt status.

20.6.3 void FLEXSPI_Init (FLEXSPI_Type ∗ *base,* const flexspi_config_t ∗ *config*)

This function enables the clock for FLEXSPI and also configures the FLEXSPI with the input configure parameters. Users should call this function before any FLEXSPI operations.

Parameters

base	FLEXSPI peripheral base address.
config	FLEXSPI configure structure.

20.6.4 void FLEXSPI_GetDefaultConfig (flexspi_config_t ∗ *config*)

Parameters

config	FLEXSPI configuration structure.

20.6.5 void FLEXSPI_Deinit (FLEXSPI_Type * *base*)

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

base	FLEXSPI peripheral base address.

20.6.6 void FLEXSPI_UpdateDllValue (FLEXSPI_Type * *base*, flexspi_device_config_t * *config*, flexspi_port_t *port*)

Parameters

base	FLEXSPI peripheral base address.
config	Flash configuration parameters.
port	FLEXSPI Operation port.

20.6.7 void FLEXSPI_SetFlashConfig (FLEXSPI_Type * *base*, flexspi_device_config_t * *config*, flexspi_port_t *port*)

This function configures the connected device relevant parameters, such as the size, command, and so on. The flash configuration value cannot have a default value. The user needs to configure it according to the connected device.

Parameters

base	FLEXSPI peripheral base address.
config	Flash configuration parameters.
port	FLEXSPI Operation port.

20.6.8 static void FLEXSPI_SoftwareReset (FLEXSPI_Type * *base*) [inline], [static]

This function sets the software reset flags for both AHB and buffer domain and resets both AHB buffer and also IP FIFOs.

Parameters

base	FLEXSPI peripheral base address.

20.6.9 static void FLEXSPI_Enable (FLEXSPI_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
enable	True means enable FLEXSPI, false means disable.

20.6.10 static void FLEXSPI_EnableInterrupts (FLEXSPI_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
mask	FLEXSPI interrupt source.

20.6.11 static void FLEXSPI_DisableInterrupts (FLEXSPI_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
mask	FLEXSPI interrupt source.

20.6.12 static void FLEXSPI_EnableTxDMA (FLEXSPI_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
enable	Enable flag for transmit DMA request. Pass true for enable, false for disable.

20.6.13 static void FLEXSPI_EnableRxDMA (FLEXSPI_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
enable	Enable flag for receive DMA request. Pass true for enable, false for disable.

20.6.14 static uint32_t FLEXSPI_GetTxFifoAddress (FLEXSPI_Type * *base*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.

Return values

The	tx fifo address.

20.6.15 static uint32_t FLEXSPI_GetRxFifoAddress (FLEXSPI_Type * *base*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.

Return values

The	rx fifo address.

20.6.16 static void FLEXSPI_ResetFifos (FLEXSPI_Type * *base,* bool *txFifo,* bool *rxFifo*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
txFifo	Pass true to reset TX FIFO.
rxFifo	Pass true to reset RX FIFO.

20.6.17 static void FLEXSPI_GetFifoCounts (FLEXSPI_Type * *base,* size_t * *txCount,* size_t * *rxCount*) [inline], [static]

Parameters

	base	FLEXSPI peripheral base address.
out	*txCount*	Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
out	*rxCount*	Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

20.6.18 static uint32_t FLEXSPI_GetInterruptStatusFlags (FLEXSPI_Type * *base*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.

Return values

interrupt	status flag, use status flag to AND flexspi_flags_t could get the related status.

20.6.19 static void FLEXSPI_ClearInterruptStatusFlags (FLEXSPI_Type * *base,* uint32_t *mask*) `[inline]`, `[static]`

Parameters

base	FLEXSPI peripheral base address.
mask	FLEXSPI interrupt source.

20.6.20 static void FLEXSPI_GetDataLearningPhase (FLEXSPI_Type * *base,* uint8_t * *portAPhase,* uint8_t * *portBPhase*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
portAPhase	Pointer to a uint8_t type variable to receive the selected clock phase on PORTA.
portBPhase	Pointer to a uint8_t type variable to receive the selected clock phase on PORTB.

20.6.21 static flexspi_arb_command_source_t FLEXSPI_GetArbitrator-CommandSource (FLEXSPI_Type * *base*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.

Return values

trigger	source of current command sequence.

20.6.22 static flexspi_ip_error_code_t FLEXSPI_GetIPCommandErrorCode (FLEXSPI_Type * *base,* uint8_t * *index*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
index	Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error	code when IP command error detected.

20.6.23 static flexspi_ahb_error_code_t FLEXSPI_GetAHBCommandErrorCode (FLEXSPI_Type * *base*, uint8_t * *index*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
index	Pointer to a uint8_t type variable to receive the sequence index when error detected.

Return values

error	code when AHB command error detected.

20.6.24 static bool FLEXSPI_GetBusIdleStatus (FLEXSPI_Type * *base*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.

Return values

true	Bus is idle.
false	Bus is busy.

20.6.25 void FLEXSPI_UpdateRxSampleClock (FLEXSPI_Type * *base*, flexspi_read_sample_clock_t *clockSource*)

Parameters

base	FLEXSPI peripheral base address.
clockSource	clockSource of type flexspi_read_sample_clock_t

20.6.26 static void FLEXSPI_EnableIPParallelMode (FLEXSPI_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
enable	True means enable parallel mode, false means disable parallel mode.

20.6.27 static void FLEXSPI_EnableAHBParallelMode (FLEXSPI_Type ∗ *base,* bool *enable*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address.
enable	True means enable parallel mode, false means disable parallel mode.

20.6.28 void FLEXSPI_UpdateLUT (FLEXSPI_Type ∗ *base,* uint32_t *index,* const uint32_t ∗ *cmd,* uint32_t *count*)

Parameters

base	FLEXSPI peripheral base address.
index	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.
cmd	Command sequence array.
count	Number of sequences.

20.6.29 static void FLEXSPI_WriteData (FLEXSPI_Type ∗ *base,* uint32_t *data,* uint8_t *fifoIndex*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address
data	The data bytes to send
fifoIndex	Destination fifo index.

20.6.30 static uint32_t FLEXSPI_ReadData (FLEXSPI_Type * *base,* uint8_t *fifoIndex*) [inline], [static]

Parameters

base	FLEXSPI peripheral base address
fifoIndex	Source fifo index.

Returns

The data in the FIFO.

20.6.31 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

base	FLEXSPI peripheral base address
buffer	The data bytes to send
size	The number of data bytes to send

Return values

kStatus_Success	write success without error
kStatus_FLEXSPI_SequenceExecutionTimeout	sequence execution timeout
kStatus_FLEXSPI_IpCommandSequenceError	IP command sequence error detected
kStatus_FLEXSPI_IpCommandGrantTimeout	IP command grant timeout detected

20.6.32 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

base	FLEXSPI peripheral base address
buffer	The data bytes to send
size	The number of data bytes to receive

Return values

kStatus_Success	read success without error
kStatus_FLEXSPI_-SequenceExecution-Timeout	sequence execution timeout
kStatus_FLEXSPI_Ip-CommandSequenceError	IP command sequencen error detected
kStatus_FLEXSPI_Ip-CommandGrantTimeout	IP command grant timeout detected

20.6.33 status_t FLEXSPI_TransferBlocking (FLEXSPI_Type * *base*, flexspi_transfer_t * *xfer*)

Parameters

base	FLEXSPI peripheral base address
xfer	pointer to the transfer structure.

Return values

kStatus_Success	command transfer success without error
kStatus_FLEXSPI_-SequenceExecution-Timeout	sequence execution timeout
kStatus_FLEXSPI_Ip-CommandSequenceError	IP command sequence error detected
kStatus_FLEXSPI_Ip-CommandGrantTimeout	IP command grant timeout detected

20.6.34 void FLEXSPI_TransferCreateHandle (FLEXSPI_Type * *base,* flexspi_handle_t * *handle,* flexspi_transfer_callback_t *callback,* void * *userData*)

Parameters

base	FLEXSPI peripheral base address.
handle	pointer to flexspi_handle_t structure to store the transfer state.
callback	pointer to user callback function.
userData	user parameter passed to the callback function.

20.6.35 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

base	FLEXSPI peripheral base address.
handle	pointer to flexspi_handle_t structure which stores the transfer state.
xfer	pointer to flexspi_transfer_t structure.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_FLEXSPI_Busy	Previous transmission still not finished.

20.6.36 status_t FLEXSPI_TransferGetCount (FLEXSPI_Type * *base*, flexspi_handle_t * *handle*, size_t * *count*)

Parameters

base	FLEXSPI peripheral base address.

handle	pointer to flexspi_handle_t structure which stores the transfer state.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

20.6.37 void FLEXSPI_TransferAbort (FLEXSPI_Type * *base,* flexspi_handle_t * *handle*)

Note

This API can be called at any time when an interrupt non-blocking transfer initiates to abort the transfer early.

Parameters

base	FLEXSPI peripheral base address.
handle	pointer to flexspi_handle_t structure which stores the transfer state

20.6.38 void FLEXSPI_TransferHandleIRQ (FLEXSPI_Type * *base,* flexspi_handle_t * *handle*)

Parameters

base	FLEXSPI peripheral base address.
handle	pointer to flexspi_handle_t structure.

20.7 FLEXSPI DMA Driver

20.7.1 Overview

Data Structures

- struct flexspi_dma_handle_t
  FLEXSPI DMA transfer handle, users should not touch the content of the handle. More...

Typedefs

- typedef void(∗ flexspi_dma_callback_t)(FLEXSPI_Type ∗base, flexspi_dma_handle_t ∗handle, status_t status, void ∗userData)
  FLEXSPI dma transfer callback function for finish and error.

Enumerations

- enum flexspi_dma_transfer_nsize_t {
  kFLEXPSI_DMAnSize1Bytes = 0x1U,
  kFLEXPSI_DMAnSize2Bytes = 0x2U,
  kFLEXPSI_DMAnSize4Bytes = 0x4U }
  dma transfer configuration

Driver version

- #define FSL_FLEXSPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
  FLEXSPI DMA driver version 2.2.1.

FLEXSPI dma Transactional

- void FLEXSPI_TransferCreateHandleDMA (FLEXSPI_Type ∗base, flexspi_dma_handle_t ∗handle, flexspi_dma_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  Initializes the FLEXSPI handle for transfer which is used in transactional functions and set the callback.
- void FLEXSPI_TransferUpdateSizeDMA (FLEXSPI_Type ∗base, flexspi_dma_handle_t ∗handle, flexspi_dma_transfer_nsize_t nsize)
  Update FLEXSPI DMA transfer source data transfer size(SSIZE) and destination data transfer size(DSIZE).
- status_t FLEXSPI_TransferDMA (FLEXSPI_Type ∗base, flexspi_dma_handle_t ∗handle, flexspi_transfer_t ∗xfer)
  Transfers FLEXSPI data using an dma non-blocking method.
- void FLEXSPI_TransferAbortDMA (FLEXSPI_Type ∗base, flexspi_dma_handle_t ∗handle)
  Aborts the transfer data using dma.

- status_t FLEXSPI_TransferGetTransferCountDMA (FLEXSPI_Type ∗base, flexspi_dma_handle_t ∗handle, size_t ∗count)
  Gets the transferred counts of transfer.

20.7.2 Data Structure Documentation

20.7.2.1 struct _flexspi_dma_handle

Data Fields

- dma_handle_t ∗ txDmaHandle
  dma handler for FLEXSPI Tx.
- dma_handle_t ∗ rxDmaHandle
  dma handler for FLEXSPI Rx.
- size_t transferSize
  Bytes need to transfer.
- flexspi_dma_transfer_nsize_t nsize
  dma SSIZE/DSIZE in each transfer.
- uint8_t nbytes
  dma minor byte transfer count initially configured.
- uint8_t count
  The transfer data count in a DMA request.
- uint32_t state
  Internal state for FLEXSPI dma transfer.
- flexspi_dma_callback_t completionCallback
  A callback function called after the dma transfer is finished.
- void ∗ userData
  User callback parameter.

Field Documentation

(1) dma_handle_t∗ flexspi_dma_handle_t::txDmaHandle

(2) dma_handle_t∗ flexspi_dma_handle_t::rxDmaHandle

(3) size_t flexspi_dma_handle_t::transferSize

(4) flexspi_dma_transfer_nsize_t flexspi_dma_handle_t::nsize

(5) uint8_t flexspi_dma_handle_t::nbytes

(6) uint8_t flexspi_dma_handle_t::count

(7) uint32_t flexspi_dma_handle_t::state

(8) flexspi_dma_callback_t flexspi_dma_handle_t::completionCallback

20.7.3 Macro Definition Documentation

20.7.3.1 #define FSL_FLEXSPI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

20.7.4 Enumeration Type Documentation

20.7.4.1 enum flexspi_dma_transfer_nsize_t

Enumerator

kFLEXPSI_DMAnSize1Bytes	Source/Destination data transfer size is 1 byte every time.
kFLEXPSI_DMAnSize2Bytes	Source/Destination data transfer size is 2 bytes every time.
kFLEXPSI_DMAnSize4Bytes	Source/Destination data transfer size is 4 bytes every time.

20.7.5 Function Documentation

20.7.5.1 void FLEXSPI_TransferCreateHandleDMA (FLEXSPI_Type * *base*, flexspi_dma_handle_t * *handle*, flexspi_dma_callback_t *callback*, void * *userData*, dma_handle_t * *txDmaHandle*, dma_handle_t * *rxDmaHandle*)

Parameters

base	FLEXSPI peripheral base address
handle	Pointer to flexspi_dma_handle_t structure
callback	FLEXSPI callback, NULL means no callback.
userData	User callback function data.
txDmaHandle	User requested DMA handle for TX DMA transfer.
rxDmaHandle	User requested DMA handle for RX DMA transfer.

20.7.5.2 void FLEXSPI_TransferUpdateSizeDMA (FLEXSPI_Type * *base*, flexspi_dma_handle_t * *handle*, flexspi_dma_transfer_nsize_t *nsize*)

Parameters

base	FLEXSPI peripheral base address
handle	Pointer to flexspi_dma_handle_t structure

nsize	FLEXSPI DMA transfer data transfer size(SSIZE/DSIZE), by default the size is kFLEXPSI_DMAnSize1Bytes(one byte).

See Also

flexspi_dma_transfer_nsize_t .

20.7.5.3 status_t FLEXSPI_TransferDMA (FLEXSPI_Type * *base,* flexspi_dma_handle_t * *handle,* flexspi_transfer_t * *xfer*)

This function writes/receives data to/from the FLEXSPI transmit/receive FIFO. This function is non-blocking.

Parameters

base	FLEXSPI peripheral base address.
handle	Pointer to flexspi_dma_handle_t structure
xfer	FLEXSPI transfer structure.

Return values

kStatus_FLEXSPI_Busy	FLEXSPI is busy transfer.
kStatus_InvalidArgument	The watermark configuration is invalid, the watermark should be power of 2 to do successfully DMA transfer.
kStatus_Success	FLEXSPI successfully start dma transfer.

20.7.5.4 void FLEXSPI_TransferAbortDMA (FLEXSPI_Type * *base,* flexspi_dma_handle_t * *handle*)

This function aborts the transfer data using dma.

Parameters

base	FLEXSPI peripheral base address.
handle	Pointer to flexspi_dma_handle_t structure

20.7.5.5 status_t FLEXSPI_TransferGetTransferCountDMA (FLEXSPI_Type * *base,* flexspi_dma_handle_t * *handle,* size_t * *count*)

Parameters

base	FLEXSPI peripheral base address.
handle	Pointer to flexspi_dma_handle_t structure.
count	Bytes transfer.

Return values

kStatus_Success	Succeed get the transfer count.
kStatus_NoTransferIn-Progress	There is not a non-blocking transaction currently in progress.

Chapter 21
FMEAS: Frequency Measure Driver

21.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Frequency Measure function of MCUXpresso SDK devices' SYSCON module.

It measures frequency of any on-chip or off-chip clock signal. The more precise and higher accuracy clock is selected as a reference clock. The resulting frequency is internally computed from the ratio of value of selected target and reference clock counters.

21.2 Frequency Measure Driver operation

INPUTMUX_AttachSignal() function has to be used to select reference and target clock signal sources.

FMEAS_StartMeasure() function starts the measurement cycle.

FMEAS_IsMeasureComplete() can be polled to check if the measurement cycle has finished.

FMEAS_GetFrequency() returns the frequency of the target clock. Frequency of the reference clock has to be provided as a parameter.

21.3 Typical use case

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

Files

- file fsl_fmeas.h

Driver version

- #define FSL_FMEAS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  Defines LPC Frequency Measure driver version 2.1.1.

FMEAS Functional Operation

- static void FMEAS_StartMeasure (FMEAS_SYSCON_Type ∗base)
  Starts a frequency measurement cycle.
- static bool FMEAS_IsMeasureComplete (FMEAS_SYSCON_Type ∗base)
  Indicates when a frequency measurement cycle is complete.
- uint32_t FMEAS_GetFrequency (FMEAS_SYSCON_Type ∗base, uint32_t refClockRate)
  Returns the computed value for a frequency measurement cycle.

21.4 Macro Definition Documentation

21.4.1 #define FSL_FMEAS_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

21.5 Function Documentation

21.5.1 static void FMEAS_StartMeasure (FMEAS_SYSCON_Type * *base*) [inline], [static]

Parameters

base	: SYSCON peripheral base address.

Set the reference clock count cycle to 2^20 times

21.5.2 static bool FMEAS_IsMeasureComplete (FMEAS_SYSCON_Type * *base*) [inline], [static]

Parameters

base	: SYSCON peripheral base address.

Returns

true if a measurement cycle is active, otherwise false.

21.5.3 uint32_t FMEAS_GetFrequency (FMEAS_SYSCON_Type * *base,* uint32_t *refClockRate*)

Parameters

base	: SYSCON peripheral base address.
refClockRate	: Reference clock rate used during the frequency measurement cycle.

Returns

Frequency in Hz.

Chapter 22
Hashcrypt: The Cryptographic Accelerator

22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Hashcrypt peripheral. The Hashcrypt peripheral provides one or more engines to perform specific symmetric crypto algorithms, including hashing and en/decryption. The cryptographic acceleration is normally used in conjunction with pure-hardware blocks for hashing and symmetric cryptography, thereby providing performance and energy efficiency for a range of cryptographic uses.

Blocking synchronous APIs are provided for selected cryptographic algorithms using Hashcrypt hardware. The driver interface intends to be easily integrated with generic software crypto libraries such as mbed-TLS. The Hashcrypt operations are complete (and results are made availabe for further usage) when a function returns. When called, these functions do not return until an Hashcrypt operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and also for plaintext or ciphertext data movements. These functions provide typical interface to upper layer or application software. There is one non-blocking function provided for the purpose of background hashing. HASHCRYPT_SHA_UpdateNonBlocking() starts hashing of an input message while the CPU can continue executing.

22.2 Hashcrypt Driver Initialization and deinitialization

Hashcrypt Driver is initialized by calling the HASHCRYPT_Init() function, it enables clock and disables reset for Hashcrypt peripheral. Hashcrypt Driver is deinitialized by calling the HASHCRYPT_Deinit() function, it sisables clock and enables reset.

22.3 Comments about API usage in RTOS

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

22.4 Comments about API usage in interrupt handler

APIs can be used from interrupt handler although execution time shall be considered (interrupt latency increases considerably).

22.5 Hashcrypt Driver Examples

22.5.1 Simple examples

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

Modules

- Hashcrypt AES
- Hashcrypt Background HASH
- Hashcrypt HASH

22.6 Hashcrypt AES

22.6.1 Overview

Data Structures

- struct hashcrypt_handle_t
  Specify HASHCRYPT's key resource. More...

Macros

- #define HASHCRYPT_AES_BLOCK_SIZE 16U
  AES block size in bytes.

Enumerations

- enum hashcrypt_aes_mode_t {
  kHASHCRYPT_AesEcb = 0U,
  kHASHCRYPT_AesCbc = 1U,
  kHASHCRYPT_AesCtr = 2U }
  AES mode.
- enum hashcrypt_aes_keysize_t {
  kHASHCRYPT_Aes128 = 0U,
  kHASHCRYPT_Aes192 = 1U,
  kHASHCRYPT_Aes256 = 2U,
  kHASHCRYPT_InvalidKey = 3U }
  Size of AES key.
- enum hashcrypt_key_t {
  kHASHCRYPT_UserKey = 0xc3c3U,
  kHASHCRYPT_SecretKey = 0x3c3cU }
  HASHCRYPT key source selection.

Functions

- status_t HASHCRYPT_AES_SetKey (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *key, size_t keySize)
  Set AES key to hashcrypt_handle_t struct and optionally to HASHCRYPT.
- status_t HASHCRYPT_AES_EncryptEcb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size)
  Encrypts AES on one or multiple 128-bit block(s).
- status_t HASHCRYPT_AES_DecryptEcb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size)
  Decrypts AES on one or multiple 128-bit block(s).
- status_t HASHCRYPT_AES_EncryptCbc (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])

Encrypts AES using CBC block mode.

- status_t HASHCRYPT_AES_DecryptCbc (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])
  Decrypts AES using CBC block mode.
- status_t HASHCRYPT_AES_CryptCtr (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *input, uint8_t *output, size_t size, uint8_t counter[HASHCRYPT_AES_BLOCK_SIZE], uint8_t counterlast[HASHCRYPT_AES_BLOCK_SIZE], size_t *szLeft)
  Encrypts or decrypts AES using CTR block mode.
- status_t HASHCRYPT_AES_CryptOfb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *input, uint8_t *output, size_t size, const uint8_t iv[HASHCRYPT_AES_BLOCK_SIZE])
  Encrypts or decrypts AES using OFB block mode.
- status_t HASHCRYPT_AES_EncryptCfb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[16])
  Encrypts AES using CFB block mode.
- status_t HASHCRYPT_AES_DecryptCfb (HASHCRYPT_Type *base, hashcrypt_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[16])
  Decrypts AES using CFB block mode.

22.6.2 Data Structure Documentation

22.6.2.1 struct _hashcrypt_handle

Data Fields

- uint32_t keyWord [8]
  Copy of user key (set by HASHCRYPT_AES_SetKey().
- hashcrypt_key_t keyType
  For operations with key (such as AES encryption/decryption), specify key type.

Field Documentation

(1) uint32_t hashcrypt_handle_t::keyWord[8]

(2) hashcrypt_key_t hashcrypt_handle_t::keyType

22.6.3 Enumeration Type Documentation

22.6.3.1 enum hashcrypt_aes_mode_t

Enumerator

kHASHCRYPT_AesEcb AES ECB mode.
kHASHCRYPT_AesCbc AES CBC mode.
kHASHCRYPT_AesCtr AES CTR mode.

22.6.3.2 enum hashcrypt_aes_keysize_t

Enumerator

kHASHCRYPT_Aes128 AES 128 bit key.
kHASHCRYPT_Aes192 AES 192 bit key.
kHASHCRYPT_Aes256 AES 256 bit key.
kHASHCRYPT_InvalidKey AES invalid key.

22.6.3.3 enum hashcrypt_key_t

Enumerator

kHASHCRYPT_UserKey HASHCRYPT user key.
kHASHCRYPT_SecretKey HASHCRYPT secret key (dedicated hw bus from PUF)

22.6.4 Function Documentation

22.6.4.1 status_t HASHCRYPT_AES_SetKey (HASHCRYPT_Type ∗ *base,* hashcrypt_handle_t ∗ *handle,* const uint8_t ∗ *key,* size_t *keySize*)

Sets the AES key for encryption/decryption with the hashcrypt_handle_t structure. The hashcrypt_handle-_t input argument specifies key source.

Parameters

base	HASHCRYPT peripheral base address.
handle	Handle used for the request.
key	0-mod-4 aligned pointer to AES key.
keySize	AES key size in bytes. Shall equal 16, 24 or 32.

Returns

status from set key operation

22.6.4.2 status_t HASHCRYPT_AES_EncryptEcb (HASHCRYPT_Type ∗ *base,* hashcrypt_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* size_t *size*)

Encrypts AES. The source plaintext and destination ciphertext can overlap in system memory.

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	plaintext	Input plain text to encrypt
`out`	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

Status from encrypt operation

22.6.4.3 status_t HASHCRYPT_AES_DecryptEcb (HASHCRYPT_Type ∗ *base,* hashcrypt_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* size_t *size*)

Decrypts AES. The source ciphertext and destination plaintext can overlap in system memory.

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	ciphertext	Input plain text to encrypt
`out`	*plaintext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.

Returns

Status from decrypt operation

22.6.4.4 status_t HASHCRYPT_AES_EncryptCbc (HASHCRYPT_Type ∗ *base,* hashcrypt_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* size_t *size,* const uint8_t *iv[16]*)

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	plaintext	Input plain text to encrypt
`out`	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

22.6.4.5 status_t HASHCRYPT_AES_DecryptCbc (HASHCRYPT_Type ∗ *base,* hashcrypt_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* size_t *size,* const uint8_t *iv[16]*)

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	ciphertext	Input cipher text to decrypt
`out`	*plaintext*	Output plain text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.

Returns

Status from decrypt operation

22.6.4.6 status_t HASHCRYPT_AES_CryptCtr (HASHCRYPT_Type ∗ *base,* hashcrypt_handle_t ∗ *handle,* const uint8_t ∗ *input,* uint8_t ∗ *output,* size_t *size,* uint8_t *counter[HASHCRYPT_AES_BLOCK_SIZE],* uint8_t *counterlast[HASHCRYPT_AES_BLOCK_SIZE],* size_t ∗ *szLeft*)

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	input	Input data for CTR block mode
`out`	*output*	Output data for CTR block mode
	size	Size of input and output data in bytes
`in,out`	*counter*	Input counter (updates on return)
`out`	*counterlast*	Output cipher of last counter, for chained CTR calls (statefull encryption). NULL can be passed if chained calls are not used.
`out`	*szLeft*	Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

22.6.4.7 status_t HASHCRYPT_AES_CryptOfb (HASHCRYPT_Type * *base*, hashcrypt_handle_t * *handle*, const uint8_t * *input*, uint8_t * *output*, size_t *size*, const uint8_t *iv[HASHCRYPT_AES_BLOCK_SIZE]*)

Encrypts or decrypts AES using OFB block mode. AES OFB mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	input	Input data for OFB block mode
`out`	*output*	Output data for OFB block mode
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

22.6.4.8 status_t HASHCRYPT_AES_EncryptCfb (HASHCRYPT_Type * *base,* hashcrypt_handle_t * *handle,* const uint8_t * *plaintext,* uint8_t * *ciphertext,* size_t *size,* const uint8_t *iv[16]*)

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	plaintext	Input plain text to encrypt
`out`	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

22.6.4.9 status_t HASHCRYPT_AES_DecryptCfb (HASHCRYPT_Type ∗ *base,* hashcrypt_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* size_t *size,* const uint8_t *iv[16]*)

Parameters

	base	HASHCRYPT peripheral base address
	handle	Handle used for this request.
	ciphertext	Input cipher text to decrypt
`out`	*plaintext*	Output plaintext text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.

Returns

Status from encrypt operation

22.7 Hashcrypt HASH

22.7.1 Overview

Data Structures

- struct hashcrypt_hash_ctx_t
  Storage type used to save hash context. More...

Macros

- #define HASHCRYPT_HASH_CTX_SIZE 30
  HASHCRYPT HASH Context size.

Typedefs

- typedef void(∗ hashcrypt_callback_t)(HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, status_t status, void ∗userData)
  HASHCRYPT background hash callback function.

Functions

- status_t HASHCRYPT_SHA (HASHCRYPT_Type ∗base, hashcrypt_algo_t algo, const uint8_t ∗input, size_t inputSize, uint8_t ∗output, size_t ∗outputSize)
  Create HASH on given data.
- status_t HASHCRYPT_SHA_Init (HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, hashcrypt_algo_t algo)
  Initialize HASH context.
- status_t HASHCRYPT_SHA_Update (HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, const uint8_t ∗input, size_t inputSize)
  Add data to current HASH.
- status_t HASHCRYPT_SHA_Finish (HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, uint8_t ∗output, size_t ∗outputSize)
  Finalize hashing.

22.7.2 Data Structure Documentation

22.7.2.1 struct hashcrypt_hash_ctx_t

Data Fields

- uint32_t x [HASHCRYPT_HASH_CTX_SIZE]
  storage

22.7.3 Macro Definition Documentation

22.7.3.1 #define HASHCRYPT_HASH_CTX_SIZE 30

22.7.4 Typedef Documentation

22.7.4.1 typedef void(∗ hashcrypt_callback_t)(HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, status_t status, void ∗userData)

22.7.5 Function Documentation

22.7.5.1 status_t HASHCRYPT_SHA (HASHCRYPT_Type ∗ *base,* hashcrypt_algo_t *algo,* const uint8_t ∗ *input,* size_t *inputSize,* uint8_t ∗ *output,* size_t ∗ *outputSize*)

Perform the full SHA in one function call. The function is blocking.

Parameters

	base	HASHCRYPT peripheral base address
	algo	Underlaying algorithm to use for hash computation.
	input	Input data
	inputSize	Size of input data in bytes
`out`	*output*	Output hash data
`out`	*outputSize*	Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

22.7.5.2 status_t HASHCRYPT_SHA_Init (HASHCRYPT_Type ∗ *base,* hashcrypt_hash_ctx_t ∗ *ctx,* hashcrypt_algo_t *algo*)

This function initializes the HASH.

Parameters

	base	HASHCRYPT peripheral base address
out	*ctx*	Output hash context
	algo	Underlaying algorithm to use for hash computation.

Returns

Status of initialization

22.7.5.3 status_t HASHCRYPT_SHA_Update (HASHCRYPT_Type * *base*, hashcrypt_hash_ctx_t * *ctx*, const uint8_t * *input*, size_t *inputSize*)

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus_Success, the running hash has been updated (HASHCRYPT has processed the input data), so the memory at `input` pointer can be released back to system. The HASHCRYPT context buffer is updated with the running hash and with all necessary information to support possible context switch.

Parameters

	base	HASHCRYPT peripheral base address
in,out	*ctx*	HASH context
	input	Input data
	inputSize	Size of input data in bytes

Returns

Status of the hash update operation

22.7.5.4 status_t HASHCRYPT_SHA_Finish (HASHCRYPT_Type * *base*, hashcrypt_hash_ctx_t * *ctx*, uint8_t * *output*, size_t * *outputSize*)

Outputs the final hash (computed by HASHCRYPT_HASH_Update()) and erases the context.

Parameters

	base	HASHCRYPT peripheral base address
in,out	*ctx*	Input hash context
out	*output*	Output hash data
in,out	*outputSize*	Optional parameter (can be passed as NULL). On function entry, it specifies the size of output[] buffer. On function return, it stores the number of updated output bytes.

Returns

Status of the hash finish operation

22.8 Hashcrypt Background HASH

22.8.1 Overview

Functions

- void HASHCRYPT_SHA_SetCallback (HASHCRYPT_Type ∗base, hashcrypt_hash_ctx_t ∗ctx, hashcrypt_callback_t callback, void ∗userData)
  Initializes the HASHCRYPT handle for background hashing.
- status_t HASHCRYPT_SHA_UpdateNonBlocking (HASHCRYPT_Type ∗base, hashcrypt_hash_-ctx_t ∗ctx, const uint8_t ∗input, size_t inputSize)
  Create running hash on given data.

22.8.2 Function Documentation

22.8.2.1 void HASHCRYPT_SHA_SetCallback (HASHCRYPT_Type ∗ *base,* hashcrypt_hash_ctx_t ∗ *ctx,* hashcrypt_callback_t *callback,* void ∗ *userData*)

This function initializes the hash context for background hashing (Non-blocking) APIs. This is less typical interface to hash function, but can be used for parallel processing, when main CPU has something else to do. Example is digital signature RSASSA-PKCS1-V1_5-VERIFY((n,e),M,S) algorithm, where background hashing of M can be started, then CPU can compute S^e mod n (in parallel with background hashing) and once the digest becomes available, CPU can proceed to comparison of EM with EM'.

Parameters

	base	HASHCRYPT peripheral base address.
`out`	*ctx*	Hash context.
	callback	Callback function.
	userData	User data (to be passed as an argument to callback function, once callback is invoked from isr).

22.8.2.2 status_t HASHCRYPT_SHA_UpdateNonBlocking (HASHCRYPT_Type ∗ *base,* hashcrypt_hash_ctx_t ∗ *ctx,* const uint8_t ∗ *input,* size_t *inputSize*)

Configures the HASHCRYPT to compute new running hash as AHB master and returns immediately. HASHCRYPT AHB Master mode supports only aligned `input` address and can be called only once per continuous block of data. Every call to this function must be preceded with HASHCRYPT_SHA_Init() and finished with HASHCRYPT_SHA_Finish(). Once callback function is invoked by HASHCRYPT isr, it should set a flag for the main application to finalize the hashing (padding) and to read out the final digest by calling HASHCRYPT_SHA_Finish().

Parameters

base	HASHCRYPT peripheral base address
ctx	Specifies callback. Last incomplete 512-bit block of the input is copied into clear buffer for padding.
input	32-bit word aligned pointer to Input data.
inputSize	Size of input data in bytes (must be word aligned)

Returns

Status of the hash update operation.

Chapter 23
I3C: I3C Driver

23.1 Overview

Modules

- I3C FreeRTOS Driver
- I3C Master DMA Driver
- I3C Master Driver
- I3C Slave Driver

Data Structures

- struct i3c_device_info_t
  I3C device information. More...

Macros

- #define I3C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/
  Timeout times for waiting flag.

Enumerations

- enum {
  kStatus_I3C_Busy = MAKE_STATUS(kStatusGroup_I3C, 0),
  kStatus_I3C_Idle = MAKE_STATUS(kStatusGroup_I3C, 1),
  kStatus_I3C_Nak = MAKE_STATUS(kStatusGroup_I3C, 2),
  kStatus_I3C_WriteAbort,
  kStatus_I3C_Term = MAKE_STATUS(kStatusGroup_I3C, 4),
  kStatus_I3C_HdrParityError = MAKE_STATUS(kStatusGroup_I3C, 5),
  kStatus_I3C_CrcError = MAKE_STATUS(kStatusGroup_I3C, 6),
  kStatus_I3C_ReadFifoError = MAKE_STATUS(kStatusGroup_I3C, 7),
  kStatus_I3C_WriteFifoError = MAKE_STATUS(kStatusGroup_I3C, 8),
  kStatus_I3C_MsgError,
  kStatus_I3C_InvalidReq = MAKE_STATUS(kStatusGroup_I3C, 10),
  kStatus_I3C_Timeout = MAKE_STATUS(kStatusGroup_I3C, 11),
  kStatus_I3C_SlaveCountExceed,
  kStatus_I3C_IBIWon,
  kStatus_I3C_OverrunError = MAKE_STATUS(kStatusGroup_I3C, 14),
  kStatus_I3C_UnderrunError = MAKE_STATUS(kStatusGroup_I3C, 15),
  kStatus_I3C_UnderrunNak,
  kStatus_I3C_InvalidStart = MAKE_STATUS(kStatusGroup_I3C, 17),
  kStatus_I3C_SdrParityError = MAKE_STATUS(kStatusGroup_I3C, 18),

kStatus_I3C_S0S1Error = MAKE_STATUS(kStatusGroup_I3C, 19) }
I3C status return codes.
- enum i3c_hdr_mode_t
  I3C HDR modes.

Driver version

- #define FSL_I3C_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))
  I3C driver version.

23.2 Data Structure Documentation

23.2.1 struct i3c_device_info_t

Data Fields

- uint8_t dynamicAddr
  Device dynamic address.
- uint8_t staticAddr
  Static address.
- uint8_t dcr
  Device characteristics register information.
- uint8_t bcr
  Bus characteristics register information.
- uint16_t vendorID
  Device vendor ID(manufacture ID).
- uint32_t partNumber
  Device part number info.
- uint16_t maxReadLength
  Maximum read length.
- uint16_t maxWriteLength
  Maximum write length.
- uint8_t hdrMode
  Support hdr mode, could be OR logic in i3c_hdr_mode.

Field Documentation

(1) uint8_t i3c_device_info_t::dynamicAddr

(2) uint8_t i3c_device_info_t::staticAddr

(3) uint8_t i3c_device_info_t::dcr

(4) uint8_t i3c_device_info_t::bcr

(5) uint16_t i3c_device_info_t::vendorID

(6) uint16_t i3c_device_info_t::maxReadLength

(7) uint16_t i3c_device_info_t::maxWriteLength

(8) uint8_t i3c_device_info_t::hdrMode

23.3 Macro Definition Documentation

23.3.1 #define I3C_RETRY_TIMES 0U /∗ Define to zero means keep waiting until the flag is assert/deassert. ∗/

23.4 Enumeration Type Documentation

23.4.1 anonymous enum

Enumerator

kStatus_I3C_Busy The master is already performing a transfer.
kStatus_I3C_Idle The slave driver is idle.
kStatus_I3C_Nak The slave device sent a NAK in response to an address.
kStatus_I3C_WriteAbort The slave device sent a NAK in response to a write.
kStatus_I3C_Term The master terminates slave read.
kStatus_I3C_HdrParityError Parity error from DDR read.
kStatus_I3C_CrcError CRC error from DDR read.
kStatus_I3C_ReadFifoError Read from M/SRDATAB register when FIFO empty.
kStatus_I3C_WriteFifoError Write to M/SWDATAB register when FIFO full.
kStatus_I3C_MsgError Message SDR/DDR mismatch or read/write message in wrong state.
kStatus_I3C_InvalidReq Invalid use of request.
kStatus_I3C_Timeout The module has stalled too long in a frame.
kStatus_I3C_SlaveCountExceed The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.
kStatus_I3C_IBIWon The I3C slave event IBI or MR or HJ won the arbitration on a header address.

kStatus_I3C_OverrunError Slave internal from-bus buffer/FIFO overrun.
kStatus_I3C_UnderrunError Slave internal to-bus buffer/FIFO underrun.
kStatus_I3C_UnderrunNak Slave internal from-bus buffer/FIFO underrun and NACK error.
kStatus_I3C_InvalidStart Slave invalid start flag.
kStatus_I3C_SdrParityError SDR parity error.
kStatus_I3C_S0S1Error S0 or S1 error.

23.4.2 enum i3c_hdr_mode_t

23.5 I3C Master Driver

23.5.1 Overview

Data Structures

- struct i3c_register_ibi_addr_t
  Structure with setting master IBI rules and slave registry. More...
- struct i3c_baudrate_hz_t
  Structure with I3C baudrate settings. More...
- struct i3c_master_config_t
  Structure with settings to initialize the I3C master module. More...
- struct i3c_master_transfer_callback_t
  i3c master callback functions. More...
- struct i3c_master_transfer_t
  Non-blocking transfer descriptor structure. More...
- struct i3c_master_handle_t
  Driver handle for master non-blocking APIs. More...

Typedefs

- typedef void(∗ i3c_master_isr_t)(I3C_Type ∗base, i3c_master_handle_t ∗handle)
  Typedef for master interrupt handler.

Enumerations

- enum _i3c_master_flags {
  kI3C_MasterBetweenFlag = I3C_MSTATUS_BETWEEN_MASK,
  kI3C_MasterNackDetectFlag = I3C_MSTATUS_NACKED_MASK,
  kI3C_MasterSlaveStartFlag = I3C_MSTATUS_SLVSTART_MASK,
  kI3C_MasterControlDoneFlag = I3C_MSTATUS_MCTRLDONE_MASK,
  kI3C_MasterCompleteFlag = I3C_MSTATUS_COMPLETE_MASK,
  kI3C_MasterRxReadyFlag = I3C_MSTATUS_RXPEND_MASK,
  kI3C_MasterTxReadyFlag = I3C_MSTATUS_TXNOTFULL_MASK,
  kI3C_MasterArbitrationWonFlag = I3C_MSTATUS_IBIWON_MASK,
  kI3C_MasterErrorFlag = I3C_MSTATUS_ERRWARN_MASK,
  kI3C_MasterSlave2MasterFlag = I3C_MSTATUS_NOWMASTER_MASK }
  I3C master peripheral flags.
- enum _i3c_master_error_flags {

kI3C_MasterErrorNackFlag = I3C_MERRWARN_NACK_MASK,
kI3C_MasterErrorWriteAbortFlag = I3C_MERRWARN_WRABT_MASK,
kI3C_MasterErrorTermFlag = I3C_MERRWARN_TERM_MASK,
kI3C_MasterErrorParityFlag = I3C_MERRWARN_HPAR_MASK,
kI3C_MasterErrorCrcFlag = I3C_MERRWARN_HCRC_MASK,
kI3C_MasterErrorReadFlag = I3C_MERRWARN_OREAD_MASK,
kI3C_MasterErrorWriteFlag = I3C_MERRWARN_OWRITE_MASK,
kI3C_MasterErrorMsgFlag = I3C_MERRWARN_MSGERR_MASK,
kI3C_MasterErrorInvalidReqFlag = I3C_MERRWARN_INVREQ_MASK,
kI3C_MasterErrorTimeoutFlag = I3C_MERRWARN_TIMEOUT_MASK,
kI3C_MasterAllErrorFlags }

I3C master error flags to indicate the causes.

- enum i3c_master_state_t {
  kI3C_MasterStateIdle = 0U,
  kI3C_MasterStateSlvReq = 1U,
  kI3C_MasterStateMsgSdr = 2U,
  kI3C_MasterStateNormAct = 3U,
  kI3C_MasterStateDdr = 4U,
  kI3C_MasterStateDaa = 5U,
  kI3C_MasterStateIbiAck = 6U,
  kI3C_MasterStateIbiRcv = 7U }

  I3C working master state.
- enum i3c_master_enable_t {
  kI3C_MasterOff = 0U,
  kI3C_MasterOn = 1U,
  kI3C_MasterCapable = 2U }

  I3C master enable configuration.
- enum i3c_master_hkeep_t {
  kI3C_MasterHighKeeperNone = 0U,
  kI3C_MasterHighKeeperWiredIn = 1U,
  kI3C_MasterPassiveSDA = 2U,
  kI3C_MasterPassiveSDASCL = 3U }

  I3C high keeper configuration.
- enum i3c_bus_request_t {
  kI3C_RequestNone = 0U,
  kI3C_RequestEmitStartAddr = 1U,
  kI3C_RequestEmitStop = 2U,
  kI3C_RequestIbiAckNack = 3U,
  kI3C_RequestProcessDAA = 4U,
  kI3C_RequestForceExit = 6U,
  kI3C_RequestAutoIbi = 7U }

  Emits the requested operation when doing in pieces vs.
- enum i3c_bus_type_t {
  kI3C_TypeI3CSdr = 0U,
  kI3C_TypeI2C = 1U,
  kI3C_TypeI3CDdr = 2U }

Bus type with EmitStartAddr.

- enum i3c_ibi_response_t {
  kI3C_IbiRespAck = 0U,
  kI3C_IbiRespNack = 1U,
  kI3C_IbiRespAckMandatory = 2U,
  kI3C_IbiRespManual = 3U }
  IBI response.
- enum i3c_ibi_type_t {
  kI3C_IbiNormal = 0U,
  kI3C_IbiHotJoin = 1U,
  kI3C_IbiMasterRequest = 2U }
  IBI type.
- enum i3c_ibi_state_t {
  kI3C_IbiReady = 0U,
  kI3C_IbiDataBuffNeed = 1U,
  kI3C_IbiAckNackPending = 2U }
  IBI state.
- enum i3c_direction_t {
  kI3C_Write = 0U,
  kI3C_Read = 1U }
  Direction of master and slave transfers.
- enum i3c_tx_trigger_level_t {
  kI3C_TxTriggerOnEmpty = 0U,
  kI3C_TxTriggerUntilOneQuarterOrLess = 1U,
  kI3C_TxTriggerUntilOneHalfOrLess = 2U,
  kI3C_TxTriggerUntilOneLessThanFull = 3U }
  Watermark of TX int/dma trigger level.
- enum i3c_rx_trigger_level_t {
  kI3C_RxTriggerOnNotEmpty = 0U,
  kI3C_RxTriggerUntilOneQuarterOrMore = 1U,
  kI3C_RxTriggerUntilOneHalfOrMore = 2U,
  kI3C_RxTriggerUntilThreeQuarterOrMore = 3U }
  Watermark of RX int/dma trigger level.
- enum _i3c_master_transfer_flags {
  kI3C_TransferDefaultFlag = 0x00U,
  kI3C_TransferNoStartFlag = 0x01U,
  kI3C_TransferRepeatedStartFlag = 0x02U,
  kI3C_TransferNoStopFlag = 0x04U,
  kI3C_TransferWordsFlag = 0x08U }
  Transfer option flags.

Initialization and deinitialization

- void I3C_MasterGetDefaultConfig (i3c_master_config_t *masterConfig)
  Provides a default configuration for the I3C master peripheral.
- void I3C_MasterInit (I3C_Type *base, const i3c_master_config_t *masterConfig, uint32_t source-

Clock_Hz)
Initializes the I3C master peripheral.
- void I3C_MasterDeinit (I3C_Type *base)
Deinitializes the I3C master peripheral.
- status_t **I3C_MasterCheckAndClearError** (I3C_Type *base, uint32_t status)
- status_t **I3C_CheckForBusyBus** (I3C_Type *base)
- static void I3C_MasterEnable (I3C_Type *base, i3c_master_enable_t enable)
Set I3C module master mode.
- void I3C_SlaveGetDefaultConfig (i3c_slave_config_t *slaveConfig)
Provides a default configuration for the I3C slave peripheral.
- void I3C_SlaveInit (I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_-Hz)
Initializes the I3C slave peripheral.
- void I3C_SlaveDeinit (I3C_Type *base)
Deinitializes the I3C slave peripheral.
- static void I3C_SlaveEnable (I3C_Type *base, bool isEnable)
Enable/Disable Slave.

Status

- static uint32_t I3C_MasterGetStatusFlags (I3C_Type *base)
Gets the I3C master status flags.
- static void I3C_MasterClearStatusFlags (I3C_Type *base, uint32_t statusMask)
Clears the I3C master status flag state.
- static uint32_t I3C_MasterGetErrorStatusFlags (I3C_Type *base)
Gets the I3C master error status flags.
- static void I3C_MasterClearErrorStatusFlags (I3C_Type *base, uint32_t statusMask)
Clears the I3C master error status flag state.
- i3c_master_state_t I3C_MasterGetState (I3C_Type *base)
Gets the I3C master state.
- static uint32_t I3C_SlaveGetStatusFlags (I3C_Type *base)
Gets the I3C slave status flags.
- static void I3C_SlaveClearStatusFlags (I3C_Type *base, uint32_t statusMask)
Clears the I3C slave status flag state.
- static uint32_t I3C_SlaveGetErrorStatusFlags (I3C_Type *base)
Gets the I3C slave error status flags.
- static void I3C_SlaveClearErrorStatusFlags (I3C_Type *base, uint32_t statusMask)
Clears the I3C slave error status flag state.
- i3c_slave_activity_state_t I3C_SlaveGetActivityState (I3C_Type *base)
Gets the I3C slave state.

Interrupts

- static void I3C_MasterEnableInterrupts (I3C_Type *base, uint32_t interruptMask)
Enables the I3C master interrupt requests.
- static void I3C_MasterDisableInterrupts (I3C_Type *base, uint32_t interruptMask)
Disables the I3C master interrupt requests.
- static uint32_t I3C_MasterGetEnabledInterrupts (I3C_Type *base)
Returns the set of currently enabled I3C master interrupt requests.

- static uint32_t I3C_MasterGetPendingInterrupts (I3C_Type *base)
  Returns the set of pending I3C master interrupt requests.
- static void I3C_SlaveEnableInterrupts (I3C_Type *base, uint32_t interruptMask)
  Enables the I3C slave interrupt requests.
- static void I3C_SlaveDisableInterrupts (I3C_Type *base, uint32_t interruptMask)
  Disables the I3C slave interrupt requests.
- static uint32_t I3C_SlaveGetEnabledInterrupts (I3C_Type *base)
  Returns the set of currently enabled I3C slave interrupt requests.
- static uint32_t I3C_SlaveGetPendingInterrupts (I3C_Type *base)
  Returns the set of pending I3C slave interrupt requests.

DMA control

- static void I3C_MasterEnableDMA (I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
  Enables or disables I3C master DMA requests.
- static uint32_t I3C_MasterGetTxFifoAddress (I3C_Type *base, uint32_t width)
  Gets I3C master transmit data register address for DMA transfer.
- static uint32_t I3C_MasterGetRxFifoAddress (I3C_Type *base, uint32_t width)
  Gets I3C master receive data register address for DMA transfer.
- static void I3C_SlaveEnableDMA (I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
  Enables or disables I3C slave DMA requests.
- static uint32_t I3C_SlaveGetTxFifoAddress (I3C_Type *base, uint32_t width)
  Gets I3C slave transmit data register address for DMA transfer.
- static uint32_t I3C_SlaveGetRxFifoAddress (I3C_Type *base, uint32_t width)
  Gets I3C slave receive data register address for DMA transfer.

FIFO control

- static void I3C_MasterSetWatermarks (I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
  Sets the watermarks for I3C master FIFOs.
- static void I3C_MasterGetFifoCounts (I3C_Type *base, size_t *rxCount, size_t *txCount)
  Gets the current number of bytes in the I3C master FIFOs.
- static void I3C_SlaveSetWatermarks (I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger_level_t rxLvl, bool flushTx, bool flushRx)
  Sets the watermarks for I3C slave FIFOs.
- static void I3C_SlaveGetFifoCounts (I3C_Type *base, size_t *rxCount, size_t *txCount)
  Gets the current number of bytes in the I3C slave FIFOs.

Bus operations

- void I3C_MasterSetBaudRate (I3C_Type *base, const i3c_baudrate_hz_t *baudRate_Hz, uint32_t sourceClock_Hz)
  Sets the I3C bus frequency for master transactions.
- static bool I3C_MasterGetBusIdleState (I3C_Type *base)

Returns whether the bus is idle.

- status_t I3C_MasterStart (I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir)
  Sends a START signal and slave address on the I2C/I3C bus.
- status_t I3C_MasterRepeatedStart (I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir)
  Sends a repeated START signal and slave address on the I2C/I3C bus.
- status_t I3C_MasterRepeatedStartWithRxSize (I3C_Type *base, i3c_bus_type_t type, uint8_t address, i3c_direction_t dir, uint8_t rxSize)
  Sends a repeated START signal and slave address on the I2C/I3C bus, receive size is also specified in the call.
- status_t I3C_MasterSend (I3C_Type *base, const void *txBuff, size_t txSize, uint32_t flags)
  Performs a polling send transfer on the I2C/I3C bus.
- status_t I3C_MasterReceive (I3C_Type *base, void *rxBuff, size_t rxSize, uint32_t flags)
  Performs a polling receive transfer on the I2C/I3C bus.
- status_t I3C_MasterStop (I3C_Type *base)
  Sends a STOP signal on the I2C/I3C bus.
- void I3C_MasterEmitRequest (I3C_Type *base, i3c_bus_request_t masterReq)
  I3C master emit request.
- static void I3C_MasterEmitIBIResponse (I3C_Type *base, i3c_ibi_response_t ibiResponse)
  I3C master emit request.
- void I3C_MasterRegisterIBI (I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)
  I3C master register IBI rule.
- void I3C_MasterGetIBIRules (I3C_Type *base, i3c_register_ibi_addr_t *ibiRule)
  I3C master get IBI rule.
- status_t I3C_MasterProcessDAA (I3C_Type *base, uint8_t *addressList, uint32_t count)
  Performs a DAA in the i3c bus.
- status_t I3C_MasterTransferBlocking (I3C_Type *base, i3c_master_transfer_t *transfer)
  Performs a master polling transfer on the I2C/I3C bus.
- void I3C_SlaveRequestEvent (I3C_Type *base, i3c_slave_event_t event)
  I3C slave request event.
- status_t I3C_SlaveSend (I3C_Type *base, const void *txBuff, size_t txSize)
  Performs a polling send transfer on the I3C bus.
- status_t I3C_SlaveReceive (I3C_Type *base, void *rxBuff, size_t rxSize)
  Performs a polling receive transfer on the I3C bus.

Non-blocking

- void I3C_MasterTransferCreateHandle (I3C_Type *base, i3c_master_handle_t *handle, const i3c_master_transfer_callback_t *callback, void *userData)
  Creates a new handle for the I3C master non-blocking APIs.
- status_t I3C_MasterTransferNonBlocking (I3C_Type *base, i3c_master_handle_t *handle, i3c_master_transfer_t *transfer)
  Performs a non-blocking transaction on the I2C/I3C bus.
- status_t I3C_MasterTransferGetCount (I3C_Type *base, i3c_master_handle_t *handle, size_t *count)
  Returns number of bytes transferred so far.
- void I3C_MasterTransferAbort (I3C_Type *base, i3c_master_handle_t *handle)
  Terminates a non-blocking I3C master transmission early.

IRQ handler

- void I3C_MasterTransferHandleIRQ (I3C_Type *base, i3c_master_handle_t *handle)
  Reusable routine to handle master interrupts.

23.5.2 Data Structure Documentation

23.5.2.1 struct i3c_register_ibi_addr_t

Data Fields

- uint8_t address [5]
  Address array for registry.
- bool ibiHasPayload
  Whether the address array has mandatory IBI byte.

Field Documentation

(1) uint8_t i3c_register_ibi_addr_t::address[5]

(2) bool i3c_register_ibi_addr_t::ibiHasPayload

23.5.2.2 struct i3c_baudrate_hz_t

Data Fields

- uint32_t i2cBaud
  Desired I2C baud rate in Hertz.
- uint32_t i3cPushPullBaud
  Desired I3C push-pull baud rate in Hertz.
- uint32_t i3cOpenDrainBaud
  Desired I3C open-drain baud rate in Hertz.

Field Documentation

(1) uint32_t i3c_baudrate_hz_t::i2cBaud

(2) uint32_t i3c_baudrate_hz_t::i3cPushPullBaud

(3) uint32_t i3c_baudrate_hz_t::i3cOpenDrainBaud

23.5.2.3 struct i3c_master_config_t

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_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

- i3c_master_enable_t enableMaster
  Enable master mode.
- bool disableTimeout
  Whether to disable timeout to prevent the ERRWARN.
- i3c_master_hkeep_t hKeep
  High keeper mode setting.
- bool enableOpenDrainStop
  Whether to emit open-drain speed STOP.
- bool enableOpenDrainHigh
  Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.
- i3c_baudrate_hz_t baudRate_Hz
  Desired baud rate settings.

Field Documentation

(1) i3c_master_enable_t i3c_master_config_t::enableMaster

(2) bool i3c_master_config_t::disableTimeout

(3) i3c_master_hkeep_t i3c_master_config_t::hKeep

(4) bool i3c_master_config_t::enableOpenDrainStop

(5) bool i3c_master_config_t::enableOpenDrainHigh

(6) i3c_baudrate_hz_t i3c_master_config_t::baudRate_Hz

23.5.2.4 struct i3c_master_transfer_callback_t

Data Fields

- void(∗ slave2Master)(I3C_Type ∗base, void ∗userData)
  Transfer complete callback.
- void(∗ ibiCallback)(I3C_Type ∗base, i3c_master_handle_t ∗handle, i3c_ibi_type_t ibiType, i3c_-ibi_state_t ibiState)
  IBI event callback.
- void(∗ transferComplete)(I3C_Type ∗base, i3c_master_handle_t ∗handle, status_t completion-Status, void ∗userData)
  Transfer complete callback.

23.5.2.5 struct _i3c_master_transfer

This structure is used to pass transaction parameters to the I3C_MasterTransferNonBlocking() API.

Data Fields

- uint32_t flags

Bit mask of options for the transfer.

- uint8_t slaveAddress
  The 7-bit slave address.
- i3c_direction_t direction
  Either kI3C_Read or kI3C_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.
- i3c_bus_type_t busType
  bus type.
- i3c_ibi_response_t ibiResponse
  ibi response during transfer.

Field Documentation

(1) uint32_t i3c_master_transfer_t::flags

See enumeration _i3c_master_transfer_flags for available options. Set to 0 or kI3C_TransferDefaultFlag for normal transfers.

(2) uint8_t i3c_master_transfer_t::slaveAddress

(3) i3c_direction_t i3c_master_transfer_t::direction

(4) uint32_t i3c_master_transfer_t::subaddress

Transferred MSB first.

(5) size_t i3c_master_transfer_t::subaddressSize

Maximum size is 4 bytes.

(6) void* i3c_master_transfer_t::data

(7) size_t i3c_master_transfer_t::dataSize

(8) i3c_bus_type_t i3c_master_transfer_t::busType

(9) i3c_ibi_response_t i3c_master_transfer_t::ibiResponse

23.5.2.6 struct _i3c_master_handle

Note

The contents of this structure are private and subject to change.

Data Fields

- uint8_t state
  Transfer state machine current state.
- uint32_t remainingBytes
  Remaining byte count in current state.
- bool isReadTerm
  Is readterm configured.
- i3c_master_transfer_t transfer
  Copy of the current transfer info.
- uint8_t ibiAddress
  Slave address which request IBI.
- uint8_t * ibiBuff
  Pointer to IBI buffer to keep ibi bytes.
- size_t ibiPayloadSize
  IBI payload size.
- i3c_ibi_type_t ibiType
  IBI type.
- i3c_master_transfer_callback_t callback
  Callback functions pointer.
- void * userData
  Application data passed to callback.

Field Documentation

(1) uint8_t i3c_master_handle_t::state

(2) uint32_t i3c_master_handle_t::remainingBytes

(3) bool i3c_master_handle_t::isReadTerm

(4) i3c_master_transfer_t i3c_master_handle_t::transfer

(5) uint8_t i3c_master_handle_t::ibiAddress

(6) uint8_t* i3c_master_handle_t::ibiBuff

(7) size_t i3c_master_handle_t::ibiPayloadSize

(8) i3c_ibi_type_t i3c_master_handle_t::ibiType

(9) i3c_master_transfer_callback_t i3c_master_handle_t::callback

(10) void* i3c_master_handle_t::userData

23.5.3 Typedef Documentation

23.5.3.1 typedef void(∗ i3c_master_isr_t)(I3C_Type ∗base, i3c_master_handle_t ∗handle)

23.5.4 Enumeration Type Documentation

23.5.4.1 enum _i3c_master_flags

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag
- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI3C_MasterBetweenFlag Between messages/DAAs flag.
kI3C_MasterNackDetectFlag NACK detected flag.
kI3C_MasterSlaveStartFlag Slave request start flag.
kI3C_MasterControlDoneFlag Master request complete flag.
kI3C_MasterCompleteFlag Transfer complete flag.
kI3C_MasterRxReadyFlag Rx data ready in Rx buffer flag.
kI3C_MasterTxReadyFlag Tx buffer ready for Tx data flag.
kI3C_MasterArbitrationWonFlag Header address won arbitration flag.
kI3C_MasterErrorFlag Error occurred flag.
kI3C_MasterSlave2MasterFlag Switch from slave to master flag.

23.5.4.2 enum _i3c_master_error_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI3C_MasterErrorNackFlag Slave NACKed the last address.
kI3C_MasterErrorWriteAbortFlag Slave NACKed the write data.
kI3C_MasterErrorTermFlag Master terminates slave read.
kI3C_MasterErrorParityFlag Parity error from DDR read.
kI3C_MasterErrorCrcFlag CRC error from DDR read.

kI3C_MasterErrorReadFlag Read from MRDATAB register when FIFO empty.
kI3C_MasterErrorWriteFlag Write to MWDATAB register when FIFO full.
kI3C_MasterErrorMsgFlag Message SDR/DDR mismatch or read/write message in wrong state.
kI3C_MasterErrorInvalidReqFlag Invalid use of request.
kI3C_MasterErrorTimeoutFlag The module has stalled too long in a frame.
kI3C_MasterAllErrorFlags All error flags.

23.5.4.3 enum i3c_master_state_t

Enumerator

kI3C_MasterStateIdle Bus stopped.
kI3C_MasterStateSlvReq Bus stopped but slave holding SDA low.
kI3C_MasterStateMsgSdr In SDR Message mode from using MWMSG_SDR.
kI3C_MasterStateNormAct In normal active SDR mode.
kI3C_MasterStateDdr In DDR Message mode.
kI3C_MasterStateDaa In ENTDAA mode.
kI3C_MasterStateIbiAck Waiting on IBI ACK/NACK decision.
kI3C_MasterStateIbiRcv receiving IBI.

23.5.4.4 enum i3c_master_enable_t

Enumerator

kI3C_MasterOff Master off.
kI3C_MasterOn Master on.
kI3C_MasterCapable Master capable.

23.5.4.5 enum i3c_master_hkeep_t

Enumerator

kI3C_MasterHighKeeperNone Use PUR to hold SCL high.
kI3C_MasterHighKeeperWiredIn Use pin_HK controls.
kI3C_MasterPassiveSDA Hi-Z for Bus Free and hold SDA.
kI3C_MasterPassiveSDASCL Hi-Z both for Bus Free, and can Hi-Z SDA for hold.

23.5.4.6 enum i3c_bus_request_t

by message.

Enumerator

kI3C_RequestNone No request.
kI3C_RequestEmitStartAddr Request to emit start and address on bus.
kI3C_RequestEmitStop Request to emit stop on bus.
kI3C_RequestIbiAckNack Manual IBI ACK or NACK.
kI3C_RequestProcessDAA Process DAA.
kI3C_RequestForceExit Request to force exit.
kI3C_RequestAutoIbi Hold in stopped state, but Auto-emit START,7E.

23.5.4.7 enum i3c_bus_type_t

Enumerator

kI3C_TypeI3CSdr SDR mode of I3C.
kI3C_TypeI2C Standard i2c protocol.
kI3C_TypeI3CDdr HDR-DDR mode of I3C.

23.5.4.8 enum i3c_ibi_response_t

Enumerator

kI3C_IbiRespAck ACK with no mandatory byte.
kI3C_IbiRespNack NACK.
kI3C_IbiRespAckMandatory ACK with mandatory byte.
kI3C_IbiRespManual Reserved.

23.5.4.9 enum i3c_ibi_type_t

Enumerator

kI3C_IbiNormal In-band interrupt.
kI3C_IbiHotJoin slave hot join.
kI3C_IbiMasterRequest slave master ship request.

23.5.4.10 enum i3c_ibi_state_t

Enumerator

kI3C_IbiReady In-band interrupt ready state, ready for user to handle.
kI3C_IbiDataBuffNeed In-band interrupt need data buffer for data receive.
kI3C_IbiAckNackPending In-band interrupt Ack/Nack pending for decision.

23.5.4.11 enum i3c_direction_t

Enumerator

kI3C_Write Master transmit.
kI3C_Read Master receive.

23.5.4.12 enum i3c_tx_trigger_level_t

Enumerator

kI3C_TxTriggerOnEmpty Trigger on empty.
kI3C_TxTriggerUntilOneQuarterOrLess Trigger on 1/4 full or less.
kI3C_TxTriggerUntilOneHalfOrLess Trigger on 1/2 full or less.
kI3C_TxTriggerUntilOneLessThanFull Trigger on 1 less than full or less.

23.5.4.13 enum i3c_rx_trigger_level_t

Enumerator

kI3C_RxTriggerOnNotEmpty Trigger on not empty.
kI3C_RxTriggerUntilOneQuarterOrMore Trigger on 1/4 full or more.
kI3C_RxTriggerUntilOneHalfOrMore Trigger on 1/2 full or more.
kI3C_RxTriggerUntilThreeQuarterOrMore Trigger on 3/4 full or more.

23.5.4.14 enum _i3c_master_transfer_flags

Note

These enumerations are intended to be OR'd together to form a bit mask of options for the _i3c_-master_transfer::flags field.

Enumerator

kI3C_TransferDefaultFlag Transfer starts with a start signal, stops with a stop signal.
kI3C_TransferNoStartFlag Don't send a start condition, address, and sub address.
kI3C_TransferRepeatedStartFlag Send a repeated start condition.
kI3C_TransferNoStopFlag Don't send a stop condition.
kI3C_TransferWordsFlag Transfer in words, else transfer in bytes.

23.5.5 Function Documentation

23.5.5.1 void I3C_MasterGetDefaultConfig (i3c_master_config_t * *masterConfig*)

This function provides the following default configuration for the I3C master peripheral:

```
*  masterConfig->enableMaster            = kI3C_MasterOn;
*  masterConfig->disableTimeout          = false;
*  masterConfig->hKeep                   = kI3C_MasterHighKeeperNone;
*  masterConfig->enableOpenDrainStop     = true;
*  masterConfig->enableOpenDrainHigh     = true;
*  masterConfig->baudRate_Hz             = 100000U;
*  masterConfig->busType                 = kI3C_TypeI2C;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with I3C_MasterInit().

Parameters

out	*masterConfig*	User provided configuration structure for default values. Refer to i3c_master_config_t.

23.5.5.2 void I3C_MasterInit (I3C_Type * *base*, const i3c_master_config_t * *masterConfig*, uint32_t *sourceClock_Hz*)

This function enables the peripheral clock and initializes the I3C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

base	The I3C peripheral base address.
masterConfig	User provided peripheral configuration. Use I3C_MasterGetDefaultConfig() to get a set of defaults that you can override.
sourceClock_Hz	Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

23.5.5.3 void I3C_MasterDeinit (I3C_Type * *base*)

This function disables the I3C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

base	The I3C peripheral base address.

23.5.5.4 static void I3C_MasterEnable (I3C_Type * *base*, i3c_master_enable_t *enable*) [inline], [static]

Parameters

base	The I3C peripheral base address.
enable	Enable master mode.

23.5.5.5 static uint32_t I3C_MasterGetStatusFlags (I3C_Type * *base*) [inline], [static]

A bit mask with the state of all I3C master status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I3C peripheral base address.

Returns

State of the status flags:
- 1: related status flag is set.
- 0: related status flag is not set.

See Also

_i3c_master_flags

23.5.5.6 static void I3C_MasterClearStatusFlags (I3C_Type * *base*, uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared:

- kI3C_MasterSlaveStartFlag
- kI3C_MasterControlDoneFlag
- kI3C_MasterCompleteFlag
- kI3C_MasterArbitrationWonFlag
- kI3C_MasterSlave2MasterFlag

Attempts to clear other flags has no effect.

Parameters

base	The I3C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i3c_-master_flags enumerators OR'd together. You may pass the result of a previous call to I3C_MasterGetStatusFlags().

See Also

_i3c_master_flags.

23.5.5.7 static uint32_t I3C_MasterGetErrorStatusFlags (I3C_Type * *base*) [inline], [static]

A bit mask with the state of all I3C master error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I3C peripheral base address.

Returns

State of the error status flags:
- 1: related status flag is set.
- 0: related status flag is not set.

See Also

_i3c_master_error_flags

23.5.5.8 static void I3C_MasterClearErrorStatusFlags (I3C_Type * *base*, uint32_t *statusMask*) [inline], [static]

Parameters

base	The I3C peripheral base address.
statusMask	A bitmask of error status flags that are to be cleared. The mask is composed of _i3c_-master_error_flags enumerators OR'd together. You may pass the result of a previous call to I3C_MasterGetStatusFlags().

See Also

_i3c_master_error_flags.

23.5.5.9 i3c_master_state_t I3C_MasterGetState (I3C_Type * *base*)

Parameters

base	The I3C peripheral base address.

Returns

I3C master state.

23.5.5.10 static void I3C_MasterEnableInterrupts (I3C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Parameters

base	The I3C peripheral base address.
interruptMask	Bit mask of interrupts to enable. See _i3c_master_flags for the set of constants that should be OR'd together to form the bit mask.

23.5.5.11 static void I3C_MasterDisableInterrupts (I3C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

All flags except kI3C_MasterBetweenFlag and kI3C_MasterNackDetectFlag can be enabled as interrupts.

Parameters

base	The I3C peripheral base address.
interruptMask	Bit mask of interrupts to disable. See _i3c_master_flags for the set of constants that should be OR'd together to form the bit mask.

23.5.5.12 static uint32_t I3C_MasterGetEnabledInterrupts (I3C_Type * *base*) [inline], [static]

Parameters

base	The I3C peripheral base address.

Returns

A bitmask composed of _i3c_master_flags enumerators OR'd together to indicate the set of enabled interrupts.

23.5.5.13 static uint32_t I3C_MasterGetPendingInterrupts (I3C_Type * *base*) [inline], [static]

Parameters

base	The I3C peripheral base address.

Returns

A bitmask composed of _i3c_master_flags enumerators OR'd together to indicate the set of pending interrupts.

23.5.5.14 static void I3C_MasterEnableDMA (I3C_Type * *base*, bool *enableTx*, bool *enableRx*, uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
enableTx	Enable flag for transmit DMA request. Pass true for enable, false for disable.
enableRx	Enable flag for receive DMA request. Pass true for enable, false for disable.
width	DMA read/write unit in bytes.

23.5.5.15 static uint32_t I3C_MasterGetTxFifoAddress (I3C_Type ∗ *base,* uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
width	DMA read/write unit in bytes.

Returns

The I3C Master Transmit Data Register address.

23.5.5.16 static uint32_t I3C_MasterGetRxFifoAddress (I3C_Type ∗ *base,* uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
width	DMA read/write unit in bytes.

Returns

The I3C Master Receive Data Register address.

23.5.5.17 static void I3C_MasterSetWatermarks (I3C_Type ∗ *base,* i3c_tx_trigger_level_t *txLvl,* i3c_rx_trigger_level_t *rxLvl,* bool *flushTx,* bool *flushRx*) [inline], [static]

Parameters

base	The I3C peripheral base address.
txLvl	Transmit FIFO watermark level. The kI3C_MasterTxReadyFlag flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
rxLvl	Receive FIFO watermark level. The kI3C_MasterRxReadyFlag flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
flushTx	true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
flushRx	true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

23.5.5.18 static void I3C_MasterGetFifoCounts (I3C_Type * *base*, size_t * *rxCount*, size_t * *txCount*) [inline], [static]

Parameters

	base	The I3C peripheral base address.
out	*txCount*	Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
out	*rxCount*	Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

23.5.5.19 void I3C_MasterSetBaudRate (I3C_Type * *base*, const i3c_baudrate_hz_t * *baudRate_Hz*, uint32_t *sourceClock_Hz*)

The I3C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

Parameters

base	The I3C peripheral base address.
baudRate_Hz	Pointer to structure of requested bus frequency in Hertz.
sourceClock_Hz	I3C functional clock frequency in Hertz.

23.5.5.20 static bool I3C_MasterGetBusIdleState (I3C_Type * *base*) [inline], [static]

Requires the master mode to be enabled.

Parameters

base	The I3C peripheral base address.

Return values

true	Bus is busy.
false	Bus is idle.

23.5.5.21 status_t I3C_MasterStart (I3C_Type * *base*, i3c_bus_type_t *type*, uint8_t *address*, i3c_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

base	The I3C peripheral base address.
type	The bus type to use in this transaction.
address	7-bit slave device address, in bits [6:0].
dir	Master transfer direction, either kI3C_Read or kI3C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

kStatus_Success	START signal and address were successfully enqueued in the transmit FIFO.
kStatus_I3C_Busy	Another master is currently utilizing the bus.

23.5.5.22 status_t I3C_MasterRepeatedStart (I3C_Type * *base*, i3c_bus_type_t *type*, uint8_t *address*, i3c_direction_t *dir*)

This function is used to send a Repeated START signal when a transfer is already in progress. Like I3C_MasterStart(), it also sends the specified 7-bit address.

Note

This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

base	The I3C peripheral base address.
type	The bus type to use in this transaction.
address	7-bit slave device address, in bits [6:0].
dir	Master transfer direction, either kI3C_Read or kI3C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.

Return values

kStatus_Success	Repeated START signal and address were successfully enqueued in the transmit FIFO.

23.5.5.23 status_t I3C_MasterRepeatedStartWithRxSize (I3C_Type * *base*, i3c_bus_type_t *type*, uint8_t *address*, i3c_direction_t *dir*, uint8_t *rxSize*)

This function is used to send a Repeated START signal when a transfer is already in progress. Like I3C_MasterStart(), it also sends the specified 7-bit address. Call this API also configures the read terminate size for the following read transfer. For example, set the rxSize = 2, the following read transfer will be terminated after two bytes of data received. Write transfer will not be affected by the rxSize configuration.

Note

This function exists primarily to maintain compatible APIs between I3C and I2C drivers, as well as to better document the intent of code that uses these APIs.

Parameters

base	The I3C peripheral base address.
type	The bus type to use in this transaction.
address	7-bit slave device address, in bits [6:0].
dir	Master transfer direction, either kI3C_Read or kI3C_Write. This parameter is used to set the R/w bit (bit 0) in the transmitted slave address.
rxSize	Read terminate size for the followed read transfer, limit to 255 bytes.

Return values

kStatus_Success	Repeated START signal and address were successfully enqueued in the transmit FIFO.

23.5.5.24 status_t I3C_MasterSend (I3C_Type * *base*, const void * *txBuff*, size_t *txSize*, uint32_t *flags*)

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns kStatus_I3C_Nak.

Parameters

base	The I3C peripheral base address.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.
flags	Bit mask of options for the transfer. See enumeration _i3c_master_transfer_flags for available options.

Return values

kStatus_Success	Data was sent successfully.
kStatus_I3C_Busy	Another master is currently utilizing the bus.
kStatus_I3C_Timeout	The module has stalled too long in a frame.
kStatus_I3C_Nak	The slave device sent a NAK in response to an address.
kStatus_I3C_WriteAbort	The slave device sent a NAK in response to a write.
kStatus_I3C_MsgError	Message SDR/DDR mismatch or read/write message in wrong state.
kStatus_I3C_WriteFifo-Error	Write to M/SWDATAB register when FIFO full.
kStatus_I3C_InvalidReq	Invalid use of request.

23.5.5.25 status_t I3C_MasterReceive (I3C_Type * *base*, void * *rxBuff*, size_t *rxSize*, uint32_t *flags*)

Parameters

base	The I3C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.
flags	Bit mask of options for the transfer. See enumeration _i3c_master_transfer_flags for available options.

Return values

kStatus_Success	Data was received successfully.
kStatus_I3C_Busy	Another master is currently utilizing the bus.
kStatus_I3C_Timeout	The module has stalled too long in a frame.
kStatus_I3C_Term	The master terminates slave read.
kStatus_I3C_HdrParity-Error	Parity error from DDR read.
kStatus_I3C_CrcError	CRC error from DDR read.
kStatus_I3C_MsgError	Message SDR/DDR mismatch or read/write message in wrong state.
kStatus_I3C_ReadFifo-Error	Read from M/SRDATAB register when FIFO empty.
kStatus_I3C_InvalidReq	Invalid use of request.

23.5.5.26 status_t I3C_MasterStop (I3C_Type * *base*)

This function does not return until the STOP signal is seen on the bus, or an error occurs.

Parameters

base	The I3C peripheral base address.

Return values

kStatus_Success	The STOP signal was successfully sent on the bus and the transaction terminated.
kStatus_I3C_Busy	Another master is currently utilizing the bus.
kStatus_I3C_Timeout	The module has stalled too long in a frame.
kStatus_I3C_InvalidReq	Invalid use of request.

23.5.5.27 void I3C_MasterEmitRequest (I3C_Type * *base,* i3c_bus_request_t *masterReq*)

Parameters

base	The I3C peripheral base address.
masterReq	I3C master request of type i3c_bus_request_t

23.5.5.28 static void I3C_MasterEmitIBIResponse (I3C_Type * *base*, i3c_ibi_response_t *ibiResponse*) [inline], [static]

Parameters

base	The I3C peripheral base address.
ibiResponse	I3C master emit IBI response of type i3c_ibi_response_t

23.5.5.29 void I3C_MasterRegisterIBI (I3C_Type * *base*, i3c_register_ibi_addr_t * *ibiRule*)

Parameters

base	The I3C peripheral base address.
ibiRule	Pointer to ibi rule description of type i3c_register_ibi_addr_t

23.5.5.30 void I3C_MasterGetIBIRules (I3C_Type * *base*, i3c_register_ibi_addr_t * *ibiRule*)

Parameters

base	The I3C peripheral base address.
ibiRule	Pointer to store the read out ibi rule description.

23.5.5.31 status_t I3C_MasterProcessDAA (I3C_Type * *base*, uint8_t * *addressList*, uint32_t *count*)

Parameters

base	The I3C peripheral base address.
addressList	The pointer for address list which is used to do DAA.
count	The address count in the address list.

Return values

kStatus_Success	The transaction was started successfully.
kStatus_I3C_Busy	Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.
kStatus_I3C_SlaveCount-Exceed	The I3C slave count has exceed the definition in I3C_MAX_DEVCNT.

23.5.5.32 status_t I3C_MasterTransferBlocking (I3C_Type * *base,* i3c_master_transfer_t * *transfer*)

Note

The API does not return until the transfer succeeds or fails due to error happens during transfer.

Parameters

base	The I3C peripheral base address.
transfer	Pointer to the transfer structure.

Return values

kStatus_Success	Data was received successfully.
kStatus_I3C_Busy	Another master is currently utilizing the bus.
kStatus_I3C_IBIWon	The I3C slave event IBI or MR or HJ won the arbitration on a header address.
kStatus_I3C_Timeout	The module has stalled too long in a frame.
kStatus_I3C_Nak	The slave device sent a NAK in response to an address.
kStatus_I3C_WriteAbort	The slave device sent a NAK in response to a write.
kStatus_I3C_Term	The master terminates slave read.
kStatus_I3C_HdrParity-Error	Parity error from DDR read.
kStatus_I3C_CrcError	CRC error from DDR read.
kStatus_I3C_MsgError	Message SDR/DDR mismatch or read/write message in wrong state.
kStatus_I3C_ReadFifo-Error	Read from M/SRDATAB register when FIFO empty.
kStatus_I3C_WriteFifo-Error	Write to M/SWDATAB register when FIFO full.
kStatus_I3C_InvalidReq	Invalid use of request.

23.5.5.33 void I3C_MasterTransferCreateHandle (I3C_Type * *base*, i3c_master_handle_t * *handle*, const i3c_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 I3C_MasterTransferAbort() API shall be called.

Note

The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

	base	The I3C peripheral base address.
out	*handle*	Pointer to the I3C master driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

23.5.5.34 status_t I3C_MasterTransferNonBlocking (I3C_Type * *base*, i3c_master_handle_t * *handle*, i3c_master_transfer_t * *transfer*)

Parameters

base	The I3C peripheral base address.
handle	Pointer to the I3C master driver handle.
transfer	The pointer to the transfer descriptor.

Return values

kStatus_Success	The transaction was started successfully.
kStatus_I3C_Busy	Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress.

23.5.5.35 status_t I3C_MasterTransferGetCount (I3C_Type * *base*, i3c_master_handle_t * *handle*, size_t * *count*)

Parameters

	base	The I3C peripheral base address.
	handle	Pointer to the I3C master driver handle.
out	*count*	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_Success	
kStatus_NoTransferInProgress	There is not a non-blocking transaction currently in progress.

23.5.5.36 void I3C_MasterTransferAbort (I3C_Type * *base*, i3c_master_handle_t * *handle*)

Note

It is not safe to call this function from an IRQ handler that has a higher priority than the I3C peripheral's IRQ priority.

Parameters

base	The I3C peripheral base address.
handle	Pointer to the I3C master driver handle.

Return values

kStatus_Success	A transaction was successfully aborted.
kStatus_I3C_Idle	There is not a non-blocking transaction currently in progress.

23.5.5.37 void I3C_MasterTransferHandleIRQ (I3C_Type * *base*, i3c_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

base	The I3C peripheral base address.
handle	Pointer to the I3C master driver handle.

23.5.5.38 void I3C_SlaveGetDefaultConfig (i3c_slave_config_t * *slaveConfig*)

This function provides the following default configuration for the I3C slave peripheral:

```
*  slaveConfig->enableslave            = true;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with I3C_SlaveInit().

Parameters

out	*slaveConfig*	User provided configuration structure for default values. Refer to i3c_slave_config_t.

23.5.5.39 void I3C_SlaveInit (I3C_Type * *base*, const i3c_slave_config_t * *slaveConfig*, uint32_t *slowClock_Hz*)

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

base	The I3C peripheral base address.
slaveConfig	User provided peripheral configuration. Use I3C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
slowClock_Hz	Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values.

23.5.5.40 void I3C_SlaveDeinit (I3C_Type * *base*)

This function disables the I3C slave peripheral and gates the clock.

Parameters

base	The I3C peripheral base address.

23.5.5.41 static void I3C_SlaveEnable (I3C_Type * *base*, bool *isEnable*) [inline], [static]

Parameters

base	The I3C peripheral base address.
isEnable	Enable or disable.

23.5.5.42 static uint32_t I3C_SlaveGetStatusFlags (I3C_Type * *base*) [inline], [static]

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I3C peripheral base address.

Returns

State of the status flags:
- 1: related status flag is set.
- 0: related status flag is not set.

See Also

_i3c_slave_flags

23.5.5.43 static void I3C_SlaveClearStatusFlags (I3C_Type * *base*, uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared:

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Attempts to clear other flags has no effect.

Parameters

base	The I3C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i3c_slave_flags enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetStatusFlags().

See Also

_i3c_slave_flags.

23.5.5.44 static uint32_t I3C_SlaveGetErrorStatusFlags (I3C_Type * *base*) [inline], [static]

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I3C peripheral base address.

Returns

State of the error status flags:
- 1: related status flag is set.
- 0: related status flag is not set.

See Also

_i3c_slave_error_flags

23.5.5.45 static void I3C_SlaveClearErrorStatusFlags (I3C_Type * *base*, uint32_t *statusMask*) [inline], [static]

Parameters

base	The I3C peripheral base address.
statusMask	A bitmask of error status flags that are to be cleared. The mask is composed of _i3c_slave_error_flags enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetErrorStatusFlags().

See Also

_i3c_slave_error_flags.

23.5.5.46 i3c_slave_activity_state_t I3C_SlaveGetActivityState (I3C_Type * *base*)

Parameters

base	The I3C peripheral base address.

Returns

I3C slave activity state, refer i3c_slave_activity_state_t.

23.5.5.47 static void I3C_SlaveEnableInterrupts (I3C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

base	The I3C peripheral base address.
interruptMask	Bit mask of interrupts to enable. See _i3c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

23.5.5.48 static void I3C_SlaveDisableInterrupts (I3C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag

- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

base	The I3C peripheral base address.
interruptMask	Bit mask of interrupts to disable. See _i3c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

23.5.5.49 static uint32_t I3C_SlaveGetEnabledInterrupts (I3C_Type * *base*) [inline], [static]

Parameters

base	The I3C peripheral base address.

Returns

A bitmask composed of _i3c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

23.5.5.50 static uint32_t I3C_SlaveGetPendingInterrupts (I3C_Type * *base*) [inline], [static]

Parameters

base	The I3C peripheral base address.

Returns

A bitmask composed of _i3c_slave_flags enumerators OR'd together to indicate the set of pending interrupts.

23.5.5.51 static void I3C_SlaveEnableDMA (I3C_Type * *base*, bool *enableTx*, bool *enableRx*, uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
enableTx	Enable flag for transmit DMA request. Pass true for enable, false for disable.
enableRx	Enable flag for receive DMA request. Pass true for enable, false for disable.
width	DMA read/write unit in bytes.

23.5.5.52 static uint32_t I3C_SlaveGetTxFifoAddress (I3C_Type ∗ *base,* uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
width	DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

23.5.5.53 static uint32_t I3C_SlaveGetRxFifoAddress (I3C_Type ∗ *base,* uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
width	DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

23.5.5.54 static void I3C_SlaveSetWatermarks (I3C_Type ∗ *base,* i3c_tx_trigger_level_t *txLvl,* i3c_rx_trigger_level_t *rxLvl,* bool *flushTx,* bool *flushRx*) [inline], [static]

Parameters

base	The I3C peripheral base address.
txLvl	Transmit FIFO watermark level. The kI3C_SlaveTxReadyFlag flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
rxLvl	Receive FIFO watermark level. The kI3C_SlaveRxReadyFlag flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
flushTx	true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
flushRx	true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

23.5.5.55 static void I3C_SlaveGetFifoCounts (I3C_Type ∗ *base*, size_t ∗ *rxCount*, size_t ∗ *txCount*) [inline], [static]

Parameters

	base	The I3C peripheral base address.
out	*txCount*	Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
out	*rxCount*	Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

23.5.5.56 void I3C_SlaveRequestEvent (I3C_Type ∗ *base*, i3c_slave_event_t *event*)

Parameters

base	The I3C peripheral base address.
event	I3C slave event of type i3c_slave_event_t

23.5.5.57 status_t I3C_SlaveSend (I3C_Type ∗ *base*, const void ∗ *txBuff*, size_t *txSize*)

Parameters

base	The I3C peripheral base address.

txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

23.5.5.58 status_t I3C_SlaveReceive (I3C_Type * *base,* void * *rxBuff,* size_t *rxSize*)

Parameters

base	The I3C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

23.6 I3C Slave Driver

23.6.1 Overview

Data Structures

- struct i3c_slave_config_t
  Structure with settings to initialize the I3C slave module. More...
- struct i3c_slave_transfer_t
  I3C slave transfer structure. More...
- struct i3c_slave_handle_t
  I3C slave handle structure. More...

Typedefs

- typedef void(∗ i3c_slave_transfer_callback_t)(I3C_Type ∗base, i3c_slave_transfer_t ∗transfer, void ∗userData)
  Slave event callback function pointer type.

Enumerations

- enum _i3c_slave_flags {
  kI3C_SlaveNotStopFlag = I3C_SSTATUS_STNOTSTOP_MASK,
  kI3C_SlaveMessageFlag = I3C_SSTATUS_STMSG_MASK,
  kI3C_SlaveRequiredReadFlag = I3C_SSTATUS_STREQRD_MASK,
  kI3C_SlaveRequiredWriteFlag = I3C_SSTATUS_STREQWR_MASK,
  kI3C_SlaveBusDAAFlag = I3C_SSTATUS_STDAA_MASK,
  kI3C_SlaveBusHDRModeFlag = I3C_SSTATUS_STHDR_MASK,
  kI3C_SlaveBusStartFlag = I3C_SSTATUS_START_MASK,
  kI3C_SlaveMatchedFlag = I3C_SSTATUS_MATCHED_MASK,
  kI3C_SlaveBusStopFlag = I3C_SSTATUS_STOP_MASK,
  kI3C_SlaveRxReadyFlag = I3C_SSTATUS_RX_PEND_MASK,
  kI3C_SlaveTxReadyFlag = I3C_SSTATUS_TXNOTFULL_MASK,
  kI3C_SlaveDynamicAddrChangedFlag,
  kI3C_SlaveReceivedCCCFlag = I3C_SSTATUS_CCC_MASK,
  kI3C_SlaveErrorFlag = I3C_SSTATUS_ERRWARN_MASK,
  kI3C_SlaveHDRCommandMatchFlag = I3C_SSTATUS_HDRMATCH_MASK,
  kI3C_SlaveCCCHandledFlag,
  kI3C_SlaveEventSentFlag = I3C_SSTATUS_EVENT_MASK,
  kI3C_SlaveIbiDisableFlag = I3C_SSTATUS_IBIDIS_MASK,
  kI3C_SlaveMasterRequestDisabledFlag = I3C_SSTATUS_MRDIS_MASK,
  kI3C_SlaveHotJoinDisabledFlag = I3C_SSTATUS_HJDIS_MASK,
  kI3C_SlaveClearFlags = kI3C_SlaveBusStartFlag | kI3C_SlaveMatchedFlag | kI3C_SlaveBusStop-Flag }

I3C slave peripheral flags.

- enum _i3c_slave_error_flags {
  kI3C_SlaveErrorOverrunFlag = I3C_SERRWARN_ORUN_MASK,
  kI3C_SlaveErrorUnderrunFlag = I3C_SERRWARN_URUN_MASK,
  kI3C_SlaveErrorUnderrunNakFlag,
  kI3C_SlaveErrorTermFlag = I3C_SERRWARN_TERM_MASK,
  kI3C_SlaveErrorInvalidStartFlag = I3C_SERRWARN_INVSTART_MASK,
  kI3C_SlaveErrorSdrParityFlag = I3C_SERRWARN_SPAR_MASK,
  kI3C_SlaveErrorHdrParityFlag = I3C_SERRWARN_HPAR_MASK,
  kI3C_SlaveErrorHdrCRCFlag = I3C_SERRWARN_HCRC_MASK,
  kI3C_SlaveErrorS0S1Flag = I3C_SERRWARN_S0S1_MASK,
  kI3C_SlaveErrorOverreadFlag = I3C_SERRWARN_OREAD_MASK,
  kI3C_SlaveErrorOverwriteFlag = I3C_SERRWARN_OWRITE_MASK }

  I3C slave error flags to indicate the causes.
- enum i3c_slave_event_t {
  kI3C_SlaveEventNormal = 0U,
  kI3C_SlaveEventIBI = 1U,
  kI3C_SlaveEventMasterReq = 2U,
  kI3C_SlaveEventHotJoinReq = 3U }

  I3C slave.event.
- enum i3c_slave_activity_state_t {
  kI3C_SlaveNoLatency = 0U,
  kI3C_SlaveLatency1Ms = 1U,
  kI3C_SlaveLatency100Ms = 2U,
  kI3C_SlaveLatency10S = 3U }

  I3C slave.activity state.
- enum i3c_slave_transfer_event_t {
  kI3C_SlaveAddressMatchEvent = 0x01U,
  kI3C_SlaveTransmitEvent = 0x02U,
  kI3C_SlaveReceiveEvent = 0x04U,
  kI3C_SlaveRequiredTransmitEvent = 0x08U,
  kI3C_SlaveStartEvent = 0x10U,
  kI3C_SlaveHDRCommandMatchEvent = 0x20U,
  kI3C_SlaveCompletionEvent = 0x40U,
  kI3C_SlaveRequestSentEvent = 0x80U,
  kI3C_SlaveReceivedCCCEvent = 0x100L,
  kI3C_SlaveAllEvents }

  Set of events sent to the callback for non blocking slave transfers.

Initialization and deinitialization

- void I3C_SlaveGetDefaultConfig (i3c_slave_config_t *slaveConfig)

  Provides a default configuration for the I3C slave peripheral.
- void I3C_SlaveInit (I3C_Type *base, const i3c_slave_config_t *slaveConfig, uint32_t slowClock_-Hz)

Initializes the I3C slave peripheral.
- void I3C_SlaveDeinit (I3C_Type *base)
  Deinitializes the I3C slave peripheral.
- static void I3C_SlaveEnable (I3C_Type *base, bool isEnable)
  Enable/Disable Slave.

Status

- static uint32_t I3C_SlaveGetStatusFlags (I3C_Type *base)
  Gets the I3C slave status flags.
- static void I3C_SlaveClearStatusFlags (I3C_Type *base, uint32_t statusMask)
  Clears the I3C slave status flag state.
- static uint32_t I3C_SlaveGetErrorStatusFlags (I3C_Type *base)
  Gets the I3C slave error status flags.
- static void I3C_SlaveClearErrorStatusFlags (I3C_Type *base, uint32_t statusMask)
  Clears the I3C slave error status flag state.
- i3c_slave_activity_state_t I3C_SlaveGetActivityState (I3C_Type *base)
  Gets the I3C slave state.

Interrupts

- static void I3C_SlaveEnableInterrupts (I3C_Type *base, uint32_t interruptMask)
  Enables the I3C slave interrupt requests.
- static void I3C_SlaveDisableInterrupts (I3C_Type *base, uint32_t interruptMask)
  Disables the I3C slave interrupt requests.
- static uint32_t I3C_SlaveGetEnabledInterrupts (I3C_Type *base)
  Returns the set of currently enabled I3C slave interrupt requests.
- static uint32_t I3C_SlaveGetPendingInterrupts (I3C_Type *base)
  Returns the set of pending I3C slave interrupt requests.

DMA control

- static void I3C_SlaveEnableDMA (I3C_Type *base, bool enableTx, bool enableRx, uint32_t width)
  Enables or disables I3C slave DMA requests.
- static uint32_t I3C_SlaveGetTxFifoAddress (I3C_Type *base, uint32_t width)
  Gets I3C slave transmit data register address for DMA transfer.
- static uint32_t I3C_SlaveGetRxFifoAddress (I3C_Type *base, uint32_t width)
  Gets I3C slave receive data register address for DMA transfer.

FIFO control

- static void I3C_SlaveSetWatermarks (I3C_Type *base, i3c_tx_trigger_level_t txLvl, i3c_rx_trigger-_level_t rxLvl, bool flushTx, bool flushRx)
  Sets the watermarks for I3C slave FIFOs.
- static void I3C_SlaveGetFifoCounts (I3C_Type *base, size_t *rxCount, size_t *txCount)
  Gets the current number of bytes in the I3C slave FIFOs.

Bus operations

- void I3C_SlaveRequestEvent (I3C_Type *base, i3c_slave_event_t event)
  I3C slave request event.
- status_t I3C_SlaveSend (I3C_Type *base, const void *txBuff, size_t txSize)
  Performs a polling send transfer on the I3C bus.
- status_t I3C_SlaveReceive (I3C_Type *base, void *rxBuff, size_t rxSize)
  Performs a polling receive transfer on the I3C bus.

Slave non-blocking

- void I3C_SlaveTransferCreateHandle (I3C_Type *base, i3c_slave_handle_t *handle, i3c_slave_-transfer_callback_t callback, void *userData)
  Creates a new handle for the I3C slave non-blocking APIs.
- status_t I3C_SlaveTransferNonBlocking (I3C_Type *base, i3c_slave_handle_t *handle, uint32_t eventMask)
  Starts accepting slave transfers.
- status_t I3C_SlaveTransferGetCount (I3C_Type *base, i3c_slave_handle_t *handle, size_t *count)
  Gets the slave transfer status during a non-blocking transfer.
- void I3C_SlaveTransferAbort (I3C_Type *base, i3c_slave_handle_t *handle)
  Aborts the slave non-blocking transfers.

Slave IRQ handler

- void I3C_SlaveTransferHandleIRQ (I3C_Type *base, i3c_slave_handle_t *handle)
  Reusable routine to handle slave interrupts.
- void I3C_SlaveRequestIBIWithData (I3C_Type *base, i3c_slave_handle_t *handle, uint8_t *data, size_t dataSize)
  I3C slave request IBI event with payload.

23.6.2 Data Structure Documentation

23.6.2.1 struct i3c_slave_config_t

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_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
  Whether to enable slave.
- bool isHotJoin

Whether to enable slave hotjoin before enable slave.

- uint8_t staticAddr
  Static address.
- uint16_t vendorID
  Device vendor ID(manufacture ID).
- bool enableRandomPart
  Whether to generate random part number, if using random part number, the partNumber variable setting is meaningless.
- uint32_t partNumber
  Device part number info.
- uint8_t dcr
  Device characteristics register information.
- uint8_t bcr
  Bus characteristics register information.
- uint8_t hdrMode
  Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.
- bool nakAllRequest
  Whether to reply NAK to all requests except broadcast CCC.
- bool ignoreS0S1Error
  Whether to ignore S0/S1 error in SDR mode.
- bool offline
  Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.
- bool matchSlaveStartStop
  Whether to assert start/stop status only the time slave is addressed.
- uint32_t maxWriteLength
  Maximum write length.
- uint32_t maxReadLength
  Maximum read length.

Field Documentation

(1) bool i3c_slave_config_t::enableSlave

(2) bool i3c_slave_config_t::isHotJoin

(3) uint8_t i3c_slave_config_t::staticAddr

(4) uint16_t i3c_slave_config_t::vendorID

(5) bool i3c_slave_config_t::enableRandomPart

(6) uint8_t i3c_slave_config_t::dcr

(7) uint8_t i3c_slave_config_t::bcr

(8) uint8_t i3c_slave_config_t::hdrMode

(9) bool i3c_slave_config_t::nakAllRequest

(10) bool i3c_slave_config_t::ignoreS0S1Error

(11) bool i3c_slave_config_t::offline

(12) bool i3c_slave_config_t::matchSlaveStartStop

(13) uint32_t i3c_slave_config_t::maxWriteLength

(14) uint32_t i3c_slave_config_t::maxReadLength

23.6.2.2 struct i3c_slave_transfer_t

Data Fields

- uint32_t event
  Reason the callback is being invoked.
- uint8_t * txData
  Transfer buffer.
- size_t txDataSize
  Transfer size.
- uint8_t * rxData
  Transfer buffer.
- size_t rxDataSize
  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.

Field Documentation

(1) uint32_t i3c_slave_transfer_t::event

(2) status_t i3c_slave_transfer_t::completionStatus

Only applies for kI3C_SlaveCompletionEvent.

(3) size_t i3c_slave_transfer_t::transferredCount

23.6.2.3 struct _i3c_slave_handle

Note

The contents of this structure are private and subject to change.

Data Fields

- i3c_slave_transfer_t transfer
  I3C 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.
- i3c_slave_transfer_callback_t callback
  Callback function called at transfer event.
- void ∗ userData
  Callback parameter passed to callback.
- uint8_t ∗ ibiData
  IBI data buffer.
- size_t ibiDataSize
  IBI data size.

Field Documentation

(1) i3c_slave_transfer_t i3c_slave_handle_t::transfer

(2) bool i3c_slave_handle_t::isBusy

(3) bool i3c_slave_handle_t::wasTransmit

(4) uint32_t i3c_slave_handle_t::eventMask

(5) uint32_t i3c_slave_handle_t::transferredCount

(6) i3c_slave_transfer_callback_t i3c_slave_handle_t::callback

(7) void∗ i3c_slave_handle_t::userData

23.6.3 Typedef Documentation

23.6.3.1 typedef void(∗ i3c_slave_transfer_callback_t)(I3C_Type ∗base, i3c_slave_transfer_t ∗transfer, void ∗userData)

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I3C_-SlaveSetCallback() function after you have created a handle.

Parameters

base	Base address for the I3C instance on which the event occurred.
transfer	Pointer to transfer descriptor containing values passed to and/or from the callback.
userData	Arbitrary pointer-sized value passed from the application.

23.6.4 Enumeration Type Documentation

23.6.4.1 enum _i3c_slave_flags

The following status register flags can be cleared:

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI3C_SlaveNotStopFlag Slave status not stop flag.

kI3C_SlaveMessageFlag Slave status message, indicating slave is listening to the bus traffic or responding.

kI3C_SlaveRequiredReadFlag Slave status required, either is master doing SDR read from slave, or is IBI pushing out.

kI3C_SlaveRequiredWriteFlag Slave status request write, master is doing SDR write to slave, except slave in ENTDAA mode.

kI3C_SlaveBusDAAFlag I3C bus is in ENTDAA mode.

kI3C_SlaveBusHDRModeFlag I3C bus is in HDR mode.

kI3C_SlaveBusStartFlag Start/Re-start event is seen since the bus was last cleared.

kI3C_SlaveMatchedFlag Slave address(dynamic/static) matched since last cleared.

kI3C_SlaveBusStopFlag Stop event is seen since the bus was last cleared.

kI3C_SlaveRxReadyFlag Rx data ready in rx buffer flag.

kI3C_SlaveTxReadyFlag Tx buffer ready for Tx data flag.

kI3C_SlaveDynamicAddrChangedFlag Slave dynamic address has been assigned, re-assigned, or lost.

kI3C_SlaveReceivedCCCFlag Slave received Common command code.

kI3C_SlaveErrorFlag Error occurred flag.

kI3C_SlaveHDRCommandMatchFlag High data rate command match.

kI3C_SlaveCCCHandledFlag Slave received Common command code is handled by I3C module.

kI3C_SlaveEventSentFlag Slave IBI/P2P/MR/HJ event has been sent.
kI3C_SlaveIbiDisableFlag Slave in band interrupt is disabled.
kI3C_SlaveMasterRequestDisabledFlag Slave master request is disabled.
kI3C_SlaveHotJoinDisabledFlag Slave Hot-Join is disabled.
kI3C_SlaveClearFlags All flags which are cleared by the driver upon starting a transfer.

23.6.4.2 enum _i3c_slave_error_flags

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

kI3C_SlaveErrorOverrunFlag Slave internal from-bus buffer/FIFO overrun.
kI3C_SlaveErrorUnderrunFlag Slave internal to-bus buffer/FIFO underrun.
kI3C_SlaveErrorUnderrunNakFlag Slave internal from-bus buffer/FIFO underrun and NACK error.
kI3C_SlaveErrorTermFlag Terminate error from master.
kI3C_SlaveErrorInvalidStartFlag Slave invalid start flag.
kI3C_SlaveErrorSdrParityFlag SDR parity error.
kI3C_SlaveErrorHdrParityFlag HDR parity error.
kI3C_SlaveErrorHdrCRCFlag HDR-DDR CRC error.
kI3C_SlaveErrorS0S1Flag S0 or S1 error.
kI3C_SlaveErrorOverreadFlag Over-read error.
kI3C_SlaveErrorOverwriteFlag Over-write error.

23.6.4.3 enum i3c_slave_event_t

Enumerator

kI3C_SlaveEventNormal Normal mode.
kI3C_SlaveEventIBI In band interrupt event.
kI3C_SlaveEventMasterReq Master request event.
kI3C_SlaveEventHotJoinReq Hot-join event.

23.6.4.4 enum i3c_slave_activity_state_t

Enumerator

kI3C_SlaveNoLatency Normal bus operation.
kI3C_SlaveLatency1Ms 1ms of latency.
kI3C_SlaveLatency100Ms 100ms of latency.
kI3C_SlaveLatency10S 10s latency.

23.6.4.5 enum i3c_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 I3C_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

kI3C_SlaveAddressMatchEvent Received the slave address after a start or repeated start.
kI3C_SlaveTransmitEvent Callback is requested to provide data to transmit (slave-transmitter role).

kI3C_SlaveReceiveEvent Callback is requested to provide a buffer in which to place received data (slave-receiver role).
kI3C_SlaveRequiredTransmitEvent Callback is requested to provide a buffer in which to place received data (slave-receiver role).
kI3C_SlaveStartEvent A start/repeated start was detected.
kI3C_SlaveHDRCommandMatchEvent Slave Match HDR Command.
kI3C_SlaveCompletionEvent A stop was detected, completing the transfer.
kI3C_SlaveRequestSentEvent Slave request event sent.
kI3C_SlaveReceivedCCCEvent Slave received CCC event, need to handle by application.
kI3C_SlaveAllEvents Bit mask of all available events.

23.6.5 Function Documentation

23.6.5.1 void I3C_SlaveGetDefaultConfig (i3c_slave_config_t * *slaveConfig*)

This function provides the following default configuration for the I3C slave peripheral:

```
*  slaveConfig->enableslave            = true;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the slave driver with I3C_SlaveInit().

Parameters

out	*slaveConfig*	User provided configuration structure for default values. Refer to i3c_slave_config_t.

23.6.5.2 void I3C_SlaveInit (I3C_Type * *base*, const i3c_slave_config_t * *slaveConfig*, uint32_t *slowClock_Hz*)

This function enables the peripheral clock and initializes the I3C slave peripheral as described by the user provided configuration.

Parameters

base	The I3C peripheral base address.
slaveConfig	User provided peripheral configuration. Use I3C_SlaveGetDefaultConfig() to get a set of defaults that you can override.
slowClock_Hz	Frequency in Hertz of the I3C slow clock. Used to calculate the bus match condition values.

23.6.5.3 void I3C_SlaveDeinit (I3C_Type * *base*)

This function disables the I3C slave peripheral and gates the clock.

Parameters

base	The I3C peripheral base address.

23.6.5.4 static void I3C_SlaveEnable (I3C_Type * *base*, bool *isEnable*) [inline], [static]

Parameters

base	The I3C peripheral base address.
isEnable	Enable or disable.

23.6.5.5 static uint32_t I3C_SlaveGetStatusFlags (I3C_Type * *base*) [inline], [static]

A bit mask with the state of all I3C slave status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I3C peripheral base address.

Returns

State of the status flags:
- 1: related status flag is set.
- 0: related status flag is not set.

See Also

_i3c_slave_flags

23.6.5.6 static void I3C_SlaveClearStatusFlags (I3C_Type * *base*, uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared:

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag

Attempts to clear other flags has no effect.

Parameters

base	The I3C peripheral base address.
statusMask	A bitmask of status flags that are to be cleared. The mask is composed of _i3c_slave_flags enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetStatusFlags().

See Also

_i3c_slave_flags.

23.6.5.7 static uint32_t I3C_SlaveGetErrorStatusFlags (I3C_Type * *base*) [inline], [static]

A bit mask with the state of all I3C slave error status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

Parameters

base	The I3C peripheral base address.

Returns

State of the error status flags:
- 1: related status flag is set.
- 0: related status flag is not set.

See Also

_i3c_slave_error_flags

23.6.5.8 static void I3C_SlaveClearErrorStatusFlags (I3C_Type * *base*, uint32_t *statusMask*) [inline], [static]

Parameters

base	The I3C peripheral base address.
statusMask	A bitmask of error status flags that are to be cleared. The mask is composed of _i3c-_slave_error_flags enumerators OR'd together. You may pass the result of a previous call to I3C_SlaveGetErrorStatusFlags().

See Also

_i3c_slave_error_flags.

23.6.5.9 i3c_slave_activity_state_t I3C_SlaveGetActivityState (I3C_Type * *base*)

Parameters

base	The I3C peripheral base address.

Returns

I3C slave activity state, refer i3c_slave_activity_state_t.

23.6.5.10 static void I3C_SlaveEnableInterrupts (I3C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Only below flags can be enabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

base	The I3C peripheral base address.
interruptMask	Bit mask of interrupts to enable. See _i3c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

23.6.5.11 static void I3C_SlaveDisableInterrupts (I3C_Type * *base*, uint32_t *interruptMask*) [inline], [static]

Only below flags can be disabled as interrupts.

- kI3C_SlaveBusStartFlag
- kI3C_SlaveMatchedFlag
- kI3C_SlaveBusStopFlag
- kI3C_SlaveRxReadyFlag
- kI3C_SlaveTxReadyFlag
- kI3C_SlaveDynamicAddrChangedFlag
- kI3C_SlaveReceivedCCCFlag
- kI3C_SlaveErrorFlag
- kI3C_SlaveHDRCommandMatchFlag
- kI3C_SlaveCCCHandledFlag
- kI3C_SlaveEventSentFlag

Parameters

base	The I3C peripheral base address.
interruptMask	Bit mask of interrupts to disable. See _i3c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

23.6.5.12 static uint32_t I3C_SlaveGetEnabledInterrupts (I3C_Type * *base*) [inline], [static]

Parameters

base	The I3C peripheral base address.

Returns

A bitmask composed of _i3c_slave_flags enumerators OR'd together to indicate the set of enabled interrupts.

23.6.5.13 static uint32_t I3C_SlaveGetPendingInterrupts (I3C_Type * *base*) [inline], [static]

Parameters

base	The I3C peripheral base address.

Returns

A bitmask composed of _i3c_slave_flags enumerators OR'd together to indicate the set of pending interrupts.

23.6.5.14 static void I3C_SlaveEnableDMA (I3C_Type * *base*, bool *enableTx*, bool *enableRx*, uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
enableTx	Enable flag for transmit DMA request. Pass true for enable, false for disable.
enableRx	Enable flag for receive DMA request. Pass true for enable, false for disable.
width	DMA read/write unit in bytes.

23.6.5.15 static uint32_t I3C_SlaveGetTxFifoAddress (I3C_Type * *base,* uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
width	DMA read/write unit in bytes.

Returns

The I3C Slave Transmit Data Register address.

23.6.5.16 static uint32_t I3C_SlaveGetRxFifoAddress (I3C_Type * *base*, uint32_t *width*) [inline], [static]

Parameters

base	The I3C peripheral base address.
width	DMA read/write unit in bytes.

Returns

The I3C Slave Receive Data Register address.

23.6.5.17 static void I3C_SlaveSetWatermarks (I3C_Type * *base*, i3c_tx_trigger_level_t *txLvl*, i3c_rx_trigger_level_t *rxLvl*, bool *flushTx*, bool *flushRx*) [inline], [static]

Parameters

base	The I3C peripheral base address.
txLvl	Transmit FIFO watermark level. The kI3C_SlaveTxReadyFlag flag is set whenever the number of words in the transmit FIFO reaches *txLvl*.
rxLvl	Receive FIFO watermark level. The kI3C_SlaveRxReadyFlag flag is set whenever the number of words in the receive FIFO reaches *rxLvl*.
flushTx	true if TX FIFO is to be cleared, otherwise TX FIFO remains unchanged.
flushRx	true if RX FIFO is to be cleared, otherwise RX FIFO remains unchanged.

23.6.5.18 static void I3C_SlaveGetFifoCounts (I3C_Type * *base*, size_t * *rxCount*, size_t * *txCount*) [inline], [static]

Parameters

	base	The I3C peripheral base address.
`out`	*txCount*	Pointer through which the current number of bytes in the transmit FIFO is returned. Pass NULL if this value is not required.
`out`	*rxCount*	Pointer through which the current number of bytes in the receive FIFO is returned. Pass NULL if this value is not required.

23.6.5.19 void I3C_SlaveRequestEvent (I3C_Type * *base*, i3c_slave_event_t *event*)

Parameters

base	The I3C peripheral base address.
event	I3C slave event of type i3c_slave_event_t

23.6.5.20 status_t I3C_SlaveSend (I3C_Type * *base*, const void * *txBuff*, size_t *txSize*)

Parameters

base	The I3C peripheral base address.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

23.6.5.21 status_t I3C_SlaveReceive (I3C_Type * *base*, void * *rxBuff*, size_t *rxSize*)

Parameters

base	The I3C peripheral base address.
rxBuff	The pointer to the data to be transferred.
rxSize	The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

23.6.5.22 void I3C_SlaveTransferCreateHandle (I3C_Type * *base*, i3c_slave_handle_t * *handle*, i3c_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 I3C_SlaveTransferAbort() API shall be called.

Note

The function also enables the NVIC IRQ for the input I3C. Need to notice that on some SoCs the I3C IRQ is connected to INTMUX, in this case user needs to enable the associated INTMUX IRQ in application.

Parameters

	base	The I3C peripheral base address.
`out`	*handle*	Pointer to the I3C slave driver handle.
	callback	User provided pointer to the asynchronous callback function.
	userData	User provided pointer to the application callback data.

23.6.5.23 status_t I3C_SlaveTransferNonBlocking (I3C_Type * *base*, i3c_slave_handle_t * *handle*, uint32_t *eventMask*)

Call this API after calling I2C_SlaveInit() and I3C_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 I3C_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 i3c_slave_transfer_event_t enumerators for the events you wish to receive. The kI3C_SlaveTransmitEvent and kI3C_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 kI3C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I3C peripheral base address.
handle	Pointer to struct: _i3c_slave_handle structure which stores the transfer state.
eventMask	Bit mask formed by OR'ing together i3c_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 kI3C_SlaveAllEvents to enable all events.

Return values

kStatus_Success	Slave transfers were successfully started.
kStatus_I3C_Busy	Slave transfers have already been started on this handle.

23.6.5.24 status_t I3C_SlaveTransferGetCount (I3C_Type * *base*, i3c_slave_handle_t * *handle*, size_t * *count*)

Parameters

	base	The I3C peripheral base address.
	handle	Pointer to i2c_slave_handle_t structure.
`out`	*count*	Pointer to a value to hold the number of bytes transferred. May be NULL if the count is not required.

Return values

kStatus_Success	
kStatus_NoTransferInProgress	

23.6.5.25 void I3C_SlaveTransferAbort (I3C_Type * *base*, i3c_slave_handle_t * *handle*)

Note

This API could be called at any time to stop slave for handling the bus events.

Parameters

base	The I3C peripheral base address.
handle	Pointer to struct: _i3c_slave_handle structure which stores the transfer state.

Return values

kStatus_Success	
kStatus_I3C_Idle	

23.6.5.26 void I3C_SlaveTransferHandleIRQ (I3C_Type * *base*, i3c_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

base	The I3C peripheral base address.
handle	Pointer to struct: _i3c_slave_handle structure which stores the transfer state.

23.6.5.27 void I3C_SlaveRequestIBIWithData (I3C_Type * *base*, i3c_slave_handle_t * *handle*, uint8_t * *data*, size_t *dataSize*)

Parameters

base	The I3C peripheral base address.
handle	Pointer to struct: _i3c_slave_handle structure which stores the transfer state.
data	Pointer to IBI data to be sent in the request.
dataSize	IBI data size.

23.7 I3C Master DMA Driver

23.8 I3C FreeRTOS Driver

Chapter 24
IAP: In Application Programming Driver

24.1 Overview

The MCUXpresso SDK provides a driver for the In Application Programming (IAP).

It provides a set of functions to call the on-chip in application programming interface. User code executing from on-chip RAM can call these function to read information like part id, read and write flash, read and write ffr.

IAP Basic Operations

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

Modules

- IAP Boot Driver
- IAP FlexSPI Driver
- IAP OTP Driver

Files

- file fsl_iap.h

Driver version

- #define FSL_IAP_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
  IAP driver version 2.1.2.

24.2 Macro Definition Documentation

24.2.1 #define FSL_IAP_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))

24.3 IAP Boot Driver

24.3.1 Overview

Data Structures

- struct iap_boot_option_t
  IAP boot option. More...

Macros

- #define IAP_BOOT_OPTION_TAG (0xEBU)
  IAP boot option tag.
- #define IAP_BOOT_OPTION_MODE_MASTER (0U)
  IAP boot option mode.

Functions

- void IAP_RunBootLoader (iap_boot_option_t *option)
  Invoke into ROM with specified boot parameters.

24.3.2 Data Structure Documentation

24.3.2.1 struct iap_boot_option_t

Field Documentation

(1) uint32_t iap_boot_option_t::bootImageIndex

(2) uint32_t iap_boot_option_t::instance

(3) uint32_t iap_boot_option_t::bootInterface

(4) uint32_t iap_boot_option_t::mode

RT600: 0: USART 1: I2C 2: SPI 3: USB HID 4:FlexSPI 7:SD 8:MMC

24.3.3 Function Documentation

24.3.3.1 void IAP_RunBootLoader (iap_boot_option_t * *option*)

Parameters

option	Boot parameters. Refer to iap_boot_option_t.

24.4 IAP FlexSPI Driver

24.4.1 Overview

Data Structures

- struct serial_nor_config_option_t
  Serial NOR Configuration Option. More...
- union flash_run_context_t
  Flash Run Context. More...
- struct flexspi_lut_seq_t
  FlexSPI LUT Sequence structure. More...
- struct flexspi_dll_time_t
  FlexSPI Dll Time Block. More...
- struct flexspi_mem_config_block_t
  FlexSPI Memory Configuration Block. More...
- struct flexspi_xfer_t
  FlexSPI Transfer Context. More...
- struct flexspi_nor_config_t
  Serial NOR configuration block. More...

Macros

- #define NOR_CMD_INDEX_READ 0
  FlexSPI LUT command.
- #define NOR_CMD_INDEX_READSTATUS 1
  1
- #define NOR_CMD_INDEX_WRITEENABLE 2
  2
- #define NOR_CMD_INDEX_ERASESECTOR 3
  3
- #define NOR_CMD_INDEX_PAGEPROGRAM 4
  4
- #define NOR_CMD_INDEX_CHIPERASE 5
  5
- #define NOR_CMD_INDEX_DUMMY 6
  6
- #define NOR_CMD_INDEX_ERASEBLOCK 7
  7
- #define NOR_CMD_LUT_SEQ_IDX_READ 0
  0 READ LUT sequence id in lookupTable stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_READSTATUS 1
  1 Read Status LUT sequence id in lookupTable stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_READSTATUS_XPI 2
  2 Read status DPI/QPI/OPI sequence id in lookupTable stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_WRITEENABLE 3
  3 Write Enable sequence id in lookupTable stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_WRITEENABLE_XPI 4
  4 Write Enable DPI/QPI/OPI sequence id in lookupTable stored in config block

- #define NOR_CMD_LUT_SEQ_IDX_ERASESECTOR 5
  5 Erase Sector sequence id in lookupTable stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_ERASEBLOCK 8
  8 Erase Block sequence id in lookupTable stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM 9
  9 Program sequence id in lookupTable stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_CHIPERASE 11
  11 Chip Erase sequence in lookupTable id stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_READ_SFDP 13
  13 Read SFDP sequence in lookupTable id stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_RESTORE_NOCMD 14
  14 Restore 0-4-4/0-8-8 mode sequence id in lookupTable stored in config block
- #define NOR_CMD_LUT_SEQ_IDX_EXIT_NOCMD 15
  15 Exit 0-4-4/0-8-8 mode sequence id in lookupTable stored in config blobk

Enumerations

- enum
  Flash Configuration Option0 device_type.
- enum
  Flash Configuration Option0 quad_mode_setting.
- enum
  Flash Configuration Option0 misc_mode.
- enum
  FLEXSPI_RESET_PIN boot configurations in OTP.
- enum
  Flash Configuration Option1 flash_connection.
- enum
  Flash Device Mode Configuration Sequence.
- enum
  Flash Config Mode Definition.
- enum {
  kFlexSpiDeviceType_SerialNOR = 1,
  kFlexSpiDeviceType_SerialNAND = 2,
  kFlexSpiDeviceType_SerialRAM = 3,
  kFlexSpiDeviceType_MCP_NOR_NAND = 0x12,
  kFlexSpiDeviceType_MCP_NOR_RAM = 0x13 }
  Flash Device Type Definition.
- enum
  Flash Pad Definitions.
- enum {
  kDeviceConfigCmdType_Generic,
  kDeviceConfigCmdType_QuadEnable,
  kDeviceConfigCmdType_Spi2Xpi,
  kDeviceConfigCmdType_Xpi2Spi,
  kDeviceConfigCmdType_Spi2NoCmd,
  kDeviceConfigCmdType_Reset }
  Flash Configuration Command Type.

- enum flexspi_operation_t {
  kFlexSpiOperation_Command = 0,
  kFlexSpiOperation_Config = 1,
  kFlexSpiOperation_Write = 2,
  kFlexSpiOperation_Read = 3,
  kFlexSpiOperation_End = kFlexSpiOperation_Read }
  FlexSPI Operation Type.

Functions

- status_t IAP_FlexspiNorInit (uint32_t instance, flexspi_nor_config_t *config)
  Initialize Serial NOR devices via FlexSPI.
- status_t IAP_FlexspiNorPageProgram (uint32_t instance, flexspi_nor_config_t *config, uint32_t dstAddr, const uint32_t *src)
  Program data to Serial NOR via FlexSPI.
- status_t IAP_FlexspiNorEraseAll (uint32_t instance, flexspi_nor_config_t *config)
  Erase all the Serial NOR devices connected on FlexSPI.
- status_t IAP_FlexspiNorErase (uint32_t instance, flexspi_nor_config_t *config, uint32_t start, uint32_t length)
  Erase Flash Region specified by address and length.
- status_t IAP_FlexspiNorEraseSector (uint32_t instance, flexspi_nor_config_t *config, uint32_t address)
  Erase one sector specified by address.
- status_t IAP_FlexspiNorEraseBlock (uint32_t instance, flexspi_nor_config_t *config, uint32_t address)
  Erase one block specified by address.
- status_t IAP_FlexspiNorGetConfig (uint32_t instance, flexspi_nor_config_t *config, serial_nor_config_option_t *option)
  Get FlexSPI NOR Configuration Block based on specified option.
- status_t IAP_FlexspiNorRead (uint32_t instance, flexspi_nor_config_t *config, uint32_t *dst, uint32_t start, uint32_t bytes)
  Read data from Flexspi NOR Flash.
- status_t IAP_FlexspiXfer (uint32_t instance, flexspi_xfer_t *xfer)
  Get FlexSPI Xfer data.
- status_t IAP_FlexspiUpdateLut (uint32_t instance, uint32_t seqIndex, const uint32_t *lutBase, uint32_t numberOfSeq)
  Update FlexSPI Lookup table.
- status_t IAP_FlexspiSetClockSource (uint32_t clockSrc)
  Set the clock source for FlexSPI.
- void IAP_FlexspiConfigClock (uint32_t instance, uint32_t freqOption, uint32_t sampleClkMode)
  Configure the flexspi interface clock frequency and data sample mode.
- status_t IAP_FlexspiNorAutoConfig (uint32_t instance, flexspi_nor_config_t *config, serial_nor_config_option_t *option)
  Configure flexspi nor automatically.

FlexSPI status.

- enum
  FlexSPI Driver status group.
- enum _flexspi_status {
  kStatus_FLEXSPI_Success = MAKE_STATUS(kStatusGroup_Generic, 0),
  kStatus_FLEXSPI_Fail = MAKE_STATUS(kStatusGroup_Generic, 1),
  kStatus_FLEXSPI_InvalidArgument = MAKE_STATUS(kStatusGroup_Generic, 4),
  kStatus_FLEXSPI_SequenceExecutionTimeout,
  kStatus_FLEXSPI_InvalidSequence = MAKE_STATUS(kStatusGroup_FlexSPI, 1),
  kStatus_FLEXSPI_DeviceTimeout = MAKE_STATUS(kStatusGroup_FlexSPI, 2),
  kStatus_FLEXSPINOR_ProgramFail,
  kStatus_FLEXSPINOR_EraseSectorFail,
  kStatus_FLEXSPINOR_EraseAllFail = MAKE_STATUS(kStatusGroup_FlexSPINOR, 2),
  kStatus_FLEXSPINOR_WaitTimeout = MAKE_STATUS(kStatusGroup_FlexSPINOR, 3) ,
  kStatus_FLEXSPINOR_WriteAlignmentError,
  kStatus_FLEXSPINOR_CommandFailure,
  kStatus_FLEXSPINOR_SFDP_NotFound = MAKE_STATUS(kStatusGroup_FlexSPINOR, 7),
  kStatus_FLEXSPINOR_Unsupported_SFDP_Version,
  kStatus_FLEXSPINOR_Flash_NotFound,
  kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed }
  FlexSPI Driver status.

24.4.2 Data Structure Documentation

24.4.2.1 struct serial_nor_config_option_t

24.4.2.2 union flash_run_context_t

24.4.2.3 struct flexspi_lut_seq_t

Data Fields

- uint8_t seqNum
  Sequence Number, valid number: 1-16.
- uint8_t seqId
  Sequence Index, valid number: 0-15.

24.4.2.4 struct flexspi_dll_time_t

24.4.2.5 struct flexspi_mem_config_block_t

Data Fields

- uint32_t tag

[0x000-0x003] Tag, fixed value 0x42464346UL

- uint32_t version

  [0x004-0x007] Version,[31:24] -'V', [23:16] - Major, [15:8] - Minor, [7:0] - bugfix
- uint32_t reserved0

  [0x008-0x00b] Reserved for future use
- uint8_t readSampleClkSrc

  [0x00c-0x00c] Read Sample Clock Source, valid value: 0/1/3
- uint8_t csHoldTime

  [0x00d-0x00d] CS hold time, default value: 3
- uint8_t csSetupTime

  [0x00e-0x00e] CS setup time, default value: 3
- uint8_t columnAddressWidth

  [0x00f-0x00f] Column Address with, for HyperBus protocol, it is fixed to 3, For
- uint8_t deviceModeCfgEnable

  Serial NAND, need to refer to datasheet.
- uint8_t deviceModeType

  [0x011-0x011] Specify the configuration command type:Quad Enable, DPI/QPI/OPI switch,
- uint16_t waitTimeCfgCommands

  Generic configuration, etc.
- flexspi_lut_seq_t deviceModeSeq

  DPI/QPI/OPI switch or reset command.
- uint32_t deviceModeArg

  sequence number, [31:16] Reserved
- uint8_t configCmdEnable

  [0x01c-0x01c] Configure command Enable Flag, 1 - Enable, 0 - Disable
- uint8_t configModeType [3]

  [0x01d-0x01f] Configure Mode Type, similar as deviceModeTpe
- flexspi_lut_seq_t configCmdSeqs [3]

  [0x020-0x02b] Sequence info for Device Configuration command, similar as deviceModeSeq
- uint32_t reserved1

  [0x02c-0x02f] Reserved for future use
- uint32_t configCmdArgs [3]

  [0x030-0x03b] Arguments/Parameters for device Configuration commands
- uint32_t reserved2

  [0x03c-0x03f] Reserved for future use
- uint32_t controllerMiscOption

  [0x040-0x043] Controller Misc Options, see Misc feature bit definitions for more
- uint8_t deviceType

  details
- uint8_t sflashPadType

  [0x045-0x045] Serial Flash Pad Type: 1 - Single, 2 - Dual, 4 - Quad, 8 - Octal
- uint8_t serialClkFreq

  [0x046-0x046] Serial Flash Frequencey, device specific definitions, See System Boot
- uint8_t lutCustomSeqEnable

  Chapter for more details.
- uint32_t reserved3 [2]

  be done using 1 LUT sequence, currently, only applicable to HyperFLASH
- uint32_t sflashA1Size

  [0x050-0x053] Size of Flash connected to A1
- uint32_t sflashA2Size

  [0x054-0x057] Size of Flash connected to A2

- uint32_t sflashB1Size
  [0x058-0x05b] Size of Flash connected to B1
- uint32_t sflashB2Size
  [0x05c-0x05f] Size of Flash connected to B2
- uint32_t csPadSettingOverride
  [0x060-0x063] CS pad setting override value
- uint32_t sclkPadSettingOverride
  [0x064-0x067] SCK pad setting override value
- uint32_t dataPadSettingOverride
  [0x068-0x06b] data pad setting override value
- uint32_t dqsPadSettingOverride
  [0x06c-0x06f] DQS pad setting override value
- uint32_t timeoutInMs
  [0x070-0x073] Timeout threshold for read status command
- uint32_t commandInterval
  [0x074-0x077] CS deselect interval between two commands
- flexspi_dll_time_t dataValidTime [2]
  [0x078-0x07b] CLK edge to data valid time for PORT A and PORT B
- uint16_t busyOffset
  [0x07c-0x07d] Busy offset, valid value: 0-31
- uint16_t busyBitPolarity
  [0x07e-0x07f] Busy flag polarity, 0 - busy flag is 1 when flash device is busy, 1 -
- uint32_t lookupTable [64]
  busy flag is 0 when flash device is busy
- flexspi_lut_seq_t lutCustomSeq [12]
  [0x180-0x1af] Customizable LUT Sequences
- uint32_t reserved4 [4]
  [0x1b0-0x1bf] Reserved for future use

Field Documentation

(1) uint8_t flexspi_mem_config_block_t::deviceModeCfgEnable

[0x010-0x010] Device Mode Configure enable flag, 1 - Enable, 0 - Disable

(2) uint16_t flexspi_mem_config_block_t::waitTimeCfgCommands

[0x012-0x013] Wait time for all configuration commands, unit: 100us, Used for

(3) flexspi_lut_seq_t flexspi_mem_config_block_t::deviceModeSeq

[0x014-0x017] Device mode sequence info, [7:0] - LUT sequence id, [15:8] - LUt

(4) uint32_t flexspi_mem_config_block_t::deviceModeArg

[0x018-0x01b] Argument/Parameter for device configuration

(5) uint8_t flexspi_mem_config_block_t::deviceType

[0x044-0x044] Device Type: See Flash Type Definition for more details

(6) uint8_t flexspi_mem_config_block_t::lutCustomSeqEnable

[0x047-0x047] LUT customization Enable, it is required if the program/erase cannot

(7) uint32_t flexspi_mem_config_block_t::reserved3[2]

[0x048-0x04f] Reserved for future use

(8) uint32_t flexspi_mem_config_block_t::lookupTable[64]

[0x080-0x17f] Lookup table holds Flash command sequences

24.4.2.6 struct flexspi_xfer_t

Data Fields

- flexspi_operation_t operation
  FlexSPI operation.
- uint32_t baseAddress
  FlexSPI operation base address.
- uint32_t seqId
  Sequence Id.
- uint32_t seqNum
  Sequence Number.
- bool isParallelModeEnable
  Is a parallel transfer.
- uint32_t * txBuffer
  Tx buffer.
- uint32_t txSize
  Tx size in bytes.
- uint32_t * rxBuffer
  Rx buffer.
- uint32_t rxSize
  Rx size in bytes.

24.4.2.7 struct flexspi_nor_config_t

Data Fields

- flexspi_mem_config_block_t memConfig
  Common memory configuration info via FlexSPI.
- uint32_t pageSize
  Page size of Serial NOR.
- uint32_t sectorSize
  Sector size of Serial NOR.
- uint8_t ipcmdSerialClkFreq
  Clock frequency for IP command.
- uint8_t isUniformBlockSize
  Sector/Block size is the same.

- uint8_t isDataOrderSwapped
  Data order (D0, D1, D2, D3) is swapped (D1,D0, D3, D2)
- uint8_t reserved0 [1]
  Reserved for future use.
- uint8_t serialNorType
  Serial NOR Flash type: 0/1/2/3.
- uint8_t needExitNoCmdMode
  Need to exit NoCmd mode before other IP command.
- uint8_t halfClkForNonReadCmd
  Half the Serial Clock for non-read command: true/false.
- uint8_t needRestoreNoCmdMode
  Need to Restore NoCmd mode after IP commmand execution.
- uint32_t blockSize
  Block size.
- uint32_t flashStateCtx
  Flash State Context.
- uint32_t reserve2 [10]
  Reserved for future use.

24.4.3 Macro Definition Documentation

24.4.3.1 #define NOR_CMD_INDEX_READ 0

0

24.4.4 Enumeration Type Documentation

24.4.4.1 anonymous enum

24.4.4.2 enum _flexspi_status

Enumerator

kStatus_FLEXSPI_Success API is executed successfully.
kStatus_FLEXSPI_Fail API is executed fails.
kStatus_FLEXSPI_InvalidArgument Invalid argument.
kStatus_FLEXSPI_SequenceExecutionTimeout The FlexSPI Sequence Execution timeout.
kStatus_FLEXSPI_InvalidSequence The FlexSPI LUT sequence invalid.
kStatus_FLEXSPI_DeviceTimeout The FlexSPI device timeout.
kStatus_FLEXSPINOR_ProgramFail Status for Page programming failure.
kStatus_FLEXSPINOR_EraseSectorFail Status for Sector Erase failure.
kStatus_FLEXSPINOR_EraseAllFail Status for Chip Erase failure.
kStatus_FLEXSPINOR_WaitTimeout Status for timeout.
kStatus_FLEXSPINOR_WriteAlignmentError Status for Alignement error.
kStatus_FLEXSPINOR_CommandFailure Status for Erase/Program Verify Error.
kStatus_FLEXSPINOR_SFDP_NotFound Status for SFDP read failure.

kStatus_FLEXSPINOR_Unsupported_SFDP_Version Status for Unrecognized SFDP version.
kStatus_FLEXSPINOR_Flash_NotFound Status for Flash detection failure.
kStatus_FLEXSPINOR_DTRRead_DummyProbeFailed Status for DDR Read dummy probe failure.

24.4.4.3 anonymous enum

24.4.4.4 anonymous enum

24.4.4.5 anonymous enum

24.4.4.6 anonymous enum

24.4.4.7 anonymous enum

Enumerator

kFlexSpiDeviceType_SerialNOR Flash devices are Serial NOR.
kFlexSpiDeviceType_SerialNAND Flash devices are Serial NAND.
kFlexSpiDeviceType_SerialRAM Flash devices are Serial RAM/HyperFLASH.
kFlexSpiDeviceType_MCP_NOR_NAND Flash device is MCP device, A1 is Serial NOR, A2 is Serial NAND.
kFlexSpiDeviceType_MCP_NOR_RAM Flash deivce is MCP device, A1 is Serial NOR, A2 is Serial RAMs.

24.4.4.8 anonymous enum

Enumerator

kDeviceConfigCmdType_Generic Generic command, for example: configure dummy cycles, drive strength, etc.
kDeviceConfigCmdType_QuadEnable Quad Enable command.
kDeviceConfigCmdType_Spi2Xpi Switch from SPI to DPI/QPI/OPI mode.
kDeviceConfigCmdType_Xpi2Spi Switch from DPI/QPI/OPI to SPI mode.
kDeviceConfigCmdType_Spi2NoCmd Switch to 0-4-4/0-8-8 mode.
kDeviceConfigCmdType_Reset Reset device command.

24.4.4.9 enum flexspi_operation_t

Enumerator

kFlexSpiOperation_Command FlexSPI operation: Only command, both TX and.
kFlexSpiOperation_Config RX buffer are ignored. FlexSPI operation: Configure device mode, the
kFlexSpiOperation_Write TX FIFO size is fixed in LUT. FlexSPI operation: Write, only TX buffer is

kFlexSpiOperation_Read effective FlexSPI operation: Read, only Rx Buffer is
kFlexSpiOperation_End effective.

24.4.5 Function Documentation

24.4.5.1 status_t IAP_FlexspiNorInit (uint32_t *instance*, flexspi_nor_config_t ∗ *config*)

This function configures the FlexSPI controller with the arguments pointed by param config.

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.2 status_t IAP_FlexspiNorPageProgram (uint32_t *instance*, flexspi_nor_config_t ∗ *config*, uint32_t *dstAddr*, const uint32_t ∗ *src*)

This function Program data to specified destination address.

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.
dstAddr	The destination address to be programmed.
src	Points to the buffer which hold the data to be programmed.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.3 status_t IAP_FlexspiNorEraseAll (uint32_t *instance*, flexspi_nor_config_t ∗ *config*)

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.4 status_t IAP_FlexspiNorErase (uint32_t *instance*, flexspi_nor_config_t * *config*, uint32_t *start*, uint32_t *length*)

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.
start	The start address to be erased.
length	The length to be erased.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.5 status_t IAP_FlexspiNorEraseSector (uint32_t *instance*, flexspi_nor_config_t * *config*, uint32_t *address*)

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.
address	The address of the sector to be erased.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.6 status_t IAP_FlexspiNorEraseBlock (uint32_t *instance*, flexspi_nor_config_t * *config*, uint32_t *address*)

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.
address	The address of the block to be erased.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.7 status_t IAP_FlexspiNorGetConfig (uint32_t *instance*, flexspi_nor_config_t * *config*, serial_nor_config_option_t * *option*)

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.
option	The Flash Configuration Option block. Refer to serial_nor_config_option_t.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.8 status_t IAP_FlexspiNorRead (uint32_t *instance*, flexspi_nor_config_t * *config*, uint32_t * *dst*, uint32_t *start*, uint32_t *bytes*)

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.
dst	Buffer address used to store the read data.
start	The Read address.
bytes	The Read size

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.9 status_t IAP_FlexspiXfer (uint32_t *instance*, flexspi_xfer_t * *xfer*)

Parameters

instance	FlexSPI controller instance, only support 0.
xfer	The FlexSPI Transfer Context block. Refer to flexspi_xfer_t.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.10 status_t IAP_FlexspiUpdateLut (uint32_t *instance*, uint32_t *seqIndex*, const uint32_t * *lutBase*, uint32_t *numberOfSeq*)

Parameters

instance	FlexSPI controller instance, only support 0.
seqIndex	The index of FlexSPI LUT to be updated.
lutBase	Points to the buffer which hold the LUT data to be programmed.
numberOfSeq	The number of LUT seq that need to be updated.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.11 status_t IAP_FlexspiSetClockSource (uint32_t *clockSrc*)

Parameters

clockSrc	Clock source for flexspi interface.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.12 void IAP_FlexspiConfigClock (uint32_t *instance*, uint32_t *freqOption*, uint32_t *sampleClkMode*)

Parameters

instance	FlexSPI controller instance, only support 0.
freqOption	FlexSPI interface clock frequency selection.
sampleClk-Mode	FlexSPI controller data sample mode.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.4.5.13 status_t IAP_FlexspiNorAutoConfig (uint32_t *instance*, flexspi_nor_config_t * *config*, serial_nor_config_option_t * *option*)

Parameters

instance	FlexSPI controller instance, only support 0.
config	The Flash configuration block. Refer to flexspi_nor_config_t.
option	The Flash Configuration Option block. Refer to serial_nor_config_option_t.

Returns

The status flags. This is a member of the enumeration _flexspi_status

24.5 IAP OTP Driver

24.5.1 Overview

Enumerations

- enum
  OTP Status Group.
- enum {
  kStatus_OTP_InvalidAddress = MAKE_STATUS(kStatusGroup_OtpGroup, 1),
  kStatus_OTP_ProgramFail = MAKE_STATUS(kStatusGroup_OtpGroup, 2),
  kStatus_OTP_CrcFail = MAKE_STATUS(kStatusGroup_OtpGroup, 3),
  kStatus_OTP_Error = MAKE_STATUS(kStatusGroup_OtpGroup, 4),
  kStatus_OTP_EccCheckFail = MAKE_STATUS(kStatusGroup_OtpGroup, 5),
  kStatus_OTP_Locked = MAKE_STATUS(kStatusGroup_OtpGroup, 6),
  kStatus_OTP_Timeout = MAKE_STATUS(kStatusGroup_OtpGroup, 7),
  kStatus_OTP_CrcCheckPass = MAKE_STATUS(kStatusGroup_OtpGroup, 8) }
  OTP Error Status definitions.

Functions

- status_t IAP_OtpInit (uint32_t src_clk_freq)
  Initialize OTP controller.
- status_t IAP_OtpDeinit (void)
  De-Initialize OTP controller.
- status_t IAP_OtpFuseRead (uint32_t addr, uint32_t *data)
  Read Fuse value from OTP Fuse Block.
- status_t IAP_OtpFuseProgram (uint32_t addr, uint32_t data, bool lock)
  Program value to OTP Fuse block.
- status_t IAP_OtpShadowRegisterReload (void)
  Reload all shadow registers from OTP fuse block.
- status_t IAP_OtpCrcCheck (uint32_t start_addr, uint32_t end_addr, uint32_t crc_addr)
  Do CRC Check via OTP controller.
- status_t IAP_OtpCrcCalc (uint32_t *src, uint32_t numberOfWords, uint32_t *crcChecksum)
  Calculate the CRC checksum for specified data for OTP.

24.5.2 Enumeration Type Documentation

24.5.2.1 anonymous enum

Enumerator

kStatus_OTP_InvalidAddress Invalid OTP address.
kStatus_OTP_ProgramFail Program Fail.
kStatus_OTP_CrcFail CrcCheck Fail.
kStatus_OTP_Error Errors happened during OTP operation.

kStatus_OTP_EccCheckFail Ecc Check failed during OTP operation.
kStatus_OTP_Locked OTP Fuse field has been locked.
kStatus_OTP_Timeout OTP operation time out.
kStatus_OTP_CrcCheckPass OTP CRC Check Pass.

24.5.3 Function Documentation

24.5.3.1 status_t IAP_OtpInit (uint32_t *src_clk_freq*)

This function enables OTP Controller clock.

Parameters

src_clk_freq	The Frequency of the source clock of OTP controller

Returns

kStatus_Success

24.5.3.2 status_t IAP_OtpDeinit (void)

This functin disables OTP Controller Clock.

Returns

kStatus_Success

24.5.3.3 status_t IAP_OtpFuseRead (uint32_t *addr,* uint32_t ∗ *data*)

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

Parameters

addr	Fuse address
data	Buffer to hold the data read from OTP Fuse block

Returns

kStatus_Success - Data read from OTP Fuse block successfully kStatus_InvalidArgument - data pointer is invalid kStatus_OTP_EccCheckFail - Ecc Check Failed kStatus_OTP_Error - Other Errors

24.5.3.4 status_t IAP_OtpFuseProgram (uint32_t *addr,* uint32_t *data,* bool *lock*)

This function program data to specified OTP Fuse address.

Parameters

addr	Fuse address
data	data to be programmed into OTP Fuse block
lock	lock the fuse field or not

Returns

kStatus_Success - Data has been programmed into OTP Fuse block successfully kStatus_OTP_-ProgramFail - Fuse programming failed kStatus_OTP_Locked - The address to be programmed into is locked kStatus_OTP_Error - Other Errors

24.5.3.5 status_t IAP_OtpShadowRegisterReload (void)

This function reloads all the shadow registers from OTP Fuse block

Returns

kStatus_Success - Shadow registers' reloadding succeeded. kStatus_OTP_EccCheckFail - Ecc Check Failed kStatus_OTP_Error - Other Errors

24.5.3.6 status_t IAP_OtpCrcCheck (uint32_t *start_addr,* uint32_t *end_addr,* uint32_t *crc_addr*)

This function checks whether data in specified fuse address ranges match the crc value in the specified CRC address and return the actual crc value as needed.

Parameters

start_addr	Start address of selected Fuse address range
end_addr	End address of selected Fuse address range
crc_addr	Address that hold CRC data

Returns

kStatus_Success CRC check succeeded, CRC value matched. kStatus_InvalidArgument - Invalid Argument kStatus_OTP_EccCheckFail Ecc Check Failed kStatus_OTP_CrcFail CRC Check Failed

24.5.3.7 status_t IAP_OtpCrcCalc (uint32_t ∗ *src,* uint32_t *numberOfWords,* uint32_t ∗ *crcChecksum*)

This function calculates the CRC checksum for specified data for OTP

Parameters

src	the source address of data
numberOf-Words	number of Fuse words
crcChecksum	Buffer to store the CRC checksum

Returns

kStatus_Success CRC checksum is computed successfully. kStatus_InvalidArgument - Invalid Argument

Chapter 25
INPUTMUX: Input Multiplexing Driver

25.1 Overview

The MCUXpresso SDK provides a driver for the Input multiplexing (INPUTMUX).

It configures the inputs to the pin interrupt block, DMA trigger, and frequency measure function. Once configured, the clock is not needed for the inputmux.

25.2 Input Multiplexing Driver operation

INPUTMUX_AttachSignal function configures the specified input

25.3 Typical use case

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

Files

- file fsl_inputmux.h
- file fsl_inputmux_connections.h

Macros

- #define SCT0_PMUX_ID 0x00U
  Periphinmux IDs.

Enumerations

- enum inputmux_connection_t {
  kINPUTMUX_Sct0PinInp0ToSct0 = 0U + (SCT0_PMUX_ID << PMUX_SHIFT) ,
  kINPUTMUX_DebugHaltedToSct0 = 23U + (SCT0_PMUX_ID << PMUX_SHIFT) ,
  kINPUTMUX_GpioPort1Pin31ToPintsel = 63U + (PINTSEL_PMUX_ID << PMUX_SHIFT) ,
  kINPUTMUX_FlexioToDspInterrupt = 40U + (DSP_INT_PMUX_ID << PMUX_SHIFT) ,
  kINPUTMUX_ClockOutToFreqmeas = 11U + (FREQMEAS_PMUX_ID << PMUX_SHIFT) ,
  kINPUTMUX_UsbFsToggleToSmartDmaInput = 75U + (SMART_DMA_PMUX_ID << PMUX-_SHIFT) ,
  kINPUTMUX_Usb1FrameToggleToTimer0CaptureChannels = 18U + (CT32BIT0_CAP_PMUX_-ID << PMUX_SHIFT) ,
  kINPUTMUX_Usb1FrameToggleToTimer1CaptureChannels = 18U + (CT32BIT1_CAP_PMUX_-ID << PMUX_SHIFT) ,
  kINPUTMUX_Usb1FrameToggleToTimer2CaptureChannels = 18U + (CT32BIT2_CAP_PMUX_-ID << PMUX_SHIFT) ,
  kINPUTMUX_Usb1FrameToggleToTimer3CaptureChannels = 18U + (CT32BIT3_CAP_PMUX_-

ID << PMUX_SHIFT) ,
kINPUTMUX_Usb1FrameToggleToTimer4CaptureChannels = 18U + (CT32BIT4_CAP_PMUX_-ID << PMUX_SHIFT) ,
kINPUTMUX_Flexspi1TxToDma0 = 26U + (DMA0_ITRIG_PMUX_ID << PMUX_SHIFT) ,
kINPUTMUX_Flexspi1TxToDma1 = 26U + (DMA1_ITRIG_PMUX_ID << PMUX_SHIFT) ,
kINPUTMUX_Dma0OtrigChannel36ToTriginChannels = 36U + (DMA0_OTRIG_PMUX_ID << PMUX_SHIFT) }
INPUTMUX connections type.

- enum inputmux_signal_t {
kINPUTMUX_Dmac0InputTriggerPint0Ena = 0U + (DMA0_ITRIG_EN0_ID << ENA_SHIFT) ,
kINPUTMUX_Dmac0InputTriggerFlexspi1TxEna = 26U + (DMA0_ITRIG_EN0_ID << ENA_-SHIFT) ,
kINPUTMUX_Dmac1InputTriggerFlexspi1TxEna = 26U + (DMA1_ITRIG_EN0_ID << ENA_-SHIFT) ,
kINPUTMUX_HashCryptToDmac0Ch36RequestEna = 4U + ((DMA0_REQ_ENA0_ID + 4) << ENA_SHIFT) }
INPUTMUX signal enable/disable type.

Functions

- void INPUTMUX_Init (INPUTMUX_Type ∗base)
Initialize INPUTMUX peripheral.
- void INPUTMUX_AttachSignal (INPUTMUX_Type ∗base, uint32_t index, inputmux_connection-_t connection)
Attaches a signal.
- void INPUTMUX_EnableSignal (INPUTMUX_Type ∗base, inputmux_signal_t signal, bool enable)
Enable/disable a signal.
- void INPUTMUX_Deinit (INPUTMUX_Type ∗base)
Deinitialize INPUTMUX peripheral.

Driver version

- #define FSL_INPUTMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
Group interrupt driver version for SDK.

25.4 Enumeration Type Documentation

25.4.1 enum inputmux_connection_t

Enumerator

kINPUTMUX_Sct0PinInp0ToSct0 SCT INMUX.
kINPUTMUX_DebugHaltedToSct0 Pin Interrupt.
kINPUTMUX_GpioPort1Pin31ToPintsel DSP Interrupt.
kINPUTMUX_FlexioToDspInterrupt Frequency measure.
kINPUTMUX_ClockOutToFreqmeas SMARTDMA input mux.
kINPUTMUX_UsbFsToggleToSmartDmaInput CTmier0 capture input mux.
kINPUTMUX_Usb1FrameToggleToTimer0CaptureChannels CTmier1 capture input mux.

kINPUTMUX_Usb1FrameToggleToTimer1CaptureChannels CTmier2 capture input mux.
kINPUTMUX_Usb1FrameToggleToTimer2CaptureChannels CTmier3 capture input mux.
kINPUTMUX_Usb1FrameToggleToTimer3CaptureChannels CTmier4 capture input mux.
kINPUTMUX_Usb1FrameToggleToTimer4CaptureChannels DMA0 ITRIG.
kINPUTMUX_Flexspi1TxToDma0 DMA1 ITRIG.
kINPUTMUX_Flexspi1TxToDma1 DMA0 OTRIG.
kINPUTMUX_Dma0OtrigChannel36ToTriginChannels DMA1 OTRIG.

25.4.2 enum inputmux_signal_t

Enumerator

kINPUTMUX_Dmac0InputTriggerPint0Ena DMA0 input trigger source enable.
kINPUTMUX_Dmac0InputTriggerFlexspi1TxEna DMA1 input trigger source enable.
kINPUTMUX_Dmac1InputTriggerFlexspi1TxEna DMA0 REQ signal.
kINPUTMUX_HashCryptToDmac0Ch36RequestEna DMA1 REQ signal.

25.5 Function Documentation

25.5.1 void INPUTMUX_Init (INPUTMUX_Type ∗ *base*)

This function enables the INPUTMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.

Return values

None.	

25.5.2 void INPUTMUX_AttachSignal (INPUTMUX_Type ∗ *base,* uint32_t *index,* inputmux_connection_t *connection*)

This function gates the INPUTPMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.
index	Destination peripheral to attach the signal to.
connection	Selects connection.

Return values

None.	

25.5.3 void INPUTMUX_EnableSignal (INPUTMUX_Type * *base*, inputmux_signal_t *signal*, bool *enable*)

This function gates the INPUTPMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.
signal	Enable signal register id and bit offset.
enable	Selects enable or disable.

Return values

None.	

25.5.4 void INPUTMUX_Deinit (INPUTMUX_Type * *base*)

This function disables the INPUTMUX clock.

Parameters

base	Base address of the INPUTMUX peripheral.

Return values

None.	

Chapter 26
LCDIF: LCD interface

26.1 Overview

The MCUXpresso SDK provides a peripheral driver for the LCD interface (LCDIF) module of MCUXpresso SDK devices.

Data Structures

- struct lcdif_dpi_config_t
  Configuration for LCDIF module to work in DBI mode. More...
- struct lcdif_fb_config_t
  LCDIF frame buffer configuration. More...
- struct lcdif_cursor_config_t
  LCDIF cursor configuration. More...
- struct lcdif_dither_config_t
  LCDIF dither configuration. More...

Macros

- #define LCDIF_MAKE_CURSOR_COLOR(r, g, b) (((r) << 16U) | ((g) << 8U) | ((b) << 0U))
  Construct the cursor color, every element should be in the range of 0 ∼ 255.
- #define LCDIF_MAKE_GAMMA_VALUE(r, g, b) (((r) << 16U) | ((g) << 8U) | ((b) << 0U))
  Construct the gamma value set to LCDIF gamma table, every element should be in the range of 0∼255.
- #define LCDIF_FB_ALIGN 128U
  The frame buffer should be 128 byte aligned.
- #define LCDIF_GAMMA_INDEX_MAX 256U
  Gamma index max value.
- #define LCDIF_CURSOR_SIZE 32U
  The cursor size is 32 x 32.

Enumerations

- enum _lcdif_polarity_flags {
  kLCDIF_VsyncActiveLow = 0U,
  kLCDIF_VsyncActiveHigh = (1U << 0U),
  kLCDIF_HsyncActiveLow = 0U,
  kLCDIF_HsyncActiveHigh = (1U << 1U),
  kLCDIF_DataEnableActiveLow = 0U,
  kLCDIF_DataEnableActiveHigh = (1U << 2U),
  kLCDIF_DriveDataOnFallingClkEdge = 0U,
  kLCDIF_DriveDataOnRisingClkEdge = (1U << 3U) }
  LCDIF signal polarity flags.

- enum lcdif_output_format_t {
  kLCDIF_Output16BitConfig1 = 0U,
  kLCDIF_Output16BitConfig2 = 1U,
  kLCDIF_Output16BitConfig3 = 2U,
  kLCDIF_Output18BitConfig1 = 3U,
  kLCDIF_Output18BitConfig2 = 4U,
  kLCDIF_Output24Bit = 5U }
  LCDIF DPI output format.
- enum lcdif_fb_format_t {
  kLCDIF_PixelFormatXRGB444 = 1,
  kLCDIF_PixelFormatXRGB4444 = 1,
  kLCDIF_PixelFormatXRGB1555 = 2,
  kLCDIF_PixelFormatRGB565 = 3,
  kLCDIF_PixelFormatXRGB8888 = 4 }
  LCDIF frame buffer pixel format.
- enum _lcdif_interrupt { kLCDIF_Display0FrameDoneInterrupt }
  LCDIF interrupt and status.
- enum lcdif_cursor_format_t {
  kLCDIF_CursorMasked = 1,
  kLCDIF_CursorARGB8888 = 2 }
  LCDIF cursor format.

Functions

- void LCDIF_DpiModeGetDefaultConfig (lcdif_dpi_config_t *config)
  Get the default configuration for to initialize the LCDIF.
- status_t LCDIF_DpiModeSetConfig (LCDIF_Type *base, uint8_t displayIndex, const lcdif_dpi_-config_t *config)
  Initialize the LCDIF to work in DPI mode.

Driver version

- #define **FSL_LCDIF_DRIVER_VERSION** (MAKE_VERSION(2, 1, 1))

Initialization and deinitialization

- status_t LCDIF_Init (LCDIF_Type *base)
  Initialize the LCDIF.
- void LCDIF_Deinit (LCDIF_Type *base)
  De-initialize the LCDIF.

Frame buffer

- void LCDIF_SetFrameBufferConfig (LCDIF_Type *base, uint8_t fbIndex, const lcdif_fb_config_t *config)
  Configure the LCDIF frame buffer.
- void LCDIF_FrameBufferGetDefaultConfig (lcdif_fb_config_t *config)
  Get default frame buffer configuration.

- static void LCDIF_SetFrameBufferAddr (LCDIF_Type ∗base, uint8_t fbIndex, uint32_t address)
  Set the frame buffer to LCDIF.
- void LCDIF_SetFrameBufferStride (LCDIF_Type ∗base, uint8_t fbIndex, uint32_t strideBytes)
  Set the frame buffer stride.

Dither

- void LCDIF_SetDitherConfig (LCDIF_Type ∗base, uint8_t displayIndex, const lcdif_dither_config_t ∗config)
  Set the dither configuration.

Gamma correction

- void LCDIF_SetGammaData (LCDIF_Type ∗base, uint8_t fbIndex, uint16_t startIndex, const uint32_t ∗gamma, uint16_t gammaLen)
  Set the gamma translation values to the LCDIF gamma table.

Interrupts

The interrupt must be enabled, otherwise the interrupt flags will not assert.

- static void LCDIF_EnableInterrupts (LCDIF_Type ∗base, uint32_t mask)
  Enables LCDIF interrupt requests.
- static void LCDIF_DisableInterrupts (LCDIF_Type ∗base, uint32_t mask)
  Disable LCDIF interrupt requests.
- static uint32_t LCDIF_GetAndClearInterruptPendingFlags (LCDIF_Type ∗base)
  Get and clear LCDIF interrupt pending status.

Cursor

Top-left point and Hot spot are two different cursor point.

Top-left point is used as the base address for the cursor. Hot spot is used in a search for a corresponding screen coordinate when a selection is made, such as when the "Enter" key or left mouse button is pushed.

```
Top-left point
     +-------------------------------+
     |                               |
     |   Hot spot                    |
     |      +-------                 |
     |      |    \                   |
     |      | \   \                  |
     |      |  \   \                 |
     |           \   \                |
     |            \   \               |
     |             \   \              |
     |              \   \             |
     |               \                |
     |                                |
     +-------------------------------+
```

For format masked, one cursor pixel is 2bits. 32x32 cursor pixels have 32 cursor color rows. Each cursor color row is 64bits.

cursorColorRow_H[31:0] = colorRow[63:32] cursorColorRow_L[31:0] = colorRow[31:0] xorCursor = cursorColorRow_H[cursorXPos[4:0]] andCursor = cursorColorRow_L[cursorXPos[4:0]]

The output cursor color is:

andCursor xorCursor cursor color 0 0 Background register color 0 1 Foreground register color 1 0 Frame buffer pixel color 1 1 Invert frame buffer pixel color

- void LCDIF_CursorGetDefaultConfig (lcdif_cursor_config_t *config)
  Get the hardware cursor default configuration.
- void LCDIF_SetCursorConfig (LCDIF_Type *base, const lcdif_cursor_config_t *config)
  Configure the cursor.
- static void LCDIF_SetCursorHotspotPosition (LCDIF_Type *base, uint16_t x, uint16_t y)
  Set the cursor hotspot postion.
- static void LCDIF_SetCursorBufferAddress (LCDIF_Type *base, uint32_t address)
  Set the cursor memory address.
- void LCDIF_SetCursorColor (LCDIF_Type *base, uint32_t background, uint32_t foreground)
  Set the cursor color.

26.2 Data Structure Documentation

26.2.1 struct lcdif_dpi_config_t

Data Fields

- uint16_t panelWidth
  Display panel width, pixels per line.
- uint16_t panelHeight
  Display panel height, how many lines per panel.
- uint8_t hsw
  HSYNC pulse width.
- uint8_t hfp
  Horizontal front porch.
- uint8_t hbp
  Horizontal back porch.
- uint8_t vsw
  VSYNC pulse width.
- uint8_t vfp
  Vrtical front porch.
- uint8_t vbp
  Vertical back porch.
- uint32_t polarityFlags
  OR'ed value of _lcdif_polarity_flags, used to contol the signal polarity.
- lcdif_output_format_t format
  DPI output format.

Field Documentation

(1) uint16_t lcdif_dpi_config_t::panelWidth

(2) uint16_t lcdif_dpi_config_t::panelHeight

(3) uint8_t lcdif_dpi_config_t::hsw

(4) uint8_t lcdif_dpi_config_t::hfp

(5) uint8_t lcdif_dpi_config_t::hbp

(6) uint8_t lcdif_dpi_config_t::vsw

(7) uint8_t lcdif_dpi_config_t::vfp

(8) uint8_t lcdif_dpi_config_t::vbp

(9) uint32_t lcdif_dpi_config_t::polarityFlags

(10) lcdif_output_format_t lcdif_dpi_config_t::format

26.2.2 struct lcdif_fb_config_t

Data Fields

- bool enable
  Enable the frame buffer output.
- bool enableGamma
  Enable the gamma correction.
- lcdif_fb_format_t format
  Frame buffer pixel format.

Field Documentation

(1) bool lcdif_fb_config_t::enable

(2) bool lcdif_fb_config_t::enableGamma

(3) lcdif_fb_format_t lcdif_fb_config_t::format

26.2.3 struct lcdif_cursor_config_t

Data Fields

- bool enable
  Enable the cursor or not.
- lcdif_cursor_format_t format
  Cursor format.
- uint8_t hotspotOffsetX
  Offset of the hotspot to top left point, range 0 ∼ 31.

- uint8_t hotspotOffsetY
  Offset of the hotspot to top left point, range 0 ∼ 31.

Field Documentation

(1) bool lcdif_cursor_config_t::enable

(2) lcdif_cursor_format_t lcdif_cursor_config_t::format

26.2.4 struct lcdif_dither_config_t

1. Decide which bit of pixel color to enhance. This is configured by the redSize, greenSize, and blueSize. For example, setting redSize=6 means it is the 6th bit starting from the MSB that we want to enhance, in other words, it is the RedColor[2]bit from RedColor[7:0]. greenSize and blueSize function in the same way.
2. Create the look-up table. a. The Look-Up Table includes 16 entries, 4 bits for each. b. The Look-Up Table provides a value U[3:0] through the index X[1:0] and Y[1:0]. c. The color value RedColor[3:0] is used to compare with this U[3:0]. d. If RedColor[3:0] > U[3:0], and RedColor[7:2] is not 6'b111111, then the final color value is: NewRedColor = RedColor[7:2] + 1'b1. e. If RedColor[3:0] <= U[3:0], then NewRedColor = RedColor[7:2].

Data Fields

- bool enable
  Enable or not.
- uint8_t redSize
  Red color size, valid region 4-8.
- uint8_t greenSize
  Green color size, valid region 4-8.
- uint8_t blueSize
  Blue color size, valid region 4-8.
- uint32_t low
  Low part of the look up table.
- uint32_t high
  High part of the look up table.

Field Documentation

(1) bool lcdif_dither_config_t::enable

(2) uint8_t lcdif_dither_config_t::redSize

(3) uint8_t lcdif_dither_config_t::greenSize

(4) uint8_t lcdif_dither_config_t::blueSize

(5) uint32_t lcdif_dither_config_t::low

(6) uint32_t lcdif_dither_config_t::high

26.3 Macro Definition Documentation

26.3.1 #define LCDIF_MAKE_CURSOR_COLOR(*r, g, b*) (((r) << 16U) | ((g) << 8U) | ((b) << 0U))

26.3.2 #define LCDIF_MAKE_GAMMA_VALUE(*r, g, b*) (((r) << 16U) | ((g) << 8U) | ((b) << 0U))

26.3.3 #define LCDIF_FB_ALIGN 128U

26.3.4 #define LCDIF_GAMMA_INDEX_MAX 256U

26.4 Enumeration Type Documentation

26.4.1 enum _lcdif_polarity_flags

Enumerator

kLCDIF_VsyncActiveLow VSYNC active low.
kLCDIF_VsyncActiveHigh VSYNC active high.
kLCDIF_HsyncActiveLow HSYNC active low.
kLCDIF_HsyncActiveHigh HSYNC active high.
kLCDIF_DataEnableActiveLow Data enable line active low.
kLCDIF_DataEnableActiveHigh Data enable line active high.
kLCDIF_DriveDataOnFallingClkEdge Drive data on falling clock edge, capture data on rising clock edge.
kLCDIF_DriveDataOnRisingClkEdge Drive data on falling clock edge, capture data on rising clock edge.

26.4.2 enum lcdif_output_format_t

Enumerator

kLCDIF_Output16BitConfig1 16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.
kLCDIF_Output16BitConfig2 16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.
kLCDIF_Output16BitConfig3 16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.
kLCDIF_Output18BitConfig1 18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.

kLCDIF_Output18BitConfig2 18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.
kLCDIF_Output24Bit 24-bit.

26.4.3 enum lcdif_fb_format_t

Enumerator

kLCDIF_PixelFormatXRGB444 XRGB4444, deprecated, use kLCDIF_PixelFormatXRGB4444 instead.
kLCDIF_PixelFormatXRGB4444 XRGB4444, 16-bit each pixel, 4-bit each element. R4G4B4 in reference manual.
kLCDIF_PixelFormatXRGB1555 XRGB1555, 16-bit each pixel, 5-bit each element. R5G5B5 in reference manual.
kLCDIF_PixelFormatRGB565 RGB565, 16-bit each pixel. R5G6B5 in reference manual.
kLCDIF_PixelFormatXRGB8888 XRGB8888, 32-bit each pixel, 8-bit each element. R8G8B8 in reference manual.

26.4.4 enum _lcdif_interrupt

Enumerator

kLCDIF_Display0FrameDoneInterrupt The last pixel of visible area in frame is shown.

26.4.5 enum lcdif_cursor_format_t

Enumerator

kLCDIF_CursorMasked Masked format.
kLCDIF_CursorARGB8888 ARGB8888.

26.5 Function Documentation

26.5.1 status_t LCDIF_Init (LCDIF_Type * *base*)

This function initializes the LCDIF to work.

Parameters

base	LCDIF peripheral base address.

Return values

kStatus_Success	Initialize successfully.

26.5.2 void LCDIF_Deinit (LCDIF_Type * *base*)

This function disables the LCDIF peripheral clock.

Parameters

base	LCDIF peripheral base address.

26.5.3 void LCDIF_DpiModeGetDefaultConfig (lcdif_dpi_config_t * *config*)

The default configuration value is:

```
   config->panelWidth = 0;
   config->panelHeight = 0;
   config->hsw = 0;
   config->hfp = 0;
   config->hbp = 0;
   config->vsw = 0;
   config->vfp = 0;
   config->vbp = 0;
   config->polarityFlags = kLCDIF_VsyncActiveLow |
      kLCDIF_HsyncActiveLow | kLCDIF_DataEnableActiveHigh |
kLCDIF_DriveDataOnFallingClkEdge; config->format =
      kLCDIF_Output24Bit;
```

Parameters

config	Pointer to the LCDIF configuration.

26.5.4 status_t LCDIF_DpiModeSetConfig (LCDIF_Type * *base*, uint8_t *displayIndex*, const lcdif_dpi_config_t * *config*)

This function configures the LCDIF DPI display.

Parameters

base	LCDIF peripheral base address.
displayIndex	Display index.
config	Pointer to the configuration structure.

Return values

kStatus_Success	Initialize successfully.
kStatus_InvalidArgument	Initialize failed because of invalid argument.

26.5.5 void LCDIF_SetFrameBufferConfig (LCDIF_Type ∗ *base,* uint8_t *fbIndex,* const lcdif_fb_config_t ∗ *config*)

Parameters

base	LCDIF peripheral base address.
fbIndex	Frame buffer index.
config	Pointer to the configuration structure.

26.5.6 void LCDIF_FrameBufferGetDefaultConfig (lcdif_fb_config_t ∗ *config*)

The default configuration is config->enable = true; config->enableGamma = false; config->format = kLCDIF_PixelFormatRGB565;

Parameters

config	Pointer to the configuration structure.

26.5.7 static void LCDIF_SetFrameBufferAddr (LCDIF_Type ∗ *base,* uint8_t *fbIndex,* uint32_t *address*) `[inline], [static]`

Parameters

base	LCDIF peripheral base address.
fbIndex	Frame buffer index.
address	Frame buffer address.

Note

The address must be 128 bytes aligned.

26.5.8 void LCDIF_SetFrameBufferStride (LCDIF_Type ∗ *base,* uint8_t *fbIndex,* uint32_t *strideBytes*)

Parameters

base	LCDIF peripheral base address.
fbIndex	Frame buffer index.
strideBytes	The stride in byte.

26.5.9 void LCDIF_SetDitherConfig (LCDIF_Type ∗ *base,* uint8_t *displayIndex,* const lcdif_dither_config_t ∗ *config*)

Parameters

base	LCDIF peripheral base address.
displayIndex	Index to configure.
config	Pointer to the configuration structure.

26.5.10 void LCDIF_SetGammaData (LCDIF_Type ∗ *base,* uint8_t *fbIndex,* uint16_t *startIndex,* const uint32_t ∗ *gamma,* uint16_t *gammaLen*)

Parameters

base	LCDIF peripheral base address.
fbIndex	The frame buffer index.
startIndex	Start index in the gamma table that the value will be set to.
gamma	The gamma values to set to the gamma table in LCDIF, could be defined using LCDIF_MAKE_GAMMA_VALUE.
gammaLen	The length of the `gamma`.

26.5.11 static void LCDIF_EnableInterrupts (LCDIF_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LCDIF peripheral base address.
mask	The interrupts to enable, pass in as OR'ed value of _lcdif_interrupt.

26.5.12 static void LCDIF_DisableInterrupts (LCDIF_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LCDIF peripheral base address.
mask	The interrupts to disable, pass in as OR'ed value of _lcdif_interrupt.

26.5.13 static uint32_t LCDIF_GetAndClearInterruptPendingFlags (LCDIF_Type * *base*) [inline], [static]

Parameters

base	LCDIF peripheral base address.

Returns

The interrupt pending status.

Note

The interrupt must be enabled, otherwise the interrupt flags will not assert.

26.5.14 void LCDIF_CursorGetDefaultConfig (lcdif_cursor_config_t * *config*)

The default configuration values are:

```
config->enable = true;
config->format = kLCDIF_CursorMasked;
config->hotspotOffsetX = 0;
config->hotspotOffsetY = 0;
```

Parameters

config	Pointer to the hardware cursor configuration structure.

26.5.15 void LCDIF_SetCursorConfig (LCDIF_Type * *base*, const lcdif_cursor_config_t * *config*)

Parameters

base	LCDIF peripheral base address.
config	Cursor configuration.

26.5.16 static void LCDIF_SetCursorHotspotPosition (LCDIF_Type * *base*, uint16_t *x*, uint16_t *y*) [inline], [static]

Parameters

base	LCDIF peripheral base address.
x	X coordinate of the hotspot, range 0 ∼ 8191.
y	Y coordinate of the hotspot, range 0 ∼ 8191.

26.5.17 static void LCDIF_SetCursorBufferAddress (LCDIF_Type * *base*, uint32_t *address*) [inline], [static]

Parameters

base	LCDIF peripheral base address.
address	Memory address.

26.5.18 void LCDIF_SetCursorColor (LCDIF_Type * *base*, uint32_t *background*, uint32_t *foreground*)

Parameters

base	LCDIF peripheral base address.
background	Background color, could be defined use LCDIF_MAKE_CURSOR_COLOR
foreground	Foreground color, could be defined use LCDIF_MAKE_CURSOR_COLOR

Chapter 27
LPADC: 12-bit SAR Analog-to-Digital Converter Driver

27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit SAR Analog-to-Digital Converter (LPADC) module of MCUXpresso SDK devices.

27.2 Typical use case

27.2.1 Polling Configuration

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

27.2.2 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_ResultFIFO0OverflowFlag = ADC_STAT_FOF0_MASK,
  kLPADC_ResultFIFO0ReadyFlag = ADC_STAT_RDY0_MASK,
  kLPADC_ResultFIFO1OverflowFlag = ADC_STAT_FOF1_MASK,
  kLPADC_ResultFIFO1ReadyFlag = ADC_STAT_RDY1_MASK }
  Define hardware flags of the module.
- enum _lpadc_interrupt_enable {
  kLPADC_ResultFIFO0OverflowInterruptEnable = ADC_IE_FOFIE0_MASK,
  kLPADC_FIFO0WatermarkInterruptEnable = ADC_IE_FWMIE0_MASK,
  kLPADC_ResultFIFO1OverflowInterruptEnable = ADC_IE_FOFIE1_MASK,
  kLPADC_FIFO1WatermarkInterruptEnable = ADC_IE_FWMIE1_MASK }
  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,
  kLPADC_TriggerPriorityPreemptSubsequently = 2U }
  Define enumeration of trigger priority policy.

Driver version

- #define FSL_LPADC_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))
  LPADC driver version 2.5.1.

Initialization & de-initialization.

- void LPADC_Init (ADC_Type ∗base, const lpadc_config_t ∗config)
  Initializes the LPADC module.
- void LPADC_GetDefaultConfig (lpadc_config_t ∗config)
  Gets an available pre-defined settings for initial configuration.
- void LPADC_Deinit (ADC_Type ∗base)
  De-initializes the LPADC module.
- static void LPADC_Enable (ADC_Type ∗base, bool enable)
  Switch on/off the LPADC module.
- static void LPADC_DoResetFIFO0 (ADC_Type ∗base)
  Do reset the conversion FIFO0.
- static void LPADC_DoResetFIFO1 (ADC_Type ∗base)
  Do reset the conversion FIFO1.
- static void LPADC_DoResetConfig (ADC_Type ∗base)
  Do reset the module's configuration.

Status

- static uint32_t LPADC_GetStatusFlags (ADC_Type ∗base)
  Get status flags.
- static void LPADC_ClearStatusFlags (ADC_Type ∗base, uint32_t mask)
  Clear status flags.

Interrupts

- static void LPADC_EnableInterrupts (ADC_Type *base, uint32_t mask)
  Enable interrupts.
- static void LPADC_DisableInterrupts (ADC_Type *base, uint32_t mask)
  Disable interrupts.

DMA Control

- static void LPADC_EnableFIFO0WatermarkDMA (ADC_Type *base, bool enable)
  Switch on/off the DMA trigger for FIFO0 watermark event.
- static void LPADC_EnableFIFO1WatermarkDMA (ADC_Type *base, bool enable)
  Switch on/off the DMA trigger for FIFO1 watermark event.

Trigger and conversion with FIFO.

- static uint32_t LPADC_GetConvResultCount (ADC_Type *base, uint8_t index)
  Get the count of result kept in conversion FIFOn.
- bool LPADC_GetConvResult (ADC_Type *base, lpadc_conv_result_t *result, uint8_t index)
  brief Get the result in conversion FIFOn.
- 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.

27.3 Data Structure Documentation

27.3.1 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_t.
- bool enableConvPause
  Enables the ADC pausing function.
- uint32_t convPauseDelay
  Controls the duration of pausing during command execution sequencing.
- uint32_t FIFO0Watermark
  FIFO0Watermark is a programmable threshold setting.
- uint32_t FIFO1Watermark
  FIFO1Watermark is a programmable threshold setting.

Field Documentation

(1) bool lpadc_config_t::enableInDozeMode

When enabled in Doze mode, immediate entries to Wait or Stop are allowed. When disabled, the ADC will wait for the current averaging iteration/FIFO storage to complete before acknowledging stop or wait mode entry.

(2) bool lpadc_config_t::enableAnalogPreliminary

(3) uint32_t lpadc_config_t::powerUpDelay

The startup delay count of (powerUpDelay $*$ 4) ADCK cycles must result in a longer delay than the analog startup time.

(4) lpadc_reference_voltage_source_t lpadc_config_t::referenceVoltageSource

(5) lpadc_power_level_mode_t lpadc_config_t::powerLevelMode

(6) lpadc_trigger_priority_policy_t lpadc_config_t::triggerPriorityPolicy

(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.

(8) uint32_t lpadc_config_t::convPauseDelay

The pause delay is a count of (convPauseDelay$*$4) ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

(9) uint32_t lpadc_config_t::FIFO0Watermark

When the number of datawords stored in the ADC Result FIFO0 is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

(10) uint32_t lpadc_config_t::FIFO1Watermark

When the number of datawords stored in the ADC Result FIFO1 is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

27.3.2 struct lpadc_conv_command_config_t

Data Fields

- lpadc_sample_scale_mode_t sampleScaleMode
  Sample scale mode.
- lpadc_sample_channel_mode_t sampleChannelMode
  Channel sample mode.
- uint32_t channelNumber
  Channel number, select the channel or channel pair.
- uint32_t chainedNextCommandNumber
  Selects the next command to be executed after this command completes.
- bool enableAutoChannelIncrement
  Loop with increment: when disabled, the "loopCount" field selects the number of times the selected channel is converted consecutively; when enabled, the "loopCount" field defines how many consecutive channels are converted as part of the command execution.
- uint32_t loopCount
  Selects how many times this command executes before finish and transition to the next command or Idle state.
- lpadc_hardware_average_mode_t hardwareAverageMode
  Hardware average selection.
- lpadc_sample_time_mode_t sampleTimeMode
  Sample time selection.
- lpadc_hardware_compare_mode_t hardwareCompareMode
  Hardware compare selection.
- uint32_t hardwareCompareValueHigh
  Compare Value High.
- uint32_t hardwareCompareValueLow
  Compare Value Low.
- bool enableWaitTrigger
  Wait for trigger assertion before execution: when disabled, this command will be automatically executed; when enabled, the active trigger must be asserted again before executing this command.

Field Documentation

(1) lpadc_sample_scale_mode_t lpadc_conv_command_config_t::sampleScaleMode

(2) lpadc_sample_channel_mode_t lpadc_conv_command_config_t::sampleChannelMode

(3) uint32_t lpadc_conv_command_config_t::channelNumber

(4) uint32_t lpadc_conv_command_config_t::chainedNextCommandNumber

1-15 is available, 0 is to terminate the chain after this command.

(5) bool lpadc_conv_command_config_t::enableAutoChannelIncrement

(6) uint32_t lpadc_conv_command_config_t::loopCount

Command executes LOOP+1 times. 0-15 is available.

(7) lpadc_hardware_average_mode_t lpadc_conv_command_config_t::hardwareAverageMode

(8) lpadc_sample_time_mode_t lpadc_conv_command_config_t::sampleTimeMode

(9) lpadc_hardware_compare_mode_t lpadc_conv_command_config_t::hardwareCompareMode

(10) uint32_t lpadc_conv_command_config_t::hardwareCompareValueHigh

The available value range is in 16-bit.

(11) uint32_t lpadc_conv_command_config_t::hardwareCompareValueLow

The available value range is in 16-bit.

(12) bool lpadc_conv_command_config_t::enableWaitTrigger

27.3.3 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.

Field Documentation

(1) uint32_t lpadc_conv_trigger_config_t::targetCommandId

(2) uint32_t lpadc_conv_trigger_config_t::delayPower

When this field is clear, then no delay is incurred. When this field is set to a non-zero value, the duration for the delay is 2^delayPower ADCK cycles. The available value range is 4-bit.

(3) uint32_t lpadc_conv_trigger_config_t::priority

If two or more triggers have the same priority level setting, the lower order trigger event has the higher priority. The lower value for this field is for the higher priority, the available value range is 1-bit.

(4) bool lpadc_conv_trigger_config_t::enableHardwareTrigger

THe software trigger is always available.

27.3.4 struct lpadc_conv_result_t

Data Fields

- uint32_t commandIdSource
  Indicate the command buffer being executed that generated this result.
- uint32_t loopCountIndex
  Indicate the loop count value during command execution that generated this result.
- uint32_t triggerIdSource
  Indicate the trigger source that initiated a conversion and generated this result.
- uint16_t convValue
  Data result.

Field Documentation

(1) uint32_t lpadc_conv_result_t::commandIdSource

(2) uint32_t lpadc_conv_result_t::loopCountIndex

(3) uint32_t lpadc_conv_result_t::triggerIdSource

(4) uint16_t lpadc_conv_result_t::convValue

27.4 Macro Definition Documentation

27.4.1 #define FSL_LPADC_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))

27.4.2 #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().

27.4.3 #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().

27.5 Enumeration Type Documentation

27.5.1 enum _lpadc_status_flags

Enumerator

kLPADC_ResultFIFO0OverflowFlag Indicates that more data has been written to the Result FIFO 0 than it can hold.

kLPADC_ResultFIFO0ReadyFlag Indicates when the number of valid datawords in the result FIFO 0 is greater than the setting watermark level.

kLPADC_ResultFIFO1OverflowFlag Indicates that more data has been written to the Result FIFO 1 than it can hold.

kLPADC_ResultFIFO1ReadyFlag Indicates when the number of valid datawords in the result FIFO 1 is greater than the setting watermark level.

27.5.2 enum _lpadc_interrupt_enable

Enumerator

kLPADC_ResultFIFO0OverflowInterruptEnable Configures ADC to generate overflow interrupt requests when FOF0 flag is asserted.

kLPADC_FIFO0WatermarkInterruptEnable Configures ADC to generate watermark interrupt requests when RDY0 flag is asserted.

kLPADC_ResultFIFO1OverflowInterruptEnable Configures ADC to generate overflow interrupt requests when FOF1 flag is asserted.

kLPADC_FIFO1WatermarkInterruptEnable Configures ADC to generate watermark interrupt requests when RDY1 flag is asserted.

27.5.3 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).

27.5.4 enum lpadc_sample_channel_mode_t

The channel sample mode configures the channel with single-end/differential/dual-single-end, side A/B.

Enumerator

kLPADC_SampleChannelSingleEndSideA Single end mode, using side A.
kLPADC_SampleChannelSingleEndSideB Single end mode, using side B.
kLPADC_SampleChannelDiffBothSideAB Differential mode, using A as plus side and B as minus side.
kLPADC_SampleChannelDiffBothSideBA Differential mode, using B as plus side and A as minus side.

27.5.5 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.
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.

27.5.6 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.

27.5.7 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.

27.5.8 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.

27.5.9 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.

27.5.10 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.

kLPADC_TriggerPriorityPreemptSubsequently If a higher priority trigger is received during command processing, the current command will be completed (averaging, looping, compare) before servicing the higher priority trigger.

27.6 Function Documentation

27.6.1 void LPADC_Init (ADC_Type ∗ *base,* const lpadc_config_t ∗ *config*)

Parameters

base	LPADC peripheral base address.
config	Pointer to configuration structure. See "lpadc_config_t".

27.6.2 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

config	Pointer to configuration structure.

27.6.3 void LPADC_Deinit (ADC_Type * *base*)

Parameters

base	LPADC peripheral base address.

27.6.4 static void LPADC_Enable (ADC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	LPADC peripheral base address.
enable	switcher to the module.

27.6.5 static void LPADC_DoResetFIFO0 (ADC_Type * *base*) [inline], [static]

Parameters

base	LPADC peripheral base address.

27.6.6 static void LPADC_DoResetFIFO1 (ADC_Type * *base*) [inline], [static]

Parameters

base	LPADC peripheral base address.

27.6.7 static void LPADC_DoResetConfig (ADC_Type * *base*) [inline], [static]

Reset all ADC internal logic and registers, except the Control Register (ADCx_CTRL).

Parameters

base	LPADC peripheral base address.

27.6.8 static uint32_t LPADC_GetStatusFlags (ADC_Type * *base*) [inline], [static]

Parameters

base	LPADC peripheral base address.

Returns

status flags' mask. See to _lpadc_status_flags.

27.6.9 static void LPADC_ClearStatusFlags (ADC_Type * *base*, uint32_t *mask*) [inline], [static]

Only the flags can be cleared by writing ADCx_STATUS register would be cleared by this API.

Parameters

base	LPADC peripheral base address.
mask	Mask value for flags to be cleared. See to _lpadc_status_flags.

27.6.10 static void LPADC_EnableInterrupts (ADC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LPADC peripheral base address.
mask	Mask value for interrupt events. See to _lpadc_interrupt_enable.

27.6.11 static void LPADC_DisableInterrupts (ADC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LPADC peripheral base address.
mask	Mask value for interrupt events. See to _lpadc_interrupt_enable.

27.6.12 static void LPADC_EnableFIFO0WatermarkDMA (ADC_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	LPADC peripheral base address.
enable	Switcher to the event.

27.6.13 static void LPADC_EnableFIFO1WatermarkDMA (ADC_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	LPADC peripheral base address.
enable	Switcher to the event.

27.6.14 static uint32_t LPADC_GetConvResultCount (ADC_Type * *base*, uint8_t *index*) [inline], [static]

Parameters

base	LPADC peripheral base address.
index	Result FIFO index.

Returns

The count of result kept in conversion FIFOn.

27.6.15 bool LPADC_GetConvResult (ADC_Type * *base,* lpadc_conv_result_t * *result,* uint8_t *index*)

param base LPADC peripheral base address. param result Pointer to structure variable that keeps the conversion result in conversion FIFOn. param index Result FIFO index.

return Status whether FIFOn entry is valid.

27.6.16 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

base	LPADC peripheral base address.
triggerId	ID for each trigger. Typically, the available value range is from 0.
config	Pointer to configuration structure. See to lpadc_conv_trigger_config_t.

27.6.17 void LPADC_GetDefaultConvTriggerConfig (lpadc_conv_trigger_config_t * *config*)

This function initializes the trigger's configuration structure with an available settings. The default values are:

```
*   config->commandIdSource       = 0U;
*   config->loopCountIndex        = 0U;
*   config->triggerIdSource       = 0U;
*   config->enableHardwareTrigger = false;
*
```

Parameters

config	Pointer to configuration structure.

27.6.18 static void LPADC_DoSoftwareTrigger (ADC_Type * *base*, uint32_t *triggerIdMask*) [inline], [static]

Parameters

base	LPADC peripheral base address.
triggerIdMask	Mask value for software trigger indexes, which count from zero.

27.6.19 void LPADC_SetConvCommandConfig (ADC_Type * *base*, uint32_t *commandId*, const lpadc_conv_command_config_t * *config*)

Parameters

base	LPADC peripheral base address.
commandId	ID for command in command buffer. Typically, the available value range is 1 - 15.
config	Pointer to configuration structure. See to lpadc_conv_command_config_t.

27.6.20 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

config	Pointer to configuration structure.

Chapter 28
CRC: Cyclic Redundancy Check Driver

28.1 Overview

MCUXpresso SDK provides a peripheral driver for the Cyclic Redundancy Check (CRC) module of MCUXpresso SDK devices.

The cyclic redundancy check (CRC) module generates 16/32-bit CRC code for error detection. The CRC module provides three variants of polynomials, a programmable seed, and other parameters required to implement a 16-bit or 32-bit CRC standard.

28.2 CRC Driver Initialization and Configuration

CRC_Init() function enables the clock for the CRC module in the LPC SYSCON block and fully (re-)configures the CRC module according to configuration structure. It also starts checksum computation by writing the seed.

The seed member of the configuration structure is the initial checksum for which new data can be added to. When starting new checksum computation, the seed should be set to the initial checksum per the CRC protocol specification. For continued checksum operation, the seed should be set to the intermediate checksum value as obtained from previous calls to CRC_GetConfig() function. After CRC_Init(), one or multiple CRC_WriteData() calls follow to update checksum with data, then CRC_Get16bitResult() or CRC_Get32bitResult() follows to read the result. CRC_Init() can be called as many times as required, which allows for runtime changes of the CRC protocol.

CRC_GetDefaultConfig() function can be used to set the module configuration structure with parameters for CRC-16/CCITT-FALSE protocol.

CRC_Deinit() function disables clock to the CRC module.

CRC_Reset() performs hardware reset of the CRC module.

28.3 CRC Write Data

The CRC_WriteData() function is used to add data to actual CRC. Internally it tries to use 32-bit reads and writes for all aligned data in the user buffer and it uses 8-bit reads and writes for all unaligned data in the user buffer. This function can update CRC with user supplied data chunks of arbitrary size, so one can update CRC byte by byte or with all bytes at once. Prior call of CRC configuration function CRC_Init() fully specifies the CRC module configuration for CRC_WriteData() call.

CRC_WriteSeed() Write seed (initial checksum) to CRC module.

28.4 CRC Get Checksum

The CRC_Get16bitResult() or CRC_Get32bitResult() function is used to read the CRC module checksum register. The bit reverse and 1's complement operations are already applied to the result if previously

configured. Use CRC_GetConfig() function to get the actual checksum without bit reverse and 1's complement applied so it can be used as seed when resuming calculation later.

CRC_Init() / CRC_WriteData() / CRC_Get16bitResult() to get final checksum.

CRC_Init() / CRC_WriteData() / ... / CRC_WriteData() / CRC_Get16bitResult() to get final checksum.

CRC_Init() / CRC_WriteData() / CRC_GetConfig() to get intermediate checksum to be used as seed value in future.

CRC_Init() / CRC_WriteData() / ... / CRC_WriteData() / CRC_GetConfig() to get intermediate checksum.

28.5 Comments about API usage in RTOS

If multiple RTOS tasks share the CRC module to compute checksums with different data and/or protocols, the following needs to be implemented by the user:

The triplets

CRC_Init() / CRC_WriteData() / CRC_Get16bitResult() or CRC_Get32bitResult() or CRC_GetConfig()

Should be protected by RTOS mutex to protect CRC module against concurrent accesses from different tasks. For example: Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crcRefer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/crc

Files

- file fsl_crc.h

Data Structures

- struct crc_config_t
  CRC protocol configuration. More...

Macros

- #define CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT 1
  Default configuration structure filled by CRC_GetDefaultConfig().

Enumerations

- enum crc_polynomial_t {
  kCRC_Polynomial_CRC_CCITT = 0U,
  kCRC_Polynomial_CRC_16 = 1U,
  kCRC_Polynomial_CRC_32 = 2U }
  CRC polynomials to use.

Functions

- void CRC_Init (CRC_Type *base, const crc_config_t *config)
  Enables and configures the CRC peripheral module.
- static void CRC_Deinit (CRC_Type *base)
  Disables the CRC peripheral module.
- void CRC_Reset (CRC_Type *base)
  resets CRC peripheral module.
- void CRC_WriteSeed (CRC_Type *base, uint32_t seed)
  Write seed to CRC peripheral module.
- void CRC_GetDefaultConfig (crc_config_t *config)
  Loads default values to CRC protocol configuration structure.
- void CRC_GetConfig (CRC_Type *base, crc_config_t *config)
  Loads actual values configured in CRC peripheral to CRC protocol configuration structure.
- void CRC_WriteData (CRC_Type *base, const uint8_t *data, size_t dataSize)
  Writes data to the CRC module.
- static uint32_t CRC_Get32bitResult (CRC_Type *base)
  Reads 32-bit checksum from the CRC module.
- static uint16_t CRC_Get16bitResult (CRC_Type *base)
  Reads 16-bit checksum from the CRC module.

Driver version

- #define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
  CRC driver version.

28.6 Data Structure Documentation

28.6.1 struct crc_config_t

This structure holds the configuration for the CRC protocol.

Data Fields

- crc_polynomial_t polynomial
  CRC polynomial.
- bool reverseIn
  Reverse bits on input.
- bool complementIn
  Perform 1's complement on input.
- bool reverseOut
  Reverse bits on output.
- bool complementOut
  Perform 1's complement on output.
- uint32_t seed
  Starting checksum value.

Field Documentation

(1) crc_polynomial_t crc_config_t::polynomial

(2) bool crc_config_t::reverseIn

(3) bool crc_config_t::complementIn

(4) bool crc_config_t::reverseOut

(5) bool crc_config_t::complementOut

(6) uint32_t crc_config_t::seed

28.7 Macro Definition Documentation

28.7.1 #define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.0.0
  - initial version
- Version 2.0.1
  - add explicit type cast when writing to WR_DATA
- Version 2.0.2
  - Fix MISRA issue
- Version 2.1.0
  - Add CRC_WriteSeed function
- Version 2.1.1
  - Fix MISRA issue

28.7.2 #define CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT 1

Uses CRC-16/CCITT-FALSE as default.

28.8 Enumeration Type Documentation

28.8.1 enum crc_polynomial_t

Enumerator

kCRC_Polynomial_CRC_CCITT x^16+x^12+x^5+1
kCRC_Polynomial_CRC_16 x^16+x^15+x^2+1
kCRC_Polynomial_CRC_32 x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+

28.9 Function Documentation

28.9.1 void CRC_Init (CRC_Type * *base,* const crc_config_t * *config*)

This functions enables the CRC peripheral clock in the LPC SYSCON block. It also configures the CRC engine and starts checksum computation by writing the seed.

Parameters

base	CRC peripheral address.
config	CRC module configuration structure.

28.9.2 static void CRC_Deinit (CRC_Type * *base*) [inline], [static]

This functions disables the CRC peripheral clock in the LPC SYSCON block.

Parameters

base	CRC peripheral address.

28.9.3 void CRC_Reset (CRC_Type * *base*)

Parameters

base	CRC peripheral address.

28.9.4 void CRC_WriteSeed (CRC_Type * *base*, uint32_t *seed*)

Parameters

base	CRC peripheral address.
seed	CRC Seed value.

28.9.5 void CRC_GetDefaultConfig (crc_config_t * *config*)

Loads default values to CRC protocol configuration structure. The default values are:

```
*   config->polynomial = kCRC_Polynomial_CRC_CCITT;
*   config->reverseIn = false;
*   config->complementIn = false;
*   config->reverseOut = false;
*   config->complementOut = false;
*   config->seed = 0xFFFFU;
*
```

Parameters

config	CRC protocol configuration structure

28.9.6 void CRC_GetConfig (CRC_Type * *base*, crc_config_t * *config*)

The values, including seed, can be used to resume CRC calculation later.

Parameters

base	CRC peripheral address.
config	CRC protocol configuration structure

28.9.7 void CRC_WriteData (CRC_Type * *base*, const uint8_t * *data*, size_t *dataSize*)

Writes input data buffer bytes to CRC data register.

Parameters

base	CRC peripheral address.
data	Input data stream, MSByte in data[0].
dataSize	Size of the input data buffer in bytes.

28.9.8 static uint32_t CRC_Get32bitResult (CRC_Type * *base*) [inline], [static]

Reads CRC data register.

Parameters

base	CRC peripheral address.

Returns

final 32-bit checksum, after configured bit reverse and complement operations.

28.9.9 static uint16_t CRC_Get16bitResult (CRC_Type * *base*) [inline], [static]

Reads CRC data register.

Parameters

base	CRC peripheral address.

Returns

final 16-bit checksum, after configured bit reverse and complement operations.

Chapter 29
DMA: Direct Memory Access Controller Driver

29.1 Overview

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

29.2 Typical use case

29.2.1 DMA Operation

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

Files

- file fsl_dma.h

Data Structures

- struct dma_descriptor_t
  DMA descriptor structure. More...
- struct dma_xfercfg_t
  DMA transfer configuration. More...
- struct dma_channel_trigger_t
  DMA channel trigger. More...
- struct dma_channel_config_t
  DMA channel trigger. More...
- struct dma_transfer_config_t
  DMA transfer configuration. More...
- struct dma_handle_t
  DMA transfer handle structure. More...

Macros

- #define DMA_MAX_TRANSFER_COUNT 0x400U
  DMA max transfer size.
- #define FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE (16U)
  DMA channel numbers.
- #define DMA_ALLOCATE_HEAD_DESCRIPTORS(name, number) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)
  DMA head descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE(name, number) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)

DMA head descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.

- #define DMA_ALLOCATE_LINK_DESCRIPTORS(name, number) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)

  DMA link descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE(name, number) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)

  DMA link descriptor table allocate macro at noncacheable section To simplify user interface, this macro will help allocate descriptor memory at noncacheable section, user just need to provide the name and the number for the allocate descriptor.
- #define DMA_ALLOCATE_DATA_TRANSFER_BUFFER(name, width) SDK_ALIGN(name, width)

  DMA transfer buffer address need to align with the transfer width.
- #define DMA_COMMON_REG_GET(base, channel, reg) (((volatile uint32_t ∗)(&((base)->COMMON[0].reg)))[DMA_CHANNEL_GROUP(channel)])

  DMA linked descriptor address algin size.
- #define DMA_DESCRIPTOR_END_ADDRESS(start, inc, bytes, width) ((uint32_t ∗)((uint32_t)(start) + (inc) ∗ (bytes) - (inc) ∗ (width)))

  DMA descriptor end address calculate.
- #define DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes)

  DMA channel transfer configurations macro.

Typedefs

- typedef void(∗ dma_callback)(struct _dma_handle ∗handle, void ∗userData, bool transferDone, uint32_t intmode)

  Define Callback function for DMA.

Enumerations

- enum { kStatus_DMA_Busy = MAKE_STATUS(kStatusGroup_DMA, 0) }

  _dma_transfer_status DMA transfer status
- enum {
  kDMA_AddressInterleave0xWidth = 0U,
  kDMA_AddressInterleave1xWidth = 1U,
  kDMA_AddressInterleave2xWidth = 2U,
  kDMA_AddressInterleave4xWidth = 4U }

  _dma_addr_interleave_size dma address interleave size
- enum {
  kDMA_Transfer8BitWidth = 1U,
  kDMA_Transfer16BitWidth = 2U,
  kDMA_Transfer32BitWidth = 4U }

  _dma_transfer_width dma transfer width
- enum dma_priority_t {

kDMA_ChannelPriority0 = 0,
kDMA_ChannelPriority1,
kDMA_ChannelPriority2,
kDMA_ChannelPriority3,
kDMA_ChannelPriority4,
kDMA_ChannelPriority5,
kDMA_ChannelPriority6,
kDMA_ChannelPriority7 }

DMA channel priority.

- enum dma_irq_t {
  kDMA_IntA,
  kDMA_IntB,
  kDMA_IntError }

  DMA interrupt flags.
- enum dma_trigger_type_t {
  kDMA_NoTrigger = 0,
  kDMA_LowLevelTrigger = DMA_CHANNEL_CFG_HWTRIGEN(1) | DMA_CHANNEL_CFG-_TRIGTYPE(1),
  kDMA_HighLevelTrigger,
  kDMA_FallingEdgeTrigger = DMA_CHANNEL_CFG_HWTRIGEN(1),
  kDMA_RisingEdgeTrigger }

  DMA trigger type.
- enum {
  kDMA_BurstSize1 = 0U,
  kDMA_BurstSize2 = 1U,
  kDMA_BurstSize4 = 2U,
  kDMA_BurstSize8 = 3U,
  kDMA_BurstSize16 = 4U,
  kDMA_BurstSize32 = 5U,
  kDMA_BurstSize64 = 6U,
  kDMA_BurstSize128 = 7U,
  kDMA_BurstSize256 = 8U,
  kDMA_BurstSize512 = 9U,
  kDMA_BurstSize1024 = 10U }

  _dma_burst_size DMA burst size
- enum dma_trigger_burst_t {

kDMA_SingleTransfer = 0,
kDMA_LevelBurstTransfer = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer1 = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer2,
kDMA_EdgeBurstTransfer4,
kDMA_EdgeBurstTransfer8,
kDMA_EdgeBurstTransfer16,
kDMA_EdgeBurstTransfer32,
kDMA_EdgeBurstTransfer64,
kDMA_EdgeBurstTransfer128,
kDMA_EdgeBurstTransfer256,
kDMA_EdgeBurstTransfer512,
kDMA_EdgeBurstTransfer1024 }
DMA trigger burst.

- enum dma_burst_wrap_t {
  kDMA_NoWrap = 0,
  kDMA_SrcWrap = DMA_CHANNEL_CFG_SRCBURSTWRAP(1),
  kDMA_DstWrap = DMA_CHANNEL_CFG_DSTBURSTWRAP(1),
  kDMA_SrcAndDstWrap }
  DMA burst wrapping.
- enum dma_transfer_type_t {
  kDMA_MemoryToMemory = 0x0U,
  kDMA_PeripheralToMemory,
  kDMA_MemoryToPeripheral,
  kDMA_StaticToStatic }
  DMA transfer type.

Driver version

- #define FSL_DMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))
  DMA driver version.

DMA initialization and De-initialization

- void DMA_Init (DMA_Type ∗base)
  Initializes DMA peripheral.
- void DMA_Deinit (DMA_Type ∗base)
  Deinitializes DMA peripheral.
- void DMA_InstallDescriptorMemory (DMA_Type ∗base, void ∗addr)
  Install DMA descriptor memory.

DMA Channel Operation

- static bool DMA_ChannelIsActive (DMA_Type ∗base, uint32_t channel)
  Return whether DMA channel is processing transfer.
- static bool DMA_ChannelIsBusy (DMA_Type ∗base, uint32_t channel)
  Return whether DMA channel is busy.
- static void DMA_EnableChannelInterrupts (DMA_Type ∗base, uint32_t channel)

Enables the interrupt source for the DMA transfer.
- static void DMA_DisableChannelInterrupts (DMA_Type *base, uint32_t channel)
  Disables the interrupt source for the DMA transfer.
- static void DMA_EnableChannel (DMA_Type *base, uint32_t channel)
  Enable DMA channel.
- static void DMA_DisableChannel (DMA_Type *base, uint32_t channel)
  Disable DMA channel.
- static void DMA_EnableChannelPeriphRq (DMA_Type *base, uint32_t channel)
  Set PERIPHREQEN of channel configuration register.
- static void DMA_DisableChannelPeriphRq (DMA_Type *base, uint32_t channel)
  Get PERIPHREQEN value of channel configuration register.
- void DMA_ConfigureChannelTrigger (DMA_Type *base, uint32_t channel, dma_channel_trigger_t *trigger)
  Set trigger settings of DMA channel.
- void DMA_SetChannelConfig (DMA_Type *base, uint32_t channel, dma_channel_trigger_t *trigger, bool isPeriph)
  set channel config.
- uint32_t DMA_GetRemainingBytes (DMA_Type *base, uint32_t channel)
  Gets the remaining bytes of the current DMA descriptor transfer.
- static void DMA_SetChannelPriority (DMA_Type *base, uint32_t channel, dma_priority_t priority)
  Set priority of channel configuration register.
- static dma_priority_t DMA_GetChannelPriority (DMA_Type *base, uint32_t channel)
  Get priority of channel configuration register.
- static void DMA_SetChannelConfigValid (DMA_Type *base, uint32_t channel)
  Set channel configuration valid.
- static void DMA_DoChannelSoftwareTrigger (DMA_Type *base, uint32_t channel)
  Do software trigger for the channel.
- static void DMA_LoadChannelTransferConfig (DMA_Type *base, uint32_t channel, uint32_t xfer)
  Load channel transfer configurations.
- void DMA_CreateDescriptor (dma_descriptor_t *desc, dma_xfercfg_t *xfercfg, void *srcAddr, void *dstAddr, void *nextDesc)
  Create application specific DMA descriptor to be used in a chain in transfer.
- void DMA_SetupDescriptor (dma_descriptor_t *desc, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)
  setup dma descriptor
- void DMA_SetupChannelDescriptor (dma_descriptor_t *desc, uint32_t xfercfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc, dma_burst_wrap_t wrapType, uint32_t burstSize)
  setup dma channel descriptor
- void DMA_LoadChannelDescriptor (DMA_Type *base, uint32_t channel, dma_descriptor_t *descriptor)
  load channel transfer decriptor.

DMA Transactional Operation

- void DMA_AbortTransfer (dma_handle_t *handle)
  Abort running transfer by handle.
- void DMA_CreateHandle (dma_handle_t *handle, DMA_Type *base, uint32_t channel)
  Creates the DMA handle.
- void DMA_SetCallback (dma_handle_t *handle, dma_callback callback, void *userData)
  Installs a callback function for the DMA transfer.

- void DMA_PrepareTransfer (dma_transfer_config_t *config, void *srcAddr, void *dstAddr, uint32_t byteWidth, uint32_t transferBytes, dma_transfer_type_t type, void *nextDesc)
  Prepares the DMA transfer structure.
- void DMA_PrepareChannelTransfer (dma_channel_config_t *config, void *srcStartAddr, void *dstStartAddr, uint32_t xferCfg, dma_transfer_type_t type, dma_channel_trigger_t *trigger, void *nextDesc)
  Prepare channel transfer configurations.
- status_t DMA_SubmitTransfer (dma_handle_t *handle, dma_transfer_config_t *config)
  Submits the DMA transfer request.
- void DMA_SubmitChannelTransferParameter (dma_handle_t *handle, uint32_t xferCfg, void *srcStartAddr, void *dstStartAddr, void *nextDesc)
  Submit channel transfer paramter directly.
- void DMA_SubmitChannelDescriptor (dma_handle_t *handle, dma_descriptor_t *descriptor)
  Submit channel descriptor.
- status_t DMA_SubmitChannelTransfer (dma_handle_t *handle, dma_channel_config_t *config)
  Submits the DMA channel transfer request.
- void DMA_StartTransfer (dma_handle_t *handle)
  DMA start transfer.
- void DMA_IRQHandle (DMA_Type *base)
  DMA IRQ handler for descriptor transfer complete.

29.3 Data Structure Documentation

29.3.1 struct dma_descriptor_t

Data Fields

- volatile uint32_t xfercfg
  Transfer configuration.
- void * srcEndAddr
  Last source address of DMA transfer.
- void * dstEndAddr
  Last destination address of DMA transfer.
- void * linkToNextDesc
  Address of next DMA descriptor in chain.

29.3.2 struct dma_xfercfg_t

Data Fields

- bool valid
  Descriptor is ready to transfer.
- bool reload
  Reload channel configuration register after current descriptor is exhausted.
- bool swtrig
  Perform software trigger.
- bool clrtrig

Clear trigger.
- bool intA
  Raises IRQ when transfer is done and set IRQA status register flag.
- bool intB
  Raises IRQ when transfer is done and set IRQB status register flag.
- uint8_t byteWidth
  Byte width of data to transfer.
- uint8_t srcInc
  Increment source address by 'srcInc' x 'byteWidth'.
- uint8_t dstInc
  Increment destination address by 'dstInc' x 'byteWidth'.
- uint16_t transferCount
  Number of transfers.

Field Documentation

(1) bool dma_xfercfg_t::swtrig

Transfer if fired when 'valid' is set

29.3.3 struct dma_channel_trigger_t

Data Fields

- dma_trigger_type_t type
  Select hardware trigger as edge triggered or level triggered.
- dma_trigger_burst_t burst
  Select whether hardware triggers cause a single or burst transfer.
- dma_burst_wrap_t wrap
  Select wrap type, source wrap or dest wrap, or both.

Field Documentation

(1) dma_trigger_type_t dma_channel_trigger_t::type

(2) dma_trigger_burst_t dma_channel_trigger_t::burst

(3) dma_burst_wrap_t dma_channel_trigger_t::wrap

29.3.4 struct dma_channel_config_t

Data Fields

- void * srcStartAddr
  Source data address.
- void * dstStartAddr
  Destination data address.
- void * nextDesc

Chain custom descriptor.
- uint32_t xferCfg
  channel transfer configurations
- dma_channel_trigger_t * trigger
  DMA trigger type.
- bool isPeriph
  select the request type

29.3.5 struct dma_transfer_config_t

Data Fields

- uint8_t * srcAddr
  Source data address.
- uint8_t * dstAddr
  Destination data address.
- uint8_t * nextDesc
  Chain custom descriptor.
- dma_xfercfg_t xfercfg
  Transfer options.
- bool isPeriph
  DMA transfer is driven by peripheral.

29.3.6 struct dma_handle_t

Data Fields

- dma_callback callback
  Callback function.
- void * userData
  Callback function parameter.
- DMA_Type * base
  DMA peripheral base address.
- uint8_t channel
  DMA channel number.

Field Documentation

(1) dma_callback dma_handle_t::callback

Invoked when transfer of descriptor with interrupt flag finishes

29.4 Macro Definition Documentation

29.4.1 #define FSL_DMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))

Version 2.4.3.

29.4.2 #define FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE (16U)

DMA head link descriptor table align size

29.4.3 #define DMA_ALLOCATE_HEAD_DESCRIPTORS(*name, number*) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)

Parameters

name	Allocate decriptor name.
number	Number of descriptor to be allocated.

29.4.4 #define DMA_ALLOCATE_HEAD_DESCRIPTORS_AT_NONCACHEABLE(*name, number*) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)

Parameters

name	Allocate decriptor name.
number	Number of descriptor to be allocated.

29.4.5 #define DMA_ALLOCATE_LINK_DESCRIPTORS(*name, number*) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)

Parameters

name	Allocate decriptor name.
number	Number of descriptor to be allocated.

29.4.6 #define DMA_ALLOCATE_LINK_DESCRIPTORS_AT_NONCACHEABLE(*name, number*) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)

Parameters

name	Allocate decriptor name.
number	Number of descriptor to be allocated.

29.4.7 #define DMA_DESCRIPTOR_END_ADDRESS(*start, inc, bytes, width*) ((uint32_t ∗)((uint32_t)(start) + (inc) ∗ (bytes) - (inc) ∗ (width)))

Parameters

start	start address
inc	address interleave size
bytes	transfer bytes
width	transfer width

29.4.8 #define DMA_CHANNEL_XFER(*reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes*)

Value:

```
DMA_CHANNEL_XFERCFG_CFGVALID_MASK | DMA_CHANNEL_XFERCFG_RELOAD(reload) | DMA_CHANNEL_XFERCFG_CLRTRIG(
      clrTrig) | \
        DMA_CHANNEL_XFERCFG_SETINTA(intA) | DMA_CHANNEL_XFERCFG_SETINTB(intB) |
               \
        DMA_CHANNEL_XFERCFG_WIDTH(width == 4UL ? 2UL : (width - 1UL)) |
               \
        DMA_CHANNEL_XFERCFG_SRCINC(srcInc == (uint32_t)
      kDMA_AddressInterleave4xWidth ? (srcInc - 1UL) : srcInc) |   \
        DMA_CHANNEL_XFERCFG_DSTINC(dstInc == (uint32_t)
      kDMA_AddressInterleave4xWidth ? (dstInc - 1UL) : dstInc) |   \
        DMA_CHANNEL_XFERCFG_XFERCOUNT(bytes / width - 1UL)
```

Parameters

reload	true is reload link descriptor after current exhaust, false is not
clrTrig	true is clear trigger status, wait software trigger, false is not

intA	enable interruptA
intB	enable interruptB
width	transfer width
srcInc	source address interleave size
dstInc	destination address interleave size
bytes	transfer bytes

29.5 Typedef Documentation

29.5.1 typedef void(∗ dma_callback)(struct _dma_handle ∗handle, void ∗userData, bool transferDone, uint32_t intmode)

29.6 Enumeration Type Documentation

29.6.1 anonymous enum

Enumerator

kStatus_DMA_Busy Channel is busy and can't handle the transfer request.

29.6.2 anonymous enum

Enumerator

kDMA_AddressInterleave0xWidth dma source/destination address no interleave
kDMA_AddressInterleave1xWidth dma source/destination address interleave 1xwidth
kDMA_AddressInterleave2xWidth dma source/destination address interleave 2xwidth
kDMA_AddressInterleave4xWidth dma source/destination address interleave 3xwidth

29.6.3 anonymous enum

Enumerator

kDMA_Transfer8BitWidth dma channel transfer bit width is 8 bit
kDMA_Transfer16BitWidth dma channel transfer bit width is 16 bit
kDMA_Transfer32BitWidth dma channel transfer bit width is 32 bit

29.6.4 enum dma_priority_t

Enumerator

kDMA_ChannelPriority0 Highest channel priority - priority 0.

kDMA_ChannelPriority1 Channel priority 1.
kDMA_ChannelPriority2 Channel priority 2.
kDMA_ChannelPriority3 Channel priority 3.
kDMA_ChannelPriority4 Channel priority 4.
kDMA_ChannelPriority5 Channel priority 5.
kDMA_ChannelPriority6 Channel priority 6.
kDMA_ChannelPriority7 Lowest channel priority - priority 7.

29.6.5 enum dma_irq_t

Enumerator

kDMA_IntA DMA interrupt flag A.
kDMA_IntB DMA interrupt flag B.
kDMA_IntError DMA interrupt flag error.

29.6.6 enum dma_trigger_type_t

Enumerator

kDMA_NoTrigger Trigger is disabled.
kDMA_LowLevelTrigger Low level active trigger.
kDMA_HighLevelTrigger High level active trigger.
kDMA_FallingEdgeTrigger Falling edge active trigger.
kDMA_RisingEdgeTrigger Rising edge active trigger.

29.6.7 anonymous enum

Enumerator

kDMA_BurstSize1 burst size 1 transfer
kDMA_BurstSize2 burst size 2 transfer
kDMA_BurstSize4 burst size 4 transfer
kDMA_BurstSize8 burst size 8 transfer
kDMA_BurstSize16 burst size 16 transfer
kDMA_BurstSize32 burst size 32 transfer
kDMA_BurstSize64 burst size 64 transfer
kDMA_BurstSize128 burst size 128 transfer
kDMA_BurstSize256 burst size 256 transfer
kDMA_BurstSize512 burst size 512 transfer
kDMA_BurstSize1024 burst size 1024 transfer

29.6.8 enum dma_trigger_burst_t

Enumerator

kDMA_SingleTransfer Single transfer.
kDMA_LevelBurstTransfer Burst transfer driven by level trigger.
kDMA_EdgeBurstTransfer1 Perform 1 transfer by edge trigger.
kDMA_EdgeBurstTransfer2 Perform 2 transfers by edge trigger.
kDMA_EdgeBurstTransfer4 Perform 4 transfers by edge trigger.
kDMA_EdgeBurstTransfer8 Perform 8 transfers by edge trigger.
kDMA_EdgeBurstTransfer16 Perform 16 transfers by edge trigger.
kDMA_EdgeBurstTransfer32 Perform 32 transfers by edge trigger.
kDMA_EdgeBurstTransfer64 Perform 64 transfers by edge trigger.
kDMA_EdgeBurstTransfer128 Perform 128 transfers by edge trigger.
kDMA_EdgeBurstTransfer256 Perform 256 transfers by edge trigger.
kDMA_EdgeBurstTransfer512 Perform 512 transfers by edge trigger.
kDMA_EdgeBurstTransfer1024 Perform 1024 transfers by edge trigger.

29.6.9 enum dma_burst_wrap_t

Enumerator

kDMA_NoWrap Wrapping is disabled.
kDMA_SrcWrap Wrapping is enabled for source.
kDMA_DstWrap Wrapping is enabled for destination.
kDMA_SrcAndDstWrap Wrapping is enabled for source and destination.

29.6.10 enum dma_transfer_type_t

Enumerator

kDMA_MemoryToMemory Transfer from memory to memory (increment source and destination)
kDMA_PeripheralToMemory Transfer from peripheral to memory (increment only destination)
kDMA_MemoryToPeripheral Transfer from memory to peripheral (increment only source)
kDMA_StaticToStatic Peripheral to static memory (do not increment source or destination)

29.7 Function Documentation

29.7.1 void DMA_Init (DMA_Type * *base*)

This function enable the DMA clock, set descriptor table and enable DMA peripheral.

Parameters

base	DMA peripheral base address.

29.7.2 void DMA_Deinit (DMA_Type * *base*)

This function gates the DMA clock.

Parameters

base	DMA peripheral base address.

29.7.3 void DMA_InstallDescriptorMemory (DMA_Type * *base,* void * *addr*)

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, althrough current DMA driver has a default DMA descriptor buffer, but it support one DMA descriptor for one channel only.

Parameters

base	DMA base address.
addr	DMA descriptor address

29.7.4 static bool DMA_ChannelIsActive (DMA_Type * *base,* uint32_t *channel*) `[inline], [static]`

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

True for active state, false otherwise.

29.7.5 static bool DMA_ChannelIsBusy (DMA_Type * *base,* uint32_t *channel*) `[inline], [static]`

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

True for busy state, false otherwise.

29.7.6 static void DMA_EnableChannelInterrupts (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

29.7.7 static void DMA_DisableChannelInterrupts (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

29.7.8 static void DMA_EnableChannel (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

29.7.9 static void DMA_DisableChannel (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

29.7.10 static void DMA_EnableChannelPeriphRq (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

29.7.11 static void DMA_DisableChannelPeriphRq (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

True for enabled PeriphRq, false for disabled.

29.7.12 void DMA_ConfigureChannelTrigger (DMA_Type * *base*, uint32_t *channel*, dma_channel_trigger_t * *trigger*)

Deprecated Do not use this function. It has been superceded by DMA_SetChannelConfig.

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
trigger	trigger configuration.

29.7.13 void DMA_SetChannelConfig (DMA_Type * *base,* uint32_t *channel,* dma_channel_trigger_t * *trigger,* bool *isPeriph*)

This function provide a interface to configure channel configuration reisters.

Parameters

base	DMA base address.
channel	DMA channel number.
trigger	channel configurations structure.
isPeriph	true is periph request, false is not.

29.7.14 uint32_t DMA_GetRemainingBytes (DMA_Type * *base,* uint32_t *channel*)

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

The number of bytes which have not been transferred yet.

29.7.15 static void DMA_SetChannelPriority (DMA_Type * *base,* uint32_t *channel,* dma_priority_t *priority*) `[inline], [static]`

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
priority	Channel priority value.

29.7.16 static dma_priority_t DMA_GetChannelPriority (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

Returns

Channel priority value.

29.7.17 static void DMA_SetChannelConfigValid (DMA_Type ∗ *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

29.7.18 static void DMA_DoChannelSoftwareTrigger (DMA_Type ∗ *base,* uint32_t *channel*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.

29.7.19 static void DMA_LoadChannelTransferConfig (DMA_Type ∗ *base,* uint32_t *channel,* uint32_t *xfer*) [inline], [static]

Parameters

base	DMA peripheral base address.
channel	DMA channel number.
xfer	transfer configurations.

29.7.20 void DMA_CreateDescriptor (dma_descriptor_t * *desc*, dma_xfercfg_t * *xfercfg*, void * *srcAddr*, void * *dstAddr*, void * *nextDesc*)

Deprecated Do not use this function. It has been superceded by DMA_SetupDescriptor.

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcAddr	Address of last item to transmit
dstAddr	Address of last item to receive.
nextDesc	Address of next descriptor in chain.

29.7.21 void DMA_SetupDescriptor (dma_descriptor_t * *desc*, uint32_t *xfercfg*, void * *srcStartAddr*, void * *dstStartAddr*, void * *nextDesc*)

Note: This function do not support configure wrap descriptor.

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcStartAddr	Start address of source address.
dstStartAddr	Start address of destination address.
nextDesc	Address of next descriptor in chain.

29.7.22 void DMA_SetupChannelDescriptor (dma_descriptor_t * *desc*, uint32_t *xfercfg*, void * *srcStartAddr*, void * *dstStartAddr*, void * *nextDesc*, dma_burst_wrap_t *wrapType*, uint32_t *burstSize*)

Note: This function support configure wrap descriptor.

Parameters

desc	DMA descriptor address.
xfercfg	Transfer configuration for DMA descriptor.
srcStartAddr	Start address of source address.
dstStartAddr	Start address of destination address.
nextDesc	Address of next descriptor in chain.
wrapType	burst wrap type.
burstSize	burst size, reference _dma_burst_size.

29.7.23 void DMA_LoadChannelDescriptor (DMA_Type * *base*, uint32_t *channel*, dma_descriptor_t * *descriptor*)

This function can be used to load desscriptor to driver internal channel descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the polling transfer, application can allocate a local descriptor memory table to prepare a descriptor firstly and then call this api to load the configured descriptor to driver descriptor table.

```
*   DMA_Init(DMA0);
*   DMA_EnableChannel(DMA0, DEMO_DMA_CHANNEL);
*   DMA_SetupDescriptor(desc, xferCfg, s_srcBuffer, &s_destBuffer[0], NULL);
*   DMA_LoadChannelDescriptor(DMA0, DEMO_DMA_CHANNEL, (
      dma_descriptor_t *)desc);
*   DMA_DoChannelSoftwareTrigger(DMA0, DEMO_DMA_CHANNEL);
*   while(DMA_ChannelIsBusy(DMA0, DEMO_DMA_CHANNEL))
*   {}
*
```

Parameters

base	DMA base address.
channel	DMA channel.
descriptor	configured DMA descriptor.

29.7.24 void DMA_AbortTransfer (dma_handle_t * *handle*)

This function aborts DMA transfer specified by handle.

Parameters

handle	DMA handle pointer.

29.7.25 void DMA_CreateHandle (dma_handle_t * *handle,* DMA_Type * *base,* uint32_t *channel*)

This function is called if using transaction API for DMA. This function initializes the internal state of DMA handle.

Parameters

handle	DMA handle pointer. The DMA handle stores callback function and parameters.
base	DMA peripheral base address.
channel	DMA channel number.

29.7.26 void DMA_SetCallback (dma_handle_t * *handle*, dma_callback *callback*, void * *userData*)

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

Parameters

handle	DMA handle pointer.
callback	DMA callback function pointer.
userData	Parameter for callback function.

29.7.27 void DMA_PrepareTransfer (dma_transfer_config_t * *config*, void * *srcAddr*, void * *dstAddr*, uint32_t *byteWidth*, uint32_t *transferBytes*, dma_transfer_type_t *type*, void * *nextDesc*)

Deprecated Do not use this function. It has been superceded by DMA_PrepareChannelTransfer. This function prepares the transfer configuration structure according to the user input.

Parameters

config	The user configuration structure of type dma_transfer_t.
srcAddr	DMA transfer source address.
dstAddr	DMA transfer destination address.
byteWidth	DMA transfer destination address width(bytes).
transferBytes	DMA transfer bytes to be transferred.
type	DMA transfer type.
nextDesc	Chain custom descriptor to transfer.

Note

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

29.7.28 void DMA_PrepareChannelTransfer (dma_channel_config_t * *config*, void * *srcStartAddr*, void * *dstStartAddr*, uint32_t *xferCfg*, dma_transfer_type_t *type*, dma_channel_trigger_t * *trigger*, void * *nextDesc*)

This function used to prepare channel transfer configurations.

Parameters

config	Pointer to DMA channel transfer configuration structure.
srcStartAddr	source start address.
dstStartAddr	destination start address.
xferCfg	xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
type	transfer type.
trigger	DMA channel trigger configurations.
nextDesc	address of next descriptor.

29.7.29 status_t DMA_SubmitTransfer (dma_handle_t * *handle*, dma_transfer_config_t * *config*)

Deprecated Do not use this function. It has been superceded by DMA_SubmitChannelTransfer.

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

handle	DMA handle pointer.
config	Pointer to DMA transfer configuration structure.

Return values

kStatus_DMA_Success	It means submit transfer request succeed.
kStatus_DMA_QueueFull	It means TCD queue is full. Submit transfer request is not allowed.
kStatus_DMA_Busy	It means the given channel is busy, need to submit request later.

29.7.30 void DMA_SubmitChannelTransferParameter (dma_handle_t * *handle,* uint32_t *xferCfg,* void * *srcStartAddr,* void * *dstStartAddr,* void * *nextDesc*)

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
   DMA_SetChannelConfig(base, channel, trigger, isPeriph);
   DMA_CreateHandle(handle, base, channel)
   DMA_SubmitChannelTransferParameter(handle,
      DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, NULL);
   DMA_StartTransfer(handle)
*
```

2. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
   define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[3]);

   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
   DMA_SetupDescriptor(nextDesc2,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
   DMA_SetChannelConfig(base, channel, trigger, isPeriph);
   DMA_CreateHandle(handle, base, channel)
   DMA_SubmitChannelTransferParameter(handle,
      DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, nextDesc0);
   DMA_StartTransfer(handle);
*
```

Parameters

handle	Pointer to DMA handle.
xferCfg	xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value.
srcStartAddr	source start address.
dstStartAddr	destination start address.
nextDesc	address of next descriptor.

29.7.31 void DMA_SubmitChannelDescriptor (dma_handle_t * *handle*, dma_descriptor_t * *descriptor*)

This function used to configue channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, this functiono is typical for the ping pong case:

1. for the ping pong case, application should responsible for the descriptor, for example, application should prepare two descriptor table with macro.

```
   define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[2]);

   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
   DMA_SetChannelConfig(base, channel, trigger, isPeriph);
   DMA_CreateHandle(handle, base, channel)
   DMA_SubmitChannelDescriptor(handle,  nextDesc0);
   DMA_StartTransfer(handle);
*
```

Parameters

handle	Pointer to DMA handle.
descriptor	descriptor to submit.

29.7.32 status_t DMA_SubmitChannelTransfer (dma_handle_t * *handle*, dma_channel_config_t * *config*)

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time. It is used for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
   DMA_CreateHandle(handle, base, channel)
   DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
      trigger,NULL);
   DMA_SubmitChannelTransfer(handle, config)
   DMA_StartTransfer(handle)
*
```

2. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
   define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);
   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
   DMA_SetupDescriptor(nextDesc2,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
   DMA_CreateHandle(handle, base, channel)
   DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
      trigger,nextDesc0);
   DMA_SubmitChannelTransfer(handle, config)
   DMA_StartTransfer(handle)
*
```

3. for the ping pong case, application should responsible for link descriptor, for example, application should prepare two descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
   define link descriptor table in application with macro
   DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);

   DMA_SetupDescriptor(nextDesc0,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
   DMA_SetupDescriptor(nextDesc1,  DMA_CHANNEL_XFER(reload, clrTrig,
      intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
   DMA_CreateHandle(handle, base, channel)
   DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
      trigger,nextDesc0);
   DMA_SubmitChannelTransfer(handle, config)
   DMA_StartTransfer(handle)
*
```

Parameters

handle	DMA handle pointer.
config	Pointer to DMA transfer configuration structure.

Return values

kStatus_DMA_Success	It means submit transfer request succeed.
kStatus_DMA_QueueFull	It means TCD queue is full. Submit transfer request is not allowed.
kStatus_DMA_Busy	It means the given channel is busy, need to submit request later.

29.7.33 void DMA_StartTransfer (dma_handle_t ∗ *handle*)

This function enables the channel request. User can call this function after submitting the transfer request It will trigger transfer start with software trigger only when hardware trigger is not used.

Parameters

handle	DMA handle pointer.

29.7.34 void DMA_IRQHandle (DMA_Type ∗ *base*)

This function clears the channel major interrupt flag and call the callback function if it is not NULL.

Parameters

base	DMA base address.

Chapter 30
GPIO: General Purpose I/O

30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Purpose I/O (GPIO) module of MCUXpresso SDK devices.

30.2 Function groups

30.2.1 Initialization and deinitialization

The function GPIO_PinInit() initializes the GPIO with specified configuration.

30.2.2 Pin manipulation

The function GPIO_PinWrite() set output state of selected GPIO pin. The function GPIO_PinRead() read input value of selected GPIO pin.

30.2.3 Port manipulation

The function GPIO_PortSet() sets the output level of selected GPIO pins to the logic 1. The function GPIO_PortClear() sets the output level of selected GPIO pins to the logic 0. The function GPIO_PortToggle() reverse the output level of selected GPIO pins. The function GPIO_PortRead() read input value of selected port.

30.2.4 Port masking

The function GPIO_PortMaskedSet() set port mask, only pins masked by 0 will be enabled in following functions. The function GPIO_PortMaskedWrite() sets the state of selected GPIO port, only pins masked by 0 will be affected. The function GPIO_PortMaskedRead() reads the state of selected GPIO port, only pins masked by 0 are enabled for read, pins masked by 1 are read as 0.

30.3 Typical use case

Example use of GPIO API. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/gpio

Files

- file fsl_gpio.h

Data Structures

- struct gpio_pin_config_t
  The GPIO pin configuration structure. More...
- struct gpio_interrupt_config_t
  Configures the interrupt generation condition. More...

Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  LPC GPIO direction definition.
- enum gpio_pin_enable_mode_t {
  kGPIO_PinIntEnableLevel = GPIO_PIN_INT_LEVEL,
  kGPIO_PinIntEnableEdge = GPIO_PIN_INT_EDGE }
  GPIO Pin Interrupt enable mode.
- enum gpio_pin_enable_polarity_t {
  kGPIO_PinIntEnableHighOrRise,
  kGPIO_PinIntEnableLowOrFall }
  GPIO Pin Interrupt enable polarity.
- enum gpio_interrupt_index_t {
  kGPIO_InterruptA = 0U,
  kGPIO_InterruptB = 1U }
  LPC GPIO interrupt index definition.

Functions

- static void GPIO_PortSet (GPIO_Type ∗base, uint32_t port, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 1.
- static void GPIO_PortClear (GPIO_Type ∗base, uint32_t port, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 0.
- static void GPIO_PortToggle (GPIO_Type ∗base, uint32_t port, uint32_t mask)
  Reverses current output logic of the multiple GPIO pins.

Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))
  LPC GPIO driver version.

GPIO Configuration

- void GPIO_PortInit (GPIO_Type ∗base, uint32_t port)
  Initializes the GPIO peripheral.
- void GPIO_PinInit (GPIO_Type ∗base, uint32_t port, uint32_t pin, const gpio_pin_config_t ∗config)

Initializes a GPIO pin used by the board.

GPIO Output Operations

- static void GPIO_PinWrite (GPIO_Type ∗base, uint32_t port, uint32_t pin, uint8_t output)
  Sets the output level of the one GPIO pin to the logic 1 or 0.

GPIO Input Operations

- static uint32_t GPIO_PinRead (GPIO_Type ∗base, uint32_t port, uint32_t pin)
  Reads the current input value of the GPIO PIN.

30.4 Data Structure Documentation

30.4.1 struct gpio_pin_config_t

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

Data Fields

- gpio_pin_direction_t pinDirection
  GPIO direction, input or output.
- uint8_t outputLogic
  Set default output logic, no use in input.

30.4.2 struct gpio_interrupt_config_t

30.5 Macro Definition Documentation

30.5.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 1, 7))

30.6 Enumeration Type Documentation

30.6.1 enum gpio_pin_direction_t

Enumerator

kGPIO_DigitalInput Set current pin as digital input.
kGPIO_DigitalOutput Set current pin as digital output.

30.6.2 enum gpio_pin_enable_mode_t

Enumerator

kGPIO_PinIntEnableLevel Generate Pin Interrupt on level mode.

kGPIO_PinIntEnableEdge Generate Pin Interrupt on edge mode.

30.6.3 enum gpio_pin_enable_polarity_t

Enumerator

kGPIO_PinIntEnableHighOrRise Generate Pin Interrupt on high level or rising edge.
kGPIO_PinIntEnableLowOrFall Generate Pin Interrupt on low level or falling edge.

30.6.4 enum gpio_interrupt_index_t

Enumerator

kGPIO_InterruptA Set current pin as interrupt A.
kGPIO_InterruptB Set current pin as interrupt B.

30.7 Function Documentation

30.7.1 void GPIO_PortInit (GPIO_Type ∗ *base,* uint32_t *port*)

This function ungates the GPIO clock.

Parameters

base	GPIO peripheral base pointer.
port	GPIO port number.

30.7.2 void GPIO_PinInit (GPIO_Type ∗ *base,* uint32_t *port,* uint32_t *pin,* const gpio_pin_config_t ∗ *config*)

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or output pin configuration:

```
* Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
```

```
*   kGPIO_DigitalOutput,
*   0,
* }
*
```

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number
config	GPIO pin configuration pointer

30.7.3 static void GPIO_PinWrite (GPIO_Type ∗ *base,* uint32_t *port,* uint32_t *pin,* uint8_t *output*) `[inline], [static]`

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number
output	GPIO pin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

30.7.4 static uint32_t GPIO_PinRead (GPIO_Type ∗ *base,* uint32_t *port,* uint32_t *pin*) `[inline], [static]`

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
pin	GPIO pin number

Return values

GPIO	port input value • 0: corresponding pin input low-logic level. • 1: corresponding pin input high-logic level.

30.7.5 static void GPIO_PortSet (GPIO_Type * *base,* uint32_t *port,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

30.7.6 static void GPIO_PortClear (GPIO_Type ∗ *base,* uint32_t *port,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

30.7.7 static void GPIO_PortToggle (GPIO_Type ∗ *base,* uint32_t *port,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer(Typically GPIO)
port	GPIO port number
mask	GPIO pin number macro

Chapter 31
IOPCTL: Input/Output Pad Controller

31.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Input/Output Pad Controller.

The IOPCTL, working in conjunction with the PIMCTL, enables the chip to share one pad for multiple signals from different peripheral interfaces.

Files

- file fsl_iopctl.h

Data Structures

- struct iopctl_group_t
  Array of IOPCTL pin definitions passed to IOPCTL_SetPinMuxing() must be in this format. More...

Macros

- #define IOPCTL_FUNC0 0x0
  IOPCTL function and mode selection definitions.
- #define IOPCTL_FUNC1 0x1
  Selects pin function 1.
- #define IOPCTL_FUNC2 0x2
  Selects pin function 2.
- #define IOPCTL_FUNC3 0x3
  Selects pin function 3.
- #define IOPCTL_FUNC4 0x4
  Selects pin function 4.
- #define IOPCTL_FUNC5 0x5
  Selects pin function 5.
- #define IOPCTL_FUNC6 0x6
  Selects pin function 6.
- #define IOPCTL_FUNC7 0x7
  Selects pin function 7.
- #define IOPCTL_FUNC8 0x8
  Selects pin function 8.
- #define IOPCTL_FUNC9 0x9
  Selects pin function 9.
- #define IOPCTL_FUNC10 0xA
  Selects pin function 10.
- #define IOPCTL_FUNC11 0xB
  Selects pin function 11.
- #define IOPCTL_FUNC12 0xC
  Selects pin function 12.

- #define IOPCTL_FUNC13 0xD
  Selects pin function 13.
- #define IOPCTL_FUNC14 0xE
  Selects pin function 14.
- #define IOPCTL_FUNC15 0xF
  Selects pin function 15.
- #define IOPCTL_PUPD_EN (0x1 << 4)
  Enables Pullup / Pulldown.
- #define IOPCTL_PULLDOWN_EN (0x0 << 5)
  Selects pull-down function.
- #define IOPCTL_PULLUP_EN (0x1 << 5)
  Selects pull-up function.
- #define IOPCTL_INBUF_EN (0x1 << 6)
  Enables buffer function on input.
- #define IOPCTL_SLEW_RATE (0x0 << 7)
  Slew Rate Control.
- #define IOPCTL_FULLDRIVE_EN (0x1 << 8)
  Selects full drive.
- #define IOPCTL_ANAMUX_EN (0x1 << 9)
  Enables analog mux function by setting 0 to bit 7.
- #define IOPCTL_PSEDRAIN_EN (0x1 << 10)
  Enables pseudo output drain function.
- #define IOPCTL_INV_EN (0x1 << 11)
  Enables invert function on input.

Functions

- __STATIC_INLINE void IOPCTL_PinMuxSet (IOPCTL_Type *base, uint8_t port, uint8_t pin, uint32_t modefunc)
  Sets I/O Pad Control pin mux.
- __STATIC_INLINE void IOPCTL_SetPinMuxing (IOPCTL_Type *base, const iopctl_group_t *pinArray, uint32_t arrayLength)
  Set all I/O Control pin muxing.

Driver version

- #define LPC_IOPCTL_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  IOPCTL driver version 2.0.0.

31.2 Data Structure Documentation

31.2.1 struct iopctl_group_t

31.3 Macro Definition Documentation

31.3.1 #define LPC_IOPCTL_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

31.3.2 #define IOPCTL_FUNC0 0x0

Note

See the User Manual for specific modes and functions supported by the various pins.Selects pin function 0

31.4 Function Documentation

31.4.1 __STATIC_INLINE void IOPCTL_PinMuxSet (IOPCTL_Type * *base,* uint8_t *port,* uint8_t *pin,* uint32_t *modefunc*)

Parameters

base	: The base of IOPCTL peripheral on the chip
port	: Port to mux
pin	: Pin to mux
modefunc	: OR'ed values of type IOPCTL_*

Returns

Nothing

31.4.2 __STATIC_INLINE void IOPCTL_SetPinMuxing (IOPCTL_Type * *base,* const iopctl_group_t * *pinArray,* uint32_t *arrayLength*)

Parameters

base	: The base of IOPCTL peripheral on the chip
pinArray	: Pointer to array of pin mux selections
arrayLength	: Number of entries in pinArray

Returns

Nothing

Chapter 32
RTC: Real Time Clock

32.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC).

32.2 Function groups

The RTC driver supports operating the module as a time counter.

32.2.1 Initialization and deinitialization

The function RTC_Init() initializes the RTC with specified configurations. The function RTC_GetDefaultConfig() gets the default configurations.

The function RTC_Deinit() disables the RTC timer and disables the module clock.

32.2.2 Set & Get Datetime

The function RTC_SetDatetime() sets the timer period in seconds. User passes in the details in date & time format by using the below data structure.

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtc The function RTC_GetDatetime() reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

32.2.3 Set & Get Alarm

The function RTC_SetAlarm() sets the alarm time period in seconds. User passes in the details in date & time format by using the datetime data structure.

The function RTC_GetAlarm() reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

32.2.4 Start & Stop timer

The function RTC_StartTimer() starts the RTC time counter.

The function RTC_StopTimer() stops the RTC time counter.

32.2.5 Status

Provides functions to get and clear the RTC status.

32.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

32.2.7 High resolution timer

Provides functions to enable high resolution timer and set and get the wake time.

32.3 Typical use case

32.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/rtc

Files

- file fsl_rtc.h

Data Structures

- struct rtc_datetime_t
  Structure is used to hold the date and time. More...

Enumerations

- enum rtc_interrupt_enable_t {
  kRTC_AlarmInterruptEnable = RTC_CTRL_ALARMDPD_EN_MASK,
  kRTC_WakeupInterruptEnable = RTC_CTRL_WAKEDPD_EN_MASK }
  List of RTC interrupts.
- enum rtc_status_flags_t {
  kRTC_AlarmFlag = RTC_CTRL_ALARM1HZ_MASK,
  kRTC_WakeupFlag = RTC_CTRL_WAKE1KHZ_MASK }
  List of RTC flags.

Functions

- static void RTC_SetSecondsTimerMatch (RTC_Type *base, uint32_t matchValue)
  Set the RTC seconds timer (1HZ) MATCH value.
- static uint32_t RTC_GetSecondsTimerMatch (RTC_Type *base)
  Read actual RTC seconds timer (1HZ) MATCH value.

- static void RTC_SetSecondsTimerCount (RTC_Type *base, uint32_t countValue)
  Set the RTC seconds timer (1HZ) COUNT value.
- static uint32_t RTC_GetSecondsTimerCount (RTC_Type *base)
  Read the actual RTC seconds timer (1HZ) COUNT value.
- static void RTC_SetWakeupCount (RTC_Type *base, uint16_t wakeupValue)
  Enable the RTC wake-up timer (1KHZ) and set countdown value to the RTC WAKE register.
- static uint16_t RTC_GetWakeupCount (RTC_Type *base)
  Read the actual value from the WAKE register value in RTC wake-up timer (1KHZ).
- static void RTC_Reset (RTC_Type *base)
  Perform a software reset on the RTC module.

Driver version

- #define FSL_RTC_DRIVER_VERSION (MAKE_VERSION(2, 1, 2))
  Version 2.1.2.

Initialization and deinitialization

- void RTC_Init (RTC_Type *base)
  Un-gate the RTC clock and enable the RTC oscillator.
- static void RTC_Deinit (RTC_Type *base)
  Stop the timer and gate the RTC clock.

Current Time & Alarm

- status_t RTC_SetDatetime (RTC_Type *base, const rtc_datetime_t *datetime)
  Set the RTC date and time according to the given time structure.
- void RTC_GetDatetime (RTC_Type *base, rtc_datetime_t *datetime)
  Get the RTC time and stores it in the given time structure.
- status_t RTC_SetAlarm (RTC_Type *base, const rtc_datetime_t *alarmTime)
  Set the RTC alarm time.
- void RTC_GetAlarm (RTC_Type *base, rtc_datetime_t *datetime)
  Return the RTC alarm time.

RTC wake-up timer (1KHZ) Enable

- static void RTC_EnableWakeupTimer (RTC_Type *base, bool enable)
  Enable the RTC wake-up timer (1KHZ).
- static uint32_t RTC_GetEnabledWakeupTimer (RTC_Type *base)
  Get the enabled status of the RTC wake-up timer (1KHZ).

Interrupt Interface

- static void RTC_EnableWakeUpTimerInterruptFromDPD (RTC_Type *base, bool enable)
  Enable the wake-up timer interrupt from deep power down mode.
- static void RTC_EnableAlarmTimerInterruptFromDPD (RTC_Type *base, bool enable)
  Enable the alarm timer interrupt from deep power down mode.
- static void RTC_EnableInterrupts (RTC_Type *base, uint32_t mask)
  Enables the selected RTC interrupts.
- static void RTC_DisableInterrupts (RTC_Type *base, uint32_t mask)

Disables the selected RTC interrupts.
- static uint32_t RTC_GetEnabledInterrupts (RTC_Type *base)
  Get the enabled RTC interrupts.

Status Interface

- static uint32_t RTC_GetStatusFlags (RTC_Type *base)
  Get the RTC status flags.
- static void RTC_ClearStatusFlags (RTC_Type *base, uint32_t mask)
  Clear the RTC status flags.

Timer Enable

- static void RTC_EnableTimer (RTC_Type *base, bool enable)
  Enable the RTC timer counter.
- static void RTC_StartTimer (RTC_Type *base)
  Starts the RTC time counter.
- static void RTC_StopTimer (RTC_Type *base)
  Stops the RTC time counter.

32.4 Data Structure Documentation

32.4.1 struct rtc_datetime_t

Data Fields

- uint16_t year
  Range from 1970 to 2099.
- uint8_t month
  Range from 1 to 12.
- uint8_t day
  Range from 1 to 31 (depending on month).
- uint8_t hour
  Range from 0 to 23.
- uint8_t minute
  Range from 0 to 59.
- uint8_t second
  Range from 0 to 59.

Field Documentation

(1) uint16_t rtc_datetime_t::year

(2) uint8_t rtc_datetime_t::month

(3) uint8_t rtc_datetime_t::day

(4) uint8_t rtc_datetime_t::hour

(5) uint8_t rtc_datetime_t::minute

(6) uint8_t rtc_datetime_t::second

32.5 Enumeration Type Documentation

32.5.1 enum rtc_interrupt_enable_t

Enumerator

kRTC_AlarmInterruptEnable Alarm interrupt.
kRTC_WakeupInterruptEnable Wake-up interrupt.

32.5.2 enum rtc_status_flags_t

Enumerator

kRTC_AlarmFlag Alarm flag.
kRTC_WakeupFlag 1kHz wake-up timer flag

32.6 Function Documentation

32.6.1 void RTC_Init (RTC_Type ∗ *base*)

Note

This API should be called at the beginning of the application using the RTC driver.

Parameters

base	RTC peripheral base address

32.6.2 static void RTC_Deinit (RTC_Type ∗ *base*) `[inline], [static]`

Parameters

base	RTC peripheral base address

32.6.3 status_t RTC_SetDatetime (RTC_Type ∗ *base,* const rtc_datetime_t ∗ *datetime*)

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

Parameters

base	RTC peripheral base address
datetime	Pointer to structure where the date and time details to set are stored

Returns

kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

32.6.4 void RTC_GetDatetime (RTC_Type * *base,* rtc_datetime_t * *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to structure where the date and time details are stored.

32.6.5 status_t RTC_SetAlarm (RTC_Type * *base,* const rtc_datetime_t * *alarmTime*)

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

base	RTC peripheral base address
alarmTime	Pointer to structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

32.6.6 void RTC_GetAlarm (RTC_Type * *base,* rtc_datetime_t * *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to structure where the alarm date and time details are stored.

32.6.7 static void RTC_EnableWakeupTimer (RTC_Type * *base,* bool *enable*) [inline], [static]

After calling this function, the RTC driver will use/un-use the RTC wake-up (1KHZ) at the same time.

Parameters

base	RTC peripheral base address
enable	Use/Un-use the RTC wake-up timer. • true: Use RTC wake-up timer at the same time. • false: Un-use RTC wake-up timer, RTC only use the normal seconds timer by default.

32.6.8 static uint32_t RTC_GetEnabledWakeupTimer (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The enabled status of RTC wake-up timer (1KHZ).

32.6.9 static void RTC_SetSecondsTimerMatch (RTC_Type * *base,* uint32_t *matchValue*) [inline], [static]

Parameters

base	RTC peripheral base address
matchValue	The value to be set into the RTC MATCH register

32.6.10 static uint32_t RTC_GetSecondsTimerMatch (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The actual RTC seconds timer (1HZ) MATCH value.

32.6.11 static void RTC_SetSecondsTimerCount (RTC_Type * *base,* uint32_t *countValue*) [inline], [static]

Parameters

base	RTC peripheral base address
countValue	The value to be loaded into the RTC COUNT register

32.6.12 static uint32_t RTC_GetSecondsTimerCount (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The actual RTC seconds timer (1HZ) COUNT value.

32.6.13 static void RTC_SetWakeupCount (RTC_Type * *base,* uint16_t *wakeupValue*) [inline], [static]

Parameters

base	RTC peripheral base address
wakeupValue	The value to be loaded into the WAKE register in RTC wake-up timer (1KHZ).

32.6.14 static uint16_t RTC_GetWakeupCount (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The actual value of the WAKE register value in RTC wake-up timer (1HZ).

32.6.15 static void RTC_EnableWakeUpTimerInterruptFromDPD (RTC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	RTC peripheral base address
enable	Enable/Disable wake-up timer interrupt from deep power down mode. • true: Enable wake-up timer interrupt from deep power down mode. • false: Disable wake-up timer interrupt from deep power down mode.

32.6.16 static void RTC_EnableAlarmTimerInterruptFromDPD (RTC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	RTC peripheral base address

enable	Enable/Disable alarm timer interrupt from deep power down mode. • true: Enable alarm timer interrupt from deep power down mode. • false: Disable alarm timer interrupt from deep power down mode.

32.6.17 static void RTC_EnableInterrupts (RTC_Type * *base*, uint32_t *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by RTC_EnableAlarmTimerInterruptFromDPD and RTC_EnableWakeUpTimerInterruptFromDPD

Parameters

base	RTC peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration rtc_interrupt_enable_t

32.6.18 static void RTC_DisableInterrupts (RTC_Type * *base*, uint32_t *mask*) [inline], [static]

Deprecated Do not use this function. It has been superceded by RTC_EnableAlarmTimerInterruptFromDPD and RTC_EnableWakeUpTimerInterruptFromDPD

Parameters

base	RTC peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration rtc_interrupt_enable_t

32.6.19 static uint32_t RTC_GetEnabledInterrupts (RTC_Type * *base*) [inline], [static]

Deprecated Do not use this function. It will be deleted in next release version.

Parameters

base	RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration rtc_interrupt_enable_t

32.6.20 static uint32_t RTC_GetStatusFlags (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration rtc_status_flags_t

32.6.21 static void RTC_ClearStatusFlags (RTC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	RTC peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration rtc_-status_flags_t

32.6.22 static void RTC_EnableTimer (RTC_Type * *base,* bool *enable*) [inline], [static]

After calling this function, the RTC inner counter increments once a second when only using the RTC seconds timer (1hz), while the RTC inner wake-up timer countdown once a millisecond when using RTC wake-up timer (1KHZ) at the same time. RTC timer contain two timers, one is the RTC normal seconds timer, the other one is the RTC wake-up timer, the RTC enable bit is the master switch for the whole RTC timer, so user can use the RTC seconds (1HZ) timer independly, but they can't use the RTC wake-up timer (1KHZ) independently.

Parameters

base	RTC peripheral base address
enable	Enable/Disable RTC Timer counter. • true: Enable RTC Timer counter. • false: Disable RTC Timer counter.

32.6.23 static void RTC_StartTimer (RTC_Type * *base*) [inline], [static]

Deprecated Do not use this function. It has been superceded by RTC_EnableTimer

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

base	RTC peripheral base address

32.6.24 static void RTC_StopTimer (RTC_Type * *base*) [inline], [static]

Deprecated Do not use this function. It has been superceded by RTC_EnableTimer

RTC's seconds register can be written to only when the timer is stopped.

Parameters

base	RTC peripheral base address

32.6.25 static void RTC_Reset (RTC_Type * *base*) [inline], [static]

This resets all RTC registers to their reset value. The bit is cleared by software explicitly clearing it.

Parameters

base	RTC peripheral base address

Chapter 33
MIPI_DSI: MIPI DSI Host Controller

33.1 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 SMARTDMA driver

33.2 MIPI DSI Driver

The MIPI DSI driver supports both video mode and command mode.

33.2.1 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.

33.2.2 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 in 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_HostProtocalViolation = (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_InterruptGroup1HtxTo = (1U << 29U),
kDSI_InterruptGroup1LrxTo = (1U << 30U),
kDSI_InterruptGroup1BtaTo = (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 {

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, 1, 2))

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.

- 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.

33.2.3 Data Structure Documentation

33.2.3.1 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 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.

Field Documentation

(1) uint8_t dsi_config_t::numLanes

(2) bool dsi_config_t::enableNonContinuousHsClk

(3) bool dsi_config_t::autoInsertEoTp

(4) uint8_t dsi_config_t::numExtraEoTp

(5) uint32_t dsi_config_t::htxTo_ByteClk

(6) uint32_t dsi_config_t::lrxHostTo_ByteClk

(7) uint32_t dsi_config_t::btaTo_ByteClk

33.2.3.2 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.

Field Documentation

(1) uint16_t dsi_dpi_config_t::pixelPayloadSize

Recommended that the line size (in pixels) is evenly divisible by this parameter.

(2) dsi_dpi_color_coding_t dsi_dpi_config_t::dpiColorCoding

(3) dsi_dpi_pixel_packet_t dsi_dpi_config_t::pixelPacket

(4) dsi_dpi_video_mode_t dsi_dpi_config_t::videoMode

(5) dsi_dpi_bllp_mode_t dsi_dpi_config_t::bllpMode

(6) uint8_t dsi_dpi_config_t::polarityFlags

(7) uint16_t dsi_dpi_config_t::hfp

(8) uint16_t dsi_dpi_config_t::hbp

(9) uint16_t dsi_dpi_config_t::hsw

(10) uint8_t dsi_dpi_config_t::vfp

(11) uint8_t dsi_dpi_config_t::vbp

(12) uint16_t dsi_dpi_config_t::panelHeight

(13) uint8_t dsi_dpi_config_t::virtualChannel

33.2.3.3 struct dsi_dphy_config_t

Data Fields

- uint32_t txHsBitClk_Hz
  The generated HS TX bit clock in Hz.
- uint8_t tClkPre_ByteClk
  TLPX + TCLK-PREPARE + TCLK-ZERO + TCLK-PRE in byte clock.
- uint8_t tClkPost_ByteClk
  TCLK-POST + T_CLK-TRAIL in byte clock.
- uint8_t tHsExit_ByteClk
  THS-EXIT in byte clock.
- 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 + 4∗UI in clk_byte.
- uint8_t tClkTrail_ByteClk
  TCLK-TRAIL + 4∗UI in clk_byte.

Field Documentation

(1) uint32_t dsi_dphy_config_t::txHsBitClk_Hz

(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.

(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.

(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.

(5) 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.

(6) 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.

(7) 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.

(8) 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.

(9) 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.

(10) 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.

33.2.3.4 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 TX 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.

Field Documentation

(1) uint8_t dsi_transfer_t::virtualChannel

(2) dsi_tx_data_type_t dsi_transfer_t::txDataType

(3) uint8_t dsi_transfer_t::flags

(4) const uint8_t∗ dsi_transfer_t::txData

(5) uint8_t∗ dsi_transfer_t::rxData

(6) uint16_t dsi_transfer_t::txDataSize

(7) uint16_t dsi_transfer_t::rxDataSize

(8) bool dsi_transfer_t::sendDscCmd

(9) uint8_t dsi_transfer_t::dscCmd

33.2.3.5 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.

Field Documentation

(1) volatile bool dsi_handle_t::isBusy

(2) dsi_transfer_t dsi_handle_t::xfer

33.2.4 Typedef Documentation

33.2.4.1 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.

33.2.5 Enumeration Type Documentation

33.2.5.1 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.

33.2.5.2 enum dsi_dpi_color_coding_t

Enumerator

kDSI_Dpi16BitConfig1 16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBBBBB.
kDSI_Dpi16BitConfig2 16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBBBBB.
kDSI_Dpi16BitConfig3 16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBBBBX.
kDSI_Dpi18BitConfig1 18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBBBBB.
kDSI_Dpi18BitConfig2 18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBBBBB.
kDSI_Dpi24Bit 24-bit.

33.2.5.3 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.

33.2.5.4 anonymous enum

Enumerator

kDSI_DpiVsyncActiveLow VSYNC active low.
kDSI_DpiHsyncActiveLow HSYNC active low.
kDSI_DpiVsyncActiveHigh VSYNC active high.
kDSI_DpiHsyncActiveHigh HSYNC active high.

33.2.5.5 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.

33.2.5.6 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.

33.2.5.7 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.

33.2.5.8 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.

33.2.5.9 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_HostProtocalViolation DSI protocal violation.
kDSI_HostResetTriggerReceived Reset trigger received.
kDSI_HostTearTriggerReceived Tear effect trigger receive.
kDSI_HostAckTriggerReceived Acknowledge trigger message received.

33.2.5.10 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 protocal violation.
kDSI_InterruptGroup1ResetTriggerReceived Reset trigger received.
kDSI_InterruptGroup1TearTriggerReceived Tear effect trigger receive.
kDSI_InterruptGroup1AckTriggerReceived Acknowledge trigger message received.
kDSI_InterruptGroup1HtxTo High speed TX timeout.
kDSI_InterruptGroup1LrxTo Low power RX timeout.
kDSI_InterruptGroup1BtaTo Host BTA timeout.
kDSI_InterruptGroup2EccOneBit Sinle bit ECC error.
kDSI_InterruptGroup2EccMultiBit Multi bit ECC error.
kDSI_InterruptGroup2CrcError CRC error.

33.2.5.11 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.

33.2.5.12 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.

33.2.5.13 anonymous enum

Enumerator

kDSI_TransferUseHighSpeed Use high speed mode or not.
kDSI_TransferPerformBTA Perform BTA or not.

33.2.6 Function Documentation

33.2.6.1 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.

Parameters

base	MIPI DSI host peripheral base address.
config	Pointer to a user-defined configuration structure.

33.2.6.2 void DSI_Deinit (MIPI_DSI_HOST_Type * *base*)

This function should be called after all bother MIPI DSI driver functions.

Parameters

base	MIPI DSI host peripheral base address.

33.2.6.3 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

config	Pointer to a user-defined configuration structure.

33.2.6.4 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

base	MIPI DSI host peripheral base address.
config	Pointer to the DPI interface configuration.
numLanes	Lane number, should be same with the setting in dsi_dpi_config_t.
dpiPixelClk-Freq_Hz	The DPI pixel clock frequency in Hz.
dsiHsBitClk-Freq_Hz	The DSI high speed bit clock frequency in Hz. It is the same with DPHY PLL output.

33.2.6.5 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_MIPI_-DSI_NO_DPHY_PLL. For these platforms, the `refClkFreq_Hz` is useless.

Parameters

base	MIPI DSI host peripheral base address.
config	Pointer to the D-PHY configuration.
refClkFreq_Hz	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.

33.2.6.6 void DSI_DeinitDphy (MIPI_DSI_HOST_Type * *base*)

Power down the D-PHY PLL and shut down D-PHY.

Parameters

base	MIPI DSI host peripheral base address.

33.2.6.7 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.

Parameters

config	Pointer to the D-PHY configuration.
txHsBitClk_Hz	High speed bit clock in Hz.
txEscClk_Hz	Esc clock in Hz.

33.2.6.8 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

base	MIPI DSI host peripheral base address.
intGroup1	Interrupts to enable in group 1.
intGroup2	Interrupts to enable in group 2.

33.2.6.9 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

base	MIPI DSI host peripheral base address.
intGroup1	Interrupts to disable in group 1.

intGroup2	Interrupts to disable in group 2.

33.2.6.10 static void DSI_GetAndClearInterruptStatus (MIPI_DSI_HOST_Type * *base,* uint32_t * *intGroup1,* uint32_t * *intGroup2*) [inline], [static]

Parameters

base	MIPI DSI host peripheral base address.
intGroup1	Group 1 interrupt status.
intGroup2	Group 2 interrupt status.

33.2.6.11 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

base	MIPI DSI host peripheral base address.
wordCount	For long packet, this is the byte count of the payload. For short packet, this is (data1 << 8) | data0.
virtualChannel	Virtual channel.
dataType	The packet data type, (DI).
flags	The transfer control flags, see _dsi_transfer_flags.

33.2.6.12 void DSI_WriteApbTxPayload (MIPI_DSI_HOST_Type * *base,* const uint8_t * *payload,* uint16_t *payloadSize*)

Write the long packet payload to TX FIFO.

Parameters

base	MIPI DSI host peripheral base address.
payload	Pointer to the payload.
payloadSize	Payload size in byte.

33.2.6.13 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.
2. The DSC command in 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

base	MIPI DSI host peripheral base address.
payload	Pointer to the payload.
payloadSize	Payload size in byte.
sendDscCmd	If set to true, the DSC command is specified by `dscCmd`, otherwise the DSC command is included in the `payload`.
dscCmd	The DSC command to send, only used when `sendDscCmd` is true.

33.2.6.14 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

base	MIPI DSI host peripheral base address.
payload	Pointer to the payload.

payloadSize	Payload size in byte.

33.2.6.15 static void DSI_SendApbPacket (MIPI_DSI_HOST_Type * *base*) [inline], [static]

Send the packet set by DSI_SetApbPacketControl.

Parameters

base	MIPI DSI host peripheral base address.

33.2.6.16 static uint32_t DSI_GetApbStatus (MIPI_DSI_HOST_Type * *base*) [inline], [static]

The return value is OR'ed value of _dsi_apb_status.

Parameters

base	MIPI DSI host peripheral base address.

Returns

The APB status.

33.2.6.17 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

base	MIPI DSI host peripheral base address.

Returns

The error status.

33.2.6.18 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

rxErrorStatus	The error status returned by DSI_GetRxErrorStatus.

Returns

The 1-bit ECC error position.

33.2.6.19 static uint32_t DSI_GetAndClearHostStatus (MIPI_DSI_HOST_Type * *base*) [inline], [static]

The host status are returned as mask value of _dsi_host_status.

Parameters

base	MIPI DSI host peripheral base address.

Returns

The DSI host status.

33.2.6.20 static uint32_t DSI_GetRxPacketHeader (MIPI_DSI_HOST_Type * *base*) [inline], [static]

Parameters

base	MIPI DSI host peripheral base address.

Returns

The RX packet header.

33.2.6.21 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

rxPktHeader	The RX packet header get by DSI_GetRxPacketHeader.

Returns

The RX packet type.

33.2.6.22 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

rxPktHeader	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.

33.2.6.23 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

rxPktHeader	The RX packet header get by DSI_GetRxPacketHeader.

Returns

The virtual channel.

33.2.6.24 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

base	MIPI DSI host peripheral base address.
xfer	Pointer to the transfer structure.

Return values

kStatus_Success	Data transfer finished with no error.
kStatus_Timeout	Transfer failed because of timeout.
kStatus_DSI_RxData-Error	RX data error, user could use DSI_GetRxErrorStatus to check the error details.
kStatus_DSI_Error-ReportReceived	Error Report packet received, user could use DSI_GetAndClearHostStatus to check the error report status.
kStatus_DSI_Not-Supported	Transfer format not supported.
kStatus_DSI_Fail	Transfer failed for other reasons.

33.2.6.25 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

base	MIPI DSI host peripheral base address.
handle	Handle pointer.
callback	Callback function.
userData	User data.

33.2.6.26 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

base	MIPI DSI host peripheral base address.
handle	pointer to dsi_handle_t structure which stores the transfer state.
xfer	Pointer to the transfer structure.

Return values

kStatus_Success	Data transfer started successfully.
kStatus_DSI_Busy	Failed to start transfer because DSI is busy with pervious transfer.
kStatus_DSI_Not-Supported	Transfer format not supported.

33.2.6.27 void DSI_TransferAbort (MIPI_DSI_HOST_Type ∗ *base,* dsi_handle_t ∗ *handle*)

Parameters

base	MIPI DSI host peripheral base address.
handle	pointer to dsi_handle_t structure which stores the transfer state.

33.2.6.28 void DSI_TransferHandleIRQ (MIPI_DSI_HOST_Type ∗ *base,* dsi_handle_t ∗ *handle*)

Parameters

base	MIPI DSI host peripheral base address.
handle	pointer to dsi_handle_t structure which stores the transfer state.

33.3 MIPI DSI SMARTDMA driver

33.3.1 Overview

Data Structures

- struct dsi_smartdma_write_mem_transfer_t
  The pixel format sent on MIPI DSI data lanes. More...
- struct dsi_smartdma_handle_t
  MIPI DSI transfer handle structure. More...

Typedefs

- typedef void(∗ dsi_smartdma_callback_t)(MIPI_DSI_HOST_Type ∗base, dsi_smartdma_handle_t ∗handle, status_t status, void ∗userData)
  MIPI DSI callback for finished transfer.

Enumerations

- enum dsi_smartdma_input_pixel_format_t {
  kDSI_SMARTDMA_InputPixelFormatRGB565,
  kDSI_SMARTDMA_InputPixelFormatRGB888,
  kDSI_SMARTDMA_InputPixelFormatXRGB8888 }
  The pixel format feed SMARTDMA.
- enum dsi_smartdma_output_pixel_format_t {
  kDSI_SMARTDMA_OutputPixelFormatRGB565,
  kDSI_SMARTDMA_OutputPixelFormatRGB888 }
  The pixel format sent on MIPI DSI data lanes.

Driver version

- #define **FSL_MIPI_DSI_SMARTDMA_DRIVER_VERSION** (MAKE_VERSION(2, 2, 1))

Transactional

- status_t DSI_TransferCreateHandleSMARTDMA (MIPI_DSI_HOST_Type ∗base, dsi_smartdma_handle_t ∗handle, dsi_smartdma_callback_t callback, void ∗userData)
  Create the MIPI DSI SMARTDMA handle.
- status_t DSI_TransferWriteMemorySMARTDMA (MIPI_DSI_HOST_Type ∗base, dsi_smartdma_handle_t ∗handle, dsi_smartdma_write_mem_transfer_t ∗xfer)
  Write display controller video memory using SMARTDMA.
- void DSI_TransferAbortSMARTDMA (MIPI_DSI_HOST_Type ∗base, dsi_smartdma_handle_t ∗handle)
  Abort current APB data transfer.

33.3.2 Data Structure Documentation

33.3.2.1 struct dsi_smartdma_write_mem_transfer_t

Data Fields

- dsi_smartdma_input_pixel_format_t inputFormat
  Input format.
- dsi_smartdma_output_pixel_format_t outputFormat
  Output format.
- const uint8_t ∗ data
  Data to send.
- size_t dataSize
  The byte count to be write.
- uint8_t virtualChannel
  Virtual channel used in the transfer, current driver always use channel 0, added for future enhancement.
- bool disablePixelByteSwap
  If set to true, the pixels are filled to MIPI DSI FIFO directly.

Field Documentation

(1) dsi_smartdma_input_pixel_format_t dsi_smartdma_write_mem_transfer_t::inputFormat

(2) dsi_smartdma_output_pixel_format_t dsi_smartdma_write_mem_transfer_t::outputFormat

(3) const uint8_t∗ dsi_smartdma_write_mem_transfer_t::data

(4) size_t dsi_smartdma_write_mem_transfer_t::dataSize

(5) uint8_t dsi_smartdma_write_mem_transfer_t::virtualChannel

(6) bool dsi_smartdma_write_mem_transfer_t::disablePixelByteSwap

If set to false, the pixel bytes are swapped then filled to MIPI DSI FIFO. For example, when set to false and frame buffer pixel format is RGB565: LSB MSB B0 B1 B2 B3 B4 G0 G1 G2 | G3 G4 G5 R0 R1 R2 R3 R4 Then the pixel filled to DSI FIFO is: LSB MSB G3 G4 G5 R0 R1 R2 R3 R4 | B0 B1 B2 B3 B4 G0 G1 G2

33.3.2.2 struct _dsi_smartdma_handle

Data Fields

- MIPI_DSI_HOST_Type ∗ dsi
  MIPI DSI peripheral.
- volatile bool isBusy
  MIPI DSI is busy with data transfer.
- dsi_smartdma_callback_t callback
  DSI callback.
- void ∗ userData

Callback parameter.
- smartdma_dsi_param_t param
  Parameter for smartdma function.
- uint32_t smartdmaStack [16]
  Stack for smartdma function.

Field Documentation

(1) MIPI_DSI_HOST_Type∗ dsi_smartdma_handle_t::dsi

(2) volatile bool dsi_smartdma_handle_t::isBusy

(3) smartdma_dsi_param_t dsi_smartdma_handle_t::param

(4) uint32_t dsi_smartdma_handle_t::smartdmaStack[16]

33.3.3 Typedef Documentation

33.3.3.1 typedef void(∗ dsi_smartdma_callback_t)(MIPI_DSI_HOST_Type ∗base, dsi_smartdma_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.

33.3.4 Enumeration Type Documentation

33.3.4.1 enum dsi_smartdma_input_pixel_format_t

Enumerator

kDSI_SMARTDMA_InputPixelFormatRGB565 RGB565.
kDSI_SMARTDMA_InputPixelFormatRGB888 RGB888.
kDSI_SMARTDMA_InputPixelFormatXRGB8888 XRGB8888.

33.3.4.2 enum dsi_smartdma_output_pixel_format_t

Enumerator

kDSI_SMARTDMA_OutputPixelFormatRGB565 RGB565.
kDSI_SMARTDMA_OutputPixelFormatRGB888 RGB888.

33.3.5 Function Documentation

33.3.5.1 status_t DSI_TransferCreateHandleSMARTDMA (MIPI_DSI_HOST_Type * *base,* dsi_smartdma_handle_t * *handle,* dsi_smartdma_callback_t *callback,* void * *userData*)

Parameters

base	MIPI DSI host peripheral base address.
handle	Handle pointer.
callback	Callback function.
userData	User data.

33.3.5.2 status_t DSI_TransferWriteMemorySMARTDMA (MIPI_DSI_HOST_Type * *base*, dsi_smartdma_handle_t * *handle*, dsi_smartdma_write_mem_transfer_t * *xfer*)

Perform data transfer using SMARTDMA, when transfer finished, upper layer could be informed through callback function.

Parameters

base	MIPI DSI host peripheral base address.
handle	pointer to dsi_smartdma_handle_t structure which stores the transfer state.
xfer	Pointer to the transfer structure.

Return values

kStatus_Success	Data transfer started successfully.
kStatus_DSI_Busy	Failed to start transfer because DSI is busy with pervious transfer.
kStatus_DSI_Not-Supported	Transfer format not supported.

33.3.5.3 void DSI_TransferAbortSMARTDMA (MIPI_DSI_HOST_Type * *base*, dsi_smartdma_handle_t * *handle*)

Parameters

base	MIPI DSI host peripheral base address.
handle	pointer to dsi_smartdma_handle_t structure which stores the transfer state.

Chapter 34
MRT: Multi-Rate Timer

34.1 Overview

The MCUXpresso SDK provides a driver for the Multi-Rate Timer (MRT) of MCUXpresso SDK devices.

34.2 Function groups

The MRT driver supports operating the module as a time counter.

34.2.1 Initialization and deinitialization

The function MRT_Init() initializes the MRT with specified configurations. The function MRT_GetDefaultConfig() gets the default configurations. The initialization function configures the MRT operating mode.

The function MRT_Deinit() stops the MRT timers and disables the module clock.

34.2.2 Timer period Operations

The function MRT_UpdateTimerPeriod() is used to update the timer period in units of count. The new value is immediately loaded or will be loaded at the end of the current time interval.

The function MRT_GetCurrentTimerCount() reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. The user can call the utility macros provided in fsl_common.h to convert to microseconds or milliseconds

34.2.3 Start and Stop timer operations

The function MRT_StartTimer() starts the timer counting. After calling this function, the timer loads the period value, counts down to 0 and depending on the timer mode it either loads the respective start value again or stop. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function MRT_StopTimer() stops the timer counting.

34.2.4 Get and release channel

These functions can be used to reserve and release a channel. The function MRT_GetIdleChannel() finds the available channel. This function returns the lowest available channel number. The function MRT_-ReleaseChannel() release the channel when the timer is using the multi-task mode. In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use.

34.2.5 Status

Provides functions to get and clear the PIT status.

34.2.6 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

34.3 Typical use case

34.3.1 MRT tick example

Updates the MRT period and toggles an LED periodically. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/mrt

Files

- file fsl_mrt.h

Data Structures

- struct mrt_config_t
  MRT configuration structure. More...

Enumerations

- enum mrt_chnl_t {
  kMRT_Channel_0 = 0U,
  kMRT_Channel_1,
  kMRT_Channel_2,
  kMRT_Channel_3 }
  List of MRT channels.
- enum mrt_timer_mode_t {
  kMRT_RepeatMode = (0 << MRT_CHANNEL_CTRL_MODE_SHIFT),
  kMRT_OneShotMode = (1 << MRT_CHANNEL_CTRL_MODE_SHIFT),
  kMRT_OneShotStallMode = (2 << MRT_CHANNEL_CTRL_MODE_SHIFT) }
  List of MRT timer modes.

- enum mrt_interrupt_enable_t { kMRT_TimerInterruptEnable = MRT_CHANNEL_CTRL_INTEN_MASK }
  List of MRT interrupts.
- enum mrt_status_flags_t {
  kMRT_TimerInterruptFlag = MRT_CHANNEL_STAT_INTFLAG_MASK,
  kMRT_TimerRunFlag = MRT_CHANNEL_STAT_RUN_MASK }
  List of MRT status flags.

Driver version

- #define FSL_MRT_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  Version 2.0.3.

Initialization and deinitialization

- void MRT_Init (MRT_Type *base, const mrt_config_t *config)
  Ungates the MRT clock and configures the peripheral for basic operation.
- void MRT_Deinit (MRT_Type *base)
  Gate the MRT clock.
- static void MRT_GetDefaultConfig (mrt_config_t *config)
  Fill in the MRT config struct with the default settings.
- static void MRT_SetupChannelMode (MRT_Type *base, mrt_chnl_t channel, const mrt_timer_mode_t mode)
  Sets up an MRT channel mode.

Interrupt Interface

- static void MRT_EnableInterrupts (MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
  Enables the MRT interrupt.
- static void MRT_DisableInterrupts (MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
  Disables the selected MRT interrupt.
- static uint32_t MRT_GetEnabledInterrupts (MRT_Type *base, mrt_chnl_t channel)
  Gets the enabled MRT interrupts.

Status Interface

- static uint32_t MRT_GetStatusFlags (MRT_Type *base, mrt_chnl_t channel)
  Gets the MRT status flags.
- static void MRT_ClearStatusFlags (MRT_Type *base, mrt_chnl_t channel, uint32_t mask)
  Clears the MRT status flags.

Read and Write the timer period

- void MRT_UpdateTimerPeriod (MRT_Type *base, mrt_chnl_t channel, uint32_t count, bool immediateLoad)
  Used to update the timer period in units of count.
- static uint32_t MRT_GetCurrentTimerCount (MRT_Type *base, mrt_chnl_t channel)
  Reads the current timer counting value.

Timer Start and Stop

- static void MRT_StartTimer (MRT_Type *base, mrt_chnl_t channel, uint32_t count)
  Starts the timer counting.
- static void MRT_StopTimer (MRT_Type *base, mrt_chnl_t channel)
  Stops the timer counting.

Get & release channel

- static uint32_t MRT_GetIdleChannel (MRT_Type *base)
  Find the available channel.
- static void MRT_ReleaseChannel (MRT_Type *base, mrt_chnl_t channel)
  Release the channel when the timer is using the multi-task mode.

34.4 Data Structure Documentation

34.4.1 struct mrt_config_t

This structure holds the configuration settings for the MRT peripheral. To initialize this structure to reasonable defaults, call the MRT_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Data Fields

- bool enableMultiTask
  true: Timers run in multi-task mode; false: Timers run in hardware status mode

34.5 Enumeration Type Documentation

34.5.1 enum mrt_chnl_t

Enumerator

kMRT_Channel_0 MRT channel number 0.
kMRT_Channel_1 MRT channel number 1.
kMRT_Channel_2 MRT channel number 2.
kMRT_Channel_3 MRT channel number 3.

34.5.2 enum mrt_timer_mode_t

Enumerator

kMRT_RepeatMode Repeat Interrupt mode.
kMRT_OneShotMode One-shot Interrupt mode.
kMRT_OneShotStallMode One-shot stall mode.

34.5.3 enum mrt_interrupt_enable_t

Enumerator

kMRT_TimerInterruptEnable Timer interrupt enable.

34.5.4 enum mrt_status_flags_t

Enumerator

kMRT_TimerInterruptFlag Timer interrupt flag.
kMRT_TimerRunFlag Indicates state of the timer.

34.6 Function Documentation

34.6.1 void MRT_Init (MRT_Type ∗ *base,* const mrt_config_t ∗ *config*)

Note

This API should be called at the beginning of the application using the MRT driver.

Parameters

base	Multi-Rate timer peripheral base address
config	Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MODCFG reigster, param config is useless.

34.6.2 void MRT_Deinit (MRT_Type ∗ *base*)

Parameters

base	Multi-Rate timer peripheral base address

34.6.3 static void MRT_GetDefaultConfig (mrt_config_t ∗ *config*) [inline], [static]

The default values are:

```
*     config->enableMultiTask = false;
*
```

Parameters

config	Pointer to user's MRT config structure.

34.6.4 static void MRT_SetupChannelMode (MRT_Type * *base*, mrt_chnl_t *channel*, const mrt_timer_mode_t *mode*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Channel that is being configured.
mode	Timer mode to use for the channel.

34.6.5 static void MRT_EnableInterrupts (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number
mask	The interrupts to enable. This is a logical OR of members of the enumeration mrt_-interrupt_enable_t

34.6.6 static void MRT_DisableInterrupts (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number
mask	The interrupts to disable. This is a logical OR of members of the enumeration mrt_-interrupt_enable_t

34.6.7 static uint32_t MRT_GetEnabledInterrupts (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration mrt_interrupt_enable_t

34.6.8 static uint32_t MRT_GetStatusFlags (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration mrt_status_flags_t

34.6.9 static void MRT_ClearStatusFlags (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number
mask	The status flags to clear. This is a logical OR of members of the enumeration mrt_status_flags_t

34.6.10 void MRT_UpdateTimerPeriod (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *count*, bool *immediateLoad*)

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

Note

User can call the utility macros provided in fsl_common.h to convert to ticks

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number
count	Timer period in units of ticks
immediateLoad	true: Load the new value immediately into the TIMER register; false: Load the new value at the end of current timer interval

34.6.11 static uint32_t MRT_GetCurrentTimerCount (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User can call the utility macros provided in fsl_common.h to convert ticks to usec or msec

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number

Returns

Current timer counting value in ticks

34.6.12 static void MRT_StartTimer (MRT_Type * *base*, mrt_chnl_t *channel*, uint32_t *count*) [inline], [static]

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

Note

User can call the utility macros provided in fsl_common.h to convert to ticks

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number.
count	Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature.

34.6.13 static void MRT_StopTimer (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

This function stops the timer from counting.

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number.

34.6.14 static uint32_t MRT_GetIdleChannel (MRT_Type * *base*) [inline], [static]

This function returns the lowest available channel number.

Parameters

base	Multi-Rate timer peripheral base address

34.6.15 static void MRT_ReleaseChannel (MRT_Type * *base*, mrt_chnl_t *channel*) [inline], [static]

In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use. The user can hold on to a channel acquired by calling MRT_GetIdleChannel() for as long as it is needed and release it by calling this function. This removes the need to ask for an available channel for every use.

Parameters

base	Multi-Rate timer peripheral base address
channel	Timer channel number.

Chapter 35
MU: Messaging Unit

35.1 Overview

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

35.2 Function description

The MU driver provides these functions:

- Functions to initialize the MU module.
- Functions to send and receive messages.
- Functions for MU flags for both MU sides.
- Functions for status flags and interrupts.
- Other miscellaneous functions.

35.2.1 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.

35.2.2 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_ReadMsg-NonBlocking() function is a non-blocking API. The MU_ReadMsg() function is the blocking API.

35.2.3 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.

35.2.4 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.

35.2.5 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_Set-ClockOnOtherCoreEnable() 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 = (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,
  kMU_ResetAssertInterruptFlag = MU_SR_RAIP_MASK,
  kMU_ResetDeassertInterruptFlag = MU_SR_RDIP_MASK,
  kMU_OtherSideInResetFlag = MU_SR_RS_MASK }
  MU status flags.
- enum _mu_interrupt_enable {
  kMU_Tx0EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 3U)),
  kMU_Tx1EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 2U)),
  kMU_Tx2EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 1U)),
  kMU_Tx3EmptyInterruptEnable = (1U << (MU_CR_TIEn_SHIFT + 0U)),
  kMU_Rx0FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 3U)),
  kMU_Rx1FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 2U)),
  kMU_Rx2FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 1U)),
  kMU_Rx3FullInterruptEnable = (1U << (MU_CR_RIEn_SHIFT + 0U)),
  kMU_GenInt0InterruptEnable = (int)(1U << (MU_CR_GIEn_SHIFT + 3U)),
  kMU_GenInt1InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 2U)),
  kMU_GenInt2InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 1U)),
  kMU_GenInt3InterruptEnable = (1U << (MU_CR_GIEn_SHIFT + 0U)),
  kMU_ResetAssertInterruptEnable = MU_CR_RAIE_MASK,
  kMU_ResetDeassertInterruptEnable = MU_CR_RDIE_MASK }
  MU interrupt source to enable.
- enum _mu_interrupt_trigger {
  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.
- enum mu_msg_reg_index_t
  MU message register.

Driver version

- #define FSL_MU_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  MU driver version.

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.

MU misc functions

- static void MU_ResetBothSides (MU_Type ∗base)
  Resets the MU for both A side and B side.

- static mu_power_mode_t MU_GetOtherCorePowerMode (MU_Type *base)
  Gets the power mode of the other core.

35.3 Macro Definition Documentation

35.3.1 #define FSL_MU_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

35.4 Enumeration Type Documentation

35.4.1 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.
kMU_ResetAssertInterruptFlag The other core reset assert interrupt pending.
kMU_ResetDeassertInterruptFlag The other core reset de-assert interrupt pending.
kMU_OtherSideInResetFlag The other side is in reset.

35.4.2 enum _mu_interrupt_enable

Enumerator

kMU_Tx0EmptyInterruptEnable TX0 empty.
kMU_Tx1EmptyInterruptEnable TX1 empty.
kMU_Tx2EmptyInterruptEnable TX2 empty.
kMU_Tx3EmptyInterruptEnable TX3 empty.
kMU_Rx0FullInterruptEnable RX0 full.
kMU_Rx1FullInterruptEnable RX1 full.
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.
kMU_ResetAssertInterruptEnable The other core reset assert interrupt.
kMU_ResetDeassertInterruptEnable The other core reset de-assert interrupt.

35.4.3 enum _mu_interrupt_trigger

Enumerator

kMU_GenInt0InterruptTrigger General purpose interrupt 0.
kMU_GenInt1InterruptTrigger General purpose interrupt 1.
kMU_GenInt2InterruptTrigger General purpose interrupt 2.
kMU_GenInt3InterruptTrigger General purpose interrupt 3.

35.5 Function Documentation

35.5.1 void MU_Init (MU_Type ∗ *base*)

This function enables the MU clock only.

Parameters

base	MU peripheral base address.

35.5.2 void MU_Deinit (MU_Type ∗ *base*)

This function disables the MU clock only.

Parameters

base	MU peripheral base address.

35.5.3 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, kMU_MsgReg0, MSG_VAL);  Write message to the TX0
      register.
*
```

Parameters

base	MU peripheral base address.
regIndex	TX register index, see mu_msg_reg_index_t.
msg	Message to send.

35.5.4 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

base	MU peripheral base address.
regIndex	MU message register, see mu_msg_reg_index_t
msg	Message to send.

35.5.5 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, kMU_MsgReg0);  Read message from RX0
      register.
*
```

Parameters

base	MU peripheral base address.
RX	register index, see mu_msg_reg_index_t.

Returns

The received message.

35.5.6 uint32_t MU_ReceiveMsg (MU_Type * *base,* uint32_t *regIndex*)

This function waits until the RX register is full and receives the message.

Parameters

base	MU peripheral base address.
regIndex	MU message register, see mu_msg_reg_index_t

Returns

The received message.

35.5.7 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

base	MU peripheral base address.
flags	The 3-bit MU flags to set.

35.5.8 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

base	MU peripheral base address.
flags	The 3-bit MU flags to set.

35.5.9 static uint32_t MU_GetFlags (MU_Type * *base*) [inline], [static]

This function gets the current 3-bit MU flags on the current side.

Parameters

base	MU peripheral base address.

Returns

flags Current value of the 3-bit flags.

35.5.10 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, kMU_MsgReg0, MSG0_VAL);
* }
* if (kMU_Tx1EmptyFlag & flags)
* {
*     The TX1 register is empty. Message can be sent.
*     MU_SendMsgNonBlocking(base, kMU_MsgReg1, MSG1_VAL);
* }
*
```

Parameters

base	MU peripheral base address.

Returns

Bit mask of the MU status flags, see _mu_status_flags.

35.5.11 static uint32_t MU_GetInterruptsPending (MU_Type * *base*) [inline], [static]

This function returns the bit mask of the pending MU IRQs.

Parameters

base	MU peripheral base address.

Returns

Bit mask of the MU IRQs pending.

35.5.12 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

base	MU peripheral base address.
mask	Bit mask of the MU status flags. See _mu_status_flags. The following flags are cleared by hardware, this function could not clear them. • kMU_Tx0EmptyFlag • kMU_Tx1EmptyFlag • kMU_Tx2EmptyFlag • kMU_Tx3EmptyFlag • kMU_Rx0FullFlag • kMU_Rx1FullFlag • kMU_Rx2FullFlag • kMU_Rx3FullFlag • kMU_EventPendingFlag • kMU_FlagsUpdatingFlag • kMU_OtherSideInResetFlag

35.5.13 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

base	MU peripheral base address.
mask	Bit mask of the MU interrupts. See _mu_interrupt_enable.

35.5.14 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);
*
```

Parameters

base	MU peripheral base address.
mask	Bit mask of the MU interrupts. See _mu_interrupt_enable.

35.5.15 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

base	MU peripheral base address.
mask	Bit mask of the interrupts to trigger. See _mu_interrupt_trigger.

Return values

kStatus_Success	Interrupts have been triggered successfully.
kStatus_Fail	Previous interrupts have not been accepted.

35.5.16 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

base	MU peripheral base address.

Note

For some platforms, only MU side A could use this function, check reference manual for details.

35.5.17 static mu_power_mode_t MU_GetOtherCorePowerMode (MU_Type * *base*) [inline], [static]

This function gets the power mode of the other core.

Parameters

base	MU peripheral base address.

Returns

Power mode of the other core.

Chapter 36
OSTIMER: OS Event Timer Driver

36.1 Overview

The MCUXpresso SDK provides a peripheral driver for the OSTIMER module of MCUXpresso SDK devices. OSTIMER driver is created to help user to operate the OSTIMER module. The OSTIMER timer can be used as a low power timer. The APIs can be used to enable the OSTIMER module, initialize it and set the match time, get the current timer count. And the raw value in OS timer register is gray-code type, so both decimal and gray-code format API were added for users. OSTIMER can be used as a wake up source from low power mode.

36.2 Function groups

The OSTIMER driver supports operating the module as a time counter.

36.2.1 Initialization and deinitialization

The OSTIMER_Init() function will initialize the OSTIMER and enable the clock for OSTIMER. The OSTIMER_Deinit() function will shut down the bus clock of OSTIMER.

36.2.2 OSTIMER status

The function OSTIMER_GetStatusFlags() will get the current status flag of OSTIMER. The function OSTIMER_ClearStatusFlag() will help clear the status flags.

36.2.3 OSTIMER set match value

For OSTIMER, allow users set the match in two ways, set match value with raw data(gray code) and st the match value with common data(decimal format). OSTIMER_SetMatchRawValue() is used with gray code and OSTIMER_SetMatchValue() is used together with decimal data.

36.2.4 OSTIMER get timer count

The OSTIMER driver allow users to get the timer count in two ways, getting the gray code value by using OSTIMER_GetCaptureRawValue() and getting the decimal data by using OSTIMER_GetCurrentTimerValue().

36.3 Typical use case

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

Files

- file fsl_ostimer.h

Typedefs

- typedef void(∗ ostimer_callback_t)(void)
  ostimer callback function.

Enumerations

- enum _ostimer_flags { kOSTIMER_MatchInterruptFlag = (OSTIMER_OSEVENT_CTRL_OSTIMER_INTRFLAG_MASK) }
  OSTIMER status flags.

Driver version

- #define FSL_OSTIMER_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  OSTIMER driver version.

Initialization and deinitialization

- void OSTIMER_Init (OSTIMER_Type ∗base)
  Initializes an OSTIMER by turning its bus clock on.
- void OSTIMER_Deinit (OSTIMER_Type ∗base)
  Deinitializes a OSTIMER instance.
- uint64_t OSTIMER_GrayToDecimal (uint64_t gray)
  Translate the value from gray-code to decimal.
- static uint64_t OSTIMER_DecimalToGray (uint64_t dec)
  Translate the value from decimal to gray-code.
- uint32_t OSTIMER_GetStatusFlags (OSTIMER_Type ∗base)
  Get OSTIMER status Flags.
- void OSTIMER_ClearStatusFlags (OSTIMER_Type ∗base, uint32_t mask)
  Clear Status Interrupt Flags.
- status_t OSTIMER_SetMatchRawValue (OSTIMER_Type ∗base, uint64_t count, ostimer_callback_t cb)
  Set the match raw value for OSTIMER.
- status_t OSTIMER_SetMatchValue (OSTIMER_Type ∗base, uint64_t count, ostimer_callback_t cb)
  Set the match value for OSTIMER.
- static void OSTIMER_SetMatchRegister (OSTIMER_Type ∗base, uint64_t value)
  Set value to OSTIMER MATCH register directly.
- static void OSTIMER_EnableMatchInterrupt (OSTIMER_Type ∗base)
  Enable the OSTIMER counter match interrupt.
- static void OSTIMER_DisableMatchInterrupt (OSTIMER_Type ∗base)
  Disable the OSTIMER counter match interrupt.

- static uint64_t OSTIMER_GetCurrentTimerRawValue (OSTIMER_Type *base)
  Get current timer raw count value from OSTIMER.
- uint64_t OSTIMER_GetCurrentTimerValue (OSTIMER_Type *base)
  Get current timer count value from OSTIMER.
- static uint64_t OSTIMER_GetCaptureRawValue (OSTIMER_Type *base)
  Get the capture value from OSTIMER.
- uint64_t OSTIMER_GetCaptureValue (OSTIMER_Type *base)
  Get the capture value from OSTIMER.
- void OSTIMER_HandleIRQ (OSTIMER_Type *base, ostimer_callback_t cb)
  OS timer interrupt Service Handler.

36.4 Macro Definition Documentation

36.4.1 #define FSL_OSTIMER_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

36.5 Typedef Documentation

36.5.1 typedef void(* ostimer_callback_t)(void)

36.6 Enumeration Type Documentation

36.6.1 enum _ostimer_flags

Enumerator

kOSTIMER_MatchInterruptFlag Match interrupt flag bit, sets if the match value was reached.

36.7 Function Documentation

36.7.1 void OSTIMER_Init (OSTIMER_Type * *base*)

36.7.2 void OSTIMER_Deinit (OSTIMER_Type * *base*)

This function shuts down OSTIMER bus clock

Parameters

base	OSTIMER peripheral base address.

36.7.3 uint64_t OSTIMER_GrayToDecimal (uint64_t *gray*)

Parameters

gray	The gray value input.

Returns

The decimal value.

36.7.4 static uint64_t OSTIMER_DecimalToGray (uint64_t *dec*) [inline], [static]

Parameters

dec	The decimal value.

Returns

The gray code of the input value.

36.7.5 uint32_t OSTIMER_GetStatusFlags (OSTIMER_Type * *base*)

This returns the status flag. Currently, only match interrupt flag can be got.

Parameters

base	OSTIMER peripheral base address.

Returns

status register value

36.7.6 void OSTIMER_ClearStatusFlags (OSTIMER_Type * *base,* uint32_t *mask*)

This clears intrrupt status flag. Currently, only match interrupt flag can be cleared.

Parameters

base	OSTIMER peripheral base address.
mask	Clear bit mask.

Returns

none

36.7.7 status_t OSTIMER_SetMatchRawValue (OSTIMER_Type * *base*, uint64_t *count*, ostimer_callback_t *cb*)

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central EVTIMER. Please note that, the data format is gray-code, if decimal data was desired, please using OSTIMER_SetMatchValue().

Parameters

base	OSTIMER peripheral base address.
count	OSTIMER timer match value.(Value is gray-code format)
cb	OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success	- Set match raw value and enable interrupt Successfully.
kStatus_Fail	- Set match raw value fail.

36.7.8 status_t OSTIMER_SetMatchValue (OSTIMER_Type * *base*, uint64_t *count*, ostimer_callback_t *cb*)

This function will set a match value for OSTIMER with an optional callback. And this callback will be called while the data in dedicated pair match register is equals to the value of central OS TIMER.

Parameters

base	OSTIMER peripheral base address.
count	OSTIMER timer match value.(Value is decimal format, and this value will be translate to Gray code internally.)
cb	OSTIMER callback (can be left as NULL if none, otherwise should be a void func(void)).

Return values

kStatus_Success	- Set match value and enable interrupt Successfully.
kStatus_Fail	- Set match value fail.

36.7.9 static void OSTIMER_SetMatchRegister (OSTIMER_Type * *base*, uint64_t *value*) [inline], [static]

This function writes the input value to OSTIMER MATCH register directly, it does not touch any other registers. Note that, the data format is gray-code. The function OSTIMER_DecimalToGray could convert decimal value to gray code.

Parameters

base	OSTIMER peripheral base address.
count	OSTIMER timer match value (Value is gray-code format).

36.7.10 static void OSTIMER_EnableMatchInterrupt (OSTIMER_Type * *base*) [inline], [static]

Enable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

base	OSTIMER peripheral base address.

36.7.11 static void OSTIMER_DisableMatchInterrupt (OSTIMER_Type * *base*) [inline], [static]

Disable the timer counter match interrupt. The interrupt happens when OSTIMER counter matches the value in MATCH registers.

Parameters

base	OSTIMER peripheral base address.

36.7.12 static uint64_t OSTIMER_GetCurrentTimerRawValue (OSTIMER_Type * *base*) [inline], [static]

This function will get a gray code type timer count value from OS timer register. The raw value of timer count is gray code format.

Parameters

base	OSTIMER peripheral base address.

Returns

Raw value of OSTIMER, gray code format.

36.7.13 uint64_t OSTIMER_GetCurrentTimerValue (OSTIMER_Type * *base*)

This function will get a decimal timer count value. The RAW value of timer count is gray code format, will be translated to decimal data internally.

Parameters

base	OSTIMER peripheral base address.

Returns

Value of OSTIMER which will be formated to decimal value.

36.7.14 static uint64_t OSTIMER_GetCaptureRawValue (OSTIMER_Type * *base*) [inline], [static]

This function will get a captured gray-code value from OSTIMER. The Raw value of timer capture is gray code format.

Parameters

base	OSTIMER peripheral base address.

Returns

Raw value of capture register, data format is gray code.

36.7.15 uint64_t OSTIMER_GetCaptureValue (OSTIMER_Type * *base*)

This function will get a capture decimal-value from OSTIMER. The RAW value of timer capture is gray code format, will be translated to decimal data internally.

Parameters

base	OSTIMER peripheral base address.

Returns

Value of capture register, data format is decimal.

36.7.16 void OSTIMER_HandleIRQ (OSTIMER_Type * *base*, ostimer_callback_t *cb*)

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in OSTIMER_SetMatchValue()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

base	OS timer peripheral base address.
cb	callback scheduled for this instance of OS timer

Returns

none

Chapter 37
OTFAD: On The Fly AES-128 Decryption Driver

37.1 Overview

The MCUXpresso SDK provides a peripheral driver for On The Fly AES-128 Decryption (OTFAD) module of MCUXpresso SDK devices.

This example code shows how to decrypt ciphertext in external memory using the OTFAD driver.

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

Data Structures

- struct otfad_encryption_config_t
  OTFAD encryption configuration structure. More...
- struct otfad_config_t
  OTFAD configuration structure. More...

Enumerations

- enum {
  kStatus_OTFAD_ResRegAccessMode = MAKE_STATUS(kStatusGroup_OTFAD, 0),
  kStatus_OTFAD_AddressError = MAKE_STATUS(kStatusGroup_OTFAD, 1),
  kStatus_OTFAD_RegionOverlap,
  kStatus_OTFAD_RegionMiss }
  Status codes for the OTFAD driver.
- enum {
  kOTFAD_Context_0 = 0U,
  kOTFAD_Context_1 = 1U,
  kOTFAD_Context_2 = 2U,
  kOTFAD_Context_3 = 3U }
  OTFAD context type.
- enum {
  kOTFAD_NRM = 0x00U,
  kOTFAD_SVM = 0x02U,
  kOTFAD_LDM = 0x03U }
  OTFAD operate mode.

Driver version

- #define FSL_OTFAD_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 3U))
  Driver version.

Initialization and deinitialization

- void OTFAD_GetDefaultConfig (otfad_config_t *config)
  OTFAD module initialization function.
- void OTFAD_Init (OTFAD_Type *base, const otfad_config_t *config)
  OTFAD module initialization function.
- void OTFAD_Deinit (OTFAD_Type *base)
  Deinitializes the OTFAD.

Status

- static uint32_t OTFAD_GetOperateMode (OTFAD_Type *base)
  OTFAD module get operate mode.
- static uint32_t OTFAD_GetStatus (OTFAD_Type *base)
  OTFAD module get status.

functional

- status_t OTFAD_SetEncryptionConfig (OTFAD_Type *base, const otfad_encryption_config_t *config)
  OTFAD module set encryption configuration.
- status_t OTFAD_GetEncryptionConfig (OTFAD_Type *base, otfad_encryption_config_t *config)
  OTFAD module get encryption configuration.
- status_t OTFAD_HitDetermination (OTFAD_Type *base, uint32_t address, uint8_t *contextIndex)
  OTFAD module do hit determination.

37.2 Data Structure Documentation

37.2.1 struct otfad_encryption_config_t

Data Fields

- bool valid
  The context is valid or not.
- bool AESdecryption
  AES decryption enable.
- uint8_t readOnly
  read write attribute for the entire set of context registers
- uint8_t contextIndex
  OTFAD context index.
- uint32_t startAddr
  Start address.
- uint32_t endAddr
  End address.
- uint32_t key [4]
  Encryption key.
- uint32_t counter [2]
  Encryption counter.

37.2.2 struct otfad_config_t

Data Fields

- bool forceSVM
  Force entry into SVM after a write.
- bool forceLDM
  Force entry into LDM after a write.
- bool restrictedRegAccess
  Restricted register access enable.
- bool enableOTFAD
  OTFAD has decryption enabled.

37.3 Macro Definition Documentation

37.3.1 #define FSL_OTFAD_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 3U))

37.4 Enumeration Type Documentation

37.4.1 anonymous enum

Enumerator

kStatus_OTFAD_ResRegAccessMode Restricted register mode.
kStatus_OTFAD_AddressError End address less than start address.
kStatus_OTFAD_RegionOverlap the OTFAD does not support any form of memory region overlap, for system accesses that hit in multiple contexts or no contexts, the fetched data is simply bypassed
kStatus_OTFAD_RegionMiss For accesses that hit in a single context, but not the selected one.

37.4.2 anonymous enum

Enumerator

kOTFAD_Context_0 context 0
kOTFAD_Context_1 context 1
kOTFAD_Context_2 context 2
kOTFAD_Context_3 context 3

37.4.3 anonymous enum

Enumerator

kOTFAD_NRM Normal Mode.
kOTFAD_SVM Security Violation Mode.
kOTFAD_LDM Logically Disabled Mode.

37.5 Function Documentation

37.5.1 void OTFAD_GetDefaultConfig (otfad_config_t * *config*)

Parameters

config	OTFAD configuration.

37.5.2 void OTFAD_Init (OTFAD_Type * *base,* const otfad_config_t * *config*)

Parameters

base	OTFAD base address.
config	OTFAD configuration.

37.5.3 static uint32_t OTFAD_GetOperateMode (OTFAD_Type * *base*) [inline], [static]

Parameters

base	OTFAD base address.

37.5.4 static uint32_t OTFAD_GetStatus (OTFAD_Type * *base*) [inline], [static]

Parameters

base	OTFAD base address.

37.5.5 status_t OTFAD_SetEncryptionConfig (OTFAD_Type * *base,* const otfad_encryption_config_t * *config*)

Parameters

base	OTFAD base address.
config	encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

37.5.6 status_t OTFAD_GetEncryptionConfig (OTFAD_Type * *base,* otfad_encryption_config_t * *config*)

Parameters

base	OTFAD base address.
config	encryption configuration.

Note: if enable keyblob process, the first 256 bytes external memory is use for keyblob data, so this region shouldn't be in OTFAD region.

37.5.7 status_t OTFAD_HitDetermination (OTFAD_Type * *base,* uint32_t *address,* uint8_t * *contextIndex*)

Parameters

base	OTFAD base address.
address	the physical address space assigned to the QuadSPI(FlexSPI) module.
contextIndex	hitted context region index.

Returns

status, such as kStatus_Success or kStatus_OTFAD_ResRegAccessMode.

Chapter 38
PINT: Pin Interrupt and Pattern Match Driver

38.1 Overview

The MCUXpresso SDK provides a driver for the Pin Interrupt and Pattern match (PINT).

It can configure one or more pins to generate a pin interrupt when the pin or pattern match conditions are met. The pins do not have to be configured as gpio pins however they must be connected to PINT via INPUTMUX. Only the pin interrupt or pattern match function can be active for interrupt generation. If the pin interrupt function is enabled then the pattern match function can be used for wakeup via RXEV.

38.2 Pin Interrupt and Pattern match Driver operation

PINT_PinInterruptConfig() function configures the pins for pin interrupt.

PINT_PatternMatchConfig() function configures the pins for pattern match.

38.2.1 Pin Interrupt use case

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

38.2.2 Pattern match use case

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

Files

- file fsl_pint.h

Typedefs

- typedef void(∗ pint_cb_t)(pint_pin_int_t pintr, uint32_t pmatch_status)
  PINT Callback function.

Enumerations

- enum pint_pin_enable_t {
  kPINT_PinIntEnableNone = 0U,
  kPINT_PinIntEnableRiseEdge = PINT_PIN_RISE_EDGE,
  kPINT_PinIntEnableFallEdge = PINT_PIN_FALL_EDGE,
  kPINT_PinIntEnableBothEdges = PINT_PIN_BOTH_EDGE,
  kPINT_PinIntEnableLowLevel = PINT_PIN_LOW_LEVEL,
  kPINT_PinIntEnableHighLevel = PINT_PIN_HIGH_LEVEL }

PINT Pin Interrupt enable type.

- enum pint_pin_int_t {
  kPINT_PinInt0 = 0U,
  kPINT_PinInt1 = 1U,
  kPINT_PinInt2 = 2U,
  kPINT_PinInt3 = 3U,
  kPINT_PinInt4 = 4U,
  kPINT_PinInt5 = 5U,
  kPINT_PinInt6 = 6U,
  kPINT_PinInt7 = 7U }

  PINT Pin Interrupt type.
- enum pint_pmatch_input_src_t {
  kPINT_PatternMatchInp0Src = 0U,
  kPINT_PatternMatchInp1Src = 1U,
  kPINT_PatternMatchInp2Src = 2U,
  kPINT_PatternMatchInp3Src = 3U,
  kPINT_PatternMatchInp4Src = 4U,
  kPINT_PatternMatchInp5Src = 5U,
  kPINT_PatternMatchInp6Src = 6U,
  kPINT_PatternMatchInp7Src = 7U,
  kPINT_SecPatternMatchInp0Src = 0U,
  kPINT_SecPatternMatchInp1Src = 1U }

  PINT Pattern Match bit slice input source type.
- enum pint_pmatch_bslice_t {
  kPINT_PatternMatchBSlice0 = 0U,
  kPINT_PatternMatchBSlice1 = 1U,
  kPINT_PatternMatchBSlice2 = 2U,
  kPINT_PatternMatchBSlice3 = 3U,
  kPINT_PatternMatchBSlice4 = 4U,
  kPINT_PatternMatchBSlice5 = 5U,
  kPINT_PatternMatchBSlice6 = 6U,
  kPINT_PatternMatchBSlice7 = 7U }

  PINT Pattern Match bit slice type.
- enum pint_pmatch_bslice_cfg_t {
  kPINT_PatternMatchAlways = 0U,
  kPINT_PatternMatchStickyRise = 1U,
  kPINT_PatternMatchStickyFall = 2U,
  kPINT_PatternMatchStickyBothEdges = 3U,
  kPINT_PatternMatchHigh = 4U,
  kPINT_PatternMatchLow = 5U,
  kPINT_PatternMatchNever = 6U,
  kPINT_PatternMatchBothEdges = 7U }

  PINT Pattern Match configuration type.

Functions

- void PINT_Init (PINT_Type *base)
  Initialize PINT peripheral.
- void PINT_PinInterruptConfig (PINT_Type *base, pint_pin_int_t intr, pint_pin_enable_t enable, pint_cb_t callback)
  Configure PINT peripheral pin interrupt.
- void PINT_PinInterruptGetConfig (PINT_Type *base, pint_pin_int_t pintr, pint_pin_enable_t *enable, pint_cb_t *callback)
  Get PINT peripheral pin interrupt configuration.
- void PINT_PinInterruptClrStatus (PINT_Type *base, pint_pin_int_t pintr)
  Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.
- static uint32_t PINT_PinInterruptGetStatus (PINT_Type *base, pint_pin_int_t pintr)
  Get Selected pin interrupt status.
- void PINT_PinInterruptClrStatusAll (PINT_Type *base)
  Clear all pin interrupts status only when pins were triggered by edge-sensitive.
- static uint32_t PINT_PinInterruptGetStatusAll (PINT_Type *base)
  Get all pin interrupts status.
- static void PINT_PinInterruptClrFallFlag (PINT_Type *base, pint_pin_int_t pintr)
  Clear Selected pin interrupt fall flag.
- static uint32_t PINT_PinInterruptGetFallFlag (PINT_Type *base, pint_pin_int_t pintr)
  Get selected pin interrupt fall flag.
- static void PINT_PinInterruptClrFallFlagAll (PINT_Type *base)
  Clear all pin interrupt fall flags.
- static uint32_t PINT_PinInterruptGetFallFlagAll (PINT_Type *base)
  Get all pin interrupt fall flags.
- static void PINT_PinInterruptClrRiseFlag (PINT_Type *base, pint_pin_int_t pintr)
  Clear Selected pin interrupt rise flag.
- static uint32_t PINT_PinInterruptGetRiseFlag (PINT_Type *base, pint_pin_int_t pintr)
  Get selected pin interrupt rise flag.
- static void PINT_PinInterruptClrRiseFlagAll (PINT_Type *base)
  Clear all pin interrupt rise flags.
- static uint32_t PINT_PinInterruptGetRiseFlagAll (PINT_Type *base)
  Get all pin interrupt rise flags.
- void PINT_PatternMatchConfig (PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
  Configure PINT pattern match.
- void PINT_PatternMatchGetConfig (PINT_Type *base, pint_pmatch_bslice_t bslice, pint_pmatch_cfg_t *cfg)
  Get PINT pattern match configuration.
- static uint32_t PINT_PatternMatchGetStatus (PINT_Type *base, pint_pmatch_bslice_t bslice)
  Get pattern match bit slice status.
- static uint32_t PINT_PatternMatchGetStatusAll (PINT_Type *base)
  Get status of all pattern match bit slices.
- uint32_t PINT_PatternMatchResetDetectLogic (PINT_Type *base)
  Reset pattern match detection logic.
- static void PINT_PatternMatchEnable (PINT_Type *base)
  Enable pattern match function.
- static void PINT_PatternMatchDisable (PINT_Type *base)
  Disable pattern match function.
- static void PINT_PatternMatchEnableRXEV (PINT_Type *base)

Enable RXEV output.
- static void PINT_PatternMatchDisableRXEV (PINT_Type *base)
  Disable RXEV output.
- void PINT_EnableCallback (PINT_Type *base)
  Enable callback.
- void PINT_DisableCallback (PINT_Type *base)
  Disable callback.
- void PINT_Deinit (PINT_Type *base)
  Deinitialize PINT peripheral.
- void PINT_EnableCallbackByIndex (PINT_Type *base, pint_pin_int_t pintIdx)
  enable callback by pin index.
- void PINT_DisableCallbackByIndex (PINT_Type *base, pint_pin_int_t pintIdx)
  disable callback by pin index.

Driver version

- #define FSL_PINT_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))
  Version 2.1.9.

38.3 Typedef Documentation

38.3.1 typedef void(* pint_cb_t)(pint_pin_int_t pintr, uint32_t pmatch_status)

38.4 Enumeration Type Documentation

38.4.1 enum pint_pin_enable_t

Enumerator

kPINT_PinIntEnableNone Do not generate Pin Interrupt.
kPINT_PinIntEnableRiseEdge Generate Pin Interrupt on rising edge.
kPINT_PinIntEnableFallEdge Generate Pin Interrupt on falling edge.
kPINT_PinIntEnableBothEdges Generate Pin Interrupt on both edges.
kPINT_PinIntEnableLowLevel Generate Pin Interrupt on low level.
kPINT_PinIntEnableHighLevel Generate Pin Interrupt on high level.

38.4.2 enum pint_pin_int_t

Enumerator

kPINT_PinInt0 Pin Interrupt 0.
kPINT_PinInt1 Pin Interrupt 1.
kPINT_PinInt2 Pin Interrupt 2.
kPINT_PinInt3 Pin Interrupt 3.
kPINT_PinInt4 Pin Interrupt 4.
kPINT_PinInt5 Pin Interrupt 5.
kPINT_PinInt6 Pin Interrupt 6.

kPINT_PinInt7 Pin Interrupt 7.

38.4.3 enum pint_pmatch_input_src_t

Enumerator

kPINT_PatternMatchInp0Src Input source 0.
kPINT_PatternMatchInp1Src Input source 1.
kPINT_PatternMatchInp2Src Input source 2.
kPINT_PatternMatchInp3Src Input source 3.
kPINT_PatternMatchInp4Src Input source 4.
kPINT_PatternMatchInp5Src Input source 5.
kPINT_PatternMatchInp6Src Input source 6.
kPINT_PatternMatchInp7Src Input source 7.
kPINT_SecPatternMatchInp0Src Input source 0.
kPINT_SecPatternMatchInp1Src Input source 1.

38.4.4 enum pint_pmatch_bslice_t

Enumerator

kPINT_PatternMatchBSlice0 Bit slice 0.
kPINT_PatternMatchBSlice1 Bit slice 1.
kPINT_PatternMatchBSlice2 Bit slice 2.
kPINT_PatternMatchBSlice3 Bit slice 3.
kPINT_PatternMatchBSlice4 Bit slice 4.
kPINT_PatternMatchBSlice5 Bit slice 5.
kPINT_PatternMatchBSlice6 Bit slice 6.
kPINT_PatternMatchBSlice7 Bit slice 7.

38.4.5 enum pint_pmatch_bslice_cfg_t

Enumerator

kPINT_PatternMatchAlways Always Contributes to product term match.
kPINT_PatternMatchStickyRise Sticky Rising edge.
kPINT_PatternMatchStickyFall Sticky Falling edge.
kPINT_PatternMatchStickyBothEdges Sticky Rising or Falling edge.
kPINT_PatternMatchHigh High level.
kPINT_PatternMatchLow Low level.
kPINT_PatternMatchNever Never contributes to product term match.
kPINT_PatternMatchBothEdges Either rising or falling edge.

38.5 Function Documentation

38.5.1 void PINT_Init (PINT_Type * *base*)

This function initializes the PINT peripheral and enables the clock.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.2 void PINT_PinInterruptConfig (PINT_Type * *base*, pint_pin_int_t *intr*, pint_pin_enable_t *enable*, pint_cb_t *callback*)

This function configures a given pin interrupt.

Parameters

base	Base address of the PINT peripheral.
intr	Pin interrupt.
enable	Selects detection logic.
callback	Callback.

Return values

None.	

38.5.3 void PINT_PinInterruptGetConfig (PINT_Type * *base*, pint_pin_int_t *pintr*, pint_pin_enable_t * *enable*, pint_cb_t * *callback*)

This function returns the configuration of a given pin interrupt.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.
enable	Pointer to store the detection logic.
callback	Callback.

Return values

None.	

38.5.4 void PINT_PinInterruptClrStatus (PINT_Type ∗ *base,* pint_pin_int_t *pintr*)

This function clears the selected pin interrupt status.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

38.5.5 static uint32_t PINT_PinInterruptGetStatus (PINT_Type ∗ *base,* pint_pin_int_t *pintr*) `[inline], [static]`

This function returns the selected pin interrupt status.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

status	= 0 No pin interrupt request. = 1 Selected Pin interrupt request active.

38.5.6 void PINT_PinInterruptClrStatusAll (PINT_Type ∗ *base*)

This function clears the status of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.7 static uint32_t PINT_PinInterruptGetStatusAll (PINT_Type * *base*) [inline], [static]

This function returns the status of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

status	Each bit position indicates the status of corresponding pin interrupt. = 0 No pin interrupt request. = 1 Pin interrupt request active.

38.5.8 static void PINT_PinInterruptClrFallFlag (PINT_Type * *base,* pint_pin_int_t *pintr*) [inline], [static]

This function clears the selected pin interrupt fall flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

38.5.9 static uint32_t PINT_PinInterruptGetFallFlag (PINT_Type * *base,* pint_pin_int_t *pintr*) [inline], [static]

This function returns the selected pin interrupt fall flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

flag	= 0 Falling edge has not been detected. = 1 Falling edge has been detected.

38.5.10 static void PINT_PinInterruptClrFallFlagAll (PINT_Type * *base*) [inline], [static]

This function clears the fall flag for all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.11 static uint32_t PINT_PinInterruptGetFallFlagAll (PINT_Type * *base*) [inline], [static]

This function returns the fall flag of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

flags	Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected.

38.5.12 static void PINT_PinInterruptClrRiseFlag (PINT_Type * *base,* pint_pin_int_t *pintr*) [inline], [static]

This function clears the selected pin interrupt rise flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

None.	

38.5.13 static uint32_t PINT_PinInterruptGetRiseFlag (PINT_Type ∗ *base,* pint_pin_int_t *pintr*) [inline], [static]

This function returns the selected pin interrupt rise flag.

Parameters

base	Base address of the PINT peripheral.
pintr	Pin interrupt.

Return values

flag	= 0 Rising edge has not been detected. = 1 Rising edge has been detected.

38.5.14 static void PINT_PinInterruptClrRiseFlagAll (PINT_Type ∗ *base*) [inline], [static]

This function clears the rise flag for all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.15 static uint32_t PINT_PinInterruptGetRiseFlagAll (PINT_Type ∗ *base*) [inline], [static]

This function returns the rise flag of all pin interrupts.

Parameters

base	Base address of the PINT peripheral.

Return values

flags	Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected.

38.5.16 void PINT_PatternMatchConfig (PINT_Type * *base*, pint_pmatch_bslice_t *bslice*, pint_pmatch_cfg_t * *cfg*)

This function configures a given pattern match bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.
cfg	Pointer to bit slice configuration.

Return values

None.	

38.5.17 void PINT_PatternMatchGetConfig (PINT_Type * *base*, pint_pmatch_bslice_t *bslice*, pint_pmatch_cfg_t * *cfg*)

This function returns the configuration of a given pattern match bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.
cfg	Pointer to bit slice configuration.

Return values

None.	

38.5.18 static uint32_t PINT_PatternMatchGetStatus (PINT_Type * *base*, pint_pmatch_bslice_t *bslice*) [inline], [static]

This function returns the status of selected bit slice.

Parameters

base	Base address of the PINT peripheral.
bslice	Pattern match bit slice number.

Return values

status	= 0 Match has not been detected. = 1 Match has been detected.

38.5.19 static uint32_t PINT_PatternMatchGetStatusAll (PINT_Type * *base*) [inline], [static]

This function returns the status of all bit slices.

Parameters

base	Base address of the PINT peripheral.

Return values

status	Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected.

38.5.20 uint32_t PINT_PatternMatchResetDetectLogic (PINT_Type * *base*)

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

base	Base address of the PINT peripheral.

Return values

pmstatus	Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected.

38.5.21 static void PINT_PatternMatchEnable (PINT_Type * *base*) [inline], [static]

This function enables the pattern match function.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.22 static void PINT_PatternMatchDisable (PINT_Type * *base*) [inline], [static]

This function disables the pattern match function.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.23 static void PINT_PatternMatchEnableRXEV (PINT_Type * *base*) [inline], [static]

This function enables the pattern match RXEV output.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.24 static void PINT_PatternMatchDisableRXEV (PINT_Type * *base*) [inline], [static]

This function disables the pattern match RXEV output.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.25 void PINT_EnableCallback (PINT_Type * *base*)

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.26 void PINT_DisableCallback (PINT_Type * *base*)

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

base	Base address of the peripheral.

Return values

None.	

38.5.27 void PINT_Deinit (PINT_Type * *base*)

This function disables the PINT clock.

Parameters

base	Base address of the PINT peripheral.

Return values

None.	

38.5.28 void PINT_EnableCallbackByIndex (PINT_Type * *base,* pint_pin_int_t *pintIdx*)

This function enables callback by pin index instead of enabling all pins.

Parameters

base	Base address of the peripheral.
pintIdx	pin index.

Return values

None.	

38.5.29 void PINT_DisableCallbackByIndex (PINT_Type * *base,* pint_pin_int_t *pintIdx*)

This function disables callback by pin index instead of disabling all pins.

Parameters

base	Base address of the peripheral.
pintIdx	pin index.

Return values

None.	

Chapter 39
POWERQUAD: PowerQuad hardware accelerator

39.1 Overview

The MCUXpresso SDK provides driver for the PowerQuad module of MCUXpresso SDK devices.

The PowerQuad hardware accelerator for (fixed/floating point/matrix operation) DSP functions is that idea is to replace some of the CMSIS DSP functionality with the hardware features provided by this IP.

PowerQuad driver provides the following CMSIS DSP compatible functions:

- Matrix functions

```
arm_mat_add_q15
arm_mat_add_q31
arm_mat_add_f32
arm_mat_sub_q15
arm_mat_sub_q31
arm_mat_sub_f32
arm_mat_mult_q15
arm_mat_mult_q31
arm_mat_mult_f32
arm_mat_inverse_f32
arm_mat_trans_q15
arm_mat_trans_q31
arm_mat_trans_f32
arm_mat_scale_q15
arm_mat_scale_q31
arm_mat_scale_f32
```

- Math functions

```
arm_sqrt_q15
arm_sqrt_q31
arm_sin_q15
arm_sin_q31
arm_sin_f32
arm_cos_q15
arm_cos_q31
arm_cos_f32
```

- Filter functions

```
arm_fir_q15
arm_fir_q31
arm_fir_f32
arm_conv_q15
arm_conv_q31
arm_conv_f32
arm_correlate_q15
arm_correlate_q31
arm_correlate_f32
```

- Transform functions

```
arm_rfft_q15
arm_rfft_q31
arm_cfft_q15
```

```
arm_cfft_q31
arm_ifft_q15
arm_ifft_q31
arm_dct4_q15
arm_dct4_q31
```

Note

CMSIS DSP compatible functions limitations

1. PowerQuad FFT engine only looks at the bottom 27 bits of the input word, down scale the input data to avoid saturation.
2. When use arm_fir_q15/arm_fir_q31/arm_fir_f32 for incremental, the new data should follow the old data. For example the array for input data is inputData[], and the array for output data is outputData[]. The first 32 input data is saved in inputData[0:31], after callling `arm_fir_xxx(&fir, inputData, outputData, 32)`, the output data is saved in outputData[0:31]. The new input data must be saved from inputData[32], then call `arm_fir_xxx(&fir, &inputData[32], &outputData[32], 32)` for incremental calculation.

The PowerQuad consists of several internal computation engines: Transform engine, Transcendental function engine, Trigonometry function engine, Dual biquad IIR filter engine, Matrix accelerator engine, FIR filter engine, CORDIC engine.

For low level APIs, all function APIs, except using coprocessor instruction and arctan/arctanh API, need to calling wait done API to wait for calculation complete.

39.2 Function groups

39.2.1 POWERQUAD functional Operation

This group implements the POWERQUAD functional API.

Data Structures

- struct pq_prescale_t
  Coprocessor prescale. More...
- struct pq_config_t
  powerquad data structure format More...
- struct pq_biquad_param_t
  Struct to save biquad parameters. More...
- struct pq_biquad_state_t
  Struct to save biquad state. More...
- struct pq_biquad_cascade_df2_instance
  Instance structure for the direct form II Biquad cascade filter. More...
- union pq_float_t
  Conversion between integer and float type. More...

Macros

- #define PQ_LN_INF PQ_LN, 1, PQ_TRANS
  Parameter used for vector ln(x)
- #define PQ_INV_INF PQ_INV, 0, PQ_TRANS
  Parameter used for vector 1/x.
- #define PQ_SQRT_INF PQ_SQRT, 0, PQ_TRANS
  Parameter used for vector sqrt(x)
- #define PQ_ISQRT_INF PQ_INVSQRT, 0, PQ_TRANS
  Parameter used for vector 1/sqrt(x)
- #define PQ_ETOX_INF PQ_ETOX, 0, PQ_TRANS
  Parameter used for vector e^x.
- #define PQ_ETONX_INF PQ_ETONX, 0, PQ_TRANS
  Parameter used for vector e^(-x)
- #define PQ_SIN_INF PQ_SIN, 1, PQ_TRIG
  Parameter used for vector sin(x)
- #define PQ_COS_INF PQ_COS, 1, PQ_TRIG
  Parameter used for vector cos(x)
- #define PQ_Initiate_Vector_Func(pSrc, pDst)
  Start 32-bit data vector calculation.
- #define PQ_End_Vector_Func() __asm volatile("POP {r2-r7}")
  End vector calculation.
- #define PQ_StartVector(PSRC, PDST, LENGTH)
  Start 32-bit data vector calculation.
- #define PQ_StartVectorFixed16(PSRC, PDST, LENGTH)
  Start 16-bit data vector calculation.
- #define PQ_StartVectorQ15(PSRC, PDST, LENGTH)
  Start Q15-bit data vector calculation.
- #define PQ_EndVector() __asm volatile("POP {r3-r10} \n")
  End vector calculation.
- #define PQ_Vector8F32(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)
  Float data vector calculation.
- #define PQ_Vector8Fixed32(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)
  Fixed 32bits data vector calculation.
- #define PQ_Vector8Fixed16(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)
  Fixed 32bits data vector calculation.
- #define PQ_Vector8Q15(BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE)
  Q15 data vector calculation.
- #define PQ_DF2_Vector8_FP(middle, last)
  Float data vector biquad direct form II calculation.
- #define PQ_DF2_Vector8_FX(middle, last)
  Fixed data vector biquad direct form II calculation.
- #define PQ_Vector8BiquadDf2F32()
  Float data vector biquad direct form II calculation.
- #define PQ_Vector8BiquadDf2Fixed32()
  Fixed 32-bit data vector biquad direct form II calculation.
- #define PQ_Vector8BiquadDf2Fixed16()
  Fixed 16-bit data vector biquad direct form II calculation.
- #define PQ_DF2_Cascade_Vector8_FP(middle, last)
  Float data vector direct form II biquad cascade filter.

- #define PQ_DF2_Cascade_Vector8_FX(middle, last)
  Fixed data vector direct form II biquad cascade filter.
- #define PQ_Vector8BiqaudDf2CascadeF32()
  Float data vector direct form II biquad cascade filter.
- #define PQ_Vector8BiqaudDf2CascadeFixed32()
  Fixed 32-bit data vector direct form II biquad cascade filter.
- #define PQ_Vector8BiqaudDf2CascadeFixed16()
  Fixed 16-bit data vector direct form II biquad cascade filter.
- #define POWERQUAD_MAKE_MATRIX_LEN(mat1Row, mat1Col, mat2Col) (((uint32_t)(mat1Row) << 0U) | ((uint32_t)(mat1Col) << 8U) | ((uint32_t)(mat2Col) << 16U))
  Make the length used for matrix functions.
- #define PQ_Q31_2_FLOAT(x) (((float)(x)) / 2147483648.0f)
  Convert Q31 to float.
- #define PQ_Q15_2_FLOAT(x) (((float)(x)) / 32768.0f)
  Convert Q15 to float.

Enumerations

- enum pq_computationengine_t {
  kPQ_CP_PQ = 0,
  kPQ_CP_MTX = 1,
  kPQ_CP_FFT = 2,
  kPQ_CP_FIR = 3,
  kPQ_CP_CORDIC = 5 }
  powerquad computation engine
- enum pq_format_t {
  kPQ_16Bit = 0,
  kPQ_32Bit = 1,
  kPQ_Float = 2 }
  powerquad data structure format type
- enum pq_cordic_iter_t {
  kPQ_Iteration_8 = 0,
  kPQ_Iteration_16,
  kPQ_Iteration_24 }
  CORDIC iteration.

Driver version

- #define FSL_POWERQUAD_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  Version.

POWERQUAD functional Operation

- void PQ_GetDefaultConfig (pq_config_t *config)
  Get default configuration.
- void PQ_SetConfig (POWERQUAD_Type *base, const pq_config_t *config)
  Set configuration with format/prescale.
- static void PQ_SetCoprocessorScaler (POWERQUAD_Type *base, const pq_prescale_t *prescale)

set coprocessor scaler for coprocessor instructions, this function is used to set output saturation and scaleing for input/output.

- void PQ_Init (POWERQUAD_Type *base)
  Initializes the POWERQUAD module.
- void PQ_Deinit (POWERQUAD_Type *base)
  De-initializes the POWERQUAD module.
- void PQ_SetFormat (POWERQUAD_Type *base, pq_computationengine_t engine, pq_format_t format)
  Set format for non-coprecessor instructions.
- static void PQ_WaitDone (POWERQUAD_Type *base)
  Wait for the completion.
- static void PQ_LnF32 (float *pSrc, float *pDst)
  Processing function for the floating-point natural log.
- static void PQ_InvF32 (float *pSrc, float *pDst)
  Processing function for the floating-point reciprocal.
- static void PQ_SqrtF32 (float *pSrc, float *pDst)
  Processing function for the floating-point square-root.
- static void PQ_InvSqrtF32 (float *pSrc, float *pDst)
  Processing function for the floating-point inverse square-root.
- static void PQ_EtoxF32 (float *pSrc, float *pDst)
  Processing function for the floating-point natural exponent.
- static void PQ_EtonxF32 (float *pSrc, float *pDst)
  Processing function for the floating-point natural exponent with negative parameter.
- static void PQ_SinF32 (float *pSrc, float *pDst)
  Processing function for the floating-point sine.
- static void PQ_CosF32 (float *pSrc, float *pDst)
  Processing function for the floating-point cosine.
- static void PQ_BiquadF32 (float *pSrc, float *pDst)
  Processing function for the floating-point biquad.
- static void PQ_DivF32 (float *x1, float *x2, float *pDst)
  Processing function for the floating-point division.
- static void PQ_Biquad1F32 (float *pSrc, float *pDst)
  Processing function for the floating-point biquad.
- static int32_t PQ_LnFixed (int32_t val)
  Processing function for the fixed natural log.
- static int32_t PQ_InvFixed (int32_t val)
  Processing function for the fixed reciprocal.
- static uint32_t PQ_SqrtFixed (uint32_t val)
  Processing function for the fixed square-root.
- static int32_t PQ_InvSqrtFixed (int32_t val)
  Processing function for the fixed inverse square-root.
- static int32_t PQ_EtoxFixed (int32_t val)
  Processing function for the Fixed natural exponent.
- static int32_t PQ_EtonxFixed (int32_t val)
  Processing function for the fixed natural exponent with negative parameter.
- static int32_t PQ_SinQ31 (int32_t val)
  Processing function for the fixed sine.
- static int16_t PQ_SinQ15 (int16_t val)
  Processing function for the fixed sine.
- static int32_t PQ_CosQ31 (int32_t val)
  Processing function for the fixed cosine.

- static int16_t PQ_CosQ15 (int16_t val)
  Processing function for the fixed sine.
- static int32_t PQ_BiquadFixed (int32_t val)
  Processing function for the fixed biquad.
- void PQ_VectorLnF32 (float *pSrc, float *pDst, int32_t length)
  Processing function for the floating-point vectorised natural log.
- void PQ_VectorInvF32 (float *pSrc, float *pDst, int32_t length)
  Processing function for the floating-point vectorised reciprocal.
- void PQ_VectorSqrtF32 (float *pSrc, float *pDst, int32_t length)
  Processing function for the floating-point vectorised square-root.
- void PQ_VectorInvSqrtF32 (float *pSrc, float *pDst, int32_t length)
  Processing function for the floating-point vectorised inverse square-root.
- void PQ_VectorEtoxF32 (float *pSrc, float *pDst, int32_t length)
  Processing function for the floating-point vectorised natural exponent.
- void PQ_VectorEtonxF32 (float *pSrc, float *pDst, int32_t length)
  Processing function for the floating-point vectorised natural exponent with negative parameter.
- void PQ_VectorSinF32 (float *pSrc, float *pDst, int32_t length)
  Processing function for the floating-point vectorised sine.
- void PQ_VectorCosF32 (float *pSrc, float *pDst, int32_t length)
  Processing function for the floating-point vectorised cosine.
- void PQ_VectorLnFixed32 (int32_t *pSrc, int32_t *pDst, int32_t length)
  Processing function for the Q31 vectorised natural log.
- void PQ_VectorInvFixed32 (int32_t *pSrc, int32_t *pDst, int32_t length)
  Processing function for the Q31 vectorised reciprocal.
- void PQ_VectorSqrtFixed32 (int32_t *pSrc, int32_t *pDst, int32_t length)
  Processing function for the 32-bit integer vectorised square-root.
- void PQ_VectorInvSqrtFixed32 (int32_t *pSrc, int32_t *pDst, int32_t length)
  Processing function for the 32-bit integer vectorised inverse square-root.
- void PQ_VectorEtoxFixed32 (int32_t *pSrc, int32_t *pDst, int32_t length)
  Processing function for the 32-bit integer vectorised natural exponent.
- void PQ_VectorEtonxFixed32 (int32_t *pSrc, int32_t *pDst, int32_t length)
  Processing function for the 32-bit integer vectorised natural exponent with negative parameter.
- void PQ_VectorSinQ15 (int16_t *pSrc, int16_t *pDst, int32_t length)
  Processing function for the Q15 vectorised sine.
- void PQ_VectorCosQ15 (int16_t *pSrc, int16_t *pDst, int32_t length)
  Processing function for the Q15 vectorised cosine.
- void PQ_VectorSinQ31 (int32_t *pSrc, int32_t *pDst, int32_t length)
  Processing function for the Q31 vectorised sine.
- void PQ_VectorCosQ31 (int32_t *pSrc, int32_t *pDst, int32_t length)
  Processing function for the Q31 vectorised cosine.
- void PQ_VectorLnFixed16 (int16_t *pSrc, int16_t *pDst, int32_t length)
  Processing function for the 16-bit integer vectorised natural log.
- void PQ_VectorInvFixed16 (int16_t *pSrc, int16_t *pDst, int32_t length)
  Processing function for the 16-bit integer vectorised reciprocal.
- void PQ_VectorSqrtFixed16 (int16_t *pSrc, int16_t *pDst, int32_t length)
  Processing function for the 16-bit integer vectorised square-root.
- void PQ_VectorInvSqrtFixed16 (int16_t *pSrc, int16_t *pDst, int32_t length)
  Processing function for the 16-bit integer vectorised inverse square-root.
- void PQ_VectorEtoxFixed16 (int16_t *pSrc, int16_t *pDst, int32_t length)
  Processing function for the 16-bit integer vectorised natural exponent.
- void PQ_VectorEtonxFixed16 (int16_t *pSrc, int16_t *pDst, int32_t length)

Processing function for the 16-bit integer vectorised natural exponent with negative parameter.

- void PQ_VectorBiqaudDf2F32 (float *pSrc, float *pDst, int32_t length)

  Processing function for the floating-point vectorised biquad direct form II.
- void PQ_VectorBiqaudDf2Fixed32 (int32_t *pSrc, int32_t *pDst, int32_t length)

  Processing function for the 32-bit integer vectorised biquad direct form II.
- void PQ_VectorBiqaudDf2Fixed16 (int16_t *pSrc, int16_t *pDst, int32_t length)

  Processing function for the 16-bit integer vectorised biquad direct form II.
- void PQ_VectorBiqaudCascadeDf2F32 (float *pSrc, float *pDst, int32_t length)

  Processing function for the floating-point vectorised biquad direct form II.
- void PQ_VectorBiqaudCascadeDf2Fixed32 (int32_t *pSrc, int32_t *pDst, int32_t length)

  Processing function for the 32-bit integer vectorised biquad direct form II.
- void PQ_VectorBiqaudCascadeDf2Fixed16 (int16_t *pSrc, int16_t *pDst, int32_t length)

  Processing function for the 16-bit integer vectorised biquad direct form II.
- int32_t PQ_ArctanFixed (POWERQUAD_Type *base, int32_t x, int32_t y, pq_cordic_iter_t iteration)

  Processing function for the fixed inverse trigonometric.
- int32_t PQ_ArctanhFixed (POWERQUAD_Type *base, int32_t x, int32_t y, pq_cordic_iter_-t iteration)

  Processing function for the fixed inverse trigonometric.
- static int32_t PQ_Biquad1Fixed (int32_t val)

  Processing function for the fixed biquad.
- void PQ_TransformCFFT (POWERQUAD_Type *base, uint32_t length, void *pData, void *p-Result)

  Processing function for the complex FFT.
- void PQ_TransformRFFT (POWERQUAD_Type *base, uint32_t length, void *pData, void *p-Result)

  Processing function for the real FFT.
- void PQ_TransformIFFT (POWERQUAD_Type *base, uint32_t length, void *pData, void *p-Result)

  Processing function for the inverse complex FFT.
- void PQ_TransformCDCT (POWERQUAD_Type *base, uint32_t length, void *pData, void *p-Result)

  Processing function for the complex DCT.
- void PQ_TransformRDCT (POWERQUAD_Type *base, uint32_t length, void *pData, void *p-Result)

  Processing function for the real DCT.
- void PQ_TransformIDCT (POWERQUAD_Type *base, uint32_t length, void *pData, void *p-Result)

  Processing function for the inverse complex DCT.
- void PQ_BiquadBackUpInternalState (POWERQUAD_Type *base, int32_t biquad_num, pq_-biquad_state_t *state)

  Processing function for backup biquad context.
- void PQ_BiquadRestoreInternalState (POWERQUAD_Type *base, int32_t biquad_num, pq_-biquad_state_t *state)

  Processing function for restore biquad context.
- void PQ_BiquadCascadeDf2Init (pq_biquad_cascade_df2_instance *S, uint8_t numStages, pq_-biquad_state_t *pState)

  Initialization function for the direct form II Biquad cascade filter.
- void PQ_BiquadCascadeDf2F32 (const pq_biquad_cascade_df2_instance *S, float *pSrc, float *p-Dst, uint32_t blockSize)

Processing function for the floating-point direct form II Biquad cascade filter.

- void PQ_BiquadCascadeDf2Fixed32 (const pq_biquad_cascade_df2_instance *S, int32_t *pSrc, int32_t *pDst, uint32_t blockSize)

  Processing function for the Q31 direct form II Biquad cascade filter.
- void PQ_BiquadCascadeDf2Fixed16 (const pq_biquad_cascade_df2_instance *S, int16_t *pSrc, int16_t *pDst, uint32_t blockSize)

  Processing function for the Q15 direct form II Biquad cascade filter.
- void PQ_FIR (POWERQUAD_Type *base, const void *pAData, int32_t ALength, const void *pBData, int32_t BLength, void *pResult, uint32_t opType)

  Processing function for the FIR.
- void PQ_FIRIncrement (POWERQUAD_Type *base, int32_t ALength, int32_t BLength, int32_t xOffset)

  Processing function for the incremental FIR.
- void PQ_MatrixAddition (POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

  Processing function for the matrix addition.
- void PQ_MatrixSubtraction (POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

  Processing function for the matrix subtraction.
- void PQ_MatrixMultiplication (POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

  Processing function for the matrix multiplication.
- void PQ_MatrixProduct (POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

  Processing function for the matrix product.
- void PQ_VectorDotProduct (POWERQUAD_Type *base, uint32_t length, void *pAData, void *pBData, void *pResult)

  Processing function for the vector dot product.
- void PQ_MatrixInversion (POWERQUAD_Type *base, uint32_t length, void *pData, void *pTmpData, void *pResult)

  Processing function for the matrix inverse.
- void PQ_MatrixTranspose (POWERQUAD_Type *base, uint32_t length, void *pData, void *pResult)

  Processing function for the matrix transpose.
- void PQ_MatrixScale (POWERQUAD_Type *base, uint32_t length, float misc, const void *pData, void *pResult)

  Processing function for the matrix scale.

39.3 Data Structure Documentation

39.3.1 struct pq_prescale_t

Data Fields

- int8_t inputPrescale

  Input prescale.
- int8_t outputPrescale

  Output prescale.
- int8_t outputSaturate

Output saturate at n bits, for example 0x11 is 8 bit space, the value will be truncated at +127 or -128.

Field Documentation

(1) int8_t pq_prescale_t::inputPrescale

(2) int8_t pq_prescale_t::outputPrescale

(3) int8_t pq_prescale_t::outputSaturate

39.3.2 struct pq_config_t

Data Fields

- pq_format_t inputAFormat
  Input A format.
- int8_t inputAPrescale
  Input A prescale, for example 1.5 can be 1.5∗2^n if you scale by 'shifting' ('scaling' by a factor of n).
- pq_format_t inputBFormat
  Input B format.
- int8_t inputBPrescale
  Input B prescale.
- pq_format_t outputFormat
  Out format.
- int8_t outputPrescale
  Out prescale.
- pq_format_t tmpFormat
  Temp format.
- int8_t tmpPrescale
  Temp prescale.
- pq_format_t machineFormat
  Machine format.
- uint32_t ∗ tmpBase
  Tmp base address.

Field Documentation

(1) pq_format_t pq_config_t::inputAFormat

(2) int8_t pq_config_t::inputAPrescale

(3) pq_format_t pq_config_t::inputBFormat

(4) int8_t pq_config_t::inputBPrescale

(5) pq_format_t pq_config_t::outputFormat

(6) int8_t pq_config_t::outputPrescale

(7) pq_format_t pq_config_t::tmpFormat

(8) int8_t pq_config_t::tmpPrescale

(9) pq_format_t pq_config_t::machineFormat

(10) uint32_t* pq_config_t::tmpBase

39.3.3 struct pq_biquad_param_t

Data Fields

- float v_n_1
  v[n-1], set to 0 when initialization.
- float v_n
  v[n], set to 0 when initialization.
- float a_1
  a[1]
- float a_2
  a[2]
- float b_0
  b[0]
- float b_1
  b[1]
- float b_2
  b[2]

Field Documentation

(1) float pq_biquad_param_t::v_n_1

(2) float pq_biquad_param_t::v_n

39.3.4 struct pq_biquad_state_t

Data Fields

- pq_biquad_param_t param
  Filter parameter.
- uint32_t compreg
  Internal register, set to 0 when initialization.

Field Documentation

(1) pq_biquad_param_t pq_biquad_state_t::param

(2) uint32_t pq_biquad_state_t::compreg

39.3.5 struct pq_biquad_cascade_df2_instance

Data Fields

- uint8_t numStages
- pq_biquad_state_t ∗ pState

Field Documentation

(1) uint8_t pq_biquad_cascade_df2_instance::numStages

Number of 2nd order stages in the filter.

(2) pq_biquad_state_t∗ pq_biquad_cascade_df2_instance::pState

Points to the array of state coefficients.

39.3.6 union pq_float_t

Data Fields

- float floatX
  Float type.
- uint32_t integerX
  Unsigned interger type.

Field Documentation

(1) float pq_float_t::floatX

(2) uint32_t pq_float_t::integerX

39.4 Macro Definition Documentation

39.4.1 #define FSL_POWERQUAD_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

39.4.2 #define PQ_Initiate_Vector_Func(*pSrc, pDst*)

Value:

```
__asm volatile(                                          \
        "MOV r0, %[psrc]          \n"                     \
        "MOV r1, %[pdst]          \n"                     \
        "PUSH {r2-r7}             \n"                     \
        "LDRD r2,r3,[r0],#8       \n" ::[psrc] "r"(pSrc), \
        [pdst] "r"(pDst)                                  \
        : "r0", "r1")
```

Start the vector calculation, the input data could be float, int32_t or Q31.

Parameters

pSrc	Pointer to the source data.
pDst	Pointer to the destination data.

39.4.3 #define PQ_End_Vector_Func( ) __asm volatile("POP {r2-r7}")

This function should be called after vector calculation.

39.4.4 #define PQ_StartVector(*PSRC, PDST, LENGTH*)

Value:

```
__asm volatile(                                         \
        "MOV r0, %[psrc]          \n"                       \
        "MOV r1, %[pdst]          \n"                       \
        "MOV r2, %[length]        \n"                       \
        "PUSH {r3-r10}            \n"                       \
        "MOV r3, #0               \n"                       \
        "MOV r10, #0              \n"                       \
        "LDRD r4,r5,[r0],#8       \n" ::[psrc] "r"(PSRC), \
        [pdst] "r"(PDST), [length] "r"(LENGTH)              \
        : "r0", "r1", "r2")
```

Start the vector calculation, the input data could be float, int32_t or Q31.

Parameters

PSRC	Pointer to the source data.
PDST	Pointer to the destination data.
LENGTH	Number of the data, must be multiple of 8.

39.4.5 #define PQ_StartVectorFixed16(*PSRC, PDST, LENGTH*)

Value:

```
__asm volatile(                                         \
        "MOV r0, %[psrc]          \n"                       \
        "MOV r1, %[pdst]          \n"                       \
        "MOV r2, %[length]        \n"                       \
        "PUSH {r3-r10}            \n"                       \
        "MOV r3, #0               \n"                       \
        "LDRSH r4,[r0],#2         \n"                       \
        "LDRSH r5,[r0],#2         \n" ::[psrc] "r"(PSRC), \
        [pdst] "r"(PDST), [length] "r"(LENGTH)              \
        : "r0", "r1", "r2")
```

Start the vector calculation, the input data could be int16_t. This function should be use with PQ_Vector8-Fixed16.

Parameters

PSRC	Pointer to the source data.
PDST	Pointer to the destination data.
LENGTH	Number of the data, must be multiple of 8.

39.4.6 #define PQ_StartVectorQ15(*PSRC, PDST, LENGTH*)

Value:

```
__asm volatile(                                          \
        "MOV r0, %[psrc]          \n"                     \
        "MOV r1, %[pdst]          \n"                     \
        "MOV r2, %[length]        \n"                     \
        "PUSH {r3-r10}            \n"                     \
        "MOV r3, #0               \n"                     \
        "LDR r5,[r0],#4           \n"                     \
        "LSL r4,r5,#16            \n"                     \
        "BFC r5,#0,#16            \n" ::[psrc] "r"(PSRC), \
        [pdst] "r"(PDST), [length] "r"(LENGTH)            \
        : "r0", "r1", "r2")
```

Start the vector calculation, the input data could be Q15. This function should be use with PQ_Vector8-Q15. This function is dedicate for SinQ15/CosQ15 vector calculation. Because PowerQuad only supports Q31 Sin/Cos fixed function, so the input Q15 data is left shift 16 bits first, after Q31 calculation, the output data is right shift 16 bits.

Parameters

PSRC	Pointer to the source data.
PDST	Pointer to the destination data.
LENGTH	Number of the data, must be multiple of 8.

39.4.7 #define PQ_EndVector( ) __asm volatile("POP {r3-r10} \n")

This function should be called after vector calculation.

39.4.8 #define PQ_Vector8F32(*BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE*)

Float data vector calculation, the input data should be float. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

39.4.9 #define PQ_Vector8Fixed32(*BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE*)

Float data vector calculation, the input data should be 32-bit integer. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. PQ_SIN_INF, PQ_COS_INF. When this function is used for sin/cos calculation, the input data should be in the format Q1.31. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int32_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

39.4.10 #define PQ_Vector8Fixed16(*BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE*)

Float data vector calculation, the input data should be 16-bit integer. The parameter could be PQ_LN_INF, PQ_INV_INF, PQ_SQRT_INF, PQ_ISQRT_INF, PQ_ETOX_INF, PQ_ETONX_INF. For example, to calculate sqrt of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 4, 9, 16, 25, 36, 49, 64};
int16_t output[VECTOR_LEN];

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8F32(PQ_SQRT_INF);
PQ_EndVector();
```

39.4.11 #define PQ_Vector8Q15(*BATCH_OPCODE, DOUBLE_READ_ADDERS, BATCH_MACHINE*)

Q15 data vector calculation, this function should only be used for sin/cos Q15 calculation, and the coprocessor output prescaler must be set to 31 before this function. This function loads Q15 data and left shift 16 bits, calculate and right shift 16 bits, then stores to the output array. The input range -1 to 1 means -pi to pi. For example, to calculate sin of a vector, use like this:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {...}
int16_t output[VECTOR_LEN];
const pq_prescale_t prescale =
{
    .inputPrescale = 0,
    .outputPrescale = 31,
    .outputSaturate = 0
};

PQ_SetCoprocessorScaler(POWERQUAD, const pq_prescale_t *prescale);

PQ_StartVectorQ15(pSrc, pDst, length);
PQ_Vector8Q15(PQ_SQRT_INF);
PQ_EndVector();
```

39.4.12 #define PQ_DF2_Vector8_FP(*middle, last*)

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FP(false,false);
PQ_DF2_Vector8_FP(true,true);
PQ_End_Vector_Func();
```

39.4.13 #define PQ_DF2_Vector8_FX(*middle, last*)

Biquad filter, the input and output data are fixed data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
```

```
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_Initiate_Vector_Func(pSrc,pDst);
PQ_DF2_Vector8_FX(false,false);
PQ_DF2_Vector8_FX(true,true);
PQ_End_Vector_Func();
```

39.4.14 #define PQ_Vector8BiquadDf2F32( )

Biquad filter, the input and output data are float data. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
float output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2F32();
PQ_EndVector();
```

39.4.15 #define PQ_Vector8BiquadDf2Fixed32( )

Biquad filter, the input and output data are Q31 or 32-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};
```

```
PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed32();
PQ_EndVector();
```

39.4.16 #define PQ_Vector8BiquadDf2Fixed16( )

Biquad filter, the input and output data are Q15 or 16-bit integer. Biquad side 0 is used. Example:

```
#define VECTOR_LEN 8
int16_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int16_t output[VECTOR_LEN];
pq_biquad_state_t state =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiquadDf2Fixed16();
PQ_EndVector();
```

39.4.17 #define PQ_DF2_Cascade_Vector8_FP(*middle, last*)

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 16
float input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};

pq_biquad_state_t state1 =
{
     .param =
     {
         .a_1 = xxx,
```

```
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FP(false, false);
PQ_DF2_Cascade_Vector8_FP(true, true);
PQ_End_Vector_Func();
```

39.4.18 #define PQ_DF2_Cascade_Vector8_FX(*middle, last*)

The input and output data are fixed data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 16
int32_t input[VECTOR_LEN] = {1024.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

pq_biquad_state_t state1 =
{
    .param =
    {
        .a_1 = xxx,
        .a_2 = xxx,
        .b_0 = xxx,
        .b_1 = xxx,
        .b_2 = xxx,
    },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_Initiate_Vector_Func(pSrc, pDst);
PQ_DF2_Cascade_Vector8_FX(false, false);
PQ_DF2_Cascade_Vector8_FX(true, true);
PQ_End_Vector_Func();
```

39.4.19 #define PQ_Vector8BiqaudDf2CascadeF32( )

The input and output data are float data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
float input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
float output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};

pq_biquad_state_t state1 =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiqaudDf2CascadeF32();
PQ_EndVector();
```

39.4.20 #define PQ_Vector8BiqaudDf2CascadeFixed32( )

The input and output data are fixed 32-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};
```

```
pq_biquad_state_t state1 =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiqaudDf2CascadeFixed32();
PQ_EndVector();
```

39.4.21 #define PQ_Vector8BiqaudDf2CascadeFixed16( )

The input and output data are fixed 16-bit data. The data flow is input -> biquad side 1 -> biquad side 0 -> output.

```
#define VECTOR_LEN 8
int32_t input[VECTOR_LEN] = {1, 2, 3, 4, 5, 6, 7, 8};
int32_t output[VECTOR_LEN];
pq_biquad_state_t state0 =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};

pq_biquad_state_t state1 =
{
     .param =
     {
         .a_1 = xxx,
         .a_2 = xxx,
         .b_0 = xxx,
         .b_1 = xxx,
         .b_2 = xxx,
     },
};

PQ_BiquadRestoreInternalState(POWERQUAD, 0, &state0);
PQ_BiquadRestoreInternalState(POWERQUAD, 1, &state1);

PQ_StartVector(input, output, VECTOR_LEN);
PQ_Vector8BiqaudDf2CascadeFixed16();
PQ_EndVector();
```

39.4.22 #define POWERQUAD_MAKE_MATRIX_LEN(*mat1Row, mat1Col, mat2Col*) (((uint32_t)(mat1Row) << 0U) | ((uint32_t)(mat1Col) << 8U) | ((uint32_t)(mat2Col) << 16U))

39.4.23 #define PQ_Q31_2_FLOAT(*x*) (((float)(x)) / 2147483648.0f)

39.4.24 #define PQ_Q15_2_FLOAT(*x*) (((float)(x)) / 32768.0f)

39.5 Enumeration Type Documentation

39.5.1 enum pq_computationengine_t

Enumerator

kPQ_CP_PQ Math engine.
kPQ_CP_MTX Matrix engine.
kPQ_CP_FFT FFT engine.
kPQ_CP_FIR FIR engine.
kPQ_CP_CORDIC CORDIC engine.

39.5.2 enum pq_format_t

Enumerator

kPQ_16Bit Int16 Fixed point.
kPQ_32Bit Int32 Fixed point.
kPQ_Float Float point.

39.5.3 enum pq_cordic_iter_t

Enumerator

kPQ_Iteration_8 Iterate 8 times.
kPQ_Iteration_16 Iterate 16 times.
kPQ_Iteration_24 Iterate 24 times.

39.6 Function Documentation

39.6.1 void PQ_GetDefaultConfig (pq_config_t * *config*)

This function initializes the POWERQUAD configuration structure to a default value. FORMAT register field definitions Bits[15:8] scaler (for scaled 'q31' formats) Bits[5:4] external format. 00b=q15, 01b=q31,

10b=float Bits[1:0] internal format. 00b=q15, 01b=q31, 10b=float POWERQUAD->INAFORMAT = (config->inputAPrescale << 8U) | (config->inputAFormat << 4U) | config->machineFormat

For all Powerquad operations internal format must be float (with the only exception being the FFT related functions, ie FFT/IFFT/DCT/IDCT which must be set to q31). The default values are: config->inputAFormat = kPQ_Float; config->inputAPrescale = 0; config->inputBFormat = kPQ_Float; config->inputBPrescale = 0; config->outputFormat = kPQ_Float; config->outputPrescale = 0; config->tmpFormat = kPQ_Float; config->tmpPrescale = 0; config->machineFormat = kPQ_Float; config->tmpBase = 0xE0000000;

Parameters

config	Pointer to "pq_config_t" structure.

39.6.2 void PQ_SetConfig (POWERQUAD_Type * *base,* const pq_config_t * *config*)

Parameters

base	POWERQUAD peripheral base address
config	Pointer to "pq_config_t" structure.

39.6.3 static void PQ_SetCoprocessorScaler (POWERQUAD_Type * *base,* const pq_prescale_t * *prescale*) `[inline]`, `[static]`

Parameters

base	POWERQUAD peripheral base address
prescale	Pointer to "pq_prescale_t" structure.

39.6.4 void PQ_Init (POWERQUAD_Type * *base*)

Parameters

base	POWERQUAD peripheral base address.

39.6.5 void PQ_Deinit (POWERQUAD_Type * *base*)

Parameters

base	POWERQUAD peripheral base address.

39.6.6 void PQ_SetFormat (POWERQUAD_Type * *base,* pq_computationengine_t *engine,* pq_format_t *format*)

Parameters

base	POWERQUAD peripheral base address
engine	Computation engine
format	Data format

39.6.7 static void PQ_WaitDone (POWERQUAD_Type * *base*) [inline], [static]

Parameters

base	POWERQUAD peripheral base address

39.6.8 static void PQ_LnF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

**pSrc*	points to the block of input data. The range of the input value is (0 +INFINITY).
**pDst*	points to the block of output data

39.6.9 static void PQ_InvF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

*pSrc	points to the block of input data. The range of the input value is non-zero.
*pDst	points to the block of output data

39.6.10 static void PQ_SqrtF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

*pSrc	points to the block of input data. The range of the input value is [0 +INFINITY).
*pDst	points to the block of output data

39.6.11 static void PQ_InvSqrtF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

*pSrc	points to the block of input data. The range of the input value is (0 +INFINITY).
*pDst	points to the block of output data

39.6.12 static void PQ_EtoxF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

*pSrc	points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
*pDst	points to the block of output data

39.6.13 static void PQ_EtonxF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

*pSrc	points to the block of input data. The range of the input value is (-INFINITY +INFINITY).
*pDst	points to the block of output data

39.6.14 static void PQ_SinF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

*pSrc	points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
*pDst	points to the block of output data

39.6.15 static void PQ_CosF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

*pSrc	points to the block of input data. The input value is in radians, the range is (-INFINITY +INFINITY).
*pDst	points to the block of output data

39.6.16 static void PQ_BiquadF32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data

39.6.17 static void PQ_DivF32 (float * *x1,* float * *x2,* float * *pDst*) [inline], [static]

Get x1 / x2.

Parameters

x1	x1
x2	x2
**pDst*	points to the block of output data

39.6.18 static void PQ_Biquad1F32 (float * *pSrc,* float * *pDst*) [inline], [static]

Parameters

**pSrc*	points to the block of input data
**pDst*	points to the block of output data

39.6.19 static int32_t PQ_LnFixed (int32_t *val*) [inline], [static]

Parameters

val	value to be calculated. The range of the input value is (0 +INFINITY).

Returns

returns ln(val).

39.6.20 static int32_t PQ_InvFixed (int32_t *val*) [inline], [static]

Parameters

val	value to be calculated. The range of the input value is non-zero.

Returns

returns inv(val).

39.6.21 static uint32_t PQ_SqrtFixed (uint32_t *val*) [inline], [static]

Parameters

val	value to be calculated. The range of the input value is [0 +INFINITY).

Returns

returns sqrt(val).

39.6.22 static int32_t PQ_InvSqrtFixed (int32_t *val*) [inline], [static]

Parameters

val	value to be calculated. The range of the input value is (0 +INFINITY).

Returns

returns 1/sqrt(val).

39.6.23 static int32_t PQ_EtoxFixed (int32_t *val*) [inline], [static]

Parameters

val	value to be calculated. The range of the input value is (-INFINITY +INFINITY).

Returns

returns etox^(val).

39.6.24 static int32_t PQ_EtonxFixed (int32_t *val*) [inline], [static]

Parameters

val	value to be calculated. The range of the input value is (-INFINITY +INFINITY).

Returns

returns etonx^(val).

39.6.25 static int32_t PQ_SinQ31 (int32_t *val*) [inline], [static]

Parameters

val	value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

Returns

returns sin(val).

39.6.26 static int16_t PQ_SinQ15 (int16_t *val*) [inline], [static]

Parameters

val	value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

Returns

returns sin(val).

39.6.27 static int32_t PQ_CosQ31 (int32_t *val*) [inline], [static]

Parameters

val	value to be calculated. The input value is [-1, 1] in Q31 format, which means [-pi, pi].

Returns

returns cos(val).

39.6.28 static int16_t PQ_CosQ15 (int16_t *val*) [inline], [static]

Parameters

val	value to be calculated. The input value is [-1, 1] in Q15 format, which means [-pi, pi].

Returns

returns sin(val).

39.6.29 static int32_t PQ_BiquadFixed (int32_t *val*) [inline], [static]

Parameters

val	value to be calculated

Returns

returns biquad(val).

39.6.30 void PQ_VectorLnF32 (float ∗ *pSrc,* float ∗ *pDst,* int32_t *length*)

Parameters

∗*pSrc*	points to the block of input data
∗*pDst*	points to the block of output data
length	the block of input data.

39.6.31 void PQ_VectorInvF32 (float ∗ *pSrc,* float ∗ *pDst,* int32_t *length*)

Parameters

∗*pSrc*	points to the block of input data
∗*pDst*	points to the block of output data
length	the block of input data.

39.6.32 void PQ_VectorSqrtF32 (float ∗ *pSrc,* float ∗ *pDst,* int32_t *length*)

Parameters

∗*pSrc*	points to the block of input data
∗*pDst*	points to the block of output data
length	the block of input data.

39.6.33 void PQ_VectorInvSqrtF32 (float ∗ *pSrc,* float ∗ *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.34 void PQ_VectorEtoxF32 (float * *pSrc,* float * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.35 void PQ_VectorEtonxF32 (float * *pSrc,* float * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.36 void PQ_VectorSinF32 (float * *pSrc,* float * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.37 void PQ_VectorCosF32 (float * *pSrc,* float * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.38 void PQ_VectorLnFixed32 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.39 void PQ_VectorInvFixed32 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.40 void PQ_VectorSqrtFixed32 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.41 void PQ_VectorInvSqrtFixed32 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.42 void PQ_VectorEtoxFixed32 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.43 void PQ_VectorEtonxFixed32 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.44 void PQ_VectorSinQ15 (int16_t * *pSrc,* int16_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.45 void PQ_VectorCosQ15 (int16_t * *pSrc,* int16_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.46 void PQ_VectorSinQ31 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.47 void PQ_VectorCosQ31 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.48 void PQ_VectorLnFixed16 (int16_t * *pSrc,* int16_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.49 void PQ_VectorInvFixed16 (int16_t * *pSrc,* int16_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.50 void PQ_VectorSqrtFixed16 (int16_t * *pSrc,* int16_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.51 void PQ_VectorInvSqrtFixed16 (int16_t * *pSrc,* int16_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block of input data.

39.6.52 void PQ_VectorEtoxFixed16 (int16_t * *pSrc,* int16_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data

length	the block of input data.

39.6.53 void PQ_VectorEtonxFixed16 (int16_t ∗ *pSrc,* int16_t ∗ *pDst,* int32_t *length*)

Parameters

∗*pSrc*	points to the block of input data
∗*pDst*	points to the block of output data
length	the block of input data.

39.6.54 void PQ_VectorBiqaudDf2F32 (float ∗ *pSrc,* float ∗ *pDst,* int32_t *length*)

Parameters

∗*pSrc*	points to the block of input data
∗*pDst*	points to the block of output data
length	the block size of input data.

39.6.55 void PQ_VectorBiqaudDf2Fixed32 (int32_t ∗ *pSrc,* int32_t ∗ *pDst,* int32_t *length*)

Parameters

∗*pSrc*	points to the block of input data
∗*pDst*	points to the block of output data
length	the block size of input data

39.6.56 void PQ_VectorBiqaudDf2Fixed16 (int16_t ∗ *pSrc,* int16_t ∗ *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block size of input data

39.6.57 void PQ_VectorBiqaudCascadeDf2F32 (float * *pSrc,* float * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block size of input data

39.6.58 void PQ_VectorBiqaudCascadeDf2Fixed32 (int32_t * *pSrc,* int32_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data
length	the block size of input data

39.6.59 void PQ_VectorBiqaudCascadeDf2Fixed16 (int16_t * *pSrc,* int16_t * *pDst,* int32_t *length*)

Parameters

*pSrc	points to the block of input data
*pDst	points to the block of output data

length	the block size of input data

39.6.60 int32_t PQ_ArctanFixed (POWERQUAD_Type * *base*, int32_t *x*, int32_t *y*, pq_cordic_iter_t *iteration*)

Parameters

base	POWERQUAD peripheral base address
x	value of opposite
y	value of adjacent
iteration	iteration times

Returns

The return value is in the range of -2^27 to 2^27, which means -pi to pi.

Note

The sum of x and y should not exceed the range of int32_t.
Larger input number gets higher output accuracy, for example the arctan(0.5), the result of PQ_ArctanFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_ArctanFixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

39.6.61 int32_t PQ_ArctanhFixed (POWERQUAD_Type * *base*, int32_t *x*, int32_t *y*, pq_cordic_iter_t *iteration*)

Parameters

base	POWERQUAD peripheral base address
x	value of opposite
y	value of adjacent

iteration	iteration times

Returns

The return value is in the range of -2^27 to 2^27, which means -1 to 1.

Note

The sum of x and y should not exceed the range of int32_t.
Larger input number gets higher output accuracy, for example the arctanh(0.5), the result of PQ_ArctanhFixed(POWERQUAD, 100000, 200000, kPQ_Iteration_24) is more accurate than PQ_ArctanhFixed(POWERQUAD, 1, 2, kPQ_Iteration_24).

39.6.62 static int32_t PQ_Biquad1Fixed (int32_t *val*) `[inline], [static]`

Parameters

val	value to be calculated

Returns

returns biquad(val).

39.6.63 void PQ_TransformCFFT (POWERQUAD_Type ∗ *base,* uint32_t *length,* void ∗ *pData,* void ∗ *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	number of input samples
pData	input data
pResult	output data.

39.6.64 void PQ_TransformRFFT (POWERQUAD_Type ∗ *base,* uint32_t *length,* void ∗ *pData,* void ∗ *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	number of input samples
pData	input data
pResult	output data.

39.6.65 void PQ_TransformIFFT (POWERQUAD_Type * *base*, uint32_t *length*, void * *pData*, void * *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	number of input samples
pData	input data
pResult	output data.

39.6.66 void PQ_TransformCDCT (POWERQUAD_Type * *base*, uint32_t *length*, void * *pData*, void * *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	number of input samples
pData	input data
pResult	output data.

39.6.67 void PQ_TransformRDCT (POWERQUAD_Type * *base*, uint32_t *length*, void * *pData*, void * *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	number of input samples
pData	input data
pResult	output data.

39.6.68 void PQ_TransformIDCT (POWERQUAD_Type * *base*, uint32_t *length*, void * *pData*, void * *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	number of input samples
pData	input data
pResult	output data.

39.6.69 void PQ_BiquadBackUpInternalState (POWERQUAD_Type * *base*, int32_t *biquad_num*, pq_biquad_state_t * *state*)

Parameters

base	POWERQUAD peripheral base address
biquad_num	biquad side
state	point to states.

39.6.70 void PQ_BiquadRestoreInternalState (POWERQUAD_Type * *base*, int32_t *biquad_num*, pq_biquad_state_t * *state*)

Parameters

base	POWERQUAD peripheral base address
biquad_num	biquad side
state	point to states.

39.6.71 void PQ_BiquadCascadeDf2Init (pq_biquad_cascade_df2_instance * *S*, uint8_t *numStages*, pq_biquad_state_t * *pState*)

Parameters

in,out	*S	points to an instance of the filter data structure.
in	*numStages*	number of 2nd order stages in the filter.
in	*pState	points to the state buffer.

39.6.72 void PQ_BiquadCascadeDf2F32 (const pq_biquad_cascade_df2_instance * *S*, float * *pSrc*, float * *pDst*, uint32_t *blockSize*)

Parameters

in	*S	points to an instance of the filter data structure.
in	*pSrc	points to the block of input data.
out	*pDst	points to the block of output data
in	*blockSize*	number of samples to process.

39.6.73 void PQ_BiquadCascadeDf2Fixed32 (const pq_biquad_cascade_-df2_instance * *S*, int32_t * *pSrc*, int32_t * *pDst*, uint32_t *blockSize*)

Parameters

in	*S	points to an instance of the filter data structure.
in	*pSrc	points to the block of input data.
out	*pDst	points to the block of output data
in	*blockSize*	number of samples to process.

39.6.74 void PQ_BiquadCascadeDf2Fixed16 (const pq_biquad_cascade_-df2_instance * *S*, int16_t * *pSrc*, int16_t * *pDst*, uint32_t *blockSize*)

Parameters

in	*S	points to an instance of the filter data structure.
in	*pSrc	points to the block of input data.
out	*pDst	points to the block of output data
in	*blockSize*	number of samples to process.

39.6.75 void PQ_FIR (POWERQUAD_Type * *base,* const void * *pAData,* int32_t *ALength,* const void * *pBData,* int32_t *BLength,* void * *pResult,* uint32_t *opType*)

Parameters

base	POWERQUAD peripheral base address
pAData	the first input sequence
ALength	number of the first input sequence
pBData	the second input sequence
BLength	number of the second input sequence
pResult	array for the output data
opType	operation type, could be PQ_FIR_FIR, PQ_FIR_CONVOLUTION, PQ_FIR_CORRELATION.

39.6.76 void PQ_FIRIncrement (POWERQUAD_Type * *base,* int32_t *ALength,* int32_t *BLength,* int32_t *xOffset*)

```
This function can be used after pq_fir() for incremental FIR
operation when new x data are available
```

Parameters

base	POWERQUAD peripheral base address
ALength	number of input samples

BLength	number of taps
xOffset	offset for number of input samples

39.6.77 void PQ_MatrixAddition (POWERQUAD_Type * *base*, uint32_t *length*, void * *pAData*, void * *pBData*, void * *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
pAData	input matrix A
pBData	input matrix B
pResult	array for the output data.

39.6.78 void PQ_MatrixSubtraction (POWERQUAD_Type * *base*, uint32_t *length*, void * *pAData*, void * *pBData*, void * *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
pAData	input matrix A
pBData	input matrix B
pResult	array for the output data.

39.6.79 void PQ_MatrixMultiplication (POWERQUAD_Type * *base*, uint32_t *length*, void * *pAData*, void * *pBData*, void * *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
pAData	input matrix A
pBData	input matrix B
pResult	array for the output data.

39.6.80 void PQ_MatrixProduct (POWERQUAD_Type ∗ *base*, uint32_t *length*, void ∗ *pAData*, void ∗ *pBData*, void ∗ *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
pAData	input matrix A
pBData	input matrix B
pResult	array for the output data.

39.6.81 void PQ_VectorDotProduct (POWERQUAD_Type ∗ *base*, uint32_t *length*, void ∗ *pAData*, void ∗ *pBData*, void ∗ *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	length of vector
pAData	input vector A
pBData	input vector B
pResult	array for the output data.

39.6.82 void PQ_MatrixInversion (POWERQUAD_Type * *base,* uint32_t *length,* void * *pData,* void * *pTmpData,* void * *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
pData	input matrix
pTmpData	input temporary matrix, pTmpData length not less than pData lenght and 1024 words is sufficient for the largest supported matrix.
pResult	array for the output data, round down for fixed point.

39.6.83 void PQ_MatrixTranspose (POWERQUAD_Type ∗ *base*, uint32_t *length*, void ∗ *pData*, void ∗ *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
pData	input matrix
pResult	array for the output data.

39.6.84 void PQ_MatrixScale (POWERQUAD_Type ∗ *base*, uint32_t *length*, float *misc*, const void ∗ *pData*, void ∗ *pResult*)

Parameters

base	POWERQUAD peripheral base address
length	rows and cols for matrix. LENGTH register configuration: LENGTH[23:16] = M2 cols LENGTH[15:8] = M1 cols LENGTH[7:0] = M1 rows This could be constructed using macro POWERQUAD_MAKE_MATRIX_LEN.
misc	scaling parameters
pData	input matrix
pResult	array for the output data.

Chapter 40
PUF: Physical Unclonable Function

40.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Physical Unclonable Function (PUF) module of MCUXpresso SDK devices. The PUF controller provides a secure key storage without injecting or provisioning device unique PUF root key.

Blocking synchronous APIs are provided for generating the activation code, intrinsic key generation, storing and reconstructing keys using PUF hardware. The PUF operations are complete (and results are made availabe for further usage) when a function returns. When called, these functions do not return until an PUF operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

40.2 PUF Driver Initialization and deinitialization

PUF Driver is initialized by calling the PUF_Init() function, it resets the PUF module, enables it's clock and enables power to PUF SRAM. PUF Driver is deinitialized by calling the PUF_Deinit() function, it disables PUF module clock, asserts peripheral reset and disables power to PUF SRAM.

40.3 Comments about API usage in RTOS

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

40.4 Comments about API usage in interrupt handler

All APIs can be used from interrupt handler although execution time shall be considered (interrupt latency of equal and lower priority interrupts increases).

40.5 PUF Driver Examples

40.5.1 Simple examples

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

Macros

- #define PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x) ((160u + (((((x) << 3) + 255u) >> 8) << 8)) >> 3)
  Get Key Code size in bytes from key size in bytes at compile time.

Enumerations

- enum puf_key_slot_t {
  kPUF_KeySlot0 = 0U,
  kPUF_KeySlot1 = 1U }
  PUF key slot.
- enum
  PUF status return codes.

Driver version

- #define FSL_PUF_DRIVER_VERSION (MAKE_VERSION(2, 1, 6))
  PUF driver version.

40.6 Macro Definition Documentation

40.6.1 #define FSL_PUF_DRIVER_VERSION (MAKE_VERSION(2, 1, 6))

Version 2.1.6.

Current version: 2.1.6

Change log:

- 2.0.0
  - Initial version.
- 2.0.1
  - Fixed puf_wait_usec function optimization issue.
- 2.0.2
  - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.
- 2.0.3
  - Fix MISRA C-2012 issue.
- 2.1.0
  - Align driver with PUF SRAM controller registers on LPCXpresso55s16.
  - Update initizalition logic .
- 2.1.1
  - Fix ARMGCC build warning .
- 2.1.2
  - Update: Add automatic big to little endian swap for user (pre-shared) keys destinated to secret hardware bus (PUF key index 0).
- 2.1.3
  - Fix MISRA C-2012 issue.
- 2.1.4
  - Replace register uint32_t ticksCount with volatile uint32_t ticksCount in puf_wait_usec() to prevent optimization out delay loop.
- 2.1.5
  - Use common SDK delay in puf_wait_usec()

- 2.1.6
  - Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.

40.6.2 #define PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(*x*) ((160u + (((((x) << 3) + 255u) >> 8) << 8)) >> 3)

40.7 Enumeration Type Documentation

40.7.1 enum puf_key_slot_t

Enumerator

kPUF_KeySlot0 PUF key slot 0.
kPUF_KeySlot1 PUF key slot 1.

40.7.2 anonymous enum

Chapter 41
SCTimer: SCTimer/PWM (SCT)

41.1 Overview

The MCUXpresso SDK provides a driver for the SCTimer Module (SCT) of MCUXpresso SDK devices.

41.2 Function groups

The SCTimer driver supports the generation of PWM signals. The driver also supports enabling events in various states of the SCTimer and the actions that will be triggered when an event occurs.

41.2.1 Initialization and deinitialization

The function SCTIMER_Init() initializes the SCTimer with specified configurations. The function SCTIMER_GetDefaultConfig() gets the default configurations.

The function SCTIMER_Deinit() halts the SCTimer counter and turns off the module clock.

41.2.2 PWM Operations

The function SCTIMER_SetupPwm() sets up SCTimer channels for PWM output. The function can set up the PWM signal properties duty cycle and level-mode (active low or high) to use. However, the same PWM period and PWM mode (edge or center-aligned) is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 1 and 100.

The function SCTIMER_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular SCTimer channel.

41.2.3 Status

Provides functions to get and clear the SCTimer status.

41.2.4 Interrupt

Provides functions to enable/disable SCTimer interrupts and get current enabled interrupts.

41.3 SCTimer State machine and operations

The SCTimer has 10 states and each state can have a set of events enabled that can trigger a user specified action when the event occurs.

41.3.1 SCTimer event operations

The user can create an event and enable it in the current state using the functions SCTIMER_CreateAndScheduleEvent() and SCTIMER_ScheduleEvent(). SCTIMER_CreateAndScheduleEvent() creates a new event based on the users preference and enables it in the current state. SCTIMER_ScheduleEvent() enables an event created earlier in the current state.

41.3.2 SCTimer state operations

The user can get the current state number by calling SCTIMER_GetCurrentState(), he can use this state number to set state transitions when a particular event is triggered.

Once the user has created and enabled events for the current state he can go to the next state by calling the function SCTIMER_IncreaseState(). The user can then start creating events to be enabled in this new state.

41.3.3 SCTimer action operations

There are a set of functions that decide what action should be taken when an event is triggered. SCTIMER_SetupCaptureAction() sets up which counter to capture and which capture register to read on event trigger. SCTIMER_SetupNextStateAction() sets up which state the SCTimer state machine should transition to on event trigger. SCTIMER_SetupOutputSetAction() sets up which pin to set on event trigger. SCTIMER_SetupOutputClearAction() sets up which pin to clear on event trigger. SCTIMER_SetupOutputToggleAction() sets up which pin to toggle on event trigger. SCTIMER_SetupCounterLimitAction() sets up which counter will be limited on event trigger. SCTIMER_SetupCounterStopAction() sets up which counter will be stopped on event trigger. SCTIMER_SetupCounterStartAction() sets up which counter will be started on event trigger. SCTIMER_SetupCounterHaltAction() sets up which counter will be halted on event trigger. SCTIMER_SetupDmaTriggerAction() sets up which DMA request will be activated on event trigger.

41.4 16-bit counter mode

The SCTimer is configurable to run as two 16-bit counters via the enableCounterUnify flag that is available in the configuration structure passed in to the SCTIMER_Init() function.

When operating in 16-bit mode, it is important the user specify the appropriate counter to use when working with the functions: SCTIMER_StartTimer(), SCTIMER_StopTimer(), SCTIMER_CreateAndScheduleEvent(), SCTIMER_SetupCaptureAction(), SCTIMER_SetupCounterLimitAction(), SCTIM-

ER_SetupCounterStopAction(), SCTIMER_SetupCounterStartAction(), and SCTIMER_SetupCounter-HaltAction().

41.5 Typical use case

41.5.1 PWM output

Output a PWM signal on 2 SCTimer channels with different duty cycles. Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/sctimer

Files

- file fsl_sctimer.h

Data Structures

- struct sctimer_pwm_signal_param_t
  Options to configure a SCTimer PWM signal. More...
- struct sctimer_config_t
  SCTimer configuration structure. More...

Typedefs

- typedef void(∗ sctimer_event_callback_t)(void)
  SCTimer callback typedef.

Enumerations

- enum sctimer_pwm_mode_t {
  kSCTIMER_EdgeAlignedPwm = 0U,
  kSCTIMER_CenterAlignedPwm }
  SCTimer PWM operation modes.
- enum sctimer_counter_t {
  kSCTIMER_Counter_L = (1U << 0),
  kSCTIMER_Counter_H = (1U << 1),
  kSCTIMER_Counter_U = (1U << 2) }
  SCTimer counters type.
- enum sctimer_input_t {
  kSCTIMER_Input_0 = 0U,
  kSCTIMER_Input_1,
  kSCTIMER_Input_2,
  kSCTIMER_Input_3,
  kSCTIMER_Input_4,
  kSCTIMER_Input_5,
  kSCTIMER_Input_6,
  kSCTIMER_Input_7 }
  List of SCTimer input pins.

- enum sctimer_out_t {
  kSCTIMER_Out_0 = 0U,
  kSCTIMER_Out_1,
  kSCTIMER_Out_2,
  kSCTIMER_Out_3,
  kSCTIMER_Out_4,
  kSCTIMER_Out_5,
  kSCTIMER_Out_6,
  kSCTIMER_Out_7,
  kSCTIMER_Out_8,
  kSCTIMER_Out_9 }
  List of SCTimer output pins.
- enum sctimer_pwm_level_select_t {
  kSCTIMER_LowTrue = 0U,
  kSCTIMER_HighTrue }
  SCTimer PWM output pulse mode: high-true, low-true or no output.
- enum sctimer_clock_mode_t {
  kSCTIMER_System_ClockMode = 0U,
  kSCTIMER_Sampled_ClockMode,
  kSCTIMER_Input_ClockMode,
  kSCTIMER_Asynchronous_ClockMode }
  SCTimer clock mode options.
- enum sctimer_clock_select_t {
  kSCTIMER_Clock_On_Rise_Input_0 = 0U,
  kSCTIMER_Clock_On_Fall_Input_0,
  kSCTIMER_Clock_On_Rise_Input_1,
  kSCTIMER_Clock_On_Fall_Input_1,
  kSCTIMER_Clock_On_Rise_Input_2,
  kSCTIMER_Clock_On_Fall_Input_2,
  kSCTIMER_Clock_On_Rise_Input_3,
  kSCTIMER_Clock_On_Fall_Input_3,
  kSCTIMER_Clock_On_Rise_Input_4,
  kSCTIMER_Clock_On_Fall_Input_4,
  kSCTIMER_Clock_On_Rise_Input_5,
  kSCTIMER_Clock_On_Fall_Input_5,
  kSCTIMER_Clock_On_Rise_Input_6,
  kSCTIMER_Clock_On_Fall_Input_6,
  kSCTIMER_Clock_On_Rise_Input_7,
  kSCTIMER_Clock_On_Fall_Input_7 }
  SCTimer clock select options.
- enum sctimer_conflict_resolution_t {
  kSCTIMER_ResolveNone = 0U,
  kSCTIMER_ResolveSet,
  kSCTIMER_ResolveClear,
  kSCTIMER_ResolveToggle }
  SCTimer output conflict resolution options.

- enum sctimer_event_active_direction_t {
  kSCTIMER_ActiveIndependent = 0U,
  kSCTIMER_ActiveInCountUp,
  kSCTIMER_ActiveInCountDown }
  List of SCTimer event generation active direction when the counters are operating in BIDIR mode.
- enum sctimer_event_t
  List of SCTimer event types.
- enum sctimer_interrupt_enable_t {
  kSCTIMER_Event0InterruptEnable = (1U << 0),
  kSCTIMER_Event1InterruptEnable = (1U << 1),
  kSCTIMER_Event2InterruptEnable = (1U << 2),
  kSCTIMER_Event3InterruptEnable = (1U << 3),
  kSCTIMER_Event4InterruptEnable = (1U << 4),
  kSCTIMER_Event5InterruptEnable = (1U << 5),
  kSCTIMER_Event6InterruptEnable = (1U << 6),
  kSCTIMER_Event7InterruptEnable = (1U << 7),
  kSCTIMER_Event8InterruptEnable = (1U << 8),
  kSCTIMER_Event9InterruptEnable = (1U << 9),
  kSCTIMER_Event10InterruptEnable = (1U << 10),
  kSCTIMER_Event11InterruptEnable = (1U << 11),
  kSCTIMER_Event12InterruptEnable = (1U << 12) }
  List of SCTimer interrupts.
- enum sctimer_status_flags_t {
  kSCTIMER_Event0Flag = (1U << 0),
  kSCTIMER_Event1Flag = (1U << 1),
  kSCTIMER_Event2Flag = (1U << 2),
  kSCTIMER_Event3Flag = (1U << 3),
  kSCTIMER_Event4Flag = (1U << 4),
  kSCTIMER_Event5Flag = (1U << 5),
  kSCTIMER_Event6Flag = (1U << 6),
  kSCTIMER_Event7Flag = (1U << 7),
  kSCTIMER_Event8Flag = (1U << 8),
  kSCTIMER_Event9Flag = (1U << 9),
  kSCTIMER_Event10Flag = (1U << 10),
  kSCTIMER_Event11Flag = (1U << 11),
  kSCTIMER_Event12Flag = (1U << 12),
  kSCTIMER_BusErrorLFlag,
  kSCTIMER_BusErrorHFlag }
  List of SCTimer flags.

Driver version

- #define FSL_SCTIMER_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))
  Version.

Initialization and deinitialization

- status_t SCTIMER_Init (SCT_Type *base, const sctimer_config_t *config)
  Ungates the SCTimer clock and configures the peripheral for basic operation.
- void SCTIMER_Deinit (SCT_Type *base)
  Gates the SCTimer clock.
- void SCTIMER_GetDefaultConfig (sctimer_config_t *config)
  Fills in the SCTimer configuration structure with the default settings.

PWM setup operations

- status_t SCTIMER_SetupPwm (SCT_Type *base, const sctimer_pwm_signal_param_t *pwmParams, sctimer_pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz, uint32_t *event)
  Configures the PWM signal parameters.
- void SCTIMER_UpdatePwmDutycycle (SCT_Type *base, sctimer_out_t output, uint8_t dutyCyclePercent, uint32_t event)
  Updates the duty cycle of an active PWM signal.

Interrupt Interface

- static void SCTIMER_EnableInterrupts (SCT_Type *base, uint32_t mask)
  Enables the selected SCTimer interrupts.
- static void SCTIMER_DisableInterrupts (SCT_Type *base, uint32_t mask)
  Disables the selected SCTimer interrupts.
- static uint32_t SCTIMER_GetEnabledInterrupts (SCT_Type *base)
  Gets the enabled SCTimer interrupts.

Status Interface

- static uint32_t SCTIMER_GetStatusFlags (SCT_Type *base)
  Gets the SCTimer status flags.
- static void SCTIMER_ClearStatusFlags (SCT_Type *base, uint32_t mask)
  Clears the SCTimer status flags.

Counter Start and Stop

- static void SCTIMER_StartTimer (SCT_Type *base, uint32_t countertoStart)
  Starts the SCTimer counter.
- static void SCTIMER_StopTimer (SCT_Type *base, uint32_t countertoStop)
  Halts the SCTimer counter.

Functions to create a new event and manage the state logic

- status_t SCTIMER_CreateAndScheduleEvent (SCT_Type *base, sctimer_event_t howToMonitor, uint32_t matchValue, uint32_t whichIO, sctimer_counter_t whichCounter, uint32_t *event)
  Create an event that is triggered on a match or IO and schedule in current state.
- void SCTIMER_ScheduleEvent (SCT_Type *base, uint32_t event)
  Enable an event in the current state.
- status_t SCTIMER_IncreaseState (SCT_Type *base)

Increase the state by 1.
- uint32_t SCTIMER_GetCurrentState (SCT_Type *base)
  Provides the current state.
- static void SCTIMER_SetCounterState (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t state)
  Set the counter current state.
- static uint16_t SCTIMER_GetCounterState (SCT_Type *base, sctimer_counter_t whichCounter)
  Get the counter current state value.

Actions to take in response to an event

- status_t SCTIMER_SetupCaptureAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t *captureRegister, uint32_t event)
  Setup capture of the counter value on trigger of a selected event.
- void SCTIMER_SetCallback (SCT_Type *base, sctimer_event_callback_t callback, uint32_t event)
  Receive noticification when the event trigger an interrupt.
- static void SCTIMER_SetupStateLdMethodAction (SCT_Type *base, uint32_t event, bool fgLoad)
  Change the load method of transition to the specified state.
- static void SCTIMER_SetupNextStateActionwithLdMethod (SCT_Type *base, uint32_t nextState, uint32_t event, bool fgLoad)
  Transition to the specified state with Load method.
- static void SCTIMER_SetupNextStateAction (SCT_Type *base, uint32_t nextState, uint32_t event)
  Transition to the specified state.
- static void SCTIMER_SetupEventActiveDirection (SCT_Type *base, sctimer_event_active_direction_t activeDirection, uint32_t event)
  Setup event active direction when the counters are operating in BIDIR mode.
- static void SCTIMER_SetupOutputSetAction (SCT_Type *base, uint32_t whichIO, uint32_t event)
  Set the Output.
- static void SCTIMER_SetupOutputClearAction (SCT_Type *base, uint32_t whichIO, uint32_t event)
  Clear the Output.
- void SCTIMER_SetupOutputToggleAction (SCT_Type *base, uint32_t whichIO, uint32_t event)
  Toggle the output level.
- static void SCTIMER_SetupCounterLimitAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
  Limit the running counter.
- static void SCTIMER_SetupCounterStopAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
  Stop the running counter.
- static void SCTIMER_SetupCounterStartAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
  Re-start the stopped counter.
- static void SCTIMER_SetupCounterHaltAction (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t event)
  Halt the running counter.
- static void SCTIMER_SetupDmaTriggerAction (SCT_Type *base, uint32_t dmaNumber, uint32_t event)
  Generate a DMA request.
- static void SCTIMER_SetCOUNTValue (SCT_Type *base, sctimer_counter_t whichCounter, uint32_t value)

Set the value of counter.
- static uint32_t SCTIMER_GetCOUNTValue (SCT_Type *base, sctimer_counter_t whichCounter)
  Get the value of counter.
- static void SCTIMER_SetEventInState (SCT_Type *base, uint32_t event, uint32_t state)
  Set the state mask bit field of EV_STATE register.
- static void SCTIMER_ClearEventInState (SCT_Type *base, uint32_t event, uint32_t state)
  Clear the state mask bit field of EV_STATE register.
- static bool SCTIMER_GetEventInState (SCT_Type *base, uint32_t event, uint32_t state)
  Get the state mask bit field of EV_STATE register.
- void SCTIMER_EventHandleIRQ (SCT_Type *base)
  SCTimer interrupt handler.

41.6 Data Structure Documentation

41.6.1 struct sctimer_pwm_signal_param_t

Data Fields

- sctimer_out_t output
  The output pin to use to generate the PWM signal.
- sctimer_pwm_level_select_t level
  PWM output active level select.
- uint8_t dutyCyclePercent
  PWM pulse width, value should be between 0 to 100 0 = always inactive signal (0% duty cycle) 100 = always active signal (100% duty cycle).

Field Documentation

(1) sctimer_pwm_level_select_t sctimer_pwm_signal_param_t::level

(2) uint8_t sctimer_pwm_signal_param_t::dutyCyclePercent

41.6.2 struct sctimer_config_t

This structure holds the configuration settings for the SCTimer peripheral. To initialize this structure to reasonable defaults, call the SCTMR_GetDefaultConfig() function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

Data Fields

- bool enableCounterUnify
  true: SCT operates as a unified 32-bit counter; false: SCT operates as two 16-bit counters.
- sctimer_clock_mode_t clockMode
  SCT clock mode value.
- sctimer_clock_select_t clockSelect
  SCT clock select value.

- bool enableBidirection_l
  true: Up-down count mode for the L or unified counter false: Up count mode only for the L or unified counter
- bool enableBidirection_h
  true: Up-down count mode for the H or unified counter false: Up count mode only for the H or unified counter.
- uint8_t prescale_l
  Prescale value to produce the L or unified counter clock.
- uint8_t prescale_h
  Prescale value to produce the H counter clock.
- uint8_t outInitState
  Defines the initial output value.
- uint8_t inputsync
  SCT INSYNC value, INSYNC field in the CONFIG register, from bit9 to bit 16.

Field Documentation

(1) bool sctimer_config_t::enableCounterUnify

User can use the 16-bit low counter and the 16-bit high counters at the same time; for Hardware limit, user can not use unified 32-bit counter and any 16-bit low/high counter at the same time.

(2) bool sctimer_config_t::enableBidirection_h

This field is used only if the enableCounterUnify is set to false

(3) uint8_t sctimer_config_t::prescale_h

This field is used only if the enableCounterUnify is set to false

(4) uint8_t sctimer_config_t::inputsync

it is used to define synchronization for input N: bit 9 = input 0 bit 10 = input 1 bit 11 = input 2 bit 12 = input 3 All other bits are reserved (bit13 ∼bit 16). How User to set the the value for the member inputsync. IE: delay for input0, and input 1, bypasses for input 2 and input 3 MACRO definition in user level. #define INPUTSYNC0 (0U) #define INPUTSYNC1 (1U) #define INPUTSYNC2 (2U) #define INPUTSYNC3 (3U) User Code. sctimerInfo.inputsync = (1 << INPUTSYNC2) | (1 << INPUTSYNC3);

41.7 Typedef Documentation

41.7.1 typedef void(∗ sctimer_event_callback_t)(void)

41.8 Enumeration Type Documentation

41.8.1 enum sctimer_pwm_mode_t

Enumerator

kSCTIMER_EdgeAlignedPwm Edge-aligned PWM.
kSCTIMER_CenterAlignedPwm Center-aligned PWM.

41.8.2 enum sctimer_counter_t

Enumerator

kSCTIMER_Counter_L 16-bit Low counter.
kSCTIMER_Counter_H 16-bit High counter.
kSCTIMER_Counter_U 32-bit Unified counter.

41.8.3 enum sctimer_input_t

Enumerator

kSCTIMER_Input_0 SCTIMER input 0.
kSCTIMER_Input_1 SCTIMER input 1.
kSCTIMER_Input_2 SCTIMER input 2.
kSCTIMER_Input_3 SCTIMER input 3.
kSCTIMER_Input_4 SCTIMER input 4.
kSCTIMER_Input_5 SCTIMER input 5.
kSCTIMER_Input_6 SCTIMER input 6.
kSCTIMER_Input_7 SCTIMER input 7.

41.8.4 enum sctimer_out_t

Enumerator

kSCTIMER_Out_0 SCTIMER output 0.
kSCTIMER_Out_1 SCTIMER output 1.
kSCTIMER_Out_2 SCTIMER output 2.
kSCTIMER_Out_3 SCTIMER output 3.
kSCTIMER_Out_4 SCTIMER output 4.
kSCTIMER_Out_5 SCTIMER output 5.
kSCTIMER_Out_6 SCTIMER output 6.
kSCTIMER_Out_7 SCTIMER output 7.
kSCTIMER_Out_8 SCTIMER output 8.
kSCTIMER_Out_9 SCTIMER output 9.

41.8.5 enum sctimer_pwm_level_select_t

Enumerator

kSCTIMER_LowTrue Low true pulses.
kSCTIMER_HighTrue High true pulses.

41.8.6 enum sctimer_clock_mode_t

Enumerator

kSCTIMER_System_ClockMode System Clock Mode.
kSCTIMER_Sampled_ClockMode Sampled System Clock Mode.
kSCTIMER_Input_ClockMode SCT Input Clock Mode.
kSCTIMER_Asynchronous_ClockMode Asynchronous Mode.

41.8.7 enum sctimer_clock_select_t

Enumerator

kSCTIMER_Clock_On_Rise_Input_0 Rising edges on input 0.
kSCTIMER_Clock_On_Fall_Input_0 Falling edges on input 0.
kSCTIMER_Clock_On_Rise_Input_1 Rising edges on input 1.
kSCTIMER_Clock_On_Fall_Input_1 Falling edges on input 1.
kSCTIMER_Clock_On_Rise_Input_2 Rising edges on input 2.
kSCTIMER_Clock_On_Fall_Input_2 Falling edges on input 2.
kSCTIMER_Clock_On_Rise_Input_3 Rising edges on input 3.
kSCTIMER_Clock_On_Fall_Input_3 Falling edges on input 3.
kSCTIMER_Clock_On_Rise_Input_4 Rising edges on input 4.
kSCTIMER_Clock_On_Fall_Input_4 Falling edges on input 4.
kSCTIMER_Clock_On_Rise_Input_5 Rising edges on input 5.
kSCTIMER_Clock_On_Fall_Input_5 Falling edges on input 5.
kSCTIMER_Clock_On_Rise_Input_6 Rising edges on input 6.
kSCTIMER_Clock_On_Fall_Input_6 Falling edges on input 6.
kSCTIMER_Clock_On_Rise_Input_7 Rising edges on input 7.
kSCTIMER_Clock_On_Fall_Input_7 Falling edges on input 7.

41.8.8 enum sctimer_conflict_resolution_t

Specifies what action should be taken if multiple events dictate that a given output should be both set and cleared at the same time

Enumerator

kSCTIMER_ResolveNone No change.
kSCTIMER_ResolveSet Set output.
kSCTIMER_ResolveClear Clear output.
kSCTIMER_ResolveToggle Toggle output.

41.8.9 enum sctimer_event_active_direction_t

Enumerator

kSCTIMER_ActiveIndependent This event is triggered regardless of the count direction.
kSCTIMER_ActiveInCountUp This event is triggered only during up-counting when BIDIR = 1.
kSCTIMER_ActiveInCountDown This event is triggered only during down-counting when BIDIR = 1.

41.8.10 enum sctimer_interrupt_enable_t

Enumerator

kSCTIMER_Event0InterruptEnable Event 0 interrupt.
kSCTIMER_Event1InterruptEnable Event 1 interrupt.
kSCTIMER_Event2InterruptEnable Event 2 interrupt.
kSCTIMER_Event3InterruptEnable Event 3 interrupt.
kSCTIMER_Event4InterruptEnable Event 4 interrupt.
kSCTIMER_Event5InterruptEnable Event 5 interrupt.
kSCTIMER_Event6InterruptEnable Event 6 interrupt.
kSCTIMER_Event7InterruptEnable Event 7 interrupt.
kSCTIMER_Event8InterruptEnable Event 8 interrupt.
kSCTIMER_Event9InterruptEnable Event 9 interrupt.
kSCTIMER_Event10InterruptEnable Event 10 interrupt.
kSCTIMER_Event11InterruptEnable Event 11 interrupt.
kSCTIMER_Event12InterruptEnable Event 12 interrupt.

41.8.11 enum sctimer_status_flags_t

Enumerator

kSCTIMER_Event0Flag Event 0 Flag.
kSCTIMER_Event1Flag Event 1 Flag.
kSCTIMER_Event2Flag Event 2 Flag.
kSCTIMER_Event3Flag Event 3 Flag.
kSCTIMER_Event4Flag Event 4 Flag.
kSCTIMER_Event5Flag Event 5 Flag.
kSCTIMER_Event6Flag Event 6 Flag.
kSCTIMER_Event7Flag Event 7 Flag.
kSCTIMER_Event8Flag Event 8 Flag.
kSCTIMER_Event9Flag Event 9 Flag.
kSCTIMER_Event10Flag Event 10 Flag.
kSCTIMER_Event11Flag Event 11 Flag.

kSCTIMER_Event12Flag Event 12 Flag.
kSCTIMER_BusErrorLFlag Bus error due to write when L counter was not halted.
kSCTIMER_BusErrorHFlag Bus error due to write when H counter was not halted.

41.9 Function Documentation

41.9.1 status_t SCTIMER_Init (SCT_Type * *base*, const sctimer_config_t * *config*)

Note

This API should be called at the beginning of the application using the SCTimer driver.

Parameters

base	SCTimer peripheral base address
config	Pointer to the user configuration structure.

Returns

kStatus_Success indicates success; Else indicates failure.

41.9.2 void SCTIMER_Deinit (SCT_Type * *base*)

Parameters

base	SCTimer peripheral base address

41.9.3 void SCTIMER_GetDefaultConfig (sctimer_config_t * *config*)

The default values are:

```
*  config->enableCounterUnify = true;
*  config->clockMode = kSCTIMER_System_ClockMode;
*  config->clockSelect = kSCTIMER_Clock_On_Rise_Input_0;
*  config->enableBidirection_l = false;
*  config->enableBidirection_h = false;
*  config->prescale_l = 0U;
*  config->prescale_h = 0U;
*  config->outInitState = 0U;
*  config->inputsync  = 0xFU;
*
```

Parameters

config	Pointer to the user configuration structure.

41.9.4 status_t SCTIMER_SetupPwm (SCT_Type * *base*, const sctimer_pwm_signal_param_t * *pwmParams*, sctimer_pwm_mode_t *mode*, uint32_t *pwmFreq_Hz*, uint32_t *srcClock_Hz*, uint32_t * *event*)

Call this function to configure the PWM signal period, mode, duty cycle, and edge. This function will create 2 events; one of the events will trigger on match with the pulse value and the other will trigger when the counter matches the PWM period. The PWM period event is also used as a limit event to reset the counter or change direction. Both events are enabled for the same state. The state number can be retrieved by calling the function SCTIMER_GetCurrentStateNumber(). The counter is set to operate as one 32-bit counter (unify bit is set to 1). The counter operates in bi-directional mode when generating a center-aligned PWM.

Note

When setting PWM output from multiple output pins, they all should use the same PWM mode i.e all PWM's should be either edge-aligned or center-aligned. When using this API, the PWM signal frequency of all the initialized channels must be the same. Otherwise all the initialized channels' PWM signal frequency is equal to the last call to the API's pwmFreq_Hz.

Parameters

base	SCTimer peripheral base address
pwmParams	PWM parameters to configure the output
mode	PWM operation mode, options available in enumeration sctimer_pwm_mode_t
pwmFreq_Hz	PWM signal frequency in Hz
srcClock_Hz	SCTimer counter clock in Hz
event	Pointer to a variable where the PWM period event number is stored

Returns

kStatus_Success on success kStatus_Fail If we have hit the limit in terms of number of events created or if an incorrect PWM dutycylce is passed in.

41.9.5 void SCTIMER_UpdatePwmDutycycle (SCT_Type * *base*, sctimer_out_t *output*, uint8_t *dutyCyclePercent*, uint32_t *event*)

Before calling this function, the counter is set to operate as one 32-bit counter (unify bit is set to 1).

Parameters

base	SCTimer peripheral base address
output	The output to configure
dutyCycle-Percent	New PWM pulse width; the value should be between 1 to 100
event	Event number associated with this PWM signal. This was returned to the user by the function SCTIMER_SetupPwm().

41.9.6 static void SCTIMER_EnableInterrupts (SCT_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SCTimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration sctimer-_interrupt_enable_t

41.9.7 static void SCTIMER_DisableInterrupts (SCT_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	SCTimer peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration sctimer-_interrupt_enable_t

41.9.8 static uint32_t SCTIMER_GetEnabledInterrupts (SCT_Type ∗ *base*) [inline], [static]

Parameters

base	SCTimer peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration sctimer_interrupt_enable_t

41.9.9 static uint32_t SCTIMER_GetStatusFlags (SCT_Type * *base*) [inline], [static]

Parameters

base	SCTimer peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration sctimer_status_flags_t

41.9.10 static void SCTIMER_ClearStatusFlags (SCT_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	SCTimer peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration sctimer_status_flags_t

41.9.11 static void SCTIMER_StartTimer (SCT_Type * *base*, uint32_t *countertoStart*) [inline], [static]

Note

In 16-bit mode, we can enable both Counter_L and Counter_H, In 32-bit mode, we only can select Counter_U.

Parameters

base	SCTimer peripheral base address
countertoStart	The SCTimer counters to enable. This is a logical OR of members of the enumeration sctimer_counter_t.

41.9.12 static void SCTIMER_StopTimer (SCT_Type * *base*, uint32_t *countertoStop*) [inline], [static]

Parameters

base	SCTimer peripheral base address
countertoStop	The SCTimer counters to stop. This is a logical OR of members of the enumeration sctimer_counter_t.

41.9.13 status_t SCTIMER_CreateAndScheduleEvent (SCT_Type * *base*, sctimer_event_t *howToMonitor*, uint32_t *matchValue*, uint32_t *whichIO*, sctimer_counter_t *whichCounter*, uint32_t * *event*)

This function will configure an event using the options provided by the user. If the event type uses the counter match, then the function will set the user provided match value into a match register and put this match register number into the event control register. The event is enabled for the current state and the event number is increased by one at the end. The function returns the event number; this event number can be used to configure actions to be done when this event is triggered.

Parameters

base	SCTimer peripheral base address
howToMonitor	Event type; options are available in the enumeration sctimer_interrupt_enable_t
matchValue	The match value that will be programmed to a match register
whichIO	The input or output that will be involved in event triggering. This field is ignored if the event type is "match only"

whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Pointer to a variable where the new event number is stored

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of events created or if we have reached the limit in terms of number of match registers

41.9.14 void SCTIMER_ScheduleEvent (SCT_Type ∗ *base,* uint32_t *event*)

This function will allow the event passed in to trigger in the current state. The event must be created earlier by either calling the function SCTIMER_SetupPwm() or function SCTIMER_CreateAndScheduleEvent() .

Parameters

base	SCTimer peripheral base address
event	Event number to enable in the current state

41.9.15 status_t SCTIMER_IncreaseState (SCT_Type ∗ *base*)

All future events created by calling the function SCTIMER_ScheduleEvent() will be enabled in this new state.

Parameters

base	SCTimer peripheral base address

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of states used

41.9.16 uint32_t SCTIMER_GetCurrentState (SCT_Type ∗ *base*)

User can use this to set the next state by calling the function SCTIMER_SetupNextStateAction().

Parameters

base	SCTimer peripheral base address

Returns

The current state

41.9.17 static void SCTIMER_SetCounterState (SCT_Type * *base*, sctimer_counter_t *whichCounter*, uint32_t *state*) [inline], [static]

The function is to set the state variable bit field of STATE register. Writing to the STATE_L, STATE_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
state	The counter current state number (only support range from 0∼31).

41.9.18 static uint16_t SCTIMER_GetCounterState (SCT_Type * *base*, sctimer_counter_t *whichCounter*) [inline], [static]

The function is to get the state variable bit field of STATE register.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The the counter current state value.

41.9.19 status_t SCTIMER_SetupCaptureAction (SCT_Type * *base*, sctimer_counter_t *whichCounter*, uint32_t * *captureRegister*, uint32_t *event*)

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
captureRegister	Pointer to a variable where the capture register number will be returned. User can read the captured value from this register when the specified event is triggered.
event	Event number that will trigger the capture

Returns

kStatus_Success on success kStatus_Error if we have hit the limit in terms of number of match/capture registers available

41.9.20 void SCTIMER_SetCallback (SCT_Type * *base,* sctimer_event_callback_t *callback,* uint32_t *event*)

If the interrupt for the event is enabled by the user, then a callback can be registered which will be invoked when the event is triggered

Parameters

base	SCTimer peripheral base address
event	Event number that will trigger the interrupt
callback	Function to invoke when the event is triggered

41.9.21 static void SCTIMER_SetupStateLdMethodAction (SCT_Type * *base,* uint32_t *event,* bool *fgLoad*) `[inline], [static]`

Change the load method of transition, it will be triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address

event	Event number that will change the method to trigger the state transition
fgLoad	The method to load highest-numbered event occurring for that state to the STATE register. • true: Load the STATEV value to STATE when the event occurs to be the next state. • false: Add the STATEV value to STATE when the event occurs to be the next state.

41.9.22 static void SCTIMER_SetupNextStateActionwithLdMethod (SCT_Type * *base*, uint32_t *nextState*, uint32_t *event*, bool *fgLoad*) [inline], [static]

This transition will be triggered by the event number that is passed in by the user, the method decide how to load the highest-numbered event occurring for that state to the STATE register.

Parameters

base	SCTimer peripheral base address
nextState	The next state SCTimer will transition to
event	Event number that will trigger the state transition
fgLoad	The method to load the highest-numbered event occurring for that state to the STATE register. • true: Load the STATEV value to STATE when the event occurs to be the next state. • false: Add the STATEV value to STATE when the event occurs to be the next state.

41.9.23 static void SCTIMER_SetupNextStateAction (SCT_Type * *base*, uint32_t *nextState*, uint32_t *event*) [inline], [static]

Deprecated Do not use this function. It has been superceded by SCTIMER_SetupNextStateActionwith-LdMethod

This transition will be triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
nextState	The next state SCTimer will transition to
event	Event number that will trigger the state transition

41.9.24 static void SCTIMER_SetupEventActiveDirection (SCT_Type * *base*, sctimer_event_active_direction_t *activeDirection*, uint32_t *event*) [inline], [static]

Parameters

base	SCTimer peripheral base address
activeDirection	Event generation active direction, see sctimer_event_active_direction_t.
event	Event number that need setup the active direction.

41.9.25 static void SCTIMER_SetupOutputSetAction (SCT_Type * *base*, uint32_t *whichIO*, uint32_t *event*) [inline], [static]

This output will be set when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichIO	The output to set
event	Event number that will trigger the output change

41.9.26 static void SCTIMER_SetupOutputClearAction (SCT_Type * *base*, uint32_t *whichIO*, uint32_t *event*) [inline], [static]

This output will be cleared when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichIO	The output to clear
event	Event number that will trigger the output change

41.9.27 void SCTIMER_SetupOutputToggleAction (SCT_Type * *base,* uint32_t *whichIO,* uint32_t *event*)

This change in the output level is triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
whichIO	The output to toggle
event	Event number that will trigger the output change

41.9.28 static void SCTIMER_SetupCounterLimitAction (SCT_Type * *base,* sctimer_counter_t *whichCounter,* uint32_t *event*) [inline], [static]

The counter is limited when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Event number that will trigger the counter to be limited

41.9.29 static void SCTIMER_SetupCounterStopAction (SCT_Type * *base,* sctimer_counter_t *whichCounter,* uint32_t *event*) [inline], [static]

The counter is stopped when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Event number that will trigger the counter to be stopped

41.9.30 static void SCTIMER_SetupCounterStartAction (SCT_Type * *base*, sctimer_counter_t *whichCounter*, uint32_t *event*) [inline], [static]

The counter will re-start when the event number that is passed in by the user is triggered.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Event number that will trigger the counter to re-start

41.9.31 static void SCTIMER_SetupCounterHaltAction (SCT_Type * *base*, sctimer_counter_t *whichCounter*, uint32_t *event*) [inline], [static]

The counter is disabled (halted) when the event number that is passed in by the user is triggered. When the counter is halted, all further events are disabled. The HALT condition can only be removed by calling the SCTIMER_StartTimer() function.

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
event	Event number that will trigger the counter to be halted

41.9.32 static void SCTIMER_SetupDmaTriggerAction (SCT_Type * *base*, uint32_t *dmaNumber*, uint32_t *event*) [inline], [static]

DMA request will be triggered by the event number that is passed in by the user.

Parameters

base	SCTimer peripheral base address
dmaNumber	The DMA request to generate
event	Event number that will trigger the DMA request

41.9.33 static void SCTIMER_SetCOUNTValue (SCT_Type * *base,* sctimer_counter_t *whichCounter,* uint32_t *value*) [inline], [static]

The function is to set the value of Count register, Writing to the COUNT_L, COUNT_H, or unified register is only allowed when the corresponding counter is halted (HALT bits are set to 1 in the CTRL register).

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.
value	the counter value update to the COUNT register.

41.9.34 static uint32_t SCTIMER_GetCOUNTValue (SCT_Type * *base,* sctimer_counter_t *whichCounter*) [inline], [static]

The function is to read the value of Count register, software can read the counter registers at any time..

Parameters

base	SCTimer peripheral base address
whichCounter	SCTimer counter to use. In 16-bit mode, we can select Counter_L and Counter_H, In 32-bit mode, we can select Counter_U.

Returns

The value of counter selected.

41.9.35 static void SCTIMER_SetEventInState (SCT_Type * *base,* uint32_t *event,* uint32_t *state*) [inline], [static]

Parameters

base	SCTimer peripheral base address
event	The EV_STATE register be set.
state	The state value in which the event is enabled to occur.

41.9.36 static void SCTIMER_ClearEventInState (SCT_Type * *base,* uint32_t *event,* uint32_t *state*) [inline], [static]

Parameters

base	SCTimer peripheral base address
event	The EV_STATE register be clear.
state	The state value in which the event is disabled to occur.

41.9.37 static bool SCTIMER_GetEventInState (SCT_Type * *base,* uint32_t *event,* uint32_t *state*) [inline], [static]

Note

This function is to check whether the event is enabled in a specific state.

Parameters

base	SCTimer peripheral base address
event	The EV_STATE register be read.
state	The state value.

Returns

The the state mask bit field of EV_STATE register.
- true: The event is enable in state.
- false: The event is disable in state.

41.9.38 void SCTIMER_EventHandleIRQ (SCT_Type * *base*)

Parameters

base	SCTimer peripheral base address.

Chapter 42
SEMA42: Hardware Semaphores Driver

42.1 Overview

The MCUXpresso SDK provides a driver for the SEMA42 module of MCUXpresso SDK devices.

The SEMA42 driver is used for multicore platforms. Before using the SEMA42, call the SEMA42_Init() function to initialize the module. Note that this function only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either the SEMA42_ResetGate() or SEMA42_ResetAllGates() functions. The function SEMA42_Deinit() deinitializes the SEMA42.

The SEMA42 provides two functions to lock the SEMA42 gate. The function SEMA42_TryLock() tries to lock the gate. If the gate has been locked by another processor, this function returns an error immediately. The function SEMA42_Lock() is a blocking method, which waits until the gate is free and locks it.

The SEMA42_Unlock() unlocks the SEMA42 gate. The gate can only be unlocked by the processor which locked it. If the gate is not locked by the current processor, this function takes no effect. The function SEMA42_GetGateStatus() returns a status whether the gate is unlocked and which processor locks the gate.

The SEMA42 gate can be reset to unlock forcefully. The function SEMA42_ResetGate() resets a specific gate. The function SEMA42_ResetAllGates() resets all gates.

42.2 Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/sema42

Macros

- #define SEMA42_GATE_NUM_RESET_ALL (64U)
  The number to reset all SEMA42 gates.
- #define SEMA42_GATEn(base, n) (((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 {

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.

42.3 Macro Definition Documentation

42.3.1 #define SEMA42_GATE_NUM_RESET_ALL (64U)

42.3.2 #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 ^ 3 = 3 1 ^ 3 = 2 2 ^ 3 = 1 3 ^ 3 = 0 4 ^ 3 = 7 5 ^ 3 = 6 6 ^ 3 = 5 7 ^ 3 = 4 ...

42.4 Enumeration Type Documentation

42.4.1 anonymous enum

Enumerator

kStatus_SEMA42_Busy SEMA42 gate has been locked by other processor.
kStatus_SEMA42_Reseting SEMA42 gate reseting is ongoing.

42.4.2 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.

42.5 Function Documentation

42.5.1 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

base	SEMA42 peripheral base address.

42.5.2 void SEMA42_Deinit (SEMA42_Type ∗ *base*)

This function de-initializes the SEMA42 module. It only disables the clock.

Parameters

base	SEMA42 peripheral base address.

42.5.3 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

base	SEMA42 peripheral base address.
gateNum	Gate number to lock.
procNum	Current processor number.

Return values

kStatus_Success	Lock the sema42 gate successfully.
kStatus_SEMA42_Busy	Sema42 gate has been locked by another processor.

42.5.4 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.

Parameters

base	SEMA42 peripheral base address.
gateNum	Gate number to lock.
procNum	Current processor number.

42.5.5 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

base	SEMA42 peripheral base address.
gateNum	Gate number to unlock.

42.5.6 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

base	SEMA42 peripheral base address.
gateNum	Gate number.

Returns

status Current status.

42.5.7 status_t SEMA42_ResetGate (SEMA42_Type * *base,* uint8_t *gateNum*)

This function resets a SEMA42 gate to an unlocked status.

Parameters

base	SEMA42 peripheral base address.
gateNum	Gate number.

Return values

kStatus_Success	SEMA42 gate is reset successfully.
kStatus_SEMA42_-Reseting	Some other reset process is ongoing.

42.5.8 static status_t SEMA42_ResetAllGates (SEMA42_Type * *base*) [inline], [static]

This function resets all SEMA42 gate to an unlocked status.

Parameters

base	SEMA42 peripheral base address.

Return values

kStatus_Success	SEMA42 is reset successfully.
kStatus_SEMA42_-Reseting	Some other reset process is ongoing.

Chapter 43
SMARTDMA: SMART DMA Driver

43.1 Overview

The MCUXpresso SDK provides a peripheral driver for the SMART DMA module of MCUXpresso SDK devices.

Data Structures

- struct smartdma_flexio_mculcd_param_t
  Parameter for FlexIO MCULCD except kSMARTDMA_FlexIO_DMA_ONELANE. More...
- struct smartdma_flexio_onelane_mculcd_param_t
  Parameter for kSMARTDMA_FlexIO_DMA_ONELANE. More...
- struct smartdma_dsi_param_t
  Parameter for MIPI DSI. More...
- struct smartdma_rgb565_rgb888_param_t
  Parameter for RGB565To888. More...

Macros

- #define SMARTDMA_DISPLAY_MEM_ADDR 0x24100000
  The s_smartdmaDisplayFirmware firmware memory address.
- #define SMARTDMA_DISPLAY_FIRMWARE_SIZE (s_smartdmaDisplayFirmwareSize)
  Size of s_smartdmaDisplayFirmware.
- #define s_smartdmaFlexioMcuLcdFirmware s_smartdmaDisplayFirmware
  Compatibility redefinition.

Typedefs

- typedef void(∗ smartdma_callback_t)(void ∗param)
  Callback function prototype for the smartdma driver.

Enumerations

- enum _smartdma_flexio_mculcd_api { ,
  kSMARTDMA_FlexIO_DMA_Reverse,
  kSMARTDMA_RGB565To888,
  kSMARTDMA_FlexIO_DMA_RGB565To888,
  kSMARTDMA_FlexIO_DMA_ARGB2RGB,
  kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap,
  kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap_Reverse,
  kSMARTDMA_MIPI_RGB565_DMA,
  kSMARTDMA_MIPI_RGB888_DMA,
  kSMARTDMA_MIPI_XRGB2RGB_DMA,
  kSMARTDMA_MIPI_RGB565_R180_DMA,
  kSMARTDMA_MIPI_RGB888_R180_DMA,
  kSMARTDMA_FlexIO_DMA_ONELANE }
  The API index when using s_smartdmaFlexioMcuLcdFirmware.

Functions

- void SMARTDMA_Init (uint32_t apiMemAddr, const void *firmware, uint32_t firmwareSizeByte)
  Initialize the SMARTDMA.
- void SMARTDMA_InitWithoutFirmware (void)
  Initialize the SMARTDMA.
- void SMARTDMA_InstallFirmware (uint32_t apiMemAddr, const void *firmware, uint32_t firmwareSizeByte)
  Install the firmware.
- void SMARTDMA_InstallCallback (smartdma_callback_t callback, void *param)
  Install the complete callback function.
- void SMARTDMA_Boot (uint32_t apiIndex, void *pParam, uint8_t mask)
  Boot the SMARTDMA to run program.
- void SMARTDMA_Deinit (void)
  Deinitialize the SMARTDMA.
- void SMARTDMA_Reset (void)
  Reset the SMARTDMA.
- void SMARTDMA_HandleIRQ (void)
  SMARTDMA IRQ.

Variables

- const uint8_t s_smartdmaDisplayFirmware []
  The firmware used for display.
- const uint32_t s_smartdmaDisplayFirmwareSize
  Size of s_smartdmaDisplayFirmware.

Driver version

- #define FSL_SMARTDMA_DRIVER_VERSION (MAKE_VERSION(2, 6, 3))
  SMARTDMA driver version.

43.2 Data Structure Documentation

43.2.1 struct smartdma_flexio_mculcd_param_t

43.2.2 struct smartdma_flexio_onelane_mculcd_param_t

43.2.3 struct smartdma_dsi_param_t

Data Fields

- const uint8_t ∗ p_buffer
  Pointer to the buffer to send.
- uint32_t buffersize
  Buffer size in byte.
- uint32_t ∗ smartdma_stack
  Stack used by SMARTDMA.
- uint32_t disablePixelByteSwap
  If set to 1, the pixels are filled to MIPI DSI FIFO directly.

Field Documentation

(1) const uint8_t∗ smartdma_dsi_param_t::p_buffer

(2) uint32_t smartdma_dsi_param_t::buffersize

(3) uint32_t∗ smartdma_dsi_param_t::smartdma_stack

(4) uint32_t smartdma_dsi_param_t::disablePixelByteSwap

If set to 0, the pixel bytes are swapped then filled to MIPI DSI FIFO. For example, when set to 0 and frame buffer pixel format is RGB565: LSB MSB B0 B1 B2 B3 B4 G0 G1 G2 | G3 G4 G5 R0 R1 R2 R3 R4 Then the pixel filled to DSI FIFO is: LSB MSB G3 G4 G5 R0 R1 R2 R3 R4 | B0 B1 B2 B3 B4 G0 G1 G2

43.2.4 struct smartdma_rgb565_rgb888_param_t

43.3 Macro Definition Documentation

43.3.1 #define SMARTDMA_DISPLAY_MEM_ADDR 0x24100000

43.3.2 #define s_smartdmaFlexioMcuLcdFirmware s_smartdmaDisplayFirmware

43.4 Typedef Documentation

43.4.1 typedef void(∗ smartdma_callback_t)(void ∗param)

43.5 Enumeration Type Documentation

43.5.1 enum _smartdma_flexio_mculcd_api

Enumerator

kSMARTDMA_FlexIO_DMA_Reverse Send data to FlexIO with reverse order.

kSMARTDMA_RGB565To888 Convert RGB565 to RGB888 and save to output memory, use parameter smartdma_rgb565_rgb888_param_t.

kSMARTDMA_FlexIO_DMA_RGB565To888 Convert RGB565 to RGB888 and send to FlexIO, use parameter smartdma_flexio_mculcd_param_t.

kSMARTDMA_FlexIO_DMA_ARGB2RGB Convert ARGB to RGB and send to FlexIO, use parameter smartdma_flexio_mculcd_param_t.

kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap Convert ARGB to RGB, then swap endian, and send to FlexIO, use parameter smartdma_flexio_mculcd_param_t.

kSMARTDMA_FlexIO_DMA_ARGB2RGB_Endian_Swap_Reverse Convert ARGB to RGB, then swap endian and reverse, and send to FlexIO, use parameter smartdma_flexio_mculcd_-param_t.

kSMARTDMA_MIPI_RGB565_DMA Send RGB565 data to MIPI DSI, use parameter smartdma-_dsi_param_t.

kSMARTDMA_MIPI_RGB888_DMA Send RGB888 data to MIPI DSI, use parameter smartdma-_dsi_param_t.

kSMARTDMA_MIPI_XRGB2RGB_DMA Send XRGB8888 data to MIPI DSI, use parameter smartdma_param_t.

kSMARTDMA_MIPI_RGB565_R180_DMA Send RGB565 data to MIPI DSI, Rotate 180, use parameter smartdma_dsi_param_t.

kSMARTDMA_MIPI_RGB888_R180_DMA Send RGB888 data to MIPI DSI, Rotate 180, use parameter smartdma_dsi_param_t.

kSMARTDMA_FlexIO_DMA_ONELANE FlexIO DMA for one SHIFTBUF, Write Data to SHIFTBUF[OFFSET].

43.6 Function Documentation

43.6.1 void SMARTDMA_Init (uint32_t *apiMemAddr,* const void ∗ *firmware,* uint32_t *firmwareSizeByte*)

Parameters

apiMemAddr	The address firmware will be copied to.

firmware	The firmware to use.
firmwareSize-Byte	Size of firmware.

Deprecated Do not use this function. It has been superceded by GPIO_PinWrite. SMARTDMA_Init-WithoutFirmware and SMARTDMA_InstallFirmware.

43.6.2 void SMARTDMA_InitWithoutFirmware (void)

This function is similar with SMARTDMA_Init, the difference is this function does not install the firmware, the firmware could be installed using SMARTDMA_InstallFirmware.

43.6.3 void SMARTDMA_InstallFirmware (uint32_t *apiMemAddr,* const void * *firmware,* uint32_t *firmwareSizeByte*)

Parameters

apiMemAddr	The address firmware will be copied to.
firmware	The firmware to use.
firmwareSize-Byte	Size of firmware.

Note

Only call this function when SMARTDMA is not busy.

43.6.4 void SMARTDMA_InstallCallback (smartdma_callback_t *callback,* void * *param*)

Parameters

callback	The callback called when smartdma program finished.

param	Parameter for the callback.

Note

Only call this function when SMARTDMA is not busy.

43.6.5 void SMARTDMA_Boot (uint32_t *apiIndex,* void ∗ *pParam,* uint8_t *mask*)

Parameters

apiIndex	Index of the API to call.
pParam	Pointer to the parameter.
mask	Value set to SMARTDMA_ARM2SMARTDMA[0:1].

Note

Only call this function when SMARTDMA is not busy.

43.7 Variable Documentation

43.7.1 const uint8_t s_smartdmaDisplayFirmware[]

Chapter 44
TRNG: True Random Number Generator

44.1 Overview

The MCUXpresso SDK provides a peripheral driver for the True Random Number Generator (TRNG) module of MCUXpresso SDK devices.

The True Random Number Generator is a hardware accelerator module that generates a 512-bit entropy as needed by an entropy consuming module or by other post processing functions. A typical entropy consumer is a pseudo random number generator (PRNG) which can be implemented to achieve both true randomness and cryptographic strength random numbers using the TRNG output as its entropy seed. The entropy generated by a TRNG is intended for direct use by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

44.2 TRNG Initialization

1. Define the TRNG user configuration structure. Use TRNG_InitUserConfigDefault() function to set it to default TRNG configuration values.
2. Initialize the TRNG module, call the TRNG_Init() function, and pass the user configuration structure. This function automatically enables the TRNG module and its clock. After that, the TRNG is enabled and the entropy generation starts working.
3. To disable the TRNG module, call the TRNG_Deinit() function.

44.3 Get random data from TRNG

1. TRNG_GetRandomData() function gets random data from the TRNG module.

This example code shows how to initialize and get random data from the TRNG driver.

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

Data Structures

- struct trng_statistical_check_limit_t
  Data structure for definition of statistical check limits. More...
- struct trng_config_t
  Data structure for the TRNG initialization. More...

Enumerations

- enum trng_sample_mode_t {
  kTRNG_SampleModeVonNeumann = 0U,
  kTRNG_SampleModeRaw = 1U,
  kTRNG_SampleModeVonNeumannRaw }
  TRNG sample mode.

- enum trng_clock_mode_t {
  kTRNG_ClockModeRingOscillator = 0U,
  kTRNG_ClockModeSystem = 1U }
  TRNG clock mode.
- enum trng_ring_osc_div_t {
  kTRNG_RingOscDiv0 = 0U,
  kTRNG_RingOscDiv2 = 1U,
  kTRNG_RingOscDiv4 = 2U,
  kTRNG_RingOscDiv8 = 3U }
  TRNG ring oscillator divide.

Functions

- status_t TRNG_GetDefaultConfig (trng_config_t ∗userConfig)
  Initializes the user configuration structure to default values.
- status_t TRNG_Init (TRNG_Type ∗base, const trng_config_t ∗userConfig)
  Initializes the TRNG.
- void TRNG_Deinit (TRNG_Type ∗base)
  Shuts down the TRNG.
- status_t TRNG_GetRandomData (TRNG_Type ∗base, void ∗data, size_t dataSize)
  Gets random data.

Driver version

- #define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 13))
  TRNG driver version 2.0.13.

44.4 Data Structure Documentation

44.4.1 struct trng_statistical_check_limit_t

Used by trng_config_t.

Data Fields

- uint32_t maximum
  Maximum limit.
- uint32_t minimum
  Minimum limit.

Field Documentation

(1) uint32_t trng_statistical_check_limit_t::maximum

(2) uint32_t trng_statistical_check_limit_t::minimum

44.4.2 struct trng_config_t

This structure initializes the TRNG by calling the TRNG_Init() function. It contains all TRNG configurations.

Data Fields

- bool lock
  Disable programmability of TRNG registers.
- trng_clock_mode_t clockMode
  Clock mode used to operate TRNG.
- trng_ring_osc_div_t ringOscDiv
  Ring oscillator divide used by TRNG.
- trng_sample_mode_t sampleMode
  Sample mode of the TRNG ring oscillator.
- uint16_t entropyDelay
  Entropy Delay.
- uint16_t sampleSize
  Sample Size.
- uint16_t sparseBitLimit
  Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated.
- uint8_t retryCount
  Retry count.
- uint8_t longRunMaxLimit
  Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.
- trng_statistical_check_limit_t monobitLimit
  Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.
- trng_statistical_check_limit_t runBit1Limit
  Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.
- trng_statistical_check_limit_t runBit2Limit
  Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.
- trng_statistical_check_limit_t runBit3Limit
  Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.
- trng_statistical_check_limit_t runBit4Limit
  Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.
- trng_statistical_check_limit_t runBit5Limit
  Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.
- trng_statistical_check_limit_t runBit6PlusLimit
  Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.
- trng_statistical_check_limit_t pokerLimit

Maximum and minimum limits for statistical check of "Poker Test".

- trng_statistical_check_limit_t frequencyCountLimit
  Maximum and minimum limits for statistical check of entropy sample frequency count.

Field Documentation

(1) bool trng_config_t::lock

(2) trng_clock_mode_t trng_config_t::clockMode

(3) trng_ring_osc_div_t trng_config_t::ringOscDiv

(4) trng_sample_mode_t trng_config_t::sampleMode

(5) uint16_t trng_config_t::entropyDelay

Defines the length (in system clocks) of each Entropy sample taken.

(6) uint16_t trng_config_t::sampleSize

Defines the total number of Entropy samples that will be taken during Entropy generation.

(7) uint16_t trng_config_t::sparseBitLimit

This limit is used only for during von Neumann sampling (enabled by TRNG_HAL_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a long period of time, it indicates that there is insufficient Entropy.

(8) uint8_t trng_config_t::retryCount

It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

(9) uint8_t trng_config_t::longRunMaxLimit

(10) trng_statistical_check_limit_t trng_config_t::monobitLimit

(11) trng_statistical_check_limit_t trng_config_t::runBit1Limit

(12) trng_statistical_check_limit_t trng_config_t::runBit2Limit

(13) trng_statistical_check_limit_t trng_config_t::runBit3Limit

(14) trng_statistical_check_limit_t trng_config_t::runBit4Limit

(15) trng_statistical_check_limit_t trng_config_t::runBit5Limit

(16) trng_statistical_check_limit_t trng_config_t::runBit6PlusLimit

(17) trng_statistical_check_limit_t trng_config_t::pokerLimit

(18) trng_statistical_check_limit_t trng_config_t::frequencyCountLimit

44.5 Macro Definition Documentation

44.5.1 #define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 13))

Current version: 2.0.13

Change log:

- version 2.0.13
  - After deepsleep it might return error, added clearing bits in TRNG_GetRandomData() and generating new entropy.
  - Modified reloading entropy in TRNG_GetRandomData(), for some data length it doesn't reloading entropy correctly.
- version 2.0.12
  - For KW34A4_SERIES, KW35A4_SERIES, KW36A4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.11
  - Add clearing pending errors in TRNG_Init().
- version 2.0.10
  - Fixed doxygen issues.
- version 2.0.9
  - Fix HIS_CCM metrics issues.
- version 2.0.8
  - For K32L2A41A_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv4.
- version 2.0.7
  - Fix MISRA 2004 issue rule 12.5.
- version 2.0.6
  - For KW35Z4_SERIES set TRNG_USER_CONFIG_DEFAULT_OSC_DIV to kTRNG_RingOscDiv8.
- version 2.0.5
  - Add possibility to define default TRNG configuration by device specific preprocessor macros for FRQMIN, FRQMAX and OSCDIV.
- version 2.0.4
  - Fix MISRA-2012 issues.
- Version 2.0.3
  - update TRNG_Init to restart entropy generation
- Version 2.0.2
  - fix MISRA issues
- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

44.6 Enumeration Type Documentation

44.6.1 enum trng_sample_mode_t

Used by trng_config_t.

Enumerator

kTRNG_SampleModeVonNeumann Use von Neumann data in both Entropy shifter and Statistical Checker.
kTRNG_SampleModeRaw Use raw data into both Entropy shifter and Statistical Checker.
kTRNG_SampleModeVonNeumannRaw Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

44.6.2 enum trng_clock_mode_t

Used by trng_config_t.

Enumerator

kTRNG_ClockModeRingOscillator Ring oscillator is used to operate the TRNG (default).
kTRNG_ClockModeSystem System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

44.6.3 enum trng_ring_osc_div_t

Used by trng_config_t.

Enumerator

kTRNG_RingOscDiv0 Ring oscillator with no divide.
kTRNG_RingOscDiv2 Ring oscillator divided-by-2.
kTRNG_RingOscDiv4 Ring oscillator divided-by-4.
kTRNG_RingOscDiv8 Ring oscillator divided-by-8.

44.7 Function Documentation

44.7.1 status_t TRNG_GetDefaultConfig (trng_config_t * *userConfig*)

This function initializes the configuration structure to default values. The default values are as follows.

```
*     userConfig->lock = 0;
*     userConfig->clockMode = kTRNG_ClockModeRingOscillator;
*     userConfig->ringOscDiv = kTRNG_RingOscDiv0;  Or  to other kTRNG_RingOscDiv[2|8]
      depending on the platform.
*     userConfig->sampleMode = kTRNG_SampleModeRaw;
*     userConfig->entropyDelay = 3200;
```

```
*     userConfig->sampleSize = 2500;
*     userConfig->sparseBitLimit = TRNG_USER_CONFIG_DEFAULT_SPARSE_BIT_LIMIT;
*     userConfig->retryCount = 63;
*     userConfig->longRunMaxLimit = 34;
*     userConfig->monobitLimit.maximum = 1384;
*     userConfig->monobitLimit.minimum = 1116;
*     userConfig->runBit1Limit.maximum = 405;
*     userConfig->runBit1Limit.minimum = 227;
*     userConfig->runBit2Limit.maximum = 220;
*     userConfig->runBit2Limit.minimum = 98;
*     userConfig->runBit3Limit.maximum = 125;
*     userConfig->runBit3Limit.minimum = 37;
*     userConfig->runBit4Limit.maximum = 75;
*     userConfig->runBit4Limit.minimum = 11;
*     userConfig->runBit5Limit.maximum = 47;
*     userConfig->runBit5Limit.minimum = 1;
*     userConfig->runBit6PlusLimit.maximum = 47;
*     userConfig->runBit6PlusLimit.minimum = 1;
*     userConfig->pokerLimit.maximum = 26912;
*     userConfig->pokerLimit.minimum = 24445;
*     userConfig->frequencyCountLimit.maximum = 25600;
*     userConfig->frequencyCountLimit.minimum = 1600;
*
```

Parameters

userConfig	User configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

44.7.2 status_t TRNG_Init (TRNG_Type ∗ *base,* const trng_config_t ∗ *userConfig*)

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

base	TRNG base address
userConfig	Pointer to the initialization configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

44.7.3 void TRNG_Deinit (TRNG_Type ∗ *base*)

This function shuts down the TRNG.

Parameters

base	TRNG base address.

44.7.4 status_t TRNG_GetRandomData (TRNG_Type * *base,* void * *data,* size_t *dataSize*)

This function gets random data from the TRNG.

Parameters

base	TRNG base address.
data	Pointer address used to store random data.
dataSize	Size of the buffer pointed by the data parameter.

Returns

random data

Chapter 45
USDHC: Ultra Secured Digital Host Controller Driver

45.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Ultra Secured Digital Host Controller (USDHC) module of MCUXpresso SDK/i.MX devices.

45.2 Typical use case

45.2.1 USDHC Operation

Refer to the driver examples codes located at *<SDK_ROOT>/boards/<BOARD>/driver_examples/usdhc*.

Cache maintain capability

The uSDHC host controller is intergrated with ADMA to have better transfer performance, so to maintain data integrity during DMA operations on the platform that has cache, USDHC driver provide a cache maintain functionality by define: FSL_SDK_ENABLE_DRIVER_CACHE_CONTROL = 1 It is suggest that the address of buffer used for read/write is align with cache line size.

Scatter gather transfer capability

The USDHC driver implement scatter gather transfer functionality, so application can submit uncontinuous data buffer in one transfer request by the scatter gather api, to have this feature, USDHC driver has below api USDHC_TransferScatterGatherADMANonBlocking This function suppport scatter gather transfer and cover the functionality of USDHC_TransferNonBlocking also, but if application would like to use the function, please enable function macro firstly, since the scatter gather functionality is disabled by default. #define FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER 1

Please note that once the macro is defined, the USDHC_TransferNonBlocking will be removed automatically.

Data Structures

- struct usdhc_adma2_descriptor_t
  Defines the ADMA2 descriptor structure. More...
- struct usdhc_capability_t
  USDHC capability information. More...
- struct usdhc_boot_config_t
  Data structure to configure the MMC boot feature. More...
- struct usdhc_config_t
  Data structure to initialize the USDHC. More...
- struct usdhc_command_t
  Card command descriptor. More...

- struct usdhc_adma_config_t
  ADMA configuration. More...
- struct usdhc_scatter_gather_data_list_t
  Card scatter gather data list. More...
- struct usdhc_scatter_gather_data_t
  Card scatter gather data descriptor. More...
- struct usdhc_scatter_gather_transfer_t
  usdhc scatter gather transfer. More...
- struct usdhc_data_t
  Card data descriptor. More...
- struct usdhc_transfer_t
  Transfer state. More...
- struct usdhc_transfer_callback_t
  USDHC callback functions. More...
- struct usdhc_handle_t
  USDHC handle. More...
- struct usdhc_host_t
  USDHC host descriptor. More...

Macros

- #define USDHC_MAX_BLOCK_COUNT (USDHC_BLK_ATT_BLKCNT_MASK >> USDHC-_BLK_ATT_BLKCNT_SHIFT)
  Maximum block count can be set one time.
- #define FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER 0U
  USDHC scatter gather feature control macro.
- #define USDHC_ADMA1_ADDRESS_ALIGN (4096U)
  The alignment size for ADDRESS filed in ADMA1's descriptor.
- #define USDHC_ADMA1_LENGTH_ALIGN (4096U)
  The alignment size for LENGTH field in ADMA1's descriptor.
- #define USDHC_ADMA2_ADDRESS_ALIGN (4U)
  The alignment size for ADDRESS field in ADMA2's descriptor.
- #define USDHC_ADMA2_LENGTH_ALIGN (4U)
  The alignment size for LENGTH filed in ADMA2's descriptor.
- #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (12U)
  The bit shift for ADDRESS filed in ADMA1's descriptor.
- #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (0xFFFFFU)
  The bit mask for ADDRESS field in ADMA1's descriptor.
- #define USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (12U)
  The bit shift for LENGTH filed in ADMA1's descriptor.
- #define USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (0xFFFFU)
  The mask for LENGTH field in ADMA1's descriptor.
- #define USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (USDHC_ADMA1_-DESCRIPTOR_LENGTH_MASK + 1U - 4096U)
  The maximum value of LENGTH filed in ADMA1's descriptor.
- #define USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (16U)
  The bit shift for LENGTH field in ADMA2's descriptor.
- #define USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK (0xFFFFU)
  The bit mask for LENGTH field in ADMA2's descriptor.
- #define USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (USDHC_ADMA2_-DESCRIPTOR_LENGTH_MASK - 3U)

The maximum value of LENGTH field in ADMA2's descriptor.

Typedefs

- typedef uint32_t usdhc_adma1_descriptor_t
  Defines the ADMA1 descriptor structure.
- typedef status_t(∗ usdhc_transfer_function_t)(USDHC_Type ∗base, usdhc_transfer_t ∗content)
  USDHC transfer function.

Enumerations

- enum {
  kStatus_USDHC_BusyTransferring = MAKE_STATUS(kStatusGroup_USDHC, 0U),
  kStatus_USDHC_PrepareAdmaDescriptorFailed = MAKE_STATUS(kStatusGroup_USDHC, 1U),
  kStatus_USDHC_SendCommandFailed = MAKE_STATUS(kStatusGroup_USDHC, 2U),
  kStatus_USDHC_TransferDataFailed = MAKE_STATUS(kStatusGroup_USDHC, 3U),
  kStatus_USDHC_DMADataAddrNotAlign = MAKE_STATUS(kStatusGroup_USDHC, 4U),
  kStatus_USDHC_ReTuningRequest = MAKE_STATUS(kStatusGroup_USDHC, 5U),
  kStatus_USDHC_TuningError = MAKE_STATUS(kStatusGroup_USDHC, 6U),
  kStatus_USDHC_NotSupport = MAKE_STATUS(kStatusGroup_USDHC, 7U),
  kStatus_USDHC_TransferDataComplete = MAKE_STATUS(kStatusGroup_USDHC, 8U),
  kStatus_USDHC_SendCommandSuccess = MAKE_STATUS(kStatusGroup_USDHC, 9U),
  kStatus_USDHC_TransferDMAComplete = MAKE_STATUS(kStatusGroup_USDHC, 10U) }
  Enum _usdhc_status.
- enum {
  kUSDHC_SupportAdmaFlag = USDHC_HOST_CTRL_CAP_ADMAS_MASK,
  kUSDHC_SupportHighSpeedFlag = USDHC_HOST_CTRL_CAP_HSS_MASK,
  kUSDHC_SupportDmaFlag = USDHC_HOST_CTRL_CAP_DMAS_MASK,
  kUSDHC_SupportSuspendResumeFlag = USDHC_HOST_CTRL_CAP_SRS_MASK,
  kUSDHC_SupportV330Flag = USDHC_HOST_CTRL_CAP_VS33_MASK,
  kUSDHC_SupportV300Flag = USDHC_HOST_CTRL_CAP_VS30_MASK,
  kUSDHC_SupportV180Flag = USDHC_HOST_CTRL_CAP_VS18_MASK,
  kUSDHC_Support4BitFlag = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 0U),
  kUSDHC_Support8BitFlag = (USDHC_HOST_CTRL_CAP_MBL_SHIFT << 1U),
  kUSDHC_SupportDDR50Flag = USDHC_HOST_CTRL_CAP_DDR50_SUPPORT_MASK,
  kUSDHC_SupportSDR104Flag = USDHC_HOST_CTRL_CAP_SDR104_SUPPORT_MASK,
  kUSDHC_SupportSDR50Flag = USDHC_HOST_CTRL_CAP_SDR50_SUPPORT_MASK }
  Enum _usdhc_capability_flag.
- enum {
  kUSDHC_WakeupEventOnCardInt = USDHC_PROT_CTRL_WECINT_MASK,
  kUSDHC_WakeupEventOnCardInsert = USDHC_PROT_CTRL_WECINS_MASK,
  kUSDHC_WakeupEventOnCardRemove = USDHC_PROT_CTRL_WECRM_MASK,
  kUSDHC_WakeupEventsAll }
  Enum _usdhc_wakeup_event.
- enum {

kUSDHC_ResetAll = USDHC_SYS_CTRL_RSTA_MASK,
kUSDHC_ResetCommand = USDHC_SYS_CTRL_RSTC_MASK,
kUSDHC_ResetData = USDHC_SYS_CTRL_RSTD_MASK,
kUSDHC_ResetTuning = USDHC_SYS_CTRL_RSTT_MASK,
kUSDHC_ResetsAll = (kUSDHC_ResetAll | kUSDHC_ResetCommand | kUSDHC_ResetData | kUSDHC_ResetTuning) }

Enum _usdhc_reset.

- enum {
  kUSDHC_EnableDmaFlag = USDHC_MIX_CTRL_DMAEN_MASK,
  kUSDHC_CommandTypeSuspendFlag = USDHC_CMD_XFR_TYP_CMDTYP(1U),
  kUSDHC_CommandTypeResumeFlag = USDHC_CMD_XFR_TYP_CMDTYP(2U),
  kUSDHC_CommandTypeAbortFlag = USDHC_CMD_XFR_TYP_CMDTYP(3U),
  kUSDHC_EnableBlockCountFlag = USDHC_MIX_CTRL_BCEN_MASK,
  kUSDHC_EnableAutoCommand12Flag = USDHC_MIX_CTRL_AC12EN_MASK,
  kUSDHC_DataReadFlag = USDHC_MIX_CTRL_DTDSEL_MASK,
  kUSDHC_MultipleBlockFlag = USDHC_MIX_CTRL_MSBSEL_MASK,
  kUSDHC_EnableAutoCommand23Flag = USDHC_MIX_CTRL_AC23EN_MASK,
  kUSDHC_ResponseLength136Flag = USDHC_CMD_XFR_TYP_RSPTYP(1U),
  kUSDHC_ResponseLength48Flag = USDHC_CMD_XFR_TYP_RSPTYP(2U),
  kUSDHC_ResponseLength48BusyFlag = USDHC_CMD_XFR_TYP_RSPTYP(3U),
  kUSDHC_EnableCrcCheckFlag = USDHC_CMD_XFR_TYP_CCCEN_MASK,
  kUSDHC_EnableIndexCheckFlag = USDHC_CMD_XFR_TYP_CICEN_MASK,
  kUSDHC_DataPresentFlag = USDHC_CMD_XFR_TYP_DPSEL_MASK }

  Enum _usdhc_transfer_flag.

- enum {
  kUSDHC_CommandInhibitFlag = USDHC_PRES_STATE_CIHB_MASK,
  kUSDHC_DataInhibitFlag = USDHC_PRES_STATE_CDIHB_MASK,
  kUSDHC_DataLineActiveFlag = USDHC_PRES_STATE_DLA_MASK,
  kUSDHC_SdClockStableFlag = USDHC_PRES_STATE_SDSTB_MASK,
  kUSDHC_WriteTransferActiveFlag = USDHC_PRES_STATE_WTA_MASK,
  kUSDHC_ReadTransferActiveFlag = USDHC_PRES_STATE_RTA_MASK,
  kUSDHC_BufferWriteEnableFlag = USDHC_PRES_STATE_BWEN_MASK,
  kUSDHC_BufferReadEnableFlag = USDHC_PRES_STATE_BREN_MASK,
  kUSDHC_ReTuningRequestFlag = USDHC_PRES_STATE_RTR_MASK,
  kUSDHC_DelaySettingFinishedFlag = USDHC_PRES_STATE_TSCD_MASK,
  kUSDHC_CardInsertedFlag = USDHC_PRES_STATE_CINST_MASK,
  kUSDHC_CommandLineLevelFlag = USDHC_PRES_STATE_CLSL_MASK,
  kUSDHC_Data0LineLevelFlag = 1U << USDHC_PRES_STATE_DLSL_SHIFT,
  kUSDHC_Data1LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 1U),
  kUSDHC_Data2LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 2U),
  kUSDHC_Data3LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 3U),
  kUSDHC_Data4LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 4U),
  kUSDHC_Data5LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 5U),
  kUSDHC_Data6LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSL_SHIFT + 6U),
  kUSDHC_Data7LineLevelFlag = (int)(1U << (USDHC_PRES_STATE_DLSL_SHIFT + 7U)) }

Enum _usdhc_present_status_flag.

- enum {
  kUSDHC_CommandCompleteFlag = USDHC_INT_STATUS_CC_MASK,
  kUSDHC_DataCompleteFlag = USDHC_INT_STATUS_TC_MASK,
  kUSDHC_BlockGapEventFlag = USDHC_INT_STATUS_BGE_MASK,
  kUSDHC_DmaCompleteFlag = USDHC_INT_STATUS_DINT_MASK,
  kUSDHC_BufferWriteReadyFlag = USDHC_INT_STATUS_BWR_MASK,
  kUSDHC_BufferReadReadyFlag = USDHC_INT_STATUS_BRR_MASK,
  kUSDHC_CardInsertionFlag = USDHC_INT_STATUS_CINS_MASK,
  kUSDHC_CardRemovalFlag = USDHC_INT_STATUS_CRM_MASK,
  kUSDHC_CardInterruptFlag = USDHC_INT_STATUS_CINT_MASK,
  kUSDHC_ReTuningEventFlag = USDHC_INT_STATUS_RTE_MASK,
  kUSDHC_TuningPassFlag = USDHC_INT_STATUS_TP_MASK,
  kUSDHC_TuningErrorFlag = USDHC_INT_STATUS_TNE_MASK,
  kUSDHC_CommandTimeoutFlag = USDHC_INT_STATUS_CTOE_MASK,
  kUSDHC_CommandCrcErrorFlag = USDHC_INT_STATUS_CCE_MASK,
  kUSDHC_CommandEndBitErrorFlag = USDHC_INT_STATUS_CEBE_MASK,
  kUSDHC_CommandIndexErrorFlag = USDHC_INT_STATUS_CIE_MASK,
  kUSDHC_DataTimeoutFlag = USDHC_INT_STATUS_DTOE_MASK,
  kUSDHC_DataCrcErrorFlag = USDHC_INT_STATUS_DCE_MASK,
  kUSDHC_DataEndBitErrorFlag = USDHC_INT_STATUS_DEBE_MASK,
  kUSDHC_AutoCommand12ErrorFlag = USDHC_INT_STATUS_AC12E_MASK,
  kUSDHC_DmaErrorFlag = USDHC_INT_STATUS_DMAE_MASK,
  kUSDHC_CommandErrorFlag,
  kUSDHC_DataErrorFlag,
  kUSDHC_ErrorFlag = (kUSDHC_CommandErrorFlag | kUSDHC_DataErrorFlag | kUSDHC_-DmaErrorFlag),
  kUSDHC_DataFlag,
  kUSDHC_DataDMAFlag = (kUSDHC_DataCompleteFlag | kUSDHC_DataErrorFlag | kUSDHC-_DmaErrorFlag),
  kUSDHC_CommandFlag = (kUSDHC_CommandErrorFlag | kUSDHC_CommandCompleteFlag),
  kUSDHC_CardDetectFlag = (kUSDHC_CardInsertionFlag | kUSDHC_CardRemovalFlag),
  kUSDHC_SDR104TuningFlag = (kUSDHC_TuningErrorFlag | kUSDHC_TuningPassFlag | kUS-DHC_ReTuningEventFlag),
  kUSDHC_AllInterruptFlags }

  Enum _usdhc_interrupt_status_flag.

- enum {
  kUSDHC_AutoCommand12NotExecutedFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12-NE_MASK,
  kUSDHC_AutoCommand12TimeoutFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12TOE-_MASK,
  kUSDHC_AutoCommand12EndBitErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12-EBE_MASK,
  kUSDHC_AutoCommand12CrcErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12CE_-

MASK,
kUSDHC_AutoCommand12IndexErrorFlag = USDHC_AUTOCMD12_ERR_STATUS_AC12IE-_MASK,
kUSDHC_AutoCommand12NotIssuedFlag = USDHC_AUTOCMD12_ERR_STATUS_CNIBA-C12E_MASK }
Enum _usdhc_auto_command12_error_status_flag.

- enum {
kUSDHC_ExecuteTuning = USDHC_AUTOCMD12_ERR_STATUS_EXECUTE_TUNING_M-ASK,
kUSDHC_TuningSampleClockSel }
Enum _usdhc_standard_tuning.
- enum {
kUSDHC_AdmaLenghMismatchFlag = USDHC_ADMA_ERR_STATUS_ADMALME_MASK,
kUSDHC_AdmaDescriptorErrorFlag = USDHC_ADMA_ERR_STATUS_ADMADCE_MASK }
Enum _usdhc_adma_error_status_flag.
- enum {
kUSDHC_AdmaErrorStateStopDma = 0x00U,
kUSDHC_AdmaErrorStateFetchDescriptor = 0x01U,
kUSDHC_AdmaErrorStateChangeAddress = 0x02U,
kUSDHC_AdmaErrorStateTransferData = 0x03U,
kUSDHC_AdmaErrorStateInvalidLength = 0x04U,
kUSDHC_AdmaErrorStateInvalidDescriptor = 0x08U,
kUSDHC_AdmaErrorState }
Enum _usdhc_adma_error_state.
- enum {
kUSDHC_ForceEventAutoCommand12NotExecuted,
kUSDHC_ForceEventAutoCommand12Timeout = USDHC_FORCE_EVENT_FEVTAC12TOE_-MASK,
kUSDHC_ForceEventAutoCommand12CrcError = USDHC_FORCE_EVENT_FEVTAC12CE_-MASK,
kUSDHC_ForceEventEndBitError = USDHC_FORCE_EVENT_FEVTAC12EBE_MASK,
kUSDHC_ForceEventAutoCommand12IndexError = USDHC_FORCE_EVENT_FEVTAC12IE_-MASK,
kUSDHC_ForceEventAutoCommand12NotIssued = USDHC_FORCE_EVENT_FEVTCNIBA-C12E_MASK,
kUSDHC_ForceEventCommandTimeout = USDHC_FORCE_EVENT_FEVTCTOE_MASK,
kUSDHC_ForceEventCommandCrcError = USDHC_FORCE_EVENT_FEVTCCE_MASK,
kUSDHC_ForceEventCommandEndBitError = USDHC_FORCE_EVENT_FEVTCEBE_MASK,
kUSDHC_ForceEventCommandIndexError = USDHC_FORCE_EVENT_FEVTCIE_MASK,
kUSDHC_ForceEventDataTimeout = USDHC_FORCE_EVENT_FEVTDTOE_MASK,
kUSDHC_ForceEventDataCrcError = USDHC_FORCE_EVENT_FEVTDCE_MASK,
kUSDHC_ForceEventDataEndBitError = USDHC_FORCE_EVENT_FEVTDEBE_MASK,
kUSDHC_ForceEventAutoCommand12Error = USDHC_FORCE_EVENT_FEVTAC12E_MAS-

K,
kUSDHC_ForceEventCardInt = (int)USDHC_FORCE_EVENT_FEVTCINT_MASK,
kUSDHC_ForceEventDmaError = USDHC_FORCE_EVENT_FEVTDMAE_MASK,
kUSDHC_ForceEventTuningError = USDHC_FORCE_EVENT_FEVTTNE_MASK,
kUSDHC_ForceEventsAll }

Enum _usdhc_force_event.

- enum usdhc_transfer_direction_t {
  kUSDHC_TransferDirectionReceive = 1U,
  kUSDHC_TransferDirectionSend = 0U }

  Data transfer direction.
- enum usdhc_data_bus_width_t {
  kUSDHC_DataBusWidth1Bit = 0U,
  kUSDHC_DataBusWidth4Bit = 1U,
  kUSDHC_DataBusWidth8Bit = 2U }

  Data transfer width.
- enum usdhc_endian_mode_t {
  kUSDHC_EndianModeBig = 0U,
  kUSDHC_EndianModeHalfWordBig = 1U,
  kUSDHC_EndianModeLittle = 2U }

  Endian mode.
- enum usdhc_dma_mode_t {
  kUSDHC_DmaModeSimple = 0U,
  kUSDHC_DmaModeAdma1 = 1U,
  kUSDHC_DmaModeAdma2 = 2U,
  kUSDHC_ExternalDMA = 3U }

  DMA mode.
- enum {
  kUSDHC_StopAtBlockGapFlag = USDHC_PROT_CTRL_SABGREQ_MASK,
  kUSDHC_ReadWaitControlFlag = USDHC_PROT_CTRL_RWCTL_MASK,
  kUSDHC_InterruptAtBlockGapFlag = USDHC_PROT_CTRL_IABG_MASK,
  kUSDHC_ReadDoneNo8CLK = USDHC_PROT_CTRL_RD_DONE_NO_8CLK_MASK,
  kUSDHC_ExactBlockNumberReadFlag = USDHC_PROT_CTRL_NON_EXACT_BLK_RD_M-
  ASK }

  Enum _usdhc_sdio_control_flag.
- enum usdhc_boot_mode_t {
  kUSDHC_BootModeNormal = 0U,
  kUSDHC_BootModeAlternative = 1U }

  MMC card boot mode.
- enum usdhc_card_command_type_t {
  kCARD_CommandTypeNormal = 0U,
  kCARD_CommandTypeSuspend = 1U,
  kCARD_CommandTypeResume = 2U,
  kCARD_CommandTypeAbort = 3U,
  kCARD_CommandTypeEmpty = 4U }

  The command type.
- enum usdhc_card_response_type_t {

kCARD_ResponseTypeNone = 0U,
kCARD_ResponseTypeR1 = 1U,
kCARD_ResponseTypeR1b = 2U,
kCARD_ResponseTypeR2 = 3U,
kCARD_ResponseTypeR3 = 4U,
kCARD_ResponseTypeR4 = 5U,
kCARD_ResponseTypeR5 = 6U,
kCARD_ResponseTypeR5b = 7U,
kCARD_ResponseTypeR6 = 8U,
kCARD_ResponseTypeR7 = 9U }

The command response type.

- enum {
kUSDHC_Adma1DescriptorValidFlag = (1U << 0U),
kUSDHC_Adma1DescriptorEndFlag = (1U << 1U),
kUSDHC_Adma1DescriptorInterrupFlag = (1U << 2U),
kUSDHC_Adma1DescriptorActivity1Flag = (1U << 4U),
kUSDHC_Adma1DescriptorActivity2Flag = (1U << 5U),
kUSDHC_Adma1DescriptorTypeNop = (kUSDHC_Adma1DescriptorValidFlag),
kUSDHC_Adma1DescriptorTypeTransfer = (kUSDHC_Adma1DescriptorActivity2Flag | kUSDHC_Adma1DescriptorValidFlag),
kUSDHC_Adma1DescriptorTypeLink,
kUSDHC_Adma1DescriptorTypeSetLength = (kUSDHC_Adma1DescriptorActivity1Flag | kUSDHC_Adma1DescriptorValidFlag) }

  Enum _usdhc_adma1_descriptor_flag.

- enum {
kUSDHC_Adma2DescriptorValidFlag = (1U << 0U),
kUSDHC_Adma2DescriptorEndFlag = (1U << 1U),
kUSDHC_Adma2DescriptorInterruptFlag = (1U << 2U),
kUSDHC_Adma2DescriptorActivity1Flag = (1U << 4U),
kUSDHC_Adma2DescriptorActivity2Flag = (1U << 5U),
kUSDHC_Adma2DescriptorTypeNop = (kUSDHC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeReserved = (kUSDHC_Adma2DescriptorActivity1Flag | kUSDHC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeTransfer = (kUSDHC_Adma2DescriptorActivity2Flag | kUSDHC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeLink }

  Enum _usdhc_adma2_descriptor_flag.

- enum {
kUSDHC_AdmaDescriptorSingleFlag = 0U,
kUSDHC_AdmaDescriptorMultipleFlag }

  Enum _usdhc_adma_flag.

- enum usdhc_burst_len_t {
kUSDHC_EnBurstLenForINCR = 0x01U,
kUSDHC_EnBurstLenForINCR4816 = 0x02U,
kUSDHC_EnBurstLenForINCR4816WRAP = 0x04U }

  DMA transfer burst len config.

- enum {
  kUSDHC_TransferDataNormal = 0U,
  kUSDHC_TransferDataTuning = 1U,
  kUSDHC_TransferDataBoot = 2U,
  kUSDHC_TransferDataBootcontinous = 3U }
  Enum _usdhc_transfer_data_type.

Driver version

- #define FSL_USDHC_DRIVER_VERSION (MAKE_VERSION(2U, 8U, 1U))
  Driver version 2.8.1.

Initialization and deinitialization

- void USDHC_Init (USDHC_Type ∗base, const usdhc_config_t ∗config)
  USDHC module initialization function.
- void USDHC_Deinit (USDHC_Type ∗base)
  Deinitializes the USDHC.
- bool USDHC_Reset (USDHC_Type ∗base, uint32_t mask, uint32_t timeout)
  Resets the USDHC.

DMA Control

- status_t USDHC_SetAdmaTableConfig (USDHC_Type ∗base, usdhc_adma_config_t ∗dmaConfig, usdhc_data_t ∗dataConfig, uint32_t flags)
  Sets the DMA descriptor table configuration.
- status_t USDHC_SetInternalDmaConfig (USDHC_Type ∗base, usdhc_adma_config_t ∗dmaConfig, const uint32_t ∗dataAddr, bool enAutoCmd23)
  Internal DMA configuration.
- status_t USDHC_SetADMA2Descriptor (uint32_t ∗admaTable, uint32_t admaTableWords, const uint32_t ∗dataBufferAddr, uint32_t dataBytes, uint32_t flags)
  Sets the ADMA2 descriptor table configuration.
- status_t USDHC_SetADMA1Descriptor (uint32_t ∗admaTable, uint32_t admaTableWords, const uint32_t ∗dataBufferAddr, uint32_t dataBytes, uint32_t flags)
  Sets the ADMA1 descriptor table configuration.
- static void USDHC_EnableInternalDMA (USDHC_Type ∗base, bool enable)
  Enables internal DMA.

Interrupts

- static void USDHC_EnableInterruptStatus (USDHC_Type ∗base, uint32_t mask)
  Enables the interrupt status.
- static void USDHC_DisableInterruptStatus (USDHC_Type ∗base, uint32_t mask)
  Disables the interrupt status.
- static void USDHC_EnableInterruptSignal (USDHC_Type ∗base, uint32_t mask)
  Enables the interrupt signal corresponding to the interrupt status flag.
- static void USDHC_DisableInterruptSignal (USDHC_Type ∗base, uint32_t mask)
  Disables the interrupt signal corresponding to the interrupt status flag.

Status

- static uint32_t USDHC_GetEnabledInterruptStatusFlags (USDHC_Type *base)
  Gets the enabled interrupt status.
- static uint32_t USDHC_GetInterruptStatusFlags (USDHC_Type *base)
  Gets the current interrupt status.
- static void USDHC_ClearInterruptStatusFlags (USDHC_Type *base, uint32_t mask)
  Clears a specified interrupt status.
- static uint32_t USDHC_GetAutoCommand12ErrorStatusFlags (USDHC_Type *base)
  Gets the status of auto command 12 error.
- static uint32_t USDHC_GetAdmaErrorStatusFlags (USDHC_Type *base)
  Gets the status of the ADMA error.
- static uint32_t USDHC_GetPresentStatusFlags (USDHC_Type *base)
  Gets a present status.

Bus Operations

- void USDHC_GetCapability (USDHC_Type *base, usdhc_capability_t *capability)
  Gets the capability information.
- static void USDHC_ForceClockOn (USDHC_Type *base, bool enable)
  Forces the card clock on.
- uint32_t USDHC_SetSdClock (USDHC_Type *base, uint32_t srcClock_Hz, uint32_t busClock_-Hz)
  Sets the SD bus clock frequency.
- bool USDHC_SetCardActive (USDHC_Type *base, uint32_t timeout)
  Sends 80 clocks to the card to set it to the active state.
- static void USDHC_AssertHardwareReset (USDHC_Type *base, bool high)
  Triggers a hardware reset.
- static void USDHC_SetDataBusWidth (USDHC_Type *base, usdhc_data_bus_width_t width)
  Sets the data transfer width.
- static void USDHC_WriteData (USDHC_Type *base, uint32_t data)
  Fills the data port.
- static uint32_t USDHC_ReadData (USDHC_Type *base)
  Retrieves the data from the data port.
- void USDHC_SendCommand (USDHC_Type *base, usdhc_command_t *command)
  Sends command function.
- static void USDHC_EnableWakeupEvent (USDHC_Type *base, uint32_t mask, bool enable)
  Enables or disables a wakeup event in low-power mode.
- static void USDHC_CardDetectByData3 (USDHC_Type *base, bool enable)
  Detects card insert status.
- static bool USDHC_DetectCardInsert (USDHC_Type *base)
  Detects card insert status.
- static void USDHC_EnableSdioControl (USDHC_Type *base, uint32_t mask, bool enable)
  Enables or disables the SDIO card control.
- static void USDHC_SetContinueRequest (USDHC_Type *base)
  Restarts a transaction which has stopped at the block GAP for the SDIO card.
- static void USDHC_RequestStopAtBlockGap (USDHC_Type *base, bool enable)
  Request stop at block gap function.
- void USDHC_SetMmcBootConfig (USDHC_Type *base, const usdhc_boot_config_t *config)
  Configures the MMC boot feature.
- static void USDHC_EnableMmcBoot (USDHC_Type *base, bool enable)
  Enables or disables the mmc boot mode.

- static void USDHC_SetForceEvent (USDHC_Type *base, uint32_t mask)
  Forces generating events according to the given mask.
- static void UDSHC_SelectVoltage (USDHC_Type *base, bool en18v)
  Selects the USDHC output voltage.
- static bool USDHC_RequestTuningForSDR50 (USDHC_Type *base)
  Checks the SDR50 mode request tuning bit.
- static bool USDHC_RequestReTuning (USDHC_Type *base)
  Checks the request re-tuning bit.
- static void USDHC_EnableAutoTuning (USDHC_Type *base, bool enable)
  The SDR104 mode auto tuning enable and disable.
- static void USDHC_SetRetuningTimer (USDHC_Type *base, uint32_t counter)
  Configs the re-tuning timer for mode 1 and mode 3.
- void USDHC_EnableAutoTuningForCmdAndData (USDHC_Type *base)
  The auto tuning enbale for CMD/DATA line.
- void USDHC_EnableManualTuning (USDHC_Type *base, bool enable)
  Manual tuning trigger or abort.
- static uint32_t USDHC_GetTuningDelayStatus (USDHC_Type *base)
  Get the tuning delay cell setting.
- status_t USDHC_SetTuningDelay (USDHC_Type *base, uint32_t preDelay, uint32_t outDelay, uint32_t postDelay)
  The tuning delay cell setting.
- status_t USDHC_AdjustDelayForManualTuning (USDHC_Type *base, uint32_t delay)
  Adjusts delay for mannual tuning.
- static void USDHC_SetStandardTuningCounter (USDHC_Type *base, uint8_t counter)
  set tuning counter tuning.
- void USDHC_EnableStandardTuning (USDHC_Type *base, uint32_t tuningStartTap, uint32_t step, bool enable)
  The enable standard tuning function.
- static uint32_t USDHC_GetExecuteStdTuningStatus (USDHC_Type *base)
  Gets execute STD tuning status.
- static uint32_t USDHC_CheckStdTuningResult (USDHC_Type *base)
  Checks STD tuning result.
- static uint32_t USDHC_CheckTuningError (USDHC_Type *base)
  Checks tuning error.
- void USDHC_EnableDDRMode (USDHC_Type *base, bool enable, uint32_t nibblePos)
  The enable/disable DDR mode.
- static void USDHC_EnableHS400Mode (USDHC_Type *base, bool enable)
  The enable/disable HS400 mode.
- static void USDHC_ResetStrobeDLL (USDHC_Type *base)
  Resets the strobe DLL.
- static void USDHC_EnableStrobeDLL (USDHC_Type *base, bool enable)
  Enables/disables the strobe DLL.
- void USDHC_ConfigStrobeDLL (USDHC_Type *base, uint32_t delayTarget, uint32_t updateInterval)
  Configs the strobe DLL delay target and update interval.
- static void USDHC_SetStrobeDllOverride (USDHC_Type *base, uint32_t delayTaps)
  *Enables manual override for slave delay chain using **STROBE_SLV_OVERRIDE_VAL**.*
- static uint32_t USDHC_GetStrobeDLLStatus (USDHC_Type *base)
  Gets the strobe DLL status.
- void USDHC_SetDataConfig (USDHC_Type *base, usdhc_transfer_direction_t dataDirection, uint32_t blockCount, uint32_t blockSize)

USDHC data configuration.

Transactional functions

- void USDHC_TransferCreateHandle (USDHC_Type *base, usdhc_handle_t *handle, const usdhc_transfer_callback_t *callback, void *userData)
  Creates the USDHC handle.
- status_t USDHC_TransferNonBlocking (USDHC_Type *base, usdhc_handle_t *handle, usdhc_adma_config_t *dmaConfig, usdhc_transfer_t *transfer)
  Transfers the command/data using an interrupt and an asynchronous method.
- status_t USDHC_TransferBlocking (USDHC_Type *base, usdhc_adma_config_t *dmaConfig, usdhc_transfer_t *transfer)
  Transfers the command/data using a blocking method.
- void USDHC_TransferHandleIRQ (USDHC_Type *base, usdhc_handle_t *handle)
  IRQ handler for the USDHC.

45.3 Data Structure Documentation

45.3.1 struct usdhc_adma2_descriptor_t

Data Fields

- uint32_t attribute
  The control and status field.
- const uint32_t * address
  The address field.

Field Documentation

(1) uint32_t usdhc_adma2_descriptor_t::attribute

(2) const uint32_t* usdhc_adma2_descriptor_t::address

45.3.2 struct usdhc_capability_t

Defines a structure to save the capability information of USDHC.

Data Fields

- uint32_t sdVersion
  Support SD card/sdio version.
- uint32_t mmcVersion
  Support EMMC card version.
- uint32_t maxBlockLength
  Maximum block length united as byte.
- uint32_t maxBlockCount
  Maximum block count can be set one time.

- uint32_t flags
  Capability flags to indicate the support information(_usdhc_capability_flag).

Field Documentation

(1) uint32_t usdhc_capability_t::sdVersion

(2) uint32_t usdhc_capability_t::mmcVersion

(3) uint32_t usdhc_capability_t::maxBlockLength

(4) uint32_t usdhc_capability_t::maxBlockCount

(5) uint32_t usdhc_capability_t::flags

45.3.3 struct usdhc_boot_config_t

Data Fields

- uint32_t ackTimeoutCount
  Timeout value for the boot ACK.
- usdhc_boot_mode_t bootMode
  Boot mode selection.
- uint32_t blockCount
  Stop at block gap value of automatic mode.
- size_t blockSize
  Block size.
- bool enableBootAck
  Enable or disable boot ACK.
- bool enableAutoStopAtBlockGap
  Enable or disable auto stop at block gap function in boot period.

Field Documentation

(1) uint32_t usdhc_boot_config_t::ackTimeoutCount

The available range is 0 ∼ 15.

(2) usdhc_boot_mode_t usdhc_boot_config_t::bootMode

(3) uint32_t usdhc_boot_config_t::blockCount

Available range is 0 ∼ 65535.

(4) size_t usdhc_boot_config_t::blockSize

(5) bool usdhc_boot_config_t::enableBootAck

(6) bool usdhc_boot_config_t::enableAutoStopAtBlockGap

45.3.4 struct usdhc_config_t

Data Fields

- uint32_t dataTimeout
  Data timeout value.
- usdhc_endian_mode_t endianMode
  Endian mode.
- uint8_t readWatermarkLevel
  Watermark level for DMA read operation.
- uint8_t writeWatermarkLevel
  Watermark level for DMA write operation.
- uint8_t readBurstLen
  Read burst len.
- uint8_t writeBurstLen
  Write burst len.

Field Documentation

(1) uint32_t usdhc_config_t::dataTimeout

(2) usdhc_endian_mode_t usdhc_config_t::endianMode

(3) uint8_t usdhc_config_t::readWatermarkLevel

Available range is 1 ∼ 128.

(4) uint8_t usdhc_config_t::writeWatermarkLevel

Available range is 1 ∼ 128.

(5) uint8_t usdhc_config_t::readBurstLen

(6) uint8_t usdhc_config_t::writeBurstLen

45.3.5 struct usdhc_command_t

Defines card command-related attribute.

Data Fields

- uint32_t index
  Command index.
- uint32_t argument
  Command argument.
- usdhc_card_command_type_t type
  Command type.
- usdhc_card_response_type_t responseType

Command response type.
- uint32_t response [4U]
  Response for this command.
- uint32_t responseErrorFlags
  Response error flag, which need to check the command reponse.
- uint32_t flags
  Cmd flags.

Field Documentation

(1) uint32_t usdhc_command_t::index

(2) uint32_t usdhc_command_t::argument

(3) usdhc_card_command_type_t usdhc_command_t::type

(4) usdhc_card_response_type_t usdhc_command_t::responseType

(5) uint32_t usdhc_command_t::response[4U]

(6) uint32_t usdhc_command_t::responseErrorFlags

(7) uint32_t usdhc_command_t::flags

45.3.6 struct usdhc_adma_config_t

Data Fields

- usdhc_dma_mode_t dmaMode
  DMA mode.
- usdhc_burst_len_t burstLen
  Burst len config.
- uint32_t * admaTable
  ADMA table address, can't be null if transfer way is ADMA1/ADMA2.
- uint32_t admaTableWords
  ADMA table length united as words, can't be 0 if transfer way is ADMA1/ADMA2.

Field Documentation

(1) usdhc_dma_mode_t usdhc_adma_config_t::dmaMode

(2) usdhc_burst_len_t usdhc_adma_config_t::burstLen

(3) uint32_t* usdhc_adma_config_t::admaTable

(4) uint32_t usdhc_adma_config_t::admaTableWords

45.3.7 struct usdhc_scatter_gather_data_list_t

Allow application register uncontinuous data buffer for data transfer.

45.3.8 struct usdhc_scatter_gather_data_t

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Data Fields

- bool enableAutoCommand12
  Enable auto CMD12.
- bool enableAutoCommand23
  Enable auto CMD23.
- bool enableIgnoreError
  Enable to ignore error event to read/write all the data.
- usdhc_transfer_direction_t dataDirection
  data direction
- uint8_t dataType
  this is used to distinguish the normal/tuning/boot data.
- size_t blockSize
  Block size.
- usdhc_scatter_gather_data_list_t sgData
  scatter gather data

Field Documentation

(1) bool usdhc_scatter_gather_data_t::enableAutoCommand12

(2) bool usdhc_scatter_gather_data_t::enableAutoCommand23

(3) bool usdhc_scatter_gather_data_t::enableIgnoreError

(4) uint8_t usdhc_scatter_gather_data_t::dataType

(5) size_t usdhc_scatter_gather_data_t::blockSize

45.3.9 struct usdhc_scatter_gather_transfer_t

Data Fields

- usdhc_scatter_gather_data_t * data
  Data to transfer.
- usdhc_command_t * command
  Command to send.

Field Documentation

(1) usdhc_scatter_gather_data_t* usdhc_scatter_gather_transfer_t::data

(2) usdhc_command_t∗ usdhc_scatter_gather_transfer_t::command

45.3.10 struct usdhc_data_t

Defines a structure to contain data-related attribute. The 'enableIgnoreError' is used when upper card driver wants to ignore the error event to read/write all the data and not to stop read/write immediately when an error event happens. For example, bus testing procedure for MMC card.

Data Fields

- bool enableAutoCommand12
  Enable auto CMD12.
- bool enableAutoCommand23
  Enable auto CMD23.
- bool enableIgnoreError
  Enable to ignore error event to read/write all the data.
- uint8_t dataType
  this is used to distinguish the normal/tuning/boot data.
- size_t blockSize
  Block size.
- uint32_t blockCount
  Block count.
- uint32_t ∗ rxData
  Buffer to save data read.
- const uint32_t ∗ txData
  Data buffer to write.

Field Documentation

(1) bool usdhc_data_t::enableAutoCommand12

(2) bool usdhc_data_t::enableAutoCommand23

(3) bool usdhc_data_t::enableIgnoreError

(4) uint8_t usdhc_data_t::dataType

(5) size_t usdhc_data_t::blockSize

(6) uint32_t usdhc_data_t::blockCount

(7) uint32_t∗ usdhc_data_t::rxData

(8) const uint32_t∗ usdhc_data_t::txData

45.3.11 struct usdhc_transfer_t

Data Fields

- usdhc_data_t * data
  Data to transfer.
- usdhc_command_t * command
  Command to send.

Field Documentation

(1) usdhc_data_t* usdhc_transfer_t::data

(2) usdhc_command_t* usdhc_transfer_t::command

45.3.12 struct usdhc_transfer_callback_t

Data Fields

- void(* CardInserted)(USDHC_Type *base, void *userData)
  Card inserted occurs when DAT3/CD pin is for card detect.
- void(* CardRemoved)(USDHC_Type *base, void *userData)
  Card removed occurs.
- void(* SdioInterrupt)(USDHC_Type *base, void *userData)
  SDIO card interrupt occurs.
- void(* BlockGap)(USDHC_Type *base, void *userData)
  stopped at block gap event
- void(* TransferComplete)(USDHC_Type *base, usdhc_handle_t *handle, status_t status, void *userData)
  Transfer complete callback.
- void(* ReTuning)(USDHC_Type *base, void *userData)
  Handle the re-tuning.

Field Documentation

(1) void(* usdhc_transfer_callback_t::TransferComplete)(USDHC_Type *base, usdhc_handle_t *handle, status_t status, void *userData)

(2) void(* usdhc_transfer_callback_t::ReTuning)(USDHC_Type *base, void *userData)

45.3.13 struct _usdhc_handle

USDHC handle typedef.

Defines the structure to save the USDHC state information and callback function.

Note

All the fields except interruptFlags and transferredWords must be allocated by the user.

Data Fields

- usdhc_data_t ∗volatile data
  Transfer parameter.
- usdhc_command_t ∗volatile command
  Transfer parameter.
- volatile uint32_t transferredWords
  Transfer status.
- usdhc_transfer_callback_t callback
  Callback function.
- void ∗ userData
  Parameter for transfer complete callback.

Field Documentation

(1) usdhc_data_t∗ volatile usdhc_handle_t::data

Data to transfer.

(2) usdhc_command_t∗ volatile usdhc_handle_t::command

Command to send.

(3) volatile uint32_t usdhc_handle_t::transferredWords

Words transferred by DATAPORT way.

(4) usdhc_transfer_callback_t usdhc_handle_t::callback

(5) void∗ usdhc_handle_t::userData

45.3.14 struct usdhc_host_t

Data Fields

- USDHC_Type ∗ base
  USDHC peripheral base address.
- uint32_t sourceClock_Hz
  USDHC source clock frequency united in Hz.
- usdhc_config_t config
  USDHC configuration.
- usdhc_capability_t capability
  USDHC capability information.
- usdhc_transfer_function_t transfer
  USDHC transfer function.

Field Documentation

(1) **USDHC_Type* usdhc_host_t::base**

(2) **uint32_t usdhc_host_t::sourceClock_Hz**

(3) **usdhc_config_t usdhc_host_t::config**

(4) **usdhc_capability_t usdhc_host_t::capability**

(5) **usdhc_transfer_function_t usdhc_host_t::transfer**

45.4 Macro Definition Documentation

45.4.1 #define FSL_USDHC_DRIVER_VERSION (MAKE_VERSION(2U, 8U, 1U))

45.4.2 #define USDHC_ADMA1_ADDRESS_ALIGN (4096U)

45.4.3 #define USDHC_ADMA1_LENGTH_ALIGN (4096U)

45.4.4 #define USDHC_ADMA2_ADDRESS_ALIGN (4U)

45.4.5 #define USDHC_ADMA2_LENGTH_ALIGN (4U)

45.4.6 #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT (12U)

Address/page field	**Reserved**		6p(*8-*3)*6/8|lightgray**Attribute**			
31 12	11 6	05	04	03	02	0
address or data length	000000	Act2	Act1	0	Int	E

ADMA1 descriptor table

45.4.7 #define USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK (0xFFFFFU)

45.4.8 #define USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT (12U)

45.4.9 #define USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK (0xFFFFU)

Act2	Act1	Comment	31-28	27-12
0	0	No op	Don’t care	
0	1	Set data length	0000	Data Length
1	0	Transfer data	Data address	
1	1	Link descriptor	Descriptor address	

ADMA2 action

45.4.10 #define USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY (USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK + 1U - 4096U)

Since the max transfer size ADMA1 support is 65535 which is indivisible by 4096, so to make sure a large data load transfer (>64KB) continuously (require the data address be always align with 4096), software will set the maximum data length for ADMA1 to (64 - 4)KB.

45.4.11 #define USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT (16U)

Address field	Length	Reserved		6p(*9-*4)*6/9|lightgray**Attribute**			
63 32	31 16	15 06	05		04	03	02
32-bit address	16-bit length	0000000000	Act2		Act1	0	Int

ADMA2 descriptor table

Act2	Act1	Comment	Operation
0	0	No op	Don’t care
0	1	Reserved	Read this line and go to next one
1	0	Transfer data	Transfer data with address and length set in this descriptor line
1	1	Link descriptor	Link to another descriptor

ADMA2 action

45.4.12 #define USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK (0xFFFFU)

45.4.13 #define USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_E-NTRY (USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK

45.5.1 typedef uint32_t usdhc_adma1_descriptor_t

45.5.2 typedef status_t(∗ usdhc_transfer_function_t)(USDHC_Type ∗base, usdhc_transfer_t ∗content)

45.6 Enumeration Type Documentation

45.6.1 anonymous enum

USDHC status.

Enumerator

kStatus_USDHC_BusyTransferring Transfer is on-going.
kStatus_USDHC_PrepareAdmaDescriptorFailed Set DMA descriptor failed.
kStatus_USDHC_SendCommandFailed Send command failed.
kStatus_USDHC_TransferDataFailed Transfer data failed.
kStatus_USDHC_DMADataAddrNotAlign Data address not aligned.
kStatus_USDHC_ReTuningRequest Re-tuning request.
kStatus_USDHC_TuningError Tuning error.
kStatus_USDHC_NotSupport Not support.
kStatus_USDHC_TransferDataComplete Transfer data complete.
kStatus_USDHC_SendCommandSuccess Transfer command complete.
kStatus_USDHC_TransferDMAComplete Transfer DMA complete.

45.6.2 anonymous enum

Host controller capabilities flag mask.

Enumerator

kUSDHC_SupportAdmaFlag Support ADMA.
kUSDHC_SupportHighSpeedFlag Support high-speed.
kUSDHC_SupportDmaFlag Support DMA.
kUSDHC_SupportSuspendResumeFlag Support suspend/resume.
kUSDHC_SupportV330Flag Support voltage 3.3V.
kUSDHC_SupportV300Flag Support voltage 3.0V.
kUSDHC_SupportV180Flag Support voltage 1.8V.
kUSDHC_Support4BitFlag Flag in HTCAPBLT_MBL's position, supporting 4-bit mode.
kUSDHC_Support8BitFlag Flag in HTCAPBLT_MBL's position, supporting 8-bit mode.
kUSDHC_SupportDDR50Flag SD version 3.0 new feature, supporting DDR50 mode.
kUSDHC_SupportSDR104Flag Support SDR104 mode.
kUSDHC_SupportSDR50Flag Support SDR50 mode.

45.6.3 anonymous enum

Wakeup event mask.

Enumerator

kUSDHC_WakeupEventOnCardInt Wakeup on card interrupt.
kUSDHC_WakeupEventOnCardInsert Wakeup on card insertion.
kUSDHC_WakeupEventOnCardRemove Wakeup on card removal.
kUSDHC_WakeupEventsAll All wakeup events.

45.6.4 anonymous enum

Reset type mask.

Enumerator

kUSDHC_ResetAll Reset all except card detection.
kUSDHC_ResetCommand Reset command line.
kUSDHC_ResetData Reset data line.
kUSDHC_ResetTuning Reset tuning circuit.
kUSDHC_ResetsAll All reset types.

45.6.5 anonymous enum

Transfer flag mask.

Enumerator

kUSDHC_EnableDmaFlag Enable DMA.
kUSDHC_CommandTypeSuspendFlag Suspend command.
kUSDHC_CommandTypeResumeFlag Resume command.
kUSDHC_CommandTypeAbortFlag Abort command.
kUSDHC_EnableBlockCountFlag Enable block count.
kUSDHC_EnableAutoCommand12Flag Enable auto CMD12.
kUSDHC_DataReadFlag Enable data read.
kUSDHC_MultipleBlockFlag Multiple block data read/write.
kUSDHC_EnableAutoCommand23Flag Enable auto CMD23.
kUSDHC_ResponseLength136Flag 136-bit response length.
kUSDHC_ResponseLength48Flag 48-bit response length.
kUSDHC_ResponseLength48BusyFlag 48-bit response length with busy status.
kUSDHC_EnableCrcCheckFlag Enable CRC check.
kUSDHC_EnableIndexCheckFlag Enable index check.
kUSDHC_DataPresentFlag Data present flag.

45.6.6 anonymous enum

Present status flag mask.

Enumerator

kUSDHC_CommandInhibitFlag Command inhibit.
kUSDHC_DataInhibitFlag Data inhibit.
kUSDHC_DataLineActiveFlag Data line active.
kUSDHC_SdClockStableFlag SD bus clock stable.
kUSDHC_WriteTransferActiveFlag Write transfer active.
kUSDHC_ReadTransferActiveFlag Read transfer active.
kUSDHC_BufferWriteEnableFlag Buffer write enable.
kUSDHC_BufferReadEnableFlag Buffer read enable.
kUSDHC_ReTuningRequestFlag Re-tuning request flag, only used for SDR104 mode.
kUSDHC_DelaySettingFinishedFlag Delay setting finished flag.
kUSDHC_CardInsertedFlag Card inserted.
kUSDHC_CommandLineLevelFlag Command line signal level.
kUSDHC_Data0LineLevelFlag Data0 line signal level.
kUSDHC_Data1LineLevelFlag Data1 line signal level.
kUSDHC_Data2LineLevelFlag Data2 line signal level.
kUSDHC_Data3LineLevelFlag Data3 line signal level.
kUSDHC_Data4LineLevelFlag Data4 line signal level.
kUSDHC_Data5LineLevelFlag Data5 line signal level.
kUSDHC_Data6LineLevelFlag Data6 line signal level.
kUSDHC_Data7LineLevelFlag Data7 line signal level.

45.6.7 anonymous enum

Interrupt status flag mask.

Enumerator

kUSDHC_CommandCompleteFlag Command complete.
kUSDHC_DataCompleteFlag Data complete.
kUSDHC_BlockGapEventFlag Block gap event.
kUSDHC_DmaCompleteFlag DMA interrupt.
kUSDHC_BufferWriteReadyFlag Buffer write ready.
kUSDHC_BufferReadReadyFlag Buffer read ready.
kUSDHC_CardInsertionFlag Card inserted.
kUSDHC_CardRemovalFlag Card removed.
kUSDHC_CardInterruptFlag Card interrupt.
kUSDHC_ReTuningEventFlag Re-Tuning event, only for SD3.0 SDR104 mode.
kUSDHC_TuningPassFlag SDR104 mode tuning pass flag.
kUSDHC_TuningErrorFlag SDR104 tuning error flag.

kUSDHC_CommandTimeoutFlag Command timeout error.
kUSDHC_CommandCrcErrorFlag Command CRC error.
kUSDHC_CommandEndBitErrorFlag Command end bit error.
kUSDHC_CommandIndexErrorFlag Command index error.
kUSDHC_DataTimeoutFlag Data timeout error.
kUSDHC_DataCrcErrorFlag Data CRC error.
kUSDHC_DataEndBitErrorFlag Data end bit error.
kUSDHC_AutoCommand12ErrorFlag Auto CMD12 error.
kUSDHC_DmaErrorFlag DMA error.
kUSDHC_CommandErrorFlag Command error.
kUSDHC_DataErrorFlag Data error.
kUSDHC_ErrorFlag All error.
kUSDHC_DataFlag Data interrupts.
kUSDHC_DataDMAFlag Data interrupts.
kUSDHC_CommandFlag Command interrupts.
kUSDHC_CardDetectFlag Card detection interrupts.
kUSDHC_SDR104TuningFlag SDR104 tuning flag.
kUSDHC_AllInterruptFlags All flags mask.

45.6.8 anonymous enum

Auto CMD12 error status flag mask.

Enumerator

kUSDHC_AutoCommand12NotExecutedFlag Not executed error.
kUSDHC_AutoCommand12TimeoutFlag Timeout error.
kUSDHC_AutoCommand12EndBitErrorFlag End bit error.
kUSDHC_AutoCommand12CrcErrorFlag CRC error.
kUSDHC_AutoCommand12IndexErrorFlag Index error.
kUSDHC_AutoCommand12NotIssuedFlag Not issued error.

45.6.9 anonymous enum

Standard tuning flag.

Enumerator

kUSDHC_ExecuteTuning Used to start tuning procedure.
kUSDHC_TuningSampleClockSel When **std_tuning_en** bit is set, this bit is used to select sampleing clock.

45.6.10 anonymous enum

ADMA error status flag mask.

Enumerator

kUSDHC_AdmaLenghMismatchFlag Length mismatch error.
kUSDHC_AdmaDescriptorErrorFlag Descriptor error.

45.6.11 anonymous enum

ADMA error state.

This state is the detail state when ADMA error has occurred.

Enumerator

kUSDHC_AdmaErrorStateStopDma Stop DMA, previous location set in the ADMA system address is errored address.
kUSDHC_AdmaErrorStateFetchDescriptor Fetch descriptor, current location set in the ADMA system address is errored address.
kUSDHC_AdmaErrorStateChangeAddress Change address, no DMA error has occurred.
kUSDHC_AdmaErrorStateTransferData Transfer data, previous location set in the ADMA system address is errored address.
kUSDHC_AdmaErrorStateInvalidLength Invalid length in ADMA descriptor.
kUSDHC_AdmaErrorStateInvalidDescriptor Invalid descriptor fetched by ADMA.
kUSDHC_AdmaErrorState ADMA error state.

45.6.12 anonymous enum

Force event bit position.

Enumerator

kUSDHC_ForceEventAutoCommand12NotExecuted Auto CMD12 not executed error.
kUSDHC_ForceEventAutoCommand12Timeout Auto CMD12 timeout error.
kUSDHC_ForceEventAutoCommand12CrcError Auto CMD12 CRC error.
kUSDHC_ForceEventEndBitError Auto CMD12 end bit error.
kUSDHC_ForceEventAutoCommand12IndexError Auto CMD12 index error.
kUSDHC_ForceEventAutoCommand12NotIssued Auto CMD12 not issued error.
kUSDHC_ForceEventCommandTimeout Command timeout error.
kUSDHC_ForceEventCommandCrcError Command CRC error.
kUSDHC_ForceEventCommandEndBitError Command end bit error.
kUSDHC_ForceEventCommandIndexError Command index error.
kUSDHC_ForceEventDataTimeout Data timeout error.

kUSDHC_ForceEventDataCrcError Data CRC error.
kUSDHC_ForceEventDataEndBitError Data end bit error.
kUSDHC_ForceEventAutoCommand12Error Auto CMD12 error.
kUSDHC_ForceEventCardInt Card interrupt.
kUSDHC_ForceEventDmaError Dma error.
kUSDHC_ForceEventTuningError Tuning error.
kUSDHC_ForceEventsAll All force event flags mask.

45.6.13 enum usdhc_transfer_direction_t

Enumerator

kUSDHC_TransferDirectionReceive USDHC transfer direction receive.
kUSDHC_TransferDirectionSend USDHC transfer direction send.

45.6.14 enum usdhc_data_bus_width_t

Enumerator

kUSDHC_DataBusWidth1Bit 1-bit mode
kUSDHC_DataBusWidth4Bit 4-bit mode
kUSDHC_DataBusWidth8Bit 8-bit mode

45.6.15 enum usdhc_endian_mode_t

Enumerator

kUSDHC_EndianModeBig Big endian mode.
kUSDHC_EndianModeHalfWordBig Half word big endian mode.
kUSDHC_EndianModeLittle Little endian mode.

45.6.16 enum usdhc_dma_mode_t

Enumerator

kUSDHC_DmaModeSimple External DMA.
kUSDHC_DmaModeAdma1 ADMA1 is selected.
kUSDHC_DmaModeAdma2 ADMA2 is selected.
kUSDHC_ExternalDMA External DMA mode selected.

45.6.17 anonymous enum

SDIO control flag mask.

Enumerator

kUSDHC_StopAtBlockGapFlag Stop at block gap.
kUSDHC_ReadWaitControlFlag Read wait control.
kUSDHC_InterruptAtBlockGapFlag Interrupt at block gap.
kUSDHC_ReadDoneNo8CLK Read done without 8 clk for block gap.
kUSDHC_ExactBlockNumberReadFlag Exact block number read.

45.6.18 enum usdhc_boot_mode_t

Enumerator

kUSDHC_BootModeNormal Normal boot.
kUSDHC_BootModeAlternative Alternative boot.

45.6.19 enum usdhc_card_command_type_t

Enumerator

kCARD_CommandTypeNormal Normal command.
kCARD_CommandTypeSuspend Suspend command.
kCARD_CommandTypeResume Resume command.
kCARD_CommandTypeAbort Abort command.
kCARD_CommandTypeEmpty Empty command.

45.6.20 enum usdhc_card_response_type_t

Defines the command response type from card to host controller.

Enumerator

kCARD_ResponseTypeNone Response type: none.
kCARD_ResponseTypeR1 Response type: R1.
kCARD_ResponseTypeR1b Response type: R1b.
kCARD_ResponseTypeR2 Response type: R2.
kCARD_ResponseTypeR3 Response type: R3.
kCARD_ResponseTypeR4 Response type: R4.
kCARD_ResponseTypeR5 Response type: R5.

kCARD_ResponseTypeR5b Response type: R5b.
kCARD_ResponseTypeR6 Response type: R6.
kCARD_ResponseTypeR7 Response type: R7.

45.6.21 anonymous enum

The mask for the control/status field in ADMA1 descriptor.

Enumerator

kUSDHC_Adma1DescriptorValidFlag Valid flag.
kUSDHC_Adma1DescriptorEndFlag End flag.
kUSDHC_Adma1DescriptorInterrupFlag Interrupt flag.
kUSDHC_Adma1DescriptorActivity1Flag Activity 1 flag.
kUSDHC_Adma1DescriptorActivity2Flag Activity 2 flag.
kUSDHC_Adma1DescriptorTypeNop No operation.
kUSDHC_Adma1DescriptorTypeTransfer Transfer data.
kUSDHC_Adma1DescriptorTypeLink Link descriptor.
kUSDHC_Adma1DescriptorTypeSetLength Set data length.

45.6.22 anonymous enum

ADMA1 descriptor control and status mask.

Enumerator

kUSDHC_Adma2DescriptorValidFlag Valid flag.
kUSDHC_Adma2DescriptorEndFlag End flag.
kUSDHC_Adma2DescriptorInterruptFlag Interrupt flag.
kUSDHC_Adma2DescriptorActivity1Flag Activity 1 mask.
kUSDHC_Adma2DescriptorActivity2Flag Activity 2 mask.
kUSDHC_Adma2DescriptorTypeNop No operation.
kUSDHC_Adma2DescriptorTypeReserved Reserved.
kUSDHC_Adma2DescriptorTypeTransfer Transfer type.
kUSDHC_Adma2DescriptorTypeLink Link type.

45.6.23 anonymous enum

ADMA descriptor configuration flag.

Enumerator

kUSDHC_AdmaDescriptorSingleFlag Try to finish the transfer in a single ADMA descriptor. If transfer size is bigger than one ADMA descriptor's ability, new another descriptor for data transfer.

kUSDHC_AdmaDescriptorMultipleFlag Create multiple ADMA descriptors within the ADMA table, this is used for mmc boot mode specifically, which need to modify the ADMA descriptor on the fly, so the flag should be used combining with stop at block gap feature.

45.6.24 enum usdhc_burst_len_t

Enumerator

kUSDHC_EnBurstLenForINCR Enable burst len for INCR.
kUSDHC_EnBurstLenForINCR4816 Enable burst len for INCR4/INCR8/INCR16.
kUSDHC_EnBurstLenForINCR4816WRAP Enable burst len for INCR4/8/16 WRAP.

45.6.25 anonymous enum

Tansfer data type definition.

Enumerator

kUSDHC_TransferDataNormal Transfer normal read/write data.
kUSDHC_TransferDataTuning Transfer tuning data.
kUSDHC_TransferDataBoot Transfer boot data.
kUSDHC_TransferDataBootcontinous Transfer boot data continuously.

45.7 Function Documentation

45.7.1 void USDHC_Init (USDHC_Type ∗ *base,* const usdhc_config_t ∗ *config*)

Configures the USDHC according to the user configuration.

Example:

```
usdhc_config_t config;
config.cardDetectDat3 = false;
config.endianMode = kUSDHC_EndianModeLittle;
config.dmaMode = kUSDHC_DmaModeAdma2;
config.readWatermarkLevel = 128U;
config.writeWatermarkLevel = 128U;
USDHC_Init(USDHC, &config);
```

Parameters

base	USDHC peripheral base address.
config	USDHC configuration information.

Return values

kStatus_Success	Operate successfully.

45.7.2 void USDHC_Deinit (USDHC_Type * *base*)

Parameters

base	USDHC peripheral base address.

45.7.3 bool USDHC_Reset (USDHC_Type * *base*, uint32_t *mask*, uint32_t *timeout*)

Parameters

base	USDHC peripheral base address.
mask	The reset type mask(_usdhc_reset).
timeout	Timeout for reset.

Return values

true	Reset successfully.
false	Reset failed.

45.7.4 status_t USDHC_SetAdmaTableConfig (USDHC_Type * *base*, usdhc_adma_config_t * *dmaConfig*, usdhc_data_t * *dataConfig*, uint32_t *flags*)

A high level DMA descriptor configuration function.

Parameters

base	USDHC peripheral base address.
dmaConfig	ADMA configuration
dataConfig	Data descriptor
flags	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

45.7.5 status_t USDHC_SetInternalDmaConfig (USDHC_Type * *base*, usdhc_adma_config_t * *dmaConfig*, const uint32_t * *dataAddr*, bool *enAutoCmd23*)

This function is used to config the USDHC DMA related registers.

Parameters

base	USDHC peripheral base address.
dmaConfig	ADMA configuration.
dataAddr	Transfer data address, a simple DMA parameter, if ADMA is used, leave it to NULL.
enAutoCmd23	Flag to indicate Auto CMD23 is enable or not, a simple DMA parameter, if ADMA is used, leave it to false.

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

45.7.6 status_t USDHC_SetADMA2Descriptor (uint32_t * *admaTable*, uint32_t *admaTableWords*, const uint32_t * *dataBufferAddr*, uint32_t *dataBytes*, uint32_t *flags*)

Parameters

admaTable	ADMA table address.
admaTable-Words	ADMA table length.
dataBufferAddr	Data buffer address.
dataBytes	Data Data length.
flags	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

45.7.7 status_t USDHC_SetADMA1Descriptor (uint32_t * *admaTable,* uint32_t *admaTableWords,* const uint32_t * *dataBufferAddr,* uint32_t *dataBytes,* uint32_t *flags*)

Parameters

admaTable	ADMA table address.
admaTable-Words	ADMA table length.
dataBufferAddr	Data buffer address.
dataBytes	Data length.
flags	ADAM descriptor flag, used to indicate to create multiple or single descriptor, please refer to enum _usdhc_adma_flag.

Return values

kStatus_OutOfRange	ADMA descriptor table length isn't enough to describe data.
kStatus_Success	Operate successfully.

45.7.8 static void USDHC_EnableInternalDMA (USDHC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
enable	enable or disable flag

45.7.9 static void USDHC_EnableInterruptStatus (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	USDHC peripheral base address.
mask	Interrupt status flags mask(_usdhc_interrupt_status_flag).

45.7.10 static void USDHC_DisableInterruptStatus (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	USDHC peripheral base address.
mask	The interrupt status flags mask(_usdhc_interrupt_status_flag).

45.7.11 static void USDHC_EnableInterruptSignal (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	USDHC peripheral base address.
mask	The interrupt status flags mask(_usdhc_interrupt_status_flag).

45.7.12 static void USDHC_DisableInterruptSignal (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	USDHC peripheral base address.
mask	The interrupt status flags mask(_usdhc_interrupt_status_flag).

45.7.13 static uint32_t USDHC_GetEnabledInterruptStatusFlags (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

Returns

Current interrupt status flags mask(_usdhc_interrupt_status_flag).

45.7.14 static uint32_t USDHC_GetInterruptStatusFlags (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

Returns

Current interrupt status flags mask(_usdhc_interrupt_status_flag).

45.7.15 static void USDHC_ClearInterruptStatusFlags (USDHC_Type * *base*, uint32_t *mask*) [inline], [static]

write 1 clears

Parameters

base	USDHC peripheral base address.
mask	The interrupt status flags mask(_usdhc_interrupt_status_flag).

45.7.16 static uint32_t USDHC_GetAutoCommand12ErrorStatusFlags (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

Returns

Auto command 12 error status flags mask(_usdhc_auto_command12_error_status_flag).

45.7.17 static uint32_t USDHC_GetAdmaErrorStatusFlags (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

Returns

ADMA error status flags mask(_usdhc_adma_error_status_flag).

45.7.18 static uint32_t USDHC_GetPresentStatusFlags (USDHC_Type * *base*) [inline], [static]

This function gets the present USDHC's status except for an interrupt status and an error status.

Parameters

base	USDHC peripheral base address.

Returns

Present USDHC's status flags mask(_usdhc_present_status_flag).

45.7.19 void USDHC_GetCapability (USDHC_Type * *base,* usdhc_capability_t * *capability*)

Parameters

base	USDHC peripheral base address.
capability	Structure to save capability information.

45.7.20 static void USDHC_ForceClockOn (USDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
enable	enable/disable flag

45.7.21 uint32_t USDHC_SetSdClock (USDHC_Type * *base*, uint32_t *srcClock_Hz*, uint32_t *busClock_Hz*)

Parameters

base	USDHC peripheral base address.
srcClock_Hz	USDHC source clock frequency united in Hz.
busClock_Hz	SD bus clock frequency united in Hz.

Returns

The nearest frequency of busClock_Hz configured for SD bus.

45.7.22 bool USDHC_SetCardActive (USDHC_Type * *base*, uint32_t *timeout*)

This function must be called each time the card is inserted to ensure that the card can receive the command correctly.

Parameters

base	USDHC peripheral base address.
timeout	Timeout to initialize card.

Return values

true	Set card active successfully.
false	Set card active failed.

45.7.23 static void USDHC_AssertHardwareReset (USDHC_Type * *base,* bool *high*) [inline], [static]

Parameters

base	USDHC peripheral base address.
high	1 or 0 level

45.7.24 static void USDHC_SetDataBusWidth (USDHC_Type * *base,* usdhc_data_bus_width_t *width*) [inline], [static]

Parameters

base	USDHC peripheral base address.
width	Data transfer width.

45.7.25 static void USDHC_WriteData (USDHC_Type * *base,* uint32_t *data*) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

base	USDHC peripheral base address.
data	The data about to be sent.

45.7.26 static uint32_t USDHC_ReadData (USDHC_Type * *base*) [inline], [static]

This function is used to implement the data transfer by Data Port instead of DMA.

Parameters

base	USDHC peripheral base address.

Returns

The data has been read.

45.7.27 void USDHC_SendCommand (USDHC_Type * *base*, usdhc_command_t * *command*)

Parameters

base	USDHC peripheral base address.
command	configuration

45.7.28 static void USDHC_EnableWakeupEvent (USDHC_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
mask	Wakeup events mask(_usdhc_wakeup_event).
enable	True to enable, false to disable.

45.7.29 static void USDHC_CardDetectByData3 (USDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
enable	enable/disable flag

45.7.30 static bool USDHC_DetectCardInsert (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

45.7.31 static void USDHC_EnableSdioControl (USDHC_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
mask	SDIO card control flags mask(_usdhc_sdio_control_flag).
enable	True to enable, false to disable.

45.7.32 static void USDHC_SetContinueRequest (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

45.7.33 static void USDHC_RequestStopAtBlockGap (USDHC_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
enable	True to stop at block gap, false to normal transfer.

45.7.34 void USDHC_SetMmcBootConfig (USDHC_Type * *base*, const usdhc_boot_config_t * *config*)

Example:

```
usdhc_boot_config_t config;
config.ackTimeoutCount = 4;
config.bootMode = kUSDHC_BootModeNormal;
config.blockCount = 5;
config.enableBootAck = true;
```

```
config.enableBoot = true;
config.enableAutoStopAtBlockGap = true;
USDHC_SetMmcBootConfig(USDHC, &config);
```

Parameters

base	USDHC peripheral base address.
config	The MMC boot configuration information.

45.7.35 static void USDHC_EnableMmcBoot (USDHC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
enable	True to enable, false to disable.

45.7.36 static void USDHC_SetForceEvent (USDHC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	USDHC peripheral base address.
mask	The force events bit posistion (_usdhc_force_event).

45.7.37 static void UDSHC_SelectVoltage (USDHC_Type * *base,* bool *en18v*) [inline], [static]

Parameters

base	USDHC peripheral base address.
en18v	True means 1.8V, false means 3.0V.

45.7.38 static bool USDHC_RequestTuningForSDR50 (USDHC_Type * *base*) [inline], [static]

When this bit set, application shall perform tuning for SDR50 mode.

Parameters

base	USDHC peripheral base address.

45.7.39 static bool USDHC_RequestReTuning (USDHC_Type * *base*) [inline], [static]

When this bit is set, user should do manual tuning or standard tuning function.

Parameters

base	USDHC peripheral base address.

45.7.40 static void USDHC_EnableAutoTuning (USDHC_Type * *base,* bool *enable*) [inline], [static]

This function should be called after tuning function execute pass, auto tuning will handle by hardware.

Parameters

base	USDHC peripheral base address.
enable	enable/disable flag

45.7.41 static void USDHC_SetRetuningTimer (USDHC_Type * *base,* uint32_t *counter*) [inline], [static]

This timer is used for standard tuning auto re-tuning,

Parameters

base	USDHC peripheral base address.
counter	timer counter value

45.7.42 void USDHC_EnableAutoTuningForCmdAndData (USDHC_Type * *base*)

Parameters

base	USDHC peripheral base address.

45.7.43 void USDHC_EnableManualTuning (USDHC_Type * *base*, bool *enable*)

User should handle the tuning cmd and find the boundary of the delay then calucate a average value which will be configured to the **CLK_TUNE_CTRL_STATUS** This function should be called before function USDHC_AdjustDelayForManualTuning.

Parameters

base	USDHC peripheral base address.
enable	tuning enable flag

45.7.44 static uint32_t USDHC_GetTuningDelayStatus (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

Return values

CLK	Tuning Control and Status register value.

45.7.45 status_t USDHC_SetTuningDelay (USDHC_Type * *base*, uint32_t *preDelay*, uint32_t *outDelay*, uint32_t *postDelay*)

Parameters

base	USDHC peripheral base address.
preDelay	Set the number of delay cells on the feedback clock between the feedback clock and CLK_PRE.

outDelay	Set the number of delay cells on the feedback clock between CLK_PRE and CLK_-OUT.
postDelay	Set the number of delay cells on the feedback clock between CLK_OUT and CLK_-POST.

Return values

kStatus_Fail	config the delay setting fail
kStatus_Success	config the delay setting success

45.7.46 status_t USDHC_AdjustDelayForManualTuning (USDHC_Type * *base*, uint32_t *delay*)

Deprecated Do not use this function. It has been superceded by USDHC_SetTuingDelay

Parameters

base	USDHC peripheral base address.
delay	setting configuration

Return values

kStatus_Fail	config the delay setting fail
kStatus_Success	config the delay setting success

45.7.47 static void USDHC_SetStandardTuningCounter (USDHC_Type * *base*, uint8_t *counter*) `[inline], [static]`

Parameters

base	USDHC peripheral base address.
counter	tuning counter

Return values

kStatus_Fail	config the delay setting fail
kStatus_Success	config the delay setting success

45.7.48 void USDHC_EnableStandardTuning (USDHC_Type * *base,* uint32_t *tuningStartTap,* uint32_t *step,* bool *enable*)

The standard tuning window and tuning counter using the default config tuning cmd is sent by the software, user need to check whether the tuning result can be used for SDR50, SDR104, and HS200 mode tuning.

Parameters

base	USDHC peripheral base address.
tuningStartTap	start tap
step	tuning step
enable	enable/disable flag

45.7.49 static uint32_t USDHC_GetExecuteStdTuningStatus (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

45.7.50 static uint32_t USDHC_CheckStdTuningResult (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

45.7.51 static uint32_t USDHC_CheckTuningError (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

45.7.52 void USDHC_EnableDDRMode (USDHC_Type * *base,* bool *enable,* uint32_t *nibblePos*)

Parameters

base	USDHC peripheral base address.
enable	enable/disable flag
nibblePos	nibble position

45.7.53 static void USDHC_EnableHS400Mode (USDHC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
enable	enable/disable flag

45.7.54 static void USDHC_ResetStrobeDLL (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

45.7.55 static void USDHC_EnableStrobeDLL (USDHC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	USDHC peripheral base address.
enable	enable/disable flag

45.7.56 void USDHC_ConfigStrobeDLL (USDHC_Type * *base,* uint32_t *delayTarget,* uint32_t *updateInterval*)

Parameters

base	USDHC peripheral base address.
delayTarget	delay target
updateInterval	update interval

45.7.57 static void USDHC_SetStrobeDllOverride (USDHC_Type * *base,* uint32_t *delayTaps*) [inline], [static]

Parameters

base	USDHC peripheral base address.
delayTaps	Valid delay taps range from 1 - 128 taps. A value of 0 selects tap 1, and a value of 0x7F selects tap 128.

45.7.58 static uint32_t USDHC_GetStrobeDLLStatus (USDHC_Type * *base*) [inline], [static]

Parameters

base	USDHC peripheral base address.

45.7.59 void USDHC_SetDataConfig (USDHC_Type * *base,* usdhc_transfer-_direction_t *dataDirection,* uint32_t *blockCount,* uint32_t *blockSize*)

Parameters

base	USDHC peripheral base address.
dataDirection	Data direction, tx or rx.
blockCount	Data block count.

blockSize	Data block size.

45.7.60 void USDHC_TransferCreateHandle (USDHC_Type * *base*, usdhc_handle_t * *handle*, const usdhc_transfer_callback_t * *callback*, void * *userData*)

Parameters

base	USDHC peripheral base address.
handle	USDHC handle pointer.
callback	Structure pointer to contain all callback functions.
userData	Callback function parameter.

45.7.61 status_t USDHC_TransferNonBlocking (USDHC_Type * *base*, usdhc_handle_t * *handle*, usdhc_adma_config_t * *dmaConfig*, usdhc_transfer_t * *transfer*)

This function sends a command and data and returns immediately. It doesn't wait for the transfer to complete or to encounter an error. The application must not call this API in multiple threads at the same time. Because of that this API doesn't support the re-entry mechanism.

Note

Call API USDHC_TransferCreateHandle when calling this API.

Parameters

base	USDHC peripheral base address.
handle	USDHC handle.
dmaConfig	ADMA configuration.
transfer	Transfer content.

Return values

kStatus_InvalidArgument	Argument is invalid.
kStatus_USDHC_Busy-Transferring	Busy transferring.
kStatus_USDHC_-PrepareAdmaDescriptor-Failed	Prepare ADMA descriptor failed.
kStatus_Success	Operate successfully.

45.7.62 status_t USDHC_TransferBlocking (USDHC_Type * *base*, usdhc_adma_config_t * *dmaConfig*, usdhc_transfer_t * *transfer*)

This function waits until the command response/data is received or the USDHC encounters an error by polling the status flag.

The application must not call this API in multiple threads at the same time. Because this API doesn't support the re-entry mechanism.

Note

There is no need to call API USDHC_TransferCreateHandle when calling this API.

Parameters

base	USDHC peripheral base address.
dmaConfig	adma configuration
transfer	Transfer content.

Return values

kStatus_InvalidArgument	Argument is invalid.
kStatus_USDHC_-PrepareAdmaDescriptor-Failed	Prepare ADMA descriptor failed.

kStatus_USDHC_SendCommandFailed	Send command failed.
kStatus_USDHC_TransferDataFailed	Transfer data failed.
kStatus_Success	Operate successfully.

45.7.63 void USDHC_TransferHandleIRQ (USDHC_Type * *base*, usdhc_handle_t * *handle*)

This function deals with the IRQs on the given host controller.

Parameters

base	USDHC peripheral base address.
handle	USDHC handle.

Chapter 46
UTICK: MictoTick Timer Driver

46.1 Overview

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

UTICK driver is created to help user to operate the UTICK module. The UTICK timer can be used as a low power timer. The APIs can be used to enable the UTICK module, initialize it and set the time. UTICK can be used as a wake up source from low power mode.

46.2 Typical use case

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

Files

- file fsl_utick.h

Typedefs

- typedef void(∗ utick_callback_t)(void)
  UTICK callback function.

Enumerations

- enum utick_mode_t {
  kUTICK_Onetime = 0x0U,
  kUTICK_Repeat = 0x1U }
  UTICK timer operational mode.

Driver version

- #define FSL_UTICK_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))
  UTICK driver version 2.0.5.

Initialization and deinitialization

- void UTICK_Init (UTICK_Type ∗base)
  Initializes an UTICK by turning its bus clock on.
- void UTICK_Deinit (UTICK_Type ∗base)
  Deinitializes a UTICK instance.
- uint32_t UTICK_GetStatusFlags (UTICK_Type ∗base)
  Get Status Flags.
- void UTICK_ClearStatusFlags (UTICK_Type ∗base)
  Clear Status Interrupt Flags.

- void UTICK_SetTick (UTICK_Type *base, utick_mode_t mode, uint32_t count, utick_callback_t cb)
  Starts UTICK.
- void UTICK_HandleIRQ (UTICK_Type *base, utick_callback_t cb)
  UTICK Interrupt Service Handler.

46.3 Macro Definition Documentation

46.3.1 #define FSL_UTICK_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

46.4 Typedef Documentation

46.4.1 typedef void(* utick_callback_t)(void)

46.5 Enumeration Type Documentation

46.5.1 enum utick_mode_t

Enumerator

kUTICK_Onetime Trigger once.
kUTICK_Repeat Trigger repeatedly.

46.6 Function Documentation

46.6.1 void UTICK_Init (UTICK_Type * *base*)

46.6.2 void UTICK_Deinit (UTICK_Type * *base*)

This function shuts down Utick bus clock

Parameters

base	UTICK peripheral base address.

46.6.3 uint32_t UTICK_GetStatusFlags (UTICK_Type * *base*)

This returns the status flag

Parameters

base	UTICK peripheral base address.

Returns

status register value

46.6.4 void UTICK_ClearStatusFlags (UTICK_Type * *base*)

This clears intr status flag

Parameters

base	UTICK peripheral base address.

Returns

none

46.6.5 void UTICK_SetTick (UTICK_Type * *base*, utick_mode_t *mode*, uint32_t *count*, utick_callback_t *cb*)

This function starts a repeat/onetime countdown with an optional callback

Parameters

base	UTICK peripheral base address.
mode	UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
count	UTICK timer mode (ie kUTICK_onetime or kUTICK_repeat)
cb	UTICK callback (can be left as NULL if none, otherwise should be a void func(void))

Returns

none

46.6.6 void UTICK_HandleIRQ (UTICK_Type * *base*, utick_callback_t *cb*)

This function handles the interrupt and refers to the callback array in the driver to callback user (as per request in UTICK_SetTick()). if no user callback is scheduled, the interrupt will simply be cleared.

Parameters

base	UTICK peripheral base address.
cb	callback scheduled for this instance of UTICK

Returns

none

Chapter 47
WWDT: Windowed Watchdog Timer Driver

47.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

47.2 Function groups

47.2.1 Initialization and deinitialization

The function WWDT_Init() initializes the watchdog timer with specified configurations. The configurations include timeout value and whether to enable watchdog after init. The function WWDT_GetDefaultConfig() gets the default configurations.

The function WWDT_Deinit() disables the watchdog and the module clock.

47.2.2 Status

Provides functions to get and clear the WWDT status.

47.2.3 Interrupt

Provides functions to enable/disable WWDT interrupts and get current enabled interrupts.

47.2.4 Watch dog Refresh

The function WWDT_Refresh() feeds the WWDT.

47.3 Typical use case

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

Files

- file fsl_wwdt.h

Data Structures

- struct wwdt_config_t
  Describes WWDT configuration structure. More...

Enumerations

- enum _wwdt_status_flags_t {
  kWWDT_TimeoutFlag = WWDT_MOD_WDTOF_MASK,
  kWWDT_WarningFlag = WWDT_MOD_WDINT_MASK }
  WWDT status flags.

Driver version

- #define FSL_WWDT_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))
  Defines WWDT driver version.

Refresh sequence

- #define WWDT_FIRST_WORD_OF_REFRESH (0xAAU)
  First word of refresh sequence.
- #define WWDT_SECOND_WORD_OF_REFRESH (0x55U)
  Second word of refresh sequence.

WWDT Initialization and De-initialization

- void WWDT_GetDefaultConfig (wwdt_config_t *config)
  Initializes WWDT configure structure.
- void WWDT_Init (WWDT_Type *base, const wwdt_config_t *config)
  Initializes the WWDT.
- void WWDT_Deinit (WWDT_Type *base)
  Shuts down the WWDT.

WWDT Functional Operation

- static void WWDT_Enable (WWDT_Type *base)
  Enables the WWDT module.
- static void WWDT_Disable (WWDT_Type *base)
  Disables the WWDT module.
- static uint32_t WWDT_GetStatusFlags (WWDT_Type *base)
  Gets all WWDT status flags.
- void WWDT_ClearStatusFlags (WWDT_Type *base, uint32_t mask)
  Clear WWDT flag.
- static void WWDT_SetWarningValue (WWDT_Type *base, uint32_t warningValue)
  Set the WWDT warning value.
- static void WWDT_SetTimeoutValue (WWDT_Type *base, uint32_t timeoutCount)
  Set the WWDT timeout value.
- static void WWDT_SetWindowValue (WWDT_Type *base, uint32_t windowValue)
  Sets the WWDT window value.
- void WWDT_Refresh (WWDT_Type *base)
  Refreshes the WWDT timer.

47.4 Data Structure Documentation

47.4.1 struct wwdt_config_t

Data Fields

- bool enableWwdt
  Enables or disables WWDT.
- bool enableWatchdogReset
  true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset
- bool enableWatchdogProtect
  true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time
- bool enableLockOscillator
  true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator
- uint32_t windowValue
  Window value, set this to 0xFFFFFF if windowing is not in effect.
- uint32_t timeoutValue
  Timeout value.
- uint32_t warningValue
  Watchdog time counter value that will generate a warning interrupt.
- uint32_t clockFreq_Hz
  Watchdog clock source frequency.

Field Documentation

(1) uint32_t wwdt_config_t::warningValue

Set this to 0 for no warning

(2) uint32_t wwdt_config_t::clockFreq_Hz

47.5 Macro Definition Documentation

47.5.1 #define FSL_WWDT_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))

47.6 Enumeration Type Documentation

47.6.1 enum _wwdt_status_flags_t

This structure contains the WWDT status flags for use in the WWDT functions.

Enumerator

kWWDT_TimeoutFlag Time-out flag, set when the timer times out.
kWWDT_WarningFlag Warning interrupt flag, set when timer is below the value WDWARNINT.

47.7 Function Documentation

47.7.1 void WWDT_GetDefaultConfig (wwdt_config_t * *config*)

This function initializes the WWDT configure structure to default value. The default value are:

```
*  config->enableWwdt = true;
*  config->enableWatchdogReset = false;
*  config->enableWatchdogProtect = false;
*  config->enableLockOscillator = false;
*  config->windowValue = 0xFFFFFFU;
*  config->timeoutValue = 0xFFFFFFU;
*  config->warningValue = 0;
*
```

Parameters

config	Pointer to WWDT config structure.

See Also

wwdt_config_t

47.7.2 void WWDT_Init (WWDT_Type * *base,* const wwdt_config_t * *config*)

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
*   wwdt_config_t config;
*   WWDT_GetDefaultConfig(&config);
*   config.timeoutValue = 0x7ffU;
*   WWDT_Init(wwdt_base,&config);
*
```

Parameters

base	WWDT peripheral base address
config	The configuration of WWDT

47.7.3 void WWDT_Deinit (WWDT_Type * *base*)

This function shuts down the WWDT.

Parameters

base	WWDT peripheral base address

47.7.4 static void WWDT_Enable (WWDT_Type * *base*) [inline], [static]

This function write value into WWDT_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

base	WWDT peripheral base address

47.7.5 static void WWDT_Disable (WWDT_Type * *base*) [inline], [static]

Deprecated Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT_MOD register to disable the WWDT.

Parameters

base	WWDT peripheral base address

47.7.6 static uint32_t WWDT_GetStatusFlags (WWDT_Type * *base*) [inline], [static]

This function gets all status flags.

Example for getting Timeout Flag:

```
*   uint32_t status;
*   status = WWDT_GetStatusFlags(wwdt_base) &
      kWWDT_TimeoutFlag;
*
```

Parameters

base	WWDT peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration _wwdt_status_flags_t

47.7.7 void WWDT_ClearStatusFlags (WWDT_Type * *base*, uint32_t *mask*)

This function clears WWDT status flag.

Example for clearing warning flag:

```
*   WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
*
```

Parameters

base	WWDT peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration _wwdt-_status_flags_t

47.7.8 static void WWDT_SetWarningValue (WWDT_Type * *base*, uint32_t *warningValue*) [inline], [static]

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

base	WWDT peripheral base address
warningValue	WWDT warning value.

47.7.9 static void WWDT_SetTimeoutValue (WWDT_Type * *base*, uint32_t *timeoutCount*) [inline], [static]

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC

register. Thus the minimum time-out interval is TWDCLK∗256∗4. If enableWatchdogProtect flag is true in wwdt_config_t config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

base	WWDT peripheral base address
timeoutCount	WWDT timeout value, count of WWDT clock tick.

47.7.10 static void WWDT_SetWindowValue (WWDT_Type * *base,* uint32_t *windowValue*) [inline], [static]

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set windowValue to 0xFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

base	WWDT peripheral base address
windowValue	WWDT window value.

47.7.11 void WWDT_Refresh (WWDT_Type * *base*)

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

base	WWDT peripheral base address

Chapter 48
Debug Console

48.1 Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data. The below picture shows the laylout of debug console.

Debug console overview

48.2 Function groups

48.2.1 Initialization

To initialize the debug console, call the DbgConsole_Init() function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate, serial_port_type_t
      device, uint32_t clkSrcFreq);
```

Select the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_Uart = 1U,
    kSerialPort_UsbCdc,
    kSerialPort_Swo,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral.

This example shows how to call the DbgConsole_Init() given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUDRATE, BOARD_DEBUG_UART_TYPE,
                            BOARD_DEBUG_UART_CLK_FREQ);
```

48.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	Description
.number	For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

length	Description
Do not support	

specifier	Description
d or i	Signed decimal integer
f	Decimal floating point
F	Decimal floating point capital letters
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer capital letters
o	Signed octal
b	Binary value
p	Pointer address
u	Unsigned decimal integer
c	Character
s	String of characters
n	Nothing printed

- Support a format specifier for SCANF following this prototype " %[∗][width][length]specifier", which is explained below

∗	Description
An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument.	

width	Description
This specifies the maximum number of characters to be read in the current reading operation.	

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

```
      version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
      toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */
```

48.2.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART

There are two macros SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART added to configure PRINTF and low level output perihperal.

- The macro SDK_DEBUGCONSOLE is used for forntend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The fucntion can be set by the macro SDK_DEBUGCONSOLE.
- The macro SDK_DEBUGCONSOLE_UART is used for backend. It is use to decide whether provide low level IO implementation to toolchain printf and scanf. For example, within MCU-Xpresso, 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.The macro does not specifically refer to the perihpheral "UART". It refers to the external perihperal similar to UART, like as USB CDC, UART, SWO, etc. So if the macro SDK_DEBUGCONSOLE_UART is not defined when tool-chain printf is calling, the semihosting will be used.

The following the matrix show the effects of SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_-UART on PRINTF and printf. The green mark is the default setting of the debug console.

SDK_DEBUGCONSOLE	SDK_DEBUGCONSOLE_UART	PRINTF	printf
DEBUGCONSOLE_-REDIRECT_TO_SDK	defined	Low level peripheral∗	Low level peripheral
DEBUGCONSOLE_-REDIRECT_TO_SDK	undefined	Low level peripheral∗	semihost
DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN	defined	Low level peripheral∗	Low level peripheral
DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN	undefined	semihost	semihost
DEBUGCONSOLE_-DISABLE	defined	No ouput	Low level peripheral
DEBUGCONSOLE_-DISABLE	undefined	No ouput	semihost

∗ the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

48.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

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

Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

Some examples use the SCANF function

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

Print out failure messages using MCUXpresso SDK __assert_func:

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

Note:

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

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.

48.4 Function groups

48.4.1 Initialization

To initialize the debug console, call the DbgConsole_Init() function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate, serial_port_type_t
      device, uint32_t clkSrcFreq);
```

Selects the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_None = 0U,
    kSerialPort_Uart = 1U,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. The debug console state is stored in the debug_console_state_t structure, such as shown here.

```
typedef struct DebugConsoleState
{
    uint8_t uartHandleBuffer[HAL_UART_HANDLE_SIZE];
    hal_uart_status_t (*putChar)(hal_uart_handle_t handle, const uint8_t *data, size_t length);
    hal_uart_status_t (*getChar)(hal_uart_handle_t handle, uint8_t *data, size_t length);
    serial_port_type_t type;
} debug_console_state_t;
```

This example shows how to call the DbgConsole_Init() given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_USART_INSTANCE, BOARD_DEBUG_USART_BAUDRATE,
      BOARD_DEBUG_USART_TYPE,
                              BOARD_DEBUG_USART_CLK_FREQ);
```

48.4.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	Description
.number	For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

length	Description
Do not support	

specifier	Description
d or i	Signed decimal integer
f	Decimal floating point
F	Decimal floating point capital letters
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer capital letters
o	Signed octal
b	Binary value
p	Pointer address
u	Unsigned decimal integer
c	Character
s	String of characters
n	Nothing printed

- Support a format specifier for SCANF following this prototype " %[∗][width][length]specifier", which is explained below

∗	Description
An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument.	

width	Description
This specifies the maximum number of characters to be read in the current reading operation.	

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char *
i	Integer: : Number optionally preceded with a + or - sign	int *
d	Decimal integer: Number optionally preceded with a + or - sign	int *
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float *
o	Octal Integer:	int *
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char *
u	Unsigned decimal integer.	unsigned int *

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

```
       version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
       toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */
```

48.4.3 SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART

There are two macros SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART added to configure PRINTF and low level output perihperal.

- The macro SDK_DEBUGCONSOLE is used for forntend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The fucntion can be set by the macro SDK_DEBUGCONSOLE.
- The macro SDK_DEBUGCONSOLE_UART is used for backend. It is use to decide whether provide low level IO implementation to toolchain printf and scanf. For example, within MCU-Xpresso, 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.The macro does not specifically refer to the perihpheral "UART". It refers to the external perihperal UART. So if the macro SDK_DEBUGCONSOLE_UART is not defined when tool-chain printf is calling, the semihosting will be used.

The following the matrix show the effects of SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_-UART on PRINTF and printf. The green mark is the default setting of the debug console.

SDK_DEBUGCONSOLE	SDK_DEBUGCONSOLE_UART	PRINTF	printf
DEBUGCONSOLE_-REDIRECT_TO_SDK	defined	UART	UART
DEBUGCONSOLE_-REDIRECT_TO_SDK	undefined	UART	semihost
DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN	defined	UART	UART
DEBUGCONSOLE_-REDIRECT_TO_TO-OLCHAIN	undefined	semihost	semihost
DEBUGCONSOLE_-DISABLE	defined	No ouput	UART
DEBUGCONSOLE_-DISABLE	undefined	No ouput	semihost

48.5 Typical use case

Some examples use the PUTCHAR & GETCHAR function

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

Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

Some examples use the SCANF function

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

Print out failure messages using MCUXpresso SDK __assert_func:

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

Note:

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

Modules

- SWO
- Semihosting

Macros

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

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

Initialization

- status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)
  Initializes the peripheral used for debug messages.
- status_t DbgConsole_Deinit (void)
  De-initializes the peripheral used for debug messages.
- status_t DbgConsole_EnterLowpower (void)
  Prepares to enter low power consumption.
- status_t DbgConsole_ExitLowpower (void)
  Restores from low power consumption.
- int DbgConsole_Printf (const char ∗fmt_s,...)
  Writes formatted output to the standard output stream.
- int DbgConsole_Vprintf (const char ∗fmt_s, va_list formatStringArg)
  Writes formatted output to the standard output stream.
- int DbgConsole_Putchar (int ch)

Writes a character to stdout.

- int DbgConsole_Scanf (char *fmt_s,...)
  Reads formatted data from the standard input stream.
- int DbgConsole_Getchar (void)
  Reads a character from standard input.
- int DbgConsole_BlockingPrintf (const char *fmt_s,...)
  Writes formatted output to the standard output stream with the blocking mode.
- int DbgConsole_BlockingVprintf (const char *fmt_s, va_list formatStringArg)
  Writes formatted output to the standard output stream with the blocking mode.
- status_t DbgConsole_Flush (void)
  Debug console flush.

48.6 Macro Definition Documentation

48.6.1 #define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U

Select toolchain printf and scanf.

48.6.2 #define DEBUGCONSOLE_REDIRECT_TO_SDK 1U

48.6.3 #define DEBUGCONSOLE_DISABLE 2U

48.6.4 #define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK

The macro only support to be redefined in project setting.

48.6.5 #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.

48.7 Function Documentation

48.7.1 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

instance	The instance of the module.If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UART adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART periheral is LPUART1. If the uart_adapter.c is added to the current project, the UART periheral is UART1.
baudRate	The desired baud rate in bits per second.
device	Low level device type for the debug console, can be one of the following. • kSerialPort_Uart, • kSerialPort_UsbCdc
clkSrcFreq	Frequency of peripheral source clock.

Returns

Indicates whether initialization was successful or not.

Return values

kStatus_Success	Execution successfully

48.7.2 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.

48.7.3 status_t DbgConsole_EnterLowpower (void)

This function is used to prepare to enter low power consumption.

Returns

Indicates whether de-initialization was successful or not.

48.7.4 status_t DbgConsole_ExitLowpower (void)

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

48.7.5 int DbgConsole_Printf (const char * *fmt_s,* ...)

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

Parameters

fmt_s	Format control string.

Returns

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

48.7.6 int DbgConsole_Vprintf (const char * *fmt_s,* va_list *formatStringArg*)

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

Parameters

fmt_s	Format control string.
formatString-Arg	Format arguments.

Returns

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

48.7.7 int DbgConsole_Putchar (int *ch*)

Call this function to write a character to stdout.

Parameters

ch	Character to be written.

Returns

Returns the character written.

48.7.8 int DbgConsole_Scanf (char * *fmt_s, ...*)

Call this function to read formatted data from the standard input stream.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Parameters

fmt_s	Format control string.

Returns

Returns the number of fields successfully converted and assigned.

48.7.9 int DbgConsole_Getchar (void)

Call this function to read a character from standard input.

Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function DbgConsole_TryGetchar to get the input char.

Returns

Returns the character read.

48.7.10 int DbgConsole_BlockingPrintf (const char * *fmt_s*, ...)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

fmt_s	Format control string.

Returns

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

48.7.11 int DbgConsole_BlockingVprintf (const char * *fmt_s*, va_list *formatStringArg*)

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set or not. The function could be used in system ISR mode with DEBUG_CONSOLE_TRANSFER_NON_BLOCKING set.

Parameters

fmt_s	Format control string.
formatString-Arg	Format arguments.

Returns

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

48.7.12 status_t DbgConsole_Flush (void)

Call this function to wait the tx buffer empty. If interrupt transfer is using, make sure the global IRQ is enable before call this function This function should be called when 1, before enter power down mode 2, log is required to print to terminal immediately

Returns

Indicates whether wait idle was successful or not.

48.7.13 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	*fmt*	Format string for printf.
in	*ap*	Arguments to printf.
in	*buf*	pointer to the buffer
	cb	print callbck function pointer

Returns

Number of characters to be print

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

Parameters

in	*line_ptr*	The input line of ASCII data.
in	*format*	Format first points to the format string.
in	*args_ptr*	The list of parameters.

Returns

Number of input items converted and assigned.

Return values

IO_EOF	When line_ptr is empty string "".

48.8 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.

48.8.1 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.

48.8.2 Guide Semihosting for Keil µVision

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

48.8.3 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.

48.8.4 Guide Semihosting for ARMGCC

Step 1: Setting up the environment

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

Add to "CMakeLists.txt"

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

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

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

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

Step 2: Building the project

1. Change "CMakeLists.txt":
   Change "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=nano.specs")"
   to "SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} –specs=rdimon.specs")"
   Replace paragraph
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-common")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffunction-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fdata-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -ffreestanding")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -fno-builtin")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mthumb")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -mapcs")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --gc-sections")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -static")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -z")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
   To
   SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
   Remove
   target_link_libraries(semihosting_ARMGCC.elf debug nosys)
2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = 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.

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.

48.8.5 Guide Semihosting for IAR

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

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.

48.8.6 Guide Semihosting for Keil µVision

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

48.8.7 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.

48.8.8 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_DEBU-

G} -z")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} muldefs")
To
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --specs=rdimon.specs ")
Remove
target_link_libraries(semihosting_ARMGCC.elf debug nosys)

2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = 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.

48.9 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.

48.9.1 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

48.9.1.1 Guide SWO for IAR

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

Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK,and click the Activate button to enable trace data collection.
6. There are three cases for this SDK_DEBUGCONSOLE_UART whether or not defined. a: if use uppercase PRINTF to output log,The SDK_DEBUGCONSOLE_UART defined or not defined will not effect debug function. b: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to zero,then debug function ok. c: if use lowercase printf to output log and defined SDK_DEBUGCONSOLE_UART to one,then debug function ok.

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

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

Step 2: Building the project

Step 3: Starting swo

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

48.9.2 Guide SWO for Keil µVision

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

Step 1: Setting up the environment

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

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

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

Step 3: Building the project

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

Step 4: Run the project

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

48.9.3 Guide SWO for MCUXpresso IDE

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

48.9.4 Guide SWO for ARMGCC

NOTE: ARMGCC has no library support SWO.

Chapter 49
Shell

49.1 Overview

This section describes the programming interface of the Shell middleware.

Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

49.2 Function groups

49.2.1 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>");
```

49.2.2 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.

49.2.3 Shell Operation

```
SHELL_Init(s_shellHandle, s_serialHandle, "Test@SHELL>");
SHELL_Task((s_shellHandle);
```

Data Structures

- struct shell_command_t
  User command data configuration structure. More...

Macros

- #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE
  Whether use non-blocking mode.
- #define SHELL_AUTO_COMPLETE (1U)
  Macro to set on/off auto-complete feature.
- #define SHELL_BUFFER_SIZE (64U)
  Macro to set console buffer size.
- #define SHELL_MAX_ARGS (8U)
  Macro to set maximum arguments in command.
- #define SHELL_HISTORY_COUNT (3U)
  Macro to set maximum count of history commands.
- #define SHELL_IGNORE_PARAMETER_COUNT (0xFF)
  Macro to bypass arguments check.
- #define SHELL_HANDLE_SIZE
  The handle size of the shell module.
- #define SHELL_USE_COMMON_TASK (0U)
  Macro to determine whether use common task.
- #define SHELL_TASK_PRIORITY (2U)
  Macro to set shell task priority.
- #define SHELL_TASK_STACK_SIZE (1000U)
  Macro to set shell task stack size.
- #define SHELL_HANDLE_DEFINE(name) uint32_t name[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the shell handle.
- #define SHELL_COMMAND_DEFINE(command, descriptor, callback, 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, const char *buffer, uint32_t length)
  Sends data to the shell output stream.
- int SHELL_Printf (shell_handle_t shellHandle, const char *formatString,...)
  Writes formatted output to the shell output stream.
- shell_status_t SHELL_WriteSynchronization (shell_handle_t shellHandle, const char *buffer, uint32_t length)
  Sends data to the shell output stream with OS synchronization.
- int SHELL_PrintfSynchronization (shell_handle_t shellHandle, const char *formatString,...)
  Writes formatted output to the shell output stream with OS synchronization.
- void SHELL_ChangePrompt (shell_handle_t shellHandle, char *prompt)
  Change shell prompt.
- void SHELL_PrintPrompt (shell_handle_t shellHandle)
  Print shell prompt.
- void SHELL_Task (shell_handle_t shellHandle)
  The task function for Shell.
- static bool SHELL_checkRunningInIsr (void)
  Check if code is running in ISR.

49.3 Data Structure Documentation

49.3.1 struct shell_command_t

Data Fields

- const char * pcCommand
  The command that is executed.
- char * pcHelpString
  String that describes how to use the command.
- const cmd_function_t pFuncCallBack
  A pointer to the callback function that returns the output generated by the command.
- uint8_t cExpectedNumberOfParameters
  Commands expect a fixed number of parameters, which may be zero.
- list_element_t link
  link of the element

Field Documentation

(1) const char* shell_command_t::pcCommand

For example "help". It must be all lower case.

(2) char∗ shell_command_t::pcHelpString

It should start with the command itself, and end with "\r\n". For example "help: Returns a list of all the commands\r\n".

(3) const cmd_function_t shell_command_t::pFuncCallBack

(4) uint8_t shell_command_t::cExpectedNumberOfParameters

49.4 Macro Definition Documentation

49.4.1 #define SHELL_NON_BLOCKING_MODE SERIAL_MANAGER_NON_BLOCKING_MODE

49.4.2 #define SHELL_AUTO_COMPLETE (1U)

49.4.3 #define SHELL_BUFFER_SIZE (64U)

49.4.4 #define SHELL_MAX_ARGS (8U)

49.4.5 #define SHELL_HISTORY_COUNT (3U)

49.4.6 #define SHELL_HANDLE_SIZE

Value:

```
(160U + SHELL_HISTORY_COUNT * SHELL_BUFFER_SIZE +
     SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + \
    SERIAL_MANAGER_WRITE_HANDLE_SIZE)
```

It is the sum of the SHELL_HISTORY_COUNT ∗ SHELL_BUFFER_SIZE + SHELL_BUFFER_SIZE + SERIAL_MANAGER_READ_HANDLE_SIZE + SERIAL_MANAGER_WRITE_HANDLE_SIZE

49.4.7 #define SHELL_USE_COMMON_TASK (0U)

49.4.8 #define SHELL_TASK_PRIORITY (2U)

49.4.9 #define SHELL_TASK_STACK_SIZE (1000U)

49.4.10 #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

name	The name string of the shell handle.

49.4.11 #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

command	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".

descriptor	The description of the command is used for showing the command usage when "help" is typing.
callback	The callback of the command is used to handle the command line when the input command is matched.
paramCount	The max parameter count of the current command.

49.4.12 #define SHELL_COMMAND(*command*) &g_shellCommand##command

This macro is used to get the shell command pointer. The macro should not be used before the macro SHELL_COMMAND_DEFINE is used.

Parameters

command	The command string of the command. The double quotes do not need. Such as exit for "exit", help for "Help", read for "read".

49.5 Typedef Documentation

49.5.1 typedef shell_status_t(∗ cmd_function_t)(shell_handle_t shellHandle, int32_t argc, char ∗∗argv)

49.6 Enumeration Type Documentation

49.6.1 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.

49.7 Function Documentation

49.7.1 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

shellHandle	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 uint32_t shellHandle[((SHELL_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];
serialHandle	The serial manager module handle pointer.
prompt	The string prompt pointer of Shell. Only the global variable can be passed.

Return values

kStatus_SHELL_Success	The shell initialization succeed.
kStatus_SHELL_Error	An error occurred when the shell is initialized.
kStatus_SHELL_OpenWriteHandleFailed	Open the write handle failed.
kStatus_SHELL_OpenReadHandleFailed	Open the read handle failed.

49.7.2 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

shellHandle	The shell module handle pointer.
shellCommand	The command element.

Return values

kStatus_SHELL_Success	Successfully register the command.
kStatus_SHELL_Error	An error occurred.

49.7.3 shell_status_t SHELL_UnregisterCommand (shell_command_t * *shellCommand*)

This function is used to unregister the shell command.

Parameters

shellCommand	The command element.

Return values

kStatus_SHELL_Success	Successfully unregister the command.

49.7.4 shell_status_t SHELL_Write (shell_handle_t *shellHandle,* const char * *buffer,* uint32_t *length*)

This function is used to send data to the shell output stream.

Parameters

shellHandle	The shell module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SHELL_Success	Successfully send data.
kStatus_SHELL_Error	An error occurred.

49.7.5 int SHELL_Printf (shell_handle_t *shellHandle,* const char * *formatString,* ...)

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

Parameters

shellHandle	The shell module handle pointer.
formatString	Format string.

Returns

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

49.7.6 shell_status_t SHELL_WriteSynchronization (shell_handle_t *shellHandle,* const char ∗ *buffer,* uint32_t *length*)

This function is used to send data to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

shellHandle	The shell module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SHELL_Success	Successfully send data.
kStatus_SHELL_Error	An error occurred.

49.7.7 int SHELL_PrintfSynchronization (shell_handle_t *shellHandle,* const char ∗ *formatString,* ...)

Call this function to write a formatted output to the shell output stream with OS synchronization, note the function could not be called in ISR.

Parameters

shellHandle	The shell module handle pointer.

formatString	Format string.

Returns

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

49.7.8 void SHELL_ChangePrompt (shell_handle_t *shellHandle,* char ∗ *prompt*)

Call this function to change shell prompt.

Parameters

shellHandle	The shell module handle pointer.
prompt	The string which will be used for command prompt

Returns

NULL.

49.7.9 void SHELL_PrintPrompt (shell_handle_t *shellHandle*)

Call this function to print shell prompt.

Parameters

shellHandle	The shell module handle pointer.

Returns

NULL.

49.7.10 void SHELL_Task (shell_handle_t *shellHandle*)

The task function for Shell; The function should be polled by upper layer. This function does not return until Shell command exit was called.

Parameters

shellHandle	The shell module handle pointer.

49.7.11 static bool SHELL_checkRunningInIsr (void) [inline], [static]

This function is used to check if code running in ISR.

Return values

TRUE	if code runing in ISR.

Chapter 50

Cards: Secure Digital Card/Embedded MultiMedia Card/SD-IO Card

50.1 Overview

The MCUXpresso SDK provides drivers to access the Secure Digital Card(up to v3.0), Embedded Multi-Media Card(up to v5.0) and sdio card(up to v3.0) based on the SDHC/USDHC/SDIF driver. Here is a simple block diagram about the drivers:

```
---------------------------
|       Application       |
---------------------------
            |
            v
---------------------------
|       Card driver       |
---------------------------
            |
            v
------------- -------------
|host driver| |OSA adapter|
------------- -------------
            |
            v
----------------------------
|  host controller driver  |
----------------------------
            |
            v
----------------------------
|host controller peripheral|
----------------------------
            |
            v
----------------------------
|          card            |
----------------------------
```

Modules

- MMC Card Driver
- SD Card Driver
- SDIO Card Driver
- SDMMC Common
- SDMMC HOST Driver
- SDMMC OSA

50.2 SDIO Card Driver

50.2.1 Overview

The SDIO card driver provide card initialization/IO direct and extend command interface.

50.2.2 SDIO CARD Operation

error log support

Not supported yet.

User configuable

Board dependency

Mutual exclusive access support for RTOS

SDIO driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
SDIO_Deinit(card);/* This function will destroy the created mutex */
SDIO_Init(card);
```

Typical use case

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

Data Structures

- struct sdio_card_t
  SDIO card state. More...

Macros

- #define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 0U)) /∗2.4.0∗/
  Middleware version.
- #define FSL_SDIO_MAX_IO_NUMS (7U)
  sdio device support maximum IO number

Typedefs

- typedef void(∗ sdio_io_irq_handler_t)(sdio_card_t ∗card, uint32_t func)
  sdio io handler

Enumerations

- enum sdio_io_direction_t {
  kSDIO_IORead = 0U,
  kSDIO_IOWrite = 1U }
  sdio io read/write direction

Initialization and deinitialization

- status_t SDIO_Init (sdio_card_t ∗card)
  SDIO card init function.
- void SDIO_Deinit (sdio_card_t ∗card)
  SDIO card deinit, include card and host deinit.
- status_t SDIO_CardInit (sdio_card_t ∗card)
  Initializes the card.
- void SDIO_CardDeinit (sdio_card_t ∗card)
  Deinitializes the card.
- status_t SDIO_HostInit (sdio_card_t ∗card)
  initialize the host.
- void SDIO_HostDeinit (sdio_card_t ∗card)
  Deinitializes the host.
- void SDIO_HostDoReset (sdio_card_t ∗card)
  reset the host.
- void SDIO_SetCardPower (sdio_card_t ∗card, bool enable)
  set card power.
- status_t SDIO_CardInActive (sdio_card_t ∗card)
  set SDIO card to inactive state
- status_t SDIO_GetCardCapability (sdio_card_t ∗card, sdio_func_num_t func)
  get SDIO card capability
- status_t SDIO_SetBlockSize (sdio_card_t ∗card, sdio_func_num_t func, uint32_t blockSize)
  set SDIO card block size
- status_t SDIO_CardReset (sdio_card_t ∗card)
  set SDIO card reset
- status_t SDIO_SetDataBusWidth (sdio_card_t ∗card, sdio_bus_width_t busWidth)
  set SDIO card data bus width
- status_t SDIO_SwitchToHighSpeed (sdio_card_t ∗card)
  switch the card to high speed
- status_t SDIO_ReadCIS (sdio_card_t ∗card, sdio_func_num_t func, const uint32_t ∗tupleList, uint32_t tupleNum)
  read SDIO card CIS for each function
- status_t SDIO_PollingCardInsert (sdio_card_t ∗card, uint32_t status)
  sdio wait card detect function.
- bool SDIO_IsCardPresent (sdio_card_t ∗card)

sdio card present check function.

IO operations

- status_t SDIO_IO_Write_Direct (sdio_card_t *card, sdio_func_num_t func, uint32_t regAddr, uint8_t *data, bool raw)
  IO direct write transfer function.
- status_t SDIO_IO_Read_Direct (sdio_card_t *card, sdio_func_num_t func, uint32_t regAddr, uint8_t *data)
  IO direct read transfer function.
- status_t SDIO_IO_RW_Direct (sdio_card_t *card, sdio_io_direction_t direction, sdio_func_num_t func, uint32_t regAddr, uint8_t dataIn, uint8_t *dataOut)
  IO direct read/write transfer function.
- status_t SDIO_IO_Write_Extended (sdio_card_t *card, sdio_func_num_t func, uint32_t regAddr, uint8_t *buffer, uint32_t count, uint32_t flags)
  IO extended write transfer function.
- status_t SDIO_IO_Read_Extended (sdio_card_t *card, sdio_func_num_t func, uint32_t regAddr, uint8_t *buffer, uint32_t count, uint32_t flags)
  IO extended read transfer function.
- status_t SDIO_EnableIOInterrupt (sdio_card_t *card, sdio_func_num_t func, bool enable)
  enable IO interrupt
- status_t SDIO_EnableIO (sdio_card_t *card, sdio_func_num_t func, bool enable)
  enable IO and wait IO ready
- status_t SDIO_SelectIO (sdio_card_t *card, sdio_func_num_t func)
  select IO
- status_t SDIO_AbortIO (sdio_card_t *card, sdio_func_num_t func)
  Abort IO transfer.
- status_t SDIO_SetDriverStrength (sdio_card_t *card, sd_driver_strength_t driverStrength)
  Set driver strength.
- status_t SDIO_EnableAsyncInterrupt (sdio_card_t *card, bool enable)
  Enable/Disable Async interrupt.
- status_t SDIO_GetPendingInterrupt (sdio_card_t *card, uint8_t *pendingInt)
  Get pending interrupt.
- status_t SDIO_IO_Transfer (sdio_card_t *card, sdio_command_t cmd, uint32_t argument, uint32_t blockSize, uint8_t *txData, uint8_t *rxData, uint16_t dataSize, uint32_t *response)
  sdio card io transfer function.
- void SDIO_SetIOIRQHandler (sdio_card_t *card, sdio_func_num_t func, sdio_io_irq_handler_-t handler)
  sdio set io IRQ handler.
- status_t SDIO_HandlePendingIOInterrupt (sdio_card_t *card)
  sdio card io pending interrupt handle function.

50.2.3 Data Structure Documentation

50.2.3.1 struct _sdio_card

sdio card descriptor

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- sdmmchost_t * host
  Host information.
- sdio_usr_param_t usrParam
  user parameter
- bool noInternalAlign
  use this flag to disable sdmmc align.
- uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]
  internal buffer
- bool isHostReady
  use this flag to indicate if need host re-init or not
- bool memPresentFlag
  indicate if memory present
- uint32_t busClock_Hz
  SD bus clock frequency united in Hz.
- uint32_t relativeAddress
  Relative address of the card.
- uint8_t sdVersion
  SD version.
- sd_timing_mode_t currentTiming
  current timing mode
- sd_driver_strength_t driverStrength
  driver strength
- sd_max_current_t maxCurrent
  card current limit
- sdmmc_operation_voltage_t operationVoltage
  card operation voltage
- uint8_t sdioVersion
  SDIO version.
- uint8_t cccrVersioin
  CCCR version.
- uint8_t ioTotalNumber
  total number of IO function
- uint32_t cccrflags
  Flags in _sd_card_flag.
- uint32_t io0blockSize
  record the io0 block size
- uint32_t ocr
  Raw OCR content, only 24bit avalible for SDIO card.
- uint32_t commonCISPointer
  point to common CIS
- sdio_common_cis_t commonCIS
  CIS table.
- sdio_fbr_t ioFBR [FSL_SDIO_MAX_IO_NUMS]
  FBR table.
- sdio_func_cis_t funcCIS [FSL_SDIO_MAX_IO_NUMS]
  function CIS table
- sdio_io_irq_handler_t ioIRQHandler [FSL_SDIO_MAX_IO_NUMS]

io IRQ handler
- uint8_t ioIntIndex
  used to record current enabled io interrupt index
- uint8_t ioIntNums
  used to record total enabled io interrupt numbers
- sdmmc_osa_mutex_t lock
  card access lock

Field Documentation

(1) bool sdio_card_t::noInternalAlign

If disable, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

50.2.4 Macro Definition Documentation

50.2.4.1 #define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 0U)) /∗2.4.0∗/

50.2.5 Enumeration Type Documentation

50.2.5.1 enum sdio_io_direction_t

Enumerator

kSDIO_IORead io read
kSDIO_IOWrite io write

50.2.6 Function Documentation

50.2.6.1 status_t SDIO_Init (sdio_card_t ∗ *card*)

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDIO_Deinit(card);
* SDIO_Init(card);
*
```

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_GoIdleFailed	
kStatus_SDMMC_HandShakeOperationConditionFailed	
kStatus_SDMMC_SDIO_InvalidCard	
kStatus_SDMMC_SDIO_InvalidVoltage	
kStatus_SDMMC_SendRelativeAddressFailed	
kStatus_SDMMC_SelectCardFailed	
kStatus_SDMMC_SDIO_SwitchHighSpeedFail	
kStatus_SDMMC_SDIO_ReadCISFail	
kStatus_SDMMC_TransferFailed	
kStatus_Success	

50.2.6.2 void SDIO_Deinit (sdio_card_t * *card*)

Please note it is a thread safe function.

Parameters

card	Card descriptor.

50.2.6.3 status_t SDIO_CardInit (sdio_card_t * *card*)

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return kStatus_SDMMC_HostNotReady.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDIO_CardDeinit(card);
* SDIO_CardInit(card);
*
```

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_HostNotReady	host is not ready.
kStatus_SDMMC_GoIdleFailed	Go idle failed.
kStatus_SDMMC_NotSupportYet	Card not support.
kStatus_SDMMC_SendOperationConditionFailed	Send operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Send CID failed.
kStatus_SDMMC_SendRelativeAddressFailed	Send relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendScrFailed	Send SCR failed.
kStatus_SDMMC_SetBusWidthFailed	Set bus width failed.
kStatus_SDMMC_SwitchHighSpeedFailed	Switch high speed failed.

kStatus_SDMMC_SetCardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

50.2.6.4 void SDIO_CardDeinit (sdio_card_t * *card*)

This function deinitializes the specific card.

Please note it is a thread safe function.

Parameters

card	Card descriptor.

50.2.6.5 status_t SDIO_HostInit (sdio_card_t * *card*)

This function deinitializes the specific host.

Parameters

card	Card descriptor.

50.2.6.6 void SDIO_HostDeinit (sdio_card_t * *card*)

This function deinitializes the host.

Parameters

card	Card descriptor.

50.2.6.7 void SDIO_HostDoReset (sdio_card_t * *card*)

This function reset the specific host.

Parameters

card	Card descriptor.

50.2.6.8 void SDIO_SetCardPower (sdio_card_t * *card,* bool *enable*)

The power off operation depend on host or the user define power on function.

Parameters

card	card descriptor.
enable	true is power on, false is power off.

50.2.6.9 status_t SDIO_CardInActive (sdio_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.10 status_t SDIO_GetCardCapability (sdio_card_t * *card,* sdio_func_num_t *func*)

Parameters

card	Card descriptor.
func	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.11 status_t SDIO_SetBlockSize (sdio_card_t * *card,* sdio_func_num_t *func,* uint32_t *blockSize*)

Parameters

card	Card descriptor.
func	io number
blockSize	block size

Return values

kStatus_SDMMC_SetCardBlockSizeFailed	
kStatus_SDMMC_SDIO_InvalidArgument	
kStatus_Success	

50.2.6.12 status_t SDIO_CardReset (sdio_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_TransferFailed	
kStatus_Success	

50.2.6.13 status_t SDIO_SetDataBusWidth (sdio_card_t * *card,* sdio_bus_width_t *busWidth*)

Parameters

card	Card descriptor.
busWidth	bus width

Return values

kStatus_SDMMC_TransferFailed	
kStatus_Success	

50.2.6.14 status_t SDIO_SwitchToHighSpeed (sdio_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_SwitchHighSpeedFail	
kStatus_Success	

50.2.6.15 status_t SDIO_ReadCIS (sdio_card_t ∗ *card,* sdio_func_num_t *func,* const uint32_t ∗ *tupleList,* uint32_t *tupleNum*)

Parameters

card	Card descriptor.
func	io number
tupleList	code list
tupleNum	code number

Return values

kStatus_SDMMC_SDIO-_ReadCISFail	
kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.16 status_t SDIO_PollingCardInsert (sdio_card_t ∗ *card,* uint32_t *status*)

Detect card through GPIO, CD, DATA3.

Parameters

card	card descriptor.
status	detect status, kSD_Inserted or kSD_Removed.

50.2.6.17 bool SDIO_IsCardPresent (sdio_card_t ∗ *card*)

Parameters

card	card descriptor.

50.2.6.18 status_t SDIO_IO_Write_Direct (sdio_card_t ∗ *card,* sdio_func_num_t *func,* uint32_t *regAddr,* uint8_t ∗ *data,* bool *raw*)

Please note it is a thread safe function.

Parameters

card	Card descriptor.
func	IO numner
regAddr	register address
data	the data pinter to write
raw	flag, indicate read after write or write only

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.19 status_t SDIO_IO_Read_Direct (sdio_card_t ∗ *card,* sdio_func_num_t *func,* uint32_t *regAddr,* uint8_t ∗ *data*)

Please note it is a thread safe function.

Parameters

card	Card descriptor.
func	IO number
regAddr	register address
data	pointer to read

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.20 status_t SDIO_IO_RW_Direct (sdio_card_t * *card,* sdio_io_direction_t *direction,* sdio_func_num_t *func,* uint32_t *regAddr,* uint8_t *dataIn,* uint8_t * *dataOut*)

Please note it is a thread safe function.

Parameters

card	Card descriptor.
direction	io access direction, please reference sdio_io_direction_t.
func	IO number
regAddr	register address
dataIn	data to write
dataOut	data pointer for readback data, support both for read and write, when application want readback the data after write command, dataOut should not be NULL.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.21 status_t SDIO_IO_Write_Extended (sdio_card_t * *card,* sdio_func_num_t *func,* uint32_t *regAddr,* uint8_t * *buffer,* uint32_t *count,* uint32_t *flags*)

Please note it is a thread safe function.

Parameters

card	Card descriptor.
func	IO number
regAddr	register address
buffer	data buffer to write
count	data count
flags	write flags

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_InvalidArgument	
kStatus_Success	

50.2.6.22 status_t SDIO_IO_Read_Extended (sdio_card_t ∗ *card,* sdio_func_num_t *func,* uint32_t *regAddr,* uint8_t ∗ *buffer,* uint32_t *count,* uint32_t *flags*)

Please note it is a thread safe function.

Parameters

card	Card descriptor.
func	IO number
regAddr	register address
buffer	data buffer to read
count	data count
flags	write flags

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_InvalidArgument	
kStatus_Success	

50.2.6.23 status_t SDIO_EnableIOInterrupt (sdio_card_t ∗ *card,* sdio_func_num_t *func,* bool *enable*)

Parameters

card	Card descriptor.
func	IO number
enable	enable/disable flag

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.24 status_t SDIO_EnableIO (sdio_card_t * *card,* sdio_func_num_t *func,* bool *enable*)

Parameters

card	Card descriptor.
func	IO number
enable	enable/disable flag

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.25 status_t SDIO_SelectIO (sdio_card_t * *card,* sdio_func_num_t *func*)

Parameters

card	Card descriptor.
func	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.26 status_t SDIO_AbortIO (sdio_card_t * *card,* sdio_func_num_t *func*)

Parameters

card	Card descriptor.
func	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.27 status_t SDIO_SetDriverStrength (sdio_card_t ∗ *card,* sd_driver_strength_t *driverStrength*)

Parameters

card	Card descriptor.
driverStrength	target driver strength.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.28 status_t SDIO_EnableAsyncInterrupt (sdio_card_t ∗ *card,* bool *enable*)

Parameters

card	Card descriptor.
enable	true is enable, false is disable.

Return values

kStatus_SDMMC_-TransferFailed	

kStatus_Success	

50.2.6.29 status_t SDIO_GetPendingInterrupt (sdio_card_t * *card*, uint8_t * *pendingInt*)

Parameters

card	Card descriptor.
pendingInt	pointer store pending interrupt

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.2.6.30 status_t SDIO_IO_Transfer (sdio_card_t * *card*, sdio_command_t *cmd*, uint32_t *argument*, uint32_t *blockSize*, uint8_t * *txData*, uint8_t * *rxData*, uint16_t *dataSize*, uint32_t * *response*)

This function can be used for trnansfer direct/extend command. Please pay attention to the non-align data buffer address transfer, if data buffer address can not meet host controller internal DMA requirement, sdio driver will try to use internal align buffer if data size is not bigger than internal buffer size, Align address transfer always can get a better performance, so if application want sdio driver make sure buffer address align,

Please note it is a thread safe function.

Parameters

card	card descriptor.
cmd	command to transfer
argument	argument to transfer
blockSize	used for block mode.
txData	tx buffer pointer or NULL

rxData	rx buffer pointer or NULL
dataSize	transfer data size
response	reponse pointer, if application want read response back, please set it to a NON-NULL pointer.

50.2.6.31 void SDIO_SetIOIRQHandler (sdio_card_t * *card*, sdio_func_num_t *func*, sdio_io_irq_handler_t *handler*)

Parameters

card	card descriptor.
func	function io number.
handler	io IRQ handler.

50.2.6.32 status_t SDIO_HandlePendingIOInterrupt (sdio_card_t * *card*)

This function is used to handle the pending io interrupt. To reigster a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, true);
* SDIO_SetIOIRQHandler(card, 0, func0_handler);
*
```

call it in interrupt callback

```
* SDIO_HandlePendingIOInterrupt(card);
*
```

To releae a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, false);
* SDIO_SetIOIRQHandler(card, 0, NULL);
*
```

Parameters

card	card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

50.3 SD Card Driver

50.3.1 Overview

The SDCARD driver provide card initialization/read/write/erase interface.

50.3.2 SD CARD Operation

error log support

Lots of error log has been added to sd relate functions, if error occurs during initial/read/write, please enable the error log print functionality with `#define` SDMMC_ENABLE_LOG_PRINT 1 And rerun the project then user can check what kind of error happened.

User configurable

```
typedef struct _sd_card
{
    sdmmchost_t *host;
    sd_usr_param_t usrParam;
    bool isHostReady;
    bool noInteralAlign;
    uint32_t busClock_Hz;
    uint32_t relativeAddress;
    uint32_t version;
    uint32_t flags;
    uint8_t internalBuffer[FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE];
    uint32_t ocr;
    sd_cid_t cid;
    sd_csd_t csd;
    sd_scr_t scr;
    sd_status_t stat;
    uint32_t blockCount;
    uint32_t blockSize;
    sd_timing_mode_t currentTiming;
    sd_driver_strength_t driverStrength;
    sd_max_current_t maxCurrent;
    sdmmc_operation_voltage_t operationVoltage;
    sdmmc_osa_mutex_t lock;
} sd_card_t;
```

Part of The variables above is user configurable,

1. host
   Application need to provide host controller base address and the host's source clock frequency, etc. For example:

   ```
   /* allocate dma descriptor buffer for host controller */
   s_host.dmaDesBuffer                                    = s_sdmmcHostDmaBuffer;
   s_host.dmaDesBufferWordsNum                            = xxx;
   /*  */
   ((sd_card_t *)card)->host                              = &s_host;
   ((sd_card_t *)card)->host->hostController.base         = BOARD_SDMMC_SD_HOST_BASEADDR;
   ((sd_card_t *)card)->host->hostController.sourceClock_Hz = BOARD_USDHC1ClockConfiguration();

   /* allocate resource for sdmmc osa layer */
   ((sd_card_t *)card)->host->hostEvent     = &s_event;
   ```

2. sdcard_usr_param_t usrParam

```
/* board layer configuration register */
((sd_card_t *)card)->usrParam.cd         = &s_cd;
((sd_card_t *)card)->usrParam.pwr        = BOARD_SDCardPowerControl;
((sd_card_t *)card)->usrParam.ioStrength = BOARD_SD_Pin_Config;
((sd_card_t *)card)->usrParam.ioVoltage  = &s_ioVoltage;
((sd_card_t *)card)->usrParam.maxFreq    = BOARD_SDMMC_SD_HOST_SUPPORT_SDR104_FREQ;

a. cd-which allow application define the card insert/remove callback function, redefine the card detect
      timeout ms and also allow application determine how to detect card.
b. pwr-which allow application redefine the card power on/off function.
c. ioStrength-which is used to switch the signal pin configurations include driver strength/speed mode
      dynamiclly for different timing(SDR/HS timing) mode, reference the function defined sdmmc_config.c
d. ioVoltage-which allow application register io voltage switch function instead of using the function host
       driver provided for SDR/HS200/HS400 timing.
e. maxFreq-which allow application set the maximum bus clock that the board support.
```

1. bool noInteralAlign
   Sdmmc include an address align internal buffer(to use host controller internal DMA), to improve read/write performance while application cannot make sure the data address used to read/write is align, set it to true will achieve a better performance.
2. sd_timing_mode_t currentTiming
   It is used to indicate the currentTiming the card is working on, however sdmmc also support preset timing mode, then sdmmc will try to switch to this timing first, if failed, a valid timing will switch to automatically. Generally, user may not set this variable if you don't know what kind of timing the card support, sdmmc will switch to the highest timing which the card support.
3. sd_driver_strength_t driverStrength
   Choose a valid card driver strength if application required and call SD_SetDriverStrength in application.
4. sd_max_current_t maxCurrent
   Choose a valid card current if application required and call SD_SetMaxCurrent in application.

Mutual exclusive access support for RTOS

SDCARD driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
SD_Deinit(card);/* This function will destroy the created mutex */
SD_Init(card);
```

Typical use case

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

Data Structures

- struct sd_card_t
  SD card state. More...

Macros

- #define FSL_SD_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 0U)) /∗2.4.0∗/
  Driver version.

Enumerations

- enum {
  kSD_SupportHighCapacityFlag = (1U << 1U),
  kSD_Support4BitWidthFlag = (1U << 2U),
  kSD_SupportSdhcFlag = (1U << 3U),
  kSD_SupportSdxcFlag = (1U << 4U),
  kSD_SupportVoltage180v = (1U << 5U),
  kSD_SupportSetBlockCountCmd = (1U << 6U),
  kSD_SupportSpeedClassControlCmd = (1U << 7U) }
  SD card flags.

SDCARD Function

- status_t SD_Init (sd_card_t ∗card)
  Initializes the card on a specific host controller.
- void SD_Deinit (sd_card_t ∗card)
  Deinitializes the card.
- status_t SD_CardInit (sd_card_t ∗card)
  Initializes the card.
- void SD_CardDeinit (sd_card_t ∗card)
  Deinitializes the card.
- status_t SD_HostInit (sd_card_t ∗card)
  initialize the host.
- void SD_HostDeinit (sd_card_t ∗card)
  Deinitializes the host.
- void SD_HostDoReset (sd_card_t ∗card)
  reset the host.
- void SD_SetCardPower (sd_card_t ∗card, bool enable)
  set card power.
- status_t SD_PollingCardInsert (sd_card_t ∗card, uint32_t status)
  sd wait card detect function.
- bool SD_IsCardPresent (sd_card_t ∗card)
  sd card present check function.
- bool SD_CheckReadOnly (sd_card_t ∗card)
  Checks whether the card is write-protected.
- status_t SD_SelectCard (sd_card_t ∗card, bool isSelected)
  Send SELECT_CARD command to set the card to be transfer state or not.
- status_t SD_ReadStatus (sd_card_t ∗card)
  Send ACMD13 to get the card current status.
- status_t SD_ReadBlocks (sd_card_t ∗card, uint8_t ∗buffer, uint32_t startBlock, uint32_t blockCount)

Reads blocks from the specific card.

- status_t SD_WriteBlocks (sd_card_t *card, const uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)
  Writes blocks of data to the specific card.
- status_t SD_EraseBlocks (sd_card_t *card, uint32_t startBlock, uint32_t blockCount)
  Erases blocks of the specific card.
- status_t SD_SetDriverStrength (sd_card_t *card, sd_driver_strength_t driverStrength)
  select card driver strength select card driver strength
- status_t SD_SetMaxCurrent (sd_card_t *card, sd_max_current_t maxCurrent)
  select max current select max operation current
- status_t SD_PollingCardStatusBusy (sd_card_t *card, uint32_t timeoutMs)
  Polling card idle status.

50.3.3 Data Structure Documentation

50.3.3.1 struct sd_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- sdmmchost_t * host
  Host configuration.
- sd_usr_param_t usrParam
  user parameter
- bool isHostReady
  use this flag to indicate if need host re-init or not
- bool noInteralAlign
  used to enable/disable the functionality of the exchange buffer
- uint32_t busClock_Hz
  SD bus clock frequency united in Hz.
- uint32_t relativeAddress
  Relative address of the card.
- uint32_t version
  Card version.
- uint32_t flags
  Flags in _sd_card_flag.
- uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]
  internal buffer
- uint32_t ocr
  Raw OCR content.
- sd_cid_t cid
  CID.
- sd_csd_t csd
  CSD.
- sd_scr_t scr
  SCR.
- sd_status_t stat

sd 512 bit status
- uint32_t blockCount
  Card total block number.
- uint32_t blockSize
  Card block size.
- sd_timing_mode_t currentTiming
  current timing mode
- sd_driver_strength_t driverStrength
  driver strength
- sd_max_current_t maxCurrent
  card current limit
- sdmmc_operation_voltage_t operationVoltage
  card operation voltage
- sdmmc_osa_mutex_t lock
  card access lock

50.3.4 Macro Definition Documentation

50.3.4.1 #define FSL_SD_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 0U)) /∗2.4.0∗/

50.3.5 Enumeration Type Documentation

50.3.5.1 anonymous enum

Enumerator

kSD_SupportHighCapacityFlag Support high capacity.
kSD_Support4BitWidthFlag Support 4-bit data width.
kSD_SupportSdhcFlag Card is SDHC.
kSD_SupportSdxcFlag Card is SDXC.
kSD_SupportVoltage180v card support 1.8v voltage
kSD_SupportSetBlockCountCmd card support cmd23 flag
kSD_SupportSpeedClassControlCmd card support speed class control flag

50.3.6 Function Documentation

50.3.6.1 status_t SD_Init (sd_card_t ∗ *card*)

This function initializes the card on a specific host controller, it is consist of host init, card detect, card init function, however user can ignore this high level function, instead of use the low level function, such as SD_CardInit, SD_HostInit, SD_CardDetect.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SD_Deinit(card);
```

```
* SD_Init(card);
*
```

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_HostNotReady	host is not ready.
kStatus_SDMMC_GoIdleFailed	Go idle failed.
kStatus_SDMMC_NotSupportYet	Card not support.
kStatus_SDMMC_HandShakeOperationConditionFailed	Send operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Send CID failed.
kStatus_SDMMC_SendRelativeAddressFailed	Send relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendScrFailed	Send SCR failed.
kStatus_SDMMC_SetDataBusWidthFailed	Set bus width failed.
kStatus_SDMMC_SwitchBusTimingFailed	Switch high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Set card block size failed.

kStatus_Success	Operate successfully.

50.3.6.2 void SD_Deinit (sd_card_t * *card*)

This function deinitializes the specific card and host. Please note it is a thread safe function.

Parameters

card	Card descriptor.

50.3.6.3 status_t SD_CardInit (sd_card_t * *card*)

This function initializes the card only, make sure the host is ready when call this function, otherwise it will return kStatus_SDMMC_HostNotReady.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SD_CardDeinit(card);
* SD_CardInit(card);
*
```

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_HostNotReady	host is not ready.
kStatus_SDMMC_GoIdleFailed	Go idle failed.
kStatus_SDMMC_NotSupportYet	Card not support.
kStatus_SDMMC_HandShakeOperationConditionFailed	Send operation condition failed.

kStatus_SDMMC_AllSendCidFailed	Send CID failed.
kStatus_SDMMC_SendRelativeAddressFailed	Send relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendScrFailed	Send SCR failed.
kStatus_SDMMC_SetDataBusWidthFailed	Set bus width failed.
kStatus_SDMMC_SwitchBusTimingFailed	Switch high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

50.3.6.4 void SD_CardDeinit (sd_card_t * *card*)

This function deinitializes the specific card. Please note it is a thread safe function.

Parameters

card	Card descriptor.

50.3.6.5 status_t SD_HostInit (sd_card_t * *card*)

This function deinitializes the specific host.

Parameters

card	Card descriptor.

50.3.6.6 void SD_HostDeinit (sd_card_t * *card*)

This function deinitializes the host.

Parameters

card	Card descriptor.

50.3.6.7 void SD_HostDoReset (sd_card_t * *card*)

This function reset the specific host.

Parameters

card	Card descriptor.

50.3.6.8 void SD_SetCardPower (sd_card_t * *card,* bool *enable*)

The power off operation depend on host or the user define power on function.

Parameters

card	card descriptor.
enable	true is power on, false is power off.

50.3.6.9 status_t SD_PollingCardInsert (sd_card_t * *card,* uint32_t *status*)

Detect card through GPIO, CD, DATA3.

Parameters

card	card descriptor.
status	detect status, kSD_Inserted or kSD_Removed.

50.3.6.10 bool SD_IsCardPresent (sd_card_t * *card*)

Parameters

card	card descriptor.

50.3.6.11 bool SD_CheckReadOnly (sd_card_t * *card*)

This function checks if the card is write-protected via the CSD register.

Parameters

card	The specific card.

Return values

true	Card is read only.
false	Card isn't read only.

50.3.6.12 status_t SD_SelectCard (sd_card_t * *card*, bool *isSelected*)

Parameters

card	Card descriptor.
isSelected	True to set the card into transfer state, false to disselect.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

50.3.6.13 status_t SD_ReadStatus (sd_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Send-ApplicationCommand-Failed	send application command failed.

kStatus_Success	Operate successfully.

50.3.6.14 status_t SD_ReadBlocks (sd_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

This function reads blocks from the specific card with default block size defined by the SDHC_CARD_-DEFAULT_BLOCK_SIZE.

Please note it is a thread safe function.

Parameters

card	Card descriptor.
buffer	The buffer to save the data read from card.
startBlock	The start block index.
blockCount	The number of blocks to read.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_Card-NotSupport	Card not support.
kStatus_SDMMC_Not-SupportYet	Not support now.
kStatus_SDMMC_Wait-WriteCompleteFailed	Send status failed.
kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Stop-TransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

50.3.6.15 status_t SD_WriteBlocks (sd_card_t * *card*, const uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

This function writes blocks to the specific card with default block size 512 bytes.

Please note,

1. It is a thread safe function.
2. It is a async write function which means that the card status may still busy after the function return.

Application can call function SD_PollingCardStatusBusy to wait card status idle after the write operation.

Parameters

card	Card descriptor.
buffer	The buffer holding the data to be written to the card.
startBlock	The start block index.
blockCount	The number of blocks to write.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_NotSupportYet	Not support now.
kStatus_SDMMC_CardNotSupport	Card not support.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_StopTransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

50.3.6.16 status_t SD_EraseBlocks (sd_card_t * *card*, uint32_t *startBlock*, uint32_t *blockCount*)

This function erases blocks of the specific card with default block size 512 bytes.

Please note,

1. It is a thread safe function.
2. It is a async erase function which means that the card status may still busy after the function return. Application can call function SD_PollingCardStatusBusy to wait card status idle after the erase operation.

Parameters

card	Card descriptor.
startBlock	The start block index.
blockCount	The number of blocks to erase.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_Success	Operate successfully.

50.3.6.17 status_t SD_SetDriverStrength (sd_card_t * *card*, sd_driver_strength_t *driverStrength*)

Parameters

card	Card descriptor.
driverStrength	Driver strength

50.3.6.18 status_t SD_SetMaxCurrent (sd_card_t * *card*, sd_max_current_t *maxCurrent*)

Parameters

card	Card descriptor.
maxCurrent	Max current

50.3.6.19 status_t SD_PollingCardStatusBusy (sd_card_t * *card*, uint32_t *timeoutMs*)

This function can be used to polling the status from busy to Idle, the function will return if the card status idle or timeout.

Parameters

card	Card descriptor.
timeoutMs	polling card status timeout value.

Return values

kStatus_Success	Operate successfully.
kStatus_SDMMC_WaitWriteCompleteFailed	CMD13 transfer failed.
kStatus_SDMMC_PollingCardIdleFailed,polling	card DAT0 idle failed.

50.4 MMC Card Driver

50.4.1 Overview

The MMCCARD driver provide card initialization/read/write/erase interface.

50.4.2 MMC CARD Operation

error log support

Not support yet

User configuable

Board dependency

Mutual exclusive access support for RTOS

MMCCARD driver has added mutual exclusive access support for init/deinit/write/read/erase function. Please note that the card init function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization.

```
MMC_Deinit(card);/* This function will destroy the created mutex */
MMC_Init(card);
```

Typical use case

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

Data Structures

- struct mmc_usr_param_t
  card user parameter More...
- struct mmc_card_t
  mmc card state More...

Macros

- #define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 5U, 0U)) /∗2.5.0∗/
  Middleware mmc version.

Typedefs

- typedef void(∗ mmc_io_strength_t)(uint32_t busFreq)
  card io strength control

Enumerations

- enum {
  kMMC_SupportHighSpeed26MHZFlag = (1U << 0U),
  kMMC_SupportHighSpeed52MHZFlag = (1U << 1U),
  kMMC_SupportHighSpeedDDR52MHZ180V300VFlag = (1 << 2U),
  kMMC_SupportHighSpeedDDR52MHZ120VFlag = (1 << 3U),
  kMMC_SupportHS200200MHZ180VFlag = (1 << 4U),
  kMMC_SupportHS200200MHZ120VFlag = (1 << 5U),
  kMMC_SupportHS400DDR200MHZ180VFlag = (1 << 6U),
  kMMC_SupportHS400DDR200MHZ120VFlag = (1 << 7U),
  kMMC_SupportHighCapacityFlag = (1U << 8U),
  kMMC_SupportAlternateBootFlag = (1U << 9U),
  kMMC_SupportDDRBootFlag = (1U << 10U),
  kMMC_SupportHighSpeedBootFlag = (1U << 11U),
  kMMC_SupportEnhanceHS400StrobeFlag = (1U << 12U) }
  MMC card flags.
- enum mmc_sleep_awake_t {
  kMMC_Sleep = 1U,
  kMMC_Awake = 0U }
  mmccard sleep/awake state

MMCCARD Function

- status_t MMC_Init (mmc_card_t ∗card)
  Initializes the MMC card and host.
- void MMC_Deinit (mmc_card_t ∗card)
  Deinitializes the card and host.
- status_t MMC_CardInit (mmc_card_t ∗card)
  Initializes the card.
- void MMC_CardDeinit (mmc_card_t ∗card)
  Deinitializes the card.
- status_t MMC_HostInit (mmc_card_t ∗card)
  initialize the host.
- void MMC_HostDeinit (mmc_card_t ∗card)
  Deinitializes the host.
- void MMC_HostDoReset (mmc_card_t ∗card)
  Resets the host.
- void MMC_HostReset (SDMMCHOST_CONFIG ∗host)
  Resets the host.
- void MMC_SetCardPower (mmc_card_t ∗card, bool enable)

Sets card power.
- bool MMC_CheckReadOnly (mmc_card_t *card)
  Checks if the card is read-only.
- status_t MMC_ReadBlocks (mmc_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)
  Reads data blocks from the card.
- status_t MMC_WriteBlocks (mmc_card_t *card, const uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)
  Writes data blocks to the card.
- status_t MMC_EraseGroups (mmc_card_t *card, uint32_t startGroup, uint32_t endGroup)
  Erases groups of the card.
- status_t MMC_SelectPartition (mmc_card_t *card, mmc_access_partition_t partitionNumber)
  Selects the partition to access.
- status_t MMC_SetBootConfig (mmc_card_t *card, const mmc_boot_config_t *config)
  Configures the boot activity of the card.
- status_t MMC_StartBoot (mmc_card_t *card, const mmc_boot_config_t *mmcConfig, uint8_t *buffer, sdmmchost_boot_config_t *hostConfig)
  MMC card start boot.
- status_t MMC_SetBootConfigWP (mmc_card_t *card, uint8_t wp)
  MMC card set boot configuration write protect.
- status_t MMC_ReadBootData (mmc_card_t *card, uint8_t *buffer, sdmmchost_boot_config_t *hostConfig)
  MMC card continuous read boot data.
- status_t MMC_StopBoot (mmc_card_t *card, uint32_t bootMode)
  MMC card stop boot mode.
- status_t MMC_SetBootPartitionWP (mmc_card_t *card, mmc_boot_partition_wp_t bootPartitionWP)
  MMC card set boot partition write protect.
- status_t MMC_EnableCacheControl (mmc_card_t *card, bool enable)
  MMC card cache control function.
- status_t MMC_FlushCache (mmc_card_t *card)
  MMC card cache flush function.
- status_t MMC_SetSleepAwake (mmc_card_t *card, mmc_sleep_awake_t state)
  MMC sets card sleep awake state.
- status_t MMC_PollingCardStatusBusy (mmc_card_t *card, bool checkStatus, uint32_t timeoutMs)
  Polling card idle status.

50.4.3 Data Structure Documentation

50.4.3.1 struct mmc_usr_param_t

Data Fields

- mmc_io_strength_t ioStrength
  switch sd io strength
- uint32_t maxFreq
  board support maximum frequency
- uint32_t capability
  board capability flag

50.4.3.2 struct mmc_card_t

Defines the card structure including the necessary fields to identify and describe the card.

Data Fields

- sdmmchost_t * host
  Host information.
- mmc_usr_param_t usrParam
  user parameter
- bool isHostReady
  Use this flag to indicate if host re-init needed or not.
- bool noInteralAlign
  Use this flag to disable sdmmc align.
- uint32_t busClock_Hz
  MMC bus clock united in Hz.
- uint32_t relativeAddress
  Relative address of the card.
- bool enablePreDefinedBlockCount
  Enable PRE-DEFINED block count when read/write.
- uint32_t flags
  Capability flag in _mmc_card_flag.
- uint8_t internalBuffer [FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE]
  raw buffer used for mmc driver internal
- uint32_t ocr
  Raw OCR content.
- mmc_cid_t cid
  CID.
- mmc_csd_t csd
  CSD.
- mmc_extended_csd_t extendedCsd
  Extended CSD.
- uint32_t blockSize
  Card block size.
- uint32_t userPartitionBlocks
  Card total block number in user partition.
- uint32_t bootPartitionBlocks
  Boot partition size united as block size.
- uint32_t eraseGroupBlocks
  Erase group size united as block size.
- mmc_access_partition_t currentPartition
  Current access partition.
- mmc_voltage_window_t hostVoltageWindowVCCQ
  application must set this value according to board specific
- mmc_voltage_window_t hostVoltageWindowVCC
  application must set this value according to board specific
- mmc_high_speed_timing_t busTiming
  indicates the current work timing mode
- mmc_data_bus_width_t busWidth
  indicates the current work bus width

- sdmmc_osa_mutex_t lock
  card access lock

Field Documentation

(1) bool mmc_card_t::noInteralAlign

If disabled, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are aligned to low level driver.

50.4.4 Macro Definition Documentation

50.4.4.1 #define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 5U, 0U)) /∗2.5.0∗/

50.4.5 Enumeration Type Documentation

50.4.5.1 anonymous enum

Enumerator

kMMC_SupportHighSpeed26MHZFlag Support high speed 26MHZ.
kMMC_SupportHighSpeed52MHZFlag Support high speed 52MHZ.
kMMC_SupportHighSpeedDDR52MHZ180V300VFlag ddr 52MHZ 1.8V or 3.0V
kMMC_SupportHighSpeedDDR52MHZ120VFlag DDR 52MHZ 1.2V.
kMMC_SupportHS200200MHZ180VFlag HS200 ,200MHZ,1.8V.
kMMC_SupportHS200200MHZ120VFlag HS200, 200MHZ, 1.2V.
kMMC_SupportHS400DDR200MHZ180VFlag HS400, DDR, 200MHZ,1.8V.
kMMC_SupportHS400DDR200MHZ120VFlag HS400, DDR, 200MHZ,1.2V.
kMMC_SupportHighCapacityFlag Support high capacity.
kMMC_SupportAlternateBootFlag Support alternate boot.
kMMC_SupportDDRBootFlag support DDR boot flag
kMMC_SupportHighSpeedBootFlag support high speed boot flag
kMMC_SupportEnhanceHS400StrobeFlag support enhance HS400 strobe

50.4.5.2 enum mmc_sleep_awake_t

Enumerator

kMMC_Sleep MMC card sleep.
kMMC_Awake MMC card awake.

50.4.6 Function Documentation

50.4.6.1 status_t MMC_Init (mmc_card_t * *card*)

Parameters

card	Card descriptor.

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex to be created redundantly, application must follow bellow sequence for card re-initialization:

```
  MMC_Deinit(card);
  MMC_Init(card);
*
```

Return values

kStatus_SDMMC_HostNotReady	Host is not ready.
kStatus_SDMMC_GoIdleFailed	Going idle failed.
kStatus_SDMMC_HandShakeOperationConditionFailed	Sending operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Sending CID failed.
kStatus_SDMMC_SetRelativeAddressFailed	Setging relative address failed.
kStatus_SDMMC_SendCsdFailed	Sending CSD failed.
kStatus_SDMMC_CardNotSupport	Card not support.
kStatus_SDMMC_SelectCardFailed	Sending SELECT_CARD command failed.
kStatus_SDMMC_SendExtendedCsdFailed	Sending EXT_CSD failed.
kStatus_SDMMC_SetDataBusWidthFailed	Setting bus width failed.

kStatus_SDMMC_SwitchBusTimingFailed	Switching high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Setting card block size failed.
kStatus_SDMMC_SetPowerClassFail	Setting card power class failed.
kStatus_Success	Operation succeeded.

50.4.6.2 void MMC_Deinit (mmc_card_t * *card*)

Note

It is a thread safe function.

Parameters

card	Card descriptor.

50.4.6.3 status_t MMC_CardInit (mmc_card_t * *card*)

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex to be created redundantly, application must follow bellow sequence for card re-initialization:

```
  MMC_CardDeinit(card);
  MMC_CardInit(card);
*
```

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_HostNotReady	Host is not ready.

kStatus_SDMMC_GoIdleFailed	Going idle failed.
kStatus_SDMMC_HandShakeOperationConditionFailed	Sending operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Sending CID failed.
kStatus_SDMMC_SetRelativeAddressFailed	Setting relative address failed.
kStatus_SDMMC_SendCsdFailed	Sending CSD failed.
kStatus_SDMMC_CardNotSupport	Card not support.
kStatus_SDMMC_SelectCardFailed	Sending SELECT_CARD command failed.
kStatus_SDMMC_SendExtendedCsdFailed	Sending EXT_CSD failed.
kStatus_SDMMC_SetDataBusWidthFailed	Setting bus width failed.
kStatus_SDMMC_SwitchBusTimingFailed	Switching high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Setting card block size failed.
kStatus_SDMMC_SetPowerClassFail	Setting card power class failed.
kStatus_Success	Operation succeeded.

50.4.6.4 void MMC_CardDeinit (mmc_card_t * *card*)

Note

It is a thread safe function.

Parameters

card	Card descriptor.

50.4.6.5 status_t MMC_HostInit (mmc_card_t * *card*)

This function deinitializes the specific host.

Parameters

card	Card descriptor.

50.4.6.6 void MMC_HostDeinit (mmc_card_t * *card*)

This function deinitializes the host.

Parameters

card	Card descriptor.

50.4.6.7 void MMC_HostDoReset (mmc_card_t * *card*)

This function resets the specific host.

Parameters

card	Card descriptor.

50.4.6.8 void MMC_HostReset (SDMMCHOST_CONFIG * *host*)

Deprecated Do not use this function. It has been superceded by MMC_HostDoReset. This function resets the specific host.

Parameters

host	Host descriptor.

50.4.6.9 void MMC_SetCardPower (mmc_card_t * *card,* bool *enable*)

Parameters

card	Card descriptor.
enable	True is powering on, false is powering off.

50.4.6.10 bool MMC_CheckReadOnly (mmc_card_t * *card*)

Parameters

card	Card descriptor.

Return values

true	Card is read only.
false	Card isn't read only.

50.4.6.11 status_t MMC_ReadBlocks (mmc_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

Note

It is a thread safe function.

Parameters

card	Card descriptor.
buffer	The buffer to save data.
startBlock	The start block index.
blockCount	The number of blocks to read.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_Card-NotSupport	Card not support.

kStatus_SDMMC_SetBlockCountFailed	Setting block count failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_StopTransmissionFailed	Stopping transmission failed.
kStatus_Success	Operation succeeded.

50.4.6.12 status_t MMC_WriteBlocks (mmc_card_t * *card*, const uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

Note

1. It is a thread safe function.
2. It is an async write function which means that the card status may still be busy after the function returns. Application can call function MMC_PollingCardStatusBusy to wait for the card status to be idle after the write operation.

Parameters

card	Card descriptor.
buffer	The buffer to save data blocks.
startBlock	Start block number to write.
blockCount	Block count.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_NotSupportYet	Not support now.
kStatus_SDMMC_SetBlockCountFailed	Setting block count failed.
kStatus_SDMMC_WaitWriteCompleteFailed	Sending status failed.

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Stop-TransmissionFailed	Stop transmission failed.
kStatus_Success	Operation succeeded.

50.4.6.13 status_t MMC_EraseGroups (mmc_card_t * *card*, uint32_t *startGroup*, uint32_t *endGroup*)

The erase command is best used to erase the entire device or a partition. Erase group is the smallest erase unit in MMC card. The erase range is [startGroup, endGroup].

Note

1. It is a thread safe function.
2. This function always polls card busy status according to the timeout value defined in the card register after all the erase command sent out.

Parameters

card	Card descriptor.
startGroup	Start group number.
endGroup	End group number.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_Wait-WriteCompleteFailed	Send status failed.
kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operation succeeded.

50.4.6.14 status_t MMC_SelectPartition (mmc_card_t * *card*, mmc_access_partition_t *partitionNumber*)

Note

It is a thread safe function.

Parameters

card	Card descriptor.
partition-Number	The partition number.

Return values

kStatus_SDMMC_-ConfigureExtendedCsd-Failed	Configuring EXT_CSD failed.
kStatus_Success	Operation succeeded.

50.4.6.15 status_t MMC_SetBootConfig (mmc_card_t * *card*, const mmc_boot_config_t * *config*)

Parameters

card	Card descriptor.
config	Boot configuration structure.

Return values

kStatus_SDMMC_Not-SupportYet	Not support now.
kStatus_SDMMC_-ConfigureExtendedCsd-Failed	Configuring EXT_CSD failed.
kStatus_SDMMC_-ConfigureBootFailed	Configuring boot failed.
kStatus_Success	Operation succeeded.

50.4.6.16 status_t MMC_StartBoot (mmc_card_t * *card*, const mmc_boot_config_t * *mmcConfig*, uint8_t * *buffer*, sdmmchost_boot_config_t * *hostConfig*)

Parameters

card	Card descriptor.
mmcConfig	The mmc Boot configuration structure.
buffer	Address to receive data.
hostConfig	Host boot configurations.

Return values

kStatus_Fail	Failed.
kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Go-IdleFailed	Resetting card failed.
kStatus_Success	Operation succeeded.

50.4.6.17 status_t MMC_SetBootConfigWP (mmc_card_t * *card*, uint8_t *wp*)

Parameters

card	Card descriptor.
wp	Write protect value.

50.4.6.18 status_t MMC_ReadBootData (mmc_card_t * *card*, uint8_t * *buffer*, sdmmchost_boot_config_t * *hostConfig*)

Parameters

card	Card descriptor.
buffer	Buffer address.
hostConfig	Host boot configurations.

50.4.6.19 status_t MMC_StopBoot (mmc_card_t * *card*, uint32_t *bootMode*)

Parameters

card	Card descriptor.
bootMode	Boot mode.

50.4.6.20 status_t MMC_SetBootPartitionWP (mmc_card_t * *card,* mmc_boot_partition_wp_t *bootPartitionWP*)

Parameters

card	Card descriptor.
bootPartition-WP	Boot partition write protect value.

50.4.6.21 status_t MMC_EnableCacheControl (mmc_card_t * *card,* bool *enable*)

The mmc device's cache is enabled by the driver by default. The cache should in typical case reduce the access time (compared to an access to the main nonvolatile storage) for both write and read.

Parameters

card	Card descriptor.
enable	True is enabling the cache, false is disabling the cache.

50.4.6.22 status_t MMC_FlushCache (mmc_card_t * *card*)

A Flush operation refers to the requirement, from the host to the device, to write the cached data to the nonvolatile memory. Prior to a flush, the device may autonomously write data to the nonvolatile memory, but after the flush operation all data in the volatile area must be written to nonvolatile memory. There is no requirement for flush due to switching between the partitions. (Note: This also implies that the cache data shall not be lost when switching between partitions). Cached data may be lost in SLEEP state, so host should flush the cache before placing the device into SLEEP state.

Parameters

card	Card descriptor.

50.4.6.23 status_t MMC_SetSleepAwake (mmc_card_t * *card*, mmc_sleep_awake_t *state*)

The Sleep/Awake command is used to initiate the state transition between Standby state and Sleep state. The memory device indicates the transition phase busy by pulling down the DAT0 line. The Sleep/Standby state is reached when the memory device stops pulling down the DAT0 line, then the function returns.

Parameters

card	Card descriptor.
state	The sleep/awake command argument, refer to mmc_sleep_awake_t.

Return values

kStatus_SDMMC_NotSupportYet	Indicates the memory device doesn't support the Sleep/Awake command.
kStatus_SDMMC_TransferFailed	Indicates command transferred fail.
kStatus_SDMMC_PollingCardIdleFailed	Indicates polling DAT0 busy timeout.
kStatus_SDMMC_DeselectCardFailed	Indicates deselect card command failed.
kStatus_SDMMC_SelectCardFailed	Indicates select card command failed.
kStatus_Success	Indicates the card state switched successfully.

50.4.6.24 status_t MMC_PollingCardStatusBusy (mmc_card_t * *card*, bool *checkStatus*, uint32_t *timeoutMs*)

This function can be used to poll the status from busy to idle, the function will return with the card status being idle or timeout or command failed.

Parameters

card	Card descriptor.
checkStatus	True is send CMD and read DAT0 status to check card status, false is read DAT0 status only.
timeoutMs	Polling card status timeout value.

Return values

kStatus_SDMMC_CardStatusIdle	Card is idle.
kStatus_SDMMC_CardStatusBusy	Card is busy.
kStatus_SDMMC_TransferFailed	Command tranfer failed.
kStatus_SDMMC_SwitchFailed	Status command reports switch error.

50.5 SDMMC HOST Driver

50.5.1 Overview

The host adapter driver provide adapter for blocking/non_blocking mode.

Modules

- USDHC HOST adapter Driver

50.6 SDMMC OSA

50.6.1 Overview

The sdmmc osa adapter provide interface of os adapter.

Data Structures

- struct sdmmc_osa_event_t
  sdmmc osa event More...
- struct sdmmc_osa_mutex_t
  sdmmc osa mutex More...

Macros

- #define SDMMC_OSA_EVENT_TRANSFER_CMD_SUCCESS (1UL << 0U)
  transfer event
- #define SDMMC_OSA_EVENT_CARD_INSERTED (1UL << 8U)
  card detect event, start from index 8
- #define SDMMC_OSA_POLLING_EVENT_BY_SEMPHORE 1
  enable semphore by default

sdmmc osa Function

- void SDMMC_OSAInit (void)
  Initialize OSA.
- status_t SDMMC_OSAEventCreate (void *eventHandle)
  OSA Create event.
- status_t SDMMC_OSAEventWait (void *eventHandle, uint32_t eventType, uint32_t timeoutMilliseconds, uint32_t *event)
  Wait event.
- status_t SDMMC_OSAEventSet (void *eventHandle, uint32_t eventType)
  set event.
- status_t SDMMC_OSAEventGet (void *eventHandle, uint32_t eventType, uint32_t *flag)
  Get event flag.
- status_t SDMMC_OSAEventClear (void *eventHandle, uint32_t eventType)
  clear event flag.
- status_t SDMMC_OSAEventDestroy (void *eventHandle)
  Delete event.
- status_t SDMMC_OSAMutexCreate (void *mutexHandle)
  Create a mutex.
- status_t SDMMC_OSAMutexLock (void *mutexHandle, uint32_t millisec)
  set event.
- status_t SDMMC_OSAMutexUnlock (void *mutexHandle)
  Get event flag.
- status_t SDMMC_OSAMutexDestroy (void *mutexHandle)
  Delete mutex.

- void SDMMC_OSADelay (uint32_t milliseconds)
  sdmmc delay.
- uint32_t SDMMC_OSADelayUs (uint32_t microseconds)
  sdmmc delay us.

50.6.2 Data Structure Documentation

50.6.2.1 struct sdmmc_osa_event_t

50.6.2.2 struct sdmmc_osa_mutex_t

50.6.3 Function Documentation

50.6.3.1 status_t SDMMC_OSAEventCreate (void * *eventHandle*)

Parameters

eventHandle	event handle.

Return values

kStatus_Fail	or kStatus_Success.

50.6.3.2 status_t SDMMC_OSAEventWait (void * *eventHandle,* uint32_t *eventType,* uint32_t *timeoutMilliseconds,* uint32_t * *event*)

Parameters

eventHandle	The event type
eventType	Timeout time in milliseconds.
timeout-Milliseconds	timeout value in ms.
event	event flags.

Return values

kStatus_Fail	or kStatus_Success.

50.6.3.3 status_t SDMMC_OSAEventSet (void ∗ *eventHandle,* uint32_t *eventType*)

Parameters

eventHandle	event handle.
eventType	The event type

Return values

kStatus_Fail	or kStatus_Success.

50.6.3.4 status_t SDMMC_OSAEventGet (void ∗ *eventHandle,* uint32_t *eventType,* uint32_t ∗ *flag*)

Parameters

eventHandle	event handle.
eventType	event type.
flag	pointer to store event value.

Return values

kStatus_Fail	or kStatus_Success.

50.6.3.5 status_t SDMMC_OSAEventClear (void ∗ *eventHandle,* uint32_t *eventType*)

Parameters

eventHandle	event handle.
eventType	The event type

Return values

kStatus_Fail	or kStatus_Success.

50.6.3.6 status_t SDMMC_OSAEventDestroy (void ∗ *eventHandle*)

Parameters

eventHandle	The event handle.

50.6.3.7 status_t SDMMC_OSAMutexCreate (void ∗ *mutexHandle*)

Parameters

mutexHandle	mutex handle.

Return values

kStatus_Fail	or kStatus_Success.

50.6.3.8 status_t SDMMC_OSAMutexLock (void ∗ *mutexHandle,* uint32_t *millisec*)

Parameters

mutexHandle	mutex handle.
millisec	The maximum number of milliseconds to wait for the mutex. If the mutex is locked, Pass the value osaWaitForever_c will wait indefinitely, pass 0 will return KOSA_-StatusTimeout immediately.

Return values

kStatus_Fail	or kStatus_Success.

50.6.3.9 status_t SDMMC_OSAMutexUnlock (void ∗ *mutexHandle*)

Parameters

mutexHandle	mutex handle.

Return values

kStatus_Fail	or kStatus_Success.

50.6.3.10 status_t SDMMC_OSAMutexDestroy (void ∗ *mutexHandle*)

Parameters

mutexHandle	The mutex handle.

50.6.3.11 void SDMMC_OSADelay (uint32_t *milliseconds*)

Parameters

milliseconds	time to delay

50.6.3.12 uint32_t SDMMC_OSADelayUs (uint32_t *microseconds*)

Parameters

microseconds	time to delay

Returns

actual delayed microseconds

50.6.4 USDHC HOST adapter Driver

50.6.4.1 Overview

The USDHC host adapter driver provide adapter for blocking/non_blocking mode.

Cache maintain capability

To maintain data integrity during DMA operations on the platform that has cache, host driver provide a cache maintain functionality by set: host.enableCacheControl = kSDMMCHOST_CacheControlRW-Buffer This is used for only when the low level driver cache maintain functionality is disabled. It is suggest that the address of buffer used for read/write is align with cache line size.

Cache line alignment maintain capability

when application submit a transfer request that the data buffer address is not align with cache line size, the potential data loss may happen during driver maintain the data cache, to aovid such issue happens, sdmmc usdhc host driver provides support on convert the unalign data transfer into align data transfer, if application would like to use the feature, please enable this functionality by define below macro firstly, #define SDMMCHOST_ENABLE_CACHE_LINE_ALIGN_TRANSFER 1 And then call SDMMCHOST_InstallCacheAlignBuffer to install a cache line size align buffer, please note the installed buffer size must not small than 2 ∗ cache line size. Please note that this functionality is support by the non blocking adapter only.

Data Structures

- struct sdmmchost_t
  sdmmc host handler More...

Macros

- #define FSL_SDMMC_HOST_ADAPTER_VERSION (MAKE_VERSION(2U, 6U, 1U)) /∗2.6.-1∗/
  Middleware adapter version.
- #define SDMMCHOST_SUPPORT_HIGH_SPEED (1U)
  sdmmc host misc capability
- #define SDMMCHOST_SUPPORT_DDR50 (SDMMCHOST_SUPPORT_DDR_MODE)
  sdmmc host sdcard DDR50 mode capability
- #define SDMMCHOST_SUPPORT_SDR104 (1U)
  sdmmc host sdcard SDR50 mode capability
- #define SDMMCHOST_SUPPORT_SDR50 (1U)
  sdmmc host sdcard SDR104/mmccard HS200 mode capability
- #define SDMMCHOST_SUPPORT_HS400 (1U)
  sdmmc host mmccard HS400 mode capability

- #define SDMMCHOST_INSTANCE_SUPPORT_8_BIT_WIDTH(host) FSL_FEATURE_USDHC_INSTANCE_SUPPORT_8_BIT_WIDTHn(host->hostController.base)
  sdmmc host instance capability
- #define SDMMCHOST_DATA3_DETECT_CARD_DELAY (10U)
  sdmmchost delay for DAT3 detect card
- #define SDMMCHOST_DMA_DESCRIPTOR_BUFFER_ALIGN_SIZE (4U)
  SDMMC host dma descriptor buffer address align size.
- #define SDMMCHOST_STANDARD_TUNING_START (10U)
  tuning configuration
- #define SDMMCHOST_TUINIG_STEP (2U)
  standard tuning stBep

Typedefs

- typedef usdhc_transfer_t sdmmchost_transfer_t
  sdmmc host transfer function

Enumerations

- enum {
  kSDMMCHOST_SupportHighSpeed = 1U << 0U,
  kSDMMCHOST_SupportSuspendResume = 1U << 1U,
  kSDMMCHOST_SupportVoltage3v3 = 1U << 2U,
  kSDMMCHOST_SupportVoltage3v0 = 1U << 3U,
  kSDMMCHOST_SupportVoltage1v8 = 1U << 4U,
  kSDMMCHOST_SupportVoltage1v2 = 1U << 5U,
  kSDMMCHOST_Support4BitDataWidth = 1U << 6U,
  kSDMMCHOST_Support8BitDataWidth = 1U << 7U,
  kSDMMCHOST_SupportDDRMode = 1U << 8U,
  kSDMMCHOST_SupportDetectCardByData3 = 1U << 9U,
  kSDMMCHOST_SupportDetectCardByCD = 1U << 10U,
  kSDMMCHOST_SupportAutoCmd12 = 1U << 11U,
  kSDMMCHOST_SupportSDR104 = 1U << 12U,
  kSDMMCHOST_SupportSDR50 = 1U << 13U,
  kSDMMCHOST_SupportHS200 = 1U << 14U,
  kSDMMCHOST_SupportHS400 = 1U << 15U }
  sdmmc host capability
- enum {
  kSDMMCHOST_EndianModeBig = 0U,
  kSDMMCHOST_EndianModeHalfWordBig = 1U,
  kSDMMCHOST_EndianModeLittle = 2U }
  host Endian mode corresponding to driver define
- enum {
  kSDMMCHOST_StandardTuning = 0U,
  kSDMMCHOST_ManualTuning = 1U }
  sdmmc host tuning type

- enum {
  kSDMMCHOST_NoCacheControl = 0U,
  kSDMMCHOST_CacheControlRWBuffer = 1U }
  sdmmc host maintain cache flag

USDHC host controller function

- void SDMMCHOST_SetCardBusWidth (sdmmchost_t *host, uint32_t dataBusWidth)
  set data bus width.
- static void SDMMCHOST_SendCardActive (sdmmchost_t *host)
  Send initilization active 80 clocks to card.
- static uint32_t SDMMCHOST_SetCardClock (sdmmchost_t *host, uint32_t targetClock)
  Set card bus clock.
- static bool SDMMCHOST_IsCardBusy (sdmmchost_t *host)
  check card status by DATA0.
- static uint32_t SDMMCHOST_GetSignalLineStatus (sdmmchost_t *host, uint32_t signalLine)
  Get signal line status.
- static void SDMMCHOST_EnableCardInt (sdmmchost_t *host, bool enable)
  enable card interrupt.
- static void SDMMCHOST_EnableDDRMode (sdmmchost_t *host, bool enable, uint32_t nibblePos)
  enable DDR mode.
- static void SDMMCHOST_EnableHS400Mode (sdmmchost_t *host, bool enable)
  enable HS400 mode.
- static void SDMMCHOST_EnableStrobeDll (sdmmchost_t *host, bool enable)
  enable STROBE DLL.
- status_t SDMMCHOST_StartBoot (sdmmchost_t *host, sdmmchost_boot_config_t *hostConfig, sdmmchost_cmd_t *cmd, uint8_t *buffer)
  start read boot data.
- status_t SDMMCHOST_ReadBootData (sdmmchost_t *host, sdmmchost_boot_config_t *hostConfig, uint8_t *buffer)
  read boot data.
- static void SDMMCHOST_EnableBoot (sdmmchost_t *host, bool enable)
  enable boot mode.
- status_t SDMMCHOST_CardIntInit (sdmmchost_t *host, void *sdioInt)
  card interrupt function.
- static void SDMMCHOST_ForceClockOn (sdmmchost_t *host, bool enable)
  force card clock on.
- void SDMMCHOST_SwitchToVoltage (sdmmchost_t *host, uint32_t voltage)
  switch to voltage.
- status_t SDMMCHOST_CardDetectInit (sdmmchost_t *host, void *cd)
  card detect init function.
- status_t SDMMCHOST_PollingCardDetectStatus (sdmmchost_t *host, uint32_t waitCardStatus, uint32_t timeout)
  Detect card insert, only need for SD cases.
- uint32_t SDMMCHOST_CardDetectStatus (sdmmchost_t *host)
  card detect status.
- status_t SDMMCHOST_Init (sdmmchost_t *host)
  Init host controller.
- void SDMMCHOST_Deinit (sdmmchost_t *host)

Deinit host controller.

- void SDMMCHOST_SetCardPower (sdmmchost_t *host, bool enable)
  host power off card function.
- status_t SDMMCHOST_TransferFunction (sdmmchost_t *host, sdmmchost_transfer_t *content)
  host transfer function.
- status_t SDMMCHOST_ExecuteTuning (sdmmchost_t *host, uint32_t tuningCmd, uint32_t *revBuf, uint32_t blockSize)
  sdmmc host excute tuning.
- void SDMMCHOST_Reset (sdmmchost_t *host)
  host reset function.
- void SDMMCHOST_ConvertDataToLittleEndian (sdmmchost_t *host, uint32_t *data, uint32_t wordSize, uint32_t format)
  sdmmc host convert data sequence to little endian sequence

50.6.4.2 Data Structure Documentation

50.6.4.2.1 struct sdmmchost_t

Data Fields

- usdhc_host_t hostController
  host configuration
- void * dmaDesBuffer
  DMA descriptor buffer address.
- uint32_t dmaDesBufferWordsNum
  DMA descriptor buffer size in byte.
- usdhc_handle_t handle
  host controller handler
- uint32_t capability
  host controller capability
- uint32_t maxBlockCount
  host controller maximum block count
- uint32_t maxBlockSize
  host controller maximum block size
- uint8_t tuningType
  host tuning type
- sdmmc_osa_event_t hostEvent
  host event handler
- void * cd
  card detect
- void * cardInt
  call back function for card interrupt
- sdmmc_osa_mutex_t lock
  host access lock

50.6.4.3 Macro Definition Documentation

50.6.4.3.1 #define FSL_SDMMC_HOST_ADAPTER_VERSION (MAKE_VERSION(2U, 6U, 1U)) /∗2.6.1∗/

50.6.4.3.2 #define SDMMCHOST_STANDARD_TUNING_START (10U)

standard tuning start point

50.6.4.4 Enumeration Type Documentation

50.6.4.4.1 anonymous enum

Enumerator

kSDMMCHOST_SupportHighSpeed high speed capability
kSDMMCHOST_SupportSuspendResume suspend resume capability
kSDMMCHOST_SupportVoltage3v3 3V3 capability
kSDMMCHOST_SupportVoltage3v0 3V0 capability
kSDMMCHOST_SupportVoltage1v8 1V8 capability
kSDMMCHOST_SupportVoltage1v2 1V2 capability
kSDMMCHOST_Support4BitDataWidth 4 bit data width capability
kSDMMCHOST_Support8BitDataWidth 8 bit data width capability
kSDMMCHOST_SupportDDRMode DDR mode capability.
kSDMMCHOST_SupportDetectCardByData3 data3 detect card capability
kSDMMCHOST_SupportDetectCardByCD CD detect card capability.
kSDMMCHOST_SupportAutoCmd12 auto command 12 capability
kSDMMCHOST_SupportSDR104 SDR104 capability.
kSDMMCHOST_SupportSDR50 SDR50 capability.
kSDMMCHOST_SupportHS200 HS200 capability.
kSDMMCHOST_SupportHS400 HS400 capability.

50.6.4.4.2 anonymous enum

Enumerator

kSDMMCHOST_EndianModeBig Big endian mode.
kSDMMCHOST_EndianModeHalfWordBig Half word big endian mode.
kSDMMCHOST_EndianModeLittle Little endian mode.

50.6.4.4.3 anonymous enum

Enumerator

kSDMMCHOST_StandardTuning standard tuning type
kSDMMCHOST_ManualTuning manual tuning type

50.6.4.4.4 anonymous enum

Enumerator

kSDMMCHOST_NoCacheControl sdmmc host cache control disabled
kSDMMCHOST_CacheControlRWBuffer sdmmc host cache control read/write buffer

50.6.4.5 Function Documentation

50.6.4.5.1 void SDMMCHOST_SetCardBusWidth (sdmmchost_t * *host*, uint32_t *dataBusWidth*)

Parameters

host	host handler
dataBusWidth	data bus width

50.6.4.5.2 static void SDMMCHOST_SendCardActive (sdmmchost_t * *host*) `[inline]`, `[static]`

Parameters

host	host handler

50.6.4.5.3 static uint32_t SDMMCHOST_SetCardClock (sdmmchost_t * *host*, uint32_t *targetClock*) `[inline]`, `[static]`

Parameters

host	host handler
targetClock	target clock frequency

Return values

actual	clock frequency can be reach.

50.6.4.5.4 static bool SDMMCHOST_IsCardBusy (sdmmchost_t * *host*) `[inline]`, `[static]`

Parameters

host	host handler

Return values

true	is busy, false is idle.

50.6.4.5.5 static uint32_t SDMMCHOST_GetSignalLineStatus (sdmmchost_t * *host,* uint32_t *signalLine*) [inline], [static]

Parameters

host	host handler
signalLine	signal line type, reference _sdmmc_signal_line

50.6.4.5.6 static void SDMMCHOST_EnableCardInt (sdmmchost_t * *host,* bool *enable*) [inline], [static]

Parameters

host	host handler
enable	true is enable, false is disable.

50.6.4.5.7 static void SDMMCHOST_EnableDDRMode (sdmmchost_t * *host,* bool *enable,* uint32_t *nibblePos*) [inline], [static]

Parameters

host	host handler
enable	true is enable, false is disable.
nibblePos	nibble position indictation. 0- the sequence is 'odd high nibble -> even high nibble -> odd low nibble -> even low nibble'; 1- the sequence is 'odd high nibble -> odd low nibble -> even high nibble -> even low nibble'.

50.6.4.5.8 static void SDMMCHOST_EnableHS400Mode (sdmmchost_t * *host,* bool *enable*) [inline], [static]

Parameters

host	host handler
enable	true is enable, false is disable.

50.6.4.5.9 static void SDMMCHOST_EnableStrobeDll (sdmmchost_t * *host*, bool *enable*) [inline], [static]

Parameters

host	host handler
enable	true is enable, false is disable.

50.6.4.5.10 status_t SDMMCHOST_StartBoot (sdmmchost_t * *host*, sdmmchost_boot_config_t * *hostConfig*, sdmmchost_cmd_t * *cmd*, uint8_t * *buffer*)

Parameters

host	host handler
hostConfig	boot configuration
cmd	boot command
buffer	buffer address

50.6.4.5.11 status_t SDMMCHOST_ReadBootData (sdmmchost_t * *host*, sdmmchost_boot_config_t * *hostConfig*, uint8_t * *buffer*)

Parameters

host	host handler
hostConfig	boot configuration
buffer	buffer address

50.6.4.5.12 static void SDMMCHOST_EnableBoot (sdmmchost_t * *host*, bool *enable*) [inline], [static]

Parameters

host	host handler
enable	true is enable, false is disable

50.6.4.5.13 status_t SDMMCHOST_CardIntInit (sdmmchost_t * *host,* void * *sdioInt*)

Parameters

host	host handler
sdioInt	card interrupt configuration

50.6.4.5.14 static void SDMMCHOST_ForceClockOn (sdmmchost_t * *host,* bool *enable*) [inline], [static]

Parameters

host	host handler
enable	true is enable, false is disable.

50.6.4.5.15 void SDMMCHOST_SwitchToVoltage (sdmmchost_t * *host,* uint32_t *voltage*)

Parameters

host	host handler
voltage	switch to voltage level.

50.6.4.5.16 status_t SDMMCHOST_CardDetectInit (sdmmchost_t * *host,* void * *cd*)

Parameters

host	host handler

cd	card detect configuration

50.6.4.5.17 status_t SDMMCHOST_PollingCardDetectStatus (sdmmchost_t * *host*, uint32_t *waitCardStatus*, uint32_t *timeout*)

Parameters

host	host handler
waitCardStatus	status which user want to wait
timeout	wait time out.

Return values

kStatus_Success	detect card insert
kStatus_Fail	card insert event fail

50.6.4.5.18 uint32_t SDMMCHOST_CardDetectStatus (sdmmchost_t * *host*)

Parameters

host	host handler

Return values

kSD_Inserted,kSD_-Removed	

50.6.4.5.19 status_t SDMMCHOST_Init (sdmmchost_t * *host*)

Thread safe function, please note that the function will create the mutex lock dynamically by default, so to avoid the mutex create redundantly, application must follow bellow sequence for card re-initialization

```
* SDMMCHOST_Deinit(host);
* SDMMCHOST_Init(host);
*
```

Parameters

host	host handler

Return values

kStatus_Success	host init success
kStatus_Fail	event fail

50.6.4.5.20 void SDMMCHOST_Deinit (sdmmchost_t * *host*)

Please note it is a thread safe function.

Parameters

host	host handler

50.6.4.5.21 void SDMMCHOST_SetCardPower (sdmmchost_t * *host,* bool *enable*)

Parameters

host	host handler
enable	true is power on, false is power down.

50.6.4.5.22 status_t SDMMCHOST_TransferFunction (sdmmchost_t * *host,* sdmmchost_transfer_t * *content*)

Please note it is a thread safe function.

Note

the host transfer function support below functionality,

1. Non-cache line size alignment check on the data buffer, it is means that no matter the data buffer used for data transfer is align with cache line size or not, sdmmc host driver will use the address directly.
2. Cache line size alignment check on the data buffer, sdmmc host driver will check the data buffer address, if the buffer is not align with cache line size, sdmmc host driver will convert it to cache line size align buffer, the functionality is enabled by #define SDMMCHOST_ENABLE_CACHE_LINE_ALIGN_TRANSFER 1 #define FSL_USDHC_ENABLE_SCATTER_GATHER_TRANSFER 1U If application would like to enable the cache line size align functionality, please make sure the SDMMCHOST_InstallCacheAlignBuffer is called before

submit data transfer request and make sure the installing buffer size is not smaller than 2 ∗ cache line size.

Parameters

host	host handler
content	transfer content.

50.6.4.5.23 status_t SDMMCHOST_ExecuteTuning (sdmmchost_t ∗ *host*, uint32_t *tuningCmd*, uint32_t ∗ *revBuf*, uint32_t *blockSize*)

Parameters

host	host handler
tuningCmd	tuning command.
revBuf	receive buffer pointer
blockSize	tuning data block size.

50.6.4.5.24 void SDMMCHOST_Reset (sdmmchost_t ∗ *host*)

Parameters

host	host handler

50.6.4.5.25 void SDMMCHOST_ConvertDataToLittleEndian (sdmmchost_t ∗ *host*, uint32_t ∗ *data*, uint32_t *wordSize*, uint32_t *format*)

Parameters

host	host handler.
data	data buffer address.
wordSize	data buffer size in word.
format	data packet format.

50.7 SDMMC Common

50.7.1 Overview

The sdmmc common function and definition.

Data Structures

- struct sd_detect_card_t
  sd card detect More...
- struct sd_io_voltage_t
  io voltage control configuration More...
- struct sd_usr_param_t
  sdcard user parameter More...
- struct sdio_card_int_t
  card interrupt application callback More...
- struct sdio_usr_param_t
  sdio user parameter More...
- struct sdio_fbr_t
  sdio card FBR register More...
- struct sdio_common_cis_t
  sdio card common CIS More...
- struct sdio_func_cis_t
  sdio card function CIS More...
- struct sd_status_t
  SD card status. More...
- struct sd_cid_t
  SD card CID register. More...
- struct sd_csd_t
  SD card CSD register. More...
- struct sd_scr_t
  SD card SCR register. More...
- struct mmc_cid_t
  MMC card CID register. More...
- struct mmc_csd_t
  MMC card CSD register. More...
- struct mmc_extended_csd_t
  MMC card Extended CSD register (unit: byte). More...
- struct mmc_extended_csd_config_t
  MMC Extended CSD configuration. More...
- struct mmc_boot_config_t
  MMC card boot configuration definition. More...

Macros

- #define SWAP_WORD_BYTE_SEQUENCE(x) (__REV(x))
  Reverse byte sequence in uint32_t.
- #define SWAP_HALF_WROD_BYTE_SEQUENCE(x) (__REV16(x))

Reverse byte sequence for each half word in uint32_t.

- #define FSL_SDMMC_MAX_VOLTAGE_RETRIES (1000U)

  Maximum loop count to check the card operation voltage range.
- #define FSL_SDMMC_MAX_CMD_RETRIES (10U)

  Maximum loop count to send the cmd.
- #define FSL_SDMMC_DEFAULT_BLOCK_SIZE (512U)

  Default block size.
- #define SDMMC_DATA_BUFFER_ALIGN_CACHE sizeof(uint32_t)

  make sure the internal buffer address is cache align
- #define FSL_SDMMC_CARD_INTERNAL_BUFFER_SIZE (FSL_SDMMC_DEFAULT_BLOCK_SIZE + SDMMC_DATA_BUFFER_ALIGN_CACHE)

  sdmmc card internal buffer size
- #define FSL_SDMMC_CARD_MAX_BUS_FREQ(max, target) ((max) == 0U ? (target) : ((max) > (target) ? (target) : (max)))

  get maximum freq
- #define SDMMC_LOG(format,...)

  SD/MMC error log.
- #define SDMMC_CLOCK_400KHZ (400000U)

  SD/MMC card initialization clock frequency.
- #define SD_CLOCK_25MHZ (25000000U)

  SD card bus frequency 1 in high-speed mode.
- #define SD_CLOCK_50MHZ (50000000U)

  SD card bus frequency 2 in high-speed mode.
- #define SD_CLOCK_100MHZ (100000000U)

  SD card bus frequency in SDR50 mode.
- #define SD_CLOCK_208MHZ (208000000U)

  SD card bus frequency in SDR104 mode.
- #define MMC_CLOCK_26MHZ (26000000U)

  MMC card bus frequency 1 in high-speed mode.
- #define MMC_CLOCK_52MHZ (52000000U)

  MMC card bus frequency 2 in high-speed mode.
- #define MMC_CLOCK_DDR52 (52000000U)

  MMC card bus frequency in high-speed DDR52 mode.
- #define MMC_CLOCK_HS200 (200000000U)

  MMC card bus frequency in high-speed HS200 mode.
- #define MMC_CLOCK_HS400 (400000000U)

  MMC card bus frequency in high-speed HS400 mode.
- #define SDMMC_MASK(bit) (1UL << (bit))

  mask convert
- #define SDMMC_R1_ALL_ERROR_FLAG

  R1 all the error flag.
- #define SDMMC_R1_CURRENT_STATE(x) (((x)&0x00001E00U) >> 9U)

  R1: current state.
- #define SDSPI_R7_VERSION_SHIFT (28U)

  The bit mask for COMMAND VERSION field in R7.
- #define SDSPI_R7_VERSION_MASK (0xFU)

  The bit mask for COMMAND VERSION field in R7.
- #define SDSPI_R7_VOLTAGE_SHIFT (8U)

  The bit shift for VOLTAGE ACCEPTED field in R7.
- #define SDSPI_R7_VOLTAGE_MASK (0xFU)

  The bit mask for VOLTAGE ACCEPTED field in R7.

- #define SDSPI_R7_VOLTAGE_27_36_MASK (0x1U << SDSPI_R7_VOLTAGE_SHIFT)
  The bit mask for VOLTAGE 2.7V to 3.6V field in R7.
- #define SDSPI_R7_ECHO_SHIFT (0U)
  The bit shift for ECHO field in R7.
- #define SDSPI_R7_ECHO_MASK (0xFFU)
  The bit mask for ECHO field in R7.
- #define SDSPI_DATA_ERROR_TOKEN_MASK (0xFU)
  Data error token mask.
- #define SDSPI_DATA_RESPONSE_TOKEN_MASK (0x1FU)
  Mask for data response bits.
- #define SDIO_CCCR_REG_NUMBER (0x16U)
  sdio card cccr register number
- #define SDIO_IO_READY_TIMEOUT_UNIT (10U)
  sdio IO ready timeout steps
- #define SDIO_CMD_ARGUMENT_RW_POS (31U)
  read/write flag position
- #define SDIO_CMD_ARGUMENT_FUNC_NUM_POS (28U)
  function number position
- #define SDIO_DIRECT_CMD_ARGUMENT_RAW_POS (27U)
  direct raw flag position
- #define SDIO_CMD_ARGUMENT_REG_ADDR_POS (9U)
  direct reg addr position
- #define SDIO_CMD_ARGUMENT_REG_ADDR_MASK (0x1FFFFU)
  direct reg addr mask
- #define SDIO_DIRECT_CMD_DATA_MASK (0xFFU)
  data mask
- #define SDIO_EXTEND_CMD_ARGUMENT_BLOCK_MODE_POS (27U)
  extended command argument block mode bit position
- #define SDIO_EXTEND_CMD_ARGUMENT_OP_CODE_POS (26U)
  extended command argument OP Code bit position
- #define SDIO_EXTEND_CMD_BLOCK_MODE_MASK (0x08000000U)
  block mode mask
- #define SDIO_EXTEND_CMD_OP_CODE_MASK (0x04000000U)
  op code mask
- #define SDIO_EXTEND_CMD_COUNT_MASK (0x1FFU)
  byte/block count mask
- #define SDIO_MAX_BLOCK_SIZE (2048U)
  max block size
- #define SDIO_FBR_BASE(x) ((x)*0x100U)
  function basic register
- #define SDIO_TPL_CODE_END (0xFFU)
  tuple end
- #define SDIO_TPL_CODE_MANIFID (0x20U)
  manufacturer ID
- #define SDIO_TPL_CODE_FUNCID (0x21U)
  function ID
- #define SDIO_TPL_CODE_FUNCE (0x22U)
  function extension tuple
- #define SDIO_OCR_VOLTAGE_WINDOW_MASK (0xFFFFU << 8U)
  sdio ocr voltage window mask
- #define SDIO_OCR_IO_NUM_MASK (7U << kSDIO_OcrIONumber)

sdio ocr reigster IO NUMBER mask

- #define SDIO_CCCR_SUPPORT_HIGHSPEED (1UL << 9U)

  UHS timing mode flag.
- #define SDIO_CCCR_DRIVER_TYPE_MASK (3U << 4U)

  Driver type flag.
- #define SDIO_CCCR_ASYNC_INT_MASK (1U)

  aync interrupt flag
- #define SDIO_CCCR_SUPPORT_8BIT_BUS (1UL << 18U)

  8 bit data bus flag
- #define MMC_OCR_V170TO195_SHIFT (7U)

  The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- #define MMC_OCR_V170TO195_MASK (0x00000080U)

  The bit mask for VOLTAGE WINDOW 1.70V to 1.95V field in OCR.
- #define MMC_OCR_V200TO260_SHIFT (8U)

  The bit shift for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- #define MMC_OCR_V200TO260_MASK (0x00007F00U)

  The bit mask for VOLTAGE WINDOW 2.00V to 2.60V field in OCR.
- #define MMC_OCR_V270TO360_SHIFT (15U)

  The bit shift for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- #define MMC_OCR_V270TO360_MASK (0x00FF8000U)

  The bit mask for VOLTAGE WINDOW 2.70V to 3.60V field in OCR.
- #define MMC_OCR_ACCESS_MODE_SHIFT (29U)

  The bit shift for ACCESS MODE field in OCR.
- #define MMC_OCR_ACCESS_MODE_MASK (0x60000000U)

  The bit mask for ACCESS MODE field in OCR.
- #define MMC_OCR_BUSY_SHIFT (31U)

  The bit shift for BUSY field in OCR.
- #define MMC_OCR_BUSY_MASK (1U << MMC_OCR_BUSY_SHIFT)

  The bit mask for BUSY field in OCR.
- #define MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT (0U)

  The bit shift for FREQUENCY UNIT field in TRANSFER SPEED(TRAN-SPEED in Extended CSD)
- #define MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK (0x07U)

  The bit mask for FRQEUENCY UNIT in TRANSFER SPEED.
- #define MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT (3U)

  The bit shift for MULTIPLIER field in TRANSFER SPEED.
- #define MMC_TRANSFER_SPEED_MULTIPLIER_MASK (0x78U)

  The bit mask for MULTIPLIER field in TRANSFER SPEED.
- #define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(CSD) ((((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)

  Read the value of FREQUENCY UNIT in TRANSFER SPEED.
- #define READ_MMC_TRANSFER_SPEED_MULTIPLIER(CSD) ((((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)

  Read the value of MULTIPLER filed in TRANSFER SPEED.
- #define MMC_POWER_CLASS_4BIT_MASK (0x0FU)

  The power class value bit mask when bus in 4 bit mode.
- #define MMC_POWER_CLASS_8BIT_MASK (0xF0U)

  The power class current value bit mask when bus in 8 bit mode.
- #define MMC_CACHE_CONTROL_ENABLE (1U)

  mmc cache control enable

- #define MMC_CACHE_TRIGGER_FLUSH (1U)
  mmc cache flush
- #define MMC_DATA_BUS_WIDTH_TYPE_NUMBER (3U)
  The number of data bus width type.
- #define MMC_PARTITION_CONFIG_PARTITION_ACCESS_SHIFT (0U)
  The bit shift for PARTITION ACCESS filed in BOOT CONFIG (BOOT_CONFIG in Extend CSD)
- #define MMC_PARTITION_CONFIG_PARTITION_ACCESS_MASK (0x00000007U)
  The bit mask for PARTITION ACCESS field in BOOT CONFIG.
- #define MMC_PARTITION_CONFIG_PARTITION_ENABLE_SHIFT (3U)
  The bit shift for PARTITION ENABLE field in BOOT CONFIG.
- #define MMC_PARTITION_CONFIG_PARTITION_ENABLE_MASK (0x00000038U)
  The bit mask for PARTITION ENABLE field in BOOT CONFIG.
- #define MMC_PARTITION_CONFIG_BOOT_ACK_SHIFT (6U)
  The bit shift for ACK field in BOOT CONFIG.
- #define MMC_PARTITION_CONFIG_BOOT_ACK_MASK (0x00000040U)
  The bit mask for ACK field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_BUS_WIDTH_SHIFT (0U)
  The bit shift for BOOT BUS WIDTH field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_BUS_WIDTH_MASK (3U)
  The bit mask for BOOT BUS WIDTH field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_RESET_BUS_CONDITION_SHIFT (2U)
  The bit shift for BOOT BUS WIDTH RESET field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_RESET_BUS_CONDITION_MASK (4U)
  The bit mask for BOOT BUS WIDTH RESET field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_BOOT_MODE_SHIFT (3U)
  The bit shift for BOOT MODE field in BOOT CONFIG.
- #define MMC_BOOT_BUS_CONDITION_BOOT_MODE_MASK (0x18U)
  The bit mask for BOOT MODE field in BOOT CONFIG.
- #define MMC_EXTENDED_CSD_BYTES (512U)
  The length of Extended CSD register, unit as bytes.
- #define MMC_DEFAULT_RELATIVE_ADDRESS (2UL)
  MMC card default relative address.
- #define SD_PRODUCT_NAME_BYTES (5U)
  SD card product name length united as bytes.
- #define SD_AU_START_VALUE (1U)
  SD AU start value.
- #define SD_UHS_AU_START_VALUE (7U)
  SD UHS AU start value.
- #define SD_TRANSFER_SPEED_RATE_UNIT_SHIFT (0U)
  The bit shift for RATE UNIT field in TRANSFER SPEED.
- #define SD_TRANSFER_SPEED_RATE_UNIT_MASK (0x07U)
  The bit mask for RATE UNIT field in TRANSFER SPEED.
- #define SD_TRANSFER_SPEED_TIME_VALUE_SHIFT (2U)
  The bit shift for TIME VALUE field in TRANSFER SPEED.
- #define SD_TRANSFER_SPEED_TIME_VALUE_MASK (0x78U)
  The bit mask for TIME VALUE field in TRANSFER SPEED.
- #define SD_RD_TRANSFER_SPEED_RATE_UNIT(x) (((x.transferSpeed) & SD_TRANSFER_-SPEED_RATE_UNIT_MASK) >> SD_TRANSFER_SPEED_RATE_UNIT_SHIFT)
  Read the value of FREQUENCY UNIT in TRANSFER SPEED field.
- #define SD_RD_TRANSFER_SPEED_TIME_VALUE(x) (((x.transferSpeed) & SD_TRANSFER-_SPEED_TIME_VALUE_MASK) >> SD_TRANSFER_SPEED_TIME_VALUE_SHIFT)

Read the value of TIME VALUE in TRANSFER SPEED field.

- #define MMC_PRODUCT_NAME_BYTES (6U)
  MMC card product name length united as bytes.
- #define MMC_SWITCH_COMMAND_SET_SHIFT (0U)
  The bit shift for COMMAND SET field in SWITCH command.
- #define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)
  The bit mask for COMMAND set field in SWITCH command.
- #define MMC_SWITCH_VALUE_SHIFT (8U)
  The bit shift for VALUE field in SWITCH command.
- #define MMC_SWITCH_VALUE_MASK (0x0000FF00U)
  The bit mask for VALUE field in SWITCH command.
- #define MMC_SWITCH_BYTE_INDEX_SHIFT (16U)
  The bit shift for BYTE INDEX field in SWITCH command.
- #define MMC_SWITCH_BYTE_INDEX_MASK (0x00FF0000U)
  The bit mask for BYTE INDEX field in SWITCH command.
- #define MMC_SWITCH_ACCESS_MODE_SHIFT (24U)
  The bit shift for ACCESS MODE field in SWITCH command.
- #define MMC_SWTICH_ACCESS_MODE_MASK (0x03000000U)
  The bit mask for ACCESS MODE field in SWITCH command.

Typedefs

- typedef void(∗ sd_cd_t)(bool isInserted, void ∗userData)
  card detect aoolication callback definition
- typedef bool(∗ sd_cd_status_t)(void)
  card detect status
- typedef void(∗ sd_io_voltage_func_t)(sdmmc_operation_voltage_t voltage)
  card switch voltage function pointer
- typedef void(∗ sd_pwr_t)(bool enable)
  card power control function pointer
- typedef void(∗ sd_io_strength_t)(uint32_t busFreq)
  card io strength control
- typedef void(∗ sdio_int_t)(void ∗userData)
  card interrupt function pointer

Enumerations

- enum {
  kStatus_SDMMC_NotSupportYet = MAKE_STATUS(kStatusGroup_SDMMC, 0U),
  kStatus_SDMMC_TransferFailed = MAKE_STATUS(kStatusGroup_SDMMC, 1U),
  kStatus_SDMMC_SetCardBlockSizeFailed = MAKE_STATUS(kStatusGroup_SDMMC, 2U),
  kStatus_SDMMC_HostNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 3U),
  kStatus_SDMMC_CardNotSupport = MAKE_STATUS(kStatusGroup_SDMMC, 4U),
  kStatus_SDMMC_AllSendCidFailed = MAKE_STATUS(kStatusGroup_SDMMC, 5U),
  kStatus_SDMMC_SendRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 6U),
  kStatus_SDMMC_SendCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 7U),
  kStatus_SDMMC_SelectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 8U),
  kStatus_SDMMC_SendScrFailed = MAKE_STATUS(kStatusGroup_SDMMC, 9U),
  kStatus_SDMMC_SetDataBusWidthFailed = MAKE_STATUS(kStatusGroup_SDMMC, 10U),
  kStatus_SDMMC_GoIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 11U),
  kStatus_SDMMC_HandShakeOperationConditionFailed,
  kStatus_SDMMC_SendApplicationCommandFailed,
  kStatus_SDMMC_SwitchFailed = MAKE_STATUS(kStatusGroup_SDMMC, 14U),
  kStatus_SDMMC_StopTransmissionFailed = MAKE_STATUS(kStatusGroup_SDMMC, 15U),
  kStatus_SDMMC_WaitWriteCompleteFailed = MAKE_STATUS(kStatusGroup_SDMMC, 16U),
  kStatus_SDMMC_SetBlockCountFailed = MAKE_STATUS(kStatusGroup_SDMMC, 17U),
  kStatus_SDMMC_SetRelativeAddressFailed = MAKE_STATUS(kStatusGroup_SDMMC, 18U),
  kStatus_SDMMC_SwitchBusTimingFailed = MAKE_STATUS(kStatusGroup_SDMMC, 19U),
  kStatus_SDMMC_SendExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 20U),
  kStatus_SDMMC_ConfigureBootFailed = MAKE_STATUS(kStatusGroup_SDMMC, 21U),
  kStatus_SDMMC_ConfigureExtendedCsdFailed = MAKE_STATUS(kStatusGroup_SDMMC, 22-U),
  kStatus_SDMMC_EnableHighCapacityEraseFailed,
  kStatus_SDMMC_SendTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 24U),
  kStatus_SDMMC_ReceiveTestPatternFailed = MAKE_STATUS(kStatusGroup_SDMMC, 25U),
  kStatus_SDMMC_SDIO_ResponseError = MAKE_STATUS(kStatusGroup_SDMMC, 26U),
  kStatus_SDMMC_SDIO_InvalidArgument,
  kStatus_SDMMC_SDIO_SendOperationConditionFail,
  kStatus_SDMMC_InvalidVoltage = MAKE_STATUS(kStatusGroup_SDMMC, 29U),
  kStatus_SDMMC_SDIO_SwitchHighSpeedFail = MAKE_STATUS(kStatusGroup_SDMMC, 30-U),
  kStatus_SDMMC_SDIO_ReadCISFail = MAKE_STATUS(kStatusGroup_SDMMC, 31U),
  kStatus_SDMMC_SDIO_InvalidCard = MAKE_STATUS(kStatusGroup_SDMMC, 32U),
  kStatus_SDMMC_TuningFail = MAKE_STATUS(kStatusGroup_SDMMC, 33U),
  kStatus_SDMMC_SwitchVoltageFail = MAKE_STATUS(kStatusGroup_SDMMC, 34U),
  kStatus_SDMMC_SwitchVoltage18VFail33VSuccess = MAKE_STATUS(kStatusGroup_SDMM-

C, 35U),
kStatus_SDMMC_ReTuningRequest = MAKE_STATUS(kStatusGroup_SDMMC, 36U),
kStatus_SDMMC_SetDriverStrengthFail = MAKE_STATUS(kStatusGroup_SDMMC, 37U),
kStatus_SDMMC_SetPowerClassFail = MAKE_STATUS(kStatusGroup_SDMMC, 38U),
kStatus_SDMMC_HostNotReady = MAKE_STATUS(kStatusGroup_SDMMC, 39U),
kStatus_SDMMC_CardDetectFailed = MAKE_STATUS(kStatusGroup_SDMMC, 40U),
kStatus_SDMMC_AuSizeNotSetProperly = MAKE_STATUS(kStatusGroup_SDMMC, 41U),
kStatus_SDMMC_PollingCardIdleFailed = MAKE_STATUS(kStatusGroup_SDMMC, 42U),
kStatus_SDMMC_DeselectCardFailed = MAKE_STATUS(kStatusGroup_SDMMC, 43U),
kStatus_SDMMC_CardStatusIdle = MAKE_STATUS(kStatusGroup_SDMMC, 44U),
kStatus_SDMMC_CardStatusBusy = MAKE_STATUS(kStatusGroup_SDMMC, 45U),
kStatus_SDMMC_CardInitFailed = MAKE_STATUS(kStatusGroup_SDMMC, 46U) }

SD/MMC card API's running status.

- enum {
  kSDMMC_SignalLineCmd = 1U,
  kSDMMC_SignalLineData0 = 2U,
  kSDMMC_SignalLineData1 = 4U,
  kSDMMC_SignalLineData2 = 8U,
  kSDMMC_SignalLineData3 = 16U,
  kSDMMC_SignalLineData4 = 32U,
  kSDMMC_SignalLineData5 = 64U,
  kSDMMC_SignalLineData6 = 128U,
  kSDMMC_SignalLineData7 = 256U }

  sdmmc signal line
- enum sdmmc_operation_voltage_t {
  kSDMMC_OperationVoltageNone = 0U,
  kSDMMC_OperationVoltage330V = 1U,
  kSDMMC_OperationVoltage300V = 2U,
  kSDMMC_OperationVoltage180V = 3U }

  card operation voltage
- enum {
  kSDMMC_BusWdith1Bit = 0U,
  kSDMMC_BusWdith4Bit = 1U,
  kSDMMC_BusWdith8Bit = 2U }

  card bus width
- enum { kSDMMC_Support8BitWidth = 1U }

  sdmmc capability flag
- enum {
  kSDMMC_DataPacketFormatLSBFirst,
  kSDMMC_DataPacketFormatMSBFirst }

  @ brief sdmmc data packet format
- enum sd_detect_card_type_t {
  kSD_DetectCardByGpioCD,
  kSD_DetectCardByHostCD,
  kSD_DetectCardByHostDATA3 }

  sd card detect type

- enum {
  kSD_Inserted = 1U,
  kSD_Removed = 0U }
  @ brief SD card detect status
- enum {
  kSD_DAT3PullDown = 0U,
  kSD_DAT3PullUp = 1U }
  @ brief SD card detect status
- enum sd_io_voltage_ctrl_type_t {
  kSD_IOVoltageCtrlNotSupport = 0U,
  kSD_IOVoltageCtrlByHost = 1U,
  kSD_IOVoltageCtrlByGpio = 2U }
  io voltage control type
- enum {
  kSDMMC_R1OutOfRangeFlag = 31,
  kSDMMC_R1AddressErrorFlag = 30,
  kSDMMC_R1BlockLengthErrorFlag = 29,
  kSDMMC_R1EraseSequenceErrorFlag = 28,
  kSDMMC_R1EraseParameterErrorFlag = 27,
  kSDMMC_R1WriteProtectViolationFlag = 26,
  kSDMMC_R1CardIsLockedFlag = 25,
  kSDMMC_R1LockUnlockFailedFlag = 24,
  kSDMMC_R1CommandCrcErrorFlag = 23,
  kSDMMC_R1IllegalCommandFlag = 22,
  kSDMMC_R1CardEccFailedFlag = 21,
  kSDMMC_R1CardControllerErrorFlag = 20,
  kSDMMC_R1ErrorFlag = 19,
  kSDMMC_R1CidCsdOverwriteFlag = 16,
  kSDMMC_R1WriteProtectEraseSkipFlag = 15,
  kSDMMC_R1CardEccDisabledFlag = 14,
  kSDMMC_R1EraseResetFlag = 13,
  kSDMMC_R1ReadyForDataFlag = 8,
  kSDMMC_R1SwitchErrorFlag = 7,
  kSDMMC_R1ApplicationCommandFlag = 5,
  kSDMMC_R1AuthenticationSequenceErrorFlag = 3 }
  Card status bit in R1.
- enum sdmmc_r1_current_state_t {
  kSDMMC_R1StateIdle = 0U,
  kSDMMC_R1StateReady = 1U,
  kSDMMC_R1StateIdentify = 2U,
  kSDMMC_R1StateStandby = 3U,
  kSDMMC_R1StateTransfer = 4U,
  kSDMMC_R1StateSendData = 5U,
  kSDMMC_R1StateReceiveData = 6U,
  kSDMMC_R1StateProgram = 7U,
  kSDMMC_R1StateDisconnect = 8U }

CURRENT_STATE filed in R1.

- enum {
  kSDSPI_R1InIdleStateFlag = (1U << 0U),
  kSDSPI_R1EraseResetFlag = (1U << 1U),
  kSDSPI_R1IllegalCommandFlag = (1U << 2U),
  kSDSPI_R1CommandCrcErrorFlag = (1U << 3U),
  kSDSPI_R1EraseSequenceErrorFlag = (1U << 4U),
  kSDSPI_R1AddressErrorFlag = (1U << 5U),
  kSDSPI_R1ParameterErrorFlag = (1U << 6U) }

  Error bit in SPI mode R1.
- enum {
  kSDSPI_R2CardLockedFlag = (1U << 0U),
  kSDSPI_R2WriteProtectEraseSkip = (1U << 1U),
  kSDSPI_R2LockUnlockFailed = (1U << 1U),
  kSDSPI_R2ErrorFlag = (1U << 2U),
  kSDSPI_R2CardControllerErrorFlag = (1U << 3U),
  kSDSPI_R2CardEccFailedFlag = (1U << 4U),
  kSDSPI_R2WriteProtectViolationFlag = (1U << 5U),
  kSDSPI_R2EraseParameterErrorFlag = (1U << 6U),
  kSDSPI_R2OutOfRangeFlag = (1U << 7U),
  kSDSPI_R2CsdOverwriteFlag = (1U << 7U) }

  Error bit in SPI mode R2.
- enum {
  kSDSPI_DataErrorTokenError = (1U << 0U),
  kSDSPI_DataErrorTokenCardControllerError = (1U << 1U),
  kSDSPI_DataErrorTokenCardEccFailed = (1U << 2U),
  kSDSPI_DataErrorTokenOutOfRange = (1U << 3U) }

  Data Error Token mask bit.
- enum sdspi_data_token_t {
  kSDSPI_DataTokenBlockRead = 0xFEU,
  kSDSPI_DataTokenSingleBlockWrite = 0xFEU,
  kSDSPI_DataTokenMultipleBlockWrite = 0xFCU,
  kSDSPI_DataTokenStopTransfer = 0xFDU }

  Data Token.
- enum sdspi_data_response_token_t {
  kSDSPI_DataResponseTokenAccepted = 0x05U,
  kSDSPI_DataResponseTokenCrcError = 0x0BU,
  kSDSPI_DataResponseTokenWriteError = 0x0DU }

  Data Response Token.
- enum sd_command_t {

kSD_SendRelativeAddress = 3U,
kSD_Switch = 6U,
kSD_SendInterfaceCondition = 8U,
kSD_VoltageSwitch = 11U,
kSD_SpeedClassControl = 20U,
kSD_EraseWriteBlockStart = 32U,
kSD_EraseWriteBlockEnd = 33U,
kSD_SendTuningBlock = 19U }

SD card individual commands.

- enum sdspi_command_t { kSDSPI_CommandCrc = 59U }

  SDSPI individual commands.

- enum sd_application_command_t {
  kSD_ApplicationSetBusWdith = 6U,
  kSD_ApplicationStatus = 13U,
  kSD_ApplicationSendNumberWriteBlocks = 22U,
  kSD_ApplicationSetWriteBlockEraseCount = 23U,
  kSD_ApplicationSendOperationCondition = 41U,
  kSD_ApplicationSetClearCardDetect = 42U,
  kSD_ApplicationSendScr = 51U }

  SD card individual application commands.

- enum {
  kSDMMC_CommandClassBasic = (1U << 0U),
  kSDMMC_CommandClassBlockRead = (1U << 2U),
  kSDMMC_CommandClassBlockWrite = (1U << 4U),
  kSDMMC_CommandClassErase = (1U << 5U),
  kSDMMC_CommandClassWriteProtect = (1U << 6U),
  kSDMMC_CommandClassLockCard = (1U << 7U),
  kSDMMC_CommandClassApplicationSpecific = (1U << 8U),
  kSDMMC_CommandClassInputOutputMode = (1U << 9U),
  kSDMMC_CommandClassSwitch = (1U << 10U) }

  SD card command class.

- enum {
  kSD_OcrPowerUpBusyFlag = 31,
  kSD_OcrHostCapacitySupportFlag = 30,
  kSD_OcrCardCapacitySupportFlag = kSD_OcrHostCapacitySupportFlag,
  kSD_OcrSwitch18RequestFlag = 24,
  kSD_OcrSwitch18AcceptFlag = kSD_OcrSwitch18RequestFlag,
  kSD_OcrVdd27_28Flag = 15,
  kSD_OcrVdd28_29Flag = 16,
  kSD_OcrVdd29_30Flag = 17,
  kSD_OcrVdd30_31Flag = 18,
  kSD_OcrVdd31_32Flag = 19,
  kSD_OcrVdd32_33Flag = 20,
  kSD_OcrVdd33_34Flag = 21,
  kSD_OcrVdd34_35Flag = 22,
  kSD_OcrVdd35_36Flag = 23 }

OCR register in SD card.

- enum {
  kSD_SpecificationVersion1_0 = (1U << 0U),
  kSD_SpecificationVersion1_1 = (1U << 1U),
  kSD_SpecificationVersion2_0 = (1U << 2U),
  kSD_SpecificationVersion3_0 = (1U << 3U) }

  SD card specification version number.
- enum sd_switch_mode_t {
  kSD_SwitchCheck = 0U,
  kSD_SwitchSet = 1U }

  SD card switch mode.
- enum {
  kSD_CsdReadBlockPartialFlag = (1U << 0U),
  kSD_CsdWriteBlockMisalignFlag = (1U << 1U),
  kSD_CsdReadBlockMisalignFlag = (1U << 2U),
  kSD_CsdDsrImplementedFlag = (1U << 3U),
  kSD_CsdEraseBlockEnabledFlag = (1U << 4U),
  kSD_CsdWriteProtectGroupEnabledFlag = (1U << 5U),
  kSD_CsdWriteBlockPartialFlag = (1U << 6U),
  kSD_CsdFileFormatGroupFlag = (1U << 7U),
  kSD_CsdCopyFlag = (1U << 8U),
  kSD_CsdPermanentWriteProtectFlag = (1U << 9U),
  kSD_CsdTemporaryWriteProtectFlag = (1U << 10U) }

  SD card CSD register flags.
- enum {
  kSD_ScrDataStatusAfterErase = (1U << 0U),
  kSD_ScrSdSpecification3 = (1U << 1U) }

  SD card SCR register flags.
- enum {
  kSD_FunctionSDR12Deafult = 0U,
  kSD_FunctionSDR25HighSpeed = 1U,
  kSD_FunctionSDR50 = 2U,
  kSD_FunctionSDR104 = 3U,
  kSD_FunctionDDR50 = 4U }

  SD timing function number.
- enum {
  kSD_GroupTimingMode = 0U,
  kSD_GroupCommandSystem = 1U,
  kSD_GroupDriverStrength = 2U,
  kSD_GroupCurrentLimit = 3U }

  SD group number.
- enum sd_timing_mode_t {
  kSD_TimingSDR12DefaultMode = 0U,
  kSD_TimingSDR25HighSpeedMode = 1U,
  kSD_TimingSDR50Mode = 2U,
  kSD_TimingSDR104Mode = 3U,

kSD_TimingDDR50Mode = 4U }

SD card timing mode flags.

- enum sd_driver_strength_t {
  kSD_DriverStrengthTypeB = 0U,
  kSD_DriverStrengthTypeA = 1U,
  kSD_DriverStrengthTypeC = 2U,
  kSD_DriverStrengthTypeD = 3U }

  SD card driver strength.
- enum sd_max_current_t {
  kSD_CurrentLimit200MA = 0U,
  kSD_CurrentLimit400MA = 1U,
  kSD_CurrentLimit600MA = 2U,
  kSD_CurrentLimit800MA = 3U }

  SD card current limit.
- enum sdmmc_command_t {
  kSDMMC_GoIdleState = 0U,
  kSDMMC_AllSendCid = 2U,
  kSDMMC_SetDsr = 4U,
  kSDMMC_SelectCard = 7U,
  kSDMMC_SendCsd = 9U,
  kSDMMC_SendCid = 10U,
  kSDMMC_StopTransmission = 12U,
  kSDMMC_SendStatus = 13U,
  kSDMMC_GoInactiveState = 15U,
  kSDMMC_SetBlockLength = 16U,
  kSDMMC_ReadSingleBlock = 17U,
  kSDMMC_ReadMultipleBlock = 18U,
  kSDMMC_SetBlockCount = 23U,
  kSDMMC_WriteSingleBlock = 24U,
  kSDMMC_WriteMultipleBlock = 25U,
  kSDMMC_ProgramCsd = 27U,
  kSDMMC_SetWriteProtect = 28U,
  kSDMMC_ClearWriteProtect = 29U,
  kSDMMC_SendWriteProtect = 30U,
  kSDMMC_Erase = 38U,
  kSDMMC_LockUnlock = 42U,
  kSDMMC_ApplicationCommand = 55U,
  kSDMMC_GeneralCommand = 56U,
  kSDMMC_ReadOcr = 58U }

  SD/MMC card common commands.
- enum {

kSDIO_RegCCCRSdioVer = 0x00U,
kSDIO_RegSDVersion = 0x01U,
kSDIO_RegIOEnable = 0x02U,
kSDIO_RegIOReady = 0x03U,
kSDIO_RegIOIntEnable = 0x04U,
kSDIO_RegIOIntPending = 0x05U,
kSDIO_RegIOAbort = 0x06U,
kSDIO_RegBusInterface = 0x07U,
kSDIO_RegCardCapability = 0x08U,
kSDIO_RegCommonCISPointer = 0x09U,
kSDIO_RegBusSuspend = 0x0C,
kSDIO_RegFunctionSelect = 0x0DU,
kSDIO_RegExecutionFlag = 0x0EU,
kSDIO_RegReadyFlag = 0x0FU,
kSDIO_RegFN0BlockSizeLow = 0x10U,
kSDIO_RegFN0BlockSizeHigh = 0x11U,
kSDIO_RegPowerControl = 0x12U,
kSDIO_RegBusSpeed = 0x13U,
kSDIO_RegUHSITimingSupport = 0x14U,
kSDIO_RegDriverStrength = 0x15U,
kSDIO_RegInterruptExtension = 0x16U }

sdio card cccr register addr

- enum sdio_command_t {
  kSDIO_SendRelativeAddress = 3U,
  kSDIO_SendOperationCondition = 5U,
  kSDIO_SendInterfaceCondition = 8U,
  kSDIO_RWIODirect = 52U,
  kSDIO_RWIOExtended = 53U }

  sdio card individual commands
- enum sdio_func_num_t {
  kSDIO_FunctionNum0,
  kSDIO_FunctionNum1,
  kSDIO_FunctionNum2,
  kSDIO_FunctionNum3,
  kSDIO_FunctionNum4,
  kSDIO_FunctionNum5,
  kSDIO_FunctionNum6,
  kSDIO_FunctionNum7,
  kSDIO_FunctionMemory }

  sdio card individual commands
- enum {

kSDIO_StatusCmdCRCError = 0x8000U,
kSDIO_StatusIllegalCmd = 0x4000U,
kSDIO_StatusR6Error = 0x2000U,
kSDIO_StatusError = 0x0800U,
kSDIO_StatusFunctionNumError = 0x0200U,
kSDIO_StatusOutofRange = 0x0100U }

sdio command response flag

- enum {
  kSDIO_OcrPowerUpBusyFlag = 31,
  kSDIO_OcrIONumber = 28,
  kSDIO_OcrMemPresent = 27,
  kSDIO_OcrVdd20_21Flag = 8,
  kSDIO_OcrVdd21_22Flag = 9,
  kSDIO_OcrVdd22_23Flag = 10,
  kSDIO_OcrVdd23_24Flag = 11,
  kSDIO_OcrVdd24_25Flag = 12,
  kSDIO_OcrVdd25_26Flag = 13,
  kSDIO_OcrVdd26_27Flag = 14,
  kSDIO_OcrVdd27_28Flag = 15,
  kSDIO_OcrVdd28_29Flag = 16,
  kSDIO_OcrVdd29_30Flag = 17,
  kSDIO_OcrVdd30_31Flag = 18,
  kSDIO_OcrVdd31_32Flag = 19,
  kSDIO_OcrVdd32_33Flag = 20,
  kSDIO_OcrVdd33_34Flag = 21,
  kSDIO_OcrVdd34_35Flag = 22,
  kSDIO_OcrVdd35_36Flag = 23 }

  sdio operation condition flag

- enum {
  kSDIO_CCCRSupportDirectCmdDuringDataTrans = (1UL << 0U),
  kSDIO_CCCRSupportMultiBlock = (1UL << 1U),
  kSDIO_CCCRSupportReadWait = (1UL << 2U),
  kSDIO_CCCRSupportSuspendResume = (1UL << 3U),
  kSDIO_CCCRSupportIntDuring4BitDataTrans = (1UL << 4U),
  kSDIO_CCCRSupportLowSpeed1Bit = (1UL << 6U),
  kSDIO_CCCRSupportLowSpeed4Bit = (1UL << 7U),
  kSDIO_CCCRSupportMasterPowerControl = (1UL << 8U),
  kSDIO_CCCRSupportHighSpeed = (1UL << 9U),
  kSDIO_CCCRSupportContinuousSPIInt = (1UL << 10U) }

  sdio capability flag

- enum {
  kSDIO_FBRSupportCSA = (1U << 0U),
  kSDIO_FBRSupportPowerSelection = (1U << 1U) }

  sdio fbr flag

- enum sdio_bus_width_t {

kSDIO_DataBus1Bit = 0x00U,
kSDIO_DataBus4Bit = 0X02U,
kSDIO_DataBus8Bit = 0X03U }
sdio bus width

- enum mmc_command_t {
kMMC_SendOperationCondition = 1U,
kMMC_SetRelativeAddress = 3U,
kMMC_SleepAwake = 5U,
kMMC_Switch = 6U,
kMMC_SendExtendedCsd = 8U,
kMMC_ReadDataUntilStop = 11U,
kMMC_BusTestRead = 14U,
kMMC_SendingBusTest = 19U,
kMMC_WriteDataUntilStop = 20U,
kMMC_SendTuningBlock = 21U,
kMMC_ProgramCid = 26U,
kMMC_EraseGroupStart = 35U,
kMMC_EraseGroupEnd = 36U,
kMMC_FastInputOutput = 39U,
kMMC_GoInterruptState = 40U }
MMC card individual commands.
- enum mmc_classified_voltage_t {
kMMC_ClassifiedVoltageHigh = 0U,
kMMC_ClassifiedVoltageDual = 1U }
MMC card classified as voltage range.
- enum mmc_classified_density_t { kMMC_ClassifiedDensityWithin2GB = 0U }
MMC card classified as density level.
- enum mmc_access_mode_t {
kMMC_AccessModeByte = 0U,
kMMC_AccessModeSector = 2U }
MMC card access mode(Access mode in OCR).
- enum mmc_voltage_window_t {
kMMC_VoltageWindowNone = 0U,
kMMC_VoltageWindow120 = 0x01U,
kMMC_VoltageWindow170to195 = 0x02U,
kMMC_VoltageWindows270to360 = 0x1FFU }
MMC card voltage window(VDD voltage window in OCR).
- enum mmc_csd_structure_version_t {
kMMC_CsdStrucureVersion10 = 0U,
kMMC_CsdStrucureVersion11 = 1U,
kMMC_CsdStrucureVersion12 = 2U,
kMMC_CsdStrucureVersionInExtcsd = 3U }
CSD structure version(CSD_STRUCTURE in CSD).
- enum mmc_specification_version_t {

kMMC_SpecificationVersion0 = 0U,
kMMC_SpecificationVersion1 = 1U,
kMMC_SpecificationVersion2 = 2U,
kMMC_SpecificationVersion3 = 3U,
kMMC_SpecificationVersion4 = 4U }

MMC card specification version(SPEC_VERS in CSD).

- enum {
  kMMC_ExtendedCsdRevision10 = 0U,
  kMMC_ExtendedCsdRevision11 = 1U,
  kMMC_ExtendedCsdRevision12 = 2U,
  kMMC_ExtendedCsdRevision13 = 3U,
  kMMC_ExtendedCsdRevision14 = 4U,
  kMMC_ExtendedCsdRevision15 = 5U,
  kMMC_ExtendedCsdRevision16 = 6U,
  kMMC_ExtendedCsdRevision17 = 7U }

  MMC card Extended CSD fix version(EXT_CSD_REV in Extended CSD)
- enum mmc_command_set_t {
  kMMC_CommandSetStandard = 0U,
  kMMC_CommandSet1 = 1U,
  kMMC_CommandSet2 = 2U,
  kMMC_CommandSet3 = 3U,
  kMMC_CommandSet4 = 4U }

  MMC card command set(COMMAND_SET in Extended CSD)
- enum {
  kMMC_SupportAlternateBoot = 1U,
  kMMC_SupportDDRBoot = 2U,
  kMMC_SupportHighSpeedBoot = 4U }

  boot support(BOOT_INFO in Extended CSD)
- enum mmc_high_speed_timing_t {
  kMMC_HighSpeedTimingNone = 0U,
  kMMC_HighSpeedTiming = 1U,
  kMMC_HighSpeed200Timing = 2U,
  kMMC_HighSpeed400Timing = 3U,
  kMMC_EnhanceHighSpeed400Timing = 4U }

  MMC card high-speed timing(HS_TIMING in Extended CSD)
- enum mmc_data_bus_width_t {
  kMMC_DataBusWidth1bit = 0U,
  kMMC_DataBusWidth4bit = 1U,
  kMMC_DataBusWidth8bit = 2U,
  kMMC_DataBusWidth4bitDDR = 5U,
  kMMC_DataBusWidth8bitDDR = 6U,
  kMMC_DataBusWidth8bitDDRSTROBE = 0x86U }

  MMC card data bus width(BUS_WIDTH in Extended CSD)
- enum mmc_boot_partition_enable_t {

kMMC_BootPartitionEnableNot = 0U,
kMMC_BootPartitionEnablePartition1 = 1U,
kMMC_BootPartitionEnablePartition2 = 2U,
kMMC_BootPartitionEnableUserAera = 7U }

MMC card boot partition enabled(BOOT_PARTITION_ENABLE in Extended CSD)

- enum mmc_boot_timing_mode_t {
  kMMC_BootModeSDRWithDefaultTiming = 0U,
  kMMC_BootModeSDRWithHighSpeedTiming = 1U,
  kMMC_BootModeDDRTiming = 2U }

  boot mode configuration Note: HS200 & HS400 is not support during BOOT operation.
- enum mmc_boot_partition_wp_t {
  kMMC_BootPartitionWPDisable = 0x50U,
  kMMC_BootPartitionPwrWPToBothPartition,
  kMMC_BootPartitionPermWPToBothPartition = 0x04U,
  kMMC_BootPartitionPwrWPToPartition1 = (1U << 7U) | 1U,
  kMMC_BootPartitionPwrWPToPartition2 = (1U << 7U) | 3U,
  kMMC_BootPartitionPermWPToPartition1,
  kMMC_BootPartitionPermWPToPartition2,
  kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2,
  kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 }

  MMC card boot partition write protect configurations All the bits in BOOT_WP register, except the two R/W bits B_PERM_WP_DIS and B_PERM_WP_EN, shall only be written once per power cycle.The protection mdde intended for both boot areas will be set with a single write.
- enum {
  kMMC_BootPartitionNotProtected = 0U,
  kMMC_BootPartitionPwrProtected = 1U,
  kMMC_BootPartitionPermProtected = 2U }

  MMC card boot partition write protect status.
- enum mmc_access_partition_t {
  kMMC_AccessPartitionUserAera = 0U,
  kMMC_AccessPartitionBoot1 = 1U,
  kMMC_AccessPartitionBoot2 = 2U,
  kMMC_AccessRPMB = 3U,
  kMMC_AccessGeneralPurposePartition1 = 4U,
  kMMC_AccessGeneralPurposePartition2 = 5U,
  kMMC_AccessGeneralPurposePartition3 = 6U,
  kMMC_AccessGeneralPurposePartition4 = 7U }

  MMC card partition to be accessed(BOOT_PARTITION_ACCESS in Extended CSD)
- enum {

kMMC_CsdReadBlockPartialFlag = (1U << 0U),
kMMC_CsdWriteBlockMisalignFlag = (1U << 1U),
kMMC_CsdReadBlockMisalignFlag = (1U << 2U),
kMMC_CsdDsrImplementedFlag = (1U << 3U),
kMMC_CsdWriteProtectGroupEnabledFlag = (1U << 4U),
kMMC_CsdWriteBlockPartialFlag = (1U << 5U),
kMMC_ContentProtectApplicationFlag = (1U << 6U),
kMMC_CsdFileFormatGroupFlag = (1U << 7U),
kMMC_CsdCopyFlag = (1U << 8U),
kMMC_CsdPermanentWriteProtectFlag = (1U << 9U),
kMMC_CsdTemporaryWriteProtectFlag = (1U << 10U) }

MMC card CSD register flags.

- enum mmc_extended_csd_access_mode_t {
  kMMC_ExtendedCsdAccessModeCommandSet = 0U,
  kMMC_ExtendedCsdAccessModeSetBits = 1U,
  kMMC_ExtendedCsdAccessModeClearBits = 2U,
  kMMC_ExtendedCsdAccessModeWriteBits = 3U }

  Extended CSD register access mode(Access mode in CMD6).
- enum mmc_extended_csd_index_t {
  kMMC_ExtendedCsdIndexFlushCache = 32U,
  kMMC_ExtendedCsdIndexCacheControl = 33U,
  kMMC_ExtendedCsdIndexBootPartitionWP = 173U,
  kMMC_ExtendedCsdIndexEraseGroupDefinition = 175U,
  kMMC_ExtendedCsdIndexBootBusConditions = 177U,
  kMMC_ExtendedCsdIndexBootConfigWP = 178U,
  kMMC_ExtendedCsdIndexPartitionConfig = 179U,
  kMMC_ExtendedCsdIndexBusWidth = 183U,
  kMMC_ExtendedCsdIndexHighSpeedTiming = 185U,
  kMMC_ExtendedCsdIndexPowerClass = 187U,
  kMMC_ExtendedCsdIndexCommandSet = 191U }

  EXT CSD byte index.
- enum {
  kMMC_DriverStrength0 = 0U,
  kMMC_DriverStrength1 = 1U,
  kMMC_DriverStrength2 = 2U,
  kMMC_DriverStrength3 = 3U,
  kMMC_DriverStrength4 = 4U }

  mmc driver strength
- enum mmc_extended_csd_flags_t {
  kMMC_ExtCsdExtPartitionSupport = (1 << 0U),
  kMMC_ExtCsdEnhancePartitionSupport = (1 << 1U),
  kMMC_ExtCsdPartitioningSupport = (1 << 2U),
  kMMC_ExtCsdPrgCIDCSDInDDRModeSupport = (1 << 3U),
  kMMC_ExtCsdBKOpsSupport = (1 << 4U),
  kMMC_ExtCsdDataTagSupport = (1 << 5U),
  kMMC_ExtCsdModeOperationCodeSupport = (1 << 6U) }

mmc extended csd flags

- enum mmc_boot_mode_t {
  kMMC_BootModeNormal = 0U,
  kMMC_BootModeAlternative = 1U }
  MMC card boot mode.

common function

tuning pattern

- status_t SDMMC_SelectCard (sdmmchost_t *host, uint32_t relativeAddress, bool isSelected)
  Selects the card to put it into transfer state.
- status_t SDMMC_SendApplicationCommand (sdmmchost_t *host, uint32_t relativeAddress)
  Sends an application command.
- status_t SDMMC_SetBlockCount (sdmmchost_t *host, uint32_t blockCount)
  Sets the block count.
- status_t SDMMC_GoIdle (sdmmchost_t *host)
  Sets the card to be idle state.
- status_t SDMMC_SetBlockSize (sdmmchost_t *host, uint32_t blockSize)
  Sets data block size.
- status_t SDMMC_SetCardInactive (sdmmchost_t *host)
  Sets card to inactive status.

50.7.2 Data Structure Documentation

50.7.2.1 struct sd_detect_card_t

Data Fields

- sd_detect_card_type_t type
  card detect type
- uint32_t cdDebounce_ms
  card detect debounce delay ms
- sd_cd_t callback
  card inserted callback which is meaningful for interrupt case
- sd_cd_status_t cardDetected
  used to check sd cd status when card detect through GPIO
- sd_dat3_pull_t dat3PullFunc
  function pointer of DATA3 pull up/down
- void * userData
  user data

50.7.2.2 struct sd_io_voltage_t

Data Fields

- sd_io_voltage_ctrl_type_t type

io voltage switch type
- sd_io_voltage_func_t func
  io voltage switch function

50.7.2.3 struct sd_usr_param_t

Data Fields

- sd_pwr_t pwr
  power control configuration pointer
- uint32_t powerOnDelayMS
  power on delay time
- uint32_t powerOffDelayMS
  power off delay time
- sd_io_strength_t ioStrength
  swicth sd io strength
- sd_io_voltage_t * ioVoltage
  switch io voltage
- sd_detect_card_t * cd
  card detect
- uint32_t maxFreq
  board support maximum frequency
- uint32_t capability
  board capability flag

50.7.2.4 struct sdio_card_int_t

Data Fields

- void * userData
  user data
- sdio_int_t cardInterrupt
  card int call back

50.7.2.5 struct sdio_usr_param_t

Data Fields

- sd_pwr_t pwr
  power control configuration pointer
- uint32_t powerOnDelayMS
  power on delay time
- uint32_t powerOffDelayMS
  power off delay time
- sd_io_strength_t ioStrength
  swicth sd io strength
- sd_io_voltage_t * ioVoltage
  switch io voltage

- sd_detect_card_t * cd
  card detect
- sdio_card_int_t * sdioInt
  card int
- uint32_t maxFreq
  board support maximum frequency
- uint32_t capability
  board capability flag

50.7.2.6 struct sdio_fbr_t

Data Fields

- uint8_t flags
  current io flags
- uint8_t ioStdFunctionCode
  current io standard function code
- uint8_t ioExtFunctionCode
  current io extended function code
- uint32_t ioPointerToCIS
  current io pointer to CIS
- uint32_t ioPointerToCSA
  current io pointer to CSA
- uint16_t ioBlockSize
  current io block size

50.7.2.7 struct sdio_common_cis_t

Data Fields

- uint16_t mID
  manufacturer code
- uint16_t mInfo
  manufacturer information
- uint8_t funcID
  function ID
- uint16_t fn0MaxBlkSize
  function 0 max block size
- uint8_t maxTransSpeed
  max data transfer speed for all function

50.7.2.8 struct sdio_func_cis_t

Data Fields

- uint8_t funcID
  function ID
- uint8_t funcInfo

function info

- uint8_t ioVersion
  level of application specification this io support
- uint32_t cardPSN
  product serial number
- uint32_t ioCSASize
  avaliable CSA size for io
- uint8_t ioCSAProperty
  CSA property.
- uint16_t ioMaxBlockSize
  io max transfer data size
- uint32_t ioOCR
  io ioeration condition
- uint8_t ioOPMinPwr
  min current in operation mode
- uint8_t ioOPAvgPwr
  average current in operation mode
- uint8_t ioOPMaxPwr
  max current in operation mode
- uint8_t ioSBMinPwr
  min current in standby mode
- uint8_t ioSBAvgPwr
  average current in standby mode
- uint8_t ioSBMaxPwr
  max current in standby mode
- uint16_t ioMinBandWidth
  io min transfer bandwidth
- uint16_t ioOptimumBandWidth
  io optimum transfer bandwidth
- uint16_t ioReadyTimeout
  timeout value from enalbe to ready
- uint16_t ioHighCurrentAvgCurrent
  `the average peak current (mA)`
  when IO operating in high current mode
- uint16_t ioHighCurrentMaxCurrent
  `the max peak current (mA)`
  when IO operating in high current mode
- uint16_t ioLowCurrentAvgCurrent
  `the average peak current (mA)`
  when IO operating in lower current mode
- uint16_t ioLowCurrentMaxCurrent
  `the max peak current (mA)`
  when IO operating in lower current mode

50.7.2.9 struct sd_status_t

Data Fields

- uint8_t busWidth
  current buswidth

- uint8_t secureMode
  secured mode
- uint16_t cardType
  sdcard type
- uint32_t protectedSize
  size of protected area
- uint8_t speedClass
  speed class of card
- uint8_t performanceMove
  Performance of move indicated by 1[MB/S]step.
- uint8_t auSize
  size of AU
- uint16_t eraseSize
  number of AUs to be erased at a time
- uint8_t eraseTimeout
  timeout value for erasing areas specified by UNIT OF ERASE AU
- uint8_t eraseOffset
  fixed offset value added to erase time
- uint8_t uhsSpeedGrade
  speed grade for UHS mode
- uint8_t uhsAuSize
  size of AU for UHS mode

50.7.2.10 struct sd_cid_t

Data Fields

- uint8_t manufacturerID
  Manufacturer ID [127:120].
- uint16_t applicationID
  OEM/Application ID [119:104].
- uint8_t productName [SD_PRODUCT_NAME_BYTES]
  Product name [103:64].
- uint8_t productVersion
  Product revision [63:56].
- uint32_t productSerialNumber
  Product serial number [55:24].
- uint16_t manufacturerData
  Manufacturing date [19:8].

50.7.2.11 struct sd_csd_t

Data Fields

- uint8_t csdStructure
  CSD structure [127:126].
- uint8_t dataReadAccessTime1
  Data read access-time-1 [119:112].
- uint8_t dataReadAccessTime2

Data read access-time-2 in clock cycles (NSAC∗100) [111:104].
- uint8_t transferSpeed
  Maximum data transfer rate [103:96].
- uint16_t cardCommandClass
  Card command classes [95:84].
- uint8_t readBlockLength
  Maximum read data block length [83:80].
- uint16_t flags
  Flags in _sd_csd_flag.
- uint32_t deviceSize
  Device size [73:62].
- uint8_t readCurrentVddMin
  Maximum read current at VDD min [61:59].
- uint8_t readCurrentVddMax
  Maximum read current at VDD max [58:56].
- uint8_t writeCurrentVddMin
  Maximum write current at VDD min [55:53].
- uint8_t writeCurrentVddMax
  Maximum write current at VDD max [52:50].
- uint8_t deviceSizeMultiplier
  Device size multiplier [49:47].
- uint8_t eraseSectorSize
  Erase sector size [45:39].
- uint8_t writeProtectGroupSize
  Write protect group size [38:32].
- uint8_t writeSpeedFactor
  Write speed factor [28:26].
- uint8_t writeBlockLength
  Maximum write data block length [25:22].
- uint8_t fileFormat
  File format [11:10].

50.7.2.12 struct sd_scr_t

Data Fields

- uint8_t scrStructure
  SCR Structure [63:60].
- uint8_t sdSpecification
  SD memory card specification version [59:56].
- uint16_t flags
  SCR flags in _sd_scr_flag.
- uint8_t sdSecurity
  Security specification supported [54:52].
- uint8_t sdBusWidths
  Data bus widths supported [51:48].
- uint8_t extendedSecurity
  Extended security support [46:43].
- uint8_t commandSupport
  Command support bits [33:32] 33-support CMD23, 32-support cmd20.

- uint32_t reservedForManufacturer
  reserved for manufacturer usage [31:0]

50.7.2.13 struct mmc_cid_t

Data Fields

- uint8_t manufacturerID
  Manufacturer ID.
- uint16_t applicationID
  OEM/Application ID.
- uint8_t productName [MMC_PRODUCT_NAME_BYTES]
  Product name.
- uint8_t productVersion
  Product revision.
- uint32_t productSerialNumber
  Product serial number.
- uint8_t manufacturerData
  Manufacturing date.

50.7.2.14 struct mmc_csd_t

Data Fields

- uint8_t csdStructureVersion
  CSD structure [127:126].
- uint8_t systemSpecificationVersion
  System specification version [125:122].
- uint8_t dataReadAccessTime1
  Data read access-time 1 [119:112].
- uint8_t dataReadAccessTime2
  Data read access-time 2 in CLOCK cycles (NSAC∗100) [111:104].
- uint8_t transferSpeed
  Max.
- uint16_t cardCommandClass
  card command classes [95:84]
- uint8_t readBlockLength
  Max.
- uint16_t flags
  Contain flags in _mmc_csd_flag.
- uint16_t deviceSize
  Device size [73:62].
- uint8_t readCurrentVddMin
  Max.
- uint8_t readCurrentVddMax
  Max.
- uint8_t writeCurrentVddMin
  Max.
- uint8_t writeCurrentVddMax

Max.
- uint8_t deviceSizeMultiplier
  Device size multiplier [49:47].
- uint8_t eraseGroupSize
  Erase group size [46:42].
- uint8_t eraseGroupSizeMultiplier
  Erase group size multiplier [41:37].
- uint8_t writeProtectGroupSize
  Write protect group size [36:32].
- uint8_t defaultEcc
  Manufacturer default ECC [30:29].
- uint8_t writeSpeedFactor
  Write speed factor [28:26].
- uint8_t maxWriteBlockLength
  Max.
- uint8_t fileFormat
  File format [11:10].
- uint8_t eccCode
  ECC code [9:8].

Field Documentation

(1) uint8_t mmc_csd_t::transferSpeed

bus clock frequency [103:96]

(2) uint8_t mmc_csd_t::readBlockLength

read data block length [83:80]

(3) uint8_t mmc_csd_t::readCurrentVddMin

read current @ VDD min [61:59]

(4) uint8_t mmc_csd_t::readCurrentVddMax

read current @ VDD max [58:56]

(5) uint8_t mmc_csd_t::writeCurrentVddMin

write current @ VDD min [55:53]

(6) uint8_t mmc_csd_t::writeCurrentVddMax

write current @ VDD max [52:50]

(7) uint8_t mmc_csd_t::maxWriteBlockLength

write data block length [25:22]

50.7.2.15 struct mmc_extended_csd_t

Data Fields

- uint8_t cacheCtrl
  < secure removal type[16]
- uint8_t partitionAttribute
  < power off notification[34]
- uint8_t userWP
  < max enhance area size [159-157]
- uint8_t bootPartitionWP
  boot write protect register[173]
- uint8_t bootWPStatus
  boot write protect status register[174]
- uint8_t highDensityEraseGroupDefinition
  High-density erase group definition [175].
- uint8_t bootDataBusConditions
  Boot bus conditions [177].
- uint8_t bootConfigProtect
  Boot config protection [178].
- uint8_t partitionConfig
  Boot configuration [179].
- uint8_t eraseMemoryContent
  Erased memory content [181].
- uint8_t dataBusWidth
  Data bus width mode [183].
- uint8_t highSpeedTiming
  High-speed interface timing [185].
- uint8_t powerClass
  Power class [187].
- uint8_t commandSetRevision
  Command set revision [189].
- uint8_t commandSet
  Command set [191].
- uint8_t extendecCsdVersion
  Extended CSD revision [192].
- uint8_t csdStructureVersion
  CSD structure version [194].
- uint8_t cardType
  Card Type [196].
- uint8_t ioDriverStrength
  IO driver strength [197].
- uint8_t partitionSwitchTimeout
  < out of interrupt busy timing [198]
- uint8_t powerClass52MHz195V
  Power Class for 52MHz @ 1.95V [200].
- uint8_t powerClass26MHz195V
  Power Class for 26MHz @ 1.95V [201].
- uint8_t powerClass52MHz360V
  Power Class for 52MHz @ 3.6V [202].
- uint8_t powerClass26MHz360V

Power Class for 26MHz @ 3.6V [203].

- uint8_t minimumReadPerformance4Bit26MHz

  Minimum Read Performance for 4bit at 26MHz [205].
- uint8_t minimumWritePerformance4Bit26MHz

  Minimum Write Performance for 4bit at 26MHz [206].
- uint8_t minimumReadPerformance8Bit26MHz4Bit52MHz

  Minimum read Performance for 8bit at 26MHz/4bit @52MHz [207].
- uint8_t minimumWritePerformance8Bit26MHz4Bit52MHz

  Minimum Write Performance for 8bit at 26MHz/4bit @52MHz [208].
- uint8_t minimumReadPerformance8Bit52MHz

  Minimum Read Performance for 8bit at 52MHz [209].
- uint8_t minimumWritePerformance8Bit52MHz

  Minimum Write Performance for 8bit at 52MHz [210].
- uint32_t sectorCount

  Sector Count [215:212].
- uint8_t sleepAwakeTimeout

  < sleep notification timeout [216]
- uint8_t sleepCurrentVCCQ

  < Production state awareness timeout [218]
- uint8_t sleepCurrentVCC

  Sleep current (VCC) [220].
- uint8_t highCapacityWriteProtectGroupSize

  High-capacity write protect group size [221].
- uint8_t reliableWriteSectorCount

  Reliable write sector count [222].
- uint8_t highCapacityEraseTimeout

  High-capacity erase timeout [223].
- uint8_t highCapacityEraseUnitSize

  High-capacity erase unit size [224].
- uint8_t accessSize

  Access size [225].
- uint8_t minReadPerformance8bitAt52MHZDDR

  < secure trim multiplier[229]
- uint8_t minWritePerformance8bitAt52MHZDDR

  Minimum write performance for 8bit at DDR 52MHZ[235].
- uint8_t powerClass200MHZVCCQ130VVCC360V

  power class for 200MHZ, at VCCQ= 1.3V,VCC=3.6V[236]
- uint8_t powerClass200MHZVCCQ195VVCC360V

  power class for 200MHZ, at VCCQ= 1.95V,VCC=3.6V[237]
- uint8_t powerClass52MHZDDR195V

  power class for 52MHZ,DDR at Vcc 1.95V[238]
- uint8_t powerClass52MHZDDR360V

  power class for 52MHZ,DDR at Vcc 3.6V[239]
- uint32_t genericCMD6Timeout

  < 1st initialization time after partitioning[241]
- uint32_t cacheSize

  cache size[252-249]
- uint8_t powerClass200MHZDDR360V

  power class for 200MHZ, DDR at VCC=2.6V[253]
- uint8_t extPartitionSupport

  < fw VERSION [261-254]

- uint8_t supportedCommandSet
  < large unit size[495]

Field Documentation

(1) uint8_t mmc_extended_csd_t::cacheCtrl

< product state awareness enablement[17]

< max preload data size[21-18]

< pre-load data size[25-22]

< FFU status [26]

< mode operation code[29]

< mode config [30] control to turn on/off cache[33]

(2) uint8_t mmc_extended_csd_t::partitionAttribute

< packed cmd fail index [35]

< packed cmd status[36]

< context configuration[51-37]

< extended partitions attribut[53-52]

< exception events status[55-54]

< exception events control[57-56]

< number of group to be released[58]

< class 6 command control[59]

< 1st initiallization after disabling sector size emu[60]

< sector size[61]

< sector size emulation[62]

< native sector size[63]

< period wakeup [131]

< package case temperature is controlled[132]

< production state awareness[133]

< enhanced user data start addr [139-136]

< enhanced user data area size[142-140]

< general purpose partition size[154-143] partition attribute [156]

(3) uint8_t mmc_extended_csd_t::userWP

< HPI management [161]

< write reliability parameter register[166]

< write reliability setting register[167]

< RPMB size multi [168]

< FW configuration[169] user write protect register[171]

(4) uint8_t mmc_extended_csd_t::partitionSwitchTimeout

partition switch timing [199]

(5) uint8_t mmc_extended_csd_t::sleepAwakeTimeout

Sleep/awake timeout [217]

(6) uint8_t mmc_extended_csd_t::sleepCurrentVCCQ

Sleep current (VCCQ) [219]

(7) uint8_t mmc_extended_csd_t::minReadPerformance8bitAt52MHZDDR

< secure erase multiplier[230]

< secure feature support[231]

< trim multiplier[232] Minimum read performance for 8bit at DDR 52MHZ[234]

(8) uint32_t mmc_extended_csd_t::genericCMD6Timeout

< correct prg sectors number[245-242]

< background operations status[246]

< power off notification timeout[247] generic CMD6 timeout[248]

(9) uint8_t mmc_extended_csd_t::extPartitionSupport

< device version[263-262]

< optimal trim size[264]

< optimal write size[265]

< optimal read size[266]

< pre EOL information[267]

< device life time estimation typeA[268]

< device life time estimation typeB[269]

< number of FW sectors correctly programmed[305-302]

< FFU argument[490-487]

< operation code timeout[491]

< support mode [493] extended partition attribute support[494]

(10) uint8_t mmc_extended_csd_t::supportedCommandSet

< context management capability[496]

< tag resource size[497]

< tag unit size[498]

< max packed write cmd[500]

< max packed read cmd[501]

< HPI feature[503] Supported Command Sets [504]

50.7.2.16 struct mmc_extended_csd_config_t

Data Fields

- mmc_command_set_t commandSet
  Command set.
- uint8_t ByteValue
  The value to set.
- uint8_t ByteIndex
  The byte index in Extended CSD(mmc_extended_csd_index_t)
- mmc_extended_csd_access_mode_t accessMode
  Access mode.

50.7.2.17 struct mmc_boot_config_t

Data Fields

- mmc_boot_mode_t bootMode
  mmc boot mode
- bool enableBootAck
  Enable boot ACK.
- mmc_boot_partition_enable_t bootPartition
  Boot partition.
- mmc_boot_timing_mode_t bootTimingMode
  boot mode
- mmc_data_bus_width_t bootDataBusWidth
  Boot data bus width.
- bool retainBootbusCondition
  If retain boot bus width and boot mode conditions.
- bool pwrBootConfigProtection
  `Disable the change of boot configuration register bits from at this point` *until next power cycle or next H/W reset operation*
- bool premBootConfigProtection
  Disable the change of boot configuration register bits permanently.
- mmc_boot_partition_wp_t bootPartitionWP

boot partition write protect configurations

50.7.3 Macro Definition Documentation

50.7.3.1 #define SDMMC_LOG(*format, ...*)

50.7.3.2 #define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(*CSD*) ((((CSD).transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)

50.7.3.3 #define READ_MMC_TRANSFER_SPEED_MULTIPLIER(*CSD*) ((((CSD).transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)

50.7.3.4 #define MMC_EXTENDED_CSD_BYTES (512U)

50.7.3.5 #define SD_PRODUCT_NAME_BYTES (5U)

50.7.3.6 #define MMC_PRODUCT_NAME_BYTES (6U)

50.7.3.7 #define MMC_SWITCH_COMMAND_SET_SHIFT (0U)

50.7.3.8 #define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)

50.7.4 Enumeration Type Documentation

50.7.4.1 anonymous enum

Enumerator

kStatus_SDMMC_NotSupportYet Haven't supported.
kStatus_SDMMC_TransferFailed Send command failed.
kStatus_SDMMC_SetCardBlockSizeFailed Set block size failed.
kStatus_SDMMC_HostNotSupport Host doesn't support.
kStatus_SDMMC_CardNotSupport Card doesn't support.
kStatus_SDMMC_AllSendCidFailed Send CID failed.
kStatus_SDMMC_SendRelativeAddressFailed Send relative address failed.
kStatus_SDMMC_SendCsdFailed Send CSD failed.
kStatus_SDMMC_SelectCardFailed Select card failed.
kStatus_SDMMC_SendScrFailed Send SCR failed.
kStatus_SDMMC_SetDataBusWidthFailed Set bus width failed.
kStatus_SDMMC_GoIdleFailed Go idle failed.
kStatus_SDMMC_HandShakeOperationConditionFailed Send Operation Condition failed.
kStatus_SDMMC_SendApplicationCommandFailed Send application command failed.

kStatus_SDMMC_SwitchFailed Switch command failed.
kStatus_SDMMC_StopTransmissionFailed Stop transmission failed.
kStatus_SDMMC_WaitWriteCompleteFailed Wait write complete failed.
kStatus_SDMMC_SetBlockCountFailed Set block count failed.
kStatus_SDMMC_SetRelativeAddressFailed Set relative address failed.
kStatus_SDMMC_SwitchBusTimingFailed Switch high speed failed.
kStatus_SDMMC_SendExtendedCsdFailed Send EXT_CSD failed.
kStatus_SDMMC_ConfigureBootFailed Configure boot failed.
kStatus_SDMMC_ConfigureExtendedCsdFailed Configure EXT_CSD failed.
kStatus_SDMMC_EnableHighCapacityEraseFailed Enable high capacity erase failed.
kStatus_SDMMC_SendTestPatternFailed Send test pattern failed.
kStatus_SDMMC_ReceiveTestPatternFailed Receive test pattern failed.
kStatus_SDMMC_SDIO_ResponseError sdio response error
kStatus_SDMMC_SDIO_InvalidArgument sdio invalid argument response error
kStatus_SDMMC_SDIO_SendOperationConditionFail sdio send operation condition fail
kStatus_SDMMC_InvalidVoltage invaild voltage
kStatus_SDMMC_SDIO_SwitchHighSpeedFail switch to high speed fail
kStatus_SDMMC_SDIO_ReadCISFail read CIS fail
kStatus_SDMMC_SDIO_InvalidCard invaild SDIO card
kStatus_SDMMC_TuningFail tuning fail
kStatus_SDMMC_SwitchVoltageFail switch voltage fail
kStatus_SDMMC_SwitchVoltage18VFail33VSuccess switch voltage fail
kStatus_SDMMC_ReTuningRequest retuning request
kStatus_SDMMC_SetDriverStrengthFail set driver strength fail
kStatus_SDMMC_SetPowerClassFail set power class fail
kStatus_SDMMC_HostNotReady host controller not ready
kStatus_SDMMC_CardDetectFailed card detect failed
kStatus_SDMMC_AuSizeNotSetProperly AU size not set properly.
kStatus_SDMMC_PollingCardIdleFailed polling card idle status failed
kStatus_SDMMC_DeselectCardFailed deselect card failed
kStatus_SDMMC_CardStatusIdle card idle
kStatus_SDMMC_CardStatusBusy card busy
kStatus_SDMMC_CardInitFailed card init failed

50.7.4.2 anonymous enum

Enumerator

kSDMMC_SignalLineCmd cmd line
kSDMMC_SignalLineData0 data line
kSDMMC_SignalLineData1 data line
kSDMMC_SignalLineData2 data line
kSDMMC_SignalLineData3 data line
kSDMMC_SignalLineData4 data line
kSDMMC_SignalLineData5 data line

kSDMMC_SignalLineData6 data line
kSDMMC_SignalLineData7 data line

50.7.4.3 enum sdmmc_operation_voltage_t

Enumerator

kSDMMC_OperationVoltageNone indicate current voltage setting is not setting by suser
kSDMMC_OperationVoltage330V card operation voltage around 3.3v
kSDMMC_OperationVoltage300V card operation voltage around 3.0v
kSDMMC_OperationVoltage180V card operation voltage around 1.8v

50.7.4.4 anonymous enum

Enumerator

kSDMMC_BusWdith1Bit card bus 1 width
kSDMMC_BusWdith4Bit card bus 4 width
kSDMMC_BusWdith8Bit card bus 8 width

50.7.4.5 anonymous enum

Enumerator

kSDMMC_Support8BitWidth 8 bit data width capability

50.7.4.6 anonymous enum

Enumerator

kSDMMC_DataPacketFormatLSBFirst usual data packet format LSB first, MSB last
kSDMMC_DataPacketFormatMSBFirst Wide width data packet format MSB first, LSB last.

50.7.4.7 enum sd_detect_card_type_t

Enumerator

kSD_DetectCardByGpioCD sd card detect by CD pin through GPIO
kSD_DetectCardByHostCD sd card detect by CD pin through host
kSD_DetectCardByHostDATA3 sd card detect by DAT3 pin through host

50.7.4.8 anonymous enum

Enumerator

kSD_Inserted card is inserted
kSD_Removed card is removed

50.7.4.9 anonymous enum

Enumerator

kSD_DAT3PullDown data3 pull down
kSD_DAT3PullUp data3 pull up

50.7.4.10 enum sd_io_voltage_ctrl_type_t

Enumerator

kSD_IOVoltageCtrlNotSupport io voltage control not support
kSD_IOVoltageCtrlByHost io voltage control by host
kSD_IOVoltageCtrlByGpio io voltage control by gpio

50.7.4.11 anonymous enum

Enumerator

kSDMMC_R1OutOfRangeFlag Out of range status bit.
kSDMMC_R1AddressErrorFlag Address error status bit.
kSDMMC_R1BlockLengthErrorFlag Block length error status bit.
kSDMMC_R1EraseSequenceErrorFlag Erase sequence error status bit.
kSDMMC_R1EraseParameterErrorFlag Erase parameter error status bit.
kSDMMC_R1WriteProtectViolationFlag Write protection violation status bit.
kSDMMC_R1CardIsLockedFlag Card locked status bit.
kSDMMC_R1LockUnlockFailedFlag lock/unlock error status bit
kSDMMC_R1CommandCrcErrorFlag CRC error status bit.
kSDMMC_R1IllegalCommandFlag Illegal command status bit.
kSDMMC_R1CardEccFailedFlag Card ecc error status bit.
kSDMMC_R1CardControllerErrorFlag Internal card controller error status bit.
kSDMMC_R1ErrorFlag A general or an unknown error status bit.
kSDMMC_R1CidCsdOverwriteFlag Cid/csd overwrite status bit.
kSDMMC_R1WriteProtectEraseSkipFlag Write protection erase skip status bit.
kSDMMC_R1CardEccDisabledFlag Card ecc disabled status bit.
kSDMMC_R1EraseResetFlag Erase reset status bit.
kSDMMC_R1ReadyForDataFlag Ready for data status bit.
kSDMMC_R1SwitchErrorFlag Switch error status bit.

kSDMMC_R1ApplicationCommandFlag Application command enabled status bit.
kSDMMC_R1AuthenticationSequenceErrorFlag error in the sequence of authentication process

50.7.4.12 enum sdmmc_r1_current_state_t

Enumerator

kSDMMC_R1StateIdle R1: current state: idle.
kSDMMC_R1StateReady R1: current state: ready.
kSDMMC_R1StateIdentify R1: current state: identification.
kSDMMC_R1StateStandby R1: current state: standby.
kSDMMC_R1StateTransfer R1: current state: transfer.
kSDMMC_R1StateSendData R1: current state: sending data.
kSDMMC_R1StateReceiveData R1: current state: receiving data.
kSDMMC_R1StateProgram R1: current state: programming.
kSDMMC_R1StateDisconnect R1: current state: disconnect.

50.7.4.13 anonymous enum

Enumerator

kSDSPI_R1InIdleStateFlag In idle state.
kSDSPI_R1EraseResetFlag Erase reset.
kSDSPI_R1IllegalCommandFlag Illegal command.
kSDSPI_R1CommandCrcErrorFlag Com crc error.
kSDSPI_R1EraseSequenceErrorFlag Erase sequence error.
kSDSPI_R1AddressErrorFlag Address error.
kSDSPI_R1ParameterErrorFlag Parameter error.

50.7.4.14 anonymous enum

Enumerator

kSDSPI_R2CardLockedFlag Card is locked.
kSDSPI_R2WriteProtectEraseSkip Write protect erase skip.
kSDSPI_R2LockUnlockFailed Lock/unlock command failed.
kSDSPI_R2ErrorFlag Unknown error.
kSDSPI_R2CardControllerErrorFlag Card controller error.
kSDSPI_R2CardEccFailedFlag Card ecc failed.
kSDSPI_R2WriteProtectViolationFlag Write protect violation.
kSDSPI_R2EraseParameterErrorFlag Erase parameter error.
kSDSPI_R2OutOfRangeFlag Out of range.
kSDSPI_R2CsdOverwriteFlag CSD overwrite.

50.7.4.15 anonymous enum

Enumerator

kSDSPI_DataErrorTokenError Data error.
kSDSPI_DataErrorTokenCardControllerError Card controller error.
kSDSPI_DataErrorTokenCardEccFailed Card ecc error.
kSDSPI_DataErrorTokenOutOfRange Out of range.

50.7.4.16 enum sdspi_data_token_t

Enumerator

kSDSPI_DataTokenBlockRead Single block read, multiple block read.
kSDSPI_DataTokenSingleBlockWrite Single block write.
kSDSPI_DataTokenMultipleBlockWrite Multiple block write.
kSDSPI_DataTokenStopTransfer Stop transmission.

50.7.4.17 enum sdspi_data_response_token_t

Enumerator

kSDSPI_DataResponseTokenAccepted Data accepted.
kSDSPI_DataResponseTokenCrcError Data rejected due to CRC error.
kSDSPI_DataResponseTokenWriteError Data rejected due to write error.

50.7.4.18 enum sd_command_t

Enumerator

kSD_SendRelativeAddress Send Relative Address.
kSD_Switch Switch Function.
kSD_SendInterfaceCondition Send Interface Condition.
kSD_VoltageSwitch Voltage Switch.
kSD_SpeedClassControl Speed Class control.
kSD_EraseWriteBlockStart Write Block Start.
kSD_EraseWriteBlockEnd Write Block End.
kSD_SendTuningBlock Send Tuning Block.

50.7.4.19 enum sdspi_command_t

Enumerator

kSDSPI_CommandCrc Command crc protection on/off.

50.7.4.20 enum sd_application_command_t

Enumerator

kSD_ApplicationSetBusWdith Set Bus Width.
kSD_ApplicationStatus Send SD status.
kSD_ApplicationSendNumberWriteBlocks Send Number Of Written Blocks.
kSD_ApplicationSetWriteBlockEraseCount Set Write Block Erase Count.
kSD_ApplicationSendOperationCondition Send Operation Condition.
kSD_ApplicationSetClearCardDetect Set Connnect/Disconnect pull up on detect pin.
kSD_ApplicationSendScr Send Scr.

50.7.4.21 anonymous enum

Enumerator

kSDMMC_CommandClassBasic Card command class 0.
kSDMMC_CommandClassBlockRead Card command class 2.
kSDMMC_CommandClassBlockWrite Card command class 4.
kSDMMC_CommandClassErase Card command class 5.
kSDMMC_CommandClassWriteProtect Card command class 6.
kSDMMC_CommandClassLockCard Card command class 7.
kSDMMC_CommandClassApplicationSpecific Card command class 8.
kSDMMC_CommandClassInputOutputMode Card command class 9.
kSDMMC_CommandClassSwitch Card command class 10.

50.7.4.22 anonymous enum

Enumerator

kSD_OcrPowerUpBusyFlag Power up busy status.
kSD_OcrHostCapacitySupportFlag Card capacity status.
kSD_OcrCardCapacitySupportFlag Card capacity status.
kSD_OcrSwitch18RequestFlag Switch to 1.8V request.
kSD_OcrSwitch18AcceptFlag Switch to 1.8V accepted.
kSD_OcrVdd27_28Flag VDD 2.7-2.8.
kSD_OcrVdd28_29Flag VDD 2.8-2.9.
kSD_OcrVdd29_30Flag VDD 2.9-3.0.
kSD_OcrVdd30_31Flag VDD 2.9-3.0.
kSD_OcrVdd31_32Flag VDD 3.0-3.1.
kSD_OcrVdd32_33Flag VDD 3.1-3.2.
kSD_OcrVdd33_34Flag VDD 3.2-3.3.
kSD_OcrVdd34_35Flag VDD 3.3-3.4.
kSD_OcrVdd35_36Flag VDD 3.4-3.5.

50.7.4.23 anonymous enum

Enumerator

kSD_SpecificationVersion1_0 SD card version 1.0-1.01.
kSD_SpecificationVersion1_1 SD card version 1.10.
kSD_SpecificationVersion2_0 SD card version 2.00.
kSD_SpecificationVersion3_0 SD card version 3.0.

50.7.4.24 enum sd_switch_mode_t

Enumerator

kSD_SwitchCheck SD switch mode 0: check function.
kSD_SwitchSet SD switch mode 1: set function.

50.7.4.25 anonymous enum

Enumerator

kSD_CsdReadBlockPartialFlag Partial blocks for read allowed [79:79].
kSD_CsdWriteBlockMisalignFlag Write block misalignment [78:78].
kSD_CsdReadBlockMisalignFlag Read block misalignment [77:77].
kSD_CsdDsrImplementedFlag DSR implemented [76:76].
kSD_CsdEraseBlockEnabledFlag Erase single block enabled [46:46].
kSD_CsdWriteProtectGroupEnabledFlag Write protect group enabled [31:31].
kSD_CsdWriteBlockPartialFlag Partial blocks for write allowed [21:21].
kSD_CsdFileFormatGroupFlag File format group [15:15].
kSD_CsdCopyFlag Copy flag [14:14].
kSD_CsdPermanentWriteProtectFlag Permanent write protection [13:13].
kSD_CsdTemporaryWriteProtectFlag Temporary write protection [12:12].

50.7.4.26 anonymous enum

Enumerator

kSD_ScrDataStatusAfterErase Data status after erases [55:55].
kSD_ScrSdSpecification3 Specification version 3.00 or higher [47:47].

50.7.4.27 anonymous enum

Enumerator

kSD_FunctionSDR12Deafult SDR12 mode & default.
kSD_FunctionSDR25HighSpeed SDR25 & high speed.

kSD_FunctionSDR50 SDR50 mode.
kSD_FunctionSDR104 SDR104 mode.
kSD_FunctionDDR50 DDR50 mode.

50.7.4.28 anonymous enum

Enumerator

kSD_GroupTimingMode acess mode group
kSD_GroupCommandSystem command system group
kSD_GroupDriverStrength driver strength group
kSD_GroupCurrentLimit current limit group

50.7.4.29 enum sd_timing_mode_t

Enumerator

kSD_TimingSDR12DefaultMode Identification mode & SDR12.
kSD_TimingSDR25HighSpeedMode High speed mode & SDR25.
kSD_TimingSDR50Mode SDR50 mode.
kSD_TimingSDR104Mode SDR104 mode.
kSD_TimingDDR50Mode DDR50 mode.

50.7.4.30 enum sd_driver_strength_t

Enumerator

kSD_DriverStrengthTypeB default driver strength
kSD_DriverStrengthTypeA driver strength TYPE A
kSD_DriverStrengthTypeC driver strength TYPE C
kSD_DriverStrengthTypeD driver strength TYPE D

50.7.4.31 enum sd_max_current_t

Enumerator

kSD_CurrentLimit200MA default current limit
kSD_CurrentLimit400MA current limit to 400MA
kSD_CurrentLimit600MA current limit to 600MA
kSD_CurrentLimit800MA current limit to 800MA

50.7.4.32 enum sdmmc_command_t

Enumerator

kSDMMC_GoIdleState Go Idle State.
kSDMMC_AllSendCid All Send CID.
kSDMMC_SetDsr Set DSR.
kSDMMC_SelectCard Select Card.
kSDMMC_SendCsd Send CSD.
kSDMMC_SendCid Send CID.
kSDMMC_StopTransmission Stop Transmission.
kSDMMC_SendStatus Send Status.
kSDMMC_GoInactiveState Go Inactive State.
kSDMMC_SetBlockLength Set Block Length.
kSDMMC_ReadSingleBlock Read Single Block.
kSDMMC_ReadMultipleBlock Read Multiple Block.
kSDMMC_SetBlockCount Set Block Count.
kSDMMC_WriteSingleBlock Write Single Block.
kSDMMC_WriteMultipleBlock Write Multiple Block.
kSDMMC_ProgramCsd Program CSD.
kSDMMC_SetWriteProtect Set Write Protect.
kSDMMC_ClearWriteProtect Clear Write Protect.
kSDMMC_SendWriteProtect Send Write Protect.
kSDMMC_Erase Erase.
kSDMMC_LockUnlock Lock Unlock.
kSDMMC_ApplicationCommand Send Application Command.
kSDMMC_GeneralCommand General Purpose Command.
kSDMMC_ReadOcr Read OCR.

50.7.4.33 anonymous enum

Enumerator

kSDIO_RegCCCRSdioVer CCCR & SDIO version.
kSDIO_RegSDVersion SD version.
kSDIO_RegIOEnable io enable register
kSDIO_RegIOReady io ready register
kSDIO_RegIOIntEnable io interrupt enable register
kSDIO_RegIOIntPending io interrupt pending register
kSDIO_RegIOAbort io abort register
kSDIO_RegBusInterface bus interface register
kSDIO_RegCardCapability card capability register
kSDIO_RegCommonCISPointer common CIS pointer register
kSDIO_RegBusSuspend bus suspend register
kSDIO_RegFunctionSelect function select register
kSDIO_RegExecutionFlag execution flag register

kSDIO_RegReadyFlag ready flag register
kSDIO_RegFN0BlockSizeLow FN0 block size register.
kSDIO_RegFN0BlockSizeHigh FN0 block size register.
kSDIO_RegPowerControl power control register
kSDIO_RegBusSpeed bus speed register
kSDIO_RegUHSITimingSupport UHS-I timing support register.
kSDIO_RegDriverStrength Driver strength register.
kSDIO_RegInterruptExtension Interrupt extension register.

50.7.4.34 enum sdio_command_t

Enumerator

kSDIO_SendRelativeAddress send relative address
kSDIO_SendOperationCondition send operation condition
kSDIO_SendInterfaceCondition send interface condition
kSDIO_RWIODirect read/write IO direct command
kSDIO_RWIOExtended read/write IO extended command

50.7.4.35 enum sdio_func_num_t

Enumerator

kSDIO_FunctionNum0 sdio function0
kSDIO_FunctionNum1 sdio function1
kSDIO_FunctionNum2 sdio function2
kSDIO_FunctionNum3 sdio function3
kSDIO_FunctionNum4 sdio function4
kSDIO_FunctionNum5 sdio function5
kSDIO_FunctionNum6 sdio function6
kSDIO_FunctionNum7 sdio function7
kSDIO_FunctionMemory for combo card

50.7.4.36 anonymous enum

Enumerator

kSDIO_StatusCmdCRCError the CRC check of the previous cmd fail
kSDIO_StatusIllegalCmd cmd illegal for the card state
kSDIO_StatusR6Error special for R6 error status
kSDIO_StatusError A general or an unknown error occurred.
kSDIO_StatusFunctionNumError invail function error
kSDIO_StatusOutofRange cmd argument was out of the allowed range

50.7.4.37 anonymous enum

Enumerator

kSDIO_OcrPowerUpBusyFlag Power up busy status.
kSDIO_OcrIONumber number of IO function
kSDIO_OcrMemPresent memory present flag
kSDIO_OcrVdd20_21Flag VDD 2.0-2.1.
kSDIO_OcrVdd21_22Flag VDD 2.1-2.2.
kSDIO_OcrVdd22_23Flag VDD 2.2-2.3.
kSDIO_OcrVdd23_24Flag VDD 2.3-2.4.
kSDIO_OcrVdd24_25Flag VDD 2.4-2.5.
kSDIO_OcrVdd25_26Flag VDD 2.5-2.6.
kSDIO_OcrVdd26_27Flag VDD 2.6-2.7.
kSDIO_OcrVdd27_28Flag VDD 2.7-2.8.
kSDIO_OcrVdd28_29Flag VDD 2.8-2.9.
kSDIO_OcrVdd29_30Flag VDD 2.9-3.0.
kSDIO_OcrVdd30_31Flag VDD 2.9-3.0.
kSDIO_OcrVdd31_32Flag VDD 3.0-3.1.
kSDIO_OcrVdd32_33Flag VDD 3.1-3.2.
kSDIO_OcrVdd33_34Flag VDD 3.2-3.3.
kSDIO_OcrVdd34_35Flag VDD 3.3-3.4.
kSDIO_OcrVdd35_36Flag VDD 3.4-3.5.

50.7.4.38 anonymous enum

Enumerator

kSDIO_CCCRSupportDirectCmdDuringDataTrans support direct cmd during data transfer
kSDIO_CCCRSupportMultiBlock support multi block mode
kSDIO_CCCRSupportReadWait support read wait
kSDIO_CCCRSupportSuspendResume support suspend resume
kSDIO_CCCRSupportIntDuring4BitDataTrans support interrupt during 4-bit data transfer
kSDIO_CCCRSupportLowSpeed1Bit support low speed 1bit mode
kSDIO_CCCRSupportLowSpeed4Bit support low speed 4bit mode
kSDIO_CCCRSupportMasterPowerControl support master power control
kSDIO_CCCRSupportHighSpeed support high speed
kSDIO_CCCRSupportContinuousSPIInt support continuous SPI interrupt

50.7.4.39 anonymous enum

Enumerator

kSDIO_FBRSupportCSA function support CSA
kSDIO_FBRSupportPowerSelection function support power selection

50.7.4.40 enum sdio_bus_width_t

Enumerator

kSDIO_DataBus1Bit 1 bit bus mode
kSDIO_DataBus4Bit 4 bit bus mode
kSDIO_DataBus8Bit 8 bit bus mode

50.7.4.41 enum mmc_command_t

Enumerator

kMMC_SendOperationCondition Send Operation Condition.
kMMC_SetRelativeAddress Set Relative Address.
kMMC_SleepAwake Sleep Awake.
kMMC_Switch Switch.
kMMC_SendExtendedCsd Send EXT_CSD.
kMMC_ReadDataUntilStop Read Data Until Stop.
kMMC_BusTestRead Test Read.
kMMC_SendingBusTest test bus width cmd
kMMC_WriteDataUntilStop Write Data Until Stop.
kMMC_SendTuningBlock MMC sending tuning block.
kMMC_ProgramCid Program CID.
kMMC_EraseGroupStart Erase Group Start.
kMMC_EraseGroupEnd Erase Group End.
kMMC_FastInputOutput Fast IO.
kMMC_GoInterruptState Go interrupt State.

50.7.4.42 enum mmc_classified_voltage_t

Enumerator

kMMC_ClassifiedVoltageHigh High-voltage MMC card.
kMMC_ClassifiedVoltageDual Dual-voltage MMC card.

50.7.4.43 enum mmc_classified_density_t

Enumerator

kMMC_ClassifiedDensityWithin2GB Density byte is less than or equal 2GB.

50.7.4.44 enum mmc_access_mode_t

Enumerator

kMMC_AccessModeByte The card should be accessed as byte.
kMMC_AccessModeSector The card should be accessed as sector.

50.7.4.45 enum mmc_voltage_window_t

Enumerator

kMMC_VoltageWindowNone voltage window is not define by user
kMMC_VoltageWindow120 Voltage window is 1.20V.
kMMC_VoltageWindow170to195 Voltage window is 1.70V to 1.95V.
kMMC_VoltageWindows270to360 Voltage window is 2.70V to 3.60V.

50.7.4.46 enum mmc_csd_structure_version_t

Enumerator

kMMC_CsdStrucureVersion10 CSD version No. 1.0
kMMC_CsdStrucureVersion11 CSD version No. 1.1
kMMC_CsdStrucureVersion12 CSD version No. 1.2
kMMC_CsdStrucureVersionInExtcsd Version coded in Extended CSD.

50.7.4.47 enum mmc_specification_version_t

Enumerator

kMMC_SpecificationVersion0 Allocated by MMCA.
kMMC_SpecificationVersion1 Allocated by MMCA.
kMMC_SpecificationVersion2 Allocated by MMCA.
kMMC_SpecificationVersion3 Allocated by MMCA.
kMMC_SpecificationVersion4 Version 4.1/4.2/4.3/4.41-4.5-4.51-5.0.

50.7.4.48 anonymous enum

Enumerator

kMMC_ExtendedCsdRevision10 Revision 1.0.
kMMC_ExtendedCsdRevision11 Revision 1.1.
kMMC_ExtendedCsdRevision12 Revision 1.2.
kMMC_ExtendedCsdRevision13 Revision 1.3 MMC4.3.
kMMC_ExtendedCsdRevision14 Revision 1.4 obsolete.

kMMC_ExtendedCsdRevision15 Revision 1.5 MMC4.41.
kMMC_ExtendedCsdRevision16 Revision 1.6 MMC4.5.
kMMC_ExtendedCsdRevision17 Revision 1.7 MMC5.0.

50.7.4.49 enum mmc_command_set_t

Enumerator

kMMC_CommandSetStandard Standard MMC.
kMMC_CommandSet1 Command set 1.
kMMC_CommandSet2 Command set 2.
kMMC_CommandSet3 Command set 3.
kMMC_CommandSet4 Command set 4.

50.7.4.50 anonymous enum

Enumerator

kMMC_SupportAlternateBoot support alternative boot mode
kMMC_SupportDDRBoot support DDR boot mode
kMMC_SupportHighSpeedBoot support high speed boot mode

50.7.4.51 enum mmc_high_speed_timing_t

Enumerator

kMMC_HighSpeedTimingNone MMC card using none high-speed timing.
kMMC_HighSpeedTiming MMC card using high-speed timing.
kMMC_HighSpeed200Timing MMC card high speed 200 timing.
kMMC_HighSpeed400Timing MMC card high speed 400 timing.
kMMC_EnhanceHighSpeed400Timing MMC card high speed 400 timing.

50.7.4.52 enum mmc_data_bus_width_t

Enumerator

kMMC_DataBusWidth1bit MMC data bus width is 1 bit.
kMMC_DataBusWidth4bit MMC data bus width is 4 bits.
kMMC_DataBusWidth8bit MMC data bus width is 8 bits.
kMMC_DataBusWidth4bitDDR MMC data bus width is 4 bits ddr.
kMMC_DataBusWidth8bitDDR MMC data bus width is 8 bits ddr.
kMMC_DataBusWidth8bitDDRSTROBE MMC data bus width is 8 bits ddr strobe mode.

50.7.4.53 enum mmc_boot_partition_enable_t

Enumerator

kMMC_BootPartitionEnableNot Device not boot enabled (default)
kMMC_BootPartitionEnablePartition1 Boot partition 1 enabled for boot.
kMMC_BootPartitionEnablePartition2 Boot partition 2 enabled for boot.
kMMC_BootPartitionEnableUserAera User area enabled for boot.

50.7.4.54 enum mmc_boot_timing_mode_t

Enumerator

kMMC_BootModeSDRWithDefaultTiming boot mode single data rate with backward compatiable timings
kMMC_BootModeSDRWithHighSpeedTiming boot mode single data rate with high speed timing
kMMC_BootModeDDRTiming boot mode dual date rate

50.7.4.55 enum mmc_boot_partition_wp_t

Enumerator

kMMC_BootPartitionWPDisable boot partition write protection disable
kMMC_BootPartitionPwrWPToBothPartition power on period write protection apply to both boot partitions
kMMC_BootPartitionPermWPToBothPartition permanent write protection apply to both boot partitions
kMMC_BootPartitionPwrWPToPartition1 power on period write protection apply to partition1
kMMC_BootPartitionPwrWPToPartition2 power on period write protection apply to partition2
kMMC_BootPartitionPermWPToPartition1 permanent write protection apply to partition1
kMMC_BootPartitionPermWPToPartition2 permanent write protection apply to partition2
kMMC_BootPartitionPermWPToPartition1PwrWPToPartition2 permanent write protection apply to partition1, power on period write protection apply to partition2
kMMC_BootPartitionPermWPToPartition2PwrWPToPartition1 permanent write protection apply to partition2, power on period write protection apply to partition1

50.7.4.56 anonymous enum

Enumerator

kMMC_BootPartitionNotProtected boot partition not protected
kMMC_BootPartitionPwrProtected boot partition is power on period write protected
kMMC_BootPartitionPermProtected boot partition is permanently protected

50.7.4.57 enum mmc_access_partition_t

Enumerator

kMMC_AccessPartitionUserAera No access to boot partition (default), normal partition.
kMMC_AccessPartitionBoot1 Read/Write boot partition 1.
kMMC_AccessPartitionBoot2 Read/Write boot partition 2.
kMMC_AccessRPMB Replay protected mem block.
kMMC_AccessGeneralPurposePartition1 access to general purpose partition 1
kMMC_AccessGeneralPurposePartition2 access to general purpose partition 2
kMMC_AccessGeneralPurposePartition3 access to general purpose partition 3
kMMC_AccessGeneralPurposePartition4 access to general purpose partition 4

50.7.4.58 anonymous enum

Enumerator

kMMC_CsdReadBlockPartialFlag Partial blocks for read allowed.
kMMC_CsdWriteBlockMisalignFlag Write block misalignment.
kMMC_CsdReadBlockMisalignFlag Read block misalignment.
kMMC_CsdDsrImplementedFlag DSR implemented.
kMMC_CsdWriteProtectGroupEnabledFlag Write protect group enabled.
kMMC_CsdWriteBlockPartialFlag Partial blocks for write allowed.
kMMC_ContentProtectApplicationFlag Content protect application.
kMMC_CsdFileFormatGroupFlag File format group.
kMMC_CsdCopyFlag Copy flag.
kMMC_CsdPermanentWriteProtectFlag Permanent write protection.
kMMC_CsdTemporaryWriteProtectFlag Temporary write protection.

50.7.4.59 enum mmc_extended_csd_access_mode_t

Enumerator

kMMC_ExtendedCsdAccessModeCommandSet Command set related setting.
kMMC_ExtendedCsdAccessModeSetBits Set bits in specific byte in Extended CSD.
kMMC_ExtendedCsdAccessModeClearBits Clear bits in specific byte in Extended CSD.
kMMC_ExtendedCsdAccessModeWriteBits Write a value to specific byte in Extended CSD.

50.7.4.60 enum mmc_extended_csd_index_t

Enumerator

kMMC_ExtendedCsdIndexFlushCache flush cache
kMMC_ExtendedCsdIndexCacheControl cache control

kMMC_ExtendedCsdIndexBootPartitionWP Boot partition write protect.
kMMC_ExtendedCsdIndexEraseGroupDefinition Erase Group Def.
kMMC_ExtendedCsdIndexBootBusConditions Boot Bus conditions.
kMMC_ExtendedCsdIndexBootConfigWP Boot config write protect.
kMMC_ExtendedCsdIndexPartitionConfig Partition Config, before BOOT_CONFIG.
kMMC_ExtendedCsdIndexBusWidth Bus Width.
kMMC_ExtendedCsdIndexHighSpeedTiming High-speed Timing.
kMMC_ExtendedCsdIndexPowerClass Power Class.
kMMC_ExtendedCsdIndexCommandSet Command Set.

50.7.4.61 anonymous enum

Enumerator

kMMC_DriverStrength0 Driver type0 ,nominal impedance 50ohm.
kMMC_DriverStrength1 Driver type1 ,nominal impedance 33ohm.
kMMC_DriverStrength2 Driver type2 ,nominal impedance 66ohm.
kMMC_DriverStrength3 Driver type3 ,nominal impedance 100ohm.
kMMC_DriverStrength4 Driver type4 ,nominal impedance 40ohm.

50.7.4.62 enum mmc_extended_csd_flags_t

Enumerator

kMMC_ExtCsdExtPartitionSupport partitioning support[160]
kMMC_ExtCsdEnhancePartitionSupport partitioning support[160]
kMMC_ExtCsdPartitioningSupport partitioning support[160]
kMMC_ExtCsdPrgCIDCSDInDDRModeSupport CMD26 and CMD27 are support dual data rate [130].
kMMC_ExtCsdBKOpsSupport background operation feature support [502]
kMMC_ExtCsdDataTagSupport data tag support[499]
kMMC_ExtCsdModeOperationCodeSupport mode operation code support[493]

50.7.4.63 enum mmc_boot_mode_t

Enumerator

kMMC_BootModeNormal Normal boot.
kMMC_BootModeAlternative Alternative boot.

50.7.5 Function Documentation

50.7.5.1 status_t SDMMC_SelectCard (sdmmchost_t * *host,* uint32_t *relativeAddress,* bool *isSelected*)

Parameters

host	host handler.
relativeAddress	Relative address.
isSelected	True to put card into transfer state.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

50.7.5.2 status_t SDMMC_SendApplicationCommand (sdmmchost_t * *host,* uint32_t *relativeAddress*)

Parameters

host	host handler.
relativeAddress	Card relative address.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_SDMMC_Card-NotSupport	Card doesn't support.
kStatus_Success	Operate successfully.

50.7.5.3 status_t SDMMC_SetBlockCount (sdmmchost_t * *host,* uint32_t *blockCount*)

Parameters

host	host handler.
blockCount	Block count.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

50.7.5.4 status_t SDMMC_GoIdle (sdmmchost_t * *host*)

Parameters

host	host handler.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

50.7.5.5 status_t SDMMC_SetBlockSize (sdmmchost_t * *host,* uint32_t *blockSize*)

Parameters

host	host handler.
blockSize	Block size.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

50.7.5.6 status_t SDMMC_SetCardInactive (sdmmchost_t * *host*)

Parameters

host	host handler.

Return values

kStatus_SDMMC_-TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

Chapter 51
CODEC Driver

51.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

Modules

- CODEC Common Driver
- CODEC I2C Driver
- CS42888 Driver
- TFA9896 Driver
- WM8904 Driver

51.2 CODEC Common Driver

51.2.1 Overview

The codec common driver provides a codec control abstraction interface.

Modules

- CODEC Adapter
- CS42888 Adapter
- TFA9896 Adapter
- WM8904 Adapter

Data Structures

- struct codec_config_t
  Initialize structure of the codec. More...
- struct codec_capability_t
  codec capability More...
- struct codec_handle_t
  Codec handle definition. More...

Macros

- #define CODEC_VOLUME_MAX_VALUE (100U)
  codec maximum volume range

Enumerations

- enum {
  kStatus_CODEC_NotSupport = MAKE_STATUS(kStatusGroup_CODEC, 0U),
  kStatus_CODEC_DeviceNotRegistered = MAKE_STATUS(kStatusGroup_CODEC, 1U),
  kStatus_CODEC_I2CBusInitialFailed,
  kStatus_CODEC_I2CCommandTransferFailed }
  CODEC status.
- enum codec_audio_protocol_t {
  kCODEC_BusI2S = 0U,
  kCODEC_BusLeftJustified = 1U,
  kCODEC_BusRightJustified = 2U,
  kCODEC_BusPCMA = 3U,
  kCODEC_BusPCMB = 4U,
  kCODEC_BusTDM = 5U }
  AUDIO format definition.

- enum {
  kCODEC_AudioSampleRate8KHz = 8000U,
  kCODEC_AudioSampleRate11025Hz = 11025U,
  kCODEC_AudioSampleRate12KHz = 12000U,
  kCODEC_AudioSampleRate16KHz = 16000U,
  kCODEC_AudioSampleRate22050Hz = 22050U,
  kCODEC_AudioSampleRate24KHz = 24000U,
  kCODEC_AudioSampleRate32KHz = 32000U,
  kCODEC_AudioSampleRate44100Hz = 44100U,
  kCODEC_AudioSampleRate48KHz = 48000U,
  kCODEC_AudioSampleRate96KHz = 96000U,
  kCODEC_AudioSampleRate192KHz = 192000U,
  kCODEC_AudioSampleRate384KHz = 384000U }
  audio sample rate definition
- enum {
  kCODEC_AudioBitWidth16bit = 16U,
  kCODEC_AudioBitWidth20bit = 20U,
  kCODEC_AudioBitWidth24bit = 24U,
  kCODEC_AudioBitWidth32bit = 32U }
  audio bit width
- enum codec_module_t {
  kCODEC_ModuleADC = 0U,
  kCODEC_ModuleDAC = 1U,
  kCODEC_ModulePGA = 2U,
  kCODEC_ModuleHeadphone = 3U,
  kCODEC_ModuleSpeaker = 4U,
  kCODEC_ModuleLinein = 5U,
  kCODEC_ModuleLineout = 6U,
  kCODEC_ModuleVref = 7U,
  kCODEC_ModuleMicbias = 8U,
  kCODEC_ModuleMic = 9U,
  kCODEC_ModuleI2SIn = 10U,
  kCODEC_ModuleI2SOut = 11U,
  kCODEC_ModuleMixer = 12U }
  audio codec module
- enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }
  audio codec module control cmd
- enum {
  kCODEC_ModuleI2SInInterfacePCM = 0U,
  kCODEC_ModuleI2SInInterfaceDSD = 1U }
  audio codec module digital interface
- enum {
  kCODEC_RecordSourceDifferentialLine = 1U,
  kCODEC_RecordSourceLineInput = 2U,
  kCODEC_RecordSourceDifferentialMic = 4U,
  kCODEC_RecordSourceDigitalMic = 8U,

kCODEC_RecordSourceSingleEndMic = 16U }

audio codec module record source value

- enum {
  kCODEC_RecordChannelLeft1 = 1U,
  kCODEC_RecordChannelLeft2 = 2U,
  kCODEC_RecordChannelLeft3 = 4U,
  kCODEC_RecordChannelRight1 = 1U,
  kCODEC_RecordChannelRight2 = 2U,
  kCODEC_RecordChannelRight3 = 4U,
  kCODEC_RecordChannelDifferentialPositive1 = 1U,
  kCODEC_RecordChannelDifferentialPositive2 = 2U,
  kCODEC_RecordChannelDifferentialPositive3 = 4U,
  kCODEC_RecordChannelDifferentialNegative1 = 8U,
  kCODEC_RecordChannelDifferentialNegative2 = 16U,
  kCODEC_RecordChannelDifferentialNegative3 = 32U }

  audio codec record channel

- enum {
  kCODEC_PlaySourcePGA = 1U,
  kCODEC_PlaySourceInput = 2U,
  kCODEC_PlaySourceDAC = 4U,
  kCODEC_PlaySourceMixerIn = 1U,
  kCODEC_PlaySourceMixerInLeft = 2U,
  kCODEC_PlaySourceMixerInRight = 4U,
  kCODEC_PlaySourceAux = 8U }

  audio codec module play source value

- enum {
  kCODEC_PlayChannelHeadphoneLeft = 1U,
  kCODEC_PlayChannelHeadphoneRight = 2U,
  kCODEC_PlayChannelSpeakerLeft = 4U,
  kCODEC_PlayChannelSpeakerRight = 8U,
  kCODEC_PlayChannelLineOutLeft = 16U,
  kCODEC_PlayChannelLineOutRight = 32U,
  kCODEC_PlayChannelLeft0 = 1U,
  kCODEC_PlayChannelRight0 = 2U,
  kCODEC_PlayChannelLeft1 = 4U,
  kCODEC_PlayChannelRight1 = 8U,
  kCODEC_PlayChannelLeft2 = 16U,
  kCODEC_PlayChannelRight2 = 32U,
  kCODEC_PlayChannelLeft3 = 64U,
  kCODEC_PlayChannelRight3 = 128U }

  codec play channel

- enum {

kCODEC_VolumeHeadphoneLeft = 1U,
kCODEC_VolumeHeadphoneRight = 2U,
kCODEC_VolumeSpeakerLeft = 4U,
kCODEC_VolumeSpeakerRight = 8U,
kCODEC_VolumeLineOutLeft = 16U,
kCODEC_VolumeLineOutRight = 32U,
kCODEC_VolumeLeft0 = 1UL << 0U,
kCODEC_VolumeRight0 = 1UL << 1U,
kCODEC_VolumeLeft1 = 1UL << 2U,
kCODEC_VolumeRight1 = 1UL << 3U,
kCODEC_VolumeLeft2 = 1UL << 4U,
kCODEC_VolumeRight2 = 1UL << 5U,
kCODEC_VolumeLeft3 = 1UL << 6U,
kCODEC_VolumeRight3 = 1UL << 7U,
kCODEC_VolumeDAC = 1UL << 8U }

codec volume setting

- enum {

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

audio codec capability

Functions

- status_t CODEC_Init (codec_handle_t *handle, codec_config_t *config)
  Codec initilization.
- status_t CODEC_Deinit (codec_handle_t *handle)
  Codec de-initilization.
- status_t CODEC_SetFormat (codec_handle_t *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t CODEC_ModuleControl (codec_handle_t *handle, codec_module_ctrl_cmd_t cmd, uint32_t data)
  codec module control.
- status_t CODEC_SetVolume (codec_handle_t *handle, uint32_t channel, uint32_t volume)
  set audio codec pl volume.
- status_t CODEC_SetMute (codec_handle_t *handle, uint32_t channel, bool mute)
  set audio codec module mute.
- status_t CODEC_SetPower (codec_handle_t *handle, codec_module_t module, bool powerOn)
  set audio codec power.
- status_t CODEC_SetRecord (codec_handle_t *handle, uint32_t recordSource)
  codec set record source.
- status_t CODEC_SetRecordChannel (codec_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t CODEC_SetPlay (codec_handle_t *handle, uint32_t playSource)
  codec set play source.

Driver version

- #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  CLOCK driver version 2.3.0.

51.2.2 Data Structure Documentation

51.2.2.1 struct codec_config_t

Data Fields

- uint32_t codecDevType
  codec type
- void * codecDevConfig
  Codec device specific configuration.

51.2.2.2 struct codec_capability_t

Data Fields

- uint32_t codecModuleCapability
  codec module capability
- uint32_t codecPlayCapability
  codec play capability
- uint32_t codecRecordCapability
  codec record capability
- uint32_t codecVolumeCapability
  codec volume capability

51.2.2.3 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

51.2.3 Macro Definition Documentation

51.2.3.1 #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

51.2.4 Enumeration Type Documentation

51.2.4.1 anonymous enum

Enumerator

kStatus_CODEC_NotSupport CODEC not support status.
kStatus_CODEC_DeviceNotRegistered CODEC device register failed status.
kStatus_CODEC_I2CBusInitialFailed CODEC i2c bus initialization failed status.
kStatus_CODEC_I2CCommandTransferFailed CODEC i2c bus command transfer failed status.

51.2.4.2 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.

51.2.4.3 anonymous enum

Enumerator

kCODEC_AudioSampleRate8KHz Sample rate 8000 Hz.
kCODEC_AudioSampleRate11025Hz Sample rate 11025 Hz.
kCODEC_AudioSampleRate12KHz Sample rate 12000 Hz.
kCODEC_AudioSampleRate16KHz Sample rate 16000 Hz.
kCODEC_AudioSampleRate22050Hz Sample rate 22050 Hz.
kCODEC_AudioSampleRate24KHz Sample rate 24000 Hz.
kCODEC_AudioSampleRate32KHz Sample rate 32000 Hz.
kCODEC_AudioSampleRate44100Hz Sample rate 44100 Hz.
kCODEC_AudioSampleRate48KHz Sample rate 48000 Hz.
kCODEC_AudioSampleRate96KHz Sample rate 96000 Hz.
kCODEC_AudioSampleRate192KHz Sample rate 192000 Hz.
kCODEC_AudioSampleRate384KHz Sample rate 384000 Hz.

51.2.4.4 anonymous enum

Enumerator

kCODEC_AudioBitWidth16bit audio bit width 16
kCODEC_AudioBitWidth20bit audio bit width 20
kCODEC_AudioBitWidth24bit audio bit width 24
kCODEC_AudioBitWidth32bit audio bit width 32

51.2.4.5 enum codec_module_t

Enumerator

kCODEC_ModuleADC codec module ADC
kCODEC_ModuleDAC codec module DAC
kCODEC_ModulePGA codec module PGA
kCODEC_ModuleHeadphone codec module headphone

kCODEC_ModuleSpeaker codec module speaker
kCODEC_ModuleLinein codec module linein
kCODEC_ModuleLineout codec module lineout
kCODEC_ModuleVref codec module VREF
kCODEC_ModuleMicbias codec module MIC BIAS
kCODEC_ModuleMic codec module MIC
kCODEC_ModuleI2SIn codec module I2S in
kCODEC_ModuleI2SOut codec module I2S out
kCODEC_ModuleMixer codec module mixer

51.2.4.6 enum codec_module_ctrl_cmd_t

Enumerator

kCODEC_ModuleSwitchI2SInInterface module digital interface siwtch.

51.2.4.7 anonymous enum

Enumerator

kCODEC_ModuleI2SInInterfacePCM Pcm interface.
kCODEC_ModuleI2SInInterfaceDSD DSD interface.

51.2.4.8 anonymous enum

Enumerator

kCODEC_RecordSourceDifferentialLine record source from differential line
kCODEC_RecordSourceLineInput record source from line input
kCODEC_RecordSourceDifferentialMic record source from differential mic
kCODEC_RecordSourceDigitalMic record source from digital microphone
kCODEC_RecordSourceSingleEndMic record source from single microphone

51.2.4.9 anonymous enum

Enumerator

kCODEC_RecordChannelLeft1 left record channel 1
kCODEC_RecordChannelLeft2 left record channel 2
kCODEC_RecordChannelLeft3 left record channel 3
kCODEC_RecordChannelRight1 right record channel 1
kCODEC_RecordChannelRight2 right record channel 2
kCODEC_RecordChannelRight3 right record channel 3
kCODEC_RecordChannelDifferentialPositive1 differential positive record channel 1

kCODEC_RecordChannelDifferentialPositive2 differential positive record channel 2
kCODEC_RecordChannelDifferentialPositive3 differential positive record channel 3
kCODEC_RecordChannelDifferentialNegative1 differential negative record channel 1
kCODEC_RecordChannelDifferentialNegative2 differential negative record channel 2
kCODEC_RecordChannelDifferentialNegative3 differential negative record channel 3

51.2.4.10 anonymous enum

Enumerator

kCODEC_PlaySourcePGA play source PGA, bypass ADC
kCODEC_PlaySourceInput play source Input3
kCODEC_PlaySourceDAC play source DAC
kCODEC_PlaySourceMixerIn play source mixer in
kCODEC_PlaySourceMixerInLeft play source mixer in left
kCODEC_PlaySourceMixerInRight play source mixer in right
kCODEC_PlaySourceAux play source mixer in AUx

51.2.4.11 anonymous enum

Enumerator

kCODEC_PlayChannelHeadphoneLeft play channel headphone left
kCODEC_PlayChannelHeadphoneRight play channel headphone right
kCODEC_PlayChannelSpeakerLeft play channel speaker left
kCODEC_PlayChannelSpeakerRight play channel speaker right
kCODEC_PlayChannelLineOutLeft play channel lineout left
kCODEC_PlayChannelLineOutRight play channel lineout right
kCODEC_PlayChannelLeft0 play channel left0
kCODEC_PlayChannelRight0 play channel right0
kCODEC_PlayChannelLeft1 play channel left1
kCODEC_PlayChannelRight1 play channel right1
kCODEC_PlayChannelLeft2 play channel left2
kCODEC_PlayChannelRight2 play channel right2
kCODEC_PlayChannelLeft3 play channel left3
kCODEC_PlayChannelRight3 play channel right3

51.2.4.12 anonymous enum

Enumerator

kCODEC_VolumeHeadphoneLeft headphone left volume
kCODEC_VolumeHeadphoneRight headphone right volume
kCODEC_VolumeSpeakerLeft speaker left volume
kCODEC_VolumeSpeakerRight speaker right volume

kCODEC_VolumeLineOutLeft lineout left volume
kCODEC_VolumeLineOutRight lineout right volume
kCODEC_VolumeLeft0 left0 volume
kCODEC_VolumeRight0 right0 volume
kCODEC_VolumeLeft1 left1 volume
kCODEC_VolumeRight1 right1 volume
kCODEC_VolumeLeft2 left2 volume
kCODEC_VolumeRight2 right2 volume
kCODEC_VolumeLeft3 left3 volume
kCODEC_VolumeRight3 right3 volume
kCODEC_VolumeDAC dac volume

51.2.4.13 anonymous enum

Enumerator

kCODEC_SupportModuleADC codec capability of module ADC
kCODEC_SupportModuleDAC codec capability of module DAC
kCODEC_SupportModulePGA codec capability of module PGA
kCODEC_SupportModuleHeadphone codec capability of module headphone
kCODEC_SupportModuleSpeaker codec capability of module speaker
kCODEC_SupportModuleLinein codec capability of module linein
kCODEC_SupportModuleLineout codec capability of module lineout
kCODEC_SupportModuleVref codec capability of module vref
kCODEC_SupportModuleMicbias codec capability of module mic bias
kCODEC_SupportModuleMic codec capability of module mic bias
kCODEC_SupportModuleI2SIn codec capability of module I2S in
kCODEC_SupportModuleI2SOut codec capability of module I2S out
kCODEC_SupportModuleMixer codec capability of module mixer
kCODEC_SupportModuleI2SInSwitchInterface codec capability of module I2S in switch interface

kCODEC_SupportPlayChannelLeft0 codec capability of play channel left 0
kCODEC_SupportPlayChannelRight0 codec capability of play channel right 0
kCODEC_SupportPlayChannelLeft1 codec capability of play channel left 1
kCODEC_SupportPlayChannelRight1 codec capability of play channel right 1
kCODEC_SupportPlayChannelLeft2 codec capability of play channel left 2
kCODEC_SupportPlayChannelRight2 codec capability of play channel right 2
kCODEC_SupportPlayChannelLeft3 codec capability of play channel left 3
kCODEC_SupportPlayChannelRight3 codec capability of play channel right 3
kCODEC_SupportPlaySourcePGA codec capability of set playback source PGA
kCODEC_SupportPlaySourceInput codec capability of set playback source INPUT
kCODEC_SupportPlaySourceDAC codec capability of set playback source DAC
kCODEC_SupportPlaySourceMixerIn codec capability of set play source Mixer in
kCODEC_SupportPlaySourceMixerInLeft codec capability of set play source Mixer in left
kCODEC_SupportPlaySourceMixerInRight codec capability of set play source Mixer in right

kCODEC_SupportPlaySourceAux codec capability of set play source aux
kCODEC_SupportRecordSourceDifferentialLine codec capability of record source differential line

kCODEC_SupportRecordSourceLineInput codec capability of record source line input
kCODEC_SupportRecordSourceDifferentialMic codec capability of record source differential mic

kCODEC_SupportRecordSourceDigitalMic codec capability of record digital mic
kCODEC_SupportRecordSourceSingleEndMic codec capability of single end mic
kCODEC_SupportRecordChannelLeft1 left record channel 1
kCODEC_SupportRecordChannelLeft2 left record channel 2
kCODEC_SupportRecordChannelLeft3 left record channel 3
kCODEC_SupportRecordChannelRight1 right record channel 1
kCODEC_SupportRecordChannelRight2 right record channel 2
kCODEC_SupportRecordChannelRight3 right record channel 3

51.2.5 Function Documentation

51.2.5.1 status_t CODEC_Init (codec_handle_t ∗ *handle,* codec_config_t ∗ *config*)

Parameters

handle	codec handle.
config	codec configurations.

Returns

kStatus_Success is success, else de-initial failed.

51.2.5.2 status_t CODEC_Deinit (codec_handle_t ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

51.2.5.3 status_t CODEC_SetFormat (codec_handle_t ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

51.2.5.4 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

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	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.

51.2.5.5 status_t CODEC_SetVolume (codec_handle_t * *handle,* uint32_t *channel,* uint32_t *volume*)

Parameters

handle	codec handle.
channel	audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

51.2.5.6 status_t CODEC_SetMute (codec_handle_t ∗ *handle,* uint32_t *channel,* bool *mute*)

Parameters

handle	codec handle.
channel	audio codec volume channel, can be a value or combine value of _codec_volume_-capability or _codec_play_channel.
mute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

51.2.5.7 status_t CODEC_SetPower (codec_handle_t ∗ *handle,* codec_module_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

51.2.5.8 status_t CODEC_SetRecord (codec_handle_t ∗ *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

51.2.5.9 status_t CODEC_SetRecordChannel (codec_handle_t * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecordChannel	audio codec record channel, reference _codec_record_channel, can be a value combine of member in _codec_record_channel.
rightRecordChannel	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.

51.2.5.10 status_t CODEC_SetPlay (codec_handle_t * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

51.3 CODEC I2C Driver

51.3.1 Overview

The codec common driver provides a codec control abstraction interface.

Data Structures

- struct codec_i2c_config_t
  CODEC I2C configurations structure. More...

Macros

- #define CODEC_I2C_MASTER_HANDLER_SIZE HAL_I2C_MASTER_HANDLE_SIZE
  codec i2c handler

Enumerations

- enum codec_reg_addr_t {
  kCODEC_RegAddr8Bit = 1U,
  kCODEC_RegAddr16Bit = 2U }
  CODEC device register address type.
- enum codec_reg_width_t {
  kCODEC_RegWidth8Bit = 1U,
  kCODEC_RegWidth16Bit = 2U,
  kCODEC_RegWidth32Bit = 4U }
  CODEC device register width.

Functions

- status_t CODEC_I2C_Init (void ∗handle, uint32_t i2cInstance, uint32_t i2cBaudrate, uint32_t i2c-SourceClockHz)
  Codec i2c bus initilization.
- status_t CODEC_I2C_Deinit (void ∗handle)
  Codec i2c de-initilization.
- status_t CODEC_I2C_Send (void ∗handle, uint8_t deviceAddress, uint32_t subAddress, uint8_t subaddressSize, uint8_t ∗txBuff, uint8_t txBuffSize)
  codec i2c send function.
- status_t CODEC_I2C_Receive (void ∗handle, uint8_t deviceAddress, uint32_t subAddress, uint8_t subaddressSize, uint8_t ∗rxBuff, uint8_t rxBuffSize)
  codec i2c receive function.

51.3.2 Data Structure Documentation

51.3.2.1 struct codec_i2c_config_t

Data Fields

- uint32_t codecI2CInstance
  i2c bus instance
- uint32_t codecI2CSourceClock
  i2c bus source clock frequency

51.3.3 Enumeration Type Documentation

51.3.3.1 enum codec_reg_addr_t

Enumerator

kCODEC_RegAddr8Bit 8-bit register address.
kCODEC_RegAddr16Bit 16-bit register address.

51.3.3.2 enum codec_reg_width_t

Enumerator

kCODEC_RegWidth8Bit 8-bit register width.
kCODEC_RegWidth16Bit 16-bit register width.
kCODEC_RegWidth32Bit 32-bit register width.

51.3.4 Function Documentation

51.3.4.1 status_t CODEC_I2C_Init (void ∗ *handle*, uint32_t *i2cInstance*, uint32_t *i2cBaudrate*, uint32_t *i2cSourceClockHz*)

Parameters

handle	i2c master handle.
i2cInstance	instance number of the i2c bus, such as 0 is corresponding to I2C0.

i2cBaudrate	i2c baudrate.
i2cSourceClockHz	i2c source clock frequency.

Returns

kStatus_HAL_I2cSuccess is success, else initial failed.

51.3.4.2 status_t CODEC_I2C_Deinit (void * *handle*)

Parameters

handle	i2c master handle.

Returns

kStatus_HAL_I2cSuccess is success, else deinitial failed.

51.3.4.3 status_t CODEC_I2C_Send (void * *handle,* uint8_t *deviceAddress,* uint32_t *subAddress,* uint8_t *subaddressSize,* uint8_t * *txBuff,* uint8_t *txBuffSize*)

Parameters

handle	i2c master handle.
deviceAddress	codec device address.
subAddress	register address.
subaddressSize	register address width.
txBuff	tx buffer pointer.
txBuffSize	tx buffer size.

Returns

kStatus_HAL_I2cSuccess is success, else send failed.

51.3.4.4 status_t CODEC_I2C_Receive (void * *handle,* uint8_t *deviceAddress,* uint32_t *subAddress,* uint8_t *subaddressSize,* uint8_t * *rxBuff,* uint8_t *rxBuffSize*)

Parameters

handle	i2c master handle.
deviceAddress	codec device address.
subAddress	register address.
subaddressSize	register address width.
rxBuff	rx buffer pointer.
rxBuffSize	rx buffer size.

Returns

kStatus_HAL_I2cSuccess is success, else receive failed.

51.4 CS42888 Driver

51.4.1 Overview

The cs42888 driver provides a codec control interface.

Data Structures

- struct cs42888_audio_format_t
  cs42888 audio format More...
- struct cs42888_config_t
  Initialize structure of CS42888. More...
- struct cs42888_handle_t
  cs42888 handler More...

Macros

- #define CS42888_I2C_HANDLER_SIZE CODEC_I2C_MASTER_HANDLER_SIZE
  CS42888 handle size.
- #define CS42888_ID 0x01U
  Define the register address of CS42888.
- #define CS42888_AOUT_MAX_VOLUME_VALUE 0xFFU
  CS42888 volume setting range.
- #define CS42888_CACHEREGNUM 28U
  Cache register number.
- #define CS42888_I2C_ADDR 0x48U
  CS42888 I2C address.
- #define CS42888_I2C_BITRATE (100000U)
  CS42888 I2C baudrate.

Typedefs

- typedef void(∗ cs42888_reset)(bool state)
  cs42888 reset function pointer

Enumerations

- enum cs42888_func_mode {
  kCS42888_ModeMasterSSM = 0x0,
  kCS42888_ModeMasterDSM = 0x1,
  kCS42888_ModeMasterQSM = 0x2,
  kCS42888_ModeSlave = 0x3 }
  CS42888 support modes.

- enum cs42888_module_t {
  kCS42888_ModuleDACPair1 = 0x2,
  kCS42888_ModuleDACPair2 = 0x4,
  kCS42888_ModuleDACPair3 = 0x8,
  kCS42888_ModuleDACPair4 = 0x10,
  kCS42888_ModuleADCPair1 = 0x20,
  kCS42888_ModuleADCPair2 = 0x40 }
  Modules in CS42888 board.
- enum cs42888_bus_t {
  kCS42888_BusLeftJustified = 0x0,
  kCS42888_BusI2S = 0x1,
  kCS42888_BusRightJustified = 0x2,
  kCS42888_BusOL1 = 0x4,
  kCS42888_BusOL2 = 0x5,
  kCS42888_BusTDM = 0x6 }
  CS42888 supported audio bus type.
- enum {
  kCS42888_AOUT1 = 1U,
  kCS42888_AOUT2 = 2U,
  kCS42888_AOUT3 = 3U,
  kCS42888_AOUT4 = 4U,
  kCS42888_AOUT5 = 5U,
  kCS42888_AOUT6 = 6U,
  kCS42888_AOUT7 = 7U,
  kCS42888_AOUT8 = 8U }
  CS428888 play channel.

Functions

- status_t CS42888_Init (cs42888_handle_t *handle, cs42888_config_t *config)
  CS42888 initialize function.
- status_t CS42888_Deinit (cs42888_handle_t *handle)
  Deinit the CS42888 codec.
- status_t CS42888_SetProtocol (cs42888_handle_t *handle, cs42888_bus_t protocol, uint32_t bitWidth)
  Set the audio transfer protocol.
- void CS42888_SetFuncMode (cs42888_handle_t *handle, cs42888_func_mode mode)
  Set CS42888 to differernt working mode.
- status_t CS42888_SelectFunctionalMode (cs42888_handle_t *handle, cs42888_func_mode adcMode, cs42888_func_mode dacMode)
  Set CS42888 to differernt functional mode.
- status_t CS42888_SetAOUTVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)
  Set the volume of different modules in CS42888.
- status_t CS42888_SetAINVolume (cs42888_handle_t *handle, uint8_t channel, uint8_t volume)
  Set the volume of different modules in CS42888.
- uint8_t CS42888_GetAOUTVolume (cs42888_handle_t *handle, uint8_t channel)

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, 3))
  cs42888 driver version 2.1.3.

51.4.2 Data Structure Documentation

51.4.2.1 struct cs42888_audio_format_t

Data Fields

- uint32_t mclk_HZ
  master clock frequency
- uint32_t sampleRate
  sample rate
- uint32_t bitWidth
  bit width

51.4.2.2 struct cs42888_config_t

Data Fields

- cs42888_bus_t bus
  Audio transfer protocol.
- cs42888_audio_format_t format
  cs42888 audio format

- cs42888_func_mode ADCMode
  CS42888 ADC function mode.
- cs42888_func_mode DACMode
  CS42888 DAC function mode.
- bool master
  true is master, false is slave
- codec_i2c_config_t i2cConfig
  i2c bus configuration
- uint8_t slaveAddress
  slave address
- cs42888_reset reset
  reset function pointer

Field Documentation

(1) cs42888_func_mode cs42888_config_t::ADCMode

(2) cs42888_func_mode cs42888_config_t::DACMode

51.4.2.3 struct cs42888_handle_t

Data Fields

- cs42888_config_t * config
  cs42888 config pointer
- uint8_t i2cHandle [CS42888_I2C_HANDLER_SIZE]
  i2c handle pointer

51.4.3 Macro Definition Documentation

51.4.3.1 #define FSL_CS42888_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

51.4.3.2 #define CS42888_ID 0x01U

51.4.3.3 #define CS42888_I2C_ADDR 0x48U

51.4.4 Enumeration Type Documentation

51.4.4.1 enum cs42888_func_mode

Enumerator

kCS42888_ModeMasterSSM master single speed mode
kCS42888_ModeMasterDSM master dual speed mode
kCS42888_ModeMasterQSM master quad speed mode
kCS42888_ModeSlave master single speed mode

51.4.4.2 enum cs42888_module_t

Enumerator

kCS42888_ModuleDACPair1 DAC pair1 (AOUT1 and AOUT2) module in CS42888.
kCS42888_ModuleDACPair2 DAC pair2 (AOUT3 and AOUT4) module in CS42888.
kCS42888_ModuleDACPair3 DAC pair3 (AOUT5 and AOUT6) module in CS42888.
kCS42888_ModuleDACPair4 DAC pair4 (AOUT7 and AOUT8) module in CS42888.
kCS42888_ModuleADCPair1 ADC pair1 (AIN1 and AIN2) module in CS42888.
kCS42888_ModuleADCPair2 ADC pair2 (AIN3 and AIN4) module in CS42888.

51.4.4.3 enum cs42888_bus_t

Enumerator

kCS42888_BusLeftJustified Left justified format, up to 24 bits.
kCS42888_BusI2S I2S format, up to 24 bits.
kCS42888_BusRightJustified Right justified, can support 16bits and 24 bits.
kCS42888_BusOL1 One-Line #1 mode.
kCS42888_BusOL2 One-Line #2 mode.
kCS42888_BusTDM TDM mode.

51.4.4.4 anonymous enum

Enumerator

kCS42888_AOUT1 aout1
kCS42888_AOUT2 aout2
kCS42888_AOUT3 aout3
kCS42888_AOUT4 aout4
kCS42888_AOUT5 aout5
kCS42888_AOUT6 aout6
kCS42888_AOUT7 aout7
kCS42888_AOUT8 aout8

51.4.5 Function Documentation

51.4.5.1 status_t CS42888_Init (cs42888_handle_t ∗ *handle,* cs42888_config_t ∗ *config*)

The second parameter is NULL to CS42888 in this version. If users want to change the settings, they have to use cs42888_write_reg() or cs42888_modify_reg() to set the register value of CS42888. Note: If the codec_config is NULL, it would initialize CS42888 using default settings. The default setting: codec_config->bus = kCS42888_BusI2S codec_config->ADCmode = kCS42888_ModeSlave codec_config->DACmode = kCS42888_ModeSlave

Parameters

handle	CS42888 handle structure.
config	CS42888 configuration structure.

51.4.5.2 status_t CS42888_Deinit (cs42888_handle_t ∗ *handle*)

This function close all modules in CS42888 to save power.

Parameters

handle	CS42888 handle structure pointer.

51.4.5.3 status_t CS42888_SetProtocol (cs42888_handle_t ∗ *handle,* cs42888_bus_t *protocol,* uint32_t *bitWidth*)

CS42888 only supports I2S, left justified, right justified, PCM A, PCM B format.

Parameters

handle	CS42888 handle structure.
protocol	Audio data transfer protocol.
bitWidth	bit width

51.4.5.4 void CS42888_SetFuncMode (cs42888_handle_t ∗ *handle,* cs42888_func_mode *mode*)

Deprecated api, Do not use it anymore. It has been superceded by CS42888_SelectFunctionalMode.

Parameters

handle	CS42888 handle structure.
mode	differenht working mode of CS42888.

51.4.5.5 status_t CS42888_SelectFunctionalMode (cs42888_handle_t ∗ *handle,* cs42888_func_mode *adcMode,* cs42888_func_mode *dacMode*)

Parameters

handle	CS42888 handle structure.
adcMode	differenht working mode of CS42888.
dacMode	differenht working mode of CS42888.

51.4.5.6 status_t CS42888_SetAOUTVolume (cs42888_handle_t ∗ *handle,* uint8_t *channel,* uint8_t *volume*)

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AOUT channel, it shall be 1∼8.
volume	Volume value need to be set.

51.4.5.7 status_t CS42888_SetAINVolume (cs42888_handle_t ∗ *handle,* uint8_t *channel,* uint8_t *volume*)

This function would set the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AIN channel, it shall be 1∼4.
volume	Volume value need to be set.

51.4.5.8 uint8_t CS42888_GetAOUTVolume (cs42888_handle_t ∗ *handle,* uint8_t *channel*)

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AOUT channel, it shall be 1∼8.

51.4.5.9 uint8_t CS42888_GetAINVolume (cs42888_handle_t ∗ *handle,* uint8_t *channel*)

This function gets the volume of CS42888 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

handle	CS42888 handle structure.
channel	AIN channel, it shall be 1∼4.

51.4.5.10 status_t CS42888_SetMute (cs42888_handle_t ∗ *handle,* uint8_t *channelMask*)

Parameters

handle	CS42888 handle structure.
channelMask	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.

51.4.5.11 status_t CS42888_SetChannelMute (cs42888_handle_t ∗ *handle,* uint8_t *channel,* bool *isMute*)

Parameters

handle	CS42888 handle structure.
channel	reference _cs42888_play_channel.
isMute	true is mute, falase is unmute.

51.4.5.12 status_t CS42888_SetModule (cs42888_handle_t ∗ *handle,* cs42888_module_t *module,* bool *isEnabled*)

Parameters

handle	CS42888 handle structure.
module	Module expected to enable.
isEnabled	Enable or disable moudles.

51.4.5.13 status_t CS42888_ConfigDataFormat (cs42888_handle_t * *handle,* uint32_t *mclk,* uint32_t *sample_rate,* uint32_t *bits*)

This function would configure the registers about the sample rate, bit depths.

Parameters

handle	CS42888 handle structure pointer.
mclk	Master clock frequency of I2S.
sample_rate	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.
bits	Bit depth of audio file (CS42888 only supports 16bit, 20bit, 24bit and 32 bit in HW).

51.4.5.14 status_t CS42888_WriteReg (cs42888_handle_t * *handle,* uint8_t *reg,* uint8_t *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
val	Value needs to write into the register.

51.4.5.15 status_t CS42888_ReadReg (cs42888_handle_t * *handle,* uint8_t *reg,* uint8_t * *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
val	Value written to.

51.4.5.16 status_t CS42888_ModifyReg (cs42888_handle_t ∗ *handle,* uint8_t *reg,* uint8_t *mask,* uint8_t *val*)

Parameters

handle	CS42888 handle structure.
reg	The register address in CS42888.
mask	The mask code for the bits want to write. The bit you want to write should be 0.
val	Value needs to write into the register.

51.4.6 CS42888 Adapter

51.4.6.1 Overview

The cs42888 adapter provides a codec unify control interface.

Macros

- #define HAL_CODEC_CS42888_HANDLER_SIZE (CS42888_I2C_HANDLER_SIZE + 4)
  codec handler size

Functions

- status_t HAL_CODEC_CS42888_Init (void *handle, void *config)
  Codec initilization.
- status_t HAL_CODEC_CS42888_Deinit (void *handle)
  Codec de-initilization.
- status_t HAL_CODEC_CS42888_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_CS42888_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_CS42888_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_CS42888_SetPower (void *handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_CS42888_SetRecord (void *handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_CS42888_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_CS42888_SetPlay (void *handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_CS42888_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
  codec module control.
- static status_t HAL_CODEC_Init (void *handle, void *config)
  Codec initilization.
- static status_t HAL_CODEC_Deinit (void *handle)
  Codec de-initilization.
- static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)

set audio codec module power.

- static status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- static status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- static status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- static status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

51.4.6.2 Function Documentation

51.4.6.2.1 status_t HAL_CODEC_CS42888_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

51.4.6.2.2 status_t HAL_CODEC_CS42888_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

51.4.6.2.3 status_t HAL_CODEC_CS42888_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.4 status_t HAL_CODEC_CS42888_SetVolume (void ∗ *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.5 status_t HAL_CODEC_CS42888_SetMute (void ∗ *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.6 status_t HAL_CODEC_CS42888_SetPower (void ∗ *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.7 status_t HAL_CODEC_CS42888_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.8 status_t HAL_CODEC_CS42888_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecordChannel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecordChannel	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.

51.4.6.2.9 status_t HAL_CODEC_CS42888_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.10 status_t HAL_CODEC_CS42888_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	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.

51.4.6.2.11 static status_t HAL_CODEC_Init (void * *handle,* void * *config*) `[inline], [static]`

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

51.4.6.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) `[inline], [static]`

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

51.4.6.2.13 static status_t HAL_CODEC_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*) [inline], [static]

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.14 static status_t HAL_CODEC_SetVolume (void ∗ *handle,* uint32_t *playChannel,* uint32_t *volume*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.15 static status_t HAL_CODEC_SetMute (void ∗ *handle,* uint32_t *playChannel,* bool *isMute*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.16 static status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*) [inline], [static]

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*) [inline], [static]

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*) [inline], [static]

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	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.

51.4.6.2.19 static status_t HAL_CODEC_SetPlay (void ∗ *handle,* uint32_t *playSource*) [inline], [static]

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

51.4.6.2.20 static status_t HAL_CODEC_ModuleControl (void ∗ *handle,* uint32_t *cmd,* uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	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.

51.5 TFA9896 Driver

51.5.1 Overview

The tfa9896 driver provides a codec control interface.

Data Structures

- struct tfa9896BiquadM_t
  biquadm More...
- struct tfa9896FilterM_t
  filter More...
- struct tfa9896_StateInfoLive_t
  status info live More...
- struct tfa9896SPKRBST_SpkrModel_t
  speaker mode More...
- struct tfa9896_audio_format_t
  Audio format configuration. More...
- struct tfa9896_config_t
  Initialize structure of TFA9896. More...
- struct tfa9896_handle_t
  tfa9896 handler More...

Macros

- #define TFA_I2C_BITRATE (400000U)
  TFA I2S bit clock.
- #define TFA9896_I2C_HANDLER_SIZE (CODEC_I2C_MASTER_HANDLER_SIZE)
  tfa9896 handle size

Typedefs

- typedef int int24
  type definition

Enumerations

- enum {
  kStatus_TFA9896_Ok = MAKE_STATUS(kStatusGroup_Generic, 0),
  kStatus_TFA9896_DSP_not_running = MAKE_STATUS(kStatusGroup_Generic, 1),
  kStatus_TFA9896_Bad_Parameter = MAKE_STATUS(kStatusGroup_Generic, 2),
  kStatus_TFA9896_NotOpen,
  kStatus_TFA9896_OutOfHandles = MAKE_STATUS(kStatusGroup_Generic, 4),
  kStatus_TFA9896_StateTimedOut,
  kStatus_TFA9896_RpcBase = MAKE_STATUS(kStatusGroup_Generic, 100),
  kStatus_TFA9896_RpcBusy = MAKE_STATUS(kStatusGroup_Generic, 101),
  kStatus_TFA9896_RpcModId = MAKE_STATUS(kStatusGroup_Generic, 102),
  kStatus_TFA9896_RpcParamId = MAKE_STATUS(kStatusGroup_Generic, 103),
  kStatus_TFA9896_RpcInfoId = MAKE_STATUS(kStatusGroup_Generic, 104),
  kStatus_TFA9896_RpcNotAllowedSpeaker,
  kStatus_TFA9896_Not_Implemented = MAKE_STATUS(kStatusGroup_Generic, 106),
  kStatus_TFA9896_Not_Supported = MAKE_STATUS(kStatusGroup_Generic, 107),
  kStatus_TFA9896_I2C_Fatal,
  kStatus_TFA9896_I2C_NonFatal,
  kStatus_TFA9896_Other = MAKE_STATUS(kStatusGroup_Generic, 1000) }
  status flag
- enum tfa9896_DMEM_t {
  DMEM_PMEM = 0,
  DMEM_XMEM = 1,
  DMEM_YMEM = 2,
  DMEM_IOMEM = 3 }
  mute dmem
- enum tfa9896_mute_t
  mute status
- enum tfa9896_SpeakerBoostStatusFlagsLive_t
  speaker boost status flag
- enum tfa9896_bit_width_t { ktfa9896_BitWidth16 = 0x0 }
  Bit width.
- enum tfa9896_protocol_t { kTFA9896_ProtocolI2S = 0x2 }
  The audio data transfer protocol.
- enum tfa9896_sample_rate_t
  Sample rate.

Functions

- uint16_t TFA9896_isFactory (tfa9896_handle_t *handle)
  check if TFA9896 Hardware factory registers are set .
- uint16_t TFA9896_GetBits (tfa9896_handle_t *handle, const uint16_t bf)
  get TFA9896 bitfield value .
- status_t TFA9896_SetBits (tfa9896_handle_t *handle, const uint16_t bf, const uint16_t value)
  set TFA9896 bitfield value .
- status_t TFA9896_DSP_System_Stable (tfa9896_handle_t *handle, int *ready)

wait for DSP to be stable before RPC communictaons .

- status_t TFA9896_ClearOneTimeCalibration (tfa9896_handle_t *handle)
  clear MTP regsiters to prepare calibraion .
- status_t TFA9896_SetOneTimeCalibration (tfa9896_handle_t *handle)
  set calibartion once registers.
- status_t TFA9896_SetFactoryValues (tfa9896_handle_t *handle)
  set MTP factory values .
- status_t TFA9896_CheckICROMversion (tfa9896_handle_t *handle, const unsigned char patchheader[ ])
  check DSP patch suitability to the TFA9896 device .
- status_t TFA9896_ProcessPatchFile (tfa9896_handle_t *handle, int length, const unsigned char *bytes)
  load DSP patch to DSP memeory .
- status_t TFA9896_DSPWriteTables (tfa9896_handle_t *handle)
  set DSP memery table .
- status_t TFA9896_DspWriteConfig (tfa9896_handle_t *handle, int length, const unsigned char *pConfigBytes)
  write DSP config file to DSP memery .
- status_t TFA9896_DspWriteSpeakerParameters (tfa9896_handle_t *handle, int length, const unsigned char *pSpeakerBytes)
  write DSP speaker file to DSP memery .
- status_t TFA9896_DspWritePreset (tfa9896_handle_t *handle, int length, const unsigned char *pPresetBytes)
  write DSP preset file to DSP memery .
- status_t TFA9896_Write_FilterBank (tfa9896_handle_t *handle, tfa9896FilterM_t *filter)
  write DSP filter params to DSP memery .
- status_t TFA9896_WaitCalibrateDone (tfa9896_handle_t *handle, int *calibrateDone)
  wait calibration to be finalized .
- status_t TFA9896_DspGetCalibrationImpedance (tfa9896_handle_t *handle, float *pRe25)
  retrieve calibration impedance value and expose it to user.
- status_t TFA9896_PrintCalibration (tfa9896_handle_t *handle)
  print calibration value .
- status_t TFA9896_DSP_Write_Mem_Word (tfa9896_handle_t *handle, unsigned short address, int value, int memtype)
  write word to IOMEM DSP memeory .
- status_t TFA9896_ReadRegister (tfa9896_handle_t *handle, uint8_t subaddress, unsigned short *value)
  read a hardware regsiter .
- status_t TFA9896_WriteRegister (tfa9896_handle_t *handle, uint8_t subaddress, uint16_t value)
  write a hardware register .
- status_t TFA9896_SetMute (tfa9896_handle_t *handle, tfa9896_mute_t mute)
  mute/unmute TFA9896 codec
- status_t TFA9896_SetVolume (tfa9896_handle_t *handle, uint8_t volume_level)
  Set amplifier volume.
- status_t TFA9896_SetOVPBypass (tfa9896_handle_t *handle)
  set OVP in bypass .
- status_t TFA9896_Init (tfa9896_handle_t *handle, tfa9896_config_t *tfa9896Config)
  Initialize TFA9896.
- status_t TFA9896_Deinit (tfa9896_handle_t *handle)
  DeInitilize TFA9896.

- status_t TFA9896_SetFormat (tfa9896_handle_t *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio input format .
- void TFA9896_Powerdown (tfa9896_handle_t *handle, int powerdown)
  power-on/off TFA98xx device.

51.5.2 Data Structure Documentation

51.5.2.1 struct tfa9896BiquadM_t

51.5.2.2 struct tfa9896FilterM_t

Data Fields

- uint8_t type
  (== enum FilterTypes, assure 8bits length)

51.5.2.3 struct tfa9896_StateInfoLive_t

51.5.2.4 struct tfa9896SPKRBST_SpkrModel_t

51.5.2.5 struct tfa9896_audio_format_t

Data Fields

- tfa9896_sample_rate_t sampleRate
  Sample rate.
- tfa9896_bit_width_t bitWidth
  Bit width.

51.5.2.6 struct tfa9896_config_t

Data Fields

- tfa9896_protocol_t protocol
  Audio transfer protocol.
- tfa9896_audio_format_t format
  Audio format.
- bool master
  true is master, false is slave
- uint8_t slaveAddress
  slave address
- codec_i2c_config_t i2cConfig
  i2c bus configuration
- uint8_t calibrate
  user handles calibration!

51.5.2.7 struct tfa9896_handle_t

Data Fields

- tfa9896_config_t * config
  tfa9896 config pointer

51.5.3 Enumeration Type Documentation

51.5.3.1 anonymous enum

Enumerator

kStatus_TFA9896_Ok kStatus_TFA9896_Ok = 0,
kStatus_TFA9896_DSP_not_running communication with the DSP failed, presumably because DSP not running kStatus_TFA9896_DSP_not_running
kStatus_TFA9896_Bad_Parameter kStatus_TFA9896_Bad_Parameter
kStatus_TFA9896_NotOpen kStatus_TFA9896_NotOpen, the given handle is not open
kStatus_TFA9896_OutOfHandles too many handles
kStatus_TFA9896_StateTimedOut the expected response did not occur within the expected time Tfa9896_Error_StateTimedOut, the expected response did not occur within the expected time
kStatus_TFA9896_RpcBase kStatus_TFA9896_RpcBase = 100,
kStatus_TFA9896_RpcBusy kStatus_TFA9896_RpcBusy = 101,
kStatus_TFA9896_RpcModId kStatus_TFA9896_RpcModId = 102,
kStatus_TFA9896_RpcParamId kStatus_TFA9896_RpcParamId = 103
kStatus_TFA9896_RpcInfoId kStatus_TFA9896_RpcInfoId = 104
kStatus_TFA9896_RpcNotAllowedSpeaker kStatus_TFA9896_RpcNotAllowedSpeaker = 105
kStatus_TFA9896_Not_Implemented kStatus_TFA9896_Not_Implemented
kStatus_TFA9896_Not_Supported kStatus_TFA9896_Not_Supported
kStatus_TFA9896_I2C_Fatal Fatal I2C error occurred kStatus_TFA9896_I2C_Fatal.
kStatus_TFA9896_I2C_NonFatal Nonfatal I2C error, and retry count reached.
kStatus_TFA9896_Other kStatus_TFA9896_Other = 1000

51.5.3.2 enum tfa9896_DMEM_t

Enumerator

DMEM_PMEM pmem
DMEM_XMEM xmem
DMEM_YMEM ymem
DMEM_IOMEM iomem

51.5.3.3 enum tfa9896_bit_width_t

Enumerator

ktfa9896_BitWidth16 16 bits

51.5.3.4 enum tfa9896_protocol_t

Enumerator

kTFA9896_ProtocolI2S I2S type.

51.5.3.5 enum tfa9896_sample_rate_t

51.5.4 Function Documentation

51.5.4.1 uint16_t TFA9896_isFactory (tfa9896_handle_t ∗ *handle*)

Parameters

handle	TFA9896 codec handle.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.2 uint16_t TFA9896_GetBits (tfa9896_handle_t ∗ *handle,* const uint16_t *bf*)

Parameters

handle	TFA9896 codec handle.
bf	to be read

Returns

Returns 16 bits read value.

51.5.4.3 status_t TFA9896_SetBits (tfa9896_handle_t ∗ *handle,* const uint16_t *bf,* const uint16_t *value*)

Parameters

handle	TFA9896 codec handle.
bf	to be set
value	to be set

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.4 status_t TFA9896_DSP_System_Stable (tfa9896_handle_t * *handle*, int * *ready*)

Parameters

handle	TFA9896 codec handle.
ready	DSP system stable status.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.5 status_t TFA9896_ClearOneTimeCalibration (tfa9896_handle_t * *handle*)

Parameters

handle	TFA9896 codec handle.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.6 status_t TFA9896_SetOneTimeCalibration (tfa9896_handle_t * *handle*)

Parameters

handle	TFA9896 codec handle.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.7 status_t TFA9896_SetFactoryValues (tfa9896_handle_t * *handle*)

Parameters

handle	TFA9896 codec handle.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.8 status_t TFA9896_CheckICROMversion (tfa9896_handle_t * *handle,* const unsigned char *patchheader[]*)

Parameters

handle	TFA9896 codec handle.
patchheader	pattern to check with.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.9 status_t TFA9896_ProcessPatchFile (tfa9896_handle_t * *handle,* int *length,* const unsigned char * *bytes*)

Parameters

handle	TFA9896 codec handle.

length	of the patch.
bytes	that contains patch data.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.10 status_t TFA9896_DSPWriteTables (tfa9896_handle_t * *handle*)

Parameters

handle	TFA9896 codec handle.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.11 status_t TFA9896_DspWriteConfig (tfa9896_handle_t * *handle*, int *length*, const unsigned char * *pConfigBytes*)

Parameters

handle	TFA9896 codec handle.
length	of the config.
pConfigBytes	that contains config data.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.12 status_t TFA9896_DspWriteSpeakerParameters (tfa9896_handle_t * *handle*, int *length*, const unsigned char * *pSpeakerBytes*)

Parameters

handle	TFA9896 codec handle.
length	of the speaker data.
pSpeakerBytes	that contains speaker data.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.13 status_t TFA9896_DspWritePreset (tfa9896_handle_t * *handle*, int *length*, const unsigned char * *pPresetBytes*)

Parameters

handle	TFA9896 codec handle.
length	of the preset data.
pPresetBytes	that contains preset data.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.14 status_t TFA9896_Write_FilterBank (tfa9896_handle_t * *handle*, tfa9896FilterM_t * *filter*)

Parameters

handle	TFA9896 codec handle.
filter	of the filter data.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.15 status_t TFA9896_WaitCalibrateDone (tfa9896_handle_t * *handle*, int * *calibrateDone*)

Parameters

handle	TFA9896 codec handle.
calibrateDone	calibration done flag.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.16 status_t TFA9896_DspGetCalibrationImpedance (tfa9896_handle_t * *handle*, float * *pRe25*)

Parameters

handle	TFA9896 codec handle.
pRe25	calibration value.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.17 status_t TFA9896_PrintCalibration (tfa9896_handle_t * *handle*)

Parameters

handle	TFA9896 codec handle.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.18 status_t TFA9896_DSP_Write_Mem_Word (tfa9896_handle_t * *handle*, unsigned short *address*, int *value*, int *memtype*)

Parameters

handle	TFA9896 codec handle.
address	to be written.
value	value write.
memtype	to be written.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.19 status_t TFA9896_ReadRegister (tfa9896_handle_t ∗ *handle,* uint8_t *subaddress,* unsigned short ∗ *value*)

Parameters

handle	TFA9896 codec handle.
subaddress	register address.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

Parameters

handle	TFA98xx codec handle.
subaddress	register address.
value	read out register value.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.20 status_t TFA9896_WriteRegister (tfa9896_handle_t ∗ *handle,* uint8_t *subaddress,* uint16_t *value*)

Parameters

handle	TFA9896 codec handle.
subaddress	register address.
value	value to write.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

Parameters

handle	TFA98xx codec handle.
subaddress	register address.
value	value to write.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.21 status_t TFA9896_SetMute (tfa9896_handle_t * *handle,* tfa9896_mute_t *mute*)

mute/unmute TFA98XX codec

Parameters

handle	TFA9896 codec handle.
mute	mute mode.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

Parameters

handle	TFA98xx codec handle.

mute	mute mode.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.22 status_t TFA9896_SetVolume (tfa9896_handle_t * *handle,* uint8_t *volume_level*)

Parameters

handle	TFA9896 codec handle.
volume_level	volume level.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.23 status_t TFA9896_SetOVPBypass (tfa9896_handle_t * *handle*)

Parameters

handle	TFA98xx codec handle.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.24 status_t TFA9896_Init (tfa9896_handle_t * *handle,* tfa9896_config_t * *tfa9896Config*)

Parameters

handle	TFA98xx codec handle.

tfa9896Config	Codec configuration.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.25 status_t TFA9896_Deinit (tfa9896_handle_t * *handle*)

Parameters

handle	TFA98xx codec handle.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.26 status_t TFA9896_SetFormat (tfa9896_handle_t * *handle*, uint32_t *mclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

Parameters

handle	TFA98xx codec handle.
mclk	master clock frequency
sampleRate	to be set.
bitWidth	to be set.

Returns

Returns kStatus_TFA9896_Ok if success, otherwise returns error code.

51.5.4.27 void TFA9896_Powerdown (tfa9896_handle_t * *handle*, int *powerdown*)

Parameters

handle	TFA98xx codec handle.
powerdown	to on/off

51.5.5 TFA9896 Adapter

51.5.5.1 Overview

The tfa9896 adapter provides a codec unify control interface.

Macros

- #define HAL_CODEC_TFA9896_HANDLER_SIZE (TFA9896_I2C_HANDLER_SIZE + 4)
  codec handler size

Functions

- status_t HAL_CODEC_TFA9896_Init (void *handle, void *config)
  Codec initilization.
- status_t HAL_CODEC_TFA9896_Deinit (void *handle)
  Codec de-initilization.
- status_t HAL_CODEC_TFA9896_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_TFA9896_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_TFA9896_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_TFA9896_SetPower (void *handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_TFA9896_SetRecord (void *handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_TFA9896_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_TFA9896_SetPlay (void *handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_TFA9896_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
  codec module control.
- static status_t HAL_CODEC_Init (void *handle, void *config)
  Codec initilization.
- static status_t HAL_CODEC_Deinit (void *handle)
  Codec de-initilization.
- static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)

set audio codec module power.
- static status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- static status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- static status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- static status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

51.5.5.2 Function Documentation

51.5.5.2.1 status_t HAL_CODEC_TFA9896_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

51.5.5.2.2 status_t HAL_CODEC_TFA9896_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

51.5.5.2.3 status_t HAL_CODEC_TFA9896_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.4 status_t HAL_CODEC_TFA9896_SetVolume (void ∗ *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.5 status_t HAL_CODEC_TFA9896_SetMute (void ∗ *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.6 status_t HAL_CODEC_TFA9896_SetPower (void ∗ *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.7 status_t HAL_CODEC_TFA9896_SetRecord (void * *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.8 status_t HAL_CODEC_TFA9896_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecordChannel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecordChannel	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.

51.5.5.2.9 status_t HAL_CODEC_TFA9896_SetPlay (void * *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.10 status_t HAL_CODEC_TFA9896_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*)

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	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.

51.5.5.2.11 static status_t HAL_CODEC_Init (void * *handle,* void * *config*) `[inline]`, `[static]`

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

51.5.5.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) `[inline]`, `[static]`

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

51.5.5.2.13 static status_t HAL_CODEC_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*) [inline], [static]

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.14 static status_t HAL_CODEC_SetVolume (void ∗ *handle,* uint32_t *playChannel,* uint32_t *volume*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.15 static status_t HAL_CODEC_SetMute (void ∗ *handle,* uint32_t *playChannel,* bool *isMute*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.16 static status_t HAL_CODEC_SetPower (void ∗ *handle,* uint32_t *module,* bool *powerOn*) [inline], [static]

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.17 static status_t HAL_CODEC_SetRecord (void ∗ *handle,* uint32_t *recordSource*) [inline], [static]

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.18 static status_t HAL_CODEC_SetRecordChannel (void ∗ *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*) [inline], [static]

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	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.

51.5.5.2.19 static status_t HAL_CODEC_SetPlay (void ∗ *handle,* uint32_t *playSource*) [inline], [static]

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

51.5.5.2.20 static status_t HAL_CODEC_ModuleControl (void ∗ *handle,* uint32_t *cmd,* uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	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.

51.6 WM8904 Driver

51.6.1 Overview

The wm8904 driver provides a codec control interface.

Data Structures

- struct wm8904_fll_config_t
  wm8904 fll configuration More...
- struct wm8904_audio_format_t
  Audio format configuration. More...
- struct wm8904_config_t
  Configuration structure of WM8904. More...
- struct wm8904_handle_t
  wm8904 codec handler More...

Macros

- #define WM8904_I2C_HANDLER_SIZE (CODEC_I2C_MASTER_HANDLER_SIZE)
  wm8904 handle size
- #define WM8904_DEBUG_REGISTER 0
  wm8904 debug macro
- #define WM8904_RESET (0x00)
  WM8904 register map.
- #define WM8904_I2C_ADDRESS (0x1A)
  WM8904 I2C address.
- #define WM8904_I2C_BITRATE (400000U)
  WM8904 I2C bit rate.
- #define WM8904_MAP_HEADPHONE_LINEOUT_MAX_VOLUME 0x3FU
  WM8904 maximum volume.

Enumerations

- enum {
  kStatus_WM8904_Success = 0x0,
  kStatus_WM8904_Fail = 0x1 }
  WM8904 status return codes.
- enum {
  kWM8904_LRCPolarityNormal = 0U,
  kWM8904_LRCPolarityInverted = 1U << 4U }
  WM8904 lrc polarity.
- enum wm8904_module_t {

kWM8904_ModuleADC = 0,
kWM8904_ModuleDAC = 1,
kWM8904_ModulePGA = 2,
kWM8904_ModuleHeadphone = 3,
kWM8904_ModuleLineout = 4 }

wm8904 module value

- enum

wm8904 play channel

- enum wm8904_timeslot_t {
kWM8904_TimeSlot0 = 0U,
kWM8904_TimeSlot1 = 1U }

WM8904 time slot.

- enum wm8904_protocol_t {
kWM8904_ProtocolI2S = 0x2,
kWM8904_ProtocolLeftJustified = 0x1,
kWM8904_ProtocolRightJustified = 0x0,
kWM8904_ProtocolPCMA = 0x3,
kWM8904_ProtocolPCMB = 0x3 | (1 << 4) }

The audio data transfer protocol.

- enum wm8904_fs_ratio_t {
kWM8904_FsRatio64X = 0x0,
kWM8904_FsRatio128X = 0x1,
kWM8904_FsRatio192X = 0x2,
kWM8904_FsRatio256X = 0x3,
kWM8904_FsRatio384X = 0x4,
kWM8904_FsRatio512X = 0x5,
kWM8904_FsRatio768X = 0x6,
kWM8904_FsRatio1024X = 0x7,
kWM8904_FsRatio1408X = 0x8,
kWM8904_FsRatio1536X = 0x9 }

The SYSCLK / fs ratio.

- enum wm8904_sample_rate_t {
kWM8904_SampleRate8kHz = 0x0,
kWM8904_SampleRate12kHz = 0x1,
kWM8904_SampleRate16kHz = 0x2,
kWM8904_SampleRate24kHz = 0x3,
kWM8904_SampleRate32kHz = 0x4,
kWM8904_SampleRate48kHz = 0x5,
kWM8904_SampleRate11025Hz = 0x6,
kWM8904_SampleRate22050Hz = 0x7,
kWM8904_SampleRate44100Hz = 0x8 }

Sample rate.

- enum wm8904_bit_width_t {
kWM8904_BitWidth16 = 0x0,
kWM8904_BitWidth20 = 0x1,
kWM8904_BitWidth24 = 0x2,

kWM8904_BitWidth32 = 0x3 }
Bit width.

- enum {
kWM8904_RecordSourceDifferentialLine = 1U,
kWM8904_RecordSourceLineInput = 2U,
kWM8904_RecordSourceDifferentialMic = 4U,
kWM8904_RecordSourceDigitalMic = 8U }
wm8904 record source
- enum {
kWM8904_RecordChannelLeft1 = 1U,
kWM8904_RecordChannelLeft2 = 2U,
kWM8904_RecordChannelLeft3 = 4U,
kWM8904_RecordChannelRight1 = 1U,
kWM8904_RecordChannelRight2 = 2U,
kWM8904_RecordChannelRight3 = 4U,
kWM8904_RecordChannelDifferentialPositive1 = 1U,
kWM8904_RecordChannelDifferentialPositive2 = 2U,
kWM8904_RecordChannelDifferentialPositive3 = 4U,
kWM8904_RecordChannelDifferentialNegative1 = 8U,
kWM8904_RecordChannelDifferentialNegative2 = 16U,
kWM8904_RecordChannelDifferentialNegative3 = 32U }
wm8904 record channel
- enum {
kWM8904_PlaySourcePGA = 1U,
kWM8904_PlaySourceDAC = 4U }
wm8904 play source
- enum wm8904_sys_clk_source_t {
kWM8904_SysClkSourceMCLK = 0U,
kWM8904_SysClkSourceFLL = 1U << 14 }
wm8904 system clock source
- enum wm8904_fll_clk_source_t { kWM8904_FLLClkSourceMCLK = 0U }
wm8904 fll clock source

Functions

- status_t WM8904_WriteRegister (wm8904_handle_t *handle, uint8_t reg, uint16_t value)
WM8904 write register.
- status_t WM8904_ReadRegister (wm8904_handle_t *handle, uint8_t reg, uint16_t *value)
WM8904 write register.
- status_t WM8904_ModifyRegister (wm8904_handle_t *handle, uint8_t reg, uint16_t mask, uint16_t value)
WM8904 modify register.
- status_t WM8904_Init (wm8904_handle_t *handle, wm8904_config_t *wm8904Config)
Initializes WM8904.
- status_t WM8904_Deinit (wm8904_handle_t *handle)
Deinitializes the WM8904 codec.
- void WM8904_GetDefaultConfig (wm8904_config_t *config)

Fills the configuration structure with default values.

- status_t WM8904_SetMasterSlave (wm8904_handle_t *handle, bool master)
  Sets WM8904 as master or slave.
- status_t WM8904_SetMasterClock (wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth)
  Sets WM8904 master clock configuration.
- status_t WM8904_SetFLLConfig (wm8904_handle_t *handle, wm8904_fll_config_t *config)
  WM8904 set PLL configuration This function will enable the GPIO1 FLL clock output function, so user can see the generated fll output clock frequency from WM8904 GPIO1.
- status_t WM8904_SetProtocol (wm8904_handle_t *handle, wm8904_protocol_t protocol)
  Sets the audio data transfer protocol.
- status_t WM8904_SetAudioFormat (wm8904_handle_t *handle, uint32_t sysclk, uint32_t sampleRate, uint32_t bitWidth)
  Sets the audio data format.
- status_t WM8904_CheckAudioFormat (wm8904_handle_t *handle, wm8904_audio_format_t *format, uint32_t mclkFreq)
  check and update the audio data format.
- status_t WM8904_SetVolume (wm8904_handle_t *handle, uint16_t volumeLeft, uint16_t volumeRight)
  Sets the module output volume.
- status_t WM8904_SetMute (wm8904_handle_t *handle, bool muteLeft, bool muteRight)
  Sets the headphone output mute.
- status_t WM8904_SelectLRCPolarity (wm8904_handle_t *handle, uint32_t polarity)
  Select LRC polarity.
- status_t WM8904_EnableDACTDMMode (wm8904_handle_t *handle, wm8904_timeslot_t timeSlot)
  Enable WM8904 DAC time slot.
- status_t WM8904_EnableADCTDMMode (wm8904_handle_t *handle, wm8904_timeslot_t timeSlot)
  Enable WM8904 ADC time slot.
- status_t WM8904_SetModulePower (wm8904_handle_t *handle, wm8904_module_t module, bool isEnabled)
  SET the module output power.
- status_t WM8904_SetDACVolume (wm8904_handle_t *handle, uint8_t volume)
  SET the DAC module volume.
- status_t WM8904_SetChannelVolume (wm8904_handle_t *handle, uint32_t channel, uint32_t volume)
  Sets the channel output volume.
- status_t WM8904_SetRecord (wm8904_handle_t *handle, uint32_t recordSource)
  SET the WM8904 record source.
- status_t WM8904_SetRecordChannel (wm8904_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  SET the WM8904 record source.
- status_t WM8904_SetPlay (wm8904_handle_t *handle, uint32_t playSource)
  SET the WM8904 play source.
- status_t WM8904_SetChannelMute (wm8904_handle_t *handle, uint32_t channel, bool isMute)
  Sets the channel mute.

Driver version

- #define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))
  WM8904 driver version 2.5.0.

51.6.2 Data Structure Documentation

51.6.2.1 struct wm8904_fll_config_t

Data Fields

- wm8904_fll_clk_source_t source
  fll reference clock source
- uint32_t refClock_HZ
  fll reference clock frequency
- uint32_t outputClock_HZ
  fll output clock frequency

51.6.2.2 struct wm8904_audio_format_t

Data Fields

- wm8904_fs_ratio_t fsRatio
  SYSCLK / fs ratio.
- wm8904_sample_rate_t sampleRate
  Sample rate.
- wm8904_bit_width_t bitWidth
  Bit width.

51.6.2.3 struct wm8904_config_t

Data Fields

- bool master
  Master or slave.
- wm8904_sys_clk_source_t sysClkSource
  system clock source
- wm8904_fll_config_t * fll
  fll configuration
- wm8904_protocol_t protocol
  Audio transfer protocol.
- wm8904_audio_format_t format
  Audio format.
- uint32_t mclk_HZ
  MCLK frequency value.
- uint16_t recordSource
  record source

- uint16_t recordChannelLeft
  record channel
- uint16_t recordChannelRight
  record channel
- uint16_t playSource
  play source
- uint8_t slaveAddress
  code device slave address
- codec_i2c_config_t i2cConfig
  i2c bus configuration

51.6.2.4 struct wm8904_handle_t

Data Fields

- wm8904_config_t * config
  wm8904 config pointer
- uint8_t i2cHandle [WM8904_I2C_HANDLER_SIZE]
  i2c handle

51.6.3 Macro Definition Documentation

51.6.3.1 #define FSL_WM8904_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))

51.6.3.2 #define WM8904_I2C_ADDRESS (0x1A)

51.6.3.3 #define WM8904_I2C_BITRATE (400000U)

51.6.4 Enumeration Type Documentation

51.6.4.1 anonymous enum

Enumerator

kStatus_WM8904_Success Success.
kStatus_WM8904_Fail Failure.

51.6.4.2 anonymous enum

Enumerator

kWM8904_LRCPolarityNormal LRC polarity normal.
kWM8904_LRCPolarityInverted LRC polarity inverted.

51.6.4.3 enum wm8904_module_t

Enumerator

kWM8904_ModuleADC moduel ADC
kWM8904_ModuleDAC module DAC
kWM8904_ModulePGA module PGA
kWM8904_ModuleHeadphone module headphone
kWM8904_ModuleLineout module line out

51.6.4.4 anonymous enum

51.6.4.5 enum wm8904_timeslot_t

Enumerator

kWM8904_TimeSlot0 time slot0
kWM8904_TimeSlot1 time slot1

51.6.4.6 enum wm8904_protocol_t

Enumerator

kWM8904_ProtocolI2S I2S type.
kWM8904_ProtocolLeftJustified Left justified mode.
kWM8904_ProtocolRightJustified Right justified mode.
kWM8904_ProtocolPCMA PCM A mode.
kWM8904_ProtocolPCMB PCM B mode.

51.6.4.7 enum wm8904_fs_ratio_t

Enumerator

kWM8904_FsRatio64X SYSCLK is 64 ∗ sample rate ∗ frame width.
kWM8904_FsRatio128X SYSCLK is 128 ∗ sample rate ∗ frame width.
kWM8904_FsRatio192X SYSCLK is 192 ∗ sample rate ∗ frame width.
kWM8904_FsRatio256X SYSCLK is 256 ∗ sample rate ∗ frame width.
kWM8904_FsRatio384X SYSCLK is 384 ∗ sample rate ∗ frame width.
kWM8904_FsRatio512X SYSCLK is 512 ∗ sample rate ∗ frame width.
kWM8904_FsRatio768X SYSCLK is 768 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1024X SYSCLK is 1024 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1408X SYSCLK is 1408 ∗ sample rate ∗ frame width.
kWM8904_FsRatio1536X SYSCLK is 1536 ∗ sample rate ∗ frame width.

51.6.4.8 enum wm8904_sample_rate_t

Enumerator

kWM8904_SampleRate8kHz 8 kHz
kWM8904_SampleRate12kHz 12kHz
kWM8904_SampleRate16kHz 16kHz
kWM8904_SampleRate24kHz 24kHz
kWM8904_SampleRate32kHz 32kHz
kWM8904_SampleRate48kHz 48kHz
kWM8904_SampleRate11025Hz 11.025kHz
kWM8904_SampleRate22050Hz 22.05kHz
kWM8904_SampleRate44100Hz 44.1kHz

51.6.4.9 enum wm8904_bit_width_t

Enumerator

kWM8904_BitWidth16 16 bits
kWM8904_BitWidth20 20 bits
kWM8904_BitWidth24 24 bits
kWM8904_BitWidth32 32 bits

51.6.4.10 anonymous enum

Enumerator

kWM8904_RecordSourceDifferentialLine record source from differential line
kWM8904_RecordSourceLineInput record source from line input
kWM8904_RecordSourceDifferentialMic record source from differential mic
kWM8904_RecordSourceDigitalMic record source from digital microphone

51.6.4.11 anonymous enum

Enumerator

kWM8904_RecordChannelLeft1 left record channel 1
kWM8904_RecordChannelLeft2 left record channel 2
kWM8904_RecordChannelLeft3 left record channel 3
kWM8904_RecordChannelRight1 right record channel 1
kWM8904_RecordChannelRight2 right record channel 2
kWM8904_RecordChannelRight3 right record channel 3
kWM8904_RecordChannelDifferentialPositive1 differential positive record channel 1
kWM8904_RecordChannelDifferentialPositive2 differential positive record channel 2
kWM8904_RecordChannelDifferentialPositive3 differential positive record channel 3

kWM8904_RecordChannelDifferentialNegative1 differential negative record channel 1
kWM8904_RecordChannelDifferentialNegative2 differential negative record channel 2
kWM8904_RecordChannelDifferentialNegative3 differential negative record channel 3

51.6.4.12 anonymous enum

Enumerator

kWM8904_PlaySourcePGA play source PGA, bypass ADC
kWM8904_PlaySourceDAC play source Input3

51.6.4.13 enum wm8904_sys_clk_source_t

Enumerator

kWM8904_SysClkSourceMCLK wm8904 system clock soure from MCLK
kWM8904_SysClkSourceFLL wm8904 system clock soure from FLL

51.6.4.14 enum wm8904_fll_clk_source_t

Enumerator

kWM8904_FLLClkSourceMCLK wm8904 FLL clock source from MCLK

51.6.5 Function Documentation

51.6.5.1 status_t WM8904_WriteRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
value	value to write.

Returns

kStatus_Success, else failed.

51.6.5.2 status_t WM8904_ReadRegister (wm8904_handle_t * *handle*, uint8_t *reg*, uint16_t * *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
value	value to read.

Returns

kStatus_Success, else failed.

51.6.5.3 status_t WM8904_ModifyRegister (wm8904_handle_t * *handle,* uint8_t *reg,* uint16_t *mask,* uint16_t *value*)

Parameters

handle	WM8904 handle structure.
reg	register address.
mask	register bits mask.
value	value to write.

Returns

kStatus_Success, else failed.

51.6.5.4 status_t WM8904_Init (wm8904_handle_t * *handle,* wm8904_config_t * *wm8904Config*)

Parameters

handle	WM8904 handle structure.
wm8904Config	WM8904 configuration structure.

51.6.5.5 status_t WM8904_Deinit (wm8904_handle_t * *handle*)

This function resets WM8904.

Parameters

handle	WM8904 handle structure.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.6 void WM8904_GetDefaultConfig (wm8904_config_t * *config*)

The default values are:

master = false; protocol = kWM8904_ProtocolI2S; format.fsRatio = kWM8904_FsRatio64X; format.-sampleRate = kWM8904_SampleRate48kHz; format.bitWidth = kWM8904_BitWidth16;

Parameters

config	default configurations of wm8904.

51.6.5.7 status_t WM8904_SetMasterSlave (wm8904_handle_t * *handle*, bool *master*)

Deprecated DO NOT USE THIS API ANYMORE. IT HAS BEEN SUPERCEDED BY WM8904_Set-MasterClock

Parameters

handle	WM8904 handle structure.
master	true for master, false for slave.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.8 status_t WM8904_SetMasterClock (wm8904_handle_t * *handle*, uint32_t *sysclk*, uint32_t *sampleRate*, uint32_t *bitWidth*)

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

Parameters

handle	WM8904 handle structure.
sysclk	system clock source frequency.
sampleRate	sample rate
bitWidth	bit width

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.9 status_t WM8904_SetFLLConfig (wm8904_handle_t * *handle,* wm8904_fll_config_t * *config*)

Parameters

handle	wm8904 handler pointer.
config	FLL configuration pointer.

51.6.5.10 status_t WM8904_SetProtocol (wm8904_handle_t * *handle,* wm8904_protocol_t *protocol*)

Parameters

handle	WM8904 handle structure.
protocol	Audio transfer protocol.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.11 status_t WM8904_SetAudioFormat (wm8904_handle_t * *handle,* uint32_t *sysclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

User should pay attention to the sysclk parameter ,When using external MCLK as system clock source, the value should be frequency of MCLK, when using FLL as system clock source, the value should be frequency of the output of FLL.

Parameters

handle	WM8904 handle structure.
sysclk	system clock source frequency.
sampleRate	Sample rate frequency in Hz.
bitWidth	Audio data bit width.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.12 status_t WM8904_CheckAudioFormat (wm8904_handle_t * *handle*, wm8904_audio_format_t * *format*, uint32_t *mclkFreq*)

This api is used check the fsRatio setting based on the mclk and sample rate, if fsRatio setting is not correct, it will correct it according to mclk and sample rate.

Parameters

handle	WM8904 handle structure.
format	audio data format
mclkFreq	mclk frequency

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.13 status_t WM8904_SetVolume (wm8904_handle_t * *handle*, uint16_t *volumeLeft*, uint16_t *volumeRight*)

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57DB, 63 for 6DB.

Parameters

handle	WM8904 handle structure.

volumeLeft	left channel volume.
volumeRight	right channel volume.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.14 status_t WM8904_SetMute (wm8904_handle_t * *handle,* bool *muteLeft,* bool *muteRight*)

Parameters

handle	WM8904 handle structure.
muteLeft	true to mute left channel, false to unmute.
muteRight	true to mute right channel, false to unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.15 status_t WM8904_SelectLRCPolarity (wm8904_handle_t * *handle,* uint32_t *polarity*)

Parameters

handle	WM8904 handle structure.
polarity	LRC clock polarity.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.16 status_t WM8904_EnableDACTDMMode (wm8904_handle_t * *handle,* wm8904_timeslot_t *timeSlot*)

Parameters

handle	WM8904 handle structure.
timeSlot	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.17 status_t WM8904_EnableADCTDMMode (wm8904_handle_t * *handle,* wm8904_timeslot_t *timeSlot*)

Parameters

handle	WM8904 handle structure.
timeSlot	timeslot number.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.18 status_t WM8904_SetModulePower (wm8904_handle_t * *handle,* wm8904_module_t *module,* bool *isEnabled*)

Parameters

handle	WM8904 handle structure.
module	wm8904 module.
isEnabled,true	is power on, false is power down.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

51.6.5.19 status_t WM8904_SetDACVolume (wm8904_handle_t * *handle,* uint8_t *volume*)

Parameters

handle	WM8904 handle structure.
volume	volume to be configured.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

51.6.5.20 status_t WM8904_SetChannelVolume (wm8904_handle_t * *handle,* uint32_t *channel,* uint32_t *volume*)

The parameter should be from 0 to 63. The resulting volume will be. 0 for -57dB, 63 for 6DB.

Parameters

handle	codec handle structure.
channel	codec channel.
volume	volume value from 0 -63.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.21 status_t WM8904_SetRecord (wm8904_handle_t * *handle,* uint32_t *recordSource*)

Parameters

handle	WM8904 handle structure.
recordSource	record source , can be a value of kCODEC_ModuleRecordSourceDifferentialLine, kCODEC_ModuleRecordSourceDifferentialMic, kCODEC_ModuleRecordSourceSingleEndMic, kCODEC_ModuleRecordSourceDigitalMic.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.5.22 status_t WM8904_SetRecordChannel (wm8904_handle_t * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	WM8904 handle structure.
leftRecord-Channel	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.
rightRecord-Channel	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..

51.6.5.23 status_t WM8904_SetPlay (wm8904_handle_t ∗ *handle,* uint32_t *playSource*)

Parameters

handle	WM8904 handle structure.
playSource	play source , can be a value of kCODEC_ModuleHeadphoneSourcePGA, kCODEC-_ModuleHeadphoneSourceDAC, kCODEC_ModuleLineoutSourcePGA, kCODEC-_ModuleLineoutSourceDAC.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

51.6.5.24 status_t WM8904_SetChannelMute (wm8904_handle_t ∗ *handle,* uint32_t *channel,* bool *isMute*)

Parameters

handle	codec handle structure.
channel	codec module name.
isMute	true is mute, false unmute.

Returns

kStatus_WM8904_Success if successful, different code otherwise.

51.6.6 WM8904 Adapter

51.6.6.1 Overview

The wm8904 adapter provides a codec unify control interface.

Macros

- #define HAL_CODEC_WM8904_HANDLER_SIZE (WM8904_I2C_HANDLER_SIZE + 4)
  codec handler size

Functions

- status_t HAL_CODEC_WM8904_Init (void *handle, void *config)
  Codec initilization.
- status_t HAL_CODEC_WM8904_Deinit (void *handle)
  Codec de-initilization.
- status_t HAL_CODEC_WM8904_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- status_t HAL_CODEC_WM8904_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- status_t HAL_CODEC_WM8904_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- status_t HAL_CODEC_WM8904_SetPower (void *handle, uint32_t module, bool powerOn)
  set audio codec module power.
- status_t HAL_CODEC_WM8904_SetRecord (void *handle, uint32_t recordSource)
  codec set record source.
- status_t HAL_CODEC_WM8904_SetRecordChannel (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- status_t HAL_CODEC_WM8904_SetPlay (void *handle, uint32_t playSource)
  codec set play source.
- status_t HAL_CODEC_WM8904_ModuleControl (void *handle, uint32_t cmd, uint32_t data)
  codec module control.
- static status_t HAL_CODEC_Init (void *handle, void *config)
  Codec initilization.
- static status_t HAL_CODEC_Deinit (void *handle)
  Codec de-initilization.
- static status_t HAL_CODEC_SetFormat (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
  set audio data format.
- static status_t HAL_CODEC_SetVolume (void *handle, uint32_t playChannel, uint32_t volume)
  set audio codec module volume.
- static status_t HAL_CODEC_SetMute (void *handle, uint32_t playChannel, bool isMute)
  set audio codec module mute.
- static status_t HAL_CODEC_SetPower (void *handle, uint32_t module, bool powerOn)

set audio codec module power.

- static status_t HAL_CODEC_SetRecord (void ∗handle, uint32_t recordSource)
  codec set record source.
- static status_t HAL_CODEC_SetRecordChannel (void ∗handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
  codec set record channel.
- static status_t HAL_CODEC_SetPlay (void ∗handle, uint32_t playSource)
  codec set play source.
- static status_t HAL_CODEC_ModuleControl (void ∗handle, uint32_t cmd, uint32_t data)
  codec module control.

51.6.6.2 Function Documentation

51.6.6.2.1 status_t HAL_CODEC_WM8904_Init (void ∗ *handle,* void ∗ *config*)

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

51.6.6.2.2 status_t HAL_CODEC_WM8904_Deinit (void ∗ *handle*)

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

51.6.6.2.3 status_t HAL_CODEC_WM8904_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*)

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.4 status_t HAL_CODEC_WM8904_SetVolume (void ∗ *handle,* uint32_t *playChannel,* uint32_t *volume*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.5 status_t HAL_CODEC_WM8904_SetMute (void ∗ *handle,* uint32_t *playChannel,* bool *isMute*)

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.6 status_t HAL_CODEC_WM8904_SetPower (void ∗ *handle,* uint32_t *module,* bool *powerOn*)

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.7 status_t HAL_CODEC_WM8904_SetRecord (void ∗ *handle,* uint32_t *recordSource*)

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.8 status_t HAL_CODEC_WM8904_SetRecordChannel (void ∗ *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*)

Parameters

handle	codec handle.
leftRecordChannel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecordChannel	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.

51.6.6.2.9 status_t HAL_CODEC_WM8904_SetPlay (void ∗ *handle,* uint32_t *playSource*)

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.10 status_t HAL_CODEC_WM8904_ModuleControl (void * *handle,* uint32_t *cmd,* uint32_t *data*)

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	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.

51.6.6.2.11 static status_t HAL_CODEC_Init (void * *handle,* void * *config*) `[inline], [static]`

Parameters

handle	codec handle.
config	codec configuration.

Returns

kStatus_Success is success, else initial failed.

51.6.6.2.12 static status_t HAL_CODEC_Deinit (void * *handle*) `[inline], [static]`

Parameters

handle	codec handle.

Returns

kStatus_Success is success, else de-initial failed.

51.6.6.2.13 static status_t HAL_CODEC_SetFormat (void ∗ *handle,* uint32_t *mclk,* uint32_t *sampleRate,* uint32_t *bitWidth*) [inline], [static]

Parameters

handle	codec handle.
mclk	master clock frequency in HZ.
sampleRate	sample rate in HZ.
bitWidth	bit width.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.14 static status_t HAL_CODEC_SetVolume (void ∗ *handle,* uint32_t *playChannel,* uint32_t *volume*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
volume	volume value, support 0 ∼ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.15 static status_t HAL_CODEC_SetMute (void ∗ *handle,* uint32_t *playChannel,* bool *isMute*) [inline], [static]

Parameters

handle	codec handle.
playChannel	audio codec play channel, can be a value or combine value of _codec_play_channel.
isMute	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.16 static status_t HAL_CODEC_SetPower (void * *handle,* uint32_t *module,* bool *powerOn*) [inline], [static]

Parameters

handle	codec handle.
module	audio codec module.
powerOn	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.17 static status_t HAL_CODEC_SetRecord (void * *handle,* uint32_t *recordSource*) [inline], [static]

Parameters

handle	codec handle.
recordSource	audio codec record source, can be a value or combine value of _codec_record_source.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.18 static status_t HAL_CODEC_SetRecordChannel (void * *handle,* uint32_t *leftRecordChannel,* uint32_t *rightRecordChannel*) [inline], [static]

Parameters

handle	codec handle.
leftRecord-Channel	audio codec record channel, reference _codec_record_channel, can be a value or combine value of member in _codec_record_channel.
rightRecord-Channel	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.

51.6.6.2.19 static status_t HAL_CODEC_SetPlay (void ∗ *handle,* uint32_t *playSource*) [inline], [static]

Parameters

handle	codec handle.
playSource	audio codec play source, can be a value or combine value of _codec_play_source.

Returns

kStatus_Success is success, else configure failed.

51.6.6.2.20 static status_t HAL_CODEC_ModuleControl (void ∗ *handle,* uint32_t *cmd,* uint32_t *data*) [inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

handle	codec handle.
cmd	module control cmd, reference _codec_module_ctrl_cmd.
data	value to write, when cmd is kCODEC_ModuleRecordSourceChannel, the data should be a value combine of channel and source, please reference macro CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN), reference codec specific driver for detail configurations.

Returns

kStatus_Success is success, else configure failed.

Chapter 52
Serial Manager

52.1 Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are UART, USB CDC and SWO.

Modules

- Serial Port SWO
- Serial Port USB
- Serial Port Uart

Data Structures

- struct serial_manager_config_t
  serial manager config structure More...
- struct serial_manager_callback_message_t
  Callback message structure. More...

Macros

- #define SERIAL_MANAGER_NON_BLOCKING_MODE (0U)
  Enable or disable serial manager non-blocking mode (1 - enable, 0 - disable)
- #define SERIAL_MANAGER_RING_BUFFER_FLOWCONTROL (0U)
  Enable or ring buffer flow control (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_UART (0U)
  Enable or disable uart port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_UART_DMA (0U)
  Enable or disable uart dma port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_USBCDC (0U)
  Enable or disable USB CDC port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SWO (0U)
  Enable or disable SWO port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_VIRTUAL (0U)
  Enable or disable USB CDC virtual port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_RPMSG (0U)
  Enable or disable rPMSG port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SPI_MASTER (0U)
  Enable or disable SPI Master port (1 - enable, 0 - disable)
- #define SERIAL_PORT_TYPE_SPI_SLAVE (0U)
  Enable or disable SPI Slave port (1 - enable, 0 - disable)
- #define SERIAL_MANAGER_TASK_HANDLE_TX (0U)
  Enable or disable SerialManager_Task() handle TX to prevent recursive calling.

- #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)
  Set the default delay time in ms used by SerialManager_WriteTimeDelay().
- #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)
  Set the default delay time in ms used by SerialManager_ReadTimeDelay().
- #define SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY (0U)
  Enable or disable SerialManager_Task() handle RX data available notify.
- #define SERIAL_MANAGER_WRITE_HANDLE_SIZE (4U)
  Set serial manager write handle size.
- #define SERIAL_MANAGER_USE_COMMON_TASK (0U)
  SERIAL_PORT_UART_HANDLE_SIZE/SERIAL_PORT_USB_CDC_HANDLE_SIZE + serial manager dedicated size.
- #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)
  Macro to determine whether use common task.
- #define SERIAL_MANAGER_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager handle.
- #define SERIAL_MANAGER_WRITE_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager write handle.
- #define SERIAL_MANAGER_READ_HANDLE_DEFINE(name) uint32_t name[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]
  Defines the serial manager read handle.
- #define SERIAL_MANAGER_TASK_PRIORITY (2U)
  Macro to set serial manager task priority.
- #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)
  Macro to set serial manager task stack size.

Typedefs

- typedef void * serial_handle_t
  The handle of the serial manager module.
- typedef void * serial_write_handle_t
  The write handle of the serial manager module.
- typedef void * serial_read_handle_t
  The read handle of the serial manager module.
- typedef void(* serial_manager_callback_t)(void *callbackParam, serial_manager_callback_message_t *message, serial_manager_status_t status)
  callback function

Enumerations

- enum serial_port_type_t {
  kSerialPort_None = 0U,
  kSerialPort_Uart = 1U,
  kSerialPort_Uart = 1U,
  kSerialPort_UsbCdc,
  kSerialPort_Swo,
  kSerialPort_Virtual,
  kSerialPort_Rpmsg,
  kSerialPort_UartDma,
  kSerialPort_SpiMaster,
  kSerialPort_SpiSlave,
  kSerialPort_None }
  serial port type
- enum serial_manager_type_t {
  kSerialManager_NonBlocking = 0x0U,
  kSerialManager_Blocking = 0x8F41U }
  serial manager type
- enum serial_manager_status_t {
  kStatus_SerialManager_Success = kStatus_Success,
  kStatus_SerialManager_Error = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1),
  kStatus_SerialManager_Busy = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2),
  kStatus_SerialManager_Notify = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3),
  kStatus_SerialManager_Canceled,
  kStatus_SerialManager_HandleConflict = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5),
  kStatus_SerialManager_RingBufferOverflow,
  kStatus_SerialManager_NotConnected = MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7) }
  serial manager error code

Functions

- serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const serial_manager_-config_t *config)
  Initializes a serial manager module with the serial manager handle and the user configuration structure.
- serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)
  De-initializes the serial manager module instance.
- serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_-write_handle_t writeHandle)
  Opens a writing handle for the serial manager module.
- serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)
  Closes a writing handle for the serial manager module.
- serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_-read_handle_t readHandle)
  Opens a reading handle for the serial manager module.
- serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t readHandle)

Closes a reading for the serial manager module.
- serial_manager_status_t SerialManager_WriteBlocking (serial_write_handle_t writeHandle, uint8_t *buffer, uint32_t length)
  Transmits data with the blocking mode.
- serial_manager_status_t SerialManager_ReadBlocking (serial_read_handle_t readHandle, uint8_t *buffer, uint32_t length)
  Reads data with the blocking mode.
- serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t serialHandle)
  Prepares to enter low power consumption.
- serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t serialHandle)
  Restores from low power consumption.
- static bool SerialManager_needPollingIsr (void)
  Check if need polling ISR.

52.2 Data Structure Documentation

52.2.1 struct serial_manager_config_t

Data Fields

- uint8_t * ringBuffer
  Ring buffer address, it is used to buffer data received by the hardware.
- uint32_t ringBufferSize
  The size of the ring buffer.
- serial_port_type_t type
  Serial port type.
- serial_manager_type_t blockType
  Serial manager port type.
- void * portConfig
  Serial port configuration.

Field Documentation

(1) uint8_t* serial_manager_config_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

52.2.2 struct serial_manager_callback_message_t

Data Fields

- uint8_t * buffer
  Transferred buffer.
- uint32_t length
  Transferred data length.

52.3 Macro Definition Documentation

52.3.1 #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)

52.3.2 #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)

52.3.3 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

Macro to determine whether use common task.

52.3.4 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)

Definition of serial manager handle size.

52.3.5 #define SERIAL_MANAGER_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager handle. Then use "(serial_handle_t)name" to get the serial manager handle.

The macro should be global and could be optional. You could also define serial manager handle by yourself.

This is an example,

```
* SERIAL_MANAGER_HANDLE_DEFINE(serialManagerHandle);
*
```

Parameters

name	The name string of the serial manager handle.

52.3.6 #define SERIAL_MANAGER_WRITE_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_WRITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager write handle. Then use "(serial_write_handle_t)name" to get the serial manager write handle.

The macro should be global and could be optional. You could also define serial manager write handle by yourself.

This is an example,

```
* SERIAL_MANAGER_WRITE_HANDLE_DEFINE(serialManagerwriteHandle);
*
```

Parameters

name	The name string of the serial manager write handle.

52.3.7 #define SERIAL_MANAGER_READ_HANDLE_DEFINE(*name*) uint32_t name[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))]

This macro is used to define a 4 byte aligned serial manager read handle. Then use "(serial_read_handle_t)name" to get the serial manager read handle.

The macro should be global and could be optional. You could also define serial manager read handle by yourself.

This is an example,

```
* SERIAL_MANAGER_READ_HANDLE_DEFINE(serialManagerReadHandle);
*
```

Parameters

name	The name string of the serial manager read handle.

52.3.8 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

52.3.9 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

52.4 Enumeration Type Documentation

52.4.1 enum serial_port_type_t

Enumerator

kSerialPort_None Serial port is none.
kSerialPort_Uart Serial port UART.
kSerialPort_Uart Serial port UART.
kSerialPort_UsbCdc Serial port USB CDC.
kSerialPort_Swo Serial port SWO.
kSerialPort_Virtual Serial port Virtual.
kSerialPort_Rpmsg Serial port RPMSG.

kSerialPort_UartDma Serial port UART DMA.
kSerialPort_SpiMaster Serial port SPIMASTER.
kSerialPort_SpiSlave Serial port SPISLAVE.
kSerialPort_None Serial port is none.

52.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.
kSerialManager_Blocking Blocking handle.

52.4.3 enum serial_manager_status_t

Enumerator

kStatus_SerialManager_Success Success.
kStatus_SerialManager_Error Failed.
kStatus_SerialManager_Busy Busy.
kStatus_SerialManager_Notify Ring buffer is not empty.
kStatus_SerialManager_Canceled the non-blocking request is canceled
kStatus_SerialManager_HandleConflict The handle is opened.
kStatus_SerialManager_RingBufferOverflow The ring buffer is overflowed.
kStatus_SerialManager_NotConnected The host is not connected.

52.5 Function Documentation

52.5.1 serial_manager_status_t SerialManager_Init (serial_handle_t *serialHandle*, const serial_manager_config_t ∗ *config*)

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter serialHandle is a pointer to point to a memory space of size SERIAL_MANAGER_HANDLE_SIZE allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to serial_port_type_t for serial port setting. These three types can be set by using serial_manager_config_t.

Example below shows how to use this API to configure the Serial Manager. For UART,

```
*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_uart_config_t uartConfig;
*   config.type = kSerialPort_Uart;
```

```
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   uartConfig.instance = 0;
*   uartConfig.clockRate = 24000000;
*   uartConfig.baudRate = 115200;
*   uartConfig.parityMode = kSerialManager_UartParityDisabled;
*   uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
*   uartConfig.enableRx = 1;
*   uartConfig.enableTx = 1;
*   uartConfig.enableRxRTS = 0;
*   uartConfig.enableTxCTS = 0;
*   config.portConfig = &uartConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

For USB CDC,

```
*   #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*   static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*   static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*   serial_manager_config_t config;
*   serial_port_usb_cdc_config_t usbCdcConfig;
*   config.type = kSerialPort_UsbCdc;
*   config.ringBuffer = &s_ringBuffer[0];
*   config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*   usbCdcConfig.controllerIndex =
      kSerialManager_UsbControllerKhci0;
*   config.portConfig = &usbCdcConfig;
*   SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*
```

Parameters

serialHandle	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 uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];
config	Pointer to user-defined configuration structure.

Return values

kStatus_SerialManager_Error	An error occurred.

kStatus_SerialManager_-Success	The Serial Manager module initialization succeed.

52.5.2 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

serialHandle	The serial manager module handle pointer.

Return values

kStatus_SerialManager_-Success	The serial manager de-initialization succeed.
kStatus_SerialManager_-Busy	Opened reading or writing handle is not closed.

52.5.3 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

serialHandle	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
writeHandle	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_HAN-DLE_DEFINE(writeHandle); or uint32_t writeHandle[((SERIAL_MANAGER_W-RITE_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];

Return values

kStatus_SerialManager_-Error	An error occurred.
kStatus_SerialManager_-HandleConflict	The writing handle was opened.
kStatus_SerialManager_-Success	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);
*
```

52.5.4 serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t *writeHandle*)

This function Closes a writing handle for the serial manager module.

Parameters

writeHandle	The serial manager module writing handle pointer.

Return values

kStatus_SerialManager_-Success	The writing handle is closed.

52.5.5 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

serialHandle	The serial manager module handle pointer. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices.
readHandle	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: SERIAL_MANAGER_READ_HANDLE_DEFINE(readHandle); or uint32_t readHandle[((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];

Return values

kStatus_SerialManager_-Error	An error occurred.
kStatus_SerialManager_-Success	The reading handle is opened.
kStatus_SerialManager_-Busy	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));
*
```

52.5.6 serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t *readHandle*)

This function Closes a reading for the serial manager module.

Parameters

readHandle	The serial manager module reading handle pointer.

Return values

kStatus_SerialManager_-Success	The reading handle is closed.

52.5.7 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

writeHandle	The serial manager module handle pointer.
buffer	Start address of the data to write.
length	Length of the data to write.

Return values

kStatus_SerialManager_-Success	Successfully sent all data.
kStatus_SerialManager_-Busy	Previous transmission still not finished; data not all sent yet.

kStatus_SerialManager_-Error	An error occurred.

52.5.8 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

readHandle	The serial manager module handle pointer.
buffer	Start address of the data to store the received data.
length	The length of the data to be received.

Return values

kStatus_SerialManager_-Success	Successfully received all data.
kStatus_SerialManager_-Busy	Previous transmission still not finished; data not all received yet.
kStatus_SerialManager_-Error	An error occurred.

52.5.9 serial_manager_status_t SerialManager_EnterLowpower (serial_handle_t *serialHandle*)

This function is used to prepare to enter low power consumption.

Parameters

serialHandle	The serial manager module handle pointer.

Return values

kStatus_SerialManager_-Success	Successful operation.

52.5.10 serial_manager_status_t SerialManager_ExitLowpower (serial_handle_t *serialHandle*)

This function is used to restore from low power consumption.

Parameters

serialHandle	The serial manager module handle pointer.

Return values

kStatus_SerialManager_-Success	Successful operation.

52.5.11 static bool SerialManager_needPollingIsr (void) [inline], [static]

This function is used to check if need polling ISR.

Return values

TRUE	if need polling.

52.6 Serial Port Uart

52.6.1 Overview

Macros

- #define SERIAL_PORT_UART_DMA_RECEIVE_DATA_LENGTH (64U)
  serial port uart handle size
- #define SERIAL_USE_CONFIGURE_STRUCTURE (0U)
  Enable or disable the confgure structure pointer.

Enumerations

- enum serial_port_uart_parity_mode_t {
  kSerialManager_UartParityDisabled = 0x0U,
  kSerialManager_UartParityEven = 0x2U,
  kSerialManager_UartParityOdd = 0x3U }
  serial port uart parity mode
- enum serial_port_uart_stop_bit_count_t {
  kSerialManager_UartOneStopBit = 0U,
  kSerialManager_UartTwoStopBit = 1U }
  serial port uart stop bit count

52.6.2 Enumeration Type Documentation

52.6.2.1 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

52.6.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

52.7 Serial Port USB

52.7.1 Overview

Modules

- USB Device Configuration

Data Structures

- struct serial_port_usb_cdc_config_t
  serial port usb config struct More...

Macros

- #define SERIAL_PORT_USB_CDC_HANDLE_SIZE (72U)
  serial port usb handle size
- #define USB_DEVICE_INTERRUPT_PRIORITY (3U)
  USB interrupt priority.

Enumerations

- enum serial_port_usb_cdc_controller_index_t {
  kSerialManager_UsbControllerKhci0 = 0U,
  kSerialManager_UsbControllerKhci1 = 1U,
  kSerialManager_UsbControllerEhci0 = 2U,
  kSerialManager_UsbControllerEhci1 = 3U,
  kSerialManager_UsbControllerLpcIp3511Fs0 = 4U,
  kSerialManager_UsbControllerLpcIp3511Fs1 = 5U,
  kSerialManager_UsbControllerLpcIp3511Hs0 = 6U,
  kSerialManager_UsbControllerLpcIp3511Hs1 = 7U,
  kSerialManager_UsbControllerOhci0 = 8U,
  kSerialManager_UsbControllerOhci1 = 9U,
  kSerialManager_UsbControllerIp3516Hs0 = 10U,
  kSerialManager_UsbControllerIp3516Hs1 = 11U }
  USB controller ID.

52.7.2 Data Structure Documentation

52.7.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- serial_port_usb_cdc_controller_index_t controllerIndex
  controller index

52.7.3 Enumeration Type Documentation

52.7.3.1 enum serial_port_usb_cdc_controller_index_t

Enumerator

kSerialManager_UsbControllerKhci0 KHCI 0U.

kSerialManager_UsbControllerKhci1 KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerEhci0 EHCI 0U.

kSerialManager_UsbControllerEhci1 EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerLpcIp3511Fs0 LPC USB IP3511 FS controller 0.

kSerialManager_UsbControllerLpcIp3511Fs1 LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerLpcIp3511Hs0 LPC USB IP3511 HS controller 0.

kSerialManager_UsbControllerLpcIp3511Hs1 LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerOhci0 OHCI 0U.

kSerialManager_UsbControllerOhci1 OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

kSerialManager_UsbControllerIp3516Hs0 IP3516HS 0U.

kSerialManager_UsbControllerIp3516Hs1 IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

52.7.4 USB Device Configuration

52.8 Serial Port SWO

52.8.1 Overview

Data Structures

- struct serial_port_swo_config_t
  serial port swo config struct More...

Macros

- #define SERIAL_PORT_SWO_HANDLE_SIZE (12U)
  serial port swo handle size

Enumerations

- enum serial_port_swo_protocol_t {
  kSerialManager_SwoProtocolManchester = 1U,
  kSerialManager_SwoProtocolNrz = 2U }
  serial port swo protocol

52.8.2 Data Structure Documentation

52.8.2.1 struct serial_port_swo_config_t

Data Fields

- uint32_t clockRate
  clock rate
- uint32_t baudRate
  baud rate
- uint32_t port
  Port used to transfer data.
- serial_port_swo_protocol_t protocol
  SWO protocol.

52.8.3 Enumeration Type Documentation

52.8.3.1 enum serial_port_swo_protocol_t

Enumerator

kSerialManager_SwoProtocolManchester SWO Manchester protocol.
kSerialManager_SwoProtocolNrz SWO UART/NRZ protocol.

Chapter 53
I2s_dma_driver

53.1 Overview

Data Structures

- struct i2s_dma_handle_t
  i2s dma handle More...

Typedefs

- typedef void(∗ i2s_dma_transfer_callback_t)(I2S_Type ∗base, i2s_dma_handle_t ∗handle, status_t completionStatus, void ∗userData)
  Callback function invoked from DMA API on completion.

Driver version

- #define FSL_I2S_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))
  I2S DMA driver version 2.3.0.

DMA API

- void I2S_TxTransferCreateHandleDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, dma_handle_t ∗dmaHandle, i2s_dma_transfer_callback_t callback, void ∗userData)
  Initializes handle for transfer of audio data.
- status_t I2S_TxTransferSendDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, i2s_transfer_t transfer)
  Begins or queue sending of the given data.
- void I2S_TransferAbortDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle)
  Aborts transfer of data.
- void I2S_RxTransferCreateHandleDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, dma_handle_t ∗dmaHandle, i2s_dma_transfer_callback_t callback, void ∗userData)
  Initializes handle for reception of audio data.
- status_t I2S_RxTransferReceiveDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, i2s_transfer_t transfer)
  Begins or queue reception of data into given buffer.
- void I2S_DMACallback (dma_handle_t ∗handle, void ∗userData, bool transferDone, uint32_t tcds)
  Invoked from DMA interrupt handler.
- void I2S_TransferInstallLoopDMADescriptorMemory (i2s_dma_handle_t ∗handle, void ∗dmaDescriptorAddr, size_t dmaDescriptorNum)
  Install DMA descriptor memory for loop transfer only.
- status_t I2S_TransferSendLoopDMA (I2S_Type ∗base, i2s_dma_handle_t ∗handle, i2s_transfer_t ∗xfer, uint32_t loopTransferCount)
  Send link transfer data using DMA.

- status_t I2S_TransferReceiveLoopDMA (I2S_Type *base, i2s_dma_handle_t *handle, i2s_transfer_t *xfer, uint32_t loopTransferCount)
  Receive link transfer data using DMA.

53.2 Data Structure Documentation

53.2.1 struct _i2s_dma_handle

Members not to be accessed / modified outside of the driver.

Data Fields

- uint32_t state
  Internal state of I2S DMA transfer.
- uint8_t bytesPerFrame
  bytes per frame
- i2s_dma_transfer_callback_t completionCallback
  Callback function pointer.
- void * userData
  Application data passed to callback.
- dma_handle_t * dmaHandle
  DMA handle.
- volatile i2s_transfer_t i2sQueue [I2S_NUM_BUFFERS]
  Transfer queue storing transfer buffers.
- volatile uint8_t queueUser
  Queue index where user's next transfer will be stored.
- volatile uint8_t queueDriver
  Queue index of buffer actually used by the driver.
- dma_descriptor_t * i2sLoopDMADescriptor
  descriptor pool pointer
- size_t i2sLoopDMADescriptorNum
  number of descriptor in descriptors pool

53.3 Macro Definition Documentation

53.3.1 #define FSL_I2S_DMA_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

53.4 Typedef Documentation

53.4.1 typedef void(* i2s_dma_transfer_callback_t)(I2S_Type *base, i2s_dma_handle_t *handle, status_t completionStatus, void *userData)

Parameters

base	I2S base pointer.
handle	pointer to I2S transaction.
completion-Status	status of the transaction.
userData	optional pointer to user arguments data.

53.5 Function Documentation

53.5.1 void I2S_TxTransferCreateHandleDMA (I2S_Type ∗ *base,* i2s_dma_handle_t ∗ *handle,* dma_handle_t ∗ *dmaHandle,* i2s_dma_transfer_callback_t *callback,* void ∗ *userData*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
dmaHandle	pointer to dma handle structure.
callback	function to be called back when transfer is done or fails.
userData	pointer to data passed to callback.

53.5.2 status_t I2S_TxTransferSendDMA (I2S_Type ∗ *base,* i2s_dma_handle_t ∗ *handle,* i2s_transfer_t *transfer*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
transfer	data buffer.

Return values

kStatus_Success	

kStatus_I2S_Busy	if all queue slots are occupied with unsent buffers.

53.5.3 void I2S_TransferAbortDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.

53.5.4 void I2S_RxTransferCreateHandleDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, dma_handle_t * *dmaHandle*, i2s_dma_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
dmaHandle	pointer to dma handle structure.
callback	function to be called back when transfer is done or fails.
userData	pointer to data passed to callback.

53.5.5 status_t I2S_RxTransferReceiveDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, i2s_transfer_t *transfer*)

Parameters

base	I2S base pointer.
handle	pointer to handle structure.
transfer	data buffer.

Return values

kStatus_Success	
kStatus_I2S_Busy	if all queue slots are occupied with buffers which are not full.

53.5.6 void I2S_DMACallback (dma_handle_t * *handle,* void * *userData,* bool *transferDone,* uint32_t *tcds*)

Parameters

handle	pointer to DMA handle structure.
userData	argument for user callback.
transferDone	if transfer was done.
tcds	

53.5.7 void I2S_TransferInstallLoopDMADescriptorMemory (i2s_dma_handle_t * *handle,* void * *dmaDescriptorAddr,* size_t *dmaDescriptorNum*)

This function used to register DMA descriptor memory for the i2s loop dma transfer.

It must be callbed before I2S_TransferSendLoopDMA/I2S_TransferReceiveLoopDMA and after I2S_RxTransferCreateHandleDMA/I2S_TxTransferCreateHandleDMA.

User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

Parameters

handle	Pointer to i2s DMA transfer handle.
dmaDescriptorAddr	DMA descriptor start address.
dmaDescriptorNum	DMA descriptor number.

53.5.8 status_t I2S_TransferSendLoopDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, i2s_transfer_t * *xfer*, uint32_t *loopTransferCount*)

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S_InstallDMADescriptorMemory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S_TransferAbortDMA to stop the loop transfer.

Parameters

base	I2S peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	I2S DMA transfer structure. See i2s_transfer_t.
loopTransferCount	loop count

Return values

kStatus_Success	

53.5.9 status_t I2S_TransferReceiveLoopDMA (I2S_Type * *base*, i2s_dma_handle_t * *handle*, i2s_transfer_t * *xfer*, uint32_t *loopTransferCount*)

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

This function support loop transfer, such as A->B->...->A, the loop transfer chain will be converted into a chain of descriptor and submit to dma. Application must be aware of that the more counts of the loop transfer, then more DMA descriptor memory required, user can use function I2S_InstallDMADescriptorMemory to register the dma descriptor memory.

As the DMA support maximum 1024 transfer count, so application must be aware of that this transfer function support maximum 1024 samples in each transfer, otherwise assert error or error status will be returned. Once the loop transfer start, application can use function I2S_TransferAbortDMA to stop the loop transfer.

Parameters

base	I2S peripheral base address.
handle	Pointer to usart_dma_handle_t structure.
xfer	I2S DMA transfer structure. See i2s_transfer_t.
loopTransfer-Count	loop count

Return values

kStatus_Success	

Chapter 54
Hashcrypt_driver

54.1 Overview

Macros

- #define HASHCRYPT_MODE_SHA1 0x1
  Algorithm definitions correspond with the values for Mode field in Control register !

Enumerations

- enum hashcrypt_algo_t {
  kHASHCRYPT_Sha1 = HASHCRYPT_MODE_SHA1,
  kHASHCRYPT_Sha256 = HASHCRYPT_MODE_SHA256,
  kHASHCRYPT_Aes = HASHCRYPT_MODE_AES }
  Algorithm used for Hashcrypt operation.

Functions

- void HASHCRYPT_Init (HASHCRYPT_Type ∗base)
  Enables clock and disables reset for HASHCRYPT peripheral.
- void HASHCRYPT_Deinit (HASHCRYPT_Type ∗base)
  Disables clock for HASHCRYPT peripheral.

Driver version

- #define FSL_HASHCRYPT_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))
  HASHCRYPT driver version.

54.2 Macro Definition Documentation

54.2.1 #define FSL_HASHCRYPT_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))

Version 2.2.3.

Current version: 2.2.3

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Support loading AES key from unaligned address
- Version 2.0.2
  - Support loading AES key from unaligned address for different compiler and core variants

- Version 2.0.3
  - Remove SHA512 and AES ICB algorithm definitions
- Version 2.0.4
  - Add SHA context switch support
- Version 2.1.0
  - Update the register name and macro to align with new header.
- Version 2.1.1
  - Fix MISRA C-2012.
- Version 2.1.2
  - Support loading AES input data from unaligned address.
- Version 2.1.3
  - Fix MISRA C-2012.
- Version 2.1.4
  - Fix context switch cannot work when switching from AES.
- Version 2.1.5
  - Add data synchronization barriere inside hashcrypt_sha_ldm_stm_16_words() to prevent possible optimization issue.
- Version 2.2.0
  - Add AES-OFB and AES-CFB mixed IP/SW modes.
- Version 2.2.1
  - Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() prevent compiler from reordering memory write when -O2 or higher is used.
- Version 2.2.2
  - Add data synchronization barrier inside hashcrypt_sha_ldm_stm_16_words() to fix optimization issue
- Version 2.2.3
  - Added check for size in hashcrypt_aes_one_block to prevent overflowing COUNT field in MEMCTRL register, if its bigger than COUNT field do a multiple runs.

54.3 Enumeration Type Documentation

54.3.1 enum hashcrypt_algo_t

Enumerator

kHASHCRYPT_Sha1 SHA_1.
kHASHCRYPT_Sha256 SHA_256.
kHASHCRYPT_Aes AES.

54.4 Function Documentation

54.4.1 void HASHCRYPT_Init (HASHCRYPT_Type ∗ *base*)

Enable clock and disable reset for HASHCRYPT.

Parameters

base	HASHCRYPT base address

54.4.2 void HASHCRYPT_Deinit (HASHCRYPT_Type ∗ *base*)

Disable clock and enable reset.

Parameters

base	HASHCRYPT base address

Chapter 55
I3c_common_driver

55.1 Overview

Data Structures

- struct i3c_config_t
  Structure with settings to initialize the I3C module, could both initialize master and slave functionality. More...

Functions

- uint32_t I3C_GetInstance (I3C_Type *base)
  Get which instance current I3C is used.
- void I3C_GetDefaultConfig (i3c_config_t *config)
  Provides a default configuration for the I3C peripheral, the configuration covers both master functionality and slave functionality.
- void I3C_Init (I3C_Type *base, const i3c_config_t *config, uint32_t sourceClock_Hz)
  Initializes the I3C peripheral.

55.2 Data Structure Documentation

55.2.1 struct i3c_config_t

This structure holds configuration settings for the I3C peripheral. To initialize this structure to reasonable defaults, call the I3C_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

Data Fields

- i3c_master_enable_t enableMaster
  Enable master mode.
- bool disableTimeout
  Whether to disable timeout to prevent the ERRWARN.
- i3c_master_hkeep_t hKeep
  High keeper mode setting.
- bool enableOpenDrainStop
  Whether to emit open-drain speed STOP.
- bool enableOpenDrainHigh
  Enable Open-Drain High to be 1 PPBAUD count for i3c messages, or 1 ODBAUD.
- i3c_baudrate_hz_t baudRate_Hz
  Desired baud rate settings.

- uint8_t masterDynamicAddress
  Main master dynamic address configuration.
- uint32_t slowClock_Hz
  Slow clock frequency for time control.
- uint32_t maxWriteLength
  Maximum write length.
- uint32_t maxReadLength
  Maximum read length.
- bool enableSlave
  Whether to enable slave.
- uint8_t staticAddr
  Static address.
- uint16_t vendorID
  Device vendor ID(manufacture ID).
- bool enableRandomPart
  Whether to generate random part number, if using random part number, the partNumber variable setting is meaningless.
- uint32_t partNumber
  Device part number info.
- uint8_t dcr
  Device characteristics register information.
- uint8_t bcr
  Bus characteristics register information.
- uint8_t hdrMode
  Support hdr mode, could be OR logic in enumeration:i3c_hdr_mode_t.
- bool nakAllRequest
  Whether to reply NAK to all requests except broadcast CCC.
- bool ignoreS0S1Error
  Whether to ignore S0/S1 error in SDR mode.
- bool offline
  Whether to wait 60 us of bus quiet or HDR request to ensure slave track SDR mode safely.
- bool matchSlaveStartStop
  Whether to assert start/stop status only the time slave is addressed.

Field Documentation

(1) i3c_master_enable_t i3c_config_t::enableMaster

(2) bool i3c_config_t::disableTimeout

(3) i3c_master_hkeep_t i3c_config_t::hKeep

(4) bool i3c_config_t::enableOpenDrainStop

(5) bool i3c_config_t::enableOpenDrainHigh

(6) i3c_baudrate_hz_t i3c_config_t::baudRate_Hz

(7) uint8_t i3c_config_t::masterDynamicAddress

(8) uint32_t i3c_config_t::slowClock_Hz

(9) uint32_t i3c_config_t::maxWriteLength

(10) uint32_t i3c_config_t::maxReadLength

(11) bool i3c_config_t::enableSlave

(12) uint8_t i3c_config_t::staticAddr

(13) uint16_t i3c_config_t::vendorID

(14) bool i3c_config_t::enableRandomPart

(15) uint8_t i3c_config_t::dcr

(16) uint8_t i3c_config_t::bcr

(17) uint8_t i3c_config_t::hdrMode

(18) bool i3c_config_t::nakAllRequest

(19) bool i3c_config_t::ignoreS0S1Error

(20) bool i3c_config_t::offline

(21) bool i3c_config_t::matchSlaveStartStop

55.3 Function Documentation

55.3.1 uint32_t I3C_GetInstance (I3C_Type * *base*)

Parameters

base	The I3C peripheral base address.

55.3.2 void I3C_GetDefaultConfig (i3c_config_t * *config*)

This function provides the following default configuration for I3C:

```
*    config->enableMaster                = kI3C_MasterCapable;
*    config->disableTimeout              = false;
*    config->hKeep                       = kI3C_MasterHighKeeperNone;
*    config->enableOpenDrainStop         = true;
*    config->enableOpenDrainHigh         = true;
*    config->baudRate_Hz.i2cBaud         = 400000U;
*    config->baudRate_Hz.i3cPushPullBaud  = 12500000U;
*    config->baudRate_Hz.i3cOpenDrainBaud = 2500000U;
*    config->masterDynamicAddress        = 0x0AU;
*    config->slowClock_Hz                = 1000000U;
*    config->enableSlave                 = true;
*    config->vendorID                    = 0x11BU;
*    config->enableRandomPart            = false;
```

```
*    config->partNumber                    = 0;
*    config->dcr                           = 0;
*    config->bcr = 0;
*    config->hdrMode              = (uint8_t)kI3C_HDRModeDDR;
*    config->nakAllRequest        = false;
*    config->ignoreS0S1Error      = false;
*    config->offline              = false;
*    config->matchSlaveStartStop = false;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the common I3C driver with I3C_Init().

Parameters

out	*config*	User provided configuration structure for default values. Refer to i3c_config_t.

55.3.3 void I3C_Init (I3C_Type * *base*, const i3c_config_t * *config*, uint32_t *sourceClock_Hz*)

This function enables the peripheral clock and initializes the I3C peripheral as described by the user provided configuration. This will initialize both the master peripheral and slave peripheral so that I3C module could work as pure master, pure slave or secondary master, etc. A software reset is performed prior to configuration.

Parameters

base	The I3C peripheral base address.
config	User provided peripheral configuration. Use I3C_GetDefaultConfig() to get a set of defaults that you can override.
sourceClock_Hz	Frequency in Hertz of the I3C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.

55.3.4 CODEC Adapter

55.3.4.1 Overview

Enumerations

- enum {
  kCODEC_WM8904,
  kCODEC_WM8960,
  kCODEC_WM8524,
  kCODEC_SGTL5000,
  kCODEC_DA7212,
  kCODEC_CS42888,
  kCODEC_CS42448,
  kCODEC_AK4497,
  kCODEC_AK4458,
  kCODEC_TFA9XXX,
  kCODEC_TFA9896 }
  codec type

55.3.4.2 Enumeration Type Documentation

55.3.4.2.1 anonymous enum

Enumerator

kCODEC_WM8904 wm8904
kCODEC_WM8960 wm8960
kCODEC_WM8524 wm8524
kCODEC_SGTL5000 sgtl5000
kCODEC_DA7212 da7212
kCODEC_CS42888 CS42888.
kCODEC_CS42448 CS42448.
kCODEC_AK4497 AK4497.
kCODEC_AK4458 ak4458
kCODEC_TFA9XXX tfa9xxx
kCODEC_TFA9896 tfa9896

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.
© 2021 NXP B.V.