

---

# Kinetis SDK v.2.0 API Reference Manual

**NXP Semiconductors**

Document Number: KSDKL4K20APIRM

Rev. 0

Jun 2016





# Contents

Chapter [Introduction](#)

Chapter [Driver errors status](#)

Chapter [Architectural Overview](#)

Chapter [Trademarks](#)

Chapter [ADC16: 16-bit SAR Analog-to-Digital Converter Driver](#)

|            |  |           |
|------------|--|-----------|
| <b>5.1</b> | <b>Overview</b>                        | <b>11</b> |
| <b>5.2</b> | <b>Typical use case</b>                | <b>11</b> |
| 5.2.1      | Polling Configuration                  | 11        |
| 5.2.2      | Interrupt Configuration                | 11        |
| <b>5.3</b> | <b>Data Structure Documentation</b>    | <b>14</b> |
| 5.3.1      | struct adc16_config_t                  | 14        |
| 5.3.2      | struct adc16_hardware_compare_config_t | 15        |
| 5.3.3      | struct adc16_channel_config_t          | 16        |
| <b>5.4</b> | <b>Macro Definition Documentation</b>  | <b>16</b> |
| 5.4.1      | FSL_ADC16_DRIVER_VERSION               | 16        |
| <b>5.5</b> | <b>Enumeration Type Documentation</b>  | <b>16</b> |
| 5.5.1      | _adc16_channel_status_flags            | 16        |
| 5.5.2      | _adc16_status_flags                    | 16        |
| 5.5.3      | adc16_clock_divider_t                  | 16        |
| 5.5.4      | adc16_resolution_t                     | 17        |
| 5.5.5      | adc16_clock_source_t                   | 17        |
| 5.5.6      | adc16_long_sample_mode_t               | 17        |
| 5.5.7      | adc16_reference_voltage_source_t       | 17        |
| 5.5.8      | adc16_hardware_compare_mode_t          | 18        |
| <b>5.6</b> | <b>Function Documentation</b>          | <b>18</b> |
| 5.6.1      | ADC16_Init                             | 18        |
| 5.6.2      | ADC16_Deinit                           | 18        |
| 5.6.3      | ADC16_GetDefaultConfig                 | 18        |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 5.6.4          | ADC16_EnableHardwareTrigger . . . . .           | 19          |
| 5.6.5          | ADC16_SetHardwareCompareConfig . . . . .        | 19          |
| 5.6.6          | ADC16_GetStatusFlags . . . . .                  | 19          |
| 5.6.7          | ADC16_ClearStatusFlags . . . . .                | 19          |
| 5.6.8          | ADC16_SetChannelConfig . . . . .                | 20          |
| 5.6.9          | ADC16_GetChannelConversionValue . . . . .       | 20          |
| 5.6.10         | ADC16_GetChannelStatusFlags . . . . .           | 21          |
| <b>Chapter</b> | <b>Clock Driver</b>                             |             |
| <b>6.1</b>     | <b>Overview . . . . .</b>                       | <b>23</b>   |
| <b>6.2</b>     | <b>Get frequency . . . . .</b>                  | <b>23</b>   |
| <b>6.3</b>     | <b>External clock frequency . . . . .</b>       | <b>23</b>   |
| <b>6.4</b>     | <b>Data Structure Documentation . . . . .</b>   | <b>30</b>   |
| 6.4.1          | struct sim_clock_config_t . . . . .             | 30          |
| 6.4.2          | struct oscr_config_t . . . . .                  | 31          |
| 6.4.3          | struct osc_config_t . . . . .                   | 31          |
| 6.4.4          | struct mcg_pll_config_t . . . . .               | 32          |
| 6.4.5          | struct mcg_config_t . . . . .                   | 32          |
| <b>6.5</b>     | <b>Macro Definition Documentation . . . . .</b> | <b>33</b>   |
| 6.5.1          | FSL_CLOCK_DRIVER_VERSION . . . . .              | 33          |
| 6.5.2          | DMAMUX_CLOCKS . . . . .                         | 33          |
| 6.5.3          | RTC_CLOCKS . . . . .                            | 33          |
| 6.5.4          | SAI_CLOCKS . . . . .                            | 34          |
| 6.5.5          | SPI_CLOCKS . . . . .                            | 34          |
| 6.5.6          | SLCD_CLOCKS . . . . .                           | 34          |
| 6.5.7          | PIT_CLOCKS . . . . .                            | 34          |
| 6.5.8          | PORT_CLOCKS . . . . .                           | 34          |
| 6.5.9          | TSI_CLOCKS . . . . .                            | 35          |
| 6.5.10         | DAC_CLOCKS . . . . .                            | 35          |
| 6.5.11         | LPTMR_CLOCKS . . . . .                          | 35          |
| 6.5.12         | ADC16_CLOCKS . . . . .                          | 35          |
| 6.5.13         | DMA_CLOCKS . . . . .                            | 35          |
| 6.5.14         | UART0_CLOCKS . . . . .                          | 36          |
| 6.5.15         | UART_CLOCKS . . . . .                           | 36          |
| 6.5.16         | TPM_CLOCKS . . . . .                            | 36          |
| 6.5.17         | I2C_CLOCKS . . . . .                            | 36          |
| 6.5.18         | FTF_CLOCKS . . . . .                            | 36          |
| 6.5.19         | CMP_CLOCKS . . . . .                            | 37          |
| 6.5.20         | SYS_CLK . . . . .                               | 37          |

# Contents

| Section Number | Title                                 | Page Number |
|----------------|---------------------------------------|-------------|
| <b>6.6</b>     | <b>Enumeration Type Documentation</b> | <b>37</b>   |
| 6.6.1          | clock_name_t                          | 37          |
| 6.6.2          | clock_usb_src_t                       | 37          |
| 6.6.3          | clock_ip_name_t                       | 37          |
| 6.6.4          | osc_mode_t                            | 37          |
| 6.6.5          | _osc_cap_load                         | 38          |
| 6.6.6          | _oscer_enable_mode                    | 38          |
| 6.6.7          | mcg_fll_src_t                         | 38          |
| 6.6.8          | mcg_irc_mode_t                        | 38          |
| 6.6.9          | mcg_dmx32_t                           | 38          |
| 6.6.10         | mcg_drs_t                             | 39          |
| 6.6.11         | mcg_pll_ref_src_t                     | 39          |
| 6.6.12         | mcg_clkout_src_t                      | 39          |
| 6.6.13         | mcg_atm_select_t                      | 39          |
| 6.6.14         | mcg_oscsel_t                          | 39          |
| 6.6.15         | mcg_pll_clk_select_t                  | 40          |
| 6.6.16         | mcg_monitor_mode_t                    | 40          |
| 6.6.17         | _mcg_status                           | 40          |
| 6.6.18         | _mcg_status_flags_t                   | 40          |
| 6.6.19         | _mcg_ircclk_enable_mode               | 40          |
| 6.6.20         | _mcg_pll_enable_mode                  | 41          |
| 6.6.21         | mcg_mode_t                            | 41          |
| <b>6.7</b>     | <b>Function Documentation</b>         | <b>41</b>   |
| 6.7.1          | CLOCK_EnableClock                     | 41          |
| 6.7.2          | CLOCK_DisableClock                    | 41          |
| 6.7.3          | CLOCK_SetEr32kClock                   | 42          |
| 6.7.4          | CLOCK_SetTpmClock                     | 42          |
| 6.7.5          | CLOCK_SetLpsci0Clock                  | 42          |
| 6.7.6          | CLOCK_SetPllFllSelClock               | 42          |
| 6.7.7          | CLOCK_SetClkOutClock                  | 42          |
| 6.7.8          | CLOCK_SetRtcClkOutClock               | 43          |
| 6.7.9          | CLOCK_EnableUsbfs0Clock               | 43          |
| 6.7.10         | CLOCK_DisableUsbfs0Clock              | 43          |
| 6.7.11         | CLOCK_SetOutDiv                       | 43          |
| 6.7.12         | CLOCK_GetFreq                         | 44          |
| 6.7.13         | CLOCK_GetCoreSysClkFreq               | 44          |
| 6.7.14         | CLOCK_GetPlatClkFreq                  | 44          |
| 6.7.15         | CLOCK_GetBusClkFreq                   | 44          |
| 6.7.16         | CLOCK_GetFlashClkFreq                 | 44          |
| 6.7.17         | CLOCK_GetPllFllSelClkFreq             | 45          |
| 6.7.18         | CLOCK_GetEr32kClkFreq                 | 45          |
| 6.7.19         | CLOCK_GetOsc0ErClkFreq                | 45          |
| 6.7.20         | CLOCK_SetSimConfig                    | 45          |
| 6.7.21         | CLOCK_SetSimSafeDivs                  | 45          |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 6.7.22         | CLOCK_GetOutClkFreq . . . . .                       | 46          |
| 6.7.23         | CLOCK_GetFllFreq . . . . .                          | 46          |
| 6.7.24         | CLOCK_GetInternalRefClkFreq . . . . .               | 46          |
| 6.7.25         | CLOCK_GetFixedFreqClkFreq . . . . .                 | 46          |
| 6.7.26         | CLOCK_GetPll0Freq . . . . .                         | 47          |
| 6.7.27         | CLOCK_SetLowPowerEnable . . . . .                   | 47          |
| 6.7.28         | CLOCK_SetInternalRefClkConfig . . . . .             | 47          |
| 6.7.29         | CLOCK_SetExternalRefClkConfig . . . . .             | 48          |
| 6.7.30         | CLOCK_EnablePll0 . . . . .                          | 48          |
| 6.7.31         | CLOCK_DisablePll0 . . . . .                         | 48          |
| 6.7.32         | CLOCK_CalcPllDiv . . . . .                          | 48          |
| 6.7.33         | CLOCK_SetOsc0MonitorMode . . . . .                  | 50          |
| 6.7.34         | CLOCK_SetPll0MonitorMode . . . . .                  | 50          |
| 6.7.35         | CLOCK_GetStatusFlags . . . . .                      | 50          |
| 6.7.36         | CLOCK_ClearStatusFlags . . . . .                    | 51          |
| 6.7.37         | OSC_SetExtRefClkConfig . . . . .                    | 51          |
| 6.7.38         | OSC_SetCapLoad . . . . .                            | 52          |
| 6.7.39         | CLOCK_InitOsc0 . . . . .                            | 52          |
| 6.7.40         | CLOCK_DeinitOsc0 . . . . .                          | 52          |
| 6.7.41         | CLOCK_SetXtal0Freq . . . . .                        | 52          |
| 6.7.42         | CLOCK_SetXtal32Freq . . . . .                       | 53          |
| 6.7.43         | CLOCK_TrimInternalRefClk . . . . .                  | 53          |
| 6.7.44         | CLOCK_GetMode . . . . .                             | 54          |
| 6.7.45         | CLOCK_SetFeiMode . . . . .                          | 54          |
| 6.7.46         | CLOCK_SetFeeMode . . . . .                          | 54          |
| 6.7.47         | CLOCK_SetFbiMode . . . . .                          | 55          |
| 6.7.48         | CLOCK_SetFbeMode . . . . .                          | 56          |
| 6.7.49         | CLOCK_SetBlpiMode . . . . .                         | 57          |
| 6.7.50         | CLOCK_SetBlpeMode . . . . .                         | 57          |
| 6.7.51         | CLOCK_SetPbeMode . . . . .                          | 58          |
| 6.7.52         | CLOCK_SetPeeMode . . . . .                          | 58          |
| 6.7.53         | CLOCK_ExternalModeToFbeModeQuick . . . . .          | 59          |
| 6.7.54         | CLOCK_InternalModeToFbiModeQuick . . . . .          | 59          |
| 6.7.55         | CLOCK_BootToFeiMode . . . . .                       | 60          |
| 6.7.56         | CLOCK_BootToFeeMode . . . . .                       | 60          |
| 6.7.57         | CLOCK_BootToBlpiMode . . . . .                      | 61          |
| 6.7.58         | CLOCK_BootToBlpeMode . . . . .                      | 61          |
| 6.7.59         | CLOCK_BootToPeeMode . . . . .                       | 62          |
| 6.7.60         | CLOCK_SetMcgConfig . . . . .                        | 62          |
| <b>6.8</b>     | <b>Variable Documentation . . . . .</b>             | <b>63</b>   |
| 6.8.1          | g_xtal0Freq . . . . .                               | 63          |
| 6.8.2          | g_xtal32Freq . . . . .                              | 63          |
| <b>6.9</b>     | <b>Multipurpose Clock Generator (MCG) . . . . .</b> | <b>64</b>   |

| Contents                                     |   |             |
|--|---|-------------|
| Section Number                               | Title   | Page Number |
| 6.9.1  | Function description . . . . .                  | 64          |
| 6.9.2  | Typical use case . . . . .                      | 66          |
| <b>Chapter CMP: Analog Comparator Driver</b> |   |             |
| <b>7.1</b>                                   | <b>Overview . . . . .</b>                       | <b>71</b>   |
| <b>7.2</b>                                   | <b>Typical use case . . . . .</b>               | <b>71</b>   |
| 7.2.1  | Polling Configuration . . . . .                 | 71          |
| 7.2.2  | Interrupt Configuration . . . . .               | 71          |
| <b>7.3</b>                                   | <b>Data Structure Documentation . . . . .</b>   | <b>74</b>   |
| 7.3.1  | struct cmp_config_t . . . . .                   | 74          |
| 7.3.2  | struct cmp_filter_config_t . . . . .            | 74          |
| 7.3.3  | struct cmp_dac_config_t . . . . .               | 75          |
| <b>7.4</b>                                   | <b>Macro Definition Documentation . . . . .</b> | <b>75</b>   |
| 7.4.1  | FSL_CMP_DRIVER_VERSION . . . . .                | 75          |
| <b>7.5</b>                                   | <b>Enumeration Type Documentation . . . . .</b> | <b>75</b>   |
| 7.5.1  | _cmp_interrupt_enable . . . . .                 | 75          |
| 7.5.2  | _cmp_status_flags . . . . .                     | 75          |
| 7.5.3  | cmp_hysteresis_mode_t . . . . .                 | 76          |
| 7.5.4  | cmp_reference_voltage_source_t . . . . .        | 76          |
| <b>7.6</b>                                   | <b>Function Documentation . . . . .</b>         | <b>76</b>   |
| 7.6.1  | CMP_Init . . . . .                              | 76          |
| 7.6.2  | CMP_Deinit . . . . .                            | 76          |
| 7.6.3  | CMP_Enable . . . . .                            | 78          |
| 7.6.4  | CMP_GetDefaultConfig . . . . .                  | 78          |
| 7.6.5  | CMP_SetInputChannels . . . . .                  | 78          |
| 7.6.6  | CMP_SetFilterConfig . . . . .                   | 79          |
| 7.6.7  | CMP_SetDACConfig . . . . .                      | 79          |
| 7.6.8  | CMP_EnableInterrupts . . . . .                  | 79          |
| 7.6.9  | CMP_DisableInterrupts . . . . .                 | 79          |
| 7.6.10                                       | CMP_GetStatusFlags . . . . .                    | 80          |
| 7.6.11                                       | CMP_ClearStatusFlags . . . . .                  | 80          |
| <b>Chapter COP: Watchdog Driver</b>          |   |             |
| <b>8.1</b>                                   | <b>Overview . . . . .</b>                       | <b>81</b>   |
| <b>8.2</b>                                   | <b>Typical use case . . . . .</b>               | <b>81</b>   |
| <b>8.3</b>                                   | <b>Data Structure Documentation . . . . .</b>   | <b>82</b>   |
| 8.3.1  | struct cop_config_t . . . . .                   | 82          |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| <b>8.4</b>     | <b>Macro Definition Documentation</b>                       | <b>82</b>   |
| 8.4.1          | FSL_COP_DRIVER_VERSION                                      | 82          |
| <b>8.5</b>     | <b>Enumeration Type Documentation</b>                       | <b>82</b>   |
| 8.5.1          | cop_clock_source_t  | 82          |
| 8.5.2          | cop_timeout_cycles_t  | 82          |
| <b>8.6</b>     | <b>Function Documentation</b>                               | <b>83</b>   |
| 8.6.1          | COP_GetDefaultConfig  | 83          |
| 8.6.2          | COP_Init  | 83          |
| 8.6.3          | COP_Disable   | 83          |
| 8.6.4          | COP_Refresh   | 85          |
| <b>Chapter</b> | <b>DAC: Digital-to-Analog Converter Driver</b>              |             |
| <b>9.1</b>     | <b>Overview</b>   | <b>87</b>   |
| <b>9.2</b>     | <b>Typical use case</b>                                     | <b>87</b>   |
| 9.2.1          | Working as a basic DAC without the hardware buffer feature. | 87          |
| 9.2.2          | Working with the hardware buffer.                           | 87          |
| <b>9.3</b>     | <b>Data Structure Documentation</b>                         | <b>90</b>   |
| 9.3.1          | struct dac_config_t   | 90          |
| 9.3.2          | struct dac_buffer_config_t                                  | 90          |
| <b>9.4</b>     | <b>Macro Definition Documentation</b>                       | <b>91</b>   |
| 9.4.1          | FSL_DAC_DRIVER_VERSION                                      | 91          |
| <b>9.5</b>     | <b>Enumeration Type Documentation</b>                       | <b>91</b>   |
| 9.5.1          | _dac_buffer_status_flags                                    | 91          |
| 9.5.2          | _dac_buffer_interrupt_enable                                | 91          |
| 9.5.3          | dac_reference_voltage_source_t                              | 91          |
| 9.5.4          | dac_buffer_trigger_mode_t                                   | 91          |
| 9.5.5          | dac_buffer_work_mode_t                                      | 91          |
| <b>9.6</b>     | <b>Function Documentation</b>                               | <b>92</b>   |
| 9.6.1          | DAC_Init  | 92          |
| 9.6.2          | DAC_Deinit  | 92          |
| 9.6.3          | DAC_GetDefaultConfig  | 92          |
| 9.6.4          | DAC_Enable  | 92          |
| 9.6.5          | DAC_EnableBuffer  | 93          |
| 9.6.6          | DAC_SetBufferConfig   | 93          |
| 9.6.7          | DAC_GetDefaultBufferConfig                                  | 93          |
| 9.6.8          | DAC_EnableBufferDMA   | 93          |
| 9.6.9          | DAC_SetBufferValue  | 94          |
| 9.6.10         | DAC_DoSoftwareTriggerBuffer                                 | 94          |



# Contents

| Section Number | Title  | Page Number |
|----------------|--|-------------|
| 9.6.11         | DAC_GetBufferReadPointer . . . . .                 | 94          |
| 9.6.12         | DAC_SetBufferReadPointer . . . . .                 | 95          |
| 9.6.13         | DAC_EnableBufferInterrupts . . . . .               | 95          |
| 9.6.14         | DAC_DisableBufferInterrupts . . . . .              | 95          |
| 9.6.15         | DAC_GetBufferStatusFlags . . . . .                 | 95          |
| 9.6.16         | DAC_ClearBufferStatusFlags . . . . .               | 95          |
| <b>Chapter</b> | <b>DMA: Direct Memory Access Controller Driver</b> |             |
| <b>10.1</b>    | <b>Overview . . . . .</b>                          | <b>97</b>   |
| <b>10.2</b>    | <b>Typical use case . . . . .</b>                  | <b>97</b>   |
| 10.2.1         | DMA Operation . . . . .                            | 97          |
| <b>10.3</b>    | <b>Data Structure Documentation . . . . .</b>      | <b>100</b>  |
| 10.3.1         | struct dma_transfer_config_t . . . . .             | 100         |
| 10.3.2         | struct dma_channel_link_config_t . . . . .         | 101         |
| 10.3.3         | struct dma_handle_t . . . . .                      | 101         |
| <b>10.4</b>    | <b>Macro Definition Documentation . . . . .</b>    | <b>102</b>  |
| 10.4.1         | FSL_DMA_DRIVER_VERSION . . . . .                   | 102         |
| <b>10.5</b>    | <b>Typedef Documentation . . . . .</b>             | <b>102</b>  |
| 10.5.1         | dma_callback . . . . .                             | 102         |
| <b>10.6</b>    | <b>Enumeration Type Documentation . . . . .</b>    | <b>102</b>  |
| 10.6.1         | _dma_channel_status_flags . . . . .                | 102         |
| 10.6.2         | dma_transfer_size_t . . . . .                      | 102         |
| 10.6.3         | dma_modulo_t . . . . .                             | 102         |
| 10.6.4         | dma_channel_link_type_t . . . . .                  | 103         |
| 10.6.5         | dma_transfer_type_t . . . . .                      | 103         |
| 10.6.6         | dma_transfer_options_t . . . . .                   | 103         |
| <b>10.7</b>    | <b>Function Documentation . . . . .</b>            | <b>104</b>  |
| 10.7.1         | DMA_Init . . . . .                                 | 104         |
| 10.7.2         | DMA_Deinit . . . . .                               | 105         |
| 10.7.3         | DMA_ResetChannel . . . . .                         | 105         |
| 10.7.4         | DMA_SetTransferConfig . . . . .                    | 105         |
| 10.7.5         | DMA_SetChannelLinkConfig . . . . .                 | 106         |
| 10.7.6         | DMA_SetSourceAddress . . . . .                     | 106         |
| 10.7.7         | DMA_SetDestinationAddress . . . . .                | 106         |
| 10.7.8         | DMA_SetTransferSize . . . . .                      | 107         |
| 10.7.9         | DMA_SetModulo . . . . .                            | 107         |
| 10.7.10        | DMA_EnableCycleSteal . . . . .                     | 107         |
| 10.7.11        | DMA_EnableAutoAlign . . . . .                      | 108         |
| 10.7.12        | DMA_EnableAsyncRequest . . . . .                   | 108         |

# Contents

| Section Number | Title                                 | Page Number |
|----------------|---------------------------------------|-------------|
| 10.7.13        | DMA_EnableInterrupts . . . . .        | 108         |
| 10.7.14        | DMA_DisableInterrupts . . . . .       | 109         |
| 10.7.15        | DMA_EnableChannelRequest . . . . .    | 109         |
| 10.7.16        | DMA_DisableChannelRequest . . . . .   | 109         |
| 10.7.17        | DMA_TriggerChannelStart . . . . .     | 109         |
| 10.7.18        | DMA_GetRemainingBytes . . . . .       | 110         |
| 10.7.19        | DMA_GetChannelStatusFlags . . . . .   | 110         |
| 10.7.20        | DMA_ClearChannelStatusFlags . . . . . | 110         |
| 10.7.21        | DMA_CreateHandle . . . . .            | 111         |
| 10.7.22        | DMA_SetCallback . . . . .             | 111         |
| 10.7.23        | DMA_PrepareTransfer . . . . .         | 111         |
| 10.7.24        | DMA_SubmitTransfer . . . . .          | 112         |
| 10.7.25        | DMA_StartTransfer . . . . .           | 112         |
| 10.7.26        | DMA_StopTransfer . . . . .            | 113         |
| 10.7.27        | DMA_AbortTransfer . . . . .           | 113         |
| 10.7.28        | DMA_HandleIRQ . . . . .               | 113         |

## Chapter DMAMUX: Direct Memory Access Multiplexer Driver

|        |  |     |
|--------|--|-----|
| 11.1   | Overview . . . . .                       | 115 |
| 11.2   | Typical use case . . . . .               | 115 |
| 11.2.1 | DMAMUX Operation . . . . .               | 115 |
| 11.3   | Macro Definition Documentation . . . . . | 115 |
| 11.3.1 | FSL_DMAMUX_DRIVER_VERSION . . . . .      | 115 |
| 11.4   | Function Documentation . . . . .         | 116 |
| 11.4.1 | DMAMUX_Init . . . . .                    | 116 |
| 11.4.2 | DMAMUX_Deinit . . . . .                  | 117 |
| 11.4.3 | DMAMUX_EnableChannel . . . . .           | 117 |
| 11.4.4 | DMAMUX_DisableChannel . . . . .          | 117 |
| 11.4.5 | DMAMUX_SetSource . . . . .               | 118 |

## Chapter C90TFS Flash Driver

|        |   |     |
|--------|---|-----|
| 12.1   | Overview . . . . .                                      | 119 |
| 12.2   | Data Structure Documentation . . . . .                  | 127 |
| 12.2.1 | struct flash_execute_in_ram_function_config_t . . . . . | 127 |
| 12.2.2 | struct flash_swap_state_config_t . . . . .              | 127 |
| 12.2.3 | struct flash_swap_ifr_field_config_t . . . . .          | 127 |
| 12.2.4 | union flash_swap_ifr_field_data_t . . . . .             | 128 |
| 12.2.5 | struct flash_operation_config_t . . . . .               | 128 |
| 12.2.6 | struct flash_config_t . . . . .                         | 129 |

# Contents

| Section Number | Title                                    | Page Number |
|----------------|--|-------------|
| <b>12.3</b>    | <b>Macro Definition Documentation</b>    | <b>130</b>  |
| 12.3.1         | MAKE_VERSION                             | 130         |
| 12.3.2         | FSL_FLASH_DRIVER_VERSION                 | 130         |
| 12.3.3         | FLASH_SSD_CONFIG_ENABLE_FLEXNVM_SUPPORT  | 130         |
| 12.3.4         | FLASH_DRIVER_IS_FLASH_RESIDENT           | 130         |
| 12.3.5         | FLASH_DRIVER_IS_EXPORTED                 | 130         |
| 12.3.6         | kStatusGroupGeneric                      | 131         |
| 12.3.7         | MAKE_STATUS                              | 131         |
| 12.3.8         | FOUR_CHAR_CODE                           | 131         |
| <b>12.4</b>    | <b>Enumeration Type Documentation</b>    | <b>131</b>  |
| 12.4.1         | _flash_driver_version_constants          | 131         |
| 12.4.2         | _flash_status                            | 131         |
| 12.4.3         | _flash_driver_api_keys                   | 132         |
| 12.4.4         | flash_margin_value_t                     | 132         |
| 12.4.5         | flash_security_state_t                   | 132         |
| 12.4.6         | flash_protection_state_t                 | 132         |
| 12.4.7         | flash_execute_only_access_state_t        | 132         |
| 12.4.8         | flash_property_tag_t                     | 133         |
| 12.4.9         | _flash_execute_in_ram_function_constants | 133         |
| 12.4.10        | flash_read_resource_option_t             | 133         |
| 12.4.11        | _flash_read_resource_range               | 134         |
| 12.4.12        | flash_flexram_function_option_t          | 134         |
| 12.4.13        | flash_swap_function_option_t             | 134         |
| 12.4.14        | flash_swap_control_option_t              | 134         |
| 12.4.15        | flash_swap_state_t                       | 135         |
| 12.4.16        | flash_swap_block_status_t                | 135         |
| 12.4.17        | flash_partition_flexram_load_option_t    | 135         |
| <b>12.5</b>    | <b>Function Documentation</b>            | <b>135</b>  |
| 12.5.1         | FLASH_Init                               | 135         |
| 12.5.2         | FLASH_SetCallback                        | 136         |
| 12.5.3         | FLASH_PrepareExecuteInRamFunctions       | 136         |
| 12.5.4         | FLASH_EraseAll                           | 137         |
| 12.5.5         | FLASH_Erase                              | 138         |
| 12.5.6         | FLASH_EraseAllExecuteOnlySegments        | 139         |
| 12.5.7         | FLASH_Program                            | 141         |
| 12.5.8         | FLASH_ProgramOnce                        | 142         |
| 12.5.9         | FLASH_ReadOnce                           | 143         |
| 12.5.10        | FLASH_GetSecurityState                   | 145         |
| 12.5.11        | FLASH_SecurityBypass                     | 146         |
| 12.5.12        | FLASH_VerifyEraseAll                     | 147         |
| 12.5.13        | FLASH_VerifyErase                        | 148         |
| 12.5.14        | FLASH_VerifyProgram                      | 149         |
| 12.5.15        | FLASH_VerifyEraseAllExecuteOnlySegments  | 150         |

# Contents

| Section Number | Title  | Page Number |
|----------------|--|-------------|
| 12.5.16        | FLASH_IsProtected . . . . .                      | 151         |
| 12.5.17        | FLASH_IsExecuteOnly . . . . .                    | 152         |
| 12.5.18        | FLASH_GetProperty . . . . .                      | 152         |
| 12.5.19        | FLASH_PflashSetProtection . . . . .              | 153         |
| 12.5.20        | FLASH_PflashGetProtection . . . . .              | 153         |
| <b>Chapter</b> | <b>GPIO: General-Purpose Input/Output Driver</b> |             |
| <b>13.1</b>    | <b>Overview . . . . .</b>                        | <b>155</b>  |
| <b>13.2</b>    | <b>Data Structure Documentation . . . . .</b>    | <b>155</b>  |
| 13.2.1         | struct gpio_pin_config_t . . . . .               | 155         |
| <b>13.3</b>    | <b>Macro Definition Documentation . . . . .</b>  | <b>156</b>  |
| 13.3.1         | FSL_GPIO_DRIVER_VERSION . . . . .                | 156         |
| <b>13.4</b>    | <b>Enumeration Type Documentation . . . . .</b>  | <b>156</b>  |
| 13.4.1         | gpio_pin_direction_t . . . . .                   | 156         |
| <b>13.5</b>    | <b>GPIO Driver . . . . .</b>                     | <b>157</b>  |
| 13.5.1         | Overview . . . . .                               | 157         |
| 13.5.2         | Typical use case . . . . .                       | 157         |
| 13.5.3         | Function Documentation . . . . .                 | 158         |
| <b>13.6</b>    | <b>FGPIO Driver . . . . .</b>                    | <b>161</b>  |
| 13.6.1         | Typical use case . . . . .                       | 161         |
| <b>Chapter</b> | <b>I2C: Inter-Integrated Circuit Driver</b>      |             |
| <b>14.1</b>    | <b>Overview . . . . .</b>                        | <b>163</b>  |
| <b>14.2</b>    | <b>I2C Driver . . . . .</b>                      | <b>164</b>  |
| 14.2.1         | Overview . . . . .                               | 164         |
| 14.2.2         | Typical use case . . . . .                       | 164         |
| 14.2.3         | Data Structure Documentation . . . . .           | 171         |
| 14.2.4         | Macro Definition Documentation . . . . .         | 175         |
| 14.2.5         | Typedef Documentation . . . . .                  | 175         |
| 14.2.6         | Enumeration Type Documentation . . . . .         | 175         |
| 14.2.7         | Function Documentation . . . . .                 | 177         |
| <b>14.3</b>    | <b>I2C eDMA Driver . . . . .</b>                 | <b>191</b>  |
| 14.3.1         | Overview . . . . .                               | 191         |
| 14.3.2         | Data Structure Documentation . . . . .           | 191         |
| 14.3.3         | Typedef Documentation . . . . .                  | 192         |
| 14.3.4         | Function Documentation . . . . .                 | 192         |

# Contents

| Section Number | Title  | Page Number |
|----------------|--|-------------|
| <b>14.4</b>    | <b>I2C DMA Driver</b>  | <b>194</b>  |
| 14.4.1         | Overview   | 194         |
| 14.4.2         | Data Structure Documentation                                     | 194         |
| 14.4.3         | Typedef Documentation  | 195         |
| 14.4.4         | Function Documentation   | 195         |
| <b>14.5</b>    | <b>I2C FreeRTOS Driver</b>                                       | <b>197</b>  |
| 14.5.1         | Overview   | 197         |
| 14.5.2         | Data Structure Documentation                                     | 197         |
| 14.5.3         | Function Documentation   | 198         |
| <b>14.6</b>    | <b>I2C <math>\mu</math>COS/II Driver</b>                         | <b>200</b>  |
| 14.6.1         | Overview   | 200         |
| 14.6.2         | Data Structure Documentation                                     | 200         |
| 14.6.3         | Function Documentation   | 201         |
| <b>14.7</b>    | <b>I2C <math>\mu</math>COS/III Driver</b>                        | <b>203</b>  |
| 14.7.1         | Overview   | 203         |
| 14.7.2         | Data Structure Documentation                                     | 203         |
| 14.7.3         | Function Documentation   | 204         |
| <b>Chapter</b> | <b>LLWU: Low-Leakage Wakeup Unit Driver</b>                      |             |
| <b>15.1</b>    | <b>Overview</b>  | <b>207</b>  |
| <b>15.2</b>    | <b>External wakeup pins configurations</b>                       | <b>207</b>  |
| <b>15.3</b>    | <b>Internal wakeup modules configurations</b>                    | <b>207</b>  |
| <b>15.4</b>    | <b>Digital pin filter for external wakeup pin configurations</b> | <b>207</b>  |
| <b>15.5</b>    | <b>Data Structure Documentation</b>                              | <b>208</b>  |
| 15.5.1         | struct llwu_external_pin_filter_mode_t                           | 208         |
| <b>15.6</b>    | <b>Macro Definition Documentation</b>                            | <b>208</b>  |
| 15.6.1         | FSL_LLWU_DRIVER_VERSION  | 208         |
| <b>15.7</b>    | <b>Enumeration Type Documentation</b>                            | <b>208</b>  |
| 15.7.1         | llwu_external_pin_mode_t   | 208         |
| 15.7.2         | llwu_pin_filter_mode_t   | 209         |
| <b>15.8</b>    | <b>Function Documentation</b>                                    | <b>209</b>  |
| 15.8.1         | LLWU_SetExternalWakeupPinMode                                    | 209         |
| 15.8.2         | LLWU_GetExternalWakeupPinFlag                                    | 209         |
| 15.8.3         | LLWU_ClearExternalWakeupPinFlag                                  | 210         |
| 15.8.4         | LLWU_EnableInternalModuleInterruptWakup                          | 211         |

# Contents

| Section Number | Title  | Page Number |
|----------------|--|-------------|
| 15.8.5         | <a href="#">LLWU_GetInternalWakeupModuleFlag</a> | 211         |
| 15.8.6         | <a href="#">LLWU_SetPinFilterMode</a>            | 211         |
| 15.8.7         | <a href="#">LLWU_GetPinFilterFlag</a>            | 212         |
| 15.8.8         | <a href="#">LLWU_ClearPinFilterFlag</a>          | 212         |

## Chapter LPSCI: Universal Asynchronous Receiver/Transmitter

|             |   |            |
|-------------|---|------------|
| <b>16.1</b> | <b>Overview</b>                             | <b>213</b> |
| <b>16.2</b> | <b>LPSCI Driver</b>                         | <b>214</b> |
| 16.2.1      | Overview                                    | 214        |
| 16.2.2      | Function groups                             | 214        |
| 16.2.3      | Typical use case                            | 215        |
| 16.2.4      | Data Structure Documentation                | 218        |
| 16.2.5      | Macro Definition Documentation              | 219        |
| 16.2.6      | Typedef Documentation                       | 219        |
| 16.2.7      | Enumeration Type Documentation              | 219        |
| 16.2.8      | Function Documentation                      | 221        |
| <b>16.3</b> | <b>LPSCI DMA Driver</b>                     | <b>233</b> |
| 16.3.1      | Overview                                    | 233        |
| 16.3.2      | Data Structure Documentation                | 233        |
| 16.3.3      | Typedef Documentation                       | 234        |
| 16.3.4      | Function Documentation                      | 234        |
| <b>16.4</b> | <b>LPSCI FreeRTOS Driver</b>                | <b>238</b> |
| 16.4.1      | Overview                                    | 238        |
| 16.4.2      | Function Documentation                      | 238        |
| <b>16.5</b> | <b>LPSCI <math>\mu</math>COS/II Driver</b>  | <b>240</b> |
| 16.5.1      | Overview                                    | 240        |
| 16.5.2      | Function Documentation                      | 240        |
| <b>16.6</b> | <b>LPSCI <math>\mu</math>COS/III Driver</b> | <b>242</b> |
| 16.6.1      | Overview                                    | 242        |
| 16.6.2      | Function Documentation                      | 242        |

## Chapter LPTMR: Low-Power Timer

|             |                                     |            |
|-------------|-------------------------------------|------------|
| <b>17.1</b> | <b>Overview</b>                     | <b>245</b> |
| <b>17.2</b> | <b>Function groups</b>              | <b>245</b> |
| 17.2.1      | Initialization and deinitialization | 245        |
| 17.2.2      | Timer period Operations             | 245        |
| 17.2.3      | Start and Stop timer operations     | 245        |
| 17.2.4      | Status                              | 246        |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 17.2.5         | Interrupt . . . . .                             | 246         |
| <b>17.3</b>    | <b>Typical use case . . . . .</b>               | <b>246</b>  |
| 17.3.1         | LPTMR tick example . . . . .                    | 246         |
| <b>17.4</b>    | <b>Data Structure Documentation . . . . .</b>   | <b>248</b>  |
| 17.4.1         | struct lptmr_config_t . . . . .                 | 248         |
| <b>17.5</b>    | <b>Enumeration Type Documentation . . . . .</b> | <b>249</b>  |
| 17.5.1         | lptmr_pin_select_t . . . . .                    | 249         |
| 17.5.2         | lptmr_pin_polarity_t . . . . .                  | 249         |
| 17.5.3         | lptmr_timer_mode_t . . . . .                    | 249         |
| 17.5.4         | lptmr_prescaler_glitch_value_t . . . . .        | 250         |
| 17.5.5         | lptmr_prescaler_clock_select_t . . . . .        | 250         |
| 17.5.6         | lptmr_interrupt_enable_t . . . . .              | 250         |
| 17.5.7         | lptmr_status_flags_t . . . . .                  | 251         |
| <b>17.6</b>    | <b>Function Documentation . . . . .</b>         | <b>251</b>  |
| 17.6.1         | LPTMR_Init . . . . .                            | 251         |
| 17.6.2         | LPTMR_Deinit . . . . .                          | 251         |
| 17.6.3         | LPTMR_GetDefaultConfig . . . . .                | 251         |
| 17.6.4         | LPTMR_EnableInterrupts . . . . .                | 252         |
| 17.6.5         | LPTMR_DisableInterrupts . . . . .               | 252         |
| 17.6.6         | LPTMR_GetEnabledInterrupts . . . . .            | 252         |
| 17.6.7         | LPTMR_GetStatusFlags . . . . .                  | 252         |
| 17.6.8         | LPTMR_ClearStatusFlags . . . . .                | 253         |
| 17.6.9         | LPTMR_SetTimerPeriod . . . . .                  | 253         |
| 17.6.10        | LPTMR_GetCurrentTimerCount . . . . .            | 253         |
| 17.6.11        | LPTMR_StartTimer . . . . .                      | 254         |
| 17.6.12        | LPTMR_StopTimer . . . . .                       | 254         |
| <b>Chapter</b> | <b>PIT: Periodic Interrupt Timer</b>            |             |
| <b>18.1</b>    | <b>Overview . . . . .</b>                       | <b>255</b>  |
| <b>18.2</b>    | <b>Function groups . . . . .</b>                | <b>255</b>  |
| 18.2.1         | Initialization and deinitialization . . . . .   | 255         |
| 18.2.2         | Timer period Operations . . . . .               | 255         |
| 18.2.3         | Start and Stop timer operations . . . . .       | 255         |
| 18.2.4         | Status . . . . .                                | 256         |
| 18.2.5         | Interrupt . . . . .                             | 256         |
| <b>18.3</b>    | <b>Typical use case . . . . .</b>               | <b>256</b>  |
| 18.3.1         | PIT tick example . . . . .                      | 256         |
| <b>18.4</b>    | <b>Data Structure Documentation . . . . .</b>   | <b>258</b>  |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 18.4.1         | struct pit_config_t . . . . .                   | 258         |
| <b>18.5</b>    | <b>Enumeration Type Documentation . . . . .</b> | <b>258</b>  |
| 18.5.1         | pit_chnl_t . . . . .                            | 258         |
| 18.5.2         | pit_interrupt_enable_t . . . . .                | 259         |
| 18.5.3         | pit_status_flags_t . . . . .                    | 259         |
| <b>18.6</b>    | <b>Function Documentation . . . . .</b>         | <b>259</b>  |
| 18.6.1         | PIT_Init . . . . .                              | 259         |
| 18.6.2         | PIT_Deinit . . . . .                            | 259         |
| 18.6.3         | PIT_GetDefaultConfig . . . . .                  | 259         |
| 18.6.4         | PIT_EnableInterrupts . . . . .                  | 260         |
| 18.6.5         | PIT_DisableInterrupts . . . . .                 | 260         |
| 18.6.6         | PIT_GetEnabledInterrupts . . . . .              | 260         |
| 18.6.7         | PIT_GetStatusFlags . . . . .                    | 261         |
| 18.6.8         | PIT_ClearStatusFlags . . . . .                  | 262         |
| 18.6.9         | PIT_SetTimerPeriod . . . . .                    | 262         |
| 18.6.10        | PIT_GetCurrentTimerCount . . . . .              | 263         |
| 18.6.11        | PIT_StartTimer . . . . .                        | 263         |
| 18.6.12        | PIT_StopTimer . . . . .                         | 263         |
| <b>Chapter</b> | <b>PMC: Power Management Controller</b>         |             |
| <b>19.1</b>    | <b>Overview . . . . .</b>                       | <b>265</b>  |
| <b>19.2</b>    | <b>Data Structure Documentation . . . . .</b>   | <b>265</b>  |
| 19.2.1         | struct pmc_low_volt_detect_config_t . . . . .   | 265         |
| 19.2.2         | struct pmc_low_volt_warning_config_t . . . . .  | 266         |
| <b>19.3</b>    | <b>Macro Definition Documentation . . . . .</b> | <b>266</b>  |
| 19.3.1         | FSL_PMC_DRIVER_VERSION . . . . .                | 266         |
| <b>19.4</b>    | <b>Function Documentation . . . . .</b>         | <b>266</b>  |
| 19.4.1         | PMC_ConfigureLowVoltDetect . . . . .            | 266         |
| 19.4.2         | PMC_GetLowVoltDetectFlag . . . . .              | 266         |
| 19.4.3         | PMC_ClearLowVoltDetectFlag . . . . .            | 267         |
| 19.4.4         | PMC_ConfigureLowVoltWarning . . . . .           | 267         |
| 19.4.5         | PMC_GetLowVoltWarningFlag . . . . .             | 267         |
| 19.4.6         | PMC_ClearLowVoltWarningFlag . . . . .           | 268         |
| <b>Chapter</b> | <b>PORT: Port Control and Interrupts</b>        |             |
| <b>20.1</b>    | <b>Overview . . . . .</b>                       | <b>271</b>  |
| <b>20.2</b>    | <b>Typical configuration use case . . . . .</b> | <b>271</b>  |
| 20.2.1         | Input PORT configuration . . . . .              | 271         |



# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 20.2.2         | I2C PORT Configuration . . . . .                | 271         |
| <b>20.3</b>    | <b>Data Structure Documentation . . . . .</b>   | <b>273</b>  |
| 20.3.1         | struct port_pin_config_t . . . . .              | 273         |
| <b>20.4</b>    | <b>Macro Definition Documentation . . . . .</b> | <b>273</b>  |
| 20.4.1         | FSL_PORT_DRIVER_VERSION . . . . .               | 273         |
| <b>20.5</b>    | <b>Enumeration Type Documentation . . . . .</b> | <b>273</b>  |
| 20.5.1         | _port_pull . . . . .                            | 273         |
| 20.5.2         | _port_slew_rate . . . . .                       | 273         |
| 20.5.3         | _port_passive_filter_enable . . . . .           | 274         |
| 20.5.4         | _port_drive_strength . . . . .                  | 274         |
| 20.5.5         | port_mux_t . . . . .                            | 274         |
| 20.5.6         | port_interrupt_t . . . . .                      | 274         |
| <b>20.6</b>    | <b>Function Documentation . . . . .</b>         | <b>274</b>  |
| 20.6.1         | PORT_SetPinConfig . . . . .                     | 274         |
| 20.6.2         | PORT_SetMultiplePinsConfig . . . . .            | 275         |
| 20.6.3         | PORT_SetPinMux . . . . .                        | 275         |
| 20.6.4         | PORT_SetPinInterruptConfig . . . . .            | 277         |
| 20.6.5         | PORT_GetPinsInterruptFlags . . . . .            | 277         |
| 20.6.6         | PORT_ClearPinsInterruptFlags . . . . .          | 278         |
| <b>Chapter</b> | <b>RCM: Reset Control Module Driver</b>         |             |
| <b>21.1</b>    | <b>Overview . . . . .</b>                       | <b>279</b>  |
| <b>21.2</b>    | <b>Data Structure Documentation . . . . .</b>   | <b>280</b>  |
| 21.2.1         | struct rcm_reset_pin_filter_config_t . . . . .  | 280         |
| <b>21.3</b>    | <b>Macro Definition Documentation . . . . .</b> | <b>280</b>  |
| 21.3.1         | FSL_RCM_DRIVER_VERSION . . . . .                | 280         |
| <b>21.4</b>    | <b>Enumeration Type Documentation . . . . .</b> | <b>280</b>  |
| 21.4.1         | rcm_reset_source_t . . . . .                    | 280         |
| 21.4.2         | rcm_run_wait_filter_mode_t . . . . .            | 280         |
| <b>21.5</b>    | <b>Function Documentation . . . . .</b>         | <b>281</b>  |
| 21.5.1         | RCM_GetPreviousResetSources . . . . .           | 281         |
| 21.5.2         | RCM_ConfigureResetPinFilter . . . . .           | 281         |
| <b>Chapter</b> | <b>RTC: Real Time Clock</b>                     |             |
| <b>22.1</b>    | <b>Overview . . . . .</b>                       | <b>283</b>  |

# Contents

| Section Number | Title                                      | Page Number |
|----------------|--|-------------|
| <b>22.2</b>    | <b>Function groups</b>                     | <b>283</b>  |
| 22.2.1         | Initialization and deinitialization        | 283         |
| 22.2.2         | Set & Get Datetime                         | 283         |
| 22.2.3         | Set & Get Alarm                            | 283         |
| 22.2.4         | Start & Stop timer                         | 284         |
| 22.2.5         | Status                                     | 284         |
| 22.2.6         | Interrupt                                  | 284         |
| 22.2.7         | RTC Oscillator                             | 284         |
| 22.2.8         | Monotonic Counter                          | 284         |
| <b>22.3</b>    | <b>Typical use case</b>                    | <b>284</b>  |
| 22.3.1         | RTC tick example                           | 284         |
| <b>22.4</b>    | <b>Data Structure Documentation</b>        | <b>287</b>  |
| 22.4.1         | struct rtc_datetime_t                      | 287         |
| 22.4.2         | struct rtc_config_t                        | 288         |
| <b>22.5</b>    | <b>Enumeration Type Documentation</b>      | <b>288</b>  |
| 22.5.1         | rtc_interrupt_enable_t                     | 288         |
| 22.5.2         | rtc_status_flags_t                         | 289         |
| <b>22.6</b>    | <b>Function Documentation</b>              | <b>289</b>  |
| 22.6.1         | RTC_Init                                   | 289         |
| 22.6.2         | RTC_Deinit                                 | 289         |
| 22.6.3         | RTC_GetDefaultConfig                       | 289         |
| 22.6.4         | RTC_SetDatetime                            | 290         |
| 22.6.5         | RTC_GetDatetime                            | 290         |
| 22.6.6         | RTC_SetAlarm                               | 290         |
| 22.6.7         | RTC_GetAlarm                               | 291         |
| 22.6.8         | RTC_EnableInterrupts                       | 291         |
| 22.6.9         | RTC_DisableInterrupts                      | 291         |
| 22.6.10        | RTC_GetEnabledInterrupts                   | 291         |
| 22.6.11        | RTC_GetStatusFlags                         | 292         |
| 22.6.12        | RTC_ClearStatusFlags                       | 292         |
| 22.6.13        | RTC_StartTimer                             | 292         |
| 22.6.14        | RTC_StopTimer                              | 292         |
| 22.6.15        | RTC_Reset                                  | 293         |
| <b>Chapter</b> | <b>SAI: Serial Audio Interface</b>         |             |
| <b>23.1</b>    | <b>Overview</b>                            | <b>295</b>  |
| <b>23.2</b>    | <b>Typical use case</b>                    | <b>295</b>  |
| 23.2.1         | SAI Send/Receive using an interrupt method | 295         |
| 23.2.2         | SAI Send/receive using a DMA method        | 296         |

# Contents

| Section Number | Title                                 | Page Number |
|----------------|---------------------------------------|-------------|
| <b>23.3</b>    | <b>Data Structure Documentation</b>   | <b>301</b>  |
| 23.3.1         | struct sai_config_t                   | 301         |
| 23.3.2         | struct sai_transfer_format_t          | 302         |
| 23.3.3         | struct sai_transfer_t                 | 302         |
| 23.3.4         | struct _sai_handle                    | 302         |
| <b>23.4</b>    | <b>Macro Definition Documentation</b> | <b>303</b>  |
| 23.4.1         | SAI_XFER_QUEUE_SIZE                   | 303         |
| <b>23.5</b>    | <b>Enumeration Type Documentation</b> | <b>303</b>  |
| 23.5.1         | _sai_status_t                         | 303         |
| 23.5.2         | sai_protocol_t                        | 303         |
| 23.5.3         | sai_master_slave_t                    | 304         |
| 23.5.4         | sai_mono_stereo_t                     | 304         |
| 23.5.5         | sai_sync_mode_t                       | 304         |
| 23.5.6         | sai_mclk_source_t                     | 304         |
| 23.5.7         | sai_bclk_source_t                     | 304         |
| 23.5.8         | _sai_interrupt_enable_t               | 305         |
| 23.5.9         | _sai_dma_enable_t                     | 305         |
| 23.5.10        | _sai_flags                            | 305         |
| 23.5.11        | sai_reset_type_t                      | 305         |
| 23.5.12        | sai_sample_rate_t                     | 305         |
| 23.5.13        | sai_word_width_t                      | 306         |
| <b>23.6</b>    | <b>Function Documentation</b>         | <b>306</b>  |
| 23.6.1         | SAI_TxInit                            | 306         |
| 23.6.2         | SAI_RxInit                            | 306         |
| 23.6.3         | SAI_TxGetDefaultConfig                | 307         |
| 23.6.4         | SAI_RxGetDefaultConfig                | 307         |
| 23.6.5         | SAI_Deinit                            | 307         |
| 23.6.6         | SAI_TxReset                           | 308         |
| 23.6.7         | SAI_RxReset                           | 308         |
| 23.6.8         | SAI_TxEnable                          | 308         |
| 23.6.9         | SAI_RxEnable                          | 308         |
| 23.6.10        | SAI_TxGetStatusFlag                   | 308         |
| 23.6.11        | SAI_TxClearStatusFlags                | 309         |
| 23.6.12        | SAI_RxGetStatusFlag                   | 309         |
| 23.6.13        | SAI_RxClearStatusFlags                | 309         |
| 23.6.14        | SAI_TxEnableInterrupts                | 310         |
| 23.6.15        | SAI_RxEnableInterrupts                | 310         |
| 23.6.16        | SAI_TxDisableInterrupts               | 311         |
| 23.6.17        | SAI_RxDisableInterrupts               | 312         |
| 23.6.18        | SAI_TxEnableDMA                       | 312         |
| 23.6.19        | SAI_RxEnableDMA                       | 313         |
| 23.6.20        | SAI_TxGetDataRegisterAddress          | 314         |

# Contents

| Section Number | Title  | Page Number |
|----------------|--|-------------|
| 23.6.21        | <a href="#">SAI_RxGetDataRegisterAddress</a>   | 314         |
| 23.6.22        | <a href="#">SAI_TxSetFormat</a>                | 315         |
| 23.6.23        | <a href="#">SAI_RxSetFormat</a>                | 316         |
| 23.6.24        | <a href="#">SAI_WriteBlocking</a>              | 316         |
| 23.6.25        | <a href="#">SAI_WriteData</a>                  | 317         |
| 23.6.26        | <a href="#">SAI_ReadBlocking</a>               | 317         |
| 23.6.27        | <a href="#">SAI_ReadData</a>                   | 317         |
| 23.6.28        | <a href="#">SAI_TransferTxCreateHandle</a>     | 318         |
| 23.6.29        | <a href="#">SAI_TransferRxCreateHandle</a>     | 318         |
| 23.6.30        | <a href="#">SAI_TransferTxSetFormat</a>        | 319         |
| 23.6.31        | <a href="#">SAI_TransferRxSetFormat</a>        | 320         |
| 23.6.32        | <a href="#">SAI_TransferSendNonBlocking</a>    | 320         |
| 23.6.33        | <a href="#">SAI_TransferReceiveNonBlocking</a> | 321         |
| 23.6.34        | <a href="#">SAI_TransferGetSendCount</a>       | 322         |
| 23.6.35        | <a href="#">SAI_TransferGetReceiveCount</a>    | 323         |
| 23.6.36        | <a href="#">SAI_TransferAbortSend</a>          | 323         |
| 23.6.37        | <a href="#">SAI_TransferAbortReceive</a>       | 324         |
| 23.6.38        | <a href="#">SAI_TransferTxHandleIRQ</a>        | 324         |
| 23.6.39        | <a href="#">SAI_TransferRxHandleIRQ</a>        | 324         |
| <b>23.7</b>    | <b>SAI DMA Driver</b>                          | <b>325</b>  |
| 23.7.1         | <a href="#">Overview</a>                       | 325         |
| 23.7.2         | <a href="#">Data Structure Documentation</a>   | 326         |
| 23.7.3         | <a href="#">Function Documentation</a>         | 326         |
| <b>23.8</b>    | <b>SAI eDMA Driver</b>                         | <b>332</b>  |
| 23.8.1         | <a href="#">Overview</a>                       | 332         |
| 23.8.2         | <a href="#">Data Structure Documentation</a>   | 333         |
| 23.8.3         | <a href="#">Function Documentation</a>         | 333         |
| <b>Chapter</b> | <b>SIM: System Integration Module Driver</b>   |             |
| <b>24.1</b>    | <b>Overview</b>                                | <b>339</b>  |
| <b>24.2</b>    | <b>Data Structure Documentation</b>            | <b>339</b>  |
| 24.2.1         | <a href="#">struct sim_uid_t</a>               | 339         |
| <b>24.3</b>    | <b>Enumeration Type Documentation</b>          | <b>340</b>  |
| 24.3.1         | <a href="#">_sim_flash_mode</a>                | 340         |
| <b>24.4</b>    | <b>Function Documentation</b>                  | <b>340</b>  |
| 24.4.1         | <a href="#">SIM_GetUniqueId</a>                | 340         |
| 24.4.2         | <a href="#">SIM_SetFlashMode</a>               | 340         |

# Contents

| Section<br>Number | Title                                   | Page<br>Number |
|-------------------|---|----------------|
| <b>Chapter</b>    | <b>SLCD: Segment LCD Driver</b>         |                |
| <b>25.1</b>       | <b>Overview</b>                         | <b>341</b>     |
| <b>25.2</b>       | <b>Typical use case</b>                 | <b>341</b>     |
| 25.2.1            | SLCD Initialization operation           | 341            |
| <b>25.3</b>       | <b>Data Structure Documentation</b>     | <b>346</b>     |
| 25.3.1            | struct slcd_fault_detect_config_t       | 346            |
| 25.3.2            | struct slcd_clock_config_t              | 346            |
| 25.3.3            | struct slcd_config_t                    | 347            |
| <b>25.4</b>       | <b>Macro Definition Documentation</b>   | <b>348</b>     |
| 25.4.1            | FSL_SLCD_DRIVER_VERSION                 | 348            |
| <b>25.5</b>       | <b>Enumeration Type Documentation</b>   | <b>348</b>     |
| 25.5.1            | slcd_power_supply_option_t              | 348            |
| 25.5.2            | slcd_regulated_voltage_trim_t           | 349            |
| 25.5.3            | slcd_load_adjust_t                      | 349            |
| 25.5.4            | slcd_clock_src_t                        | 350            |
| 25.5.5            | slcd_alt_clock_div_t                    | 350            |
| 25.5.6            | slcd_clock_prescaler_t                  | 350            |
| 25.5.7            | slcd_duty_cycle_t                       | 351            |
| 25.5.8            | slcd_phase_type_t                       | 351            |
| 25.5.9            | slcd_phase_index_t                      | 351            |
| 25.5.10           | slcd_display_mode_t                     | 352            |
| 25.5.11           | slcd_blink_mode_t                       | 352            |
| 25.5.12           | slcd_blink_rate_t                       | 352            |
| 25.5.13           | slcd_fault_detect_clock_prescaler_t     | 352            |
| 25.5.14           | slcd_fault_detect_sample_window_width_t | 353            |
| 25.5.15           | slcd_interrupt_enable_t                 | 353            |
| 25.5.16           | slcd_lowpower_behavior                  | 353            |
| <b>25.6</b>       | <b>Function Documentation</b>           | <b>353</b>     |
| 25.6.1            | SLCD_Init                               | 353            |
| 25.6.2            | SLCD_Deinit                             | 354            |
| 25.6.3            | SLCD_GetDefaultConfig                   | 354            |
| 25.6.4            | SLCD_StartDisplay                       | 354            |
| 25.6.5            | SLCD_StopDisplay                        | 355            |
| 25.6.6            | SLCD_StartBlinkMode                     | 355            |
| 25.6.7            | SLCD_StopBlinkMode                      | 355            |
| 25.6.8            | SLCD_SetBackPlanePhase                  | 355            |
| 25.6.9            | SLCD_SetFrontPlaneSegments              | 356            |
| 25.6.10           | SLCD_SetFrontPlaneOnePhase              | 356            |
| 25.6.11           | SLCD_GetFaultDetectCounter              | 357            |
| 25.6.12           | SLCD_EnableInterrupts                   | 357            |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 25.6.13        | SLCD_DisableInterrupts . . . . .                | 358         |
| 25.6.14        | SLCD_GetInterruptStatus . . . . .               | 358         |
| 25.6.15        | SLCD_ClearInterruptStatus . . . . .             | 358         |
| <b>Chapter</b> | <b>SMC: System Mode Controller Driver</b>       |             |
| <b>26.1</b>    | <b>Overview . . . . .</b>                       | <b>361</b>  |
| <b>26.2</b>    | <b>Macro Definition Documentation . . . . .</b> | <b>362</b>  |
| 26.2.1         | FSL_SMC_DRIVER_VERSION . . . . .                | 362         |
| <b>26.3</b>    | <b>Enumeration Type Documentation . . . . .</b> | <b>362</b>  |
| 26.3.1         | smc_power_mode_protection_t . . . . .           | 362         |
| 26.3.2         | smc_power_state_t . . . . .                     | 362         |
| 26.3.3         | smc_run_mode_t . . . . .                        | 363         |
| 26.3.4         | smc_stop_mode_t . . . . .                       | 363         |
| 26.3.5         | smc_partial_stop_option_t . . . . .             | 363         |
| 26.3.6         | _smc_status . . . . .                           | 363         |
| <b>26.4</b>    | <b>Function Documentation . . . . .</b>         | <b>363</b>  |
| 26.4.1         | SMC_SetPowerModeProtection . . . . .            | 363         |
| 26.4.2         | SMC_GetPowerModeState . . . . .                 | 364         |
| 26.4.3         | SMC_SetPowerModeRun . . . . .                   | 364         |
| 26.4.4         | SMC_SetPowerModeWait . . . . .                  | 364         |
| 26.4.5         | SMC_SetPowerModeStop . . . . .                  | 365         |
| 26.4.6         | SMC_SetPowerModeVlpr . . . . .                  | 365         |
| 26.4.7         | SMC_SetPowerModeVlps . . . . .                  | 365         |
| 26.4.8         | SMC_SetPowerModeVlps . . . . .                  | 366         |
| <b>Chapter</b> | <b>SPI: Serial Peripheral Interface Driver</b>  |             |
| <b>27.1</b>    | <b>Overview . . . . .</b>                       | <b>367</b>  |
| <b>27.2</b>    | <b>SPI Driver . . . . .</b>                     | <b>368</b>  |
| 27.2.1         | Overview . . . . .                              | 368         |
| 27.2.2         | Typical use case . . . . .                      | 368         |
| 27.2.3         | Data Structure Documentation . . . . .          | 373         |
| 27.2.4         | Macro Definition Documentation . . . . .        | 375         |
| 27.2.5         | Enumeration Type Documentation . . . . .        | 375         |
| 27.2.6         | Function Documentation . . . . .                | 377         |
| <b>27.3</b>    | <b>SPI DMA Driver . . . . .</b>                 | <b>387</b>  |
| 27.3.1         | Overview . . . . .                              | 387         |
| 27.3.2         | Data Structure Documentation . . . . .          | 388         |
| 27.3.3         | Typedef Documentation . . . . .                 | 388         |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 27.3.4         | Function Documentation . . . . .                    | 388         |
| <b>27.4</b>    | <b>SPI FreeRTOS driver . . . . .</b>                | <b>393</b>  |
| 27.4.1         | Overview . . . . .                                  | 393         |
| 27.4.2         | Data Structure Documentation . . . . .              | 393         |
| 27.4.3         | Function Documentation . . . . .                    | 394         |
| <b>27.5</b>    | <b>SPI <math>\mu</math>COS/II driver . . . . .</b>  | <b>396</b>  |
| 27.5.1         | Overview . . . . .                                  | 396         |
| 27.5.2         | Data Structure Documentation . . . . .              | 396         |
| 27.5.3         | Function Documentation . . . . .                    | 397         |
| <b>27.6</b>    | <b>SPI <math>\mu</math>COS/III driver . . . . .</b> | <b>399</b>  |
| 27.6.1         | Overview . . . . .                                  | 399         |
| 27.6.2         | Data Structure Documentation . . . . .              | 399         |
| 27.6.3         | Function Documentation . . . . .                    | 400         |
| <b>Chapter</b> | <b>TPM: Timer PWM Module</b>                        |             |
| <b>28.1</b>    | <b>Overview . . . . .</b>                           | <b>403</b>  |
| <b>28.2</b>    | <b>Typical use case . . . . .</b>                   | <b>404</b>  |
| 28.2.1         | PWM output . . . . .                                | 404         |
| <b>28.3</b>    | <b>Data Structure Documentation . . . . .</b>       | <b>408</b>  |
| 28.3.1         | struct tpm_chnl_pwm_signal_param_t . . . . .        | 408         |
| 28.3.2         | struct tpm_config_t . . . . .                       | 408         |
| <b>28.4</b>    | <b>Enumeration Type Documentation . . . . .</b>     | <b>409</b>  |
| 28.4.1         | tpm_chnl_t . . . . .                                | 409         |
| 28.4.2         | tpm_pwm_mode_t . . . . .                            | 409         |
| 28.4.3         | tpm_pwm_level_select_t . . . . .                    | 409         |
| 28.4.4         | tpm_trigger_select_t . . . . .                      | 410         |
| 28.4.5         | tpm_output_compare_mode_t . . . . .                 | 410         |
| 28.4.6         | tpm_input_capture_edge_t . . . . .                  | 410         |
| 28.4.7         | tpm_clock_source_t . . . . .                        | 410         |
| 28.4.8         | tpm_clock_prescale_t . . . . .                      | 410         |
| 28.4.9         | tpm_interrupt_enable_t . . . . .                    | 411         |
| 28.4.10        | tpm_status_flags_t . . . . .                        | 411         |
| <b>28.5</b>    | <b>Function Documentation . . . . .</b>             | <b>411</b>  |
| 28.5.1         | TPM_Init . . . . .                                  | 411         |
| 28.5.2         | TPM_Deinit . . . . .                                | 412         |
| 28.5.3         | TPM_GetDefaultConfig . . . . .                      | 412         |
| 28.5.4         | TPM_SetupPwm . . . . .                              | 412         |
| 28.5.5         | TPM_UpdatePwmDutycycle . . . . .                    | 413         |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 28.5.6         | TPM_UpdateChnlEdgeLevelSelect . . . . .                         | 413         |
| 28.5.7         | TPM_SetupInputCapture . . . . .                                 | 414         |
| 28.5.8         | TPM_SetupOutputCompare . . . . .                                | 414         |
| 28.5.9         | TPM_EnableInterrupts . . . . .                                  | 414         |
| 28.5.10        | TPM_DisableInterrupts . . . . .                                 | 415         |
| 28.5.11        | TPM_GetEnabledInterrupts . . . . .                              | 415         |
| 28.5.12        | TPM_GetStatusFlags . . . . .                                    | 415         |
| 28.5.13        | TPM_ClearStatusFlags . . . . .                                  | 415         |
| 28.5.14        | TPM_StartTimer . . . . .  | 416         |
| 28.5.15        | TPM_StopTimer . . . . .   | 416         |
| <b>Chapter</b> | <b>UART: Universal Asynchronous Receiver/Transmitter Driver</b> |             |
| <b>29.1</b>    | <b>Overview . . . . .</b>                                       | <b>417</b>  |
| <b>29.2</b>    | <b>UART Driver . . . . .</b>                                    | <b>418</b>  |
| 29.2.1         | Overview . . . . .  | 418         |
| 29.2.2         | Typical use case . . . . .                                      | 418         |
| 29.2.3         | Data Structure Documentation . . . . .                          | 426         |
| 29.2.4         | Macro Definition Documentation . . . . .                        | 428         |
| 29.2.5         | Typedef Documentation . . . . .                                 | 428         |
| 29.2.6         | Enumeration Type Documentation . . . . .                        | 428         |
| 29.2.7         | Function Documentation . . . . .                                | 430         |
| <b>29.3</b>    | <b>UART DMA Driver . . . . .</b>                                | <b>442</b>  |
| 29.3.1         | Overview . . . . .  | 442         |
| 29.3.2         | Data Structure Documentation . . . . .                          | 442         |
| 29.3.3         | Typedef Documentation . . . . .                                 | 443         |
| 29.3.4         | Function Documentation . . . . .                                | 443         |
| <b>29.4</b>    | <b>UART eDMA Driver . . . . .</b>                               | <b>447</b>  |
| 29.4.1         | Overview . . . . .  | 447         |
| 29.4.2         | Data Structure Documentation . . . . .                          | 447         |
| 29.4.3         | Typedef Documentation . . . . .                                 | 448         |
| 29.4.4         | Function Documentation . . . . .                                | 448         |
| <b>29.5</b>    | <b>UART FreeRTOS Driver . . . . .</b>                           | <b>452</b>  |
| 29.5.1         | Overview . . . . .  | 452         |
| 29.5.2         | Data Structure Documentation . . . . .                          | 452         |
| 29.5.3         | Function Documentation . . . . .                                | 453         |
| <b>29.6</b>    | <b>UART <math>\mu</math>COS/II Driver . . . . .</b>             | <b>456</b>  |
| 29.6.1         | Overview . . . . .  | 456         |
| 29.6.2         | Data Structure Documentation . . . . .                          | 456         |
| 29.6.3         | Function Documentation . . . . .                                | 457         |



# Contents

| Section Number | Title  | Page Number |
|----------------|--|-------------|
| <b>29.7</b>    | <b>UART <math>\mu</math>COS/III Driver</b> . . . . . | <b>460</b>  |
| 29.7.1         | Overview . . . . .                                   | 460         |
| 29.7.2         | Data Structure Documentation . . . . .               | 460         |
| 29.7.3         | Function Documentation . . . . .                     | 461         |
| <b>Chapter</b> | <b>Debug Console</b>                                 |             |
| <b>30.1</b>    | <b>Overview</b> . . . . .                            | <b>465</b>  |
| <b>30.2</b>    | <b>Function groups</b> . . . . .                     | <b>465</b>  |
| 30.2.1         | Initialization . . . . .                             | 465         |
| 30.2.2         | Advanced Feature . . . . .                           | 466         |
| <b>30.3</b>    | <b>Typical use case</b> . . . . .                    | <b>469</b>  |
| <b>30.4</b>    | <b>Semihosting</b> . . . . .                         | <b>471</b>  |
| 30.4.1         | Guide Semihosting for IAR . . . . .                  | 471         |
| 30.4.2         | Guide Semihosting for Keil $\mu$ Vision . . . . .    | 471         |
| 30.4.3         | Guide Semihosting for KDS . . . . .                  | 473         |
| 30.4.4         | Guide Semihosting for ATL . . . . .                  | 473         |
| 30.4.5         | Guide Semihosting for ARMGCC . . . . .               | 474         |
| <b>Chapter</b> | <b>Notification Framework</b>                        |             |
| <b>31.1</b>    | <b>Overview</b> . . . . .                            | <b>477</b>  |
| <b>31.2</b>    | <b>Notifier Overview</b> . . . . .                   | <b>477</b>  |
| <b>31.3</b>    | <b>Data Structure Documentation</b> . . . . .        | <b>479</b>  |
| 31.3.1         | struct notifier_notification_block_t . . . . .       | 479         |
| 31.3.2         | struct notifier_callback_config_t . . . . .          | 480         |
| 31.3.3         | struct notifier_handle_t . . . . .                   | 480         |
| <b>31.4</b>    | <b>Typedef Documentation</b> . . . . .               | <b>481</b>  |
| 31.4.1         | notifier_user_config_t . . . . .                     | 481         |
| 31.4.2         | notifier_user_function_t . . . . .                   | 481         |
| 31.4.3         | notifier_callback_t . . . . .                        | 482         |
| <b>31.5</b>    | <b>Enumeration Type Documentation</b> . . . . .      | <b>482</b>  |
| 31.5.1         | _notifier_status . . . . .                           | 482         |
| 31.5.2         | notifier_policy_t . . . . .                          | 483         |
| 31.5.3         | notifier_notification_type_t . . . . .               | 483         |
| 31.5.4         | notifier_callback_type_t . . . . .                   | 483         |
| <b>31.6</b>    | <b>Function Documentation</b> . . . . .              | <b>484</b>  |
| 31.6.1         | NOTIFIER_CreateHandle . . . . .                      | 484         |

# Contents

| Section Number             | Title   | Page Number |
|----------------------------|---|-------------|
| 31.6.2                     | NOTIFIER_SwitchConfig . . . . .                 | 485         |
| 31.6.3                     | NOTIFIER_GetErrorCallbackIndex . . . . .        | 486         |
| <b>Chapter Shell</b>       |   |             |
| <b>32.1</b>                | <b>Overview . . . . .</b>                       | <b>487</b>  |
| <b>32.2</b>                | <b>Function groups . . . . .</b>                | <b>487</b>  |
| 32.2.1                     | Initialization . . . . .                        | 487         |
| 32.2.2                     | Advanced Feature . . . . .                      | 487         |
| 32.2.3                     | Shell Operation . . . . .                       | 488         |
| <b>32.3</b>                | <b>Data Structure Documentation . . . . .</b>   | <b>489</b>  |
| 32.3.1                     | struct shell_context_struct . . . . .           | 489         |
| 32.3.2                     | struct shell_command_context_t . . . . .        | 490         |
| 32.3.3                     | struct shell_command_context_list_t . . . . .   | 490         |
| <b>32.4</b>                | <b>Macro Definition Documentation . . . . .</b> | <b>491</b>  |
| 32.4.1                     | SHELL_USE_HISTORY . . . . .                     | 491         |
| 32.4.2                     | SHELL_SEARCH_IN_HIST . . . . .                  | 491         |
| 32.4.3                     | SHELL_USE_FILE_STREAM . . . . .                 | 491         |
| 32.4.4                     | SHELL_AUTO_COMPLETE . . . . .                   | 491         |
| 32.4.5                     | SHELL_BUFFER_SIZE . . . . .                     | 491         |
| 32.4.6                     | SHELL_MAX_ARGS . . . . .                        | 491         |
| 32.4.7                     | SHELL_HIST_MAX . . . . .                        | 491         |
| 32.4.8                     | SHELL_MAX_CMD . . . . .                         | 491         |
| <b>32.5</b>                | <b>Typedef Documentation . . . . .</b>          | <b>491</b>  |
| 32.5.1                     | send_data_cb_t . . . . .                        | 491         |
| 32.5.2                     | recv_data_cb_t . . . . .                        | 491         |
| 32.5.3                     | printf_data_t . . . . .                         | 491         |
| 32.5.4                     | cmd_function_t . . . . .                        | 491         |
| <b>32.6</b>                | <b>Enumeration Type Documentation . . . . .</b> | <b>491</b>  |
| 32.6.1                     | fun_key_status_t . . . . .                      | 491         |
| <b>32.7</b>                | <b>Function Documentation . . . . .</b>         | <b>492</b>  |
| 32.7.1                     | SHELL_Init . . . . .                            | 492         |
| 32.7.2                     | SHELL_RegisterCommand . . . . .                 | 492         |
| 32.7.3                     | SHELL_Main . . . . .                            | 492         |
| <b>Chapter DMA Manager</b> |   |             |
| <b>33.1</b>                | <b>Overview . . . . .</b>                       | <b>495</b>  |
| <b>33.2</b>                | <b>Function groups . . . . .</b>                | <b>495</b>  |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 33.2.1         | DMAMGR Initialization and De-initialization . . . . .       | 495         |
| 33.2.2         | DMAMGR Operation . . . . .                                  | 495         |
| <b>33.3</b>    | <b>Typical use case . . . . .</b>                           | <b>495</b>  |
| 33.3.1         | DMAMGR static channel allocate . . . . .                    | 495         |
| 33.3.2         | DMAMGR dynamic channel allocate . . . . .                   | 495         |
| <b>33.4</b>    | <b>Macro Definition Documentation . . . . .</b>             | <b>496</b>  |
| 33.4.1         | DMAMGR_DYNAMIC_ALLOCATE . . . . .                           | 496         |
| <b>33.5</b>    | <b>Enumeration Type Documentation . . . . .</b>             | <b>496</b>  |
| 33.5.1         | _dma_manager_status . . . . .                               | 496         |
| <b>33.6</b>    | <b>Function Documentation . . . . .</b>                     | <b>496</b>  |
| 33.6.1         | DMAMGR_Init . . . . .                                       | 496         |
| 33.6.2         | DMAMGR_Deinit . . . . .                                     | 497         |
| 33.6.3         | DMAMGR_RequestChannel . . . . .                             | 497         |
| 33.6.4         | DMAMGR_ReleaseChannel . . . . .                             | 497         |
| <b>Chapter</b> | <b>Secured Digital Card/Embedded MultiMedia Card (CARD)</b> |             |
| <b>34.1</b>    | <b>Overview . . . . .</b>                                   | <b>499</b>  |
| <b>34.2</b>    | <b>Data Structure Documentation . . . . .</b>               | <b>502</b>  |
| 34.2.1         | struct sd_card_t . . . . .                                  | 502         |
| 34.2.2         | struct mmc_card_t . . . . .                                 | 503         |
| 34.2.3         | struct mmc_boot_config_t . . . . .                          | 504         |
| <b>34.3</b>    | <b>Macro Definition Documentation . . . . .</b>             | <b>504</b>  |
| 34.3.1         | FSL_SDMMC_DRIVER_VERSION . . . . .                          | 504         |
| <b>34.4</b>    | <b>Enumeration Type Documentation . . . . .</b>             | <b>504</b>  |
| 34.4.1         | _sdmmc_status . . . . .                                     | 504         |
| 34.4.2         | _sd_card_flag . . . . .                                     | 505         |
| 34.4.3         | _mmc_card_flag . . . . .                                    | 505         |
| <b>34.5</b>    | <b>Function Documentation . . . . .</b>                     | <b>506</b>  |
| 34.5.1         | SD_Init . . . . .   | 506         |
| 34.5.2         | SD_Deinit . . . . .   | 507         |
| 34.5.3         | SD_CheckReadOnly . . . . .                                  | 508         |
| 34.5.4         | SD_ReadBlocks . . . . .                                     | 508         |
| 34.5.5         | SD_WriteBlocks . . . . .                                    | 509         |
| 34.5.6         | SD_EraseBlocks . . . . .                                    | 510         |
| 34.5.7         | MMC_Init . . . . .  | 510         |
| 34.5.8         | MMC_Deinit . . . . .  | 511         |
| 34.5.9         | MMC_CheckReadOnly . . . . .                                 | 511         |

# Contents

| Section Number | Title   | Page Number |
|----------------|---|-------------|
| 34.5.10        | MMC_ReadBlocks . . . . .                        | 512         |
| 34.5.11        | MMC_WriteBlocks . . . . .                       | 512         |
| 34.5.12        | MMC_EraseGroups . . . . .                       | 513         |
| 34.5.13        | MMC_SelectPartition . . . . .                   | 514         |
| 34.5.14        | MMC_SetBootConfig . . . . .                     | 514         |
| <b>Chapter</b> | <b>SPI based Secured Digital Card (SDSPI)</b>   |             |
| <b>35.1</b>    | <b>Overview . . . . .</b>                       | <b>517</b>  |
| <b>35.2</b>    | <b>Data Structure Documentation . . . . .</b>   | <b>519</b>  |
| 35.2.1         | struct sdspi_command_t . . . . .                | 519         |
| 35.2.2         | struct sdspi_host_t . . . . .                   | 519         |
| 35.2.3         | struct sdspi_card_t . . . . .                   | 519         |
| <b>35.3</b>    | <b>Enumeration Type Documentation . . . . .</b> | <b>520</b>  |
| 35.3.1         | _sdspi_status . . . . .                         | 520         |
| 35.3.2         | _sdspi_card_flag . . . . .                      | 521         |
| 35.3.3         | sdspi_response_type_t . . . . .                 | 521         |
| <b>35.4</b>    | <b>Function Documentation . . . . .</b>         | <b>521</b>  |
| 35.4.1         | SDSPI_Init . . . . .                            | 521         |
| 35.4.2         | SDSPI_Deinit . . . . .                          | 522         |
| 35.4.3         | SDSPI_CheckReadOnly . . . . .                   | 522         |
| 35.4.4         | SDSPI_ReadBlocks . . . . .                      | 523         |
| 35.4.5         | SDSPI_WriteBlocks . . . . .                     | 523         |

# Chapter 1

## Introduction

The Kinetis Software Development Kit (KSDK) 2.0 is a collection of software enablement, for NXP Kinetis Microcontrollers, that includes peripheral drivers, high-level stacks including USB and lwIP, integration with WolfSSL and mbed TLS cryptography libraries, other middleware packages (multicore support and FatFS), and integrated RTOS support for FreeRTOS,  $\mu$ C/OS-II, and  $\mu$ C/OS-III. In addition to the base enablement, the KSDK is augmented with demo applications, driver example projects, and API documentation to help users quickly leverage the support of the Kinetis SDK. The Kinetis Expert (KEx) Web UI is available to provide access to all Kinetis SDK packages. See the *Kinetis SDK v.2.0.0 Release Notes* (document KSDK200RN) and the supported Devices section at [www.nxp.com/ksdk](http://www.nxp.com/ksdk) for details.

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

- ARM<sup>®</sup> and DSP standard libraries, and CMSIS-compliant device header files which provide direct access to the peripheral registers.
- Open-source peripheral drivers that provide stateless, high-performance, ease-of-use APIs. Communication drivers provide higher-level transactional APIs for a higher-performance option.
- Open-source RTOS wrapper driver built on on top of KSDK peripheral drivers and leverage native RTOS services to better comply to the RTOS cases.
- Real time operation systems (RTOS) including FreeRTOS OS,  $\mu$ C/OS-II, and  $\mu$ C/OS-III.
- Stacks and middleware in source or object formats including:
  - A USB device, host, and OTG stack with comprehensive USB class support.
  - CMSIS-DSP, a suite of common signal processing functions.
  - FatFs, a FAT file system for small embedded systems.
  - Encryption software utilizing the mmCAU hardware acceleration.
  - SDMMC, a software component supporting SD Cards and eMMC.
  - mbedTLS, cryptographic SSL/TLS libraries.
  - lwIP, a light-weight TCP/IP stack.
  - WolfSSL, a cryptography and SSL/TLS library.
  - EMV L1 that complies to EMV-v4.3\_Book\_1 specification.
  - DMA Manager, a software component used for managing on-chip DMA channel resources.
  - The Kinetis 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:

- Atollic TrueSTUDIO
- GNU toolchain for ARM<sup>®</sup> Cortex<sup>®</sup> -M with Cmake build system
- IAR Embedded Workbench
- Keil MDK
- Kinetis Design Studio

The peripheral drivers and RTOS driver wrappers can be used across multiple devices within the Kinetis product family without modification. The configuration items for each driver are encapsulated into C

language data structures. Kinetis device-specific configuration information is provided as part of the KSDK 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 Kinetis SDK folder structure is organized to reduce the total number of includes required to compile a project.

| Deliverable   | Location   |
|---|--|
| Examples  | <install_dir>/examples/                              |
| Demo Applications   | <install_dir>/examples/<board_name>/demo_apps/       |
| Driver Examples   | <install_dir>/examples/<board_name>/driver_examples/ |
| Documentation   | <install_dir>/docs/                                  |
| USB Documentation   | <install_dir>/docs/usb/                              |
| lwIP Documentation  | <install_dir>/docs/tcpip/lwip/                       |
| Middleware  | <install_dir>/middleware/                            |
| DMA Manager   | <install_dir>/dma_manager_<version>/                 |
| FatFs   | <install_dir>/middleware/fatfs_<version>             |
| lwIP TCP/IP   | <install_dir>/middleware/lwip_<version>/             |
| mmCAU   | <install_dir>/mmcau_<version>/                       |
| SDMMC Support   | <install_dir>/sdmmc_<version>/                       |
| USB Stack   | <install_dir>/middleware/usb_<version>               |
| Drivers   | <install_dir>/<device_name>/drivers/                 |
| CMSIS Standard ARM Cortex-M Headers, math and DSP Libraries | <install_dir>/<device_name>/CMSIS/                   |
| Device Startup and Linker                                   | <install_dir>/<device_name>/<toolchain>/             |
| KSDK Utilities  | <install_dir>/<device_name>/utilities/               |
| RTOS Kernels  | <install_dir>/rtos/                                  |

Table 2: KSDK Folder Structure

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 Kinetis SDK documents, see the [kex.nxp.com/apidoc](http://kex.nxp.com/apidoc).

## Chapter 2

### Driver errors status

- #kStatus\_DMA\_Busy = 5000
- kStatus\_SAI\_TxBusy = 1900
- kStatus\_SAI\_RxBusy = 1901
- kStatus\_SAI\_TxError = 1902
- kStatus\_SAI\_RxError = 1903
- kStatus\_SAI\_QueueFull = 1904
- kStatus\_SAI\_TxIdle = 1905
- kStatus\_SAI\_RxIdle = 1906
- kStatus\_SMC\_StopAbort = 3900
- kStatus\_SPI\_Busy = 1400
- kStatus\_SPI\_Idle = 1401
- kStatus\_SPI\_Error = 1402
- kStatus\_NOTIFIER\_ErrorNotificationBefore = 9800
- kStatus\_NOTIFIER\_ErrorNotificationAfter = 9801
- kStatus\_DMAMGR\_ChannelOccupied = 5200
- kStatus\_DMAMGR\_ChannelNotUsed = 5201
- kStatus\_DMAMGR\_NoFreeChannel = 5202
- kStatus\_DMAMGR\_ChannelNotMatchSource = 5203





## Chapter 3

### Architectural Overview

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

#### Overview

The Kinetis 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 Kinetis SDK
5. Demo Applications based on the Kinetis SDK

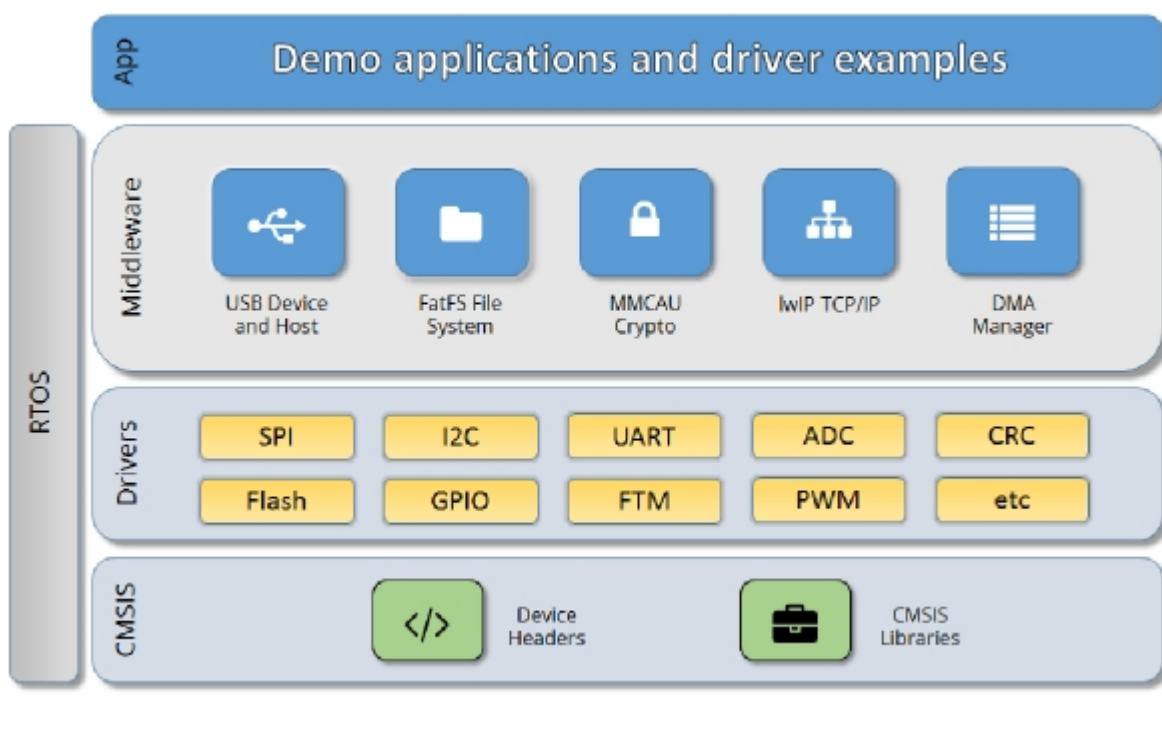


Figure 1: KSDK Block Diagram

#### Kinetis MCU header files

Each supported Kinetis MCU device in the KSDK 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 a access to the peripheral registers through pointers and predefined bit masks. In addition to the overall SoC memory-mapped header file, the KSDK 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 KSDK 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.

## **KSDK Peripheral Drivers**

The KSDK peripheral drivers mainly consist of low-level functional APIs for the Kinetis MCU product family on-chip peripherals and also of high-level transactional APIs for some bus drivers/DMA driver/e-DMA driver to quickly enable the peripherals and perform transfers.

All KSDK 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 KSDK 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 Kinetis devices. It's 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_Driver_IRQHandler
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 KSDK drivers with transactional APIs provide the reimplement of the second layer function inside of the peripheral driver. If the KSDK drivers with transactional APIs are linked into the image, the SPI0\_DriverIRQHandler is replaced with the function implemented in the KSDK 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 KSDK 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 Kinetis MCU device to another. An overall Peripheral Feature Header File is provided for the KSDK-supported MCU device to define the features or configuration differences for each Kinetis sub-family device.

## Application

See the *Getting Started with Kinetis SDK (KSDK) v2.0* document (KSDK20GSUG).



## Chapter 4

### Trademarks

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

How to Reach Us:

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

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

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 Freescale 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. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [freescale.com/SalesTermsandConditions](http://freescale.com/SalesTermsandConditions)

Freescale, the Freescale logo, Kinetis, and Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Tower is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM, ARM powered logo, Keil, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016 Freescale Semiconductors, Inc.



## Chapter 5

# ADC16: 16-bit SAR Analog-to-Digital Converter Driver

### 5.1 Overview

The KSDK provides a Peripheral driver for the 16-bit SAR Analog-to-Digital Converter (ADC16) module of Kinetis devices.

### 5.2 Typical use case

#### 5.2.1 Polling Configuration

```
adc16_config_t adc16ConfigStruct;
adc16_channel_config_t adc16ChannelConfigStruct;

ADC16_Init(DEMO_ADC16_INSTANCE);
ADC16_GetDefaultConfig(&adc16ConfigStruct);
ADC16_Configure(DEMO_ADC16_INSTANCE, &adc16ConfigStruct);
ADC16_EnableHardwareTrigger(DEMO_ADC16_INSTANCE, false);
#if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) && FSL_FEATURE_ADC16_HAS_CALIBRATION
if (kStatus_Success == ADC16_DoAutoCalibration(DEMO_ADC16_INSTANCE))
{
    PRINTF("ADC16_DoAutoCalibration() Done.\r\n");
}
else
{
    PRINTF("ADC16_DoAutoCalibration() Failed.\r\n");
}
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION

adc16ChannelConfigStruct.channelNumber = DEMO_ADC16_USER_CHANNEL;
adc16ChannelConfigStruct.enableInterruptOnConversionCompleted =
    false;
#if defined(FSL_FEATURE_ADC16_HAS_DIFF_MODE) && FSL_FEATURE_ADC16_HAS_DIFF_MODE
adc16ChannelConfigStruct.enableDifferentialConversion = false;
#endif // FSL_FEATURE_ADC16_HAS_DIFF_MODE

while(1)
{
    GETCHAR(); // Input any key in terminal console.
    ADC16_ChannelConfigure(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
    while (kADC16_ChannelConversionDoneFlag !=
        ADC16_ChannelGetStatusFlags(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP))
    {
    }
    PRINTF("ADC Value: %d\r\n", ADC16_ChannelGetConversionValue(DEMO_ADC16_INSTANCE,
        DEMO_ADC16_CHANNEL_GROUP));
}
```

#### 5.2.2 Interrupt Configuration

```
volatile bool g_Adc16ConversionDoneFlag = false;
volatile uint32_t g_Adc16ConversionValue;
volatile uint32_t g_Adc16InterruptCount = 0U;
```

## Typical use case

```
// ...

adc16_config_t adc16ConfigStruct;
adc16_channel_config_t adc16ChannelConfigStruct;

ADC16_Init (DEMO_ADC16_INSTANCE);
ADC16_GetDefaultConfig(&adc16ConfigStruct);
ADC16_Configure(DEMO_ADC16_INSTANCE, &adc16ConfigStruct);
ADC16_EnableHardwareTrigger(DEMO_ADC16_INSTANCE, false);
#if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) && FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (ADC16_DoAutoCalibration(DEMO_ADC16_INSTANCE))
    {
        PRINTF("ADC16_DoAutoCalibration() Done.\r\n");
    }
    else
    {
        PRINTF("ADC16_DoAutoCalibration() Failed.\r\n");
    }
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION

adc16ChannelConfigStruct.channelNumber = DEMO_ADC16_USER_CHANNEL;
adc16ChannelConfigStruct.enableInterruptOnConversionCompleted =
    true; // Enable the interrupt.
#if defined(FSL_FEATURE_ADC16_HAS_DIFF_MODE) && FSL_FEATURE_ADC16_HAS_DIFF_MODE
    adc16ChannelConfigStruct.enableDifferentialConversion = false;
#endif // FSL_FEATURE_ADC16_HAS_DIFF_MODE

while(1)
{
    GETCHAR(); // Input any key in terminal console.
    g_Adc16ConversionDoneFlag = false;
    ADC16_ChannelConfigure(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
    while (!g_Adc16ConversionDoneFlag)
    {
    }
    PRINTF("ADC Value: %d\r\n", g_Adc16ConversionValue);
    PRINTF("ADC Interrupt Count: %d\r\n", g_Adc16InterruptCount);
}

// ...

void DEMO_ADC16_IRQHandler(void)
{
    g_Adc16ConversionDoneFlag = true;
    // Read conversion result to clear the conversion completed flag.
    g_Adc16ConversionValue = ADC16_ChannelConversionValue(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP);
    g_Adc16InterruptCount++;
}
```

## Data Structures

- struct [adc16\\_config\\_t](#)  
*ADC16 converter configuration. [More...](#)*
- struct [adc16\\_hardware\\_compare\\_config\\_t](#)  
*ADC16 Hardware compare configuration. [More...](#)*
- struct [adc16\\_channel\\_config\\_t](#)  
*ADC16 channel conversion configuration. [More...](#)*

## Enumerations

- enum [\\_adc16\\_channel\\_status\\_flags](#) { [kADC16\\_ChannelConversionDoneFlag](#) = ADC\_SC1\_COCO\_MASK }



- *Channel status flags.*  
enum `_adc16_status_flags` { `kADC16_ActiveFlag` = `ADC_SC2_ADACT_MASK` }
- *Converter status flags.*  
enum `adc16_clock_divider_t` {  
  `kADC16_ClockDivider1` = 0U,  
  `kADC16_ClockDivider2` = 1U,  
  `kADC16_ClockDivider4` = 2U,  
  `kADC16_ClockDivider8` = 3U }
- *Clock divider for the converter.*  
enum `adc16_resolution_t` {  
  `kADC16_Resolution8or9Bit` = 0U,  
  `kADC16_Resolution12or13Bit` = 1U,  
  `kADC16_Resolution10or11Bit` = 2U,  
  `kADC16_ResolutionSE8Bit` = `kADC16_Resolution8or9Bit`,  
  `kADC16_ResolutionSE12Bit` = `kADC16_Resolution12or13Bit`,  
  `kADC16_ResolutionSE10Bit` = `kADC16_Resolution10or11Bit` }
- *Converter's resolution.*  
enum `adc16_clock_source_t` {  
  `kADC16_ClockSourceAlt0` = 0U,  
  `kADC16_ClockSourceAlt1` = 1U,  
  `kADC16_ClockSourceAlt2` = 2U,  
  `kADC16_ClockSourceAlt3` = 3U,  
  `kADC16_ClockSourceAsynchronousClock` = `kADC16_ClockSourceAlt3` }
- *Clock source.*  
enum `adc16_long_sample_mode_t` {  
  `kADC16_LongSampleCycle24` = 0U,  
  `kADC16_LongSampleCycle16` = 1U,  
  `kADC16_LongSampleCycle10` = 2U,  
  `kADC16_LongSampleCycle6` = 3U,  
  `kADC16_LongSampleDisabled` = 4U }
- *Long sample mode.*  
enum `adc16_reference_voltage_source_t` {  
  `kADC16_ReferenceVoltageSourceVref` = 0U,  
  `kADC16_ReferenceVoltageSourceValt` = 1U }
- *Reference voltage source.*  
enum `adc16_hardware_compare_mode_t` {  
  `kADC16_HardwareCompareMode0` = 0U,  
  `kADC16_HardwareCompareMode1` = 1U,  
  `kADC16_HardwareCompareMode2` = 2U,  
  `kADC16_HardwareCompareMode3` = 3U }
- *Hardware compare mode.*

## Driver version

- #define `FSL_ADC16_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 0))  
  ADC16 driver version 2.0.0.

### Initialization

- void [ADC16\\_Init](#) (ADC\_Type \*base, const [adc16\\_config\\_t](#) \*config)  
*Initializes the ADC16 module.*
- void [ADC16\\_Deinit](#) (ADC\_Type \*base)  
*De-initializes the ADC16 module.*
- void [ADC16\\_GetDefaultConfig](#) ([adc16\\_config\\_t](#) \*config)  
*Gets an available pre-defined settings for converter's configuration.*

### Advanced Feature

- static void [ADC16\\_EnableHardwareTrigger](#) (ADC\_Type \*base, bool enable)  
*Enables the hardware trigger mode.*
- void [ADC16\\_SetHardwareCompareConfig](#) (ADC\_Type \*base, const [adc16\\_hardware\\_compare\\_config\\_t](#) \*config)  
*Configures the hardware compare mode.*
- uint32\_t [ADC16\\_GetStatusFlags](#) (ADC\_Type \*base)  
*Gets the status flags of the converter.*
- void [ADC16\\_ClearStatusFlags](#) (ADC\_Type \*base, uint32\_t mask)  
*Clears the status flags of the converter.*

### Conversion Channel

- void [ADC16\\_SetChannelConfig](#) (ADC\_Type \*base, uint32\_t channelGroup, const [adc16\\_channel\\_config\\_t](#) \*config)  
*Configures the conversion channel.*
- static uint32\_t [ADC16\\_GetChannelConversionValue](#) (ADC\_Type \*base, uint32\_t channelGroup)  
*Gets the conversion value.*
- uint32\_t [ADC16\\_GetChannelStatusFlags](#) (ADC\_Type \*base, uint32\_t channelGroup)  
*Gets the status flags of channel.*

## 5.3 Data Structure Documentation

### 5.3.1 struct [adc16\\_config\\_t](#)

#### Data Fields

- [adc16\\_reference\\_voltage\\_source\\_t](#) [referenceVoltageSource](#)  
*Select the reference voltage source.*
- [adc16\\_clock\\_source\\_t](#) [clockSource](#)  
*Select the input clock source to converter.*
- bool [enableAsynchronousClock](#)  
*Enable the asynchronous clock output.*
- [adc16\\_clock\\_divider\\_t](#) [clockDivider](#)  
*Select the divider of input clock source.*
- [adc16\\_resolution\\_t](#) [resolution](#)  
*Select the sample resolution mode.*
- [adc16\\_long\\_sample\\_mode\\_t](#) [longSampleMode](#)  
*Select the long sample mode.*
- bool [enableHighSpeed](#)

- *Enable the high-speed mode.*  
bool [enableLowPower](#)
- *Enable low power.*  
bool [enableContinuousConversion](#)
- *Enable continuous conversion mode.*

#### 5.3.1.0.0.1 Field Documentation

5.3.1.0.0.1.1 `adc16_reference_voltage_source_t` `adc16_config_t::referenceVoltageSource`

5.3.1.0.0.1.2 `adc16_clock_source_t` `adc16_config_t::clockSource`

5.3.1.0.0.1.3 `bool` `adc16_config_t::enableAsynchronousClock`

5.3.1.0.0.1.4 `adc16_clock_divider_t` `adc16_config_t::clockDivider`

5.3.1.0.0.1.5 `adc16_resolution_t` `adc16_config_t::resolution`

5.3.1.0.0.1.6 `adc16_long_sample_mode_t` `adc16_config_t::longSampleMode`

5.3.1.0.0.1.7 `bool` `adc16_config_t::enableHighSpeed`

5.3.1.0.0.1.8 `bool` `adc16_config_t::enableLowPower`

5.3.1.0.0.1.9 `bool` `adc16_config_t::enableContinuousConversion`

#### 5.3.2 struct `adc16_hardware_compare_config_t`

##### Data Fields

- `adc16_hardware_compare_mode_t` `hardwareCompareMode`  
*Select the hardware compare mode.*
- `int16_t` `value1`  
*Setting value1 for hardware compare mode.*
- `int16_t` `value2`  
*Setting value2 for hardware compare mode.*

#### 5.3.2.0.0.2 Field Documentation

5.3.2.0.0.2.1 `adc16_hardware_compare_mode_t` `adc16_hardware_compare_config_t::hardwareCompareMode`

See "`adc16_hardware_compare_mode_t`".

## Enumeration Type Documentation

5.3.2.0.0.2.2 int16\_t adc16\_hardware\_compare\_config\_t::value1

5.3.2.0.0.2.3 int16\_t adc16\_hardware\_compare\_config\_t::value2

### 5.3.3 struct adc16\_channel\_config\_t

#### Data Fields

- uint32\_t [channelNumber](#)  
*Setting the conversion channel number.*
- bool [enableInterruptOnConversionCompleted](#)  
*Generate an interrupt request once the conversion is completed.*

#### 5.3.3.0.0.3 Field Documentation

5.3.3.0.0.3.1 uint32\_t adc16\_channel\_config\_t::channelNumber

The available range is 0-31. See channel connection information for each chip in Reference Manual document.

5.3.3.0.0.3.2 bool adc16\_channel\_config\_t::enableInterruptOnConversionCompleted

## 5.4 Macro Definition Documentation

5.4.1 #define FSL\_ADC16\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 5.5 Enumeration Type Documentation

5.5.1 enum \_adc16\_channel\_status\_flags

Enumerator

*kADC16\_ChannelConversionDoneFlag* Conversion done.

5.5.2 enum \_adc16\_status\_flags

Enumerator

*kADC16\_ActiveFlag* Converter is active.

5.5.3 enum adc16\_clock\_divider\_t

Enumerator

*kADC16\_ClockDivider1* For divider 1 from the input clock to the module.

***kADC16\_ClockDivider2*** For divider 2 from the input clock to the module.  
***kADC16\_ClockDivider4*** For divider 4 from the input clock to the module.  
***kADC16\_ClockDivider8*** For divider 8 from the input clock to the module.

#### 5.5.4 enum adc16\_resolution\_t

Enumerator

***kADC16\_Resolution8or9Bit*** Single End 8-bit or Differential Sample 9-bit.  
***kADC16\_Resolution12or13Bit*** Single End 12-bit or Differential Sample 13-bit.  
***kADC16\_Resolution10or11Bit*** Single End 10-bit or Differential Sample 11-bit.  
***kADC16\_ResolutionSE8Bit*** Single End 8-bit.  
***kADC16\_ResolutionSE12Bit*** Single End 12-bit.  
***kADC16\_ResolutionSE10Bit*** Single End 10-bit.

#### 5.5.5 enum adc16\_clock\_source\_t

Enumerator

***kADC16\_ClockSourceAlt0*** Selection 0 of the clock source.  
***kADC16\_ClockSourceAlt1*** Selection 1 of the clock source.  
***kADC16\_ClockSourceAlt2*** Selection 2 of the clock source.  
***kADC16\_ClockSourceAlt3*** Selection 3 of the clock source.  
***kADC16\_ClockSourceAsynchronousClock*** Using internal asynchronous clock.

#### 5.5.6 enum adc16\_long\_sample\_mode\_t

Enumerator

***kADC16\_LongSampleCycle24*** 20 extra ADCK cycles, 24 ADCK cycles total.  
***kADC16\_LongSampleCycle16*** 12 extra ADCK cycles, 16 ADCK cycles total.  
***kADC16\_LongSampleCycle10*** 6 extra ADCK cycles, 10 ADCK cycles total.  
***kADC16\_LongSampleCycle6*** 2 extra ADCK cycles, 6 ADCK cycles total.  
***kADC16\_LongSampleDisabled*** Disable the long sample feature.

#### 5.5.7 enum adc16\_reference\_voltage\_source\_t

Enumerator

***kADC16\_ReferenceVoltageSourceVref*** For external pins pair of VrefH and VrefL.  
***kADC16\_ReferenceVoltageSourceValt*** For alternate reference pair of ValtH and ValtL.

## Function Documentation

### 5.5.8 enum adc16\_hardware\_compare\_mode\_t

Enumerator

**kADC16\_HardwareCompareMode0**  $x < \text{value1}$ .  
**kADC16\_HardwareCompareMode1**  $x > \text{value1}$ .  
**kADC16\_HardwareCompareMode2** if  $\text{value1} \leq \text{value2}$ , then  $x < \text{value1} \parallel x > \text{value2}$ ; else,  $\text{value1} > x > \text{value2}$ .  
**kADC16\_HardwareCompareMode3** if  $\text{value1} \leq \text{value2}$ , then  $\text{value1} \leq x \leq \text{value2}$ ; else  $x \geq \text{value1} \parallel x \leq \text{value2}$ .

## 5.6 Function Documentation

### 5.6.1 void ADC16\_Init ( ADC\_Type \* *base*, const adc16\_config\_t \* *config* )

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | ADC16 peripheral base address.                            |
| <i>config</i> | Pointer to configuration structure. See "adc16_config_t". |

### 5.6.2 void ADC16\_Deinit ( ADC\_Type \* *base* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | ADC16 peripheral base address. |
|-------------|--------------------------------|

### 5.6.3 void ADC16\_GetDefaultConfig ( adc16\_config\_t \* *config* )

This function initializes the converter configuration structure with an available settings. The default values are:

```
* config->referenceVoltageSource = kADC16_ReferenceVoltageSourceVref
* ;
* config->clockSource = kADC16_ClockSourceAsynchronousClock
* ;
* config->enableAsynchronousClock = true;
* config->clockDivider = kADC16_ClockDivider8;
* config->resolution = kADC16_ResolutionSE12Bit;
* config->longSampleMode = kADC16_LongSampleDisabled;
* config->enableHighSpeed = false;
* config->enableLowPower = false;
* config->enableContinuousConversion = false;
*
```

## Parameters

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

#### 5.6.4 static void ADC16\_EnableHardwareTrigger ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | ADC16 peripheral base address.   |
| <i>enable</i> | Switcher of hardware trigger feature. "true" means to enable, "false" means not. |

#### 5.6.5 void ADC16\_SetHardwareCompareConfig ( ADC\_Type \* *base*, const adc16\_hardware\_compare\_config\_t \* *config* )

The hardware compare mode provides a way to process the conversion result automatically by hardware. Only the result in compare range is available. To compare the range, see "adc16\_hardware\_compare\_mode\_t", or the reference manual document for more detailed information.

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | ADC16 peripheral base address.  |
| <i>config</i> | Pointer to "adc16_hardware_compare_config_t" structure. Passing "NULL" is to disable the feature. |

#### 5.6.6 uint32\_t ADC16\_GetStatusFlags ( ADC\_Type \* *base* )

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | ADC16 peripheral base address. |
|-------------|--------------------------------|

## Returns

Flags' mask if indicated flags are asserted. See "\_adc16\_status\_flags".

#### 5.6.7 void ADC16\_ClearStatusFlags ( ADC\_Type \* *base*, uint32\_t *mask* )

## Function Documentation

### Parameters

|             |  |
|-------------|--|
| <i>base</i> | ADC16 peripheral base address.                               |
| <i>mask</i> | Mask value for the cleared flags. See "_adc16_status_flags". |

### 5.6.8 void ADC16\_SetChannelConfig ( ADC\_Type \* *base*, uint32\_t *channelGroup*, const adc16\_channel\_config\_t \* *config* )

This operation triggers the conversion if in software trigger mode. When in hardware trigger mode, this API configures the channel while the external trigger source helps to trigger the conversion.

Note that the "Channel Group" has a detailed description. To allow sequential conversions of the ADC to be triggered by internal peripherals, the ADC can have more than one group of status and control register, one for each conversion. The channel group parameter indicates which group of registers are used channel group 0 is for Group A registers and channel group 1 is for Group B registers. The channel groups are used in a "ping-pong" approach to control the ADC operation. At any point, only one of the channel groups is actively controlling ADC conversions. Channel group 0 is used for both software and hardware trigger modes of operation. Channel groups 1 and greater indicate potentially multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the MCU reference manual about the number of SC1n registers (channel groups) specific to this device. None of the channel groups 1 or greater are used for software trigger operation and therefore writes to these channel groups do not initiate a new conversion. Updating channel group 0 while a different channel group is actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

### Parameters

|                     |   |
|---------------------|---|
| <i>base</i>         | ADC16 peripheral base address.  |
| <i>channelGroup</i> | Channel group index.  |
| <i>config</i>       | Pointer to "adc16_channel_config_t" structure for conversion channel. |

### 5.6.9 static uint32\_t ADC16\_GetChannelConversionValue ( ADC\_Type \* *base*, uint32\_t *channelGroup* ) [inline], [static]

### Parameters



|                     |                                |
|---------------------|--------------------------------|
| <i>base</i>         | ADC16 peripheral base address. |
| <i>channelGroup</i> | Channel group index.           |

Returns

Conversion value.

#### 5.6.10 uint32\_t ADC16\_GetChannelStatusFlags ( ADC\_Type \* *base*, uint32\_t *channelGroup* )

Parameters

|                     |                                |
|---------------------|--------------------------------|
| <i>base</i>         | ADC16 peripheral base address. |
| <i>channelGroup</i> | Channel group index.           |

Returns

Flags' mask if indicated flags are asserted. See "\_adc16\_channel\_status\_flags".



## Chapter 6 Clock Driver

### 6.1 Overview

The KSDK provides APIs for Kinetis devices clock operation.

### 6.2 Get frequency

A centralized function `CLOCK_GetFreq` gets different clock type frequencies by passing a clock name. For example, pass a `kCLOCK_CoreSysClk` to get the core clock and pass a `kCLOCK_BusClk` to get the bus clock. Additionally, there are separate functions to get frequency, for example, use `CLOCK_GetCoreSysClkFreq` to get the core clock frequency and `CLOCK_GetBusClkFreq` to get the bus clock frequency. Using these functions reduces the image size.

### 6.3 External clock frequency

The external clocks `EXTAL0/EXTAL1/EXTAL32` are decided by the board level design. The Clock driver uses variables `g_xtal0Freq/g_xtal1Freq/g_xtal32Freq` to save clock frequencies. Likewise, the APIs `CLOCK_SetXtal0Freq`, `CLOCK_SetXtal1Freq` and `CLOCK_SetXtal32Freq` are used to set these variables.

The upper layer must set these values correctly, for example, after `OSC0(SYSOSC)` is initialized using `CLOCK_InitOsc0` or `CLOCK_InitSysOsc`, the upper layer should call the `CLOCK_SetXtal0Freq`. Otherwise, the clock frequency get functions may not get valid values. This is useful for multicore platforms where only one core calls `CLOCK_InitOsc0` to initialize `OSC0` and other cores call `CLOCK_SetXtal0Freq`.

### Modules

- [Multipurpose Clock Generator \(MCG\)](#)

### Files

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

### Data Structures

- struct [sim\\_clock\\_config\\_t](#)  
*SIM configuration structure for clock setting. [More...](#)*
- struct [oscer\\_config\\_t](#)  
*OSC configuration for OSCERCLK. [More...](#)*
- struct [osc\\_config\\_t](#)  
*OSC Initialization Configuration Structure. [More...](#)*
- struct [mcg\\_pll\\_config\\_t](#)  
*MCG PLL configuration. [More...](#)*
- struct [mcg\\_config\\_t](#)  
*MCG mode change configuration structure. [More...](#)*

### Macros

- #define [DMAMUX\\_CLOCKS](#)  
*Clock ip name array for DMAMUX.*
- #define [RTC\\_CLOCKS](#)  
*Clock ip name array for RTC.*
- #define [SAI\\_CLOCKS](#)  
*Clock ip name array for SAI.*
- #define [SPI\\_CLOCKS](#)  
*Clock ip name array for SPI.*
- #define [SLCD\\_CLOCKS](#)  
*Clock ip name array for SLCD.*
- #define [PIT\\_CLOCKS](#)  
*Clock ip name array for PIT.*
- #define [PORT\\_CLOCKS](#)  
*Clock ip name array for PORT.*
- #define [TSI\\_CLOCKS](#)  
*Clock ip name array for TSI.*
- #define [DAC\\_CLOCKS](#)  
*Clock ip name array for DAC.*
- #define [LPTMR\\_CLOCKS](#)  
*Clock ip name array for LPTMR.*
- #define [ADC16\\_CLOCKS](#)  
*Clock ip name array for ADC16.*
- #define [DMA\\_CLOCKS](#)  
*Clock ip name array for DMA.*
- #define [UART0\\_CLOCKS](#)  
*Clock ip name array for LPSCI/UART0.*
- #define [UART\\_CLOCKS](#)  
*Clock ip name array for UART.*
- #define [TPM\\_CLOCKS](#)  
*Clock ip name array for TPM.*
- #define [I2C\\_CLOCKS](#)  
*Clock ip name array for I2C.*
- #define [FTF\\_CLOCKS](#)  
*Clock ip name array for FTF.*
- #define [CMP\\_CLOCKS](#)  
*Clock ip name array for CMP.*
- #define [LPO\\_CLK\\_FREQ](#) 1000U  
*LPO clock frequency.*
- #define [SYS\\_CLK kCLOCK\\_CoreSysClk](#)  
*Peripherals clock source definition.*

## Enumerations

- enum `clock_name_t` {  
`kCLOCK_CoreSysClk`,  
`kCLOCK_PlatClk`,  
`kCLOCK_BusClk`,  
`kCLOCK_FlashClk`,  
`kCLOCK_PllFllSelClk`,  
`kCLOCK_Er32kClk`,  
`kCLOCK_Osc0ErClk`,  
`kCLOCK_McgFixedFreqClk`,  
`kCLOCK_McgInternalRefClk`,  
`kCLOCK_McgFllClk`,  
`kCLOCK_McgPll0Clk`,  
`kCLOCK_McgExtPllClk`,  
`kCLOCK_LpoClk` }  
*Clock name used to get clock frequency.*
- enum `clock_usb_src_t` {  
`kCLOCK_UsbSrcPll0` = `SIM_SOPT2_USBSRC(1U) | SIM_SOPT2_PLLFLLSEL(1U)`,  
`kCLOCK_UsbSrcExt` = `SIM_SOPT2_USBSRC(0U)` }  
*USB clock source definition.*
- enum `clock_ip_name_t`  
*Clock gate name used for `CLOCK_EnableClock/CLOCK_DisableClock`.*
- enum `osc_mode_t` {  
`kOSC_ModeExt` = `0U`,  
`kOSC_ModeOscLowPower` = `MCG_C2_EREFS0_MASK`,  
`kOSC_ModeOscHighGain` }  
*OSC work mode.*
- enum `_osc_cap_load` {  
`kOSC_Cap2P` = `OSC_CR_SC2P_MASK`,  
`kOSC_Cap4P` = `OSC_CR_SC4P_MASK`,  
`kOSC_Cap8P` = `OSC_CR_SC8P_MASK`,  
`kOSC_Cap16P` = `OSC_CR_SC16P_MASK` }  
*Oscillator capacitor load setting.*
- enum `_oscer_enable_mode` {  
`kOSC_ErClkEnable` = `OSC_CR_ERCLKEN_MASK`,  
`kOSC_ErClkEnableInStop` = `OSC_CR_EREFTSTEN_MASK` }  
*OSCERCLK enable mode.*
- enum `mcg_fll_src_t` {  
`kMCG_FllSrcExternal`,  
`kMCG_FllSrcInternal` }  
*MCG FLL reference clock source select.*
- enum `mcg_irc_mode_t` {  
`kMCG_IrcSlow`,  
`kMCG_IrcFast` }  
*MCG internal reference clock select.*
- enum `mcg_dmx32_t` {

## External clock frequency

kMCG\_Dmx32Default,  
kMCG\_Dmx32Fine }

*MCG DCO Maximum Frequency with 32.768 kHz Reference.*

- enum mcg\_drs\_t {  
kMCG\_DrsLow,  
kMCG\_DrsMid,  
kMCG\_DrsMidHigh,  
kMCG\_DrsHigh }

*MCG DCO range select.*

- enum mcg\_pll\_ref\_src\_t {  
kMCG\_PllRefOsc0,  
kMCG\_PllRefOsc1 }

*MCG PLL reference clock select.*

- enum mcg\_clkout\_src\_t {  
kMCG\_ClkOutSrcOut,  
kMCG\_ClkOutSrcInternal,  
kMCG\_ClkOutSrcExternal }

*MCGOUT clock source.*

- enum mcg\_atm\_select\_t {  
kMCG\_AtmSel32k,  
kMCG\_AtmSel4m }

*MCG Automatic Trim Machine Select.*

- enum mcg\_oscsel\_t {  
kMCG\_OscselOsc,  
kMCG\_OscselRtc }

*MCG OSC Clock Select.*

- enum mcg\_pll\_clk\_select\_t { kMCG\_PllClkSelPll0 }

*MCG PLLCS select.*

- enum mcg\_monitor\_mode\_t {  
kMCG\_MonitorNone,  
kMCG\_MonitorInt,  
kMCG\_MonitorReset }

*MCG clock monitor mode.*

- enum \_mcg\_status {  
kStatus\_MCG\_ModeUnreachable = MAKE\_STATUS(kStatusGroup\_MCG, 0),  
kStatus\_MCG\_ModeInvalid = MAKE\_STATUS(kStatusGroup\_MCG, 1),  
kStatus\_MCG\_AtmBusClockInvalid = MAKE\_STATUS(kStatusGroup\_MCG, 2),  
kStatus\_MCG\_AtmDesiredFreqInvalid = MAKE\_STATUS(kStatusGroup\_MCG, 3),  
kStatus\_MCG\_AtmIrcUsed = MAKE\_STATUS(kStatusGroup\_MCG, 4),  
kStatus\_MCG\_AtmHardwareFail = MAKE\_STATUS(kStatusGroup\_MCG, 5),  
kStatus\_MCG\_SourceUsed = MAKE\_STATUS(kStatusGroup\_MCG, 6) }

*MCG status.*

- enum \_mcg\_status\_flags\_t {  
kMCG\_Osc0LostFlag = (1U << 0U),  
kMCG\_Osc0InitFlag = (1U << 1U),  
kMCG\_Pll0LostFlag = (1U << 5U),  
kMCG\_Pll0LockFlag = (1U << 6U) }

- MCG status flags.*

  - enum `_mcg_irclk_enable_mode` {  
`kMCG_IrclkEnable` = `MCG_C1_IRCLKEN_MASK`,  
`kMCG_IrclkEnableInStop` = `MCG_C1_IREFSTEN_MASK` }

*MCG internal reference clock (MCGIRCLK) enable mode definition.*
- enum `_mcg_pll_enable_mode` {  
`kMCG_PllEnableIndependent` = `MCG_C5_PLLCLKEN0_MASK`,  
`kMCG_PllEnableInStop` = `MCG_C5_PLLSTEN0_MASK` }

*MCG PLL clock enable mode definition.*
- enum `mcg_mode_t` {  
`kMCG_ModeFEI` = 0U,  
`kMCG_ModeFBI`,  
`kMCG_ModeBLPI`,  
`kMCG_ModeFEE`,  
`kMCG_ModeFBE`,  
`kMCG_ModeBLPE`,  
`kMCG_ModePBE`,  
`kMCG_ModePEE`,  
`kMCG_ModeError` }

*MCG mode definitions.*

## Functions

- static void `CLOCK_EnableClock` (`clock_ip_name_t` name)  
*Enable the clock for specific IP.*
- static void `CLOCK_DisableClock` (`clock_ip_name_t` name)  
*Disable the clock for specific IP.*
- static void `CLOCK_SetEr32kClock` (`uint32_t` src)  
*Set ERCLK32K source.*
- static void `CLOCK_SetTpmClock` (`uint32_t` src)  
*Set TPM clock source.*
- static void `CLOCK_SetLpsci0Clock` (`uint32_t` src)  
*Set LPSCI0 (UART0) clock source.*
- static void `CLOCK_SetPllFllSelClock` (`uint32_t` src)  
*Set PLLFLLSEL clock source.*
- static void `CLOCK_SetClkOutClock` (`uint32_t` src)  
*Set CLKOUT source.*
- static void `CLOCK_SetRtcClkOutClock` (`uint32_t` src)  
*Set RTC\_CLKOUT source.*
- bool `CLOCK_EnableUsbfs0Clock` (`clock_usb_src_t` src, `uint32_t` freq)  
*Enable USB FS clock.*
- static void `CLOCK_DisableUsbfs0Clock` (void)  
*Disable USB FS clock.*
- static void `CLOCK_SetOutDiv` (`uint32_t` outdiv1, `uint32_t` outdiv4)  
*System clock divider.*
- `uint32_t` `CLOCK_GetFreq` (`clock_name_t` clockName)  
*Gets the clock frequency for a specific clock name.*
- `uint32_t` `CLOCK_GetCoreSysClkFreq` (void)  
*Get the core clock or system clock frequency.*

## External clock frequency

- uint32\_t [CLOCK\\_GetPlatClkFreq](#) (void)  
*Get the platform clock frequency.*
- uint32\_t [CLOCK\\_GetBusClkFreq](#) (void)  
*Get the bus clock frequency.*
- uint32\_t [CLOCK\\_GetFlashClkFreq](#) (void)  
*Get the flash clock frequency.*
- uint32\_t [CLOCK\\_GetPllFllSelClkFreq](#) (void)  
*Get the output clock frequency selected by SIM[PLL\_FLLSEL].*
- uint32\_t [CLOCK\\_GetEr32kClkFreq](#) (void)  
*Get the external reference 32K clock frequency (ERCLK32K).*
- uint32\_t [CLOCK\\_GetOsc0ErClkFreq](#) (void)  
*Get the OSC0 external reference clock frequency (OSC0ERCLK).*
- void [CLOCK\\_SetSimConfig](#) (sim\_clock\_config\_t const \*config)  
*Set the clock configure in SIM module.*
- static void [CLOCK\\_SetSimSafeDivs](#) (void)  
*Set the system clock dividers in SIM to safe value.*

## Variables

- uint32\_t [g\\_xtal0Freq](#)  
*External XTAL0 (OSC0) clock frequency.*
- uint32\_t [g\\_xtal32Freq](#)  
*External XTAL32/EXTAL32/RTC\_CLKIN clock frequency.*

## Driver version

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

## MCG frequency functions.

- uint32\_t [CLOCK\\_GetOutClkFreq](#) (void)  
*Gets the MCG output clock (MCGOUTCLK) frequency.*
- uint32\_t [CLOCK\\_GetFllFreq](#) (void)  
*Gets the MCG FLL clock (MCGFLLCLK) frequency.*
- uint32\_t [CLOCK\\_GetInternalRefClkFreq](#) (void)  
*Gets the MCG internal reference clock (MCGIRCLK) frequency.*
- uint32\_t [CLOCK\\_GetFixedFreqClkFreq](#) (void)  
*Gets the MCG fixed frequency clock (MCGFFCLK) frequency.*
- uint32\_t [CLOCK\\_GetPll0Freq](#) (void)  
*Gets the MCG PLL0 clock (MCGPLL0CLK) frequency.*

## MCG clock configuration.

- static void [CLOCK\\_SetLowPowerEnable](#) (bool enable)  
*Enables or disables the MCG low power.*
- status\_t [CLOCK\\_SetInternalRefClkConfig](#) (uint8\_t enableMode, mcg\_irc\_mode\_t ircs, uint8\_t fcr-div)  
*Configures the Internal Reference clock (MCGIRCLK).*
- status\_t [CLOCK\\_SetExternalRefClkConfig](#) (mcg\_oscsel\_t oscsel)



- *Selects the MCG external reference clock.*
- void [CLOCK\\_EnablePll0](#) ([mcg\\_pll\\_config\\_t](#) const \*config)  
*Enables the PLL0 in FLL mode.*
- static void [CLOCK\\_DisablePll0](#) (void)  
*Disables the PLL0 in FLL mode.*
- uint32\_t [CLOCK\\_CalcPllDiv](#) (uint32\_t refFreq, uint32\_t desireFreq, uint8\_t \*prdiv, uint8\_t \*vdiv)  
*Calculates the PLL divider setting for a desired output frequency.*

### MCG clock lock monitor functions.

- void [CLOCK\\_SetOsc0MonitorMode](#) ([mcg\\_monitor\\_mode\\_t](#) mode)  
*Sets the OSC0 clock monitor mode.*
- void [CLOCK\\_SetPll0MonitorMode](#) ([mcg\\_monitor\\_mode\\_t](#) mode)  
*Sets the PLL0 clock monitor mode.*
- uint32\_t [CLOCK\\_GetStatusFlags](#) (void)  
*Gets the MCG status flags.*
- void [CLOCK\\_ClearStatusFlags](#) (uint32\_t mask)  
*Clears the MCG status flags.*

### OSC configuration

- static void [OSC\\_SetExtRefClkConfig](#) (OSC\_Type \*base, [oscer\\_config\\_t](#) const \*config)  
*Configures the OSC external reference clock (OSCERCLK).*
- static void [OSC\\_SetCapLoad](#) (OSC\_Type \*base, uint8\_t capLoad)  
*Sets the capacitor load configuration for the oscillator.*
- void [CLOCK\\_InitOsc0](#) ([osc\\_config\\_t](#) const \*config)  
*Initializes the OSC0.*
- void [CLOCK\\_DeinitOsc0](#) (void)  
*Deinitializes the OSC0.*

### External clock frequency

- static void [CLOCK\\_SetXtal0Freq](#) (uint32\_t freq)  
*Sets the XTAL0 frequency based on board settings.*
- static void [CLOCK\\_SetXtal32Freq](#) (uint32\_t freq)  
*Sets the XTAL32/RTC\_CLKIN frequency based on board settings.*

### MCG auto-trim machine.

- status\_t [CLOCK\\_TrimInternalRefClk](#) (uint32\_t extFreq, uint32\_t desireFreq, uint32\_t \*actualFreq, [mcg\\_atm\\_select\\_t](#) atms)  
*Auto trims the internal reference clock.*

### MCG mode functions.

- [mcg\\_mode\\_t](#) [CLOCK\\_GetMode](#) (void)  
*Gets the current MCG mode.*
- status\_t [CLOCK\\_SetFeiMode](#) ([mcg\\_dm32\\_t](#) dm32, [mcg\\_drs\\_t](#) drs, void(\*flStableDelay)(void))  
*Sets the MCG to FEI mode.*

## Data Structure Documentation

- status\_t [CLOCK\\_SetFeeMode](#) (uint8\_t frdiv, [mcg\\_dmx32\\_t](#) dmx32, [mcg\\_drs\\_t](#) drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FEE mode.*
- status\_t [CLOCK\\_SetFbiMode](#) ([mcg\\_dmx32\\_t](#) dmx32, [mcg\\_drs\\_t](#) drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FBI mode.*
- status\_t [CLOCK\\_SetFbeMode](#) (uint8\_t frdiv, [mcg\\_dmx32\\_t](#) dmx32, [mcg\\_drs\\_t](#) drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FBE mode.*
- status\_t [CLOCK\\_SetBlpiMode](#) (void)  
*Sets the MCG to BLPI mode.*
- status\_t [CLOCK\\_SetBlpeMode](#) (void)  
*Sets the MCG to BLPE mode.*
- status\_t [CLOCK\\_SetPbeMode](#) ([mcg\\_pll\\_clk\\_select\\_t](#) pllcs, [mcg\\_pll\\_config\\_t](#) const \*config)  
*Sets the MCG to PBE mode.*
- status\_t [CLOCK\\_SetPeeMode](#) (void)  
*Sets the MCG to PEE mode.*
- status\_t [CLOCK\\_ExternalModeToFbeModeQuick](#) (void)  
*Switches the MCG to FBE mode from the external mode.*
- status\_t [CLOCK\\_InternalModeToFbiModeQuick](#) (void)  
*Switches the MCG to FBI mode from internal modes.*
- status\_t [CLOCK\\_BootToFeiMode](#) ([mcg\\_dmx32\\_t](#) dmx32, [mcg\\_drs\\_t](#) drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FEI mode during system boot up.*
- status\_t [CLOCK\\_BootToFeeMode](#) ([mcg\\_oscsel\\_t](#) oscsel, uint8\_t frdiv, [mcg\\_dmx32\\_t](#) dmx32, [mcg\\_drs\\_t](#) drs, void(\*fllStableDelay)(void))  
*Sets the MCG to FEE mode during system bootup.*
- status\_t [CLOCK\\_BootToBlpiMode](#) (uint8\_t frdiv, [mcg\\_irc\\_mode\\_t](#) ircs, uint8\_t ircEnableMode)  
*Sets the MCG to BLPI mode during system boot up.*
- status\_t [CLOCK\\_BootToBlpeMode](#) ([mcg\\_oscsel\\_t](#) oscsel)  
*Sets the MCG to BLPE mode during sytem boot up.*
- status\_t [CLOCK\\_BootToPeeMode](#) ([mcg\\_oscsel\\_t](#) oscsel, [mcg\\_pll\\_clk\\_select\\_t](#) pllcs, [mcg\\_pll\\_config\\_t](#) const \*config)  
*Sets the MCG to PEE mode during system boot up.*
- status\_t [CLOCK\\_SetMcgConfig](#) ([mcg\\_config\\_t](#) const \*config)  
*Sets the MCG to a target mode.*

## 6.4 Data Structure Documentation

### 6.4.1 struct sim\_clock\_config\_t

#### Data Fields

- uint8\_t [pllFllSel](#)  
*PLL/FLL selection.*
- uint8\_t [er32kSrc](#)  
*ERCLK32K source selection.*
- uint32\_t [clkdiv1](#)  
*SIM\_CLKDIV1.*

#### 6.4.1.0.0.4 Field Documentation

6.4.1.0.0.4.1 `uint8_t sim_clock_config_t::pllFilSel`

6.4.1.0.0.4.2 `uint8_t sim_clock_config_t::er32kSrc`

6.4.1.0.0.4.3 `uint32_t sim_clock_config_t::clkdiv1`

### 6.4.2 `struct oscr_config_t`

#### Data Fields

- `uint8_t enableMode`  
*OSCERCLK enable mode.*

#### 6.4.2.0.0.5 Field Documentation

6.4.2.0.0.5.1 `uint8_t oscr_config_t::enableMode`

OR'ed value of `_oscer_enable_mode`.

### 6.4.3 `struct osc_config_t`

Defines the configuration data structure to initialize the OSC. When porting to a new board, set the following members according to the board setting:

1. `freq`: The external frequency.
2. `workMode`: The OSC module mode.

#### Data Fields

- `uint32_t freq`  
*External clock frequency.*
- `uint8_t capLoad`  
*Capacitor load setting.*
- `osc_mode_t workMode`  
*OSC work mode setting.*
- `oscer_config_t oscrConfig`  
*Configuration for OSCERCLK.*

## Data Structure Documentation

### 6.4.3.0.0.6 Field Documentation

6.4.3.0.0.6.1 `uint32_t osc_config_t::freq`

6.4.3.0.0.6.2 `uint8_t osc_config_t::capLoad`

6.4.3.0.0.6.3 `osc_mode_t osc_config_t::workMode`

6.4.3.0.0.6.4 `oscer_config_t osc_config_t::oscerConfig`

### 6.4.4 `struct mcg_pll_config_t`

#### Data Fields

- `uint8_t enableMode`  
*Enable mode.*
- `uint8_t prdiv`  
*Reference divider PRDIV.*
- `uint8_t vdiv`  
*VCO divider VDIV.*

### 6.4.4.0.0.7 Field Documentation

6.4.4.0.0.7.1 `uint8_t mcg_pll_config_t::enableMode`

OR'ed value of `_mcg_pll_enable_mode`.

6.4.4.0.0.7.2 `uint8_t mcg_pll_config_t::prdiv`

6.4.4.0.0.7.3 `uint8_t mcg_pll_config_t::vdiv`

### 6.4.5 `struct mcg_config_t`

When porting to a new board, set the following members according to the board setting:

1. `frdiv`: If the FLL uses the external reference clock, set this value to ensure that the external reference clock divided by `frdiv` is in the 31.25 kHz to 39.0625 kHz range.
2. The PLL reference clock divider `PRDIV`: PLL reference clock frequency after `PRDIV` should be in the `FSL_FEATURE_MCG_PLL_REF_MIN` to `FSL_FEATURE_MCG_PLL_REF_MAX` range.

#### Data Fields

- `mcg_mode_t mcgMode`  
*MCG mode.*
- `uint8_t irclkEnableMode`  
*MCGIRCLK enable mode.*
- `mcg_irc_mode_t ircs`  
*Source, MCG\_C2[IRCS].*

- `uint8_t fcrdiv`  
*Divider, MCG\_SC[FCRDIV].*
- `uint8_t frdiv`  
*Divider MCG\_C1[FRDIV].*
- `mcg_drs_t drs`  
*DCO range MCG\_C4[DRST\_DRS].*
- `mcg_dmx32_t dmx32`  
*MCG\_C4[DMX32].*
- `mcg_pll_config_t pll0Config`  
*MCGPLL0CLK configuration.*

#### 6.4.5.0.0.8 Field Documentation

6.4.5.0.0.8.1 `mcg_mode_t mcg_config_t::mcgMode`

6.4.5.0.0.8.2 `uint8_t mcg_config_t::ircclkEnableMode`

6.4.5.0.0.8.3 `mcg_irc_mode_t mcg_config_t::ircs`

6.4.5.0.0.8.4 `uint8_t mcg_config_t::fcrdiv`

6.4.5.0.0.8.5 `uint8_t mcg_config_t::frdiv`

6.4.5.0.0.8.6 `mcg_drs_t mcg_config_t::drs`

6.4.5.0.0.8.7 `mcg_dmx32_t mcg_config_t::dmx32`

6.4.5.0.0.8.8 `mcg_pll_config_t mcg_config_t::pll0Config`

### 6.5 Macro Definition Documentation

6.5.1 `#define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

6.5.2 `#define DMAMUX_CLOCKS`

Value:

```
{
    \
    kCLOCK_Dmamux0 \
}
```

6.5.3 `#define RTC_CLOCKS`

Value:

```
{
    \
    kCLOCK_Rtc0 \
}
```

## Macro Definition Documentation

### 6.5.4 #define SAI\_CLOCKS

**Value:**

```
{  
    \kCLOCK_Sai0 \  
}
```

### 6.5.5 #define SPI\_CLOCKS

**Value:**

```
{  
    \kCLOCK_Spi0, kCLOCK_Spi1 \  
}
```

### 6.5.6 #define SLCD\_CLOCKS

**Value:**

```
{  
    \kCLOCK_Slcd0 \  
}
```

### 6.5.7 #define PIT\_CLOCKS

**Value:**

```
{  
    \kCLOCK_Pit0 \  
}
```

### 6.5.8 #define PORT\_CLOCKS

**Value:**

```
{  
    \kCLOCK_PortA, kCLOCK_PortB, kCLOCK_PortC, kCLOCK_PortD, kCLOCK_PortE \  
}
```

### 6.5.9 #define TSI\_CLOCKS

Value:

```
{  
    \kCLOCK_Tsi0 \  
}
```

### 6.5.10 #define DAC\_CLOCKS

Value:

```
{  
    \kCLOCK_Dac0 \  
}
```

### 6.5.11 #define LPTMR\_CLOCKS

Value:

```
{  
    \kCLOCK_Lptmr0 \  
}
```

### 6.5.12 #define ADC16\_CLOCKS

Value:

```
{  
    \kCLOCK_Adc0 \  
}
```

### 6.5.13 #define DMA\_CLOCKS

Value:

```
{  
    \kCLOCK_Dma0 \  
}
```

### 6.5.14 #define UART0\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Uart0 \
}
```

### 6.5.15 #define UART\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_IpInvalid, kCLOCK_Uart1, kCLOCK_Uart2 \
}
```

### 6.5.16 #define TPM\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Tpm0, kCLOCK_Tpm1, kCLOCK_Tpm2 \
}
```

### 6.5.17 #define I2C\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_I2c0, kCLOCK_I2c1 \
}
```

### 6.5.18 #define FTF\_CLOCKS

**Value:**

```
{
    \
    kCLOCK_Ftf0 \
}
```



### 6.5.19 #define CMP\_CLOCKS

Value:

```
{
    \
    kCLOCK_Cmp0 \
}
```

### 6.5.20 #define SYS\_CLK kCLOCK\_CoreSysClk

## 6.6 Enumeration Type Documentation

### 6.6.1 enum clock\_name\_t

Enumerator

*kCLOCK\_CoreSysClk* Core/system clock.  
*kCLOCK\_PlatClk* Platform clock.  
*kCLOCK\_BusClk* Bus clock.  
*kCLOCK\_FlashClk* Flash clock.  
*kCLOCK\_PllFllSelClk* The clock after SIM[PLL\_FLLSEL].  
*kCLOCK\_Er32kClk* External reference 32K clock (ERCLK32K)  
*kCLOCK\_Osc0ErClk* OSC0 external reference clock (OSC0ERCLK)  
*kCLOCK\_McgFixedFreqClk* MCG fixed frequency clock (MCGFFCLK)  
*kCLOCK\_McgInternalRefClk* MCG internal reference clock (MCGIRCLK)  
*kCLOCK\_McgFllClk* MCGFLLCLK.  
*kCLOCK\_McgPll0Clk* MCGPLL0CLK.  
*kCLOCK\_McgExtPllClk* EXT\_PLLCLK.  
*kCLOCK\_LpoClk* LPO clock.

### 6.6.2 enum clock\_usb\_src\_t

Enumerator

*kCLOCK\_UsbSrcPll0* Use PLL0.  
*kCLOCK\_UsbSrcExt* Use USB\_CLKIN.

### 6.6.3 enum clock\_ip\_name\_t

### 6.6.4 enum osc\_mode\_t

Enumerator

*kOSC\_ModeExt* Use an external clock.

## Enumeration Type Documentation

***kOSC\_ModeOscLowPower*** Oscillator low power.

***kOSC\_ModeOscHighGain*** Oscillator high gain.

### 6.6.5 enum \_osc\_cap\_load

Enumerator

***kOSC\_Cap2P*** 2 pF capacitor load

***kOSC\_Cap4P*** 4 pF capacitor load

***kOSC\_Cap8P*** 8 pF capacitor load

***kOSC\_Cap16P*** 16 pF capacitor load

### 6.6.6 enum \_oscer\_enable\_mode

Enumerator

***kOSC\_ErClkEnable*** Enable.

***kOSC\_ErClkEnableInStop*** Enable in stop mode.

### 6.6.7 enum mcg\_fl\_src\_t

Enumerator

***kMCG\_FlSrcExternal*** External reference clock is selected.

***kMCG\_FlSrcInternal*** The slow internal reference clock is selected.

### 6.6.8 enum mcg\_irc\_mode\_t

Enumerator

***kMCG\_IrcSlow*** Slow internal reference clock selected.

***kMCG\_IrcFast*** Fast internal reference clock selected.

### 6.6.9 enum mcg\_dmx32\_t

Enumerator

***kMCG\_Dmx32Default*** DCO has a default range of 25%.

***kMCG\_Dmx32Fine*** DCO is fine-tuned for maximum frequency with 32.768 kHz reference.

### 6.6.10 enum mcg\_drs\_t

Enumerator

*kMCG\_DrsLow* Low frequency range.  
*kMCG\_DrsMid* Mid frequency range.  
*kMCG\_DrsMidHigh* Mid-High frequency range.  
*kMCG\_DrsHigh* High frequency range.

### 6.6.11 enum mcg\_pll\_ref\_src\_t

Enumerator

*kMCG\_PllRefOsc0* Selects OSC0 as PLL reference clock.  
*kMCG\_PllRefOsc1* Selects OSC1 as PLL reference clock.

### 6.6.12 enum mcg\_clkout\_src\_t

Enumerator

*kMCG\_ClkOutSrcOut* Output of the FLL is selected (reset default)  
*kMCG\_ClkOutSrcInternal* Internal reference clock is selected.  
*kMCG\_ClkOutSrcExternal* External reference clock is selected.

### 6.6.13 enum mcg\_atm\_select\_t

Enumerator

*kMCG\_AtmSel32k* 32 kHz Internal Reference Clock selected  
*kMCG\_AtmSel4m* 4 MHz Internal Reference Clock selected

### 6.6.14 enum mcg\_oscsel\_t

Enumerator

*kMCG\_OscselOsc* Selects System Oscillator (OSCCLK)  
*kMCG\_OscselRtc* Selects 32 kHz RTC Oscillator.

## Enumeration Type Documentation

### 6.6.15 enum mcg\_pll\_clk\_select\_t

Enumerator

*kMCG\_PllClkSelPll0* PLL0 output clock is selected.

### 6.6.16 enum mcg\_monitor\_mode\_t

Enumerator

*kMCG\_MonitorNone* Clock monitor is disabled.

*kMCG\_MonitorInt* Trigger interrupt when clock lost.

*kMCG\_MonitorReset* System reset when clock lost.

### 6.6.17 enum \_mcg\_status

Enumerator

*kStatus\_MCG\_ModeUnreachable* Can't switch to target mode.

*kStatus\_MCG\_ModeInvalid* Current mode invalid for the specific function.

*kStatus\_MCG\_AtmBusClockInvalid* Invalid bus clock for ATM.

*kStatus\_MCG\_AtmDesiredFreqInvalid* Invalid desired frequency for ATM.

*kStatus\_MCG\_AtmIrcUsed* IRC is used when using ATM.

*kStatus\_MCG\_AtmHardwareFail* Hardware fail occurs during ATM.

*kStatus\_MCG\_SourceUsed* Can't change the clock source because it is in use.

### 6.6.18 enum \_mcg\_status\_flags\_t

Enumerator

*kMCG\_Osc0LostFlag* OSC0 lost.

*kMCG\_Osc0InitFlag* OSC0 crystal initialized.

*kMCG\_Pll0LostFlag* PLL0 lost.

*kMCG\_Pll0LockFlag* PLL0 locked.

### 6.6.19 enum \_mcg\_ircclk\_enable\_mode

Enumerator

*kMCG\_IrcclkEnable* MCGIRCLK enable.

*kMCG\_IrcclkEnableInStop* MCGIRCLK enable in stop mode.

### 6.6.20 enum \_mcg\_pll\_enable\_mode

Enumerator

***kMCG\_PllEnableIndependent*** MCGPLLCLK enable independent of the MCG clock mode. Generally, the PLL is disabled in FLL modes (FEI/FBI/FEE/FBE). Setting the PLL clock enable independent, enables the PLL in the FLL modes.

***kMCG\_PllEnableInStop*** MCGPLLCLK enable in STOP mode.

### 6.6.21 enum mcg\_mode\_t

Enumerator

***kMCG\_ModeFEI*** FEI - FLL Engaged Internal.

***kMCG\_ModeFBI*** FBI - FLL Bypassed Internal.

***kMCG\_ModeBLPI*** BLPI - Bypassed Low Power Internal.

***kMCG\_ModeFEE*** FEE - FLL Engaged External.

***kMCG\_ModeFBE*** FBE - FLL Bypassed External.

***kMCG\_ModeBLPE*** BLPE - Bypassed Low Power External.

***kMCG\_ModePBE*** PBE - PLL Bypassed External.

***kMCG\_ModePEE*** PEE - PLL Engaged External.

***kMCG\_ModeError*** Unknown mode.

## 6.7 Function Documentation

### 6.7.1 static void CLOCK\_EnableClock ( clock\_ip\_name\_t *name* ) [inline], [static]

Parameters

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

### 6.7.2 static void CLOCK\_DisableClock ( clock\_ip\_name\_t *name* ) [inline], [static]

Parameters

---

## Function Documentation

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

### 6.7.3 static void CLOCK\_SetEr32kClock ( uint32\_t src ) [inline], [static]

Parameters

|            |   |
|------------|---|
| <i>src</i> | The value to set ERCLK32K clock source. |
|------------|---|

### 6.7.4 static void CLOCK\_SetTpmClock ( uint32\_t src ) [inline], [static]

Parameters

|            |                                    |
|------------|------------------------------------|
| <i>src</i> | The value to set TPM clock source. |
|------------|------------------------------------|

### 6.7.5 static void CLOCK\_SetLpsci0Clock ( uint32\_t src ) [inline], [static]

Parameters

|            |  |
|------------|--|
| <i>src</i> | The value to setSet LPSCI0 (UART0) clock source. |
|------------|--|

### 6.7.6 static void CLOCK\_SetPllFllSelClock ( uint32\_t src ) [inline], [static]

Parameters

|            |  |
|------------|--|
| <i>src</i> | The value to set PLLFLLSEL clock source. |
|------------|--|

### 6.7.7 static void CLOCK\_SetClkOutClock ( uint32\_t src ) [inline], [static]

Parameters

---

|            |                                 |
|------------|---------------------------------|
| <i>src</i> | The value to set CLKOUT source. |
|------------|---------------------------------|

### 6.7.8 static void CLOCK\_SetRtcClkOutClock ( uint32\_t *src* ) [inline], [static]

Parameters

|            |                                     |
|------------|-------------------------------------|
| <i>src</i> | The value to set RTC_CLKOUT source. |
|------------|-------------------------------------|

### 6.7.9 bool CLOCK\_EnableUsbfs0Clock ( clock\_usb\_src\_t *src*, uint32\_t *freq* )

Parameters

|             |   |
|-------------|---|
| <i>src</i>  | USB FS clock source.                    |
| <i>freq</i> | The frequency specified by <i>src</i> . |

Return values

|              |   |
|--------------|---|
| <i>true</i>  | The clock is set successfully.                          |
| <i>false</i> | The clock source is invalid to get proper USB FS clock. |

### 6.7.10 static void CLOCK\_DisableUsbfs0Clock ( void ) [inline], [static]

Disable USB FS clock.

### 6.7.11 static void CLOCK\_SetOutDiv ( uint32\_t *outdiv1*, uint32\_t *outdiv4* ) [inline], [static]

Set the SIM\_CLKDIV1[OUTDIV1], SIM\_CLKDIV1[OUTDIV4].

Parameters

---

## Function Documentation

|                |                               |
|----------------|-------------------------------|
| <i>outdiv1</i> | Clock 1 output divider value. |
| <i>outdiv4</i> | Clock 4 output divider value. |

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

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock\_name\_t. The MCG must be properly configured before using this function.

Parameters

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

Returns

Clock frequency value in Hertz

### 6.7.13 uint32\_t CLOCK\_GetCoreSysClkFreq ( void )

Returns

Clock frequency in Hz.

### 6.7.14 uint32\_t CLOCK\_GetPlatClkFreq ( void )

Returns

Clock frequency in Hz.

### 6.7.15 uint32\_t CLOCK\_GetBusClkFreq ( void )

Returns

Clock frequency in Hz.

### 6.7.16 uint32\_t CLOCK\_GetFlashClkFreq ( void )

Returns

Clock frequency in Hz.



**6.7.17 uint32\_t CLOCK\_GetPIIFllSelClkFreq ( void )**

Returns

Clock frequency in Hz.

**6.7.18 uint32\_t CLOCK\_GetEr32kClkFreq ( void )**

Returns

Clock frequency in Hz.

**6.7.19 uint32\_t CLOCK\_GetOsc0ErClkFreq ( void )**

Returns

Clock frequency in Hz.

**6.7.20 void CLOCK\_SetSimConfig ( sim\_clock\_config\_t const \* *config* )**

This function sets system layer clock settings in SIM module.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>config</i> | Pointer to the configure structure. |
|---------------|-------------------------------------|

**6.7.21 static void CLOCK\_SetSimSafeDivs ( void ) [inline], [static]**

The system level clocks (core clock, bus clock, flexbus clock and flash clock) must be in allowed ranges. During MCG clock mode switch, the MCG output clock changes then the system level clocks may be out of range. This function could be used before MCG mode change, to make sure system level clocks are in allowed range.

Parameters

## Function Documentation

|               |                                     |
|---------------|-------------------------------------|
| <i>config</i> | Pointer to the configure structure. |
|---------------|-------------------------------------|

### 6.7.22 uint32\_t CLOCK\_GetOutClkFreq ( void )

This function gets the MCG output clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGOUTCLK.

### 6.7.23 uint32\_t CLOCK\_GetFllFreq ( void )

This function gets the MCG FLL clock frequency in Hz based on the current MCG register value. The FLL is enabled in FEI/FBI/FEE/FBE mode and disabled in low power state in other modes.

Returns

The frequency of MCGFLLCLK.

### 6.7.24 uint32\_t CLOCK\_GetInternalRefClkFreq ( void )

This function gets the MCG internal reference clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGIRCLK.

### 6.7.25 uint32\_t CLOCK\_GetFixedFreqClkFreq ( void )

This function gets the MCG fixed frequency clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGFFCLK.

### 6.7.26 `uint32_t CLOCK_GetPll0Freq ( void )`

This function gets the MCG PLL0 clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGPLL0CLK.

### 6.7.27 `static void CLOCK_SetLowPowerEnable ( bool enable ) [inline], [static]`

Enabling the MCG low power disables the PLL and FLL in bypass modes. In other words, in FBE and PBE modes, enabling low power sets the MCG to BLPE mode. In FBI and PBI modes, enabling low power sets the MCG to BLPI mode. When disabling the MCG low power, the PLL or FLL are enabled based on MCG settings.

Parameters

|               |   |
|---------------|---|
| <i>enable</i> | True to enable MCG low power, false to disable MCG low power. |
|---------------|---|

### 6.7.28 `status_t CLOCK_SetInternalRefClkConfig ( uint8_t enableMode, mcg_irc_mode_t ircs, uint8_t fcrdiv )`

This function sets the MCGIRCLK base on parameters. It also selects the IRC source. If the fast IRC is used, this function sets the fast IRC divider. This function also sets whether the MCGIRCLK is enabled in stop mode. Calling this function in FBI/PBI/BLPI modes may change the system clock. As a result, using the function in these modes it is not allowed.

Parameters

|                   |  |
|-------------------|--|
| <i>enableMode</i> | MCGIRCLK enable mode, OR'ed value of <a href="#">_mcg_ircclk_enable_mode</a> . |
| <i>ircs</i>       | MCGIRCLK clock source, choose fast or slow.                                    |
| <i>fcrdiv</i>     | Fast IRC divider setting (FCRDIV).   |

Return values

---

## Function Documentation

|                                |  |
|--------------------------------|--|
| <i>kStatus_MCG_Source-Used</i> | Because the internal reference clock is used as a clock source, the configuration should not be changed. Otherwise, a glitch occurs. |
| <i>kStatus_Success</i>         | MCGIRCLK configuration finished successfully.  |

### 6.7.29 **status\_t** CLOCK\_SetExternalRefClkConfig ( **mcg\_oscsel\_t** *oscsel* )

Selects the MCG external reference clock source, changes the MCG\_C7[OSCSEL], and waits for the clock source to be stable. Because the external reference clock should not be changed in FEE/FBE/BLP-E/PBE/PEE modes, do not call this function in these modes.

Parameters

|               |  |
|---------------|--|
| <i>oscsel</i> | MCG external reference clock source, MCG_C7[OSCSEL]. |
|---------------|--|

Return values

|                                |  |
|--------------------------------|--|
| <i>kStatus_MCG_Source-Used</i> | Because the external reference clock is used as a clock source, the configuration should not be changed. Otherwise, a glitch occurs. |
| <i>kStatus_Success</i>         | External reference clock set successfully.   |

### 6.7.30 **void** CLOCK\_EnablePll0 ( **mcg\_pll\_config\_t** const \* *config* )

This function sets up the PLL0 in FLL mode and reconfigures the PLL0. Ensure that the PLL reference clock is enabled before calling this function and that the PLL0 is not used as a clock source. The function CLOCK\_CalcPllDiv gets the correct PLL divider values.

Parameters

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

### 6.7.31 **static void** CLOCK\_DisablePll0 ( **void** ) [**inline**], [**static**]

This function disables the PLL0 in FLL mode. It should be used together with the [CLOCK\\_EnablePll0](#).

### 6.7.32 **uint32\_t** CLOCK\_CalcPllDiv ( **uint32\_t** *refFreq*, **uint32\_t** *desireFreq*, **uint8\_t** \* *prdiv*, **uint8\_t** \* *vddiv* )

This function calculates the correct reference clock divider (PRDIV) and VCO divider (VDIV) to generate a desired PLL output frequency. It returns the closest frequency match with the corresponding PRDIV/-

VDIV returned from parameters. If a desired frequency is not valid, this function returns 0.

## Function Documentation

### Parameters

|                    |  |
|--------------------|--|
| <i>refFreq</i>     | PLL reference clock frequency.                 |
| <i>desiredFreq</i> | Desired PLL output frequency.                  |
| <i>prdiv</i>       | PRDIV value to generate desired PLL frequency. |
| <i>vdiv</i>        | VDIV value to generate desired PLL frequency.  |

### Returns

Closest frequency match that the PLL was able generate.

### 6.7.33 void CLOCK\_SetOsc0MonitorMode ( mcg\_monitor\_mode\_t mode )

This function sets the OSC0 clock monitor mode. See [mcg\\_monitor\\_mode\\_t](#) for details.

#### Parameters

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

### 6.7.34 void CLOCK\_SetPll0MonitorMode ( mcg\_monitor\_mode\_t mode )

This function sets the PLL0 clock monitor mode. See [mcg\\_monitor\\_mode\\_t](#) for details.

#### Parameters

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

### 6.7.35 uint32\_t CLOCK\_GetStatusFlags ( void )

This function gets the MCG clock status flags. All status flags are returned as a logical OR of the enumeration [\\_mcg\\_status\\_flags\\_t](#). To check a specific flag, compare the return value with the flag.

Example:

```
// To check the clock lost lock status of OSC0 and PLL0.
uint32_t mcgFlags;

mcgFlags = CLOCK_GetStatusFlags();

if (mcgFlags & kMCG_Osc0LostFlag)
{
    // OSC0 clock lock lost. Do something.
}
```

```

if (mcgFlags & kMCG_Pll0LostFlag)
{
    // PLL0 clock lock lost. Do something.
}

```

#### Returns

Logical OR value of the [\\_mcg\\_status\\_flags\\_t](#).

### 6.7.36 void CLOCK\_ClearStatusFlags ( uint32\_t *mask* )

This function clears the MCG clock lock lost status. The parameter is a logical OR value of the flags to clear. See [\\_mcg\\_status\\_flags\\_t](#).

#### Example:

```

// To clear the clock lost lock status flags of OSC0 and PLL0.
CLOCK_ClearStatusFlags(kMCG_Osc0LostFlag | kMCG_Pll0LostFlag);

```

#### Parameters

|             |   |
|-------------|---|
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">_mcg_status_flags_t</a> . |
|-------------|---|

### 6.7.37 static void OSC\_SetExtRefClkConfig ( OSC\_Type \* *base*, oscr\_config\_t const \* *config* ) [inline], [static]

This function configures the OSC external reference clock (OSCERCLK). This is an example to enable the OSCERCLK in normal and stop modes and also set the output divider to 1:

```

oscer_config_t config =
{
    .enableMode = kOSC_ErClkEnable |
                  kOSC_ErClkEnableInStop,
    .erclkDiv   = 1U,
};

OSC_SetExtRefClkConfig(OSC, &config);

```

## Function Documentation

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | OSC peripheral address.                 |
| <i>config</i> | Pointer to the configuration structure. |

### 6.7.38 static void OSC\_SetCapLoad ( OSC\_Type \* *base*, uint8\_t *capLoad* ) [inline], [static]

This function sets the specified capacitors configuration for the oscillator. This should be done in the early system level initialization function call based on the system configuration.

### Parameters

|                |  |
|----------------|--|
| <i>base</i>    | OSC peripheral address.  |
| <i>capLoad</i> | OR'ed value for the capacitor load option, see <a href="#">_osc_cap_load</a> . |

Example:

```
// To enable only 2 pF and 8 pF capacitor load, please use like this.  
OSC_SetCapLoad(OSC, kOSC_Cap2P | kOSC_Cap8P);
```

### 6.7.39 void CLOCK\_InitOsc0 ( osc\_config\_t const \* *config* )

This function initializes the OSC0 according to the board configuration.

### Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to the OSC0 configuration structure. |
|---------------|--|

### 6.7.40 void CLOCK\_DeinitOsc0 ( void )

This function deinitializes the OSC0.

### 6.7.41 static void CLOCK\_SetXtal0Freq ( uint32\_t *freq* ) [inline], [static]



## Parameters

|             |   |
|-------------|---|
| <i>freq</i> | The XTAL0/EXTAL0 input clock frequency in Hz. |
|-------------|---|

**6.7.42 static void CLOCK\_SetXtal32Freq ( uint32\_t *freq* ) [inline], [static]**

## Parameters

|             |   |
|-------------|---|
| <i>freq</i> | The XTAL32/EXTAL32/RTC_CLKIN input clock frequency in Hz. |
|-------------|---|

**6.7.43 status\_t CLOCK\_TrimInternalRefClk ( uint32\_t *extFreq*, uint32\_t *desiredFreq*, uint32\_t \* *actualFreq*, mcg\_atm\_select\_t *atms* )**

This function trims the internal reference clock by using the external clock. If successful, it returns the `kStatus_Success` and the frequency after trimming is received in the parameter `actualFreq`. If an error occurs, the error code is returned.

## Parameters

|                    |  |
|--------------------|--|
| <i>extFreq</i>     | External clock frequency, which should be a bus clock. |
| <i>desiredFreq</i> | Frequency to trim to.                                  |
| <i>actualFreq</i>  | Actual frequency after trimming.                       |
| <i>atms</i>        | Trim fast or slow internal reference clock.            |

## Return values

|   |  |
|---|--|
| <i>kStatus_Success</i>                    | ATM success.   |
| <i>kStatus_MCG_AtmBus-ClockInvalid</i>    | The bus clock is not in allowed range for the ATM.             |
| <i>kStatus_MCG_Atm-DesiredFreqInvalid</i> | MCGIRCLK could not be trimmed to the desired frequency.        |
| <i>kStatus_MCG_AtmIrc-Used</i>            | Could not trim because MCGIRCLK is used as a bus clock source. |

## Function Documentation

|                                     |                                |
|-------------------------------------|--------------------------------|
| <i>kStatus_MCG_Atm-HardwareFail</i> | Hardware fails while trimming. |
|-------------------------------------|--------------------------------|

### 6.7.44 mcg\_mode\_t CLOCK\_GetMode ( void )

This function checks the MCG registers and determines the current MCG mode.

Returns

Current MCG mode or error code; See [mcg\\_mode\\_t](#).

### 6.7.45 status\_t CLOCK\_SetFeiMode ( mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*) (void) fllStableDelay )

This function sets the MCG to FEI mode. If setting to FEI mode fails from the current mode, this function returns an error.

Parameters

|                       |   |
|-----------------------|---|
| <i>dmx32</i>          | DMX32 in FEI mode.  |
| <i>drs</i>            | The DCO range selection.  |
| <i>fllStableDelay</i> | Delay function to ensure that the FLL is stable. Passing NULL does not cause a delay. |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

Note

If *dmx32* is set to *kMCG\_Dmx32Fine*, the slow IRC must not be trimmed to a frequency above 32768 Hz.

### 6.7.46 status\_t CLOCK\_SetFeeMode ( uint8\_t frdiv, mcg\_dmx32\_t dmx32, mcg\_drs\_t drs, void(\*) (void) fllStableDelay )

This function sets the MCG to FEE mode. If setting to FEE mode fails from the current mode, this function returns an error.

## Parameters

|                      |   |
|----------------------|---|
| <i>frdiv</i>         | FLL reference clock divider setting, FRDIV.                                     |
| <i>dmx32</i>         | DMX32 in FEE mode.  |
| <i>drs</i>           | The DCO range selection.  |
| <i>flStableDelay</i> | Delay function to make sure FLL is stable. Passing NULL does not cause a delay. |

## Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

#### 6.7.47 **status\_t** CLOCK\_SetFbiMode ( **mcg\_dmx32\_t** *dmx32*, **mcg\_drs\_t** *drs*, **void(\*)**(**void**) *flStableDelay* )

This function sets the MCG to FBI mode. If setting to FBI mode fails from the current mode, this function returns an error.

## Parameters

|                      |   |
|----------------------|---|
| <i>dmx32</i>         | DMX32 in FBI mode.  |
| <i>drs</i>           | The DCO range selection.  |
| <i>flStableDelay</i> | Delay function to make sure FLL is stable. If the FLL is not used in FBI mode, this parameter can be NULL. Passing NULL does not cause a delay. |

## Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

## Function Documentation

### Note

If `dmx32` is set to `kMCG_Dmx32Fine`, the slow IRC must not be trimmed to frequency above 32768 Hz.

**6.7.48** `status_t CLOCK_SetFbeMode ( uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(*)(void) flStableDelay )`

This function sets the MCG to FBE mode. If setting to FBE mode fails from the current mode, this function returns an error.

## Parameters

|                       |   |
|-----------------------|---|
| <i>frdiv</i>          | FLL reference clock divider setting, FRDIV.   |
| <i>dmx32</i>          | DMX32 in FBE mode.  |
| <i>drs</i>            | The DCO range selection.  |
| <i>fllStableDelay</i> | Delay function to make sure FLL is stable. If the FLL is not used in FBE mode, this parameter can be NULL. Passing NULL does not cause a delay. |

## Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

**6.7.49 status\_t CLOCK\_SetBlpiMode ( void )**

This function sets the MCG to BLPI mode. If setting to BLPI mode fails from the current mode, this function returns an error.

## Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

**6.7.50 status\_t CLOCK\_SetBlpeMode ( void )**

This function sets the MCG to BLPE mode. If setting to BLPE mode fails from the current mode, this function returns an error.

## Return values

|                                     |                                      |
|-------------------------------------|--------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode. |
|-------------------------------------|--------------------------------------|

## Function Documentation

|                        |   |
|------------------------|---|
| <i>kStatus_Success</i> | Switched to the target mode successfully. |
|------------------------|---|

### 6.7.51 **status\_t** CLOCK\_SetPbeMode ( **mcg\_pll\_clk\_select\_t** *pllcs*, **mcg\_pll\_config\_t** const \* *config* )

This function sets the MCG to PBE mode. If setting to PBE mode fails from the current mode, this function returns an error.

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>pllcs</i>  | The PLL selection, PLLCS.         |
| <i>config</i> | Pointer to the PLL configuration. |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

Note

1. The parameter *pllcs* selects the PLL. For platforms with only one PLL, the parameter *pllcs* is kept for interface compatibility.
2. The parameter *config* is the PLL configuration structure. On some platforms, it is possible to choose the external PLL directly, which renders the configuration structure not necessary. In this case, pass in NULL. For example: `CLOCK_SetPbeMode(kMCG_OscselOsc, kMCG_PllClkSelExtPll, NULL);`

### 6.7.52 **status\_t** CLOCK\_SetPeeMode ( **void** )

This function sets the MCG to PEE mode.

Return values

|                                     |                                      |
|-------------------------------------|--------------------------------------|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode. |
|-------------------------------------|--------------------------------------|

|                        |   |
|------------------------|---|
| <i>kStatus_Success</i> | Switched to the target mode successfully. |
|------------------------|---|

## Note

This function only changes the CLKS to use the PLL/FLL output. If the PRDIV/VDIV are different than in the PBE mode, set them up in PBE mode and wait. When the clock is stable, switch to PEE mode.

### 6.7.53 **status\_t** CLOCK\_ExternalModeToFbeModeQuick ( void )

This function switches the MCG from external modes (PEE/PBE/BLPE/FEE) to the FBE mode quickly. The external clock is used as the system clock source and PLL is disabled. However, the FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEE mode to FEI mode:

```
* CLOCK_ExternalModeToFbeModeQuick();
* CLOCK_SetFeiMode(...);
*
```

## Return values

|                                 |   |
|---------------------------------|---|
| <i>kStatus_Success</i>          | Switched successfully.  |
| <i>kStatus_MCG_Mode-Invalid</i> | If the current mode is not an external mode, do not call this function. |

### 6.7.54 **status\_t** CLOCK\_InternalModeToFbiModeQuick ( void )

This function switches the MCG from internal modes (PEI/PBI/BLPI/FEI) to the FBI mode quickly. The MCGIRCLK is used as the system clock source and PLL is disabled. However, FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEI mode to FEE mode:

```
* CLOCK_InternalModeToFbiModeQuick();
* CLOCK_SetFeeMode(...);
*
```

## Function Documentation

Return values

|                                 |   |
|---------------------------------|---|
| <i>kStatus_Success</i>          | Switched successfully.  |
| <i>kStatus_MCG_Mode-Invalid</i> | If the current mode is not an internal mode, do not call this function. |

### 6.7.55 **status\_t** CLOCK\_BootToFeiMode ( mcg\_dmx32\_t *dmx32*, mcg\_drs\_t *drs*, void(\*)*(void) flStableDelay* )

This function sets the MCG to FEI mode from the reset mode. It can also be used to set up MCG during system boot up.

Parameters

|                      |  |
|----------------------|--|
| <i>dmx32</i>         | DMX32 in FEI mode.                               |
| <i>drs</i>           | The DCO range selection.                         |
| <i>flStableDelay</i> | Delay function to ensure that the FLL is stable. |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

Note

If *dmx32* is set to *kMCG\_Dmx32Fine*, the slow IRC must not be trimmed to frequency above 32768 Hz.

### 6.7.56 **status\_t** CLOCK\_BootToFeeMode ( mcg\_oscsel\_t *oscsel*, uint8\_t *frdiv*, mcg\_dmx32\_t *dmx32*, mcg\_drs\_t *drs*, void(\*)*(void) flStableDelay* )

This function sets MCG to FEE mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters



|                       |  |
|-----------------------|--|
| <i>oscsel</i>         | OSC clock select, OSCSEL.                        |
| <i>frdiv</i>          | FLL reference clock divider setting, FRDIV.      |
| <i>dmx32</i>          | DMX32 in FEE mode.                               |
| <i>drs</i>            | The DCO range selection.                         |
| <i>fllStableDelay</i> | Delay function to ensure that the FLL is stable. |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

#### 6.7.57 **status\_t** CLOCK\_BootToBlpiMode ( uint8\_t *fcrdiv*, mcg\_irc\_mode\_t *ircs*, uint8\_t *ircEnableMode* )

This function sets the MCG to BLPI mode from the reset mode. It can also be used to set up the MCG during sytem boot up.

Parameters

|                      |  |
|----------------------|--|
| <i>fcrdiv</i>        | Fast IRC divider, FCRDIV.  |
| <i>ircs</i>          | The internal reference clock to select, IRCS.                                      |
| <i>ircEnableMode</i> | The MCGIRCLK enable mode, OR'ed value of <a href="#">_mcg_ircclk_enable_mode</a> . |

Return values

|                                |   |
|--------------------------------|---|
| <i>kStatus_MCG_Source-Used</i> | Could not change MCGIRCLK setting.        |
| <i>kStatus_Success</i>         | Switched to the target mode successfully. |

#### 6.7.58 **status\_t** CLOCK\_BootToBlpeMode ( mcg\_oscsel\_t *oscsel* )

This function sets the MCG to BLPE mode from the reset mode. It can also be used to set up the MCG during sytem boot up.

## Function Documentation

### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>oscsel</i> | OSC clock select, MCG_C7[OSCSEL]. |
|---------------|-----------------------------------|

### Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

### 6.7.59 **status\_t** CLOCK\_BootToPeeMode ( **mcg\_oscsel\_t** *oscsel*, **mcg\_pll\_clk\_select\_t** *pllcs*, **mcg\_pll\_config\_t** *config* )

This function sets the MCG to PEE mode from reset mode. It can also be used to set up the MCG during system boot up.

### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>oscsel</i> | OSC clock select, MCG_C7[OSCSEL]. |
| <i>pllcs</i>  | The PLL selection, PLLCS.         |
| <i>config</i> | Pointer to the PLL configuration. |

### Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_MCG_Mode-Unreachable</i> | Could not switch to the target mode.      |
| <i>kStatus_Success</i>              | Switched to the target mode successfully. |

### 6.7.60 **status\_t** CLOCK\_SetMcgConfig ( **mcg\_config\_t** *config* )

This function sets MCG to a target mode defined by the configuration structure. If switching to the target mode fails, this function chooses the correct path.

### Parameters

|               |   |
|---------------|---|
| <i>config</i> | Pointer to the target MCG mode configuration structure. |
|---------------|---|

### Returns

Return `kStatus_Success` if switched successfully; Otherwise, it returns an error code [\\_mcg\\_status](#).

## Note

If the external clock is used in the target mode, ensure that it is enabled. For example, if the OSC0 is used, set up OSC0 correctly before calling this function.

## 6.8 Variable Documentation

### 6.8.1 uint32\_t g\_xtal0Freq

The XTAL0/EXTAL0 (OSC0) clock frequency in Hz. When the clock is set up, use the function `CLOCK_SetXtal0Freq` to set the value in the clock driver. For example, if XTAL0 is 8 MHz:

```
* CLOCK_InitOsc0(...); // Set up the OSC0
* CLOCK_SetXtal0Freq(8000000); // Set the XTAL0 value to the clock driver.
*
```

This is important for the multicore platforms where only one core needs to set up the OSC0 using the `CLOCK_InitOsc0`. All other cores need to call the `CLOCK_SetXtal0Freq` to get a valid clock frequency.

### 6.8.2 uint32\_t g\_xtal32Freq

The XTAL32/EXTAL32/RTC\_CLKIN clock frequency in Hz. When the clock is set up, use the function `CLOCK_SetXtal32Freq` to set the value in the clock driver.

This is important for the multicore platforms where only one core needs to set up the clock. All other cores need to call the `CLOCK_SetXtal32Freq` to get a valid clock frequency.

### 6.9 Multipurpose Clock Generator (MCG)

The KSDK provides a peripheral driver for the MCG module of Kinetis devices.

#### 6.9.1 Function description

MCG driver provides these functions:

- Functions to get the MCG clock frequency.
- Functions to configure the MCG clock, such as PLLCLK and MCGIRCLK.
- Functions for the MCG clock lock lost monitor.
- Functions for the OSC configuration.
- Functions for the MCG auto-trim machine.
- Functions for the MCG mode.

##### 6.9.1.1 MCG frequency functions

MCG module provides clocks, such as MCGOUTCLK, MCGIRCLK, MCGFFCLK, MCGFLLCLK and MCGPLLCLK. The MCG driver provides functions to get the frequency of these clocks, such as [CLOCK\\_GetOutClkFreq\(\)](#), [CLOCK\\_GetInternalRefClkFreq\(\)](#), [CLOCK\\_GetFixedFreqClkFreq\(\)](#), [CLOCK\\_GetFllFreq\(\)](#), [CLOCK\\_GetPlloFreq\(\)](#), [CLOCK\\_GetPll1Freq\(\)](#), and [CLOCK\\_GetExtPllFreq\(\)](#). These functions get the clock frequency based on the current MCG registers.

##### 6.9.1.2 MCG clock configuration

The MCG driver provides functions to configure the internal reference clock (MCGIRCLK), the external reference clock, and MCGPLLCLK.

The function [CLOCK\\_SetInternalRefClkConfig\(\)](#) configures the MCGIRCLK, including the source and the divider. Do not change MCGIRCLK when the MCG mode is BLPI/FBI/PBI because the MCGIRCLK is used as a system clock in these modes and changing settings makes the system clock unstable.

The function [CLOCK\\_SetExternalRefClkConfig\(\)](#) configures the external reference clock source (MCG\_C7[OSCSEL]). Do not call this function when the MCG mode is BLPE/FBE/PBE/FEE/PEE because the external reference clock is used as a clock source in these modes. Changing the external reference clock source requires at least a 50 micro seconds wait. The function [CLOCK\\_SetExternalRefClkConfig\(\)](#) implements a for loop delay internally. The for loop delay assumes that the system clock is 96 MHz, which ensures at least 50 micro seconds delay. However, when the system clock is slow, the delay time may significantly increase. This for loop count can be optimized for better performance for specific cases.

The MCGPLLCLK is disabled in FBE/FEE/FBI/FEI modes by default. Applications can enable the MCGPLLCLK in these modes using the functions [CLOCK\\_EnablePllo\(\)](#) and [CLOCK\\_EnablePll1\(\)](#). To enable the MCGPLLCLK, the PLL reference clock divider (PRDIV) and the PLL VCO divider (VDIV) must be set to a proper value. The function [CLOCK\\_CalcPllDiv\(\)](#) helps to get the PRDIV/VDIV.

### 6.9.1.3 MCG clock lock monitor functions

The MCG module monitors the OSC and the PLL clock lock status. The MCG driver provides the functions to set the clock monitor mode, check the clock lost status, and clear the clock lost status.

### 6.9.1.4 OSC configuration

The MCG is needed together with the OSC module to enable the OSC clock. The function [CLOCK\\_InitOsc0\(\)](#) [CLOCK\\_InitOsc1](#) uses the MCG and OSC to initialize the OSC. The OSC should be configured based on the board design.

### 6.9.1.5 MCG auto-trim machine

The MCG provides an auto-trim machine to trim the MCG internal reference clock based on the external reference clock (BUS clock). During clock trimming, the MCG must not work in FEI/FBI/BLPI/PBI/PEI modes. The function [CLOCK\\_TrimInternalRefClk\(\)](#) is used for the auto clock trimming.

### 6.9.1.6 MCG mode functions

The function [CLOCK\\_GetMcgMode](#) returns the current MCG mode. The MCG can only switch between the neighbouring modes. If the target mode is not current mode's neighbouring mode, the application must choose the proper switch path. For example, to switch to PEE mode from FEI mode, use FEI -> FBE -> PBE -> PEE.

For the MCG modes, the MCG driver provides three kinds of functions:

The first type of functions involve functions [CLOCK\\_SetXxxMode](#), such as [CLOCK\\_SetFeiMode\(\)](#). These functions only set the MCG mode from neighbouring modes. If switching to the target mode directly from current mode is not possible, the functions return an error.

The second type of functions are the functions [CLOCK\\_BootToXxxMode](#), such as [CLOCK\\_BootToFeiMode\(\)](#). These functions set the MCG to specific modes from reset mode. Because the source mode and target mode are specific, these functions choose the best switch path. The functions are also useful to set up the system clock during boot up.

The third type of functions is the [CLOCK\\_SetMcgConfig\(\)](#). This function chooses the right path to switch to the target mode. It is easy to use, but introduces a large code size.

Whenever the FLL settings change, there should be a 1 millisecond delay to ensure that the FLL is stable. The function [CLOCK\\_SetMcgConfig\(\)](#) implements a for loop delay internally to ensure that the FLL is stable. The for loop delay assumes that the system clock is 96 MHz, which ensures at least 1 millisecond delay. However, when the system clock is slow, the delay time may increase significantly. The for loop count can be optimized for better performance according to a specific case.

## Multipurpose Clock Generator (MCG)

### 6.9.2 Typical use case

The function `CLOCK_SetMcgConfig` is used to switch between any modes. However, this heavy-light function introduces a large code size. This section shows how to use the mode function to implement a quick and light-weight switch between typical specific modes. Note that the step to enable the external clock is not included in the following steps. T Enable the corresponding clock before using it as a clock source.

#### 6.9.2.1 Switch between BLPI and FEI

| Use case    | Steps                      | Functions   |
|-------------|----------------------------|---|
| BLPI -> FEI | BLPI -> FBI                | <code>CLOCK_InternalModeToFbiModeQuick(...)</code>                      |
|             | FBI -> FEI                 | <code>CLOCK_SetFeiMode(...)</code>                                      |
|             | Configure MCGIRCLK if need | <code>CLOCK_SetInternalRefClkConfig(...)</code>                         |
| FEI -> BLPI | Configure MCGIRCLK if need | <code>CLOCK_SetInternalRefClkConfig(...)</code>                         |
|             | FEI -> FBI                 | <code>CLOCK_SetFbiMode(...)</code> with <code>flStableDelay=NULL</code> |
|             | FBI -> BLPI                | <code>CLOCK_SetLowPowerEnable(true)</code>                              |

#### 6.9.2.2 Switch between BLPI and FEE

| Use case    | Steps                                | Functions   |
|-------------|--------------------------------------|---|
| BLPI -> FEE | BLPI -> FBI                          | <code>CLOCK_InternalModeToFbiModeQuick(...)</code>                      |
|             | Change external clock source if need | <code>CLOCK_SetExternalRefClkConfig(...)</code>                         |
|             | FBI -> FEE                           | <code>CLOCK_SetFeeMode(...)</code>                                      |
| FEE -> BLPI | Configure MCGIRCLK if need           | <code>CLOCK_SetInternalRefClkConfig(...)</code>                         |
|             | FEE -> FBI                           | <code>CLOCK_SetFbiMode(...)</code> with <code>flStableDelay=NULL</code> |
|             | FBI -> BLPI                          | <code>CLOCK_SetLowPowerEnable(true)</code>                              |

### 6.9.2.3 Switch between BLPI and PEE

| Use case    | Steps                                | Functions                                     |
|-------------|--------------------------------------|---|
| BLPI -> PEE | BLPI -> FBI                          | CLOCK_InternalModeToFbi-ModeQuick(...)        |
|             | Change external clock source if need | CLOCK_SetExternalRefClk-Config(...)           |
|             | FBI -> FBE                           | CLOCK_SetFbeMode(...) // fl-StableDelay=NULL  |
|             | FBE -> PBE                           | CLOCK_SetPbeMode(...)                         |
|             | PBE -> PEE                           | CLOCK_SetPeeMode(...)                         |
| PEE -> BLPI | PEE -> FBE                           | CLOCK_ExternalModeToFbe-ModeQuick(...)        |
|             | Configure MCGIRCLK if need           | CLOCK_SetInternalRefClk-Config(...)           |
|             | FBE -> FBI                           | CLOCK_SetFbiMode(...) with flStableDelay=NULL |
|             | FBI -> BLPI                          | CLOCK_SetLowPower-Enable(true)                |

### 6.9.2.4 Switch between BLPE and PEE

This table applies when using the same external clock source (MCG\_C7[OSCSEL]) in BLPE mode and PEE mode.

| Use case    | Steps       | Functions                              |
|-------------|-------------|--|
| BLPE -> PEE | BLPE -> PBE | CLOCK_SetPbeMode(...)                  |
|             | PBE -> PEE  | CLOCK_SetPeeMode(...)                  |
| PEE -> BLPE | PEE -> FBE  | CLOCK_ExternalModeToFbe-ModeQuick(...) |
|             | FBE -> BLPE | CLOCK_SetLowPower-Enable(true)         |

If using different external clock sources (MCG\_C7[OSCSEL]) in BLPE mode and PEE mode, call the [CLOCK\\_SetExternalRefClkConfig\(\)](#) in FBI or FEI mode to change the external reference clock.

| Use case | Steps       | Functions                              |
|----------|-------------|--|
|          | BLPE -> FBE | CLOCK_ExternalModeToFbe-ModeQuick(...) |

## Multipurpose Clock Generator (MCG)

|             |               |   |
|-------------|---------------|---|
|             | FBE -> FBI    | CLOCK_SetFbiMode(...) with flStableDelay=NULL |
|             | Change source | CLOCK_SetExternalRefClkConfig(...)            |
|             | FBI -> FBE    | CLOCK_SetFbeMode(...) with flStableDelay=NULL |
|             | FBE -> PBE    | CLOCK_SetPbeMode(...)                         |
|             | PBE -> PEE    | CLOCK_SetPeeMode(...)                         |
| PEE -> BLPE | PEE -> FBE    | CLOCK_ExternalModeToFbeModeQuick(...)         |
|             | FBE -> FBI    | CLOCK_SetFbiMode(...) with flStableDelay=NULL |
|             | Change source | CLOCK_SetExternalRefClkConfig(...)            |
|             | PBI -> FBE    | CLOCK_SetFbeMode(...) with flStableDelay=NULL |
|             | FBE -> BLPE   | CLOCK_SetLowPowerEnable(true)                 |

### 6.9.2.5 Switch between BLPE and FEE

This table applies when using the same external clock source (MCG\_C7[OSCSEL]) in BLPE mode and FEE mode.

| Use case    | Steps       | Functions                             |
|-------------|-------------|---------------------------------------|
| BLPE -> FEE | BLPE -> FBE | CLOCK_ExternalModeToFbeModeQuick(...) |
|             | FBE -> FEE  | CLOCK_SetFeeMode(...)                 |
| FEE -> BLPE | PEE -> FBE  | CLOCK_SetPbeMode(...)                 |
|             | FBE -> BLPE | CLOCK_SetLowPowerEnable(true)         |

If using different external clock sources (MCG\_C7[OSCSEL]) in BLPE mode and FEE mode, call the [CLOCK\\_SetExternalRefClkConfig\(\)](#) in FBI or FEI mode to change the external reference clock.

| Use case    | Steps       | Functions                             |
|-------------|-------------|---------------------------------------|
| BLPE -> FEE | BLPE -> FBE | CLOCK_ExternalModeToFbeModeQuick(...) |



## Multipurpose Clock Generator (MCG)

|             |               |   |
|-------------|---------------|---|
|             | FBE -> FBI    | CLOCK_SetFbiMode(...) with flStableDelay=NULL |
|             | Change source | CLOCK_SetExternalRefClk-Config(...)           |
|             | FBI -> FEE    | CLOCK_SetFeeMode(...)                         |
| FEE -> BLPE | FEE -> FBI    | CLOCK_SetFbiMode(...) with flStableDelay=NULL |
|             | Change source | CLOCK_SetExternalRefClk-Config(...)           |
|             | PBI -> FBE    | CLOCK_SetFbeMode(...) with flStableDelay=NULL |
|             | FBE -> BLPE   | CLOCK_SetLowPower-Enable(true)                |

### 6.9.2.6 Switch between BLPI and PEI

| Use case    | Steps                      | Functions                              |
|-------------|----------------------------|--|
| BLPI -> PEI | BLPI -> PBI                | CLOCK_SetPbiMode(...)                  |
|             | PBI -> PEI                 | CLOCK_SetPeiMode(...)                  |
|             | Configure MCGIRCLK if need | CLOCK_SetInternalRefClk-Config(...)    |
| PEI -> BLPI | Configure MCGIRCLK if need | CLOCK_SetInternalRefClk-Config         |
|             | PEI -> FBI                 | CLOCK_InternalModeToFbi-ModeQuick(...) |
|             | FBI -> BLPI                | CLOCK_SetLowPower-Enable(true)         |



## Chapter 7

# CMP: Analog Comparator Driver

### 7.1 Overview

The KSDK provides a peripheral driver for the Analog Comparator (CMP) module of Kinetis devices.

The CMP driver is a basic comparator with advanced features. The APIs for the basic comparator enable the CMP as a general comparator, which compares two voltages of the two input channels and creates the output of the comparator result. The APIs for advanced features can be used as the plug-in function based on the basic comparator. They can process the comparator's output with hardware support.

### 7.2 Typical use case

#### 7.2.1 Polling Configuration

```
int main(void)
{
    cmp_config_t mCmpConfigStruct;
    cmp_dac_config_t mCmpDacConfigStruct;

    // ...

    // Configures the comparator.
    CMP_Init(DEMO_CMP_INSTANCE);
    CMP_GetDefaultConfig(&mCmpConfigStruct);
    CMP_Configure(DEMO_CMP_INSTANCE, &mCmpConfigStruct);

    // Configures the DAC channel.
    mCmpDacConfigStruct.referenceVoltageSource =
        kCMP_VrefSourceVin2; // VCC.
    mCmpDacConfigStruct.DACValue = 32U; // Half voltage of logic high-level.
    CMP_SetDACConfig(DEMO_CMP_INSTANCE, &mCmpDacConfigStruct);
    CMP_SetInputChannels(DEMO_CMP_INSTANCE, DEMO_CMP_USER_CHANNEL, DEMO_CMP_DAC_CHANNEL);
}

while (1)
{
    if (0U != (kCMP_OutputAssertEventFlag &
        CMP_GetStatusFlags(DEMO_CMP_INSTANCE)))
    {
        // Do something.
    }
    else
    {
        // Do something.
    }
}
```

#### 7.2.2 Interrupt Configuration

```
volatile uint32_t g_CmpFlags = 0U;
```

## Typical use case

```
// ...

void DEMO_CMP_IRQ_HANDLER_FUNC(void)
{
    g_CmpFlags = CMP_GetStatusFlags(DEMO_CMP_INSTANCE);
    CMP_ClearStatusFlags(DEMO_CMP_INSTANCE, kCMP_OutputRisingEventFlag |
        kCMP_OutputFallingEventFlag);
    if (0U != (g_CmpFlags & kCMP_OutputRisingEventFlag))
    {
        // Do something.
    }
    else if (0U != (g_CmpFlags & kCMP_OutputFallingEventFlag))
    {
        // Do something.
    }
}

int main(void)
{
    cmp_config_t mCmpConfigStruct;
    cmp_dac_config_t mCmpDacConfigStruct;

    // ...
    EnableIRQ(DEMO_CMP_IRQ_ID);
    // ...

    // Configures the comparator.
    CMP_Init(DEMO_CMP_INSTANCE);
    CMP_GetDefaultConfig(&mCmpConfigStruct);
    CMP_Configure(DEMO_CMP_INSTANCE, &mCmpConfigStruct);

    // Configures the DAC channel.
    mCmpDacConfigStruct.referenceVoltageSource =
        kCMP_VrefSourceVin2; // VCC.
    mCmpDacConfigStruct.DACValue = 32U; // Half voltage of logic high-level.
    CMP_SetDACConfig(DEMO_CMP_INSTANCE, &mCmpDacConfigStruct);
    CMP_SetInputChannels(DEMO_CMP_INSTANCE, DEMO_CMP_USER_CHANNEL, DEMO_CMP_DAC_CHANNEL
        );

    // Enables the output rising and falling interrupts.
    CMP_EnableInterrupts(DEMO_CMP_INSTANCE,
        kCMP_OutputRisingInterruptEnable |
        kCMP_OutputFallingInterruptEnable);

    while (1)
    {
    }
}
```

## Data Structures

- struct `cmp_config_t`  
Configuration for the comparator. [More...](#)
- struct `cmp_filter_config_t`  
Configuration for the filter. [More...](#)
- struct `cmp_dac_config_t`  
Configuration for the internal DAC. [More...](#)

## Enumerations

- enum `_cmp_interrupt_enable` {  
    `kCMP_OutputRisingInterruptEnable` = `CMP_SCR_IER_MASK`,  
    `kCMP_OutputFallingInterruptEnable` = `CMP_SCR_IEF_MASK` }

- *Interrupt enable/disable mask.*
- enum `_cmp_status_flags` {  
`kCMP_OutputRisingEventFlag` = `CMP_SCR_CFR_MASK`,  
`kCMP_OutputFallingEventFlag` = `CMP_SCR_CFF_MASK`,  
`kCMP_OutputAssertEventFlag` = `CMP_SCR_COUT_MASK` }
- *Status flags' mask.*
- enum `cmp_hysteresis_mode_t` {  
`kCMP_HysteresisLevel0` = `0U`,  
`kCMP_HysteresisLevel1` = `1U`,  
`kCMP_HysteresisLevel2` = `2U`,  
`kCMP_HysteresisLevel3` = `3U` }
- *CMP Hysteresis mode.*
- enum `cmp_reference_voltage_source_t` {  
`kCMP_VrefSourceVin1` = `0U`,  
`kCMP_VrefSourceVin2` = `1U` }
- *CMP Voltage Reference source.*

## Driver version

- #define `FSL_CMP_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 0))  
*CMP driver version 2.0.0.*

## Initialization

- void `CMP_Init` (`CMP_Type` \*base, const `cmp_config_t` \*config)  
*Initializes the CMP.*
- void `CMP_Deinit` (`CMP_Type` \*base)  
*De-initializes the CMP module.*
- static void `CMP_Enable` (`CMP_Type` \*base, bool enable)  
*Enables/disables the CMP module.*
- void `CMP_GetDefaultConfig` (`cmp_config_t` \*config)  
*Initializes the CMP user configuration structure.*
- void `CMP_SetInputChannels` (`CMP_Type` \*base, uint8\_t positiveChannel, uint8\_t negativeChannel)  
*Sets the input channels for the comparator.*

## Advanced Features

- void `CMP_SetFilterConfig` (`CMP_Type` \*base, const `cmp_filter_config_t` \*config)  
*Configures the filter.*
- void `CMP_SetDACConfig` (`CMP_Type` \*base, const `cmp_dac_config_t` \*config)  
*Configures the internal DAC.*
- void `CMP_EnableInterrupts` (`CMP_Type` \*base, uint32\_t mask)  
*Enables the interrupts.*
- void `CMP_DisableInterrupts` (`CMP_Type` \*base, uint32\_t mask)  
*Disables the interrupts.*

## Results

- uint32\_t `CMP_GetStatusFlags` (`CMP_Type` \*base)

## Data Structure Documentation

- Gets the status flags.*
- void [CMP\\_ClearStatusFlags](#) (CMP\_Type \*base, uint32\_t mask)  
*Clears the status flags.*

## 7.3 Data Structure Documentation

### 7.3.1 struct cmp\_config\_t

#### Data Fields

- bool [enableCmp](#)  
*Enable the CMP module.*
- [cmp\\_hysteresis\\_mode\\_t](#) [hysteresisMode](#)  
*CMP Hysteresis mode.*
- bool [enableHighSpeed](#)  
*Enable High-speed 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.*

#### 7.3.1.0.0.9 Field Documentation

##### 7.3.1.0.0.9.1 bool cmp\_config\_t::enableCmp

##### 7.3.1.0.0.9.2 cmp\_hysteresis\_mode\_t cmp\_config\_t::hysteresisMode

##### 7.3.1.0.0.9.3 bool cmp\_config\_t::enableHighSpeed

##### 7.3.1.0.0.9.4 bool cmp\_config\_t::enableInvertOutput

##### 7.3.1.0.0.9.5 bool cmp\_config\_t::useUnfilteredOutput

##### 7.3.1.0.0.9.6 bool cmp\_config\_t::enablePinOut

### 7.3.2 struct cmp\_filter\_config\_t

#### Data Fields

- uint8\_t [filterCount](#)  
*Filter Sample Count.*
- uint8\_t [filterPeriod](#)  
*Filter Sample Period.*

### 7.3.2.0.0.10 Field Documentation

#### 7.3.2.0.0.10.1 uint8\_t cmp\_filter\_config\_t::filterCount

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

#### 7.3.2.0.0.10.2 uint8\_t cmp\_filter\_config\_t::filterPeriod

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

## 7.3.3 struct cmp\_dac\_config\_t

### Data Fields

- [cmp\\_reference\\_voltage\\_source\\_t referenceVoltageSource](#)  
*Supply voltage reference source.*
- uint8\_t [DACValue](#)  
*Value for DAC Output Voltage.*

### 7.3.3.0.0.11 Field Documentation

#### 7.3.3.0.0.11.1 cmp\_reference\_voltage\_source\_t cmp\_dac\_config\_t::referenceVoltageSource

#### 7.3.3.0.0.11.2 uint8\_t cmp\_dac\_config\_t::DACValue

Available range is 0-63.

## 7.4 Macro Definition Documentation

### 7.4.1 #define FSL\_CMP\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

## 7.5 Enumeration Type Documentation

### 7.5.1 enum \_cmp\_interrupt\_enable

Enumerator

- kCMP\_OutputRisingInterruptEnable*** Comparator interrupt enable rising.  
***kCMP\_OutputFallingInterruptEnable*** Comparator interrupt enable falling.

### 7.5.2 enum \_cmp\_status\_flags

Enumerator

- kCMP\_OutputRisingEventFlag*** Rising-edge on compare output has occurred.  
***kCMP\_OutputFallingEventFlag*** Falling-edge on compare output has occurred.

## Function Documentation

***kCMP\_OutputAssertEventFlag*** Return the current value of the analog comparator output.

### 7.5.3 enum cmp\_hysteresis\_mode\_t

Enumerator

***kCMP\_HysteresisLevel0*** Hysteresis level 0.  
***kCMP\_HysteresisLevel1*** Hysteresis level 1.  
***kCMP\_HysteresisLevel2*** Hysteresis level 2.  
***kCMP\_HysteresisLevel3*** Hysteresis level 3.

### 7.5.4 enum cmp\_reference\_voltage\_source\_t

Enumerator

***kCMP\_VrefSourceVin1*** Vin1 is selected as resistor ladder network supply reference Vin.  
***kCMP\_VrefSourceVin2*** Vin2 is selected as resistor ladder network supply reference Vin.

## 7.6 Function Documentation

### 7.6.1 void CMP\_Init ( CMP\_Type \* *base*, const cmp\_config\_t \* *config* )

This function initializes the CMP module. The operations included are:

- Enabling the clock for CMP module.
- Configuring the comparator.
- Enabling the CMP module. Note: For some devices, multiple CMP instance share the same clock gate. In this case, to enable the clock for any instance enables all the CMPs. Check the chip reference manual for the clock assignment of the CMP.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | CMP peripheral base address.        |
| <i>config</i> | Pointer to configuration structure. |

### 7.6.2 void CMP\_Deinit ( CMP\_Type \* *base* )

This function de-initializes the CMP module. The operations included are:

- Disabling the CMP module.
- Disabling the clock for CMP module.



This function disables the clock for the CMP. Note: For some devices, multiple CMP instance shares the same clock gate. In this case, before disabling the clock for the CMP, ensure that all the CMP instances are not used.

## Function Documentation

### Parameters

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

### 7.6.3 static void CMP\_Enable ( CMP\_Type \* *base*, bool *enable* ) [inline], [static]

### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | CMP peripheral base address. |
| <i>enable</i> | Enable the module or not.    |

### 7.6.4 void CMP\_GetDefaultConfig ( cmp\_config\_t \* *config* )

This function initializes the user configuration structure to these default values:

```
* config->enableCmp          = true;
* config->hysteresisMode     = kCMP_HysteresisLevel0;
* config->enableHighSpeed    = false;
* config->enableInvertOutput  = false;
* config->useUnfilteredOutput = false;
* config->enablePinOut       = false;
* config->enableTriggerMode  = false;
*
```

### Parameters

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

### 7.6.5 void CMP\_SetInputChannels ( CMP\_Type \* *base*, uint8\_t *positiveChannel*, uint8\_t *negativeChannel* )

This function sets the input channels for the comparator. Note that two input channels cannot be set as same in the application. When the user selects the same input from the analog mux to the positive and negative port, the comparator is disabled automatically.

### Parameters

|                         |   |
|-------------------------|---|
| <i>base</i>             | CMP peripheral base address.                                |
| <i>positive-Channel</i> | Positive side input channel number. Available range is 0-7. |
| <i>negative-Channel</i> | Negative side input channel number. Available range is 0-7. |

### 7.6.6 void CMP\_SetFilterConfig ( CMP\_Type \* *base*, const cmp\_filter\_config\_t \* *config* )

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | CMP peripheral base address.        |
| <i>config</i> | Pointer to configuration structure. |

### 7.6.7 void CMP\_SetDACConfig ( CMP\_Type \* *base*, const cmp\_dac\_config\_t \* *config* )

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | CMP peripheral base address.   |
| <i>config</i> | Pointer to configuration structure. "NULL" is for disabling the feature. |

### 7.6.8 void CMP\_EnableInterrupts ( CMP\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |   |
|-------------|---|
| <i>base</i> | CMP peripheral base address.                            |
| <i>mask</i> | Mask value for interrupts. See "_cmp_interrupt_enable". |

### 7.6.9 void CMP\_DisableInterrupts ( CMP\_Type \* *base*, uint32\_t *mask* )

## Function Documentation

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | CMP peripheral base address.                            |
| <i>mask</i> | Mask value for interrupts. See "_cmp_interrupt_enable". |

### 7.6.10 uint32\_t CMP\_GetStatusFlags ( CMP\_Type \* *base* )

### Parameters

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

### Returns

Mask value for the asserted flags. See "\_cmp\_status\_flags".

### 7.6.11 void CMP\_ClearStatusFlags ( CMP\_Type \* *base*, uint32\_t *mask* )

### Parameters

|             |  |
|-------------|--|
| <i>base</i> | CMP peripheral base address.                       |
| <i>mask</i> | Mask value for the flags. See "_cmp_status_flags". |

## Chapter 8

# COP: Watchdog Driver

### 8.1 Overview

The KSDK provides a peripheral driver for the Computer Operating Properly module (COP) of Kinetis devices.

### 8.2 Typical use case

```
cop_config_t config;
COP_GetDefaultConfig(&config);
config.timeoutCycles = kCOP_2Power8CyclesOr2Power16Cycles;
COP_Init(sim_base, &config);
```

### Data Structures

- struct `cop_config_t`  
*Describes COP configuration structure. [More...](#)*

### Enumerations

- enum `cop_clock_source_t` {  
    `kCOP_LpoClock` = 0U,  
    `kCOP_BusClock` = 3U }  
    *COP clock source selection.*
- enum `cop_timeout_cycles_t` {  
    `kCOP_2Power5CyclesOr2Power13Cycles` = 1U,  
    `kCOP_2Power8CyclesOr2Power16Cycles` = 2U,  
    `kCOP_2Power10CyclesOr2Power18Cycles` = 3U }  
    *Define the COP timeout cycles.*

### Driver version

- #define `FSL_COP_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
    *COP driver version 2.0.0.*

### COP refresh sequence.

- #define `COP_FIRST_BYTE_OF_REFRESH` (0x55U)  
    *First byte of refresh sequence.*
- #define `COP_SECOND_BYTE_OF_REFRESH` (0xAAU)  
    *Second byte of refresh sequence.*

## Enumeration Type Documentation

### COP Functional Operation

- void [COP\\_GetDefaultConfig](#) ([cop\\_config\\_t](#) \*config)  
*Initializes the COP configuration structure.*
- void [COP\\_Init](#) (SIM\_Type \*base, const [cop\\_config\\_t](#) \*config)  
*Initializes the COP module.*
- static void [COP\\_Disable](#) (SIM\_Type \*base)  
*De-initializes the COP module.*
- void [COP\\_Refresh](#) (SIM\_Type \*base)  
*Refreshes the COP timer.*

### 8.3 Data Structure Documentation

#### 8.3.1 struct cop\_config\_t

##### Data Fields

- bool [enableWindowMode](#)  
*COP run mode: window mode or normal mode.*
- [cop\\_clock\\_source\\_t](#) [clockSource](#)  
*Set COP clock source.*
- [cop\\_timeout\\_cycles\\_t](#) [timeoutCycles](#)  
*Set COP timeout value.*

### 8.4 Macro Definition Documentation

#### 8.4.1 #define FSL\_COP\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

### 8.5 Enumeration Type Documentation

#### 8.5.1 enum cop\_clock\_source\_t

Enumerator

*kCOP\_LpoClock* COP clock sourced from LPO.  
*kCOP\_BusClock* COP clock sourced from Bus clock.

#### 8.5.2 enum cop\_timeout\_cycles\_t

Enumerator

*kCOP\_2Power5CyclesOr2Power13Cycles* 2<sup>5</sup> or 2<sup>13</sup> clock cycles  
*kCOP\_2Power8CyclesOr2Power16Cycles* 2<sup>8</sup> or 2<sup>16</sup> clock cycles  
*kCOP\_2Power10CyclesOr2Power18Cycles* 2<sup>10</sup> or 2<sup>18</sup> clock cycles

## 8.6 Function Documentation

### 8.6.1 void COP\_GetDefaultConfig ( cop\_config\_t \* *config* )

This function initializes the COP configuration structure to default values. The default values are:

```
*  copConfig->enableWindowMode = false;
*  copConfig->timeoutMode = kCOP_LongTimeoutMode;
*  copConfig->enableStop = false;
*  copConfig->enableDebug = false;
*  copConfig->clockSource = kCOP_LpoClock;
*  copConfig->timeoutCycles = kCOP_2Power10CyclesOr2Power18Cycles;
*
```

Parameters

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

See Also

[cop\\_config\\_t](#)

### 8.6.2 void COP\_Init ( SIM\_Type \* *base*, const cop\_config\_t \* *config* )

This function configures the COP. After it is called, the COP starts running according to the configuration. Because all COP control registers are write-once only, the COP\_Init function and the COP\_Disable function can be called only once. A second call has no effect.

Example:

```
*  cop_config_t config;
*  COP_GetDefaultConfig(&config);
*  config.timeoutCycles = kCOP_2Power8CyclesOr2Power16Cycles
*  ;
*  COP_Init(sim_base, &config);
*
```

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SIM peripheral base address. |
| <i>config</i> | The configuration of COP.    |

### 8.6.3 static void COP\_Disable ( SIM\_Type \* *base* ) [inline], [static]

This dedicated function is not provided. Instead, the COP\_Disable function can be used to disable the COP.

## Function Documentation

Disables the COP module.

This function disables the COP Watchdog. Note: The COP configuration register is a write-once after reset. To disable the COP Watchdog, call this function first.



Parameters

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

#### 8.6.4 void COP\_Refresh ( SIM\_Type \* *base* )

This function feeds the COP.

Parameters

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



## Chapter 9

# DAC: Digital-to-Analog Converter Driver

### 9.1 Overview

The KSDK provides a peripheral driver for the Digital-to-Analog Converter (DAC) module of Kinetis devices.

The DAC driver includes a basic DAC module (converter) and DAC buffer.

The basic DAC module supports operations unique to the DAC converter in each DAC instance. The APIs in this part are used in the initialization phase, which is necessary for enabling the DAC module in the application. The APIs enable/disable the clock, enable/disable the module, and configure the converter. Call the initial APIs to prepare the DAC module for the application.

The DAC buffer operates the DAC hardware buffer. The DAC module supports a hardware buffer to keep a group of DAC values to be converted. This feature supports updating the DAC output value automatically by triggering the buffer read pointer to move in the buffer. Use the APIs to configure the hardware buffer's trigger mode, watermark, work mode, and use size. Additionally, the APIs operate the DMA, interrupts, flags, the pointer (index of buffer), item values, and so on.

### 9.2 Typical use case

#### 9.2.1 Working as a basic DAC without the hardware buffer feature.

```
// ...

// Configures the DAC.
DAC_GetDefaultConfig(&dacConfigStruct);
DAC_Init(DEMO_DAC_INSTANCE, &dacConfigStruct);
DAC_Enable(DEMO_DAC_INSTANCE, true);
DAC_SetBufferReadPointer(DEMO_DAC_INSTANCE, 0U);

// ...

DAC_SetBufferValue(DEMO_DAC_INSTANCE, 0U, dacValue);
```

#### 9.2.2 Working with the hardware buffer.

```
// ...

EnableIRQ(DEMO_DAC_IRQ_ID);

// ...

// Configures the DAC.
DAC_GetDefaultConfig(&dacConfigStruct);
DAC_Init(DEMO_DAC_INSTANCE, &dacConfigStruct);
DAC_Enable(DEMO_DAC_INSTANCE, true);
// Configures the DAC buffer.
DAC_GetDefaultBufferConfig(&dacBufferConfigStruct);
```

## Typical use case

```
DAC_SetBufferConfig(DEMO_DAC_INSTANCE, &dacBufferConfigStruct);
DAC_SetBufferReadPointer(DEMO_DAC_INSTANCE, 0U); // Make sure the read pointer
to the start.
for (index = 0U, dacValue = 0; index < DEMO_DAC_USED_BUFFER_SIZE; index++, dacValue += (0xFFFU /
DEMO_DAC_USED_BUFFER_SIZE))
{
    DAC_SetBufferValue(DEMO_DAC_INSTANCE, index, dacValue);
}
// Clears flags.
#if defined(FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION) && FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
g_DacBufferWatermarkInterruptFlag = false;
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
g_DacBufferReadPointerTopPositionInterruptFlag = false;
g_DacBufferReadPointerBottomPositionInterruptFlag = false;

// Enables interrupts.
mask = 0U;
#if defined(FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION) && FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
mask |= kDAC_BufferWatermarkInterruptEnable;
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
mask |= kDAC_BufferReadPointerTopInterruptEnable |
        kDAC_BufferReadPointerBottomInterruptEnable;
DAC_EnableBuffer(DEMO_DAC_INSTANCE, true);
DAC_EnableBufferInterrupts(DEMO_DAC_INSTANCE, mask);

// ISR for the DAC interrupt.
void DEMO_DAC_IRQ_HANDLER_FUNC(void)
{
    uint32_t flags = DAC_GetBufferStatusFlags(DEMO_DAC_INSTANCE);

    #if defined(FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION) && FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
    if (kDAC_BufferWatermarkFlag == (kDAC_BufferWatermarkFlag & flags))
    {
        g_DacBufferWatermarkInterruptFlag = true;
    }
    #endif // FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION
    if (kDAC_BufferReadPointerTopPositionFlag == (
        kDAC_BufferReadPointerTopPositionFlag & flags))
    {
        g_DacBufferReadPointerTopPositionInterruptFlag = true;
    }
    if (kDAC_BufferReadPointerBottomPositionFlag == (
        kDAC_BufferReadPointerBottomPositionFlag & flags))
    {
        g_DacBufferReadPointerBottomPositionInterruptFlag = true;
    }
    DAC_ClearBufferStatusFlags(DEMO_DAC_INSTANCE, flags); /* Clear flags. */
}
```

## Data Structures

- struct `dac_config_t`  
DAC module configuration. [More...](#)
- struct `dac_buffer_config_t`  
DAC buffer configuration. [More...](#)

## Enumerations

- enum `_dac_buffer_status_flags` {  
    `kDAC_BufferReadPointerTopPositionFlag` = `DAC_SR_DACBFRPTF_MASK`,  
    `kDAC_BufferReadPointerBottomPositionFlag` = `DAC_SR_DACBFRPBF_MASK` }  
DAC buffer flags.

- enum `_dac_buffer_interrupt_enable` {  
`kDAC_BufferReadPointerTopInterruptEnable` = `DAC_C0_DACBTIEN_MASK`,  
`kDAC_BufferReadPointerBottomInterruptEnable` = `DAC_C0_DACBBIEN_MASK` }  
*DAC buffer interrupts.*
- enum `dac_reference_voltage_source_t` {  
`kDAC_ReferenceVoltageSourceVref1` = `0U`,  
`kDAC_ReferenceVoltageSourceVref2` = `1U` }  
*DAC reference voltage source.*
- enum `dac_buffer_trigger_mode_t` {  
`kDAC_BufferTriggerByHardwareMode` = `0U`,  
`kDAC_BufferTriggerBySoftwareMode` = `1U` }  
*DAC buffer trigger mode.*
- enum `dac_buffer_work_mode_t` {  
`kDAC_BufferWorkAsNormalMode` = `0U`,  
`kDAC_BufferWorkAsOneTimeScanMode` }  
*DAC buffer work mode.*

## Driver version

- #define `FSL_DAC_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 1))  
*DAC driver version 2.0.1.*

## Initialization

- void `DAC_Init` (`DAC_Type` \*base, const `dac_config_t` \*config)  
*Initializes the DAC module.*
- void `DAC_Deinit` (`DAC_Type` \*base)  
*De-initializes the DAC module.*
- void `DAC_GetDefaultConfig` (`dac_config_t` \*config)  
*Initializes the DAC user configuration structure.*
- static void `DAC_Enable` (`DAC_Type` \*base, bool enable)  
*Enables the DAC module.*

## Buffer

- static void `DAC_EnableBuffer` (`DAC_Type` \*base, bool enable)  
*Enables the DAC buffer.*
- void `DAC_SetBufferConfig` (`DAC_Type` \*base, const `dac_buffer_config_t` \*config)  
*Configures the CMP buffer.*
- void `DAC_GetDefaultBufferConfig` (`dac_buffer_config_t` \*config)  
*Initializes the DAC buffer configuration structure.*
- static void `DAC_EnableBufferDMA` (`DAC_Type` \*base, bool enable)  
*Enables the DMA for DAC buffer.*
- void `DAC_SetBufferValue` (`DAC_Type` \*base, `uint8_t` index, `uint16_t` value)  
*Sets the value for items in the buffer.*
- static void `DAC_DoSoftwareTriggerBuffer` (`DAC_Type` \*base)  
*Triggers the buffer by software and updates the read pointer of the DAC buffer.*
- static `uint8_t` `DAC_GetBufferReadPointer` (`DAC_Type` \*base)  
*Gets the current read pointer of the DAC buffer.*
- void `DAC_SetBufferReadPointer` (`DAC_Type` \*base, `uint8_t` index)

## Data Structure Documentation

- Sets the current read pointer of the DAC buffer.*
- void [DAC\\_EnableBufferInterrupts](#) (DAC\_Type \*base, uint32\_t mask)  
*Enables interrupts for the DAC buffer.*
- void [DAC\\_DisableBufferInterrupts](#) (DAC\_Type \*base, uint32\_t mask)  
*Disables interrupts for the DAC buffer.*
- uint32\_t [DAC\\_GetBufferStatusFlags](#) (DAC\_Type \*base)  
*Gets the flags of events for the DAC buffer.*
- void [DAC\\_ClearBufferStatusFlags](#) (DAC\_Type \*base, uint32\_t mask)  
*Clears the flags of events for the DAC buffer.*

## 9.3 Data Structure Documentation

### 9.3.1 struct dac\_config\_t

#### Data Fields

- [dac\\_reference\\_voltage\\_source\\_t](#) [referenceVoltageSource](#)  
*Select the DAC reference voltage source.*
- bool [enableLowPowerMode](#)  
*Enable the low-power mode.*

#### 9.3.1.0.0.12 Field Documentation

9.3.1.0.0.12.1 [dac\\_reference\\_voltage\\_source\\_t](#) [dac\\_config\\_t::referenceVoltageSource](#)

9.3.1.0.0.12.2 bool [dac\\_config\\_t::enableLowPowerMode](#)

### 9.3.2 struct dac\_buffer\_config\_t

#### Data Fields

- [dac\\_buffer\\_trigger\\_mode\\_t](#) [triggerMode](#)  
*Select the buffer's trigger mode.*
- [dac\\_buffer\\_work\\_mode\\_t](#) [workMode](#)  
*Select the buffer's work mode.*
- uint8\_t [upperLimit](#)  
*Set the upper limit for buffer index.*

#### 9.3.2.0.0.13 Field Documentation

9.3.2.0.0.13.1 [dac\\_buffer\\_trigger\\_mode\\_t](#) [dac\\_buffer\\_config\\_t::triggerMode](#)

9.3.2.0.0.13.2 [dac\\_buffer\\_work\\_mode\\_t](#) [dac\\_buffer\\_config\\_t::workMode](#)

9.3.2.0.0.13.3 uint8\_t [dac\\_buffer\\_config\\_t::upperLimit](#)

Normally, 0-15 is available for buffer with 16 item.

## 9.4 Macro Definition Documentation

### 9.4.1 #define FSL\_DAC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 9.5 Enumeration Type Documentation

### 9.5.1 enum \_dac\_buffer\_status\_flags

Enumerator

***kDAC\_BufferReadPointerTopPositionFlag*** DAC Buffer Read Pointer Top Position Flag.

***kDAC\_BufferReadPointerBottomPositionFlag*** DAC Buffer Read Pointer Bottom Position Flag.

### 9.5.2 enum \_dac\_buffer\_interrupt\_enable

Enumerator

***kDAC\_BufferReadPointerTopInterruptEnable*** DAC Buffer Read Pointer Top Flag Interrupt Enable.

***kDAC\_BufferReadPointerBottomInterruptEnable*** DAC Buffer Read Pointer Bottom Flag Interrupt Enable.

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

Enumerator

***kDAC\_ReferenceVoltageSourceVref1*** The DAC selects DACREF\_1 as the reference voltage.

***kDAC\_ReferenceVoltageSourceVref2*** The DAC selects DACREF\_2 as the reference voltage.

### 9.5.4 enum dac\_buffer\_trigger\_mode\_t

Enumerator

***kDAC\_BufferTriggerByHardwareMode*** The DAC hardware trigger is selected.

***kDAC\_BufferTriggerBySoftwareMode*** The DAC software trigger is selected.

### 9.5.5 enum dac\_buffer\_work\_mode\_t

Enumerator

***kDAC\_BufferWorkAsNormalMode*** Normal mode.

***kDAC\_BufferWorkAsOneTimeScanMode*** One-Time Scan mode.

## Function Documentation

### 9.6 Function Documentation

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

This function initializes the DAC module, including:

- Enabling the clock for DAC module.
- Configuring the DAC converter with a user configuration.
- Enabling the DAC module.

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | DAC peripheral base address.                                |
| <i>config</i> | Pointer to the configuration structure. See "dac_config_t". |

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

This function de-initializes the DAC module, including:

- Disabling the DAC module.
- Disabling the clock for the DAC module.

Parameters

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

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

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

```
* config->referenceVoltageSource = kDAC_ReferenceVoltageSourceVref2;  
* config->enableLowPowerMode = false;  
*
```

Parameters

|               |   |
|---------------|---|
| <i>config</i> | Pointer to the configuration structure. See "dac_config_t". |
|---------------|---|

#### 9.6.4 static void DAC\_Enable ( DAC\_Type \* *base*, bool *enable* ) [inline], [static]



## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | DAC peripheral base address.  |
| <i>enable</i> | Enables/disables the feature. |

### 9.6.5 static void DAC\_EnableBuffer ( DAC\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | DAC peripheral base address.  |
| <i>enable</i> | Enables/disables the feature. |

### 9.6.6 void DAC\_SetBufferConfig ( DAC\_Type \* *base*, const dac\_buffer\_config\_t \* *config* )

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | DAC peripheral base address.                                       |
| <i>config</i> | Pointer to the configuration structure. See "dac_buffer_config_t". |

### 9.6.7 void DAC\_GetDefaultBufferConfig ( dac\_buffer\_config\_t \* *config* )

This function initializes the DAC buffer configuration structure to a default value. The default values are:

```
* config->triggerMode = kDAC_BufferTriggerBySoftwareMode;
* config->watermark   = kDAC_BufferWatermark1Word;
* config->workMode     = kDAC_BufferWorkAsNormalMode;
* config->upperLimit   = DAC_DATL_COUNT - 1U;
*
```

## Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to the configuration structure. See "dac_buffer_config_t". |
|---------------|--|

### 9.6.8 static void DAC\_EnableBufferDMA ( DAC\_Type \* *base*, bool *enable* ) [inline], [static]

## Function Documentation

### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | DAC peripheral base address.  |
| <i>enable</i> | Enables/disables the feature. |

### 9.6.9 void DAC\_SetBufferValue ( DAC\_Type \* *base*, uint8\_t *index*, uint16\_t *value* )

### Parameters

|              |  |
|--------------|--|
| <i>base</i>  | DAC peripheral base address.   |
| <i>index</i> | Setting index for items in the buffer. The available index should not exceed the size of the DAC buffer. |
| <i>value</i> | Setting value for items in the buffer. 12-bits are available.  |

### 9.6.10 static void DAC\_DoSoftwareTriggerBuffer ( DAC\_Type \* *base* ) [inline], [static]

This function triggers the function by software. The read pointer of the DAC buffer is updated with one step after this function is called. Changing the read pointer depends on the buffer's work mode.

### Parameters

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

### 9.6.11 static uint8\_t DAC\_GetBufferReadPointer ( DAC\_Type \* *base* ) [inline], [static]

This function gets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated by software trigger or hardware trigger.

### Parameters

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

### Returns

Current read pointer of DAC buffer.

**9.6.12 void DAC\_SetBufferReadPointer ( DAC\_Type \* *base*, uint8\_t *index* )**

This function sets the current read pointer of the DAC buffer. The current output value depends on the item indexed by the read pointer. It is updated by software trigger or hardware trigger. After the read pointer changes, the DAC output value also changes.

Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | DAC peripheral base address.         |
| <i>index</i> | Setting index value for the pointer. |

**9.6.13 void DAC\_EnableBufferInterrupts ( DAC\_Type \* *base*, uint32\_t *mask* )**

Parameters

|             |  |
|-------------|--|
| <i>base</i> | DAC peripheral base address.                                   |
| <i>mask</i> | Mask value for interrupts. See "_dac_buffer_interrupt_enable". |

**9.6.14 void DAC\_DisableBufferInterrupts ( DAC\_Type \* *base*, uint32\_t *mask* )**

Parameters

|             |  |
|-------------|--|
| <i>base</i> | DAC peripheral base address.                                   |
| <i>mask</i> | Mask value for interrupts. See "_dac_buffer_interrupt_enable". |

**9.6.15 uint32\_t DAC\_GetBufferStatusFlags ( DAC\_Type \* *base* )**

Parameters

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

Returns

Mask value for the asserted flags. See "\_dac\_buffer\_status\_flags".

**9.6.16 void DAC\_ClearBufferStatusFlags ( DAC\_Type \* *base*, uint32\_t *mask* )**

## Function Documentation

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | DAC peripheral base address.                            |
| <i>mask</i> | Mask value for flags. See "_dac_buffer_status_flags_t". |

## Chapter 10

# DMA: Direct Memory Access Controller Driver

### 10.1 Overview

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

### 10.2 Typical use case

#### 10.2.1 DMA Operation

```
dma_transfer_config_t transferConfig;
uint32_t transferDone = false;

DMA_Init(DMA0);
DMA_CreateHandle(&g_DMA_Handle, DMA0, channel);
DMA_InstallCallback(&g_DMA_Handle, DMA_Callback, &transferDone);
DMA_PrepareTransfer(&transferConfig, srcAddr, srcWidth, destAddr, destWidth,
    transferBytes,
    kDMA_MemoryToMemory);
DMA_SubmitTransfer(&g_DMA_Handle, &transferConfig, true);
DMA_StartTransfer(&g_DMA_Handle);
/* Wait for DMA transfer finish */
while (transferDone != true);
```

### Data Structures

- struct [dma\\_transfer\\_config\\_t](#)  
*DMA transfer configuration structure. [More...](#)*
- struct [dma\\_channel\\_link\\_config\\_t](#)  
*DMA transfer configuration structure. [More...](#)*
- struct [dma\\_handle\\_t](#)  
*DMA DMA handle structure. [More...](#)*

### Typedefs

- typedef void(\* [dma\\_callback](#))(struct \_dma\_handle \*handle, void \*userData)  
*Callback function prototype for the DMA driver.*

### Enumerations

- enum [\\_dma\\_channel\\_status\\_flags](#) {  
    [kDMA\\_TransactionsBCRFlag](#) = DMA\_DSR\_BCR\_BCR\_MASK,  
    [kDMA\\_TransactionsDoneFlag](#) = DMA\_DSR\_BCR\_DONE\_MASK,  
    [kDMA\\_TransactionsBusyFlag](#) = DMA\_DSR\_BCR\_BSY\_MASK,  
    [kDMA\\_TransactionsRequestFlag](#) = DMA\_DSR\_BCR\_REQ\_MASK,  
    [kDMA\\_BusErrorOnDestinationFlag](#) = DMA\_DSR\_BCR\_BED\_MASK,  
    [kDMA\\_BusErrorOnSourceFlag](#) = DMA\_DSR\_BCR\_BES\_MASK,  
    [kDMA\\_ConfigurationErrorFlag](#) = DMA\_DSR\_BCR\_CE\_MASK }

## Typical use case

- status flag for the DMA driver.*
  - enum `dma_transfer_size_t` {  
    `kDMA_Transfersize32bits` = 0x0U,  
    `kDMA_Transfersize8bits`,  
    `kDMA_Transfersize16bits` }
- DMA transfer size type.*
  - enum `dma_modulo_t` {  
    `kDMA_ModuloDisable` = 0x0U,  
    `kDMA_Modulo16Bytes`,  
    `kDMA_Modulo32Bytes`,  
    `kDMA_Modulo64Bytes`,  
    `kDMA_Modulo128Bytes`,  
    `kDMA_Modulo256Bytes`,  
    `kDMA_Modulo512Bytes`,  
    `kDMA_Modulo1KBytes`,  
    `kDMA_Modulo2KBytes`,  
    `kDMA_Modulo4KBytes`,  
    `kDMA_Modulo8KBytes`,  
    `kDMA_Modulo16KBytes`,  
    `kDMA_Modulo32KBytes`,  
    `kDMA_Modulo64KBytes`,  
    `kDMA_Modulo128KBytes`,  
    `kDMA_Modulo256KBytes` }
- Configuration type for the DMA modulo.*
  - enum `dma_channel_link_type_t` {  
    `kDMA_ChannelLinkDisable` = 0x0U,  
    `kDMA_ChannelLinkChannel1AndChannel2`,  
    `kDMA_ChannelLinkChannel1`,  
    `kDMA_ChannelLinkChannel1AfterBCR0` }
- DMA channel link type.*
  - enum `dma_transfer_type_t` {  
    `kDMA_MemoryToMemory` = 0x0U,  
    `kDMA_PeripheralToMemory`,  
    `kDMA_MemoryToPeripheral` }
- DMA transfer type.*
  - enum `dma_transfer_options_t` {  
    `kDMA_NoOptions` = 0x0U,  
    `kDMA_EnableInterrupt` }
- DMA transfer options.*
  - enum `_dma_transfer_status`
- DMA transfer status.*

## Driver version

- #define `FSL_DMA_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)  
*DMA driver version 2.0.1.*

## DMA Initialization and De-initialization

- void [DMA\\_Init](#) (DMA\_Type \*base)  
*Initializes the DMA peripheral.*
- void [DMA\\_Deinit](#) (DMA\_Type \*base)  
*Deinitializes the DMA peripheral.*

## DMA Channel Operation

- void [DMA\\_ResetChannel](#) (DMA\_Type \*base, uint32\_t channel)  
*Resets the DMA channel.*
- void [DMA\\_SetTransferConfig](#) (DMA\_Type \*base, uint32\_t channel, const [dma\\_transfer\\_config\\_t](#) \*config)  
*Configures the DMA transfer attribute.*
- void [DMA\\_SetChannelLinkConfig](#) (DMA\_Type \*base, uint32\_t channel, const [dma\\_channel\\_link\\_config\\_t](#) \*config)  
*Configures the DMA channel link feature.*
- static void [DMA\\_SetSourceAddress](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t srcAddr)  
*Sets the DMA source address for the DMA transfer.*
- static void [DMA\\_SetDestinationAddress](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t destAddr)  
*Sets the DMA destination address for the DMA transfer.*
- static void [DMA\\_SetTransferSize](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t size)  
*Sets the DMA transfer size for the DMA transfer.*
- void [DMA\\_SetModulo](#) (DMA\_Type \*base, uint32\_t channel, [dma\\_modulo\\_t](#) srcModulo, [dma\\_modulo\\_t](#) destModulo)  
*Sets the DMA modulo for the DMA transfer.*
- static void [DMA\\_EnableCycleSteal](#) (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables the DMA cycle steal for the DMA transfer.*
- static void [DMA\\_EnableAutoAlign](#) (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables the DMA auto align for the DMA transfer.*
- static void [DMA\\_EnableAsyncRequest](#) (DMA\_Type \*base, uint32\_t channel, bool enable)  
*Enables the DMA async request for the DMA transfer.*
- static void [DMA\\_EnableInterrupts](#) (DMA\_Type \*base, uint32\_t channel)  
*Enables an interrupt for the DMA transfer.*
- static void [DMA\\_DisableInterrupts](#) (DMA\_Type \*base, uint32\_t channel)  
*Disables an interrupt for the DMA transfer.*

## DMA Channel Transfer Operation

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

## DMA Channel Status Operation

- static uint32\_t [DMA\\_GetRemainingBytes](#) (DMA\_Type \*base, uint32\_t channel)  
*Gets the remaining bytes of the current DMA transfer.*
- static uint32\_t [DMA\\_GetChannelStatusFlags](#) (DMA\_Type \*base, uint32\_t channel)

## Data Structure Documentation

- Gets the DMA channel status flags.*
- static void [DMA\\_ClearChannelStatusFlags](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Clears the DMA channel status flags.*

## DMA Channel Transactional Operation

- 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)  
*Sets the DMA callback function.*
- void [DMA\\_PrepareTransfer](#) (dma\_transfer\_config\_t \*config, void \*srcAddr, uint32\_t srcWidth, void \*destAddr, uint32\_t destWidth, uint32\_t transferBytes, dma\_transfer\_type\_t type)  
*Prepares the DMA transfer configuration structure.*
- status\_t [DMA\\_SubmitTransfer](#) (dma\_handle\_t \*handle, const dma\_transfer\_config\_t \*config, uint32\_t options)  
*Submits the DMA transfer request.*
- static void [DMA\\_StartTransfer](#) (dma\_handle\_t \*handle)  
*DMA starts a transfer.*
- static void [DMA\\_StopTransfer](#) (dma\_handle\_t \*handle)  
*DMA stops a transfer.*
- void [DMA\\_AbortTransfer](#) (dma\_handle\_t \*handle)  
*DMA aborts a transfer.*
- void [DMA\\_HandleIRQ](#) (dma\_handle\_t \*handle)  
*DMA IRQ handler for current transfer complete.*

## 10.3 Data Structure Documentation

### 10.3.1 struct dma\_transfer\_config\_t

#### Data Fields

- uint32\_t [srcAddr](#)  
*DMA transfer source address.*
- uint32\_t [destAddr](#)  
*DMA destination address.*
- bool [enableSrcIncrement](#)  
*Source address increase after each transfer.*
- [dma\\_transfer\\_size\\_t](#) [srcSize](#)  
*Source transfer size unit.*
- bool [enableDestIncrement](#)  
*Destination address increase after each transfer.*
- [dma\\_transfer\\_size\\_t](#) [destSize](#)  
*Destination transfer unit.*
- uint32\_t [transferSize](#)  
*The number of bytes to be transferred.*



**10.3.1.0.0.14 Field Documentation****10.3.1.0.0.14.1** uint32\_t dma\_transfer\_config\_t::srcAddr**10.3.1.0.0.14.2** uint32\_t dma\_transfer\_config\_t::destAddr**10.3.1.0.0.14.3** bool dma\_transfer\_config\_t::enableSrcIncrement**10.3.1.0.0.14.4** dma\_transfer\_size\_t dma\_transfer\_config\_t::srcSize**10.3.1.0.0.14.5** bool dma\_transfer\_config\_t::enableDestIncrement**10.3.1.0.0.14.6** dma\_transfer\_size\_t dma\_transfer\_config\_t::destSize**10.3.1.0.0.14.7** uint32\_t dma\_transfer\_config\_t::transferSize**10.3.2 struct dma\_channel\_link\_config\_t****Data Fields**

- [dma\\_channel\\_link\\_type\\_t linkType](#)  
*Channel link type.*
- uint32\_t [channel1](#)  
*The index of channel 1.*
- uint32\_t [channel2](#)  
*The index of channel 2.*

**10.3.2.0.0.15 Field Documentation****10.3.2.0.0.15.1** dma\_channel\_link\_type\_t dma\_channel\_link\_config\_t::linkType**10.3.2.0.0.15.2** uint32\_t dma\_channel\_link\_config\_t::channel1**10.3.2.0.0.15.3** uint32\_t dma\_channel\_link\_config\_t::channel2**10.3.3 struct dma\_handle\_t****Data Fields**

- DMA\_Type \* [base](#)  
*DMA peripheral address.*
- uint8\_t [channel](#)  
*DMA channel used.*
- [dma\\_callback callback](#)  
*DMA callback function.*
- void \* [userData](#)  
*Callback parameter.*

## Enumeration Type Documentation

### 10.3.3.0.0.16 Field Documentation

10.3.3.0.0.16.1 DMA\_Type\* dma\_handle\_t::base

10.3.3.0.0.16.2 uint8\_t dma\_handle\_t::channel

10.3.3.0.0.16.3 dma\_callback dma\_handle\_t::callback

10.3.3.0.0.16.4 void\* dma\_handle\_t::userData

## 10.4 Macro Definition Documentation

10.4.1 #define FSL\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 10.5 Typedef Documentation

10.5.1 typedef void(\* dma\_callback)(struct \_dma\_handle \*handle, void \*userData)

## 10.6 Enumeration Type Documentation

10.6.1 enum \_dma\_channel\_status\_flags

### Enumerator

*kDMA\_TransactionsBCRFlag* Contains the number of bytes yet to be transferred for a given block.

*kDMA\_TransactionsDoneFlag* Transactions Done.

*kDMA\_TransactionsBusyFlag* Transactions Busy.

*kDMA\_TransactionsRequestFlag* Transactions Request.

*kDMA\_BusErrorOnDestinationFlag* Bus Error on Destination.

*kDMA\_BusErrorOnSourceFlag* Bus Error on Source.

*kDMA\_ConfigurationErrorFlag* Configuration Error.

10.6.2 enum dma\_transfer\_size\_t

### Enumerator

*kDMA\_Transfersize32bits* 32 bits are transferred for every read/write

*kDMA\_Transfersize8bits* 8 bits are transferred for every read/write

*kDMA\_Transfersize16bits* 16 bits are transferred for every read/write

10.6.3 enum dma\_modulo\_t

### Enumerator

*kDMA\_ModuloDisable* Buffer disabled.

***kDMA\_Modulo16Bytes*** Circular buffer size is 16 bytes.  
***kDMA\_Modulo32Bytes*** Circular buffer size is 32 bytes.  
***kDMA\_Modulo64Bytes*** Circular buffer size is 64 bytes.  
***kDMA\_Modulo128Bytes*** Circular buffer size is 128 bytes.  
***kDMA\_Modulo256Bytes*** Circular buffer size is 256 bytes.  
***kDMA\_Modulo512Bytes*** Circular buffer size is 512 bytes.  
***kDMA\_Modulo1KBytes*** Circular buffer size is 1 KB.  
***kDMA\_Modulo2KBytes*** Circular buffer size is 2 KB.  
***kDMA\_Modulo4KBytes*** Circular buffer size is 4 KB.  
***kDMA\_Modulo8KBytes*** Circular buffer size is 8 KB.  
***kDMA\_Modulo16KBytes*** Circular buffer size is 16 KB.  
***kDMA\_Modulo32KBytes*** Circular buffer size is 32 KB.  
***kDMA\_Modulo64KBytes*** Circular buffer size is 64 KB.  
***kDMA\_Modulo128KBytes*** Circular buffer size is 128 KB.  
***kDMA\_Modulo256KBytes*** Circular buffer size is 256 KB.

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

Enumerator

***kDMA\_ChannelLinkDisable*** No channel link.  
***kDMA\_ChannelLinkChannel1AndChannel2*** Perform a link to channel LCH1 after each cycle-steal transfer. followed by a link to LCH2 after the BCR decrements to 0.  
***kDMA\_ChannelLinkChannel1*** Perform a link to LCH1 after each cycle-steal transfer.  
***kDMA\_ChannelLinkChannel1AfterBCR0*** Perform a link to LCH1 after the BCR decrements.

#### 10.6.5 enum dma\_transfer\_type\_t

Enumerator

***kDMA\_MemoryToMemory*** Memory to Memory transfer.  
***kDMA\_PeripheralToMemory*** Peripheral to Memory transfer.  
***kDMA\_MemoryToPeripheral*** Memory to Peripheral transfer.

#### 10.6.6 enum dma\_transfer\_options\_t

Enumerator

***kDMA\_NoOptions*** Transfer without options.  
***kDMA\_EnableInterrupt*** Enable interrupt while transfer complete.

### 10.7 Function Documentation

#### 10.7.1 void DMA\_Init ( DMA\_Type \* *base* )

This function ungates the DMA clock.

## Parameters

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

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

This function gates the DMA clock.

## Parameters

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

**10.7.3 void DMA\_ResetChannel ( DMA\_Type \* *base*, uint32\_t *channel* )**

Sets all register values to reset values and enables the cycle steal and auto stop channel request features.

## Parameters

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

**10.7.4 void DMA\_SetTransferConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const dma\_transfer\_config\_t \* *config* )**

This function configures the transfer attribute including the source address, destination address, transfer size, and so on. This example shows how to set up the the [dma\\_transfer\\_config\\_t](#) parameters and how to call the DMA\_ConfigBasicTransfer function.

```
*  dma_transfer_config_t transferConfig;
*  memset(&transferConfig, 0, sizeof(transferConfig));
*  transferConfig.srcAddr = (uint32_t)srcAddr;
*  transferConfig.destAddr = (uint32_t)destAddr;
*  transferConfig.enableSrcIncrement = true;
*  transferConfig.enableDestIncrement = true;
*  transferConfig.srcSize = kDMA_Transfersize32bits;
*  transferConfig.destSize = kDMA_Transfersize32bits;
*  transferConfig.transferSize = sizeof(uint32_t) * BUFF_LENGTH;
*  DMA_SetTransferConfig(DMA0, 0, &transferConfig);
*
```

## Function Documentation

### Parameters

|                |  |
|----------------|--|
| <i>base</i>    | DMA peripheral base address.                         |
| <i>channel</i> | DMA channel number.                                  |
| <i>config</i>  | Pointer to the DMA transfer configuration structure. |

### 10.7.5 void DMA\_SetChannelLinkConfig ( DMA\_Type \* *base*, uint32\_t *channel*, const dma\_channel\_link\_config\_t \* *config* )

This function allows DMA channels to have their transfers linked. The current DMA channel triggers a DMA request to the linked channels (LCH1 or LCH2) depending on the channel link type. Perform a link to channel LCH1 after each cycle-steal transfer followed by a link to LCH2 after the BCR decrements to 0 if the type is kDMA\_ChannelLinkChannel1AndChannel2. Perform a link to LCH1 after each cycle-steal transfer if the type is kDMA\_ChannelLinkChannel1. Perform a link to LCH1 after the BCR decrements to 0 if the type is kDMA\_ChannelLinkChannel1AfterBCR0.

### Parameters

|                |  |
|----------------|--|
| <i>base</i>    | DMA peripheral base address.                         |
| <i>channel</i> | DMA channel number.                                  |
| <i>config</i>  | Pointer to the channel link configuration structure. |

### 10.7.6 static void DMA\_SetSourceAddress ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *srcAddr* ) [inline], [static]

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |
| <i>srcAddr</i> | DMA source address.          |

### 10.7.7 static void DMA\_SetDestinationAddress ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *destAddr* ) [inline], [static]

## Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | DMA peripheral base address. |
| <i>channel</i>  | DMA channel number.          |
| <i>destAddr</i> | DMA destination address.     |

### 10.7.8 static void DMA\_SetTransferSize ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *size* ) [inline], [static]

## Parameters

|                |  |
|----------------|--|
| <i>base</i>    | DMA peripheral base address.           |
| <i>channel</i> | DMA channel number.                    |
| <i>size</i>    | The number of bytes to be transferred. |

### 10.7.9 void DMA\_SetModulo ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_modulo\_t *srcModulo*, dma\_modulo\_t *destModulo* )

This function defines a specific address range specified to be the value after (SAR + SSIZE)/(DAR + DS-IZE) calculation is performed or the original register value. It provides the ability to implement a circular data queue easily.

## Parameters

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

### 10.7.10 static void DMA\_EnableCycleSteal ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

If the cycle steal feature is enabled (true), the DMA controller forces a single read/write transfer per request, or it continuously makes read/write transfers until the BCR decrements to 0.

## Function Documentation

### Parameters

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

#### 10.7.11 static void DMA\_EnableAutoAlign ( DMA\_Type \* *base*, uint32\_t *channel*, bool *enable* ) [inline], [static]

If the auto align feature is enabled (true), the appropriate address register increments, regardless of DINC or SINC.

### Parameters

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

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

If the async request feature is enabled (true), the DMA supports asynchronous DREQs while the MCU is in stop mode.

### Parameters

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

#### 10.7.13 static void DMA\_EnableInterrupts ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]



## Parameters

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

**10.7.14 static void DMA\_DisableInterrupts ( DMA\_Type \* *base*, uint32\_t *channel* )**  
**[inline], [static]**

## Parameters

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

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

## Parameters

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

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

## Parameters

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

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

This function starts only one read/write iteration.

## Function Documentation

### Parameters

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

**10.7.18** `static uint32_t DMA_GetRemainingBytes ( DMA_Type * base, uint32_t channel ) [inline], [static]`

### Parameters

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

### Returns

The number of bytes which have not been transferred yet.

**10.7.19** `static uint32_t DMA_GetChannelStatusFlags ( DMA_Type * base, uint32_t channel ) [inline], [static]`

### Parameters

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

### Returns

The mask of the channel status. Use the `_dma_channel_status_flags` type to decode the return 32 bit variables.

**10.7.20** `static void DMA_ClearChannelStatusFlags ( DMA_Type * base, uint32_t channel, uint32_t mask ) [inline], [static]`

## Parameters

|                |  |
|----------------|--|
| <i>base</i>    | DMA peripheral base address.   |
| <i>channel</i> | DMA channel number.  |
| <i>mask</i>    | The mask of the channel status to be cleared. Use the defined <code>_dma_channel_status-flags</code> type. |

### 10.7.21 void DMA\_CreateHandle ( dma\_handle\_t \* *handle*, DMA\_Type \* *base*, uint32\_t *channel* )

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

## Parameters

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

### 10.7.22 void DMA\_SetCallback ( dma\_handle\_t \* *handle*, dma\_callback *callback*, void \* *userData* )

This callback is called in the DMA IRQ handler. Use the callback to do something after the current transfer complete.

## Parameters

|                 |   |
|-----------------|---|
| <i>handle</i>   | DMA handle pointer.   |
| <i>callback</i> | DMA callback function pointer.  |
| <i>userData</i> | Parameter for callback function. If it is not needed, just set to NULL. |

### 10.7.23 void DMA\_PrepareTransfer ( dma\_transfer\_config\_t \* *config*, void \* *srcAddr*, uint32\_t *srcWidth*, void \* *destAddr*, uint32\_t *destWidth*, uint32\_t *transferBytes*, dma\_transfer\_type\_t *type* )

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

## Function Documentation

### Parameters

|                      |   |
|----------------------|---|
| <i>config</i>        | Pointer to the user configuration structure of type <a href="#">dma_transfer_config_t</a> . |
| <i>srcAddr</i>       | DMA transfer source address.  |
| <i>srcWidth</i>      | DMA transfer source address width (byte).   |
| <i>destAddr</i>      | DMA transfer destination address.   |
| <i>destWidth</i>     | DMA transfer destination address width (byte).  |
| <i>transferBytes</i> | DMA transfer bytes to be transferred.   |
| <i>type</i>          | DMA transfer type.  |

### 10.7.24 **status\_t DMA\_SubmitTransfer ( dma\_handle\_t \* *handle*, const dma\_transfer\_config\_t \* *config*, uint32\_t *options* )**

This function submits the DMA transfer request according to the transfer configuration structure.

### Parameters

|                |  |
|----------------|--|
| <i>handle</i>  | DMA handle pointer.  |
| <i>config</i>  | Pointer to DMA transfer configuration structure.                                     |
| <i>options</i> | Additional configurations for transfer. Use the defined dma_transfer_options_t type. |

### Return values

|                            |  |
|----------------------------|--|
| <i>kStatus_DMA_Success</i> | It indicates that the DMA submit transfer request succeeded.               |
| <i>kStatus_DMA_Busy</i>    | It indicates that the DMA is busy. Submit transfer request is not allowed. |

### Note

This function can't process multi transfer request.

### 10.7.25 **static void DMA\_StartTransfer ( dma\_handle\_t \* *handle* ) [inline], [static]**

This function enables the channel request. Call this function after submitting a transfer request.

## Parameters

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

## Return values

|                            |   |
|----------------------------|---|
| <i>kStatus_DMA_Success</i> | It indicates that the DMA start transfer succeed. |
| <i>kStatus_DMA_Busy</i>    | It indicates that the DMA has started a transfer. |

### 10.7.26 static void DMA\_StopTransfer ( dma\_handle\_t \* *handle* ) [inline], [static]

This function disables the channel request to stop a DMA transfer. The transfer can be resumed by calling the DMA\_StartTransfer.

## Parameters

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

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

This function disables the channel request and clears all status bits. Submit another transfer after calling this API.

## Parameters

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

### 10.7.28 void DMA\_HandleIRQ ( dma\_handle\_t \* *handle* )

This function clears the channel interrupt flag and calls the callback function if it is not NULL.

## Parameters

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



# Chapter 11

## DMAMUX: Direct Memory Access Multiplexer Driver

### 11.1 Overview

The KSDK provides a peripheral driver for the Direct Memory Access Multiplexer (DMAMUX) of Kinetis devices.

### 11.2 Typical use case

#### 11.2.1 DMAMUX Operation

```
DMAMUX_Init(DMAMUX0);
DMAMUX_SetSource(DMAMUX0, channel, source);
DMAMUX_EnableChannel(DMAMUX0, channel);
...
DMAMUX_DisableChannel(DMAMUX, channel);
DMAMUX_Deinit(DMAMUX0);
```

### Driver version

- #define `FSL_DMAMUX_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)  
*DMAMUX driver version 2.0.1.*

### DMAMUX Initialize and De-initialize

- void `DMAMUX_Init` (DMAMUX\_Type \*base)  
*Initializes DMAMUX peripheral.*
- void `DMAMUX_Deinit` (DMAMUX\_Type \*base)  
*Deinitializes DMAMUX peripheral.*

### DMAMUX Channel Operation

- static void `DMAMUX_EnableChannel` (DMAMUX\_Type \*base, uint32\_t channel)  
*Enable DMAMUX channel.*
- static void `DMAMUX_DisableChannel` (DMAMUX\_Type \*base, uint32\_t channel)  
*Disable DMAMUX channel.*
- static void `DMAMUX_SetSource` (DMAMUX\_Type \*base, uint32\_t channel, uint32\_t source)  
*Configure DMAMUX channel source.*

### 11.3 Macro Definition Documentation

#### 11.3.1 #define `FSL_DMAMUX_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)

### 11.4 Function Documentation

#### 11.4.1 void DMAMUX\_Init ( DMAMUX\_Type \* *base* )

This function ungates the DMAMUX clock.



## Parameters

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

**11.4.2 void DMAMUX\_Deinit ( DMAMUX\_Type \* *base* )**

This function gate the DMAMUX clock.

## Parameters

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

**11.4.3 static void DMAMUX\_EnableChannel ( DMAMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function enable DMAMUX channel to work.

## Parameters

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

**11.4.4 static void DMAMUX\_DisableChannel ( DMAMUX\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function disable DMAMUX channel.

## Note

User must disable DMAMUX channel before configuring it.

## Parameters

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

## Function Documentation

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

**11.4.5 static void DMAMUX\_SetSource ( DMAMUX\_Type \* *base*, uint32\_t *channel*, uint32\_t *source* ) [inline], [static]**

### Parameters

|                |   |
|----------------|---|
| <i>base</i>    | DMAMUX peripheral base address.                       |
| <i>channel</i> | DMAMUX channel number.                                |
| <i>source</i>  | Channel source which is used to trigger DMA transfer. |

## Chapter 12

### C90TFS Flash Driver

#### 12.1 Overview

The flash provides the C90TFS Flash driver of Kinetis devices with the C90TFS Flash module inside. The flash driver provides general APIs to handle specific operations on C90TFS/FTFx Flash module. The user can use those APIs directly in the application. In addition, it provides internal functions called by the driver. Although these functions are not meant to be called from the user's application directly, the APIs can still be used.

#### Data Structures

- struct [flash\\_execute\\_in\\_ram\\_function\\_config\\_t](#)  
*Flash execute-in-RAM function information. [More...](#)*
- struct [flash\\_swap\\_state\\_config\\_t](#)  
*Flash Swap information. [More...](#)*
- struct [flash\\_swap\\_ifr\\_field\\_config\\_t](#)  
*Flash Swap IFR fields. [More...](#)*
- union [flash\\_swap\\_ifr\\_field\\_data\\_t](#)  
*Flash Swap IFR field data. [More...](#)*
- struct [flash\\_operation\\_config\\_t](#)  
*Active flash information for current operation. [More...](#)*
- struct [flash\\_config\\_t](#)  
*Flash driver state information. [More...](#)*

#### Typedefs

- typedef void(\* [flash\\_callback\\_t](#))(void)  
*callback type used for pflash block*

#### Enumerations

- enum [flash\\_margin\\_value\\_t](#) {  
    [kFLASH\\_MarginValueNormal](#),  
    [kFLASH\\_MarginValueUser](#),  
    [kFLASH\\_MarginValueFactory](#),  
    [kFLASH\\_MarginValueInvalid](#) }  
*Enumeration for supported flash margin levels.*
- enum [flash\\_security\\_state\\_t](#) {  
    [kFLASH\\_SecurityStateNotSecure](#),  
    [kFLASH\\_SecurityStateBackdoorEnabled](#),  
    [kFLASH\\_SecurityStateBackdoorDisabled](#) }  
*Enumeration for the three possible flash security states.*

## Overview

- enum `flash_protection_state_t` {  
    `kFLASH_ProtectionStateUnprotected`,  
    `kFLASH_ProtectionStateProtected`,  
    `kFLASH_ProtectionStateMixed` }  
    *Enumeration for the three possible flash protection levels.*
- enum `flash_execute_only_access_state_t` {  
    `kFLASH_AccessStateUnLimited`,  
    `kFLASH_AccessStateExecuteOnly`,  
    `kFLASH_AccessStateMixed` }  
    *Enumeration for the three possible flash execute access levels.*
- enum `flash_property_tag_t` {  
    `kFLASH_PropertyPflashSectorSize` = 0x00U,  
    `kFLASH_PropertyPflashTotalSize` = 0x01U,  
    `kFLASH_PropertyPflashBlockSize` = 0x02U,  
    `kFLASH_PropertyPflashBlockCount` = 0x03U,  
    `kFLASH_PropertyPflashBlockBaseAddr` = 0x04U,  
    `kFLASH_PropertyPflashFacSupport` = 0x05U,  
    `kFLASH_PropertyPflashAccessSegmentSize` = 0x06U,  
    `kFLASH_PropertyPflashAccessSegmentCount` = 0x07U,  
    `kFLASH_PropertyFlexRamBlockBaseAddr` = 0x08U,  
    `kFLASH_PropertyFlexRamTotalSize` = 0x09U,  
    `kFLASH_PropertyDflashSectorSize` = 0x10U,  
    `kFLASH_PropertyDflashTotalSize` = 0x11U,  
    `kFLASH_PropertyDflashBlockSize` = 0x12U,  
    `kFLASH_PropertyDflashBlockCount` = 0x13U,  
    `kFLASH_PropertyDflashBlockBaseAddr` = 0x14U }  
    *Enumeration for various flash properties.*
- enum `_flash_execute_in_ram_function_constants` {  
    `kFLASH_ExecuteInRamFunctionMaxSizeInWords` = 16U,  
    `kFLASH_ExecuteInRamFunctionTotalNum` = 2U }  
    *Constants for execute-in-RAM flash function.*
- enum `flash_read_resource_option_t` {  
    `kFLASH_ResourceOptionFlashIfr`,  
    `kFLASH_ResourceOptionVersionId` = 0x01U }  
    *Enumeration for the two possible options of flash read resource command.*
- enum `_flash_read_resource_range` {  
    `kFLASH_ResourceRangePflashIfrSizeInBytes` = 256U,  
    `kFLASH_ResourceRangeVersionIdSizeInBytes` = 8U,  
    `kFLASH_ResourceRangeVersionIdStart` = 0x00U,  
    `kFLASH_ResourceRangeVersionIdEnd` = 0x07U ,  
    `kFLASH_ResourceRangePflashSwapIfrEnd`,  
    `kFLASH_ResourceRangeDflashIfrStart` = 0x800000U,  
    `kFLASH_ResourceRangeDflashIfrEnd` = 0x8003FFU }  
    *Enumeration for the range of special-purpose flash resource.*
- enum `flash_flexram_function_option_t` {  
    `kFLASH_FlexramFunctionOptionAvailableAsRam` = 0xFFU,

- `kFLASH_FlexramFunctionOptionAvailableForEeprom = 0x00U }`  
*Enumeration for the two possible options of set flexram function command.*
- enum `_flash_acceleration_ram_property`  
*Enumeration for acceleration RAM property.*
- enum `flash_swap_function_option_t` {  
`kFLASH_SwapFunctionOptionEnable = 0x00U,`  
`kFLASH_SwapFunctionOptionDisable = 0x01U }`  
*Enumeration for the possible options of Swap function.*
- enum `flash_swap_control_option_t` {  
`kFLASH_SwapControlOptionInitializeSystem = 0x01U,`  
`kFLASH_SwapControlOptionSetInUpdateState = 0x02U,`  
`kFLASH_SwapControlOptionSetInCompleteState = 0x04U,`  
`kFLASH_SwapControlOptionReportStatus = 0x08U,`  
`kFLASH_SwapControlOptionDisableSystem = 0x10U }`  
*Enumeration for the possible options of Swap Control commands.*
- enum `flash_swap_state_t` {  
`kFLASH_SwapStateUninitialized = 0x00U,`  
`kFLASH_SwapStateReady = 0x01U,`  
`kFLASH_SwapStateUpdate = 0x02U,`  
`kFLASH_SwapStateUpdateErased = 0x03U,`  
`kFLASH_SwapStateComplete = 0x04U,`  
`kFLASH_SwapStateDisabled = 0x05U }`  
*Enumeration for the possible flash swap status.*
- enum `flash_swap_block_status_t` {  
`kFLASH_SwapBlockStatusLowerHalfProgramBlocksAtZero,`  
`kFLASH_SwapBlockStatusUpperHalfProgramBlocksAtZero }`  
*Enumeration for the possible flash swap block status*
- enum `flash_partition_flexram_load_option_t` {  
`kFLASH_PartitionFlexramLoadOptionLoadedWithValidEepromData,`  
`kFLASH_PartitionFlexramLoadOptionNotLoaded = 0x01U }`  
*Enumeration for FlexRAM load during reset option.*

## Flash version

- enum `_flash_driver_version_constants` {  
`kFLASH_DriverVersionName = 'F',`  
`kFLASH_DriverVersionMajor = 2,`  
`kFLASH_DriverVersionMinor = 1,`  
`kFLASH_DriverVersionBugfix = 0 }`  
*FLASH driver version for ROM.*
- #define `MAKE_VERSION`(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))  
*Construct the version number for drivers.*
- #define `FSL_FLASH_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 0))  
*FLASH driver version for SDK.*

## Flash configuration

- #define `FLASH_SSD_CONFIG_ENABLE_FLEXNVM_SUPPORT` 1

## Overview

- Whether to support FlexNVM in flash driver.*
  - #define **FLASH\_SSD\_IS\_FLEXNVM\_ENABLED** (**FLASH\_SSD\_CONFIG\_ENABLE\_FLEXNVM\_SUPPORT** && **FSL\_FEATURE\_FLASH\_HAS\_FLEX\_NVM**)
- Whether the FlexNVM is enabled in flash driver.*
  - #define **FLASH\_DRIVER\_IS\_FLASH\_RESIDENT** 1
- Flash driver location.*
  - #define **FLASH\_DRIVER\_IS\_EXPORTED** 0
- Flash Driver Export option.*

## Flash status

- enum **\_flash\_status** {  
    **kStatus\_FLASH\_Success** = MAKE\_STATUS(kStatusGroupGeneric, 0),  
    **kStatus\_FLASH\_InvalidArgument** = MAKE\_STATUS(kStatusGroupGeneric, 4),  
    **kStatus\_FLASH\_SizeError** = MAKE\_STATUS(kStatusGroupFlashDriver, 0),  
    **kStatus\_FLASH\_AlignmentError**,  
    **kStatus\_FLASH\_AddressError** = MAKE\_STATUS(kStatusGroupFlashDriver, 2),  
    **kStatus\_FLASH\_AccessError**,  
    **kStatus\_FLASH\_ProtectionViolation**,  
    **kStatus\_FLASH\_CommandFailure**,  
    **kStatus\_FLASH\_UnknownProperty** = MAKE\_STATUS(kStatusGroupFlashDriver, 6),  
    **kStatus\_FLASH\_EraseKeyError** = MAKE\_STATUS(kStatusGroupFlashDriver, 7),  
    **kStatus\_FLASH\_RegionExecuteOnly** = MAKE\_STATUS(kStatusGroupFlashDriver, 8),  
    **kStatus\_FLASH\_ExecuteInRamFunctionNotReady**,  
    **kStatus\_FLASH\_PartitionStatusUpdateFailure**,  
    **kStatus\_FLASH\_SetFlexramAsEepromError**,  
    **kStatus\_FLASH\_RecoverFlexramAsRamError**,  
    **kStatus\_FLASH\_SetFlexramAsRamError** = MAKE\_STATUS(kStatusGroupFlashDriver, 13),  
    **kStatus\_FLASH\_RecoverFlexramAsEepromError**,  
    **kStatus\_FLASH\_CommandNotSupported** = MAKE\_STATUS(kStatusGroupFlashDriver, 15),  
    **kStatus\_FLASH\_SwapSystemNotInUninitialized**,  
    **kStatus\_FLASH\_SwapIndicatorAddressError** }  
    *Flash driver status codes.*
  - #define **kStatusGroupGeneric** 0
- Flash driver status group.*
  - #define **kStatusGroupFlashDriver** 1
- #define **MAKE\_STATUS**(group, code) (((group)\*100) + (code)))  
    *Construct a status code value from a group and code number.*

## Flash API key

- enum **\_flash\_driver\_api\_keys** { **kFLASH\_ApiEraseKey** = **FOUR\_CHAR\_CODE**('k', 'f', 'e', 'k') }  
    *Enumeration for flash driver API keys.*
- #define **FOUR\_CHAR\_CODE**(a, b, c, d) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))  
    *Construct the four char code for flash driver API key.*

## Initialization

- status\_t **FLASH\_Init** (flash\_config\_t \*config)  
*Initializes global flash properties structure members.*
- status\_t **FLASH\_SetCallback** (flash\_config\_t \*config, flash\_callback\_t callback)  
*Set the desired flash callback function.*
- status\_t **FLASH\_PrepareExecuteInRamFunctions** (flash\_config\_t \*config)  
*Prepare flash execute-in-RAM functions.*

## Erasing

- status\_t **FLASH\_EraseAll** (flash\_config\_t \*config, uint32\_t key)  
*Erases entire flash.*
- status\_t **FLASH\_Erase** (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key)  
*Erases flash sectors encompassed by parameters passed into function.*
- status\_t **FLASH\_EraseAllExecuteOnlySegments** (flash\_config\_t \*config, uint32\_t key)  
*Erases entire flash, including protected sectors.*

## Programming

- status\_t **FLASH\_Program** (flash\_config\_t \*config, uint32\_t start, uint32\_t \*src, uint32\_t lengthInBytes)  
*Programs flash with data at locations passed in through parameters.*
- status\_t **FLASH\_ProgramOnce** (flash\_config\_t \*config, uint32\_t index, uint32\_t \*src, uint32\_t lengthInBytes)  
*Programs Program Once Field through parameters.*

## Reading

Programs flash with data at locations passed in through parameters via Program Section command

This function programs the flash memory with desired data for a given flash area as determined by the start address and length.

Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>start</i>         | The start address of the desired flash memory to be programmed. Must be word-aligned.        |
| <i>src</i>           | Pointer to the source buffer of data that is to be programmed into the flash.                |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be programmed. Must be word-aligned. |

## Overview

### Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_AlignmentError</i>               | Parameter is not aligned with specified baseline.                       |
| <i>kStatus_FLASH_AddressError</i>                 | Address is out of range.  |
| <i>kStatus_FLASH_SetFlexramAsRamError</i>         | Failed to set flexram as RAM  |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |
| <i>kStatus_FLASH_RecoverFlexramAsEepromError</i>  | Failed to recover flexram as eeprom                                     |

Programs EEPROM with data at locations passed in through parameters

This function programs the Emulated EEPROM with desired data for a given flash area as determined by the start address and length.

### Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>start</i>         | The start address of the desired flash memory to be programmed. Must be word-aligned.        |
| <i>src</i>           | Pointer to the source buffer of data that is to be programmed into the flash.                |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be programmed. Must be word-aligned. |

### Return values



|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                  | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>          | Invalid argument is provided.   |
| <i>kStatus_FLASH_AddressError</i>             | Address is out of range.  |
| <i>kStatus_FLASH_SetFlexramAsEepromError</i>  | Failed to set flexram as eeprom.  |
| <i>kStatus_FLASH_ProtectionViolation</i>      | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_RecoverFlexramAsRamError</i> | Failed to recover flexram as RAM  |

- status\_t **FLASH\_ReadOnce** (flash\_config\_t \*config, uint32\_t index, uint32\_t \*dst, uint32\_t lengthInBytes)

*Read resource with data at locations passed in through parameters.*

## Security

- status\_t **FLASH\_GetSecurityState** (flash\_config\_t \*config, flash\_security\_state\_t \*state)  
*Returns the security state via the pointer passed into the function.*
- status\_t **FLASH\_SecurityBypass** (flash\_config\_t \*config, const uint8\_t \*backdoorKey)  
*Allows user to bypass security with a backdoor key.*

## Verification

- status\_t **FLASH\_VerifyEraseAll** (flash\_config\_t \*config, flash\_margin\_value\_t margin)  
*Verifies erasure of entire flash at specified margin level.*
- status\_t **FLASH\_VerifyErase** (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, flash\_margin\_value\_t margin)  
*Verifies erasure of desired flash area at specified margin level.*
- status\_t **FLASH\_VerifyProgram** (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, const uint32\_t \*expectedData, flash\_margin\_value\_t margin, uint32\_t \*failedAddress, uint32\_t \*failedData)  
*Verifies programming of desired flash area at specified margin level.*
- status\_t **FLASH\_VerifyEraseAllExecuteOnlySegments** (flash\_config\_t \*config, flash\_margin\_value\_t margin)  
*Verifies if the program flash executeonly segments have been erased to the specified read margin level.*

## Protection

- status\_t **FLASH\_IsProtected** (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, flash\_protection\_state\_t \*protection\_state)  
*Returns the protection state of desired flash area via the pointer passed into the function.*
- status\_t **FLASH\_IsExecuteOnly** (flash\_config\_t \*config, uint32\_t start, uint32\_t lengthInBytes, flash\_execute\_only\_access\_state\_t \*access\_state)  
*Returns the access state of desired flash area via the pointer passed into the function.*

## Overview

## Properties

- status\_t [FLASH\\_GetProperty](#) (flash\_config\_t \*config, flash\_property\_tag\_t whichProperty, uint32\_t \*value)  
*Returns the desired flash property.*

## Flash Protection Utilities

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Parameters

|                              |  |
|------------------------------|--|
| <i>config</i>                | Pointer to storage for the driver runtime state.   |
| <i>option</i>                | The option used to set FlexRAM load behavior during reset.   |
| <i>eeepromData-SizeCode</i>  | Determines the amount of FlexRAM used in each of the available EEPROM subsystems.  |
| <i>flexnvm-PartitionCode</i> | Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions. |

Return values

|  |   |
|--|---|
| <a href="#">kStatus_FLASH_Success</a>                      | API was executed successfully.  |
| <a href="#">kStatus_FLASH_InvalidArgument</a>              | Invalid argument is provided.   |
| <a href="#">kStatus_FLASH_ExecuteInRamFunctionNotReady</a> | Execute-in-RAM function is not available.                               |
| <a href="#">kStatus_FLASH_AccessError</a>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <a href="#">kStatus_FLASH_ProtectionViolation</a>          | The program/erase operation is requested to execute on protected areas. |
| <a href="#">kStatus_FLASH_CommandFailure</a>               | Run-time error during command execution.                                |

- status\_t [FLASH\\_PflashSetProtection](#) (flash\_config\_t \*config, uint32\_t protectStatus)  
*Set PFLASH Protection to the intended protection status.*
- status\_t [FLASH\\_PflashGetProtection](#) (flash\_config\_t \*config, uint32\_t \*protectStatus)  
*Get PFLASH Protection Status.*

## 12.2 Data Structure Documentation

### 12.2.1 struct flash\_execute\_in\_ram\_function\_config\_t

#### Data Fields

- uint32\_t [activeFunctionCount](#)  
*Number of available execute-in-RAM functions.*
- uint32\_t \* [flashRunCommand](#)  
*execute-in-RAM function: flash\_run\_command.*
- uint32\_t \* [flashCacheClearCommand](#)  
*execute-in-RAM function: flash\_cache\_clear\_command.*

#### 12.2.1.0.0.17 Field Documentation

12.2.1.0.0.17.1 uint32\_t flash\_execute\_in\_ram\_function\_config\_t::activeFunctionCount

12.2.1.0.0.17.2 uint32\_t\* flash\_execute\_in\_ram\_function\_config\_t::flashRunCommand

12.2.1.0.0.17.3 uint32\_t\* flash\_execute\_in\_ram\_function\_config\_t::flashCacheClearCommand

### 12.2.2 struct flash\_swap\_state\_config\_t

#### Data Fields

- [flash\\_swap\\_state\\_t](#) flashSwapState  
*Current swap system status.*
- [flash\\_swap\\_block\\_status\\_t](#) currentSwapBlockStatus  
*Current swap block status.*
- [flash\\_swap\\_block\\_status\\_t](#) nextSwapBlockStatus  
*Next swap block status.*

#### 12.2.2.0.0.18 Field Documentation

12.2.2.0.0.18.1 flash\_swap\_state\_t flash\_swap\_state\_config\_t::flashSwapState

12.2.2.0.0.18.2 flash\_swap\_block\_status\_t flash\_swap\_state\_config\_t::currentSwapBlockStatus

12.2.2.0.0.18.3 flash\_swap\_block\_status\_t flash\_swap\_state\_config\_t::nextSwapBlockStatus

### 12.2.3 struct flash\_swap\_ifr\_field\_config\_t

#### Data Fields

- uint16\_t [swapIndicatorAddress](#)  
*Swap indicator address field.*
- uint16\_t [swapEnableWord](#)  
*Swap enable word field.*
- uint8\_t [reserved0](#) [4]

## Data Structure Documentation

*Reserved field.*

### 12.2.3.0.0.19 Field Documentation

12.2.3.0.0.19.1 uint16\_t flash\_swap\_ifr\_field\_config\_t::swapIndicatorAddress

12.2.3.0.0.19.2 uint16\_t flash\_swap\_ifr\_field\_config\_t::swapEnableWord

12.2.3.0.0.19.3 uint8\_t flash\_swap\_ifr\_field\_config\_t::reserved0[4]

### 12.2.4 union flash\_swap\_ifr\_field\_data\_t

#### Data Fields

- uint32\_t [flashSwapIfrData](#) [2]  
*Flash Swap IFR field data.*
- [flash\\_swap\\_ifr\\_field\\_config\\_t](#) [flashSwapIfrField](#)  
*Flash Swap IFR field struct.*

### 12.2.4.0.0.20 Field Documentation

12.2.4.0.0.20.1 uint32\_t flash\_swap\_ifr\_field\_data\_t::flashSwapIfrData[2]

12.2.4.0.0.20.2 flash\_swap\_ifr\_field\_config\_t flash\_swap\_ifr\_field\_data\_t::flashSwapIfrField

### 12.2.5 struct flash\_operation\_config\_t

#### Data Fields

- uint32\_t [convertedAddress](#)  
*Converted address for current flash type.*
- uint32\_t [activeSectorSize](#)  
*Sector size of current flash type.*
- uint32\_t [activeBlockSize](#)  
*Block size of current flash type.*
- uint32\_t [blockWriteUnitSize](#)  
*write unit size.*
- uint32\_t [sectorCmdAddressAligment](#)  
*Erase sector command address alignment.*
- uint32\_t [partCmdAddressAligment](#)  
*Program/Verify part command address alignment.*
- 32\_t [resourceCmdAddressAligment](#)  
*Read resource command address alignment.*
- uint32\_t [checkCmdAddressAligment](#)  
*Program check command address alignment.*

### 12.2.5.0.0.21 Field Documentation

12.2.5.0.0.21.1 `uint32_t flash_operation_config_t::convertedAddress`

12.2.5.0.0.21.2 `uint32_t flash_operation_config_t::activeSectorSize`

12.2.5.0.0.21.3 `uint32_t flash_operation_config_t::activeBlockSize`

12.2.5.0.0.21.4 `uint32_t flash_operation_config_t::blockWriteUnitSize`

12.2.5.0.0.21.5 `uint32_t flash_operation_config_t::sectorCmdAddressAligment`

12.2.5.0.0.21.6 `uint32_t flash_operation_config_t::partCmdAddressAligment`

12.2.5.0.0.21.7 `uint32_t flash_operation_config_t::resourceCmdAddressAligment`

12.2.5.0.0.21.8 `uint32_t flash_operation_config_t::checkCmdAddressAligment`

## 12.2.6 struct flash\_config\_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

### Data Fields

- `uint32_t PFlashBlockBase`  
*Base address of the first PFlash block.*
- `uint32_t PFlashTotalSize`  
*Size of all combined PFlash block.*
- `uint32_t PFlashBlockCount`  
*Number of PFlash blocks.*
- `uint32_t PFlashSectorSize`  
*Size in bytes of a sector of PFlash.*
- `flash_callback_t PFlashCallback`  
*Callback function for flash API.*
- `uint32_t PFlashAccessSegmentSize`  
*Size in bytes of a access segment of PFlash.*
- `uint32_t PFlashAccessSegmentCount`  
*Number of PFlash access segments.*
- `uint32_t * flashExecuteInRamFunctionInfo`  
*Info struct of flash execute-in-RAM function.*
- `uint32_t FlexRAMBlockBase`  
*For FlexNVM device, this is the base address of FlexRAM For non-FlexNVM device, this is the base address of acceleration RAM memory.*
- `uint32_t FlexRAMTotalSize`  
*For FlexNVM device, this is the size of FlexRAM For non-FlexNVM device, this is the size of acceleration RAM memory.*
- `uint32_t DFlashBlockBase`  
*For FlexNVM device, this is the base address of D-Flash memory (FlexNVM memory); For non-FlexNVM*

## Macro Definition Documentation

- device, this field is unused.*
  - uint32\_t [DFlashTotalSize](#)  
*For FlexNVM device, this is total size of the FlexNVM memory; For non-FlexNVM device, this field is unused.*
  - uint32\_t [EEpromTotalSize](#)  
*For FlexNVM device, this is the size in byte of EEPROM area which was partitioned from FlexRAM; For non-FlexNVM device, this field is unused.*

### 12.2.6.0.0.22 Field Documentation

12.2.6.0.0.22.1 uint32\_t flash\_config\_t::PFlashTotalSize

12.2.6.0.0.22.2 uint32\_t flash\_config\_t::PFlashBlockCount

12.2.6.0.0.22.3 uint32\_t flash\_config\_t::PFlashSectorSize

12.2.6.0.0.22.4 flash\_callback\_t flash\_config\_t::PFlashCallback

12.2.6.0.0.22.5 uint32\_t flash\_config\_t::PFlashAccessSegmentSize

12.2.6.0.0.22.6 uint32\_t flash\_config\_t::PFlashAccessSegmentCount

12.2.6.0.0.22.7 uint32\_t\* flash\_config\_t::flashExecuteInRamFunctionInfo

## 12.3 Macro Definition Documentation

12.3.1 **#define MAKE\_VERSION( major, minor, bugfix ) (((major) << 16) | ((minor) << 8) | (bugfix))**

12.3.2 **#define FSL\_FLASH\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))**

Version 2.1.0.

12.3.3 **#define FLASH\_SSD\_CONFIG\_ENABLE\_FLEXNVM\_SUPPORT 1**

Enable FlexNVM support by default.

12.3.4 **#define FLASH\_DRIVER\_IS\_FLASH\_RESIDENT 1**

Used for flash resident application.

12.3.5 **#define FLASH\_DRIVER\_IS\_EXPORTED 0**

Used for SDK application.

### 12.3.6 #define kStatusGroupGeneric 0

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

### 12.3.8 #define FOUR\_CHAR\_CODE( a, b, c, d ) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))

## 12.4 Enumeration Type Documentation

### 12.4.1 enum \_flash\_driver\_version\_constants

Enumerator

***kFLASH\_DriverVersionName*** Flash driver version name.  
***kFLASH\_DriverVersionMajor*** Major flash driver version.  
***kFLASH\_DriverVersionMinor*** Minor flash driver version.  
***kFLASH\_DriverVersionBugfix*** Bugfix for flash driver version.

### 12.4.2 enum \_flash\_status

Enumerator

***kStatus\_FLASH\_Success*** API is executed successfully.  
***kStatus\_FLASH\_InvalidArgument*** Invalid argument.  
***kStatus\_FLASH\_SizeError*** Error size.  
***kStatus\_FLASH\_AlignmentError*** Parameter is not aligned with specified baseline.  
***kStatus\_FLASH\_AddressError*** Address is out of range.  
***kStatus\_FLASH\_AccessError*** Invalid instruction codes and out-of bounds addresses.  
***kStatus\_FLASH\_ProtectionViolation*** The program/erase operation is requested to execute on protected areas.  
***kStatus\_FLASH\_CommandFailure*** Run-time error during command execution.  
***kStatus\_FLASH\_UnknownProperty*** Unknown property.  
***kStatus\_FLASH\_EraseKeyError*** API erase key is invalid.  
***kStatus\_FLASH\_RegionExecuteOnly*** Current region is execute only.  
***kStatus\_FLASH\_ExecuteInRamFunctionNotReady*** Execute-in-RAM function is not available.  
***kStatus\_FLASH\_PartitionStatusUpdateFailure*** Failed to update partition status.  
***kStatus\_FLASH\_SetFlexramAsEepromError*** Failed to set flexram as eeprom.  
***kStatus\_FLASH\_RecoverFlexramAsRamError*** Failed to recover flexram as RAM.  
***kStatus\_FLASH\_SetFlexramAsRamError*** Failed to set flexram as RAM.  
***kStatus\_FLASH\_RecoverFlexramAsEepromError*** Failed to recover flexram as eeprom.  
***kStatus\_FLASH\_CommandNotSupported*** Flash API is not supported.  
***kStatus\_FLASH\_SwapSystemNotInUninitialized*** Swap system is not in uninitialized state.  
***kStatus\_FLASH\_SwapIndicatorAddressError*** Swap indicator address is invalid.

## Enumeration Type Documentation

### 12.4.3 enum \_flash\_driver\_api\_keys

#### Note

The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

#### Enumerator

***kFLASH\_ApiEraseKey*** Key value used to validate all flash erase APIs.

### 12.4.4 enum flash\_margin\_value\_t

#### Enumerator

***kFLASH\_MarginValueNormal*** Use the 'normal' read level for 1s.

***kFLASH\_MarginValueUser*** Apply the 'User' margin to the normal read-1 level.

***kFLASH\_MarginValueFactory*** Apply the 'Factory' margin to the normal read-1 level.

***kFLASH\_MarginValueInvalid*** Not real margin level, Used to determine the range of valid margin level.

### 12.4.5 enum flash\_security\_state\_t

#### Enumerator

***kFLASH\_SecurityStateNotSecure*** Flash is not secure.

***kFLASH\_SecurityStateBackdoorEnabled*** Flash backdoor is enabled.

***kFLASH\_SecurityStateBackdoorDisabled*** Flash backdoor is disabled.

### 12.4.6 enum flash\_protection\_state\_t

#### Enumerator

***kFLASH\_ProtectionStateUnprotected*** Flash region is not protected.

***kFLASH\_ProtectionStateProtected*** Flash region is protected.

***kFLASH\_ProtectionStateMixed*** Flash is mixed with protected and unprotected region.

### 12.4.7 enum flash\_execute\_only\_access\_state\_t

#### Enumerator

***kFLASH\_AccessStateUnLimited*** Flash region is unLimited.



***kFLASH\_AccessStateExecuteOnly*** Flash region is execute only.

***kFLASH\_AccessStateMixed*** Flash is mixed with unLimited and execute only region.

#### 12.4.8 enum flash\_property\_tag\_t

Enumerator

***kFLASH\_PropertyPflashSectorSize*** Pflash sector size property.

***kFLASH\_PropertyPflashTotalSize*** Pflash total size property.

***kFLASH\_PropertyPflashBlockSize*** Pflash block size property.

***kFLASH\_PropertyPflashBlockCount*** Pflash block count property.

***kFLASH\_PropertyPflashBlockBaseAddr*** Pflash block base address property.

***kFLASH\_PropertyPflashFacSupport*** Pflash fac support property.

***kFLASH\_PropertyPflashAccessSegmentSize*** Pflash access segment size property.

***kFLASH\_PropertyPflashAccessSegmentCount*** Pflash access segment count property.

***kFLASH\_PropertyFlexRamBlockBaseAddr*** FlexRam block base address property.

***kFLASH\_PropertyFlexRamTotalSize*** FlexRam total size property.

***kFLASH\_PropertyDflashSectorSize*** Dflash sector size property.

***kFLASH\_PropertyDflashTotalSize*** Dflash total size property.

***kFLASH\_PropertyDflashBlockSize*** Dflash block count property.

***kFLASH\_PropertyDflashBlockCount*** Dflash block base address property.

***kFLASH\_PropertyDflashBlockBaseAddr*** Eeprom total size property.

#### 12.4.9 enum \_flash\_execute\_in\_ram\_function\_constants

Enumerator

***kFLASH\_ExecuteInRamFunctionMaxSizeInWords*** Max size of execute-in-RAM function.

***kFLASH\_ExecuteInRamFunctionTotalNum*** Total number of execute-in-RAM functions.

#### 12.4.10 enum flash\_read\_resource\_option\_t

Enumerator

***kFLASH\_ResourceOptionFlashIfr*** Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR.

***kFLASH\_ResourceOptionVersionId*** Select code for Version ID.

### 12.4.11 enum \_flash\_read\_resource\_range

Enumerator

*kFLASH\_ResourceRangePflashIfrSizeInBytes* Pflash IFR size in byte.  
*kFLASH\_ResourceRangeVersionIdSizeInBytes* Version ID IFR size in byte.  
*kFLASH\_ResourceRangeVersionIdStart* Version ID IFR start address.  
*kFLASH\_ResourceRangeVersionIdEnd* Version ID IFR end address.  
*kFLASH\_ResourceRangePflashSwapIfrEnd* Pflash swap IFR end address.  
*kFLASH\_ResourceRangeDflashIfrStart* Dflash IFR start address.  
*kFLASH\_ResourceRangeDflashIfrEnd* Dflash IFR end address.

### 12.4.12 enum flash\_flexram\_function\_option\_t

Enumerator

*kFLASH\_FlexramFunctionOptionAvailableAsRam* Option used to make FlexRAM available as RAM.  
*kFLASH\_FlexramFunctionOptionAvailableForEeprom* Option used to make FlexRAM available for EEPROM.

### 12.4.13 enum flash\_swap\_function\_option\_t

Enumerator

*kFLASH\_SwapFunctionOptionEnable* Option used to enable Swap function.  
*kFLASH\_SwapFunctionOptionDisable* Option used to Disable Swap function.

### 12.4.14 enum flash\_swap\_control\_option\_t

Enumerator

*kFLASH\_SwapControlOptionInitializeSystem* Option used to Initialize Swap System.  
*kFLASH\_SwapControlOptionSetInUpdateState* Option used to Set Swap in Update State.  
*kFLASH\_SwapControlOptionSetInCompleteState* Option used to Set Swap in Complete State.  
*kFLASH\_SwapControlOptionReportStatus* Option used to Report Swap Status.  
*kFLASH\_SwapControlOptionDisableSystem* Option used to Disable Swap Status.

### 12.4.15 enum flash\_swap\_state\_t

Enumerator

***kFLASH\_SwapStateUninitialized*** Flash swap system is in uninitialized state.

***kFLASH\_SwapStateReady*** Flash swap system is in ready state.

***kFLASH\_SwapStateUpdate*** Flash swap system is in update state.

***kFLASH\_SwapStateUpdateErased*** Flash swap system is in updateErased state.

***kFLASH\_SwapStateComplete*** Flash swap system is in complete state.

***kFLASH\_SwapStateDisabled*** Flash swap system is in disabled state.

### 12.4.16 enum flash\_swap\_block\_status\_t

Enumerator

***kFLASH\_SwapBlockStatusLowerHalfProgramBlocksAtZero*** Swap block status is that lower half program block at zero.

***kFLASH\_SwapBlockStatusUpperHalfProgramBlocksAtZero*** Swap block status is that upper half program block at zero.

### 12.4.17 enum flash\_partition\_flexram\_load\_option\_t

Enumerator

***kFLASH\_PartitionFlexramLoadOptionLoadedWithValidEepromData*** FlexRAM is loaded with valid EEPROM data during reset sequence.

***kFLASH\_PartitionFlexramLoadOptionNotLoaded*** FlexRAM is not loaded during reset sequence.

## 12.5 Function Documentation

### 12.5.1 status\_t FLASH\_Init ( flash\_config\_t \* config )

This function checks and initializes Flash module for the other Flash APIs.

Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to storage for the driver runtime state. |
|---------------|--|

## Function Documentation

Return values

|   |   |
|---|---|
| <a href="#"><i>kStatus_FLASH_Success</i></a>                      | API was executed successfully.            |
| <a href="#"><i>kStatus_FLASH_InvalidArgument</i></a>              | Invalid argument is provided.             |
| <a href="#"><i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i></a> | Execute-in-RAM function is not available. |
| <a href="#"><i>kStatus_FLASH_PartitionStatusUpdateFailure</i></a> | Failed to update partition status.        |

### 12.5.2 **status\_t FLASH\_SetCallback ( flash\_config\_t \* *config*, flash\_callback\_t *callback* )**

Parameters

|                 |  |
|-----------------|--|
| <i>config</i>   | Pointer to storage for the driver runtime state. |
| <i>callback</i> | callback function to be stored in driver         |

Return values

|  |                                |
|--|--------------------------------|
| <a href="#"><i>kStatus_FLASH_Success</i></a>         | API was executed successfully. |
| <a href="#"><i>kStatus_FLASH_InvalidArgument</i></a> | Invalid argument is provided.  |

### 12.5.3 **status\_t FLASH\_PrepareExecuteInRamFunctions ( flash\_config\_t \* *config* )**

Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to storage for the driver runtime state. |
|---------------|--|

Return values

|  |                                |
|--|--------------------------------|
| <a href="#"><i>kStatus_FLASH_Success</i></a> | API was executed successfully. |
|--|--------------------------------|

|                                      |                               |
|--------------------------------------|-------------------------------|
| <i>kStatus_FLASH_InvalidArgument</i> | Invalid argument is provided. |
|--------------------------------------|-------------------------------|

#### 12.5.4 status\_t FLASH\_EraseAll ( flash\_config\_t \* config, uint32\_t key )

Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to storage for the driver runtime state. |
| <i>key</i>    | value used to validate all flash erase APIs.     |

Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_EraseKeyError</i>                | API erase key is invalid.   |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |
| <i>kStatus_FLASH_PartitionStatusUpdateFailure</i> | Failed to update partition status                                       |

#### 12.5.5 status\_t FLASH\_Erase ( flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, uint32\_t key )

This function erases the appropriate number of flash sectors based on the desired start address and length.

## Function Documentation

### Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>start</i>         | The start address of the desired flash memory to be erased. The start address does not need to be sector aligned but must be word-aligned. |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be erased. Must be word aligned.   |
| <i>key</i>           | value used to validate all flash erase APIs.   |

### Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_MisalignmentError</i>            | Parameter is not aligned with specified baseline.                       |
| <i>kStatus_FLASH_AddressError</i>                 | Address is out of range.  |
| <i>kStatus_FLASH_EraseKeyError</i>                | API erase key is invalid.   |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |

### 12.5.6 **status\_t FLASH\_EraseAllExecuteOnlySegments ( flash\_config\_t \* config, uint32\_t key )**

#### Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to storage for the driver runtime state. |
| <i>key</i>    | value used to validate all flash erase APIs.     |

## Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_EraseKeyError</i>                | API erase key is invalid.   |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |
| <i>kStatus_FLASH_PartitionStatusUpdateFailure</i> | Failed to update partition status                                       |

Erases all program flash execute-only segments defined by the FXACC registers.

## Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to storage for the driver runtime state. |
| <i>key</i>    | value used to validate all flash erase APIs.     |

## Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.            |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.             |
| <i>kStatus_FLASH_EraseKeyError</i>                | API erase key is invalid.                 |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available. |

## Function Documentation

|  |   |
|--|---|
| <i>kStatus_FLASH_Access-Error</i>        | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH-ProtectionViolation</i> | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH-CommandFailure</i>      | Run-time error during command execution.                                |

### 12.5.7 status\_t FLASH\_Program ( flash\_config\_t \* config, uint32\_t start, uint32\_t \* src, uint32\_t lengthInBytes )

This function programs the flash memory with desired data for a given flash area as determined by the start address and length.

#### Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>start</i>         | The start address of the desired flash memory to be programmed. Must be word-aligned.        |
| <i>src</i>           | Pointer to the source buffer of data that is to be programmed into the flash.                |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be programmed. Must be word-aligned. |

#### Return values

|  |   |
|--|---|
| <i>kStatus_FLASH_Success</i>                       | API was executed successfully.  |
| <i>kStatus_FLASH_Invalid-Argument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH-AlignmentError</i>                | Parameter is not aligned with specified baseline.                       |
| <i>kStatus_FLASH_Address-Error</i>                 | Address is out of range.  |
| <i>kStatus_FLASH_Execute-InRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_Access-Error</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH-ProtectionViolation</i>           | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH-CommandFailure</i>                | Run-time error during command execution.                                |



### 12.5.8 **status\_t FLASH\_ProgramOnce ( flash\_config\_t \* *config*, uint32\_t *index*, uint32\_t \* *src*, uint32\_t *lengthInBytes* )**

This function programs the Program Once Field with desired data for a given flash area as determined by the index and length.

Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>index</i>         | The index indicating which area of Program Once Field to be programmed.                      |
| <i>src</i>           | Pointer to the source buffer of data that is to be programmed into the Program Once Field.   |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be programmed. Must be word-aligned. |

Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH-ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH-CommandFailure</i>               | Run-time error during command execution.                                |

### 12.5.9 **status\_t FLASH\_ReadOnce ( flash\_config\_t \* *config*, uint32\_t *index*, uint32\_t \* *dst*, uint32\_t *lengthInBytes* )**

This function reads the flash memory with desired location for a given flash area as determined by the start address and length.

## Function Documentation

### Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.                                       |
| <i>start</i>         | The start address of the desired flash memory to be programmed. Must be word-aligned.  |
| <i>dst</i>           | Pointer to the destination buffer of data that is used to store data to be read.       |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be read. Must be word-aligned. |
| <i>option</i>        | The resource option which indicates which area should be read back.                    |

### Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_AlignmentError</i>               | Parameter is not aligned with specified baseline.                       |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |

Read Program Once Field through parameters

This function reads the read once feild with given index and length

### Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>index</i>         | The index indicating the area of program once field to be read.                              |
| <i>dst</i>           | Pointer to the destination buffer of data that is used to store data to be read.             |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be programmed. Must be word-aligned. |

Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |

#### 12.5.10 **status\_t FLASH\_GetSecurityState ( flash\_config\_t \* *config*, flash\_security\_state\_t \* *state* )**

This function retrieves the current Flash security status, including the security enabling state and the backdoor key enabling state.

Parameters

|               |   |
|---------------|---|
| <i>config</i> | Pointer to storage for the driver runtime state.                    |
| <i>state</i>  | Pointer to the value returned for the current security status code: |

Return values

|                                      |                                |
|--------------------------------------|--------------------------------|
| <i>kStatus_FLASH_Success</i>         | API was executed successfully. |
| <i>kStatus_FLASH_InvalidArgument</i> | Invalid argument is provided.  |

#### 12.5.11 **status\_t FLASH\_SecurityBypass ( flash\_config\_t \* *config*, const uint8\_t \* *backdoorKey* )**

If the MCU is in secured state, this function will unsecure the MCU by comparing the provided backdoor key with ones in the Flash Configuration Field.

## Function Documentation

### Parameters

|                    |   |
|--------------------|---|
| <i>config</i>      | Pointer to storage for the driver runtime state.        |
| <i>backdoorKey</i> | Pointer to the user buffer containing the backdoor key. |

### Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH-ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH-CommandFailure</i>               | Run-time error during command execution.                                |

### 12.5.12 **status\_t FLASH\_VerifyEraseAll ( flash\_config\_t \* *config*, flash\_margin\_value\_t *margin* )**

This function will check to see if the flash have been erased to the specified read margin level.

### Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to storage for the driver runtime state. |
| <i>margin</i> | Read margin choice                               |

## Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |

### 12.5.13 **status\_t FLASH\_VerifyErase ( flash\_config\_t \* config, uint32\_t start, uint32\_t lengthInBytes, flash\_margin\_value\_t margin )**

This function will check the appropriate number of flash sectors based on the desired start address and length to see if the flash have been erased to the specified read margin level.

## Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>start</i>         | The start address of the desired flash memory to be verified. The start address does not need to be sector aligned but must be word-aligned. |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be verified. Must be word-aligned.   |
| <i>margin</i>        | Read margin choice   |

## Return values

|                                      |                                |
|--------------------------------------|--------------------------------|
| <i>kStatus_FLASH_Success</i>         | API was executed successfully. |
| <i>kStatus_FLASH_InvalidArgument</i> | Invalid argument is provided.  |

## Function Documentation

|   |   |
|---|---|
| <i>kStatus_FLASH_AlignmentError</i>               | Parameter is not aligned with specified baseline.                       |
| <i>kStatus_FLASH_AddressError</i>                 | Address is out of range.  |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |

**12.5.14** `status_t FLASH_VerifyProgram ( flash_config_t * config, uint32_t start, uint32_t lengthInBytes, const uint32_t * expectedData, flash_margin_value_t margin, uint32_t * failedAddress, uint32_t * failedData )`

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it with expected data for a given flash area as determined by the start address and length.

Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>start</i>         | The start address of the desired flash memory to be verified. Must be word-aligned.  |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be verified. Must be word-aligned.   |
| <i>expectedData</i>  | Pointer to the expected data that is to be verified against.   |
| <i>margin</i>        | Read margin choice   |
| <i>failedAddress</i> | Pointer to returned failing address.   |
| <i>failedData</i>    | Pointer to returned failing data. Some derivatives do not included failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure. |

## Return values

|   |   |
|---|---|
| <i>kStatus_FLASH_Success</i>                      | API was executed successfully.  |
| <i>kStatus_FLASH_InvalidArgument</i>              | Invalid argument is provided.   |
| <i>kStatus_FLASH_AlignmentError</i>               | Parameter is not aligned with specified baseline.                       |
| <i>kStatus_FLASH_AddressError</i>                 | Address is out of range.  |
| <i>kStatus_FLASH_ExecuteInRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_AccessError</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH_ProtectionViolation</i>          | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH_CommandFailure</i>               | Run-time error during command execution.                                |

### 12.5.15 **status\_t FLASH\_VerifyEraseAllExecuteOnlySegments ( flash\_config\_t \* config, flash\_margin\_value\_t margin )**

## Parameters

|               |  |
|---------------|--|
| <i>config</i> | Pointer to storage for the driver runtime state. |
| <i>margin</i> | Read margin choice                               |

## Return values

|                                      |                                |
|--------------------------------------|--------------------------------|
| <i>kStatus_FLASH_Success</i>         | API was executed successfully. |
| <i>kStatus_FLASH_InvalidArgument</i> | Invalid argument is provided.  |

## Function Documentation

|  |   |
|--|---|
| <i>kStatus_FLASH_Execute-InRamFunctionNotReady</i> | Execute-in-RAM function is not available.                               |
| <i>kStatus_FLASH_Access-Error</i>                  | Invalid instruction codes and out-of bounds addresses.                  |
| <i>kStatus_FLASH-ProtectionViolation</i>           | The program/erase operation is requested to execute on protected areas. |
| <i>kStatus_FLASH-CommandFailure</i>                | Run-time error during command execution.                                |

### 12.5.16 **status\_t FLASH\_IsProtected ( flash\_config\_t \* *config*, uint32\_t *start*, uint32\_t *lengthInBytes*, flash\_protection\_state\_t \* *protection\_state* )**

This function retrieves the current Flash protect status for a given flash area as determined by the start address and length.

#### Parameters

|                         |  |
|-------------------------|--|
| <i>config</i>           | Pointer to storage for the driver runtime state.   |
| <i>start</i>            | The start address of the desired flash memory to be checked. Must be word-aligned.               |
| <i>lengthInBytes</i>    | The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.        |
| <i>protection_state</i> | Pointer to the value returned for the current protection status code for the desired flash area. |

#### Return values

|                                       |   |
|---------------------------------------|---|
| <i>kStatus_FLASH_Success</i>          | API was executed successfully.                    |
| <i>kStatus_FLASH_Invalid-Argument</i> | Invalid argument is provided.                     |
| <i>kStatus_FLASH-AlignmentError</i>   | Parameter is not aligned with specified baseline. |
| <i>kStatus_FLASH_Address-Error</i>    | Address is out of range.                          |



### 12.5.17 **status\_t FLASH\_IsExecuteOnly ( flash\_config\_t \* *config*, uint32\_t *start*, uint32\_t *lengthInBytes*, flash\_execute\_only\_access\_state\_t \* *access\_state* )**

This function retrieves the current Flash access status for a given flash area as determined by the start address and length.

Parameters

|                      |  |
|----------------------|--|
| <i>config</i>        | Pointer to storage for the driver runtime state.   |
| <i>start</i>         | The start address of the desired flash memory to be checked. Must be word-aligned.           |
| <i>lengthInBytes</i> | The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.    |
| <i>access_state</i>  | Pointer to the value returned for the current access status code for the desired flash area. |

Return values

|                                       |   |
|---------------------------------------|---|
| <i>kStatus_FLASH_Success</i>          | API was executed successfully.                    |
| <i>kStatus_FLASH_Invalid-Argument</i> | Invalid argument is provided.                     |
| <i>kStatus_FLASH_-AlignmentError</i>  | Parameter is not aligned with specified baseline. |
| <i>kStatus_FLASH_Address-Error</i>    | Address is out of range.                          |

### 12.5.18 **status\_t FLASH\_GetProperty ( flash\_config\_t \* *config*, flash\_property\_tag\_t *whichProperty*, uint32\_t \* *value* )**

Parameters

|                      |   |
|----------------------|---|
| <i>config</i>        | Pointer to storage for the driver runtime state.                              |
| <i>whichProperty</i> | The desired property from the list of properties in enum flash_property_tag_t |

## Function Documentation

|              |  |
|--------------|--|
| <i>value</i> | Pointer to the value returned for the desired flash property |
|--------------|--|

Return values

|  |                                |
|--|--------------------------------|
| <a href="#"><i>kStatus_FLASH_Success</i></a>         | API was executed successfully. |
| <a href="#"><i>kStatus_FLASH_InvalidArgument</i></a> | Invalid argument is provided.  |
| <a href="#"><i>kStatus_FLASH_UnknownProperty</i></a> | unknown property tag           |

### 12.5.19 **status\_t FLASH\_PflashSetProtection ( flash\_config\_t \* *config*, uint32\_t *protectStatus* )**

Parameters

|                      |   |
|----------------------|---|
| <i>config</i>        | Pointer to storage for the driver runtime state.  |
| <i>protectStatus</i> | The expected protect status user wants to set to PFlash protection register. Each bit is corresponding to protection of 1/32 of the total PFlash. The least significant bit is corresponding to the lowest address area of P-Flash. The most significant bit is corresponding to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected. |

Return values

|  |  |
|--|--|
| <a href="#"><i>kStatus_FLASH_Success</i></a>         | API was executed successfully.           |
| <a href="#"><i>kStatus_FLASH_InvalidArgument</i></a> | Invalid argument is provided.            |
| <a href="#"><i>kStatus_FLASH_CommandFailure</i></a>  | Run-time error during command execution. |

### 12.5.20 **status\_t FLASH\_PflashGetProtection ( flash\_config\_t \* *config*, uint32\_t \* *protectStatus* )**

#### Parameters

|                      |   |
|----------------------|---|
| <i>config</i>        | Pointer to storage for the driver runtime state.  |
| <i>protectStatus</i> | Protect status returned by PFlash IP. Each bit is corresponding to protection of 1/32 of the total PFlash. The least significant bit is corresponding to the lowest address area of PFlash. The most significant bit is corresponding to the highest address area of PFlash. There are two possible cases as below: 0: this area is protected. 1: this area is unprotected. |

#### Return values

|                                       |                                |
|---------------------------------------|--------------------------------|
| <i>kStatus_FLASH_Success</i>          | API was executed successfully. |
| <i>kStatus_FLASH_Invalid-Argument</i> | Invalid argument is provided.  |



## Chapter 13

# GPIO: General-Purpose Input/Output Driver

### 13.1 Overview

#### Modules

- [FGPIO Driver](#)
- [GPIO Driver](#)

#### Data Structures

- struct [gpio\\_pin\\_config\\_t](#)  
*The GPIO pin configuration structure. [More...](#)*

#### Enumerations

- enum [gpio\\_pin\\_direction\\_t](#) {  
    [kGPIO\\_DigitalInput](#) = 0U,  
    [kGPIO\\_DigitalOutput](#) = 1U }  
*GPIO direction definition.*

#### Driver version

- #define [FSL\\_GPIO\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 0))  
*GPIO driver version 2.1.0.*

### 13.2 Data Structure Documentation

#### 13.2.1 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 Note : In some use cases, the corresponding port property should be configured in advance with the [PORT\\_SetPinConfig\(\)](#)

#### Data Fields

- [gpio\\_pin\\_direction\\_t](#) [pinDirection](#)  
*GPIO direction, input or output.*
- uint8\_t [outputLogic](#)  
*Set default output logic, no use in input.*

## Enumeration Type Documentation

### 13.3 Macro Definition Documentation

#### 13.3.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

### 13.4 Enumeration Type Documentation

#### 13.4.1 enum gpio\_pin\_direction\_t

Enumerator

***kGPIO\_DigitalInput*** Set current pin as digital input.

***kGPIO\_DigitalOutput*** Set current pin as digital output.

## 13.5 GPIO Driver

### 13.5.1 Overview

The KSDK provides a peripheral driver for the General-Purpose Input/Output (GPIO) module of Kinetis devices.

### 13.5.2 Typical use case

#### 13.5.2.1 Output Operation

```
/* Output pin configuration */
gpio_pin_config_t led_config =
{
    kGpioDigitalOutput,
    1,
};
/* Sets the configuration */
GPIO_PinInit(GPIO_LED, LED_PINNUM, &led_config);
```

#### 13.5.2.2 Input Operation

```
/* Input pin configuration */
PORT_SetPinInterruptConfig(BOARD_SW2_PORT, BOARD_SW2_GPIO_PIN,
    kPORT_InterruptFallingEdge);
NVIC_EnableIRQ(BOARD_SW2_IRQ);
gpio_pin_config_t sw1_config =
{
    kGpioDigitalInput,
    0,
};
/* Sets the input pin configuration */
GPIO_PinInit(GPIO_SW1, SW1_PINNUM, &sw1_config);
```

## GPIO Configuration

- void [GPIO\\_PinInit](#) (GPIO\_Type \*base, uint32\_t pin, const [gpio\\_pin\\_config\\_t](#) \*config)  
*Initializes a GPIO pin used by the board.*

## GPIO Output Operations

- static void [GPIO\\_WritePinOutput](#) (GPIO\_Type \*base, uint32\_t pin, uint8\_t output)  
*Sets the output level of the multiple GPIO pins to the logic 1 or 0.*
- static void [GPIO\\_SetPinsOutput](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_ClearPinsOutput](#) (GPIO\_Type \*base, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_TogglePinsOutput](#) (GPIO\_Type \*base, uint32\_t mask)  
*Reverses current output logic of the multiple GPIO pins.*

## GPIO Driver

### GPIO Input Operations

- static uint32\_t [GPIO\\_ReadPinInput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the whole GPIO port.*

### GPIO Interrupt

- uint32\_t [GPIO\\_GetPinsInterruptFlags](#) (GPIO\_Type \*base)  
*Reads whole GPIO port interrupt status flag.*
- void [GPIO\\_ClearPinsInterruptFlags](#) (GPIO\_Type \*base, uint32\_t mask)  
*Clears multiple GPIO pin interrupt status flag.*

### 13.5.3 Function Documentation

#### 13.5.3.1 void GPIO\_PinInit ( GPIO\_Type \* *base*, uint32\_t *pin*, const gpio\_pin\_config\_t \* *config* )

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the [GPIO\\_PinInit\(\)](#) function.

This is an example to define an input pin or output pin configuration:

```
* // Define a digital input pin configuration,  
* gpio_pin_config_t config =  
* {  
*     kGPIO_DigitalInput,  
*     0,  
* }  
* //Define a digital output pin configuration,  
* gpio_pin_config_t config =  
* {  
*     kGPIO_DigitalOutput,  
*     0,  
* }  
*
```

#### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.) |
| <i>pin</i>    | GPIO port pin number  |
| <i>config</i> | GPIO pin configuration pointer                                |

#### 13.5.3.2 static void GPIO\_WritePinOutput ( GPIO\_Type \* *base*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]



## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.)  |
| <i>pin</i>    | GPIO pin number  |
| <i>output</i> | GPIO pin output logic level. <ul style="list-style-type: none"> <li>• 0: corresponding pin output low-logic level.</li> <li>• 1: corresponding pin output high-logic level.</li> </ul> |

### 13.5.3.3 static void GPIO\_SetPinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |   |
|-------------|---|
| <i>base</i> | GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro   |

### 13.5.3.4 static void GPIO\_ClearPinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |   |
|-------------|---|
| <i>base</i> | GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro   |

### 13.5.3.5 static void GPIO\_TogglePinsOutput ( GPIO\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |   |
|-------------|---|
| <i>base</i> | GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro   |

### 13.5.3.6 static uint32\_t GPIO\_ReadPinInput ( GPIO\_Type \* *base*, uint32\_t *pin* ) [inline], [static]

## GPIO Driver

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.) |
| <i>pin</i>  | GPIO pin number   |

### Return values

|             |   |
|-------------|---|
| <i>GPIO</i> | port input value <ul style="list-style-type: none"><li>• 0: corresponding pin input low-logic level.</li><li>• 1: corresponding pin input high-logic level.</li></ul> |
|-------------|---|

### 13.5.3.7 uint32\_t GPIO\_GetPinsInterruptFlags ( GPIO\_Type \* *base* )

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.) |
|-------------|---|

### Return values

|                |   |
|----------------|---|
| <i>Current</i> | GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt. |
|----------------|---|

### 13.5.3.8 void GPIO\_ClearPinsInterruptFlags ( GPIO\_Type \* *base*, uint32\_t *mask* )

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | GPIO peripheral base pointer(GPIOA, GPIOB, GPIOC, and so on.) |
| <i>mask</i> | GPIO pin number macro   |

## 13.6 FGPIO Driver

This chapter describes the programming interface of the FGPIO driver. The FGPIO driver configures the FGPIO module and provides a functional interface to build the GPIO application.

### Note

FGPIO (Fast GPIO) is only available in a few MCUs. FGPIO and GPIO share the same peripheral but use different registers. FGPIO is closer to the core than the regular GPIO and it's faster to read and write.

### 13.6.1 Typical use case

#### 13.6.1.1 Output Operation

```
/* Output pin configuration */
gpio_pin_config_t led_config =
{
    kGpioDigitalOutput,
    1,
};
/* Sets the configuration */
FGPIO_PinInit(FGPIO_LED, LED_PINNUM, &led_config);
```

#### 13.6.1.2 Input Operation

```
/* Input pin configuration */
PORT_SetPinInterruptConfig(BOARD_SW2_PORT, BOARD_SW2_FGPIO_PIN,
    kPORT_InterruptFallingEdge);
NVIC_EnableIRQ(BOARD_SW2_IRQ);
gpio_pin_config_t sw1_config =
{
    kGpioDigitalInput,
    0,
};
/* Sets the input pin configuration */
FGPIO_PinInit(FGPIO_SW1, SW1_PINNUM, &sw1_config);
```





## Chapter 14

### I2C: Inter-Integrated Circuit Driver

#### 14.1 Overview

##### Modules

- [I2C DMA Driver](#)
- [I2C Driver](#)
- [I2C FreeRTOS Driver](#)
- [I2C eDMA Driver](#)
- [I2C  \$\mu\$ COS/II Driver](#)
- [I2C  \$\mu\$ COS/III Driver](#)

### 14.2 I2C Driver

#### 14.2.1 Overview

The KSDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of Kinetis 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.

#### 14.2.2 Typical use case

##### 14.2.2.1 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);

if(result)
{
    /* If error occurs, send STOP. */
}
```

```

    I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
    return result;
}

while(!(I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR) & kI2C_IntPendingFlag))
{

}

/* Wait all data sent out, send STOP. */
I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);

```

### 14.2.2.2 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;

```

### 14.2.2.3 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;

```

## I2C Driver

```
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;

DMAMGR_RequestChannel((dma_request_source_t)DMA_REQUEST_SRC, 0, &dmaHandle);

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;
```

### 14.2.2.4 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;

```

#### 14.2.2.5 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;

```

### Data Structures

- struct [i2c\\_master\\_config\\_t](#)  
*I2C master user configuration. [More...](#)*
- struct [i2c\\_slave\\_config\\_t](#)  
*I2C slave user configuration. [More...](#)*
- struct [i2c\\_master\\_transfer\\_t](#)  
*I2C master transfer structure. [More...](#)*
- struct [i2c\\_master\\_handle\\_t](#)  
*I2C master handle structure. [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\\_master\\_transfer\\_callback\\_t](#) )(I2C\_Type \*base, i2c\_master\_handle\_t \*handle, status\_t status, void \*userData)  
*I2C master transfer callback typedef.*
- typedef void(\* [i2c\\_slave\\_transfer\\_callback\\_t](#) )(I2C\_Type \*base, [i2c\\_slave\\_transfer\\_t](#) \*xfer, void \*userData)  
*I2C slave transfer callback typedef.*

### Enumerations

- enum [\\_i2c\\_status](#) {  
    [kStatus\\_I2C\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_I2C, 0),  
    [kStatus\\_I2C\\_Idle](#) = MAKE\_STATUS(kStatusGroup\_I2C, 1),  
    [kStatus\\_I2C\\_Nak](#) = MAKE\_STATUS(kStatusGroup\_I2C, 2),  
    [kStatus\\_I2C\\_ArbitrationLost](#) = MAKE\_STATUS(kStatusGroup\_I2C, 3),  
    [kStatus\\_I2C\\_Timeout](#) = MAKE\_STATUS(kStatusGroup\_I2C, 4) }  
*I2C status return codes.*
- enum [\\_i2c\\_flags](#) {  
    [kI2C\\_ReceiveNakFlag](#) = I2C\_S\_RXAK\_MASK,  
    [kI2C\\_IntPendingFlag](#) = I2C\_S\_IICIF\_MASK,  
    [kI2C\\_TransferDirectionFlag](#) = I2C\_S\_SRW\_MASK,  
    [kI2C\\_RangeAddressMatchFlag](#) = I2C\_S\_RAM\_MASK,  
    [kI2C\\_ArbitrationLostFlag](#) = I2C\_S\_ARBL\_MASK,  
    [kI2C\\_BusBusyFlag](#) = I2C\_S\_BUSY\_MASK,  
    [kI2C\\_AddressMatchFlag](#) = I2C\_S\_IAAS\_MASK,  
    [kI2C\\_TransferCompleteFlag](#) = I2C\_S\_TCF\_MASK }  
*I2C peripheral flags.*
- enum [\\_i2c\\_interrupt\\_enable](#) { [kI2C\\_GlobalInterruptEnable](#) = I2C\_C1\_IICIE\_MASK }  
*I2C feature interrupt source.*

- enum `i2c_direction_t` {  
`kI2C_Write` = 0x0U,  
`kI2C_Read` = 0x1U }  
*Direction of master and slave transfers.*
- enum `i2c_slave_address_mode_t` {  
`kI2C_Address7bit` = 0x0U,  
`kI2C_RangeMatch` = 0x2U }  
*Addressing mode.*
- enum `_i2c_master_transfer_flags` {  
`kI2C_TransferDefaultFlag` = 0x0U,  
`kI2C_TransferNoStartFlag` = 0x1U,  
`kI2C_TransferRepeatedStartFlag` = 0x2U,  
`kI2C_TransferNoStopFlag` = 0x4U }  
*I2C transfer control flag.*
- enum `i2c_slave_transfer_event_t` {  
`kI2C_SlaveAddressMatchEvent` = 0x01U,  
`kI2C_SlaveTransmitEvent` = 0x02U,  
`kI2C_SlaveReceiveEvent` = 0x04U,  
`kI2C_SlaveTransmitAckEvent` = 0x08U,  
`kI2C_SlaveCompletionEvent` = 0x20U,  
`kI2C_SlaveAllEvents` }  
*Set of events sent to the callback for nonblocking slave transfers.*

## Driver version

- #define `FSL_I2C_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 1))  
*I2C driver version 2.0.1.*

## Initialization and deinitialization

- void `I2C_MasterInit` (`I2C_Type` \*base, const `i2c_master_config_t` \*masterConfig, uint32\_t src-Clock\_Hz)  
*Initializes the I2C peripheral.*
- void `I2C_SlaveInit` (`I2C_Type` \*base, const `i2c_slave_config_t` \*slaveConfig)  
*Initializes the I2C peripheral.*
- void `I2C_MasterDeinit` (`I2C_Type` \*base)  
*De-initializes the I2C master peripheral.*
- void `I2C_SlaveDeinit` (`I2C_Type` \*base)  
*De-initializes the I2C slave peripheral.*
- void `I2C_MasterGetDefaultConfig` (`i2c_master_config_t` \*masterConfig)  
*Sets the I2C master configuration structure to default values.*
- void `I2C_SlaveGetDefaultConfig` (`i2c_slave_config_t` \*slaveConfig)  
*Sets the I2C slave configuration structure to default values.*
- static void `I2C_Enable` (`I2C_Type` \*base, bool enable)  
*Enables or disables the I2C peripheral operation.*

## I2C Driver

### Status

- `uint32_t I2C_MasterGetStatusFlags (I2C_Type *base)`  
*Gets the I2C status flags.*
- `static uint32_t I2C_SlaveGetStatusFlags (I2C_Type *base)`  
*Gets the I2C status flags.*
- `static void I2C_MasterClearStatusFlags (I2C_Type *base, uint32_t statusMask)`  
*Clears the I2C status flag state.*
- `static void I2C_SlaveClearStatusFlags (I2C_Type *base, uint32_t statusMask)`  
*Clears the I2C status flag state.*

### Interrupts

- `void I2C_EnableInterrupts (I2C_Type *base, uint32_t mask)`  
*Enables I2C interrupt requests.*
- `void I2C_DisableInterrupts (I2C_Type *base, uint32_t mask)`  
*Disables I2C interrupt requests.*

### DMA Control

- `static uint32_t I2C_GetDataRegAddr (I2C_Type *base)`  
*Gets the I2C tx/rx data register address.*

### Bus Operations

- `void I2C_MasterSetBaudRate (I2C_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)`  
*Sets the I2C master transfer baud rate.*
- `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.*
- `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 uint8_t *txBuff, size_t txSize)`  
*Performs a polling send transaction on the I2C bus without a STOP signal.*
- `status_t I2C_MasterReadBlocking (I2C_Type *base, uint8_t *rxBuff, size_t rxSize)`  
*Performs a polling receive transaction on the I2C bus with a STOP signal.*
- `status_t I2C_SlaveWriteBlocking (I2C_Type *base, const uint8_t *txBuff, size_t txSize)`  
*Performs a polling send transaction on the I2C bus.*
- `void I2C_SlaveReadBlocking (I2C_Type *base, uint8_t *rxBuff, size_t rxSize)`  
*Performs a polling receive transaction 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.*

## Transactional

- void [I2C\\_MasterTransferCreateHandle](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, i2c\_master\_transfer\_callback\_t callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- status\_t [I2C\\_MasterTransferNonBlocking](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, i2c\_master\_transfer\_t \*xfer)  
*Performs a master interrupt non-blocking transfer on the I2C bus.*
- status\_t [I2C\\_MasterTransferGetCount](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the master transfer status during a interrupt non-blocking transfer.*
- void [I2C\\_MasterTransferAbort](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle)  
*Aborts an interrupt non-blocking transfer early.*
- void [I2C\\_MasterTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
*Master interrupt handler.*
- void [I2C\\_SlaveTransferCreateHandle](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, i2c\_slave\_transfer\_callback\_t callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- status\_t [I2C\\_SlaveTransferNonBlocking](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- void [I2C\\_SlaveTransferAbort](#) (I2C\_Type \*base, i2c\_slave\_handle\_t \*handle)  
*Aborts the slave transfer.*
- 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.*
- void [I2C\\_SlaveTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
*Slave interrupt handler.*

## 14.2.3 Data Structure Documentation

### 14.2.3.1 struct i2c\_master\_config\_t

#### Data Fields

- bool [enableMaster](#)  
*Enables the I2C peripheral at initialization time.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud rate configuration of I2C peripheral.*
- uint8\_t [glitchFilterWidth](#)  
*Controls the width of the glitch.*

## I2C Driver

### 14.2.3.1.0.23 Field Documentation

14.2.3.1.0.23.1 bool i2c\_master\_config\_t::enableMaster

14.2.3.1.0.23.2 uint32\_t i2c\_master\_config\_t::baudRate\_Bps

14.2.3.1.0.23.3 uint8\_t i2c\_master\_config\_t::glitchFilterWidth

### 14.2.3.2 struct i2c\_slave\_config\_t

#### Data Fields

- bool [enableSlave](#)  
*Enables the I2C peripheral at initialization time.*
- bool [enableGeneralCall](#)  
*Enable general call addressing mode.*
- bool [enableWakeUp](#)  
*Enables/disables waking up MCU from low-power mode.*
- bool [enableBaudRateCtl](#)  
*Enables/disables independent slave baud rate on SCL in very fast I2C modes.*
- uint16\_t [slaveAddress](#)  
*Slave address configuration.*
- uint16\_t [upperAddress](#)  
*Maximum boundary slave address used in range matching mode.*
- [i2c\\_slave\\_address\\_mode\\_t](#) [addressingMode](#)  
*Addressing mode configuration of i2c\_slave\_address\_mode\_config\_t.*

### 14.2.3.2.0.24 Field Documentation

14.2.3.2.0.24.1 bool i2c\_slave\_config\_t::enableSlave

14.2.3.2.0.24.2 bool i2c\_slave\_config\_t::enableGeneralCall

14.2.3.2.0.24.3 bool i2c\_slave\_config\_t::enableWakeUp

14.2.3.2.0.24.4 bool i2c\_slave\_config\_t::enableBaudRateCtl

14.2.3.2.0.24.5 uint16\_t i2c\_slave\_config\_t::slaveAddress

14.2.3.2.0.24.6 uint16\_t i2c\_slave\_config\_t::upperAddress

14.2.3.2.0.24.7 [i2c\\_slave\\_address\\_mode\\_t](#) i2c\_slave\_config\_t::addressingMode

### 14.2.3.3 struct i2c\_master\_transfer\_t

#### Data Fields

- uint32\_t [flags](#)  
*Transfer flag which controls the transfer.*
- uint8\_t [slaveAddress](#)  
*7-bit slave address.*

- [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.*

#### 14.2.3.3.0.25 Field Documentation

**14.2.3.3.0.25.1 [uint32\\_t i2c\\_master\\_transfer\\_t::flags](#)**

**14.2.3.3.0.25.2 [uint8\\_t i2c\\_master\\_transfer\\_t::slaveAddress](#)**

**14.2.3.3.0.25.3 [i2c\\_direction\\_t i2c\\_master\\_transfer\\_t::direction](#)**

**14.2.3.3.0.25.4 [uint32\\_t i2c\\_master\\_transfer\\_t::subaddress](#)**

Transferred MSB first.

**14.2.3.3.0.25.5 [uint8\\_t i2c\\_master\\_transfer\\_t::subaddressSize](#)**

**14.2.3.3.0.25.6 [uint8\\_t\\* volatile i2c\\_master\\_transfer\\_t::data](#)**

**14.2.3.3.0.25.7 [volatile size\\_t i2c\\_master\\_transfer\\_t::dataSize](#)**

#### 14.2.3.4 [struct \\_i2c\\_master\\_handle](#)

I2C master handle typedef.

#### Data Fields

- [i2c\\_master\\_transfer\\_t transfer](#)  
*I2C master transfer copy.*
- [size\\_t transferSize](#)  
*Total bytes to be transferred.*
- [uint8\\_t state](#)  
*Transfer state maintained during transfer.*
- [i2c\\_master\\_transfer\\_callback\\_t completionCallback](#)  
*Callback function called when transfer finished.*
- [void \\* userData](#)  
*Callback parameter passed to callback function.*

## I2C Driver

### 14.2.3.4.0.26 Field Documentation

14.2.3.4.0.26.1 `i2c_master_transfer_t i2c_master_handle_t::transfer`

14.2.3.4.0.26.2 `size_t i2c_master_handle_t::transferSize`

14.2.3.4.0.26.3 `uint8_t i2c_master_handle_t::state`

14.2.3.4.0.26.4 `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`

14.2.3.4.0.26.5 `void* i2c_master_handle_t::userData`

### 14.2.3.5 struct `i2c_slave_transfer_t`

#### Data Fields

- `i2c_slave_transfer_event_t event`  
*Reason the callback is being invoked.*
- `uint8_t *volatile data`  
*Transfer buffer.*
- `volatile size_t dataSize`  
*Transfer size.*
- `status_t completionStatus`  
*Success or error code describing how the transfer completed.*
- `size_t transferredCount`  
*Number of bytes actually transferred since start or last repeated start.*

### 14.2.3.5.0.27 Field Documentation

14.2.3.5.0.27.1 `i2c_slave_transfer_event_t i2c_slave_transfer_t::event`

14.2.3.5.0.27.2 `uint8_t* volatile i2c_slave_transfer_t::data`

14.2.3.5.0.27.3 `volatile size_t i2c_slave_transfer_t::dataSize`

14.2.3.5.0.27.4 `status_t i2c_slave_transfer_t::completionStatus`

Only applies for `kI2C_SlaveCompletionEvent`.

14.2.3.5.0.27.5 `size_t i2c_slave_transfer_t::transferredCount`

### 14.2.3.6 struct `_i2c_slave_handle`

I2C slave handle typedef.

#### Data Fields

- `bool isBusy`  
*Whether transfer is busy.*
- `i2c_slave_transfer_t transfer`



- *I2C slave transfer copy.*  
uint32\_t **eventMask**  
*Mask of enabled events.*
- **i2c\_slave\_transfer\_callback\_t** callback  
*Callback function called at transfer event.*
- void \* **userData**  
*Callback parameter passed to callback.*

#### 14.2.3.6.0.28 Field Documentation

14.2.3.6.0.28.1 bool i2c\_slave\_handle\_t::isBusy

14.2.3.6.0.28.2 i2c\_slave\_transfer\_t i2c\_slave\_handle\_t::transfer

14.2.3.6.0.28.3 uint32\_t i2c\_slave\_handle\_t::eventMask

14.2.3.6.0.28.4 i2c\_slave\_transfer\_callback\_t i2c\_slave\_handle\_t::callback

14.2.3.6.0.28.5 void\* i2c\_slave\_handle\_t::userData

#### 14.2.4 Macro Definition Documentation

14.2.4.1 #define FSL\_I2C\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

#### 14.2.5 Typedef Documentation

14.2.5.1 typedef void(\* i2c\_master\_transfer\_callback\_t)(I2C\_Type \*base,  
i2c\_master\_handle\_t \*handle, status\_t status, void \*userData)

14.2.5.2 typedef void(\* i2c\_slave\_transfer\_callback\_t)(I2C\_Type \*base,  
i2c\_slave\_transfer\_t \*xfer, void \*userData)

#### 14.2.6 Enumeration Type Documentation

14.2.6.1 enum \_i2c\_status

Enumerator

**kStatus\_I2C\_Busy** I2C is busy with current transfer.  
**kStatus\_I2C\_Idle** Bus is Idle.  
**kStatus\_I2C\_Nak** NAK received during transfer.  
**kStatus\_I2C\_ArbitrationLost** Arbitration lost during transfer.  
**kStatus\_I2C\_Timeout** Wait event timeout.

## I2C Driver

### 14.2.6.2 enum \_i2c\_flags

The following status register flags can be cleared:

- [kI2C\\_ArbitrationLostFlag](#)
- [kI2C\\_IntPendingFlag](#)
- #kI2C\_StartDetectFlag
- #kI2C\_StopDetectFlag

Note

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

Enumerator

***kI2C\_ReceiveNakFlag*** I2C receive NAK flag.  
***kI2C\_IntPendingFlag*** I2C interrupt pending flag.  
***kI2C\_TransferDirectionFlag*** I2C transfer direction flag.  
***kI2C\_RangeAddressMatchFlag*** I2C range address match flag.  
***kI2C\_ArbitrationLostFlag*** I2C arbitration lost flag.  
***kI2C\_BusBusyFlag*** I2C bus busy flag.  
***kI2C\_AddressMatchFlag*** I2C address match flag.  
***kI2C\_TransferCompleteFlag*** I2C transfer complete flag.

### 14.2.6.3 enum \_i2c\_interrupt\_enable

Enumerator

***kI2C\_GlobalInterruptEnable*** I2C global interrupt.

### 14.2.6.4 enum i2c\_direction\_t

Enumerator

***kI2C\_Write*** Master transmit to slave.  
***kI2C\_Read*** Master receive from slave.

### 14.2.6.5 enum i2c\_slave\_address\_mode\_t

Enumerator

***kI2C\_Address7bit*** 7-bit addressing mode.  
***kI2C\_RangeMatch*** Range address match addressing mode.

#### 14.2.6.6 enum \_i2c\_master\_transfer\_flags

Enumerator

***kI2C\_TransferDefaultFlag*** Transfer starts with a start signal, stops with a stop signal.

***kI2C\_TransferNoStartFlag*** Transfer starts without a start signal.

***kI2C\_TransferRepeatedStartFlag*** Transfer starts with a repeated start signal.

***kI2C\_TransferNoStopFlag*** Transfer ends without a stop signal.

#### 14.2.6.7 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\_SlaveTransmitAckEvent*** Callback needs to either transmit an ACK or NACK.

***kI2C\_SlaveCompletionEvent*** A stop was detected or finished transfer, completing the transfer.

***kI2C\_SlaveAllEvents*** Bit mask of all available events.

### 14.2.7 Function Documentation

#### 14.2.7.1 void I2C\_MasterInit ( I2C\_Type \* *base*, const i2c\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )

Call this API to ungate the I2C clock and configure the I2C with master configuration.

Note

This API should be called at the beginning of the application to use the I2C driver, or any operation to the I2C module may cause a hard fault because clock is not enabled. The configuration structure can be filled by user from scratch, or be set with default values by [I2C\\_MasterGetDefaultConfig\(\)](#). After calling this API, the master is ready to transfer. Example:

## I2C Driver

```
* i2c_master_config_t config = {
* .enableMaster = true,
* .enableStopHold = false,
* .highDrive = false,
* .baudRate_Bps = 100000,
* .glitchFilterWidth = 0
* };
* I2C_MasterInit(I2C0, &config, 12000000U);
*
```

### Parameters

|                     |   |
|---------------------|---|
| <i>base</i>         | I2C base pointer                          |
| <i>masterConfig</i> | pointer to master configuration structure |
| <i>srcClock_Hz</i>  | I2C peripheral clock frequency in Hz      |

### 14.2.7.2 void I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig* )

Call this API to ungate the I2C clock and initializes the I2C with slave configuration.

#### Note

This API should be called at the beginning of the application to use the I2C driver, or any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by [I2C\\_SlaveGetDefaultConfig\(\)](#), or can be filled by the user.

Example

```
* i2c_slave_config_t config = {
* .enableSlave = true,
* .enableGeneralCall = false,
* .addressingMode = kI2C_Address7bit,
* .slaveAddress = 0x1DU,
* .enableWakeUp = false,
* .enablehighDrive = false,
* .enableBaudRateCtl = false
* };
* I2C_SlaveInit(I2C0, &config);
*
```

### Parameters

|                    |  |
|--------------------|--|
| <i>base</i>        | I2C base pointer                         |
| <i>slaveConfig</i> | pointer to slave configuration structure |

### 14.2.7.3 void I2C\_MasterDeinit ( I2C\_Type \* *base* )

Call this API to gate the I2C clock. The I2C master module can't work unless the I2C\_MasterInit is called.

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

**14.2.7.4 void I2C\_SlaveDeinit ( I2C\_Type \* *base* )**

Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C\_SlaveInit is called to enable the clock.

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

**14.2.7.5 void I2C\_MasterGetDefaultConfig ( i2c\_master\_config\_t \* *masterConfig* )**

The purpose of this API is to get the configuration structure initialized for use in the I2C\_MasterConfigure(). Use the initialized structure unchanged in I2C\_MasterConfigure(), or modify some fields of the structure before calling I2C\_MasterConfigure(). Example:

```
* i2c_master_config_t config;
* I2C_MasterGetDefaultConfig(&config);
*
```

## Parameters

|                     |  |
|---------------------|--|
| <i>masterConfig</i> | Pointer to the master configuration structure. |
|---------------------|--|

**14.2.7.6 void I2C\_SlaveGetDefaultConfig ( i2c\_slave\_config\_t \* *slaveConfig* )**

The purpose of this API is to get the configuration structure initialized for use in I2C\_SlaveConfigure(). Modify fields of the structure before calling the I2C\_SlaveConfigure(). Example:

```
* i2c_slave_config_t config;
* I2C_SlaveGetDefaultConfig(&config);
*
```

## Parameters

---

## I2C Driver

|                    |   |
|--------------------|---|
| <i>slaveConfig</i> | Pointer to the slave configuration structure. |
|--------------------|---|

### 14.2.7.7 static void I2C\_Enable ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C base pointer                                    |
| <i>enable</i> | pass true to enable module, false to disable module |

### 14.2.7.8 uint32\_t I2C\_MasterGetStatusFlags ( I2C\_Type \* *base* )

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

Returns

status flag, use status flag to AND [\\_i2c\\_flags](#) to get the related status.

### 14.2.7.9 static uint32\_t I2C\_SlaveGetStatusFlags ( I2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

Returns

status flag, use status flag to AND [\\_i2c\\_flags](#) to get the related status.

### 14.2.7.10 static void I2C\_MasterClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared: kI2C\_ArbitrationLostFlag and kI2C\_IntPendingFlag

## Parameters

|                   |   |
|-------------------|---|
| <i>base</i>       | I2C base pointer  |
| <i>statusMask</i> | The status flag mask, defined in type <code>i2c_status_flag_t</code> . The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• <code>kI2C_StartDetectFlag</code> (if available)</li> <li>• <code>kI2C_StopDetectFlag</code> (if available)</li> <li>• <code>kI2C_ArbitrationLostFlag</code></li> <li>• <code>kI2C_IntPendingFlagFlag</code></li> </ul> |

#### 14.2.7.11 static void I2C\_SlaveClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared: `kI2C_ArbitrationLostFlag` and `kI2C_IntPendingFlag`

## Parameters

|                   |   |
|-------------------|---|
| <i>base</i>       | I2C base pointer  |
| <i>statusMask</i> | The status flag mask, defined in type <code>i2c_status_flag_t</code> . The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• <code>kI2C_StartDetectFlag</code> (if available)</li> <li>• <code>kI2C_StopDetectFlag</code> (if available)</li> <li>• <code>kI2C_ArbitrationLostFlag</code></li> <li>• <code>kI2C_IntPendingFlagFlag</code></li> </ul> |

#### 14.2.7.12 void I2C\_EnableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | I2C base pointer   |
| <i>mask</i> | interrupt source The parameter can be combination of the following source if defined: <ul style="list-style-type: none"> <li>• <code>kI2C_GlobalInterruptEnable</code></li> <li>• <code>kI2C_StopDetectInterruptEnable</code>/<code>kI2C_StartDetectInterruptEnable</code></li> <li>• <code>kI2C_SdaTimeoutInterruptEnable</code></li> </ul> |

#### 14.2.7.13 void I2C\_DisableInterrupts ( I2C\_Type \* *base*, uint32\_t *mask* )

## I2C Driver

### Parameters

|             |  |
|-------------|--|
| <i>base</i> | I2C base pointer   |
| <i>mask</i> | interrupt source The parameter can be combination of the following source if defined: <ul style="list-style-type: none"><li>• kI2C_GlobalInterruptEnable</li><li>• kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable</li><li>• kI2C_SdaTimeoutInterruptEnable</li></ul> |

**14.2.7.14** `static uint32_t I2C_GetDataRegAddr ( I2C_Type * base ) [inline],  
[static]`

This API is used to provide a transfer address for I2C DMA transfer configuration.

### Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | I2C base pointer |
|-------------|------------------|

### Returns

data register address

**14.2.7.15** `void I2C_MasterSetBaudRate ( I2C_Type * base, uint32_t baudRate_Bps,  
uint32_t srcClock_Hz )`

### Parameters

|                     |                            |
|---------------------|----------------------------|
| <i>base</i>         | I2C base pointer           |
| <i>baudRate_Bps</i> | the baud rate value in bps |
| <i>srcClock_Hz</i>  | Source clock               |

**14.2.7.16** `status_t I2C_MasterStart ( I2C_Type * base, uint8_t address, i2c_direction_t  
direction )`

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.



## Parameters

|                  |   |
|------------------|---|
| <i>base</i>      | I2C peripheral base pointer                   |
| <i>address</i>   | 7-bit slave device address.                   |
| <i>direction</i> | Master transfer directions(transmit/receive). |

## Return values

|                         |                                     |
|-------------------------|-------------------------------------|
| <i>kStatus_Success</i>  | Successfully send the start signal. |
| <i>kStatus_I2C_Busy</i> | Current bus is busy.                |

**14.2.7.17 status\_t I2C\_MasterStop ( I2C\_Type \* *base* )**

## Return values

|                            |                                    |
|----------------------------|------------------------------------|
| <i>kStatus_Success</i>     | Successfully send the stop signal. |
| <i>kStatus_I2C_Timeout</i> | Send stop signal failed, timeout.  |

**14.2.7.18 status\_t I2C\_MasterRepeatedStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* )**

## Parameters

|                  |   |
|------------------|---|
| <i>base</i>      | I2C peripheral base pointer                   |
| <i>address</i>   | 7-bit slave device address.                   |
| <i>direction</i> | Master transfer directions(transmit/receive). |

## Return values

|                         |   |
|-------------------------|---|
| <i>kStatus_Success</i>  | Successfully send the start signal.                         |
| <i>kStatus_I2C_Busy</i> | Current bus is busy but not occupied by current I2C master. |

**14.2.7.19 status\_t I2C\_MasterWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize* )**

## I2C Driver

### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | The I2C peripheral base pointer.                   |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

### Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive NAK during transfer. |

#### 14.2.7.20 **status\_t I2C\_MasterReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize* )**

### Note

The I2C\_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C peripheral base pointer.                        |
| <i>rxBuff</i> | The pointer to the data to store the received data. |
| <i>rxSize</i> | The length in bytes of the data to be received.     |

### Return values

|                            |  |
|----------------------------|--|
| <i>kStatus_Success</i>     | Successfully complete the data transmission. |
| <i>kStatus_I2C_Timeout</i> | Send stop signal failed, timeout.            |

#### 14.2.7.21 **status\_t I2C\_SlaveWriteBlocking ( I2C\_Type \* *base*, const uint8\_t \* *txBuff*, size\_t *txSize* )**

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | The I2C peripheral base pointer.                   |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

## Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive NAK during transfer. |

**14.2.7.22** `void I2C_SlaveReadBlocking ( I2C_Type * base, uint8_t * rxBuff, size_t rxSize )`

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C peripheral base pointer.                        |
| <i>rxBuff</i> | The pointer to the data to store the received data. |
| <i>rxSize</i> | The length in bytes of the data to be received.     |

**14.2.7.23** `status_t I2C_MasterTransferBlocking ( I2C_Type * base, i2c_master_transfer_t * xfer )`

## I2C Driver

### Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

### Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | I2C peripheral base address.       |
| <i>xfer</i> | Pointer to the transfer structure. |

### Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Busy</i>             | Previous transmission still not finished.    |
| <i>kStatus_I2C_Timeout</i>          | Transfer error, wait signal timeout.         |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive NAK during transfer. |

**14.2.7.24 void I2C\_MasterTransferCreateHandle ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_callback\_t *callback*, void \* *userData* )**

### Parameters

|                 |   |
|-----------------|---|
| <i>base</i>     | I2C base pointer.   |
| <i>handle</i>   | pointer to i2c_master_handle_t structure to store the transfer state. |
| <i>callback</i> | pointer to user callback function.                                    |
| <i>userData</i> | user parameter passed to the callback function.                       |

**14.2.7.25 status\_t I2C\_MasterTransferNonBlocking ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )**

### Note

Calling the API returns immediately after transfer initiates. The user needs to call I2C\_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus\_I2C\_Busy, the transfer is finished.

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | I2C base pointer.  |
| <i>handle</i> | pointer to <code>i2c_master_handle_t</code> structure which stores the transfer state. |
| <i>xfer</i>   | pointer to <a href="#">i2c_master_transfer_t</a> structure.                            |

## Return values

|                            |   |
|----------------------------|---|
| <i>kStatus_Success</i>     | Successfully start the data transmission. |
| <i>kStatus_I2C_Busy</i>    | Previous transmission still not finished. |
| <i>kStatus_I2C_Timeout</i> | Transfer error, wait signal timeout.      |

#### 14.2.7.26 `status_t I2C_MasterTransferGetCount ( I2C_Type * base, i2c_master_handle_t * handle, size_t * count )`

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | I2C base pointer.  |
| <i>handle</i> | pointer to <code>i2c_master_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.                    |

## Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

#### 14.2.7.27 `void I2C_MasterTransferAbort ( I2C_Type * base, 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

|               |   |
|---------------|---|
| <i>base</i>   | I2C base pointer.   |
| <i>handle</i> | pointer to <code>i2c_master_handle_t</code> structure which stores the transfer state |

**14.2.7.28** void I2C\_MasterTransferHandleIRQ ( I2C\_Type \* *base*, void \* *i2cHandle* )

## Parameters

|                  |  |
|------------------|--|
| <i>base</i>      | I2C base pointer.                                      |
| <i>i2cHandle</i> | pointer to <code>i2c_master_handle_t</code> structure. |

**14.2.7.29** `void I2C_SlaveTransferCreateHandle ( I2C_Type * base, i2c_slave_handle_t * handle, i2c_slave_transfer_callback_t callback, void * userData )`

## Parameters

|                 |   |
|-----------------|---|
| <i>base</i>     | I2C base pointer.   |
| <i>handle</i>   | pointer to <code>i2c_slave_handle_t</code> structure to store the transfer state. |
| <i>callback</i> | pointer to user callback function.  |
| <i>userData</i> | user parameter passed to the callback function.                                   |

**14.2.7.30** `status_t I2C_SlaveTransferNonBlocking ( I2C_Type * base, i2c_slave_handle_t * handle, uint32_t eventMask )`

Call this API after calling the [I2C\\_SlaveInit\(\)](#) and [I2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes events to the callback that was passed into the call to [I2C\\_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 `i2c_slave_transfer_event_t` enumerators for the events you wish to receive. The `kI2C_SlaveTransmitEvent` and `#kLPI2C_SlaveReceiveEvent` events are always enabled and do not need to be included in the mask. Alternatively, pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the `kI2C_SlaveAllEvents` constant is provided as a convenient way to enable all events.

## Parameters

|                  |  |
|------------------|--|
| <i>base</i>      | The I2C peripheral base address.   |
| <i>handle</i>    | Pointer to <code>i2c_slave_handle_t</code> structure which stores the transfer state.  |
| <i>eventMask</i> | Bit mask formed by OR'ing together <code>i2c_slave_transfer_event_t</code> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <code>kI2C_SlaveAllEvents</code> to enable all events. |

## I2C Driver

### Return values

|                         |   |
|-------------------------|---|
| <i>#kStatus_Success</i> | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

### 14.2.7.31 void I2C\_SlaveTransferAbort ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle* )

#### Note

This API can be called at any time to stop slave for handling the bus events.

### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | I2C base pointer.  |
| <i>handle</i> | pointer to i2c_slave_handle_t structure which stores the transfer state. |

### 14.2.7.32 status\_t I2C\_SlaveTransferGetCount ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C base pointer.   |
| <i>handle</i> | pointer to i2c_slave_handle_t structure.                            |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

### 14.2.7.33 void I2C\_SlaveTransferHandleIRQ ( I2C\_Type \* *base*, void \* *i2cHandle* )

### Parameters

|                  |   |
|------------------|---|
| <i>base</i>      | I2C base pointer.   |
| <i>i2cHandle</i> | pointer to i2c_slave_handle_t structure which stores the transfer state |



## 14.3 I2C eDMA Driver

### 14.3.1 Overview

#### Data Structures

- struct [i2c\\_master\\_edma\\_handle\\_t](#)  
*I2C master eDMA transfer structure. [More...](#)*

#### Typedefs

- typedef void(\* [i2c\\_master\\_edma\\_transfer\\_callback\\_t](#))(I2C\_Type \*base, i2c\_master\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
*I2C master eDMA transfer callback typedef.*

### I2C Block eDMA Transfer Operation

- void [I2C\\_MasterCreateEDMAHandle](#) (I2C\_Type \*base, i2c\_master\_edma\_handle\_t \*handle, [i2c\\_master\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, edma\_handle\_t \*edmaHandle)  
*Init the I2C handle which is used in transactional functions.*
- status\_t [I2C\\_MasterTransferEDMA](#) (I2C\_Type \*base, i2c\_master\_edma\_handle\_t \*handle, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a master eDMA non-blocking transfer on the I2C bus.*
- status\_t [I2C\\_MasterTransferGetCountEDMA](#) (I2C\_Type \*base, i2c\_master\_edma\_handle\_t \*handle, size\_t \*count)  
*Get master transfer status during a eDMA non-blocking transfer.*
- void [I2C\\_MasterTransferAbortEDMA](#) (I2C\_Type \*base, i2c\_master\_edma\_handle\_t \*handle)  
*Abort a master eDMA non-blocking transfer in a early time.*

### 14.3.2 Data Structure Documentation

#### 14.3.2.1 struct [i2c\\_master\\_edma\\_handle](#)

I2C master eDMA handle typedef.

#### Data Fields

- [i2c\\_master\\_transfer\\_t](#) transfer  
*I2C master transfer struct.*
- size\_t [transferSize](#)  
*Total bytes to be transferred.*
- uint8\_t [state](#)  
*I2C master transfer status.*
- edma\_handle\_t \* [dmaHandle](#)

## I2C eDMA Driver

*The eDMA handler used.*

- [i2c\\_master\\_edma\\_transfer\\_callback\\_t completionCallback](#)  
*Callback function called after eDMA transfer finished.*
- void \* [userData](#)  
*Callback parameter passed to callback function.*

### 14.3.2.1.0.29 Field Documentation

14.3.2.1.0.29.1 `i2c_master_transfer_t i2c_master_edma_handle_t::transfer`

14.3.2.1.0.29.2 `size_t i2c_master_edma_handle_t::transferSize`

14.3.2.1.0.29.3 `uint8_t i2c_master_edma_handle_t::state`

14.3.2.1.0.29.4 `edma_handle_t* i2c_master_edma_handle_t::dmaHandle`

14.3.2.1.0.29.5 `i2c_master_edma_transfer_callback_t i2c_master_edma_handle_t::completion-  
Callback`

14.3.2.1.0.29.6 `void* i2c_master_edma_handle_t::userData`

### 14.3.3 Typedef Documentation

14.3.3.1 `typedef void(* i2c_master_edma_transfer_callback_t)(I2C_Type *base,  
i2c_master_edma_handle_t *handle, status_t status, void *userData)`

### 14.3.4 Function Documentation

14.3.4.1 `void I2C_MasterCreateEDMAHandle ( I2C_Type * base, i2c_master_edma_  
handle_t * handle, i2c_master_edma_transfer_callback_t callback, void *  
userData, edma_handle_t * edmaHandle )`

Parameters

|                   |  |
|-------------------|--|
| <i>base</i>       | I2C peripheral base address.                   |
| <i>handle</i>     | pointer to i2c_master_edma_handle_t structure. |
| <i>callback</i>   | pointer to user callback function.             |
| <i>userData</i>   | user param passed to the callback function.    |
| <i>edmaHandle</i> | eDMA handle pointer.                           |

14.3.4.2 `status_t I2C_MasterTransferEDMA ( I2C_Type * base, i2c_  
master_edma_handle_t * handle, i2c_master_transfer_t * xfer  
)`

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C peripheral base address.  |
| <i>handle</i> | pointer to <code>i2c_master_edma_handle_t</code> structure.           |
| <i>xfer</i>   | pointer to transfer structure of <code>i2c_master_transfer_t</code> . |

## Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Busy</i>             | Previous transmission still not finished.    |
| <i>kStatus_I2C_Timeout</i>          | Transfer error, wait signal timeout.         |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive Nak during transfer. |

#### 14.3.4.3 `status_t I2C_MasterTransferGetCountEDMA ( I2C_Type * base, i2c_master_edma_handle_t * handle, size_t * count )`

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C peripheral base address.  |
| <i>handle</i> | pointer to <code>i2c_master_edma_handle_t</code> structure.         |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

#### 14.3.4.4 `void I2C_MasterTransferAbortEDMA ( I2C_Type * base, i2c_master_edma_handle_t * handle )`

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C peripheral base address.                                |
| <i>handle</i> | pointer to <code>i2c_master_edma_handle_t</code> structure. |

## I2C DMA Driver

### 14.4 I2C DMA Driver

#### 14.4.1 Overview

##### Data Structures

- struct [i2c\\_master\\_dma\\_handle\\_t](#)  
*I2C master dma transfer structure. [More...](#)*

##### 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.*

##### 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.*

#### 14.4.2 Data Structure Documentation

##### 14.4.2.1 struct [i2c\\_master\\_dma\\_handle](#)

I2C master dma handle typedef.

##### Data Fields

- [i2c\\_master\\_transfer\\_t](#) transfer  
*I2C master transfer struct.*
- size\_t [transferSize](#)  
*Total bytes to be transferred.*
- uint8\_t [state](#)  
*I2C master transfer status.*
- [dma\\_handle\\_t](#) \* [dmaHandle](#)

*The DMA handler used.*

- [i2c\\_master\\_dma\\_transfer\\_callback\\_t completionCallback](#)  
*Callback function called after dma transfer finished.*
- void \* [userData](#)  
*Callback parameter passed to callback function.*

#### 14.4.2.1.0.30 Field Documentation

14.4.2.1.0.30.1 `i2c_master_transfer_t i2c_master_dma_handle_t::transfer`

14.4.2.1.0.30.2 `size_t i2c_master_dma_handle_t::transferSize`

14.4.2.1.0.30.3 `uint8_t i2c_master_dma_handle_t::state`

14.4.2.1.0.30.4 `dma_handle_t* i2c_master_dma_handle_t::dmaHandle`

14.4.2.1.0.30.5 `i2c_master_dma_transfer_callback_t i2c_master_dma_handle_t::completion-Callback`

14.4.2.1.0.30.6 `void* i2c_master_dma_handle_t::userData`

#### 14.4.3 Typedef Documentation

14.4.3.1 `typedef void(* i2c_master_dma_transfer_callback_t)(I2C_Type *base, i2c_master_dma_handle_t *handle, status_t status, void *userData)`

#### 14.4.4 Function Documentation

14.4.4.1 `void I2C_MasterTransferCreateHandleDMA ( I2C_Type * base, i2c_master_dma_handle_t * handle, i2c_master_dma_transfer_callback_t callback, void * userData, dma_handle_t * dmaHandle )`

Parameters

|                  |  |
|------------------|--|
| <i>base</i>      | I2C peripheral base address                  |
| <i>handle</i>    | pointer to i2c_master_dma_handle_t structure |
| <i>callback</i>  | pointer to user callback function            |
| <i>userData</i>  | user param passed to the callback function   |
| <i>dmaHandle</i> | DMA handle pointer                           |

14.4.4.2 `status_t I2C_MasterTransferDMA ( I2C_Type * base, i2c_master_dma_handle_t * handle, i2c_master_transfer_t * xfer )`

## I2C DMA Driver

### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | I2C peripheral base address  |
| <i>handle</i> | pointer to <code>i2c_master_dma_handle_t</code> structure              |
| <i>xfer</i>   | pointer to transfer structure of <a href="#">i2c_master_transfer_t</a> |

### Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Busy</i>             | Previous transmission still not finished.    |
| <i>kStatus_I2C_Timeout</i>          | Transfer error, wait signal timeout.         |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive Nak during transfer. |

#### 14.4.4.3 `status_t I2C_MasterTransferGetCountDMA ( I2C_Type * base, i2c_master_dma_handle_t * handle, size_t * count )`

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C peripheral base address   |
| <i>handle</i> | pointer to <code>i2c_master_dma_handle_t</code> structure           |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

#### 14.4.4.4 `void I2C_MasterTransferAbortDMA ( I2C_Type * base, i2c_master_dma_handle_t * handle )`

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | I2C peripheral base address                               |
| <i>handle</i> | pointer to <code>i2c_master_dma_handle_t</code> structure |

## 14.5 I2C FreeRTOS Driver

### 14.5.1 Overview

#### Data Structures

- struct [i2c\\_rtos\\_handle\\_t](#)  
*I2C FreeRTOS handle. [More...](#)*

#### 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.*

### 14.5.2 Data Structure Documentation

#### 14.5.2.1 struct i2c\_rtos\_handle\_t

##### Data Fields

- I2C\_Type \* [base](#)  
*I2C base address.*
- i2c\_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 transfer.*
- SemaphoreHandle\_t [sem](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_EVENT \* [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_SEM [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 14.5.3 Function Documentation

**14.5.3.1** `status_t I2C_RTOS_Init ( i2c_rtos_handle_t * handle, I2C_Type * base, const i2c_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the I2C module and the related RTOS context.



## Parameters

|                     |  |
|---------------------|--|
| <i>handle</i>       | The RTOS I2C handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the I2C instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up I2C in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the I2C module.                              |

## Returns

status of the operation.

### 14.5.3.2 status\_t I2C\_RTOS\_Deinit ( i2c\_rtos\_handle\_t \* *handle* )

This function deinitializes the I2C module and the related RTOS context.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | The RTOS I2C handle. |
|---------------|----------------------|

### 14.5.3.3 status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )

This function performs an I2C transfer according to data given in the transfer structure.

## Parameters

|                 |   |
|-----------------|---|
| <i>handle</i>   | The RTOS I2C handle.                          |
| <i>transfer</i> | Structure specifying the transfer parameters. |

## Returns

status of the operation.

### 14.6 I2C $\mu$ COS/II Driver

#### 14.6.1 Overview

##### Data Structures

- struct [i2c\\_rtos\\_handle\\_t](#)  
*I2C FreeRTOS handle. [More...](#)*

##### 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.*

#### 14.6.2 Data Structure Documentation

##### 14.6.2.1 struct i2c\_rtos\_handle\_t

##### Data Fields

- I2C\_Type \* [base](#)  
*I2C base address.*
- i2c\_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 transfer.*
- SemaphoreHandle\_t [sem](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_EVENT \* [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_SEM [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 14.6.3 Function Documentation

**14.6.3.1** `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.

## I2C $\mu$ COS/II Driver

### Parameters

|                     |  |
|---------------------|--|
| <i>handle</i>       | The RTOS I2C handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the I2C instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up I2C in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the I2C module.                              |

### Returns

status of the operation.

#### 14.6.3.2 **status\_t I2C\_RTOS\_Deinit ( i2c\_rtos\_handle\_t \* *handle* )**

This function deinitializes the I2C module and the related RTOS context.

### Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | The RTOS I2C handle. |
|---------------|----------------------|

#### 14.6.3.3 **status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )**

This function performs an I2C transfer according to data given in the transfer structure.

### Parameters

|                 |   |
|-----------------|---|
| <i>handle</i>   | The RTOS I2C handle.                          |
| <i>transfer</i> | Structure specifying the transfer parameters. |

### Returns

status of the operation.

## 14.7 I2C $\mu$ COS/III Driver

### 14.7.1 Overview

#### Data Structures

- struct [i2c\\_rtos\\_handle\\_t](#)  
*I2C FreeRTOS handle. [More...](#)*

#### 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.*

### 14.7.2 Data Structure Documentation

#### 14.7.2.1 struct i2c\_rtos\_handle\_t

##### Data Fields

- I2C\_Type \* [base](#)  
*I2C base address.*
- i2c\_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 transfer.*
- SemaphoreHandle\_t [sem](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_EVENT \* [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_SEM [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 14.7.3 Function Documentation

**14.7.3.1** `status_t I2C_RTOS_Init ( i2c_rtos_handle_t * handle, I2C_Type * base, const i2c_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the I2C module and the related RTOS context.

## Parameters

|                     |  |
|---------------------|--|
| <i>handle</i>       | The RTOS I2C handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the I2C instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up I2C in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the I2C module.                              |

## Returns

status of the operation.

### 14.7.3.2 **status\_t I2C\_RTOS\_Deinit ( i2c\_rtos\_handle\_t \* *handle* )**

This function deinitializes the I2C module and the related RTOS context.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | The RTOS I2C handle. |
|---------------|----------------------|

### 14.7.3.3 **status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )**

This function performs an I2C transfer according to data given in the transfer structure.

## Parameters

|                 |   |
|-----------------|---|
| <i>handle</i>   | The RTOS I2C handle.                          |
| <i>transfer</i> | Structure specifying the transfer parameters. |

## Returns

status of the operation.





## Chapter 15

### LLWU: Low-Leakage Wakeup Unit Driver

#### 15.1 Overview

The KSDK provides a Peripheral driver for the Low-Leakage Wakeup Unit (LLWU) module of Kinetis devices. The LLWU module allows the user to select external pin sources and internal modules as a wake-up source from low-leakage power modes.

#### 15.2 External wakeup pins configurations

Configures the external wakeup pins' working modes, gets and clears the wake pin flags. External wakeup pins are accessed by `pinIndex` which is started from 1. Numbers of external pins depend on the SoC configuration.

#### 15.3 Internal wakeup modules configurations

Enables/disables the internal wakeup modules, and gets the modules flags. Internal modules are accessed by `moduleIndex` which is started from 1. Numbers of external pins depend the on SoC configuration.

#### 15.4 Digital pin filter for external wakeup pin configurations

Configures the digital pin filter of the external wakeup pins' working modes, gets and clears the pin filter flags. Digital pins filters are accessed by `filterIndex` which is started from 1. Numbers of external pins depends on the SoC configuration.

#### Data Structures

- struct `llwu_external_pin_filter_mode_t`  
*External input pin filter control structure. [More...](#)*

#### Enumerations

- enum `llwu_external_pin_mode_t` {  
    `kLLWU_ExternalPinDisable` = 0U,  
    `kLLWU_ExternalPinRisingEdge` = 1U,  
    `kLLWU_ExternalPinFallingEdge` = 2U,  
    `kLLWU_ExternalPinAnyEdge` = 3U }  
*External input pin control modes.*
- enum `llwu_pin_filter_mode_t` {  
    `kLLWU_PinFilterDisable` = 0U,  
    `kLLWU_PinFilterRisingEdge` = 1U,  
    `kLLWU_PinFilterFallingEdge` = 2U,  
    `kLLWU_PinFilterAnyEdge` = 3U }  
*Digital filter control modes.*

## Enumeration Type Documentation

### Driver version

- #define **FSL\_LLWU\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 0, 1))  
*LLWU driver version 2.0.1.*

### Low-Leakage Wakeup Unit Control APIs

- void **LLWU\_SetExternalWakeupPinMode** (LLWU\_Type \*base, uint32\_t pinIndex, **llwu\_external\_pin\_mode\_t** pinMode)  
*Sets the external input pin source mode.*
- bool **LLWU\_GetExternalWakeupPinFlag** (LLWU\_Type \*base, uint32\_t pinIndex)  
*Gets the external wakeup source flag.*
- void **LLWU\_ClearExternalWakeupPinFlag** (LLWU\_Type \*base, uint32\_t pinIndex)  
*Clears the external wakeup source flag.*
- static void **LLWU\_EnableInternalModuleInterruptWakup** (LLWU\_Type \*base, uint32\_t moduleIndex, bool enable)  
*Enables/disables the internal module source.*
- static bool **LLWU\_GetInternalWakeupModuleFlag** (LLWU\_Type \*base, uint32\_t moduleIndex)  
*Gets the external wakeup source flag.*
- void **LLWU\_SetPinFilterMode** (LLWU\_Type \*base, uint32\_t filterIndex, **llwu\_external\_pin\_filter\_mode\_t** filterMode)  
*Sets the pin filter configuration.*
- bool **LLWU\_GetPinFilterFlag** (LLWU\_Type \*base, uint32\_t filterIndex)  
*Gets the pin filter configuration.*
- void **LLWU\_ClearPinFilterFlag** (LLWU\_Type \*base, uint32\_t filterIndex)  
*Clear the pin filter configuration.*

## 15.5 Data Structure Documentation

### 15.5.1 struct llwu\_external\_pin\_filter\_mode\_t

#### Data Fields

- uint32\_t **pinIndex**  
*Pin number.*
- **llwu\_pin\_filter\_mode\_t** filterMode  
*Filter mode.*

## 15.6 Macro Definition Documentation

### 15.6.1 #define FSL\_LLWU\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 15.7 Enumeration Type Documentation

### 15.7.1 enum llwu\_external\_pin\_mode\_t

Enumerator

**kLLWU\_ExternalPinDisable** Pin disabled as wakeup input.

***kLLWU\_ExternalPinRisingEdge*** Pin enabled with rising edge detection.

***kLLWU\_ExternalPinFallingEdge*** Pin enabled with falling edge detection.

***kLLWU\_ExternalPinAnyEdge*** Pin enabled with any change detection.

## 15.7.2 enum llwu\_pin\_filter\_mode\_t

Enumerator

***kLLWU\_PinFilterDisable*** Filter disabled.

***kLLWU\_PinFilterRisingEdge*** Filter positive edge detection.

***kLLWU\_PinFilterFallingEdge*** Filter negative edge detection.

***kLLWU\_PinFilterAnyEdge*** Filter any edge detection.

## 15.8 Function Documentation

### 15.8.1 void LLWU\_SetExternalWakeupPinMode ( LLWU\_Type \* *base*, uint32\_t *pinIndex*, llwu\_external\_pin\_mode\_t *pinMode* )

This function sets the external input pin source mode that is used as a wake up source.

Parameters

|                 |  |
|-----------------|--|
| <i>base</i>     | LLWU peripheral base address.  |
| <i>pinIndex</i> | pin index which to be enabled as external wakeup source, start from 1. |
| <i>pinMode</i>  | pin configuration mode defined in llwu_external_pin_modes_t            |

### 15.8.2 bool LLWU\_GetExternalWakeupPinFlag ( LLWU\_Type \* *base*, uint32\_t *pinIndex* )

This function checks the external pin flag to detect whether the MCU is woke up by the specific pin.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>base</i>     | LLWU peripheral base address. |
| <i>pinIndex</i> | pin index, start from 1.      |

Returns

true if the specific pin is wake up source.

## Function Documentation

### 15.8.3 void LLWU\_ClearExternalWakeupPinFlag ( LLWU\_Type \* *base*, uint32\_t *pinIndex* )

This function clears the external wakeup source flag for a specific pin.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>base</i>     | LLWU peripheral base address. |
| <i>pinIndex</i> | pin index, start from 1.      |

#### 15.8.4 static void LLWU\_EnableInternalModuleInterruptWakeup ( LLWU\_Type \* *base*, uint32\_t *moduleIndex*, bool *enable* ) [inline], [static]

This function enables/disables the internal module source mode that is used as a wake up source.

Parameters

|                    |   |
|--------------------|---|
| <i>base</i>        | LLWU peripheral base address.   |
| <i>moduleIndex</i> | module index which to be enabled as internal wakeup source, start from 1. |
| <i>enable</i>      | enable or disable setting   |

#### 15.8.5 static bool LLWU\_GetInternalWakeupModuleFlag ( LLWU\_Type \* *base*, uint32\_t *moduleIndex* ) [inline], [static]

This function checks the external pin flag to detect whether the system is woke up by the specific pin.

Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>base</i>        | LLWU peripheral base address. |
| <i>moduleIndex</i> | module index, start from 1.   |

Returns

true if the specific pin is wake up source.

#### 15.8.6 void LLWU\_SetPinFilterMode ( LLWU\_Type \* *base*, uint32\_t *filterIndex*, llwu\_external\_pin\_filter\_mode\_t *filterMode* )

This function sets the pin filter configuration.

## Function Documentation

### Parameters

|                    |   |
|--------------------|---|
| <i>base</i>        | LLWU peripheral base address.   |
| <i>filterIndex</i> | pin filter index which used to enable/disable the digital filter, start from 1. |
| <i>filterMode</i>  | filter mode configuration   |

### 15.8.7 bool LLWU\_GetPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )

This function gets the pin filter flag.

### Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>base</i>        | LLWU peripheral base address.   |
| <i>filterIndex</i> | pin filter index, start from 1. |

### Returns

true if the flag is a source of existing a low-leakage power mode.

### 15.8.8 void LLWU\_ClearPinFilterFlag ( LLWU\_Type \* *base*, uint32\_t *filterIndex* )

This function clear the pin filter flag.

### Parameters

|                    |  |
|--------------------|--|
| <i>base</i>        | LLWU peripheral base address.                              |
| <i>filterIndex</i> | pin filter index which to be clear the flag, start from 1. |



## Chapter 16

# LPSCI: Universal Asynchronous Receiver/Transmitter

### 16.1 Overview

#### Modules

- [LPSCI DMA Driver](#)
- [LPSCI Driver](#)
- [LPSCI FreeRTOS Driver](#)
- [LPSCI  \$\mu\$ COS/II Driver](#)
- [LPSCI  \$\mu\$ COS/III Driver](#)

### 16.2 LPSCI Driver

#### 16.2.1 Overview

The KSDK provides a peripheral driver for the Inter-Integrated Circuit (LPSCI) module of Kinetis devices. The LPSCI driver can be split into 2 parts: functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for the LPSCI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires knowledge of the LPSCI peripheral and how to organize functional APIs to meet the application requirements. All functional APIs use the peripheral base address as the first parameter. The LPSCI functional operation groups provide the functional APIs set.

The transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral quickly and also in the user's application if the code size and performance of transactional APIs can satisfy the user's requirements. If there are special requirements for the code size and performance, see the transactional API implementation and write custom code. All transactional APIs use the `lpsci_handle_t` as the first parameter. Initialize the handle by calling the `LPSCI_CreateHandle()` API.

Transactional APIs support queue feature for both transmit/receive. Whenever the user calls the `LPSCI_SendDataIRQ()` or `LPSCI_ReceiveDataIRQ()`, the transfer structure is queued into the internally maintained software queue. The driver automatically continues the transmit/receive if the queue is not empty. When a transfer is finished, the callback is called to inform the user about the completion.

The LPSCI transactional APIs support the background receive. Provide the ringbuffer address and size while calling the `LPSCI_CreateHandle()` API. The driver automatically starts receiving the data from the receive buffer into the ringbuffer. When the user makes subsequent calls to the `LPSCI_ReceiveDataIRQ()`, the driver provides the received data in the ringbuffer for user buffer directly and queues the left buffer into the receive queue.

#### 16.2.2 Function groups

##### 16.2.2.1 LPSCI functional Operation

This function group implements the LPSCI functional API. Functional APIs are feature-oriented.

##### 16.2.2.2 LPSCI transactional Operation

This function group implements the LPSCI transactional API.

##### 16.2.2.3 LPSCI transactional Operation

This function group implements the LPSCI DMA transactional API.



## 16.2.3 Typical use case

### 16.2.3.1 LPSCI Operation

```
uint8_t ch;
LPSCI_GetDefaultConfig(UART0, &user_config);
user_config.baudRate = 115200U;

LPSCI_Configure(UART0, &user_config, 120000000U);

LPSCI_WriteData(UART0, txbuff, sizeof(txbuff));

while(1)
{
    LPSCI_ReadData(UART0, &ch, 1);
    LPSCI_WriteData(UART0, &ch, 1);
}
```

### 16.2.3.2 LPSCI Send/Receive using an interrupt method

### 16.2.3.3 LPSCI Receive using the ringbuffer feature

### 16.2.3.4 LPSCI Send/Receive using the DMA method

## Data Structures

- struct `lpsci_config_t`  
LPSCI configure structure. [More...](#)
- struct `lpsci_transfer_t`  
LPSCI transfer structure. [More...](#)

## Driver version

- enum `_lpsci_status` {  
`kStatus_LPSCI_TxBusy` = MAKE\_STATUS(kStatusGroup\_LPSCI, 0),  
`kStatus_LPSCI_RxBusy` = MAKE\_STATUS(kStatusGroup\_LPSCI, 1),  
`kStatus_LPSCI_TxIdle` = MAKE\_STATUS(kStatusGroup\_LPSCI, 2),  
`kStatus_LPSCI_RxIdle` = MAKE\_STATUS(kStatusGroup\_LPSCI, 3),  
`kStatus_LPSCI_FlagCannotClearManually`,  
`kStatus_LPSCI_BaudrateNotSupport`,  
`kStatus_LPSCI_Error` = MAKE\_STATUS(kStatusGroup\_LPSCI, 6),  
`kStatus_LPSCI_RxRingBufferOverrun`,  
`kStatus_LPSCI_RxHardwareOverrun` = MAKE\_STATUS(kStatusGroup\_LPSCI, 8),  
`kStatus_LPSCI_NoiseError` = MAKE\_STATUS(kStatusGroup\_LPSCI, 9),  
`kStatus_LPSCI_FramingError` = MAKE\_STATUS(kStatusGroup\_LPSCI, 10),  
`kStatus_LPSCI_ParityError` = MAKE\_STATUS(kStatusGroup\_LPSCI, 11) }  
*Error codes for the LPSCI driver.*

## LPSCI Driver

- enum `lpsci_parity_mode_t` {  
    `kLPSCI_ParityDisabled` = 0x0U,  
    `kLPSCI_ParityEven` = 0x2U,  
    `kLPSCI_ParityOdd` = 0x3U }  
    *LPSCI parity mode.*
- enum `lpsci_stop_bit_count_t` {  
    `kLPSCI_OneStopBit` = 0U,  
    `kLPSCI_TwoStopBit` = 1U }  
    *LPSCI stop bit count.*
- enum `_lpsci_interrupt_enable_t` {  
    `kLPSCI_RxActiveEdgeInterruptEnable` = (UART0\_BDH\_RXEDGIE\_MASK),  
    `kLPSCI_TxDataRegEmptyInterruptEnable` = (UART0\_C2\_TIE\_MASK << 8),  
    `kLPSCI_TransmissionCompleteInterruptEnable` = (UART0\_C2\_TCIE\_MASK << 8),  
    `kLPSCI_RxDataRegFullInterruptEnable` = (UART0\_C2\_RIE\_MASK << 8),  
    `kLPSCI_IdleLineInterruptEnable` = (UART0\_C2\_ILIE\_MASK << 8),  
    `kLPSCI_RxOverrunInterruptEnable` = (UART0\_C3\_ORIE\_MASK << 16),  
    `kLPSCI_NoiseErrorInterruptEnable` = (UART0\_C3\_NEIE\_MASK << 16),  
    `kLPSCI_FramingErrorInterruptEnable` = (UART0\_C3\_FEIE\_MASK << 16),  
    `kLPSCI_ParityErrorInterruptEnable` = (UART0\_C3\_PEIE\_MASK << 16) }  
    *LPSCI interrupt configuration structure, default settings all disabled.*
- enum `_lpsci_status_flag_t` {  
    `kLPSCI_TxDataRegEmptyFlag` = (UART0\_S1\_TDRE\_MASK),  
    `kLPSCI_TransmissionCompleteFlag`,  
    `kLPSCI_RxDataRegFullFlag`,  
    `kLPSCI_IdleLineFlag` = (UART0\_S1\_IDLE\_MASK),  
    `kLPSCI_RxOverrunFlag`,  
    `kLPSCI_NoiseErrorFlag` = (UART0\_S1\_NF\_MASK),  
    `kLPSCI_FramingErrorFlag`,  
    `kLPSCI_ParityErrorFlag` = (UART0\_S1\_PF\_MASK),  
    `kLPSCI_RxActiveEdgeFlag`,  
    `kLPSCI_RxActiveFlag` }  
    *LPSCI status flags.*
- typedef void(\* `lpsci_transfer_callback_t` )(UART0\_Type \*base, `lpsci_handle_t` \*handle, `status_t` status, void \*userData)  
    *LPSCI transfer callback function.*
- #define `FSL_LPSCI_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 1))  
    *LPSCI driver version 2.0.1.*

## Initialization and deinitialization

- `status_t LPSCI_Init` (UART0\_Type \*base, const `lpsci_config_t` \*config, `uint32_t` srcClock\_Hz)  
    *Initializes an LPSCI instance with the user configuration structure and the peripheral clock.*
- void `LPSCI_Deinit` (UART0\_Type \*base)  
    *Deinitializes an LPSCI instance.*
- void `LPSCI_GetDefaultConfig` (`lpsci_config_t` \*config)  
    *Gets the default configuration structure and saves the configuration to a user-provided pointer.*

- status\_t [LPSCI\\_SetBaudRate](#) (UART0\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the LPSCI instance baudrate.*

## Status

- uint32\_t [LPSCI\\_GetStatusFlags](#) (UART0\_Type \*base)  
*Gets LPSCI status flags.*
- status\_t [LPSCI\\_ClearStatusFlags](#) (UART0\_Type \*base, uint32\_t mask)

## Interrupts

- void [LPSCI\\_EnableInterrupts](#) (UART0\_Type \*base, uint32\_t mask)  
*Enables an LPSCI interrupt according to a provided mask.*
- void [LPSCI\\_DisableInterrupts](#) (UART0\_Type \*base, uint32\_t mask)  
*Disables the LPSCI interrupt according to a provided mask.*
- uint32\_t [LPSCI\\_GetEnabledInterrupts](#) (UART0\_Type \*base)  
*Gets the enabled LPSCI interrupts.*

## Bus Operations

- static void [LPSCI\\_EnableTx](#) (UART0\_Type \*base, bool enable)  
*Enables or disables the LPSCI transmitter.*
- static void [LPSCI\\_EnableRx](#) (UART0\_Type \*base, bool enable)  
*Enables or disables the LPSCI receiver.*
- static void [LPSCI\\_WriteByte](#) (UART0\_Type \*base, uint8\_t data)  
*Writes to the TX register.*
- static uint8\_t [LPSCI\\_ReadByte](#) (UART0\_Type \*base)  
*Reads the RX data register.*
- void [LPSCI\\_WriteBlocking](#) (UART0\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*
- status\_t [LPSCI\\_ReadBlocking](#) (UART0\_Type \*base, uint8\_t \*data, size\_t length)  
*Reads the RX register using a non-blocking method.*

## Transactional

- void [LPSCI\\_TransferCreateHandle](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle, lpsci\_transfer\_callback\_t callback, void \*userData)  
*Initializes the LPSCI handle.*
- void [LPSCI\\_TransferStartRingBuffer](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void [LPSCI\\_TransferStopRingBuffer](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*

## LPSCI Driver

- status\_t [LPSCI\\_TransferSendNonBlocking](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle, lpsci\_transfer\_t \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void [LPSCI\\_TransferAbortSend](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- status\_t [LPSCI\\_TransferGetSendCount](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been written to LPSCI TX register.*
- status\_t [LPSCI\\_TransferReceiveNonBlocking](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle, lpsci\_transfer\_t \*xfer, size\_t \*receivedBytes)  
*Receives buffer of data using the interrupt method.*
- void [LPSCI\\_TransferAbortReceive](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle)  
*Aborts interrupt driven data receiving.*
- status\_t [LPSCI\\_TransferGetReceiveCount](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*
- void [LPSCI\\_TransferHandleIRQ](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle)  
*LPSCI IRQ handle function.*
- void [LPSCI\\_TransferHandleErrorIRQ](#) (UART0\_Type \*base, lpsci\_handle\_t \*handle)  
*LPSCI Error IRQ handle function.*

## 16.2.4 Data Structure Documentation

### 16.2.4.1 struct lpsci\_config\_t

#### Data Fields

- uint32\_t [baudRate\\_Bps](#)  
*LPSCI baud rate.*
- lpsci\_parity\_mode\_t [parityMode](#)  
*Parity mode, disabled (default), even, odd.*
- bool [enableTx](#)  
*Enable TX.*
- bool [enableRx](#)  
*Enable RX.*

### 16.2.4.2 struct lpsci\_transfer\_t

#### Data Fields

- uint8\_t \* [data](#)  
*The buffer of data to be transfer.*
- size\_t [dataSize](#)  
*The byte count to be transfer.*

### 16.2.4.2.0.31 Field Documentation

16.2.4.2.0.31.1 `uint8_t* lpsci_transfer_t::data`

16.2.4.2.0.31.2 `size_t lpsci_transfer_t::dataSize`

### 16.2.5 Macro Definition Documentation

16.2.5.1 `#define FSL_LPSCI_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

### 16.2.6 Typedef Documentation

16.2.6.1 `typedef void(* lpsci_transfer_callback_t)(UART0_Type *base, lpsci_handle_t *handle, status_t status, void *userData)`

### 16.2.7 Enumeration Type Documentation

#### 16.2.7.1 `enum _lpsci_status`

Enumerator

*kStatus\_LPSCI\_TxBusy* Transmitter is busy.  
*kStatus\_LPSCI\_RxBusy* Receiver is busy.  
*kStatus\_LPSCI\_TxIdle* Transmitter is idle.  
*kStatus\_LPSCI\_RxIdle* Receiver is idle.  
*kStatus\_LPSCI\_FlagCannotClearManually* Status flag can't be manually cleared.  
*kStatus\_LPSCI\_BaudrateNotSupport* Baudrate is not support in current clock source.  
*kStatus\_LPSCI\_Error* Error happens on LPSCI.  
*kStatus\_LPSCI\_RxRingBufferOverflow* LPSCI RX software ring buffer overrun.  
*kStatus\_LPSCI\_RxHardwareOverflow* LPSCI RX receiver overrun.  
*kStatus\_LPSCI\_NoiseError* LPSCI noise error.  
*kStatus\_LPSCI\_FramingError* LPSCI framing error.  
*kStatus\_LPSCI\_ParityError* LPSCI parity error.

#### 16.2.7.2 `enum lpsci_parity_mode_t`

Enumerator

*kLPSCI\_ParityDisabled* Parity disabled.  
*kLPSCI\_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.  
*kLPSCI\_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

### 16.2.7.3 enum lpsci\_stop\_bit\_count\_t

Enumerator

***kLPSCI\_OneStopBit*** One stop bit.  
***kLPSCI\_TwoStopBit*** Two stop bits.

### 16.2.7.4 enum \_lpsci\_interrupt\_enable\_t

This structure contains the settings for all LPSCI interrupt configurations.

Enumerator

***kLPSCI\_RxActiveEdgeInterruptEnable*** RX Active Edge interrupt.  
***kLPSCI\_TxDataRegEmptyInterruptEnable*** Transmit data register empty interrupt.  
***kLPSCI\_TransmissionCompleteInterruptEnable*** Transmission complete interrupt.  
***kLPSCI\_RxDataRegFullInterruptEnable*** Receiver data register full interrupt.  
***kLPSCI\_IdleLineInterruptEnable*** Idle line interrupt.  
***kLPSCI\_RxOverrunInterruptEnable*** Receiver Overrun interrupt.  
***kLPSCI\_NoiseErrorInterruptEnable*** Noise error flag interrupt.  
***kLPSCI\_FramingErrorInterruptEnable*** Framing error flag interrupt.  
***kLPSCI\_ParityErrorInterruptEnable*** Parity error flag interrupt.

### 16.2.7.5 enum \_lpsci\_status\_flag\_t

This provides constants for the LPSCI status flags for use in the LPSCI functions.

Enumerator

***kLPSCI\_TxDataRegEmptyFlag*** Tx data register empty flag, sets when Tx buffer is empty.  
***kLPSCI\_TransmissionCompleteFlag*** Transmission complete flag, sets when transmission activity complete.  
***kLPSCI\_RxDataRegFullFlag*** Rx data register full flag, sets when the receive data buffer is full.  
***kLPSCI\_IdleLineFlag*** Idle line detect flag, sets when idle line detected.  
***kLPSCI\_RxOverrunFlag*** Rx Overrun, sets when new data is received before data is read from receive register.  
***kLPSCI\_NoiseErrorFlag*** Rx takes 3 samples of each received bit. If any of these samples differ, noise flag sets  
***kLPSCI\_FramingErrorFlag*** Frame error flag, sets if logic 0 was detected where stop bit expected.  
***kLPSCI\_ParityErrorFlag*** If parity enabled, sets upon parity error detection.  
***kLPSCI\_RxActiveEdgeFlag*** Rx pin active edge interrupt flag, sets when active edge detected.  
***kLPSCI\_RxActiveFlag*** Receiver Active Flag (RAF), sets at beginning of valid start bit.

## 16.2.8 Function Documentation

### 16.2.8.1 `status_t LPSCI_Init ( UART0_Type * base, const lpsci_config_t * config, uint32_t srcClock_Hz )`

This function configures the LPSCI module with user-defined settings. The user can configure the configuration structure and can also get the default configuration by calling the [LPSCI\\_GetDefaultConfig\(\)](#) function. Example below shows how to use this API to configure the LPSCI.

```
* lpsci_config_t lpsciConfig;
* lpsciConfig.baudRate_Bps = 115200U;
* lpsciConfig.parityMode = kLPSCI_ParityDisabled;
* lpsciConfig.stopBitCount = kLPSCI_OneStopBit;
* LPSCI_Init(UART0, &lpsciConfig, 20000000U);
*
```

#### Parameters

|                    |  |
|--------------------|--|
| <i>base</i>        | LPSCI peripheral base address.                   |
| <i>config</i>      | Pointer to user-defined configuration structure. |
| <i>srcClock_Hz</i> | LPSCI clock source frequency in HZ.              |

#### Return values

|   |  |
|---|--|
| <i>kStatus_LPSCI_BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                  | LPSCI initialize succeed                         |

### 16.2.8.2 `void LPSCI_Deinit ( UART0_Type * base )`

This function waits for TX complete, disables TX and RX, and disables the LPSCI clock.

#### Parameters

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

### 16.2.8.3 `void LPSCI_GetDefaultConfig ( lpsci_config_t * config )`

This function initializes the LPSCI configure structure to default value. the default value are: lpsciConfig->baudRate\_Bps = 115200U; lpsciConfig->parityMode = kLPSCI\_ParityDisabled; lpsciConfig->stopBitCount = kLPSCI\_OneStopBit; lpsciConfig->enableTx = false; lpsciConfig->enableRx = false;

## LPSCI Driver

### Parameters

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

#### 16.2.8.4 status\_t LPSCI\_SetBaudRate ( UART0\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the LPSCI module baudrate. This function is used to update the LPSCI module baudrate after the LPSCI module is initialized with the LPSCI\_Init.

```
* LPSCI_SetBaudRate(UART0, 115200U, 200000000U);  
*
```

### Parameters

|                     |                                     |
|---------------------|-------------------------------------|
| <i>base</i>         | LPSCI peripheral base address.      |
| <i>baudRate_Bps</i> | LPSCI baudrate to be set.           |
| <i>srcClock_Hz</i>  | LPSCI clock source frequency in HZ. |

### Return values

|   |  |
|---|--|
| <i>kStatus_LPSCI_-<br/>BaudrateNotSupport</i> | Baudrate is not supported in the current clock source. |
| <i>kStatus_Success</i>                        | Set baudrate succeed                                   |

#### 16.2.8.5 uint32\_t LPSCI\_GetStatusFlags ( UART0\_Type \* *base* )

This function gets all LPSCI status flags. The flags are returned as the logical OR value of the enumerators `_lpsci_flags`. To check a specific status, compare the return value to the enumerators in `_LPSCI_flags`. For example, to check whether the TX is empty:

```
* if (kLPSCI_TxDataRegEmptyFlag |  
* LPSCI_GetStatusFlags(UART0))  
* {  
*     ...  
* }  
*
```



## Parameters

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

## Returns

LPSCI status flags which are ORed by the enumerators in the `_lpsci_flags`.

### 16.2.8.6 void LPSCI\_EnableInterrupts ( UART0\_Type \* *base*, uint32\_t *mask* )

This function enables the LPSCI interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpsci_interrupt_enable`. For example, to enable the TX empty interrupt and RX full interrupt:

```
* LPSCI_EnableInterrupts (UART0,
* kLPSCI_TxDataRegEmptyInterruptEnable |
* kLPSCI_RxDataRegFullInterruptEnable);
```

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | LPSCI peripheral base address.   |
| <i>mask</i> | The interrupts to enable. Logical OR of <code>_lpsci_interrupt_enable</code> . |

### 16.2.8.7 void LPSCI\_DisableInterrupts ( UART0\_Type \* *base*, uint32\_t *mask* )

This function disables the LPSCI interrupts according to a provided mask. The mask is a logical OR of enumeration members. See `_lpsci_interrupt_enable`. For example, to disable TX empty interrupt and RX full interrupt:

```
* LPSCI_DisableInterrupts (UART0,
* kLPSCI_TxDataRegEmptyInterruptEnable |
* kLPSCI_RxDataRegFullInterruptEnable);
```

## Parameters

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

## LPSCI Driver

|             |   |
|-------------|---|
| <i>mask</i> | The interrupts to disable. Logical OR of <code>_LPSCI_interrupt_enable</code> . |
|-------------|---|

### 16.2.8.8 `uint32_t LPSCI_GetEnabledInterrupts ( UART0_Type * base )`

This function gets the enabled LPSCI interrupts, which are returned as the logical OR value of the enumerators `_lpsci_interrupt_enable`. To check a specific interrupts enable status, compare the return value to the enumerators in `_LPSCI_interrupt_enable`. For example, to check whether TX empty interrupt is enabled:

```
*    uint32_t enabledInterrupts = LPSCI_GetEnabledInterrupts(UART0);  
*  
*    if (kLPSCI_TxDataRegEmptyInterruptEnable & enabledInterrupts)  
*    {  
*        ...  
*    }  
*
```

#### Parameters

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

#### Returns

LPSCI interrupt flags which are logical OR of the enumerators in `_LPSCI_interrupt_enable`.

### 16.2.8.9 `static void LPSCI_EnableTx ( UART0_Type * base, bool enable ) [inline], [static]`

This function enables or disables the LPSCI transmitter.

#### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPSCI peripheral base address.    |
| <i>enable</i> | True to enable, false to disable. |

### 16.2.8.10 `static void LPSCI_EnableRx ( UART0_Type * base, bool enable ) [inline], [static]`

This function enables or disables the LPSCI receiver.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | LPSCI peripheral base address.    |
| <i>enable</i> | True to enable, false to disable. |

#### 16.2.8.11 static void LPSCI\_WriteByte ( UART0\_Type \* *base*, uint8\_t *data* ) [inline], [static]

This function writes data to the TX register directly. The upper layer must ensure that the TX register is empty before calling this function.

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | LPSCI peripheral base address. |
| <i>data</i> | Data write to TX register.     |

#### 16.2.8.12 static uint8\_t LPSCI\_ReadByte ( UART0\_Type \* *base* ) [inline], [static]

This function polls the RX register, waits for the RX register to be full, and reads data from the TX register.

## Parameters

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

## Returns

Data read from RX data register.

#### 16.2.8.13 void LPSCI\_WriteBlocking ( UART0\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )

This function polls the TX register, waits for the TX register empty, and writes data to the TX buffer.

## Note

This function does not check whether all the data has been sent out to bus, so before disable TX, check kLPSCI\_TransmissionCompleteFlag to ensure the TX is finished.

## LPSCI Driver

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | LPSCI peripheral base address.      |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

#### 16.2.8.14 **status\_t LPSCI\_ReadBlocking ( UART0\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )**

This function reads data from the TX register directly. The upper layer must ensure that the RX register is full before calling this function.

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | LPSCI peripheral base address.                          |
| <i>data</i>   | Start address of the buffer to store the received data. |
| <i>length</i> | Size of the buffer.                                     |

### Return values

|   |   |
|---|---|
| <i>kStatus_LPSCI_Rx-HardwareOverrun</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_LPSCI_Noise-Error</i>        | Noise error happened while receiving data.      |
| <i>kStatus_LPSCI_Framing-Error</i>      | Framing error happened while receiving data.    |
| <i>kStatus_LPSCI_Parity-Error</i>       | Parity error happened while receiving data.     |
| <i>kStatus_Success</i>                  | Successfully received all data.                 |

#### 16.2.8.15 **void LPSCI\_TransferCreateHandle ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle*, lpsci\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the LPSCI handle, which can be used for other LPSCI transactional APIs. Usually, for a specified LPSCI instance, call this API once to get the initialized handle.

LPSCI driver supports the "background" receiving, which means that the user can set up an RX ring buffer optionally. Data received are stored into the ring buffer even when the user doesn't call the [LPSCI\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, get the received data from the ring buffer directly. The ring buffer is disabled if pass NULL as `ringBuffer`.

## Parameters

|                       |  |
|-----------------------|--|
| <i>handle</i>         | LPSCI handle pointer.  |
| <i>base</i>           | LPSCI peripheral base address.   |
| <i>ringBuffer</i>     | Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.   |

#### 16.2.8.16 void LPSCI\_TransferStartRingBuffer ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )

This function sets up the RX ring buffer to a specific LPSCI handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't call the [LPSCI\\_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, only 31 bytes are used for saving data.

## Parameters

|                       |  |
|-----------------------|--|
| <i>base</i>           | LPSCI peripheral base address.   |
| <i>handle</i>         | LPSCI handle pointer.  |
| <i>ringBuffer</i>     | Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.   |

#### 16.2.8.17 void LPSCI\_TransferStopRingBuffer ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ringbuffer.

## Parameters

---

## LPSCI Driver

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPSCI peripheral base address. |
| <i>handle</i> | LPSCI handle pointer.          |

### 16.2.8.18 **status\_t LPSCI\_TransferSendNonBlocking ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle*, lpsci\_transfer\_t \* *xfer* )**

This function sends data using the 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, LPSCI driver calls the callback function and passes the [kStatus\\_LPSCI\\_TxIdle](#) as status parameter.

Note

The kStatus\_LPSCI\_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. Before disabling the TX, check the kLPSCI\_TransmissionCompleteFlag to ensure that the TX is complete.

Parameters

|               |   |
|---------------|---|
| <i>handle</i> | LPSCI handle pointer.                                 |
| <i>xfer</i>   | LPSCI transfer structure, refer to #LPSCI_transfer_t. |

Return values

|                                |  |
|--------------------------------|--|
| <i>kStatus_Success</i>         | Successfully start the data transmission.  |
| <i>kStatus_LPSCI_TxBusy</i>    | Previous transmission still not finished, data not all written to the TX register. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.  |

### 16.2.8.19 **void LPSCI\_TransferAbortSend ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle* )**

This function aborts the interrupt driven data send.

Parameters

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

### 16.2.8.20 **status\_t LPSCI\_TransferGetSendCount ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been written to LPSCI TX register by interrupt method.

## Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPSCI peripheral base address. |
| <i>handle</i> | LPSCI handle pointer.          |
| <i>count</i>  | Send bytes count.              |

## Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                          |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

#### 16.2.8.21 **status\_t LPSCI\_TransferReceiveNonBlocking ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle*, lpsci\_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 LPSCI driver. When new data arrives, the receive request is serviced first. When all data is received, the LPSCI driver notifies the upper layer through a callback function and passes the status parameter [kStatus\\_LPSCI\\_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 the function returns with the parameter *receivedBytes* set to 5. For the remaining 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the LPSCI driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the *xfer->data*. When all data is received, the upper layer is notified.

## Parameters

|                      |  |
|----------------------|--|
| <i>handle</i>        | LPSCI handle pointer.  |
| <i>xfer</i>          | lpsci transfer structure. See <a href="#">lpsci_transfer_t</a> . |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.                    |

## Return values

## LPSCI Driver

|                                |  |
|--------------------------------|--|
| <i>kStatus_Success</i>         | Successfully queue the transfer into transmit queue. |
| <i>kStatus_LPSCI_RxBusy</i>    | Previous receive request is not finished.            |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                    |

### 16.2.8.22 void LPSCI\_TransferAbortReceive ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle* )

This function aborts interrupt driven data receiving.

Parameters

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

### 16.2.8.23 status\_t LPSCI\_TransferGetReceiveCount ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPSCI peripheral base address. |
| <i>handle</i> | LPSCI handle pointer.          |
| <i>count</i>  | Receive bytes count.           |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                               |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

### 16.2.8.24 void LPSCI\_TransferHandleIRQ ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle* )

This function handles the LPSCI transmit and receive IRQ request.



Parameters

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

**16.2.8.25 void LPSCI\_TransferHandleErrorIRQ ( UART0\_Type \* *base*, lpsci\_handle\_t \* *handle* )**

This function handle the LPSCI error IRQ request.

Parameters

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

### 16.3 LPSCI DMA Driver

#### 16.3.1 Overview

##### Data Structures

- struct [lpsci\\_dma\\_handle\\_t](#)  
*LPSCI DMA handle. [More...](#)*

##### Typedefs

- typedef void(\* [lpsci\\_dma\\_transfer\\_callback\\_t](#))(UART0\_Type \*base, lpsci\_dma\_handle\_t \*handle, status\_t status, void \*userData)  
*LPSCI transfer callback function.*

##### eDMA transactional

- void [LPSCI\\_TransferCreateHandleDMA](#) (UART0\_Type \*base, lpsci\_dma\_handle\_t \*handle, [lpsci\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*txDmaHandle, [dma\\_handle\\_t](#) \*rxDmaHandle)  
*Initializes the LPSCI handle which is used in transactional functions.*
- status\_t [LPSCI\\_TransferSendDMA](#) (UART0\_Type \*base, lpsci\_dma\_handle\_t \*handle, [lpsci\\_transfer\\_t](#) \*xfer)  
*Sends data using DMA.*
- status\_t [LPSCI\\_TransferReceiveDMA](#) (UART0\_Type \*base, lpsci\_dma\_handle\_t \*handle, [lpsci\\_transfer\\_t](#) \*xfer)  
*Receives data using DMA.*
- void [LPSCI\\_TransferAbortSendDMA](#) (UART0\_Type \*base, lpsci\_dma\_handle\_t \*handle)  
*Aborts the sent data using DMA.*
- void [LPSCI\\_TransferAbortReceiveDMA](#) (UART0\_Type \*base, lpsci\_dma\_handle\_t \*handle)  
*Aborts the receive data using DMA.*
- status\_t [LPSCI\\_TransferGetSendCountDMA](#) (UART0\_Type \*base, lpsci\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes written to the LPSCI TX register.*
- status\_t [LPSCI\\_TransferGetReceiveCountDMA](#) (UART0\_Type \*base, lpsci\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Gets the number of bytes that have been received.*

#### 16.3.2 Data Structure Documentation

##### 16.3.2.1 struct \_lpsci\_dma\_handle

##### Data Fields

- UART0\_Type \* [base](#)

- *LPSCI peripheral base address.*  
• `lpsci_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.*

#### 16.3.2.1.0.32 Field Documentation

16.3.2.1.0.32.1 `UART0_Type* lpsci_dma_handle_t::base`

16.3.2.1.0.32.2 `lpsci_dma_transfer_callback_t lpsci_dma_handle_t::callback`

16.3.2.1.0.32.3 `void* lpsci_dma_handle_t::userData`

16.3.2.1.0.32.4 `size_t lpsci_dma_handle_t::rxDataSizeAll`

16.3.2.1.0.32.5 `size_t lpsci_dma_handle_t::txDataSizeAll`

16.3.2.1.0.32.6 `dma_handle_t* lpsci_dma_handle_t::txDmaHandle`

16.3.2.1.0.32.7 `dma_handle_t* lpsci_dma_handle_t::rxDmaHandle`

16.3.2.1.0.32.8 `volatile uint8_t lpsci_dma_handle_t::txState`

#### 16.3.3 Typedef Documentation

16.3.3.1 `typedef void(* lpsci_dma_transfer_callback_t)(UART0_Type *base,  
lpsci_dma_handle_t *handle, status_t status, void *userData)`

#### 16.3.4 Function Documentation

16.3.4.1 `void LPSCI_TransferCreateHandleDMA ( UART0_Type * base,  
lpsci_dma_handle_t * handle, lpsci_dma_transfer_callback_t callback, void *  
userData, dma_handle_t * txDmaHandle, dma_handle_t * rxDmaHandle )`

## LPSCI DMA Driver

### Parameters

|                    |  |
|--------------------|--|
| <i>handle</i>      | Pointer to <code>lpsci_dma_handle_t</code> structure |
| <i>base</i>        | LPSCI peripheral base address                        |
| <i>rxDmaHandle</i> | User requested DMA handle for RX DMA transfer        |
| <i>txDmaHandle</i> | User requested DMA handle for TX DMA transfer        |

### 16.3.4.2 **status\_t LPSCI\_TransferSendDMA ( UART0\_Type \* *base*, lpsci\_dma\_handle\_t \* *handle*, lpsci\_transfer\_t \* *xfer* )**

This function sends data using DMA. This is a non-blocking function, which returns immediately. When all data is sent, the send callback function is called.

### Parameters

|               |  |
|---------------|--|
| <i>handle</i> | LPSCI handle pointer.  |
| <i>xfer</i>   | LPSCI DMA transfer structure, see <a href="#">lpsci_transfer_t</a> . |

### Return values

|                                |                               |
|--------------------------------|-------------------------------|
| <i>kStatus_Success</i>         | if successful, others failed. |
| <i>kStatus_LPSCI_TxBusy</i>    | Previous transfer on going.   |
| <i>kStatus_InvalidArgument</i> | Invalid argument.             |

### 16.3.4.3 **status\_t LPSCI\_TransferReceiveDMA ( UART0\_Type \* *base*, lpsci\_dma\_handle\_t \* *handle*, lpsci\_transfer\_t \* *xfer* )**

This function receives data using DMA. This is a non-blocking function, which returns immediately. When all data is received, the receive callback function is called.

### Parameters

|               |  |
|---------------|--|
| <i>handle</i> | Pointer to <code>lpsci_dma_handle_t</code> structure                 |
| <i>xfer</i>   | LPSCI DMA transfer structure, see <a href="#">lpsci_transfer_t</a> . |

### Return values

---

|                                |                               |
|--------------------------------|-------------------------------|
| <i>kStatus_Success</i>         | if successful, others failed. |
| <i>kStatus_LPSCI_RxBusy</i>    | Previous transfer on going.   |
| <i>kStatus_InvalidArgument</i> | Invalid argument.             |

#### 16.3.4.4 void LPSCI\_TransferAbortSendDMA ( UART0\_Type \* *base*, lpsci\_dma\_handle\_t \* *handle* )

This function aborts the sent data using DMA.

Parameters

|               |  |
|---------------|--|
| <i>handle</i> | Pointer to lpsci_dma_handle_t structure. |
|---------------|--|

#### 16.3.4.5 void LPSCI\_TransferAbortReceiveDMA ( UART0\_Type \* *base*, lpsci\_dma\_handle\_t \* *handle* )

This function aborts the receive data using DMA.

Parameters

|               |  |
|---------------|--|
| <i>handle</i> | Pointer to lpsci_dma_handle_t structure. |
|---------------|--|

#### 16.3.4.6 status\_t LPSCI\_TransferGetSendCountDMA ( UART0\_Type \* *base*, lpsci\_dma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been written to the LPSCI TX register by DMA.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPSCI peripheral base address. |
| <i>handle</i> | LPSCI handle pointer.          |
| <i>count</i>  | Send bytes count.              |

Return values

---

## LPSCI DMA Driver

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

### 16.3.4.7 **status\_t LPSCI\_TransferGetReceiveCountDMA ( UART0\_Type \* *base*, lpsci\_dma\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been received.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | LPSCI peripheral base address. |
| <i>handle</i> | LPSCI handle pointer.          |
| <i>count</i>  | Receive bytes count.           |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                                     |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

## 16.4 LPSCI FreeRTOS Driver

### 16.4.1 Overview

#### LPSCI RTOS Operation

- int [LPSCI\\_RTOS\\_Init](#) (lpsci\_rtos\_handle\_t \*handle, lpsci\_handle\_t \*t\_handle, const struct rtos\_lpsci\_config \*cfg)  
*Initializes an LPSCI instance for operation in RTOS.*
- int [LPSCI\\_RTOS\\_Deinit](#) (lpsci\_rtos\_handle\_t \*handle)  
*Deinitializes an LPSCI instance for operation.*

#### LPSCI transactional Operation

- int [LPSCI\\_RTOS\\_Send](#) (lpsci\_rtos\_handle\_t \*handle, const uint8\_t \*buffer, uint32\_t length)  
*Send data in background.*
- int [LPSCI\\_RTOS\\_Receive](#) (lpsci\_rtos\_handle\_t \*handle, uint8\_t \*buffer, uint32\_t length, size\_t \*received)  
*Receives data.*

### 16.4.2 Function Documentation

#### 16.4.2.1 int LPSCI\_RTOS\_Init ( lpsci\_rtos\_handle\_t \* *handle*, lpsci\_handle\_t \* *t\_handle*, const struct rtos\_lpsci\_config \* *cfg* )

Parameters

|                 |   |
|-----------------|---|
| <i>handle</i>   | The RTOS LPSCI handle, the pointer to allocated space for RTOS context.             |
| <i>t_handle</i> | The pointer to allocated space where to store transactional layer internal state.   |
| <i>cfg</i>      | The pointer to the parameters required to configure the LPSCI after initialization. |

Returns

0 succeed, others failed

#### 16.4.2.2 int LPSCI\_RTOS\_Deinit ( lpsci\_rtos\_handle\_t \* *handle* )

This function deinitializes the LPSCI module, set all register value to reset value and releases the resources.

## LPSCI FreeRTOS Driver

### Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPSCI handle. |
|---------------|------------------------|

#### 16.4.2.3 int LPSCI\_RTOS\_Send ( lpsci\_rtos\_handle\_t \* *handle*, const uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is synchronous API. If the HW buffer is full, the task is in the blocked state.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS LPSCI handle.         |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 16.4.2.4 int LPSCI\_RTOS\_Receive ( lpsci\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

It is synchronous API.

This function receives data from LPSCI. If any data is immediately available it is returned immediately and the number of bytes received.

### Parameters

|                 |  |
|-----------------|--|
| <i>handle</i>   | The RTOS LPSCI handle.   |
| <i>buffer</i>   | The pointer to buffer where to write received data.                            |
| <i>length</i>   | The number of bytes to receive.  |
| <i>received</i> | The pointer to variable of size_t where the number of received data is filled. |



## 16.5 LPSCI $\mu$ COS/II Driver

### 16.5.1 Overview

#### LPSCI RTOS Operation

- int [LPSCI\\_RTOS\\_Init](#) (lpsci\_rtos\_handle\_t \*handle, lpsci\_handle\_t \*t\_handle, const struct rtos\_lpsci\_config \*cfg)  
*Initializes an LPSCI instance for operation in RTOS.*
- int [LPSCI\\_RTOS\\_Deinit](#) (lpsci\_rtos\_handle\_t \*handle)  
*Deinitializes an LPSCI instance for operation.*

#### LPSCI transactional Operation

- int [LPSCI\\_RTOS\\_Send](#) (lpsci\_rtos\_handle\_t \*handle, const uint8\_t \*buffer, uint32\_t length)  
*Send data in background.*
- int [LPSCI\\_RTOS\\_Receive](#) (lpsci\_rtos\_handle\_t \*handle, uint8\_t \*buffer, uint32\_t length, size\_t \*received)  
*Receives data.*

### 16.5.2 Function Documentation

#### 16.5.2.1 int LPSCI\_RTOS\_Init ( lpsci\_rtos\_handle\_t \* *handle*, lpsci\_handle\_t \* *t\_handle*, const struct rtos\_lpsci\_config \* *cfg* )

Parameters

|                       |   |
|-----------------------|---|
| <i>handle</i>         | The RTOS LPSCI handle, the pointer to allocated space for RTOS context.             |
| <i>lpsci_t_handle</i> | The pointer to allocated space where to store transactional layer internal state.   |
| <i>cfg</i>            | The pointer to the parameters required to configure the LPSCI after initialization. |

Returns

0 succeed, others failed

#### 16.5.2.2 int LPSCI\_RTOS\_Deinit ( lpsci\_rtos\_handle\_t \* *handle* )

This function deinitializes the LPSCI module, set all register value to reset value and releases the resources.

## LPSCI $\mu$ COS/II Driver

### Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPSCI handle. |
|---------------|------------------------|

### 16.5.2.3 int LPSCI\_RTOS\_Send ( lpsci\_rtos\_handle\_t \* *handle*, const uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is synchronous API. If the HW buffer is full, the task is in the blocked state.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS LPSCI handle.         |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

### 16.5.2.4 int LPSCI\_RTOS\_Receive ( lpsci\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

It is synchronous API.

This function receives data from LPSCI. If any data is immediately available it is returned immediately and the number of bytes received.

### Parameters

|                 |  |
|-----------------|--|
| <i>handle</i>   | The RTOS LPSCI handle.   |
| <i>buffer</i>   | The pointer to buffer where to write received data.                            |
| <i>length</i>   | The number of bytes to receive.  |
| <i>received</i> | The pointer to variable of size_t where the number of received data is filled. |

## 16.6 LPSCI $\mu$ COS/III Driver

### 16.6.1 Overview

#### LPSCI RTOS Operation

- int [LPSCI\\_RTOS\\_Init](#) (lpsci\_rtos\_handle\_t \*handle, lpsci\_handle\_t \*t\_handle, const struct rtos\_lpsci\_config \*cfg)  
*Initializes an LPSCI instance for operation in RTOS.*
- int [LPSCI\\_RTOS\\_Deinit](#) (lpsci\_rtos\_handle\_t \*handle)  
*Deinitializes an LPSCI instance for operation.*

#### LPSCI transactional Operation

- int [LPSCI\\_RTOS\\_Send](#) (lpsci\_rtos\_handle\_t \*handle, const uint8\_t \*buffer, uint32\_t length)  
*Send data in background.*
- int [LPSCI\\_RTOS\\_Receive](#) (lpsci\_rtos\_handle\_t \*handle, uint8\_t \*buffer, uint32\_t length, size\_t \*received)  
*Receives data.*

### 16.6.2 Function Documentation

#### 16.6.2.1 int LPSCI\_RTOS\_Init ( lpsci\_rtos\_handle\_t \* *handle*, lpsci\_handle\_t \* *t\_handle*, const struct rtos\_lpsci\_config \* *cfg* )

Parameters

|                       |   |
|-----------------------|---|
| <i>handle</i>         | The RTOS LPSCI handle, the pointer to allocated space for RTOS context.             |
| <i>lpsci_t_handle</i> | The pointer to allocated space where to store transactional layer internal state.   |
| <i>cfg</i>            | The pointer to the parameters required to configure the LPSCI after initialization. |

Returns

0 succeed, others failed

#### 16.6.2.2 int LPSCI\_RTOS\_Deinit ( lpsci\_rtos\_handle\_t \* *handle* )

This function deinitializes the LPSCI module, set all register value to reset value and releases the resources.

## LPSCI $\mu$ COS/III Driver

### Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS LPSCI handle. |
|---------------|------------------------|

### 16.6.2.3 int LPSCI\_RTOS\_Send ( lpsci\_rtos\_handle\_t \* *handle*, const uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is synchronous API. If the HW buffer is full, the task is in the blocked state.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS LPSCI handle.         |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

### 16.6.2.4 int LPSCI\_RTOS\_Receive ( lpsci\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

It is synchronous API.

This function receives data from LPSCI. If any data is immediately available it is returned immediately and the number of bytes received.

### Parameters

|                 |  |
|-----------------|--|
| <i>handle</i>   | The RTOS LPSCI handle.   |
| <i>buffer</i>   | The pointer to buffer where to write received data.                            |
| <i>length</i>   | The number of bytes to receive.  |
| <i>received</i> | The pointer to variable of size_t where the number of received data is filled. |

## Chapter 17

### LPTMR: Low-Power Timer

#### 17.1 Overview

The KSDK provides a driver for the Low-Power Timer (LPTMR) of Kinetis devices.

#### 17.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

##### 17.2.1 Initialization and deinitialization

The function [LPTMR\\_Init\(\)](#) initializes the LPTMR with specified configurations. The function [LPTMR\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the LPTMR for timer or pulse counter mode. It also sets up the LPTMR's free running mode operation and clock source.

The function [LPTMR\\_DeInit\(\)](#) disables the LPTMR module and gate the module clock.

##### 17.2.2 Timer period Operations

The function [LPTMR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers counts from 0 till it equals the count value set here.

The function [LPTMR\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. User can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds

##### 17.2.3 Start and Stop timer operations

The function [LPTMR\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer counts up to the count value set earlier via the [LPTMR\\_SetPeriod\(\)](#) function. Each time the timer reaches count value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

The function [LPTMR\\_StopTimer\(\)](#) stops the timer counting and resets the timer's counter register

## Typical use case

### 17.2.4 Status

Provides functions to get and clear the LPTMR status.

### 17.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get current enabled interrupts.

## 17.3 Typical use case

### 17.3.1 LPTMR tick example

Updates the LPTMR period and toggles an LED periodically.

```
int main(void)
{
    uint32_t currentCounter = 0U;
    lptmr_config_t lptmrConfig;

    LED_INIT();

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();

    /* Configure LPTMR */
    LPTMR_GetDefaultConfig(&lptmrConfig);

    /* Initialize the LPTMR */
    LPTMR_Init(LPTMR0, &lptmrConfig);

    /* Set timer period */
    LPTMR_SetTimerPeriod(LPTMR0, USEC_TO_COUNT(1000000U, LPTMR_SOURCE_CLOCK));

    /* Enable timer interrupt */
    LPTMR_EnableInterrupts(LPTMR0,
        kLPTMR_TimerInterruptEnable);

    /* Enable at the NVIC */
    EnableIRQ(LPTMR0_IRQn);

    PRINTF("Low Power Timer Example\r\n");

    /* Start counting */
    LPTMR_StartTimer(LPTMR0);
    while (1)
    {
        if (currentCounter != lptmrCounter)
        {
            currentCounter = lptmrCounter;
            PRINTF("LPTMR interrupt No.%d \r\n", currentCounter);
        }
    }
}
```

## Data Structures

- struct [lptmr\\_config\\_t](#)  
*LPTMR config structure. [More...](#)*

## Enumerations

- enum `lptmr_pin_select_t` {  
`kLPTMR_PinSelectInput_0` = 0x0U,  
`kLPTMR_PinSelectInput_1` = 0x1U,  
`kLPTMR_PinSelectInput_2` = 0x2U,  
`kLPTMR_PinSelectInput_3` = 0x3U }  
*LPTMR pin selection, used in pulse counter mode.*
- enum `lptmr_pin_polarity_t` {  
`kLPTMR_PinPolarityActiveHigh` = 0x0U,  
`kLPTMR_PinPolarityActiveLow` = 0x1U }  
*LPTMR pin polarity, used in pulse counter mode.*
- enum `lptmr_timer_mode_t` {  
`kLPTMR_TimerModeTimeCounter` = 0x0U,  
`kLPTMR_TimerModePulseCounter` = 0x1U }  
*LPTMR timer mode selection.*
- enum `lptmr_prescaler_glitch_value_t` {  
`kLPTMR_Prescale_Glitch_0` = 0x0U,  
`kLPTMR_Prescale_Glitch_1` = 0x1U,  
`kLPTMR_Prescale_Glitch_2` = 0x2U,  
`kLPTMR_Prescale_Glitch_3` = 0x3U,  
`kLPTMR_Prescale_Glitch_4` = 0x4U,  
`kLPTMR_Prescale_Glitch_5` = 0x5U,  
`kLPTMR_Prescale_Glitch_6` = 0x6U,  
`kLPTMR_Prescale_Glitch_7` = 0x7U,  
`kLPTMR_Prescale_Glitch_8` = 0x8U,  
`kLPTMR_Prescale_Glitch_9` = 0x9U,  
`kLPTMR_Prescale_Glitch_10` = 0xAU,  
`kLPTMR_Prescale_Glitch_11` = 0xBU,  
`kLPTMR_Prescale_Glitch_12` = 0xCU,  
`kLPTMR_Prescale_Glitch_13` = 0xDU,  
`kLPTMR_Prescale_Glitch_14` = 0xEU,  
`kLPTMR_Prescale_Glitch_15` = 0xFU }  
*LPTMR prescaler/glitch filter values.*
- enum `lptmr_prescaler_clock_select_t` {  
`kLPTMR_PrescalerClock_0` = 0x0U,  
`kLPTMR_PrescalerClock_1` = 0x1U,  
`kLPTMR_PrescalerClock_2` = 0x2U,  
`kLPTMR_PrescalerClock_3` = 0x3U }  
*LPTMR prescaler/glitch filter clock select.*
- enum `lptmr_interrupt_enable_t` { `kLPTMR_TimerInterruptEnable` = `LPTMR_CSR_TIE_MASK` }  
*List of LPTMR interrupts.*
- enum `lptmr_status_flags_t` { `kLPTMR_TimerCompareFlag` = `LPTMR_CSR_TCF_MASK` }  
*List of LPTMR status flags.*

## Driver version

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

### Initialization and deinitialization

- void [LPTMR\\_Init](#) (LPTMR\_Type \*base, const [lptmr\\_config\\_t](#) \*config)  
*Ungate the LPTMR clock and configures the peripheral for basic operation.*
- void [LPTMR\\_Deinit](#) (LPTMR\_Type \*base)  
*Gate the LPTMR clock.*
- void [LPTMR\\_GetDefaultConfig](#) ([lptmr\\_config\\_t](#) \*config)  
*Fill in the LPTMR config struct with the default settings.*

### Interrupt Interface

- static void [LPTMR\\_EnableInterrupts](#) (LPTMR\_Type \*base, uint32\_t mask)  
*Enables the selected LPTMR interrupts.*
- static void [LPTMR\\_DisableInterrupts](#) (LPTMR\_Type \*base, uint32\_t mask)  
*Disables the selected LPTMR interrupts.*
- static uint32\_t [LPTMR\\_GetEnabledInterrupts](#) (LPTMR\_Type \*base)  
*Gets the enabled LPTMR interrupts.*

### Status Interface

- static uint32\_t [LPTMR\\_GetStatusFlags](#) (LPTMR\_Type \*base)  
*Gets the LPTMR status flags.*
- static void [LPTMR\\_ClearStatusFlags](#) (LPTMR\_Type \*base, uint32\_t mask)  
*Clears the LPTMR status flags.*

### Read and Write the timer period

- static void [LPTMR\\_SetTimerPeriod](#) (LPTMR\_Type \*base, uint16\_t ticks)  
*Sets the timer period in units of count.*
- static uint16\_t [LPTMR\\_GetCurrentTimerCount](#) (LPTMR\_Type \*base)  
*Reads the current timer counting value.*

### Timer Start and Stop

- static void [LPTMR\\_StartTimer](#) (LPTMR\_Type \*base)  
*Starts the timer counting.*
- static void [LPTMR\\_StopTimer](#) (LPTMR\_Type \*base)  
*Stops the timer counting.*

## 17.4 Data Structure Documentation

### 17.4.1 struct [lptmr\\_config\\_t](#)

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the [LPTMR\\_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

- [lptmr\\_timer\\_mode\\_t](#) timerMode  
*Time counter mode or pulse counter mode.*
- [lptmr\\_pin\\_select\\_t](#) pinSelect  
*LPTMR pulse input pin select; used only in pulse counter mode.*
- [lptmr\\_pin\\_polarity\\_t](#) pinPolarity  
*LPTMR pulse input pin polarity; used only in pulse counter mode.*
- bool [enableFreeRunning](#)  
*true: enable free running, counter is reset on overflow false: counter is reset when the compare flag is set*
- bool [bypassPrescaler](#)  
*true: bypass prescaler; false: use clock from prescaler*
- [lptmr\\_prescaler\\_clock\\_select\\_t](#) prescalerClockSource  
*LPTMR clock source.*
- [lptmr\\_prescaler\\_glitch\\_value\\_t](#) value  
*Prescaler or glitch filter value.*

## 17.5 Enumeration Type Documentation

### 17.5.1 enum lptmr\_pin\_select\_t

Enumerator

- kLPTMR\_PinSelectInput\_0*** Pulse counter input 0 is selected.  
***kLPTMR\_PinSelectInput\_1*** Pulse counter input 1 is selected.  
***kLPTMR\_PinSelectInput\_2*** Pulse counter input 2 is selected.  
***kLPTMR\_PinSelectInput\_3*** Pulse counter input 3 is selected.

### 17.5.2 enum lptmr\_pin\_polarity\_t

Enumerator

- kLPTMR\_PinPolarityActiveHigh*** Pulse Counter input source is active-high.  
***kLPTMR\_PinPolarityActiveLow*** Pulse Counter input source is active-low.

### 17.5.3 enum lptmr\_timer\_mode\_t

Enumerator

- kLPTMR\_TimerModeTimeCounter*** Time Counter mode.  
***kLPTMR\_TimerModePulseCounter*** Pulse Counter mode.

### 17.5.4 enum lptmr\_prescaler\_glitch\_value\_t

Enumerator

|   |  |
|---|--|
| <b><i>kLPTMR_Prescale_Glitch_0</i></b>  | Prescaler divide 2, glitch filter does not support this setting. |
| <b><i>kLPTMR_Prescale_Glitch_1</i></b>  | Prescaler divide 4, glitch filter 2.                             |
| <b><i>kLPTMR_Prescale_Glitch_2</i></b>  | Prescaler divide 8, glitch filter 4.                             |
| <b><i>kLPTMR_Prescale_Glitch_3</i></b>  | Prescaler divide 16, glitch filter 8.                            |
| <b><i>kLPTMR_Prescale_Glitch_4</i></b>  | Prescaler divide 32, glitch filter 16.                           |
| <b><i>kLPTMR_Prescale_Glitch_5</i></b>  | Prescaler divide 64, glitch filter 32.                           |
| <b><i>kLPTMR_Prescale_Glitch_6</i></b>  | Prescaler divide 128, glitch filter 64.                          |
| <b><i>kLPTMR_Prescale_Glitch_7</i></b>  | Prescaler divide 256, glitch filter 128.                         |
| <b><i>kLPTMR_Prescale_Glitch_8</i></b>  | Prescaler divide 512, glitch filter 256.                         |
| <b><i>kLPTMR_Prescale_Glitch_9</i></b>  | Prescaler divide 1024, glitch filter 512.                        |
| <b><i>kLPTMR_Prescale_Glitch_10</i></b> | Prescaler divide 2048 glitch filter 1024.                        |
| <b><i>kLPTMR_Prescale_Glitch_11</i></b> | Prescaler divide 4096, glitch filter 2048.                       |
| <b><i>kLPTMR_Prescale_Glitch_12</i></b> | Prescaler divide 8192, glitch filter 4096.                       |
| <b><i>kLPTMR_Prescale_Glitch_13</i></b> | Prescaler divide 16384, glitch filter 8192.                      |
| <b><i>kLPTMR_Prescale_Glitch_14</i></b> | Prescaler divide 32768, glitch filter 16384.                     |
| <b><i>kLPTMR_Prescale_Glitch_15</i></b> | Prescaler divide 65536, glitch filter 32768.                     |

### 17.5.5 enum lptmr\_prescaler\_clock\_select\_t

Note

Clock connections are SoC-specific

Enumerator

|                                       |   |
|---------------------------------------|---|
| <b><i>kLPTMR_PrescalerClock_0</i></b> | Prescaler/glitch filter clock 0 selected. |
| <b><i>kLPTMR_PrescalerClock_1</i></b> | Prescaler/glitch filter clock 1 selected. |
| <b><i>kLPTMR_PrescalerClock_2</i></b> | Prescaler/glitch filter clock 2 selected. |
| <b><i>kLPTMR_PrescalerClock_3</i></b> | Prescaler/glitch filter clock 3 selected. |

### 17.5.6 enum lptmr\_interrupt\_enable\_t

Enumerator

|   |                         |
|---|-------------------------|
| <b><i>kLPTMR_TimerInterruptEnable</i></b> | Timer interrupt enable. |
|---|-------------------------|

### 17.5.7 enum lptmr\_status\_flags\_t

Enumerator

***kLPTMR\_TimerCompareFlag*** Timer compare flag.

## 17.6 Function Documentation

### 17.6.1 void LPTMR\_Init ( LPTMR\_Type \* *base*, const lptmr\_config\_t \* *config* )

Note

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

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | LPTMR peripheral base address             |
| <i>config</i> | Pointer to user's LPTMR config structure. |

### 17.6.2 void LPTMR\_Deinit ( LPTMR\_Type \* *base* )

Parameters

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

### 17.6.3 void LPTMR\_GetDefaultConfig ( lptmr\_config\_t \* *config* )

The default values are:

```
* config->timerMode = kLPTMR_TimerModeTimeCounter;
* config->pinSelect = kLPTMR_PinSelectInput_0;
* config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
* config->enableFreeRunning = false;
* config->bypassPrescaler = true;
* config->prescalerClockSource = kLPTMR_PrescalerClock_1;
* config->value = kLPTMR_Prescale_Glitch_0;
*
```

Parameters

## Function Documentation

|               |   |
|---------------|---|
| <i>config</i> | Pointer to user's LPTMR config structure. |
|---------------|---|

**17.6.4 static void LPTMR\_EnableInterrupts ( LPTMR\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | LPTMR peripheral base address   |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">lptmr_interrupt_enable_t</a> |

**17.6.5 static void LPTMR\_DisableInterrupts ( LPTMR\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

### Parameters

|             |  |
|-------------|--|
| <i>base</i> | LPTMR peripheral base address  |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">lptmr_interrupt_enable_t</a> |

**17.6.6 static uint32\_t LPTMR\_GetEnabledInterrupts ( LPTMR\_Type \* *base* )**  
**[inline], [static]**

### Parameters

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

### Returns

The enabled interrupts. This is the logical OR of members of the enumeration [lptmr\\_interrupt\\_enable\\_t](#)

**17.6.7 static uint32\_t LPTMR\_GetStatusFlags ( LPTMR\_Type \* *base* )** **[inline], [static]**

## Parameters

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

## Returns

The status flags. This is the logical OR of members of the enumeration [lptmr\\_status\\_flags\\_t](#)

### 17.6.8 static void LPTMR\_ClearStatusFlags ( LPTMR\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | LPTMR peripheral base address  |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">lptmr_status_flags_t</a> |

### 17.6.9 static void LPTMR\_SetTimerPeriod ( LPTMR\_Type \* *base*, uint16\_t *ticks* ) [inline], [static]

Timers counts from 0 till it equals the count value set here. The count value is written to the CMR register.

## Note

1. The TCF flag is set with the CNR equals the count provided here and then increments.
2. User can call the utility macros provided in `fsl_common.h` to convert to ticks

## Parameters

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | LPTMR peripheral base address  |
| <i>ticks</i> | Timer period in units of ticks |

### 17.6.10 static uint16\_t LPTMR\_GetCurrentTimerCount ( LPTMR\_Type \* *base* ) [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

## Function Documentation

### Parameters

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

### Returns

Current counter value in ticks

#### 17.6.11 **static void LPTMR\_StartTimer ( LPTMR\_Type \* *base* ) [inline], [static]**

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

### Parameters

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

#### 17.6.12 **static void LPTMR\_StopTimer ( LPTMR\_Type \* *base* ) [inline], [static]**

This function stops the timer counting and resets the timer's counter register

### Parameters

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

## Chapter 18

### PIT: Periodic Interrupt Timer

#### 18.1 Overview

The KSDK provides a driver for the Periodic Interrupt Timer (PIT) of Kinetis devices.

#### 18.2 Function groups

The PIT driver supports operating the module as a time counter.

##### 18.2.1 Initialization and deinitialization

The function [PIT\\_Init\(\)](#) initializes the PIT with specified configurations. The function [PIT\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the PIT operation in debug mode.

The function [PIT\\_SetTimerChainMode\(\)](#) configures the chain mode operation of each PIT channel.

The function [PIT\\_Deinit\(\)](#) disables the PIT timers and disables the module clock.

##### 18.2.2 Timer period Operations

The function [PITR\\_SetTimerPeriod\(\)](#) sets the timer period in units of count. Timers begin counting down from the value set by this function until it reaches 0.

The function [PIT\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. User can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds

##### 18.2.3 Start and Stop timer operations

The function [PIT\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value set earlier via the [PIT\\_SetPeriod\(\)](#) function and starts counting down to 0. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function [PIT\\_StopTimer\(\)](#) stops the timer counting.

## Typical use case

### 18.2.4 Status

Provides functions to get and clear the PIT status.

### 18.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 18.3 Typical use case

### 18.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically.

```
int main(void)
{
    /* Structure of initialize PIT */
    pit_config_t pitConfig;

    /* Initialize and enable LED */
    LED_INIT();

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();

    PIT_GetDefaultConfig(&pitConfig);

    /* Init pit module */
    PIT_Init(PIT, &pitConfig);

    /* Set timer period for channel 0 */
    PIT_SetTimerPeriod(PIT, kPIT_Chnl_0, USEC_TO_COUNT(1000000U,
        PIT_SOURCE_CLOCK));

    /* Enable timer interrupts for channel 0 */
    PIT_EnableInterrupts(PIT, kPIT_Chnl_0,
        kPIT_TimerInterruptEnable);

    /* Enable at the NVIC */
    EnableIRQ(PIT_IRQ_ID);

    /* Start channel 0 */
    PRINTF("\r\nStarting channel No.0 ...");
    PIT_StartTimer(PIT, kPIT_Chnl_0);

    while (true)
    {
        /* Check whether occur interrupt and toggle LED */
        if (true == pitIsrFlag)
        {
            PRINTF("\r\n Channel No.0 interrupt is occurred !");
            LED_TOGGLE();
            pitIsrFlag = false;
        }
    }
}
```



## Data Structures

- struct [pit\\_config\\_t](#)  
*PIT config structure. [More...](#)*

## Enumerations

- enum [pit\\_chnl\\_t](#) {  
  [kPIT\\_Chnl\\_0](#) = 0U,  
  [kPIT\\_Chnl\\_1](#),  
  [kPIT\\_Chnl\\_2](#),  
  [kPIT\\_Chnl\\_3](#) }  
*List of PIT channels.*
- enum [pit\\_interrupt\\_enable\\_t](#) { [kPIT\\_TimerInterruptEnable](#) = [PIT\\_TCTRL\\_TIE\\_MASK](#) }  
*List of PIT interrupts.*
- enum [pit\\_status\\_flags\\_t](#) { [kPIT\\_TimerFlag](#) = [PIT\\_TFLG\\_TIF\\_MASK](#) }  
*List of PIT status flags.*

## Driver version

- #define [FSL\\_PIT\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- void [PIT\\_Init](#) (PIT\_Type \*base, const [pit\\_config\\_t](#) \*config)  
*Un gates the PIT clock, enables the PIT module and configures the peripheral for basic operation.*
- void [PIT\\_Deinit](#) (PIT\_Type \*base)  
*Gate the PIT clock and disable the PIT module.*
- static void [PIT\\_GetDefaultConfig](#) ([pit\\_config\\_t](#) \*config)  
*Fill in the PIT config struct with the default settings.*

## Interrupt Interface

- static void [PIT\\_EnableInterrupts](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, uint32\_t mask)  
*Enables the selected PIT interrupts.*
- static void [PIT\\_DisableInterrupts](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, uint32\_t mask)  
*Disables the selected PIT interrupts.*
- static uint32\_t [PIT\\_GetEnabledInterrupts](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)  
*Gets the enabled PIT interrupts.*

## Status Interface

- static uint32\_t [PIT\\_GetStatusFlags](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)  
*Gets the PIT status flags.*
- static void [PIT\\_ClearStatusFlags](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, uint32\_t mask)  
*Clears the PIT status flags.*

## Enumeration Type Documentation

### Read and Write the timer period

- static void [PIT\\_SetTimerPeriod](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel, uint32\_t count)  
*Sets the timer period in units of count.*
- static uint32\_t [PIT\\_GetCurrentTimerCount](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)  
*Reads the current timer counting value.*

### Timer Start and Stop

- static void [PIT\\_StartTimer](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)  
*Starts the timer counting.*
- static void [PIT\\_StopTimer](#) (PIT\_Type \*base, [pit\\_chnl\\_t](#) channel)  
*Stops the timer counting.*

## 18.4 Data Structure Documentation

### 18.4.1 struct pit\_config\_t

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the [PIT\\_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 [enableRunInDebug](#)  
*true: Timers run in debug mode; false: Timers stop in debug mode*

## 18.5 Enumeration Type Documentation

### 18.5.1 enum pit\_chnl\_t

Note

Actual number of available channels is SoC dependent

Enumerator

*kPIT\_Chnl\_0* PIT channel number 0.  
*kPIT\_Chnl\_1* PIT channel number 1.  
*kPIT\_Chnl\_2* PIT channel number 2.  
*kPIT\_Chnl\_3* PIT channel number 3.

### 18.5.2 enum pit\_interrupt\_enable\_t

Enumerator

***kPIT\_TimerInterruptEnable*** Timer interrupt enable.

### 18.5.3 enum pit\_status\_flags\_t

Enumerator

***kPIT\_TimerFlag*** Timer flag.

## 18.6 Function Documentation

### 18.6.1 void PIT\_Init ( PIT\_Type \* *base*, const pit\_config\_t \* *config* )

Note

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

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | PIT peripheral base address            |
| <i>config</i> | Pointer to user's PIT config structure |

### 18.6.2 void PIT\_Deinit ( PIT\_Type \* *base* )

Parameters

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

### 18.6.3 static void PIT\_GetDefaultConfig ( pit\_config\_t \* *config* ) [inline], [static]

The default values are:

```
* config->enableRunInDebug = false;
*
```

## Function Documentation

### Parameters

|               |   |
|---------------|---|
| <i>config</i> | Pointer to user's PIT config structure. |
|---------------|---|

**18.6.4 static void PIT\_EnableInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]**

### Parameters

|                |   |
|----------------|---|
| <i>base</i>    | PIT peripheral base address   |
| <i>channel</i> | Timer channel number  |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

**18.6.5 static void PIT\_DisableInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]**

### Parameters

|                |  |
|----------------|--|
| <i>base</i>    | PIT peripheral base address  |
| <i>channel</i> | Timer channel number   |
| <i>mask</i>    | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">pit_interrupt_enable_t</a> |

**18.6.6 static uint32\_t PIT\_GetEnabledInterrupts ( PIT\_Type \* *base*, pit\_chnl\_t *channel* ) [inline], [static]**

### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

### Returns

The enabled interrupts. This is the logical OR of members of the enumeration [pit\\_interrupt\\_enable\\_t](#)

**18.6.7** `static uint32_t PIT_GetStatusFlags ( PIT_Type * base, pit_chnl_t channel )`  
`[inline], [static]`

## Function Documentation

### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

### Returns

The status flags. This is the logical OR of members of the enumeration [pit\\_status\\_flags\\_t](#)

**18.6.8 static void PIT\_ClearStatusFlags ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]**

### Parameters

|                |  |
|----------------|--|
| <i>base</i>    | PIT peripheral base address  |
| <i>channel</i> | Timer channel number   |
| <i>mask</i>    | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">pit_status_flags_t</a> |

**18.6.9 static void PIT\_SetTimerPeriod ( PIT\_Type \* *base*, pit\_chnl\_t *channel*, uint32\_t *count* ) [inline], [static]**

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded after the timer expires.

### Note

User can call the utility macros provided in fsl\_common.h to convert to ticks

### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

|              |                                |
|--------------|--------------------------------|
| <i>count</i> | Timer period in units of ticks |
|--------------|--------------------------------|

#### 18.6.10 **static uint32\_t PIT\_GetCurrentTimerCount ( PIT\_Type \* *base*, pit\_chnl\_t *channel* ) [inline], [static]**

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User can call the utility macros provided in fsl\_common.h to convert ticks to usec or msec

Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | PIT peripheral base address |
| <i>channel</i> | Timer channel number        |

Returns

Current timer counting value in ticks

#### 18.6.11 **static void PIT\_StartTimer ( PIT\_Type \* *base*, pit\_chnl\_t *channel* ) [inline], [static]**

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

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

#### 18.6.12 **static void PIT\_StopTimer ( PIT\_Type \* *base*, pit\_chnl\_t *channel* ) [inline], [static]**

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT\_DRV\_StartTimer.

## Function Documentation

### Parameters

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



## Chapter 19

# PMC: Power Management Controller

### 19.1 Overview

The KSDK provides a Peripheral driver for the Power Management Controller (PMC) module of Kinetis devices. The PMC module contains internal voltage regulator, power on reset, low-voltage detect system, and high-voltage detect system.

### Data Structures

- struct [pmc\\_low\\_volt\\_detect\\_config\\_t](#)  
*Low-Voltage Detect Configuration Structure. [More...](#)*
- struct [pmc\\_low\\_volt\\_warning\\_config\\_t](#)  
*Low-Voltage Warning Configuration Structure. [More...](#)*

### Driver version

- #define [FSL\\_PMC\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 0))  
*PMC driver version.*

### Power Management Controller Control APIs

- void [PMC\\_ConfigureLowVoltDetect](#) (PMC\_Type \*base, const [pmc\\_low\\_volt\\_detect\\_config\\_t](#) \*config)  
*Configure the low-voltage detect setting.*
- static bool [PMC\\_GetLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Get Low-Voltage Detect Flag status.*
- static void [PMC\\_ClearLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Acknowledge to clear the Low-voltage Detect flag.*
- void [PMC\\_ConfigureLowVoltWarning](#) (PMC\_Type \*base, const [pmc\\_low\\_volt\\_warning\\_config\\_t](#) \*config)  
*Configure the low-voltage warning setting.*
- static bool [PMC\\_GetLowVoltWarningFlag](#) (PMC\_Type \*base)  
*Get Low-Voltage Warning Flag status.*
- static void [PMC\\_ClearLowVoltWarningFlag](#) (PMC\_Type \*base)  
*Acknowledge to Low-Voltage Warning flag.*

### 19.2 Data Structure Documentation

#### 19.2.1 struct pmc\_low\_volt\_detect\_config\_t

#### Data Fields

- bool [enableInt](#)

## Function Documentation

- *Enable interrupt when low-voltage detect.*  
• bool [enableReset](#)  
*Enable system reset when low-voltage detect.*

### 19.2.2 struct pmc\_low\_volt\_warning\_config\_t

#### Data Fields

- bool [enableInt](#)  
*Enable interrupt when low-voltage warning.*

## 19.3 Macro Definition Documentation

### 19.3.1 #define FSL\_PMC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

Version 2.0.0.

## 19.4 Function Documentation

### 19.4.1 void PMC\_ConfigureLowVoltDetect ( PMC\_Type \* *base*, const pmc\_low\_volt\_detect\_config\_t \* *config* )

This function configures the low-voltage detect setting, including the trip point voltage setting, enable interrupt or not, enable system reset or not.

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | PMC peripheral base address.                |
| <i>config</i> | Low-Voltage detect configuration structure. |

### 19.4.2 static bool PMC\_GetLowVoltDetectFlag ( PMC\_Type \* *base* ) [inline], [static]

This function reads the current LVDF status. If it returns 1, a low-voltage event is detected.

Parameters

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

## Returns

Current low-voltage detect flag

- true: Low-voltage detected
- false: Low-voltage not detected

### 19.4.3 static void PMC\_ClearLowVoltDetectFlag ( PMC\_Type \* *base* ) [inline], [static]

This function acknowledges the low-voltage detection errors (write 1 to clear LVDF).

## Parameters

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

### 19.4.4 void PMC\_ConfigureLowVoltWarning ( PMC\_Type \* *base*, const pmc\_low\_volt\_warning\_config\_t \* *config* )

This function configures the low-voltage warning setting, including the trip point voltage setting and enable interrupt or not.

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | PMC peripheral base address.                 |
| <i>config</i> | Low-Voltage warning configuration structure. |

### 19.4.5 static bool PMC\_GetLowVoltWarningFlag ( PMC\_Type \* *base* ) [inline], [static]

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

## Parameters

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

## Returns

Current LVWF status

- true: Low-Voltage Warning Flag is set.
- false: the Low-Voltage Warning does not happen.

---

## Function Documentation

**19.4.6 static void PMC\_ClearLowVoltWarningFlag ( PMC\_Type \* *base* )**  
**[inline], [static]**

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

## Parameters

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



## Chapter 20

# PORT: Port Control and Interrupts

### 20.1 Overview

The KSDK provides a driver for the Port Control and Interrupts (PORT) module of Kinetis devices.

### 20.2 Typical configuration use case

#### 20.2.1 Input PORT configuration

```
/* Input pin PORT configuration */
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
/* Sets the configuration */
PORT_SetPinConfig(PORTA, 4, &config);
```

#### 20.2.2 I2C PORT Configuration

```
/* I2C pin PORT configuration */
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainEnable,
    kPORT_LowDriveStrength,
    kPORT_MuxAlt5,
    kPORT_UnLockRegister,
};
PORT_SetPinConfig(PORTE, 24u, &config);
PORT_SetPinConfig(PORTE, 25u, &config);
```

### Data Structures

- struct `port_pin_config_t`  
*PORT pin configuration structure. [More...](#)*

### Enumerations

- enum `_port_pull` {  
    `kPORT_PullDisable` = 0U,  
    `kPORT_PullDown` = 2U,  
    `kPORT_PullUp` = 3U }

## Typical configuration use case

- Internal resistor pull feature selection.*
  - enum `_port_slew_rate` {  
    `kPORT_FastSlewRate` = 0U,  
    `kPORT_SlowSlewRate` = 1U }
- Slew rate selection.*
  - enum `_port_passive_filter_enable` {  
    `kPORT_PassiveFilterDisable` = 0U,  
    `kPORT_PassiveFilterEnable` = 1U }
- Passive filter feature enable/disable.*
  - enum `_port_drive_strength` {  
    `kPORT_LowDriveStrength` = 0U,  
    `kPORT_HighDriveStrength` = 1U }
- Configures the drive strength.*
  - enum `port_mux_t` {  
    `kPORT_PinDisabledOrAnalog` = 0U,  
    `kPORT_MuxAsGpio` = 1U,  
    `kPORT_MuxAlt2` = 2U,  
    `kPORT_MuxAlt3` = 3U,  
    `kPORT_MuxAlt4` = 4U,  
    `kPORT_MuxAlt5` = 5U,  
    `kPORT_MuxAlt6` = 6U,  
    `kPORT_MuxAlt7` = 7U }
- Pin mux selection.*
  - enum `port_interrupt_t` {  
    `kPORT_InterruptOrDMADisabled` = 0x0U,  
    `kPORT_InterruptLogicZero` = 0x8U,  
    `kPORT_InterruptRisingEdge` = 0x9U,  
    `kPORT_InterruptFallingEdge` = 0xAU,  
    `kPORT_InterruptEitherEdge` = 0xBU,  
    `kPORT_InterruptLogicOne` = 0xCU }
- Configures the interrupt generation condition.*

## Driver version

- #define `FSL_PORT_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 1))  
    Version 2.0.1.

## Configuration

- static void `PORT_SetPinConfig` (`PORT_Type` \*base, uint32\_t pin, const `port_pin_config_t` \*config)  
    Sets the port PCR register.
- static void `PORT_SetMultiplePinsConfig` (`PORT_Type` \*base, uint32\_t mask, const `port_pin_config_t` \*config)  
    Sets the port PCR register for multiple pins.
- static void `PORT_SetPinMux` (`PORT_Type` \*base, uint32\_t pin, `port_mux_t` mux)  
    Configures the pin muxing.



## Interrupt

- static void [PORT\\_SetPinInterruptConfig](#) (PORT\_Type \*base, uint32\_t pin, [port\\_interrupt\\_t](#) config)  
*Configures the port pin interrupt/DMA request.*
- static uint32\_t [PORT\\_GetPinsInterruptFlags](#) (PORT\_Type \*base)  
*Reads the whole port status flag.*
- static void [PORT\\_ClearPinsInterruptFlags](#) (PORT\_Type \*base, uint32\_t mask)  
*Clears the multiple pin interrupt status flag.*

## 20.3 Data Structure Documentation

### 20.3.1 struct port\_pin\_config\_t

#### Data Fields

- uint16\_t [pullSelect](#): 2  
*No-pull/pull-down/pull-up select.*
- uint16\_t [slewRate](#): 1  
*Fast/slow slew rate Configure.*
- uint16\_t [passiveFilterEnable](#): 1  
*Passive filter enable/disable.*
- uint16\_t [driveStrength](#): 1  
*Fast/slow drive strength configure.*
- uint16\_t [mux](#): 3  
*Pin mux Configure.*

## 20.4 Macro Definition Documentation

### 20.4.1 #define FSL\_PORT\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 20.5 Enumeration Type Documentation

### 20.5.1 enum \_port\_pull

Enumerator

***kPORT\_PullDisable*** Internal pull-up/down resistor is disabled.  
***kPORT\_PullDown*** Internal pull-down resistor is enabled.  
***kPORT\_PullUp*** Internal pull-up resistor is enabled.

### 20.5.2 enum \_port\_slew\_rate

Enumerator

***kPORT\_FastSlewRate*** Fast slew rate is configured.  
***kPORT\_SlowSlewRate*** Slow slew rate is configured.

## Function Documentation

### 20.5.3 enum \_port\_passive\_filter\_enable

Enumerator

***kPORT\_PassiveFilterDisable*** Fast slew rate is configured.

***kPORT\_PassiveFilterEnable*** Slow slew rate is configured.

### 20.5.4 enum \_port\_drive\_strength

Enumerator

***kPORT\_LowDriveStrength*** Low-drive strength is configured.

***kPORT\_HighDriveStrength*** High-drive strength is configured.

### 20.5.5 enum port\_mux\_t

Enumerator

***kPORT\_PinDisabledOrAnalog*** Corresponding pin is disabled, but is used as an analog pin.

***kPORT\_MuxAsGpio*** Corresponding pin is configured as GPIO.

***kPORT\_MuxAlt2*** Chip-specific.

***kPORT\_MuxAlt3*** Chip-specific.

***kPORT\_MuxAlt4*** Chip-specific.

***kPORT\_MuxAlt5*** Chip-specific.

***kPORT\_MuxAlt6*** Chip-specific.

***kPORT\_MuxAlt7*** Chip-specific.

### 20.5.6 enum port\_interrupt\_t

Enumerator

***kPORT\_InterruptOrDMADisabled*** Interrupt/DMA request is disabled.

***kPORT\_InterruptLogicZero*** Interrupt when logic zero.

***kPORT\_InterruptRisingEdge*** Interrupt on rising edge.

***kPORT\_InterruptFallingEdge*** Interrupt on falling edge.

***kPORT\_InterruptEitherEdge*** Interrupt on either edge.

***kPORT\_InterruptLogicOne*** Interrupt when logic one.

## 20.6 Function Documentation

### 20.6.1 static void PORT\_SetPinConfig ( PORT\_Type \* *base*, uint32\_t *pin*, const port\_pin\_config\_t \* *config* ) [inline], [static]

This is an example to define an input pin or output pin PCR configuration:

```

* // Define a digital input pin PCR configuration
* port_pin_config_t config = {
*     kPORT_PullUp,
*     kPORT_FastSlewRate,
*     kPORT_PassiveFilterDisable,
*     kPORT_OpenDrainDisable,
*     kPORT_LowDriveStrength,
*     kPORT_MuxAsGpio,
*     kPORT_UnLockRegister,
* };
*

```

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | PORT peripheral base pointer.              |
| <i>pin</i>    | PORT pin number.                           |
| <i>config</i> | PORT PCR register configuration structure. |

### 20.6.2 static void PORT\_SetMultiplePinsConfig ( PORT\_Type \* *base*, uint32\_t *mask*, const port\_pin\_config\_t \* *config* ) [inline], [static]

This is an example to define input pins or output pins PCR configuration:

```

* // Define a digital input pin PCR configuration
* port_pin_config_t config = {
*     kPORT_PullUp ,
*     kPORT_PullEnable,
*     kPORT_FastSlewRate,
*     kPORT_PassiveFilterDisable,
*     kPORT_OpenDrainDisable,
*     kPORT_LowDriveStrength,
*     kPORT_MuxAsGpio,
*     kPORT_UnlockRegister,
* };
*

```

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | PORT peripheral base pointer.              |
| <i>mask</i>   | PORT pin number macro.                     |
| <i>config</i> | PORT PCR register configuration structure. |

### 20.6.3 static void PORT\_SetPinMux ( PORT\_Type \* *base*, uint32\_t *pin*, port\_mux\_t *mux* ) [inline], [static]

## Function Documentation

### Parameters

|             |  |
|-------------|--|
| <i>base</i> | PORT peripheral base pointer.  |
| <i>pin</i>  | PORT pin number.   |
| <i>mux</i>  | <p>pin muxing slot selection.</p> <ul style="list-style-type: none"><li>• <a href="#">kPORT_PinDisabledOrAnalog</a>: Pin disabled or work in analog function.</li><li>• <a href="#">kPORT_MuxAsGpio</a> : Set as GPIO.</li><li>• <a href="#">kPORT_MuxAlt2</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt3</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt4</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt5</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt6</a> : chip-specific.</li><li>• <a href="#">kPORT_MuxAlt7</a> : chip-specific. : This function is NOT recommended to use together with the <code>PORT_SetPinsConfig</code>, because the <code>PORT_SetPinsConfig</code> need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : <code>kPORT_PinDisabledOrAnalog</code>). This function is recommended to use to reset the pin mux</li></ul> |

#### 20.6.4 static void PORT\_SetPinInterruptConfig ( PORT\_Type \* *base*, uint32\_t *pin*, port\_interrupt\_t *config* ) [inline], [static]

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | PORT peripheral base pointer.  |
| <i>pin</i>    | PORT pin number.   |
| <i>config</i> | PORT pin interrupt configuration. <ul style="list-style-type: none"> <li>• <a href="#">kPORT_InterruptOrDMADisabled</a>: Interrupt/DMA request disabled.</li> <li>• #kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit).</li> <li>• #kPORT_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit).</li> <li>• #kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit).</li> <li>• #kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit).</li> <li>• #kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit).</li> <li>• #kPORT_FlagEitherEdge : Flag sets on either edge(if the Flag states exit).</li> <li>• <a href="#">kPORT_InterruptLogicZero</a> : Interrupt when logic zero.</li> <li>• <a href="#">kPORT_InterruptRisingEdge</a> : Interrupt on rising edge.</li> <li>• <a href="#">kPORT_InterruptFallingEdge</a>: Interrupt on falling edge.</li> <li>• <a href="#">kPORT_InterruptEitherEdge</a> : Interrupt on either edge.</li> <li>• <a href="#">kPORT_InterruptLogicOne</a> : Interrupt when logic one.</li> <li>• #kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit).</li> <li>• #kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).</li> </ul> |

#### 20.6.5 static uint32\_t PORT\_GetPinsInterruptFlags ( PORT\_Type \* *base* ) [inline], [static]

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

## Function Documentation

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | PORT peripheral base pointer. |
|-------------|-------------------------------|

### Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

### 20.6.6 static void PORT\_ClearPinsInterruptFlags ( PORT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | PORT peripheral base pointer. |
| <i>mask</i> | PORT pin number macro.        |

## Chapter 21

# RCM: Reset Control Module Driver

### 21.1 Overview

The KSDK provides a Peripheral driver for the Reset Control Module (RCM) module of Kinetis devices.

### Data Structures

- struct [rcm\\_reset\\_pin\\_filter\\_config\\_t](#)  
*Reset pin filter configuration. [More...](#)*

### Enumerations

- enum [rcm\\_reset\\_source\\_t](#) {  
    [kRCM\\_SourceLvd](#) = RCM\_SRS0\_LVD\_MASK,  
    [kRCM\\_SourceWdog](#) = RCM\_SRS0\_WDOG\_MASK,  
    [kRCM\\_SourcePin](#) = RCM\_SRS0\_PIN\_MASK,  
    [kRCM\\_SourcePor](#) = RCM\_SRS0\_POR\_MASK,  
    [kRCM\\_SourceLockup](#) = RCM\_SRS1\_LOCKUP\_MASK << 8U,  
    [kRCM\\_SourceSw](#) = RCM\_SRS1\_SW\_MASK << 8U,  
    [kRCM\\_SourceSackerr](#) = RCM\_SRS1\_SACKERR\_MASK << 8U }  
    *System Reset Source Name definitions.*
- enum [rcm\\_run\\_wait\\_filter\\_mode\\_t](#) {  
    [kRCM\\_FilterDisable](#) = 0U,  
    [kRCM\\_FilterBusClock](#) = 1U,  
    [kRCM\\_FilterLpoClock](#) = 2U }  
    *Reset pin filter select in Run and Wait modes.*

### Driver version

- #define [FSL\\_RCM\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 1))  
    *RCM driver version 2.0.1.*

### Reset Control Module APIs

- static uint32\_t [RCM\\_GetPreviousResetSources](#) (RCM\_Type \*base)  
    *Gets the reset source status which caused a previous reset.*
- void [RCM\\_ConfigureResetPinFilter](#) (RCM\_Type \*base, const [rcm\\_reset\\_pin\\_filter\\_config\\_t](#) \*config)  
    *Configures the reset pin filter.*

## Enumeration Type Documentation

### 21.2 Data Structure Documentation

#### 21.2.1 struct rcm\_reset\_pin\_filter\_config\_t

##### Data Fields

- bool [enableFilterInStop](#)  
*Reset pin filter select in stop mode.*
- [rcm\\_run\\_wait\\_filter\\_mode\\_t](#) [filterInRunWait](#)  
*Reset pin filter in run/wait mode.*
- uint8\_t [busClockFilterCount](#)  
*Reset pin bus clock filter width.*

##### 21.2.1.0.0.33 Field Documentation

21.2.1.0.0.33.1 bool rcm\_reset\_pin\_filter\_config\_t::enableFilterInStop

21.2.1.0.0.33.2 rcm\_run\_wait\_filter\_mode\_t rcm\_reset\_pin\_filter\_config\_t::filterInRunWait

21.2.1.0.0.33.3 uint8\_t rcm\_reset\_pin\_filter\_config\_t::busClockFilterCount

### 21.3 Macro Definition Documentation

21.3.1 #define FSL\_RCM\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 21.4 Enumeration Type Documentation

#### 21.4.1 enum rcm\_reset\_source\_t

Enumerator

***kRCM\_SourceLvd*** Low-voltage detect reset.  
***kRCM\_SourceWdog*** Watchdog reset.  
***kRCM\_SourcePin*** External pin reset.  
***kRCM\_SourcePor*** Power on reset.  
***kRCM\_SourceLockup*** Core lock up reset.  
***kRCM\_SourceSw*** Software reset.  
***kRCM\_SourceSackerr*** Parameter could get all reset flags.

#### 21.4.2 enum rcm\_run\_wait\_filter\_mode\_t

Enumerator

***kRCM\_FilterDisable*** All filtering disabled.  
***kRCM\_FilterBusClock*** Bus clock filter enabled.  
***kRCM\_FilterLpoClock*** LPO clock filter enabled.



## 21.5 Function Documentation

### 21.5.1 static uint32\_t RCM\_GetPreviousResetSources ( RCM\_Type \* *base* ) [inline], [static]

This function gets the current reset source status. Use source masks defined in the `rcm_reset_source_t` to get the desired source status.

Example:

```
uint32_t resetStatus;

// To get all reset source statuses.
resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceAll;

// To test whether the MCU is reset using Watchdog.
resetStatus = RCM_GetPreviousResetSources(RCM) &
              kRCM_SourceWdog;

// To test multiple reset sources.
resetStatus = RCM_GetPreviousResetSources(RCM) & (
              kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

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

Returns

All reset source status bit map.

### 21.5.2 void RCM\_ConfigureResetPinFilter ( RCM\_Type \* *base*, const rcm\_reset\_pin\_filter\_config\_t \* *config* )

This function sets the reset pin filter including the filter source, filter width, and so on.

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | RCM peripheral base address.            |
| <i>config</i> | Pointer to the configuration structure. |



## Chapter 22

### RTC: Real Time Clock

#### 22.1 Overview

The KSDK provides a driver for the Real Time Clock (RTC) of Kinetis devices.

#### 22.2 Function groups

The RTC driver supports operating the module as a time counter.

##### 22.2.1 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.

##### 22.2.2 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.

```
typedef struct _rtc_datetime
{
    uint16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
} rtc_datetime_t;
```

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.

##### 22.2.3 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.

## Typical use case

### 22.2.4 Start & Stop timer

The function `RTC_StartTimer()` starts the RTC time counter.

The function `RTC_StopTimer()` stops the RTC time counter.

### 22.2.5 Status

Provides functions to get and clear the RTC status.

### 22.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

### 22.2.7 RTC Oscillator

Some SoC's allow control of the RTC oscillator through the RTC module.

The function `RTC_SetOscCapLoad()` allows the user to modify the capacitor load configuration of the RTC oscillator.

### 22.2.8 Monotonic Counter

Some SoC's have a 64-bit Monotonic counter available in the RTC module.

The function `RTC_SetMonotonicCounter()` writes a 64-bit to the counter.

The function `RTC_GetMonotonicCounter()` reads the monotonic counter and returns the 64-bit counter value to the user.

The function `RTC_IncrementMonotonicCounter()` increments the Monotonic Counter by one.

## 22.3 Typical use case

### 22.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm.

```
int main(void)
{
    uint32_t sec;
    uint32_t currSeconds;
    rtc_datetime_t date;
    rtc_config_t rtcConfig;

    /* Board pin, clock, debug console init */
```

```

BOARD_InitHardware();
/* Init RTC */
RTC_GetDefaultConfig(&rtcConfig);
RTC_Init(RTC, &rtcConfig);
/* Select RTC clock source */
BOARD_SetRtcClockSource();

PRINTF("RTC example: set up time to wake up an alarm\r\n");

/* Set a start date time and start RT */
date.year = 2014U;
date.month = 12U;
date.day = 25U;
date.hour = 19U;
date.minute = 0;
date.second = 0;

/* RTC time counter has to be stopped before setting the date & time in the TSR register */
RTC_StopTimer(RTC);

/* Set RTC time to default */
RTC_SetDatetime(RTC, &date);

/* Enable RTC alarm interrupt */
RTC_EnableInterrupts(RTC, kRTC_AlarmInterruptEnable);

/* Enable at the NVIC */
EnableIRQ(RTC_IRQn);

/* Start the RTC time counter */
RTC_StartTimer(RTC);

/* This loop will set the RTC alarm */
while (1)
{
    busyWait = true;
    /* Get date time */
    RTC_GetDatetime(RTC, &date);

    /* print default time */
    PRINTF("Current datetime: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n", date.
year, date.month, date.day, date.hour,
        date.minute, date.second);

    /* Get alarm time from user */
    sec = 0;
    PRINTF("Please input the number of second to wait for alarm \r\n");
    PRINTF("The second must be positive value\r\n");
    while (sec < 1)
    {
        SCANF("%d", &sec);
    }

    /* Read the RTC seconds register to get current time in seconds */
    currSeconds = RTC->TSR;

    /* Add alarm seconds to current time */
    currSeconds += sec;

    /* Set alarm time in seconds */
    RTC->TAR = currSeconds;

    /* Get alarm time */
    RTC_GetAlarm(RTC, &date);

    /* Print alarm time */
    PRINTF("Alarm will occur at: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n", date.
year, date.month, date.day,

```

## Typical use case

```
    date.hour, date.minute, date.second);

    /* Wait until alarm occurs */
    while (busyWait)
    {
    }

    PRINTF("\r\n Alarm occurs !!!! ");
}
}
```

## Data Structures

- struct [rtc\\_datetime\\_t](#)  
*Structure is used to hold the date and time. [More...](#)*
- struct [rtc\\_config\\_t](#)  
*RTC config structure. [More...](#)*

## Enumerations

- enum [rtc\\_interrupt\\_enable\\_t](#) {  
    [kRTC\\_TimeInvalidInterruptEnable](#) = RTC\_IER\_TIIE\_MASK,  
    [kRTC\\_TimeOverflowInterruptEnable](#) = RTC\_IER\_TOIE\_MASK,  
    [kRTC\\_AlarmInterruptEnable](#) = RTC\_IER\_TAIE\_MASK,  
    [kRTC\\_SecondsInterruptEnable](#) = RTC\_IER\_TSIE\_MASK }  
*List of RTC interrupts.*
- enum [rtc\\_status\\_flags\\_t](#) {  
    [kRTC\\_TimeInvalidFlag](#) = RTC\_SR\_TIF\_MASK,  
    [kRTC\\_TimeOverflowFlag](#) = RTC\_SR\_TOF\_MASK,  
    [kRTC\\_AlarmFlag](#) = RTC\_SR\_TAF\_MASK }  
*List of RTC flags.*

## Functions

- static void [RTC\\_Reset](#) (RTC\_Type \*base)  
*Performs a software reset on the RTC module.*

## Driver version

- #define [FSL\\_RTC\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- void [RTC\\_Init](#) (RTC\_Type \*base, const [rtc\\_config\\_t](#) \*config)  
*Ungates the RTC clock and configures the peripheral for basic operation.*
- static void [RTC\\_Deinit](#) (RTC\_Type \*base)  
*Stop the timer and gate the RTC clock.*
- void [RTC\\_GetDefaultConfig](#) ([rtc\\_config\\_t](#) \*config)  
*Fill in the RTC config struct with the default settings.*

## Current Time & Alarm

- status\_t [RTC\\_SetDatetime](#) (RTC\_Type \*base, const [rtc\\_datetime\\_t](#) \*datetime)  
*Sets the RTC date and time according to the given time structure.*
- void [RTC\\_GetDatetime](#) (RTC\_Type \*base, [rtc\\_datetime\\_t](#) \*datetime)  
*Gets the RTC time and stores it in the given time structure.*
- status\_t [RTC\\_SetAlarm](#) (RTC\_Type \*base, const [rtc\\_datetime\\_t](#) \*alarmTime)  
*Sets the RTC alarm time.*
- void [RTC\\_GetAlarm](#) (RTC\_Type \*base, [rtc\\_datetime\\_t](#) \*datetime)  
*Returns the RTC alarm time.*

## Interrupt Interface

- 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)  
*Gets the enabled RTC interrupts.*

## Status Interface

- static uint32\_t [RTC\\_GetStatusFlags](#) (RTC\_Type \*base)  
*Gets the RTC status flags.*
- void [RTC\\_ClearStatusFlags](#) (RTC\_Type \*base, uint32\_t mask)  
*Clears the RTC status flags.*

## Timer Start and Stop

- static void [RTC\\_StartTimer](#) (RTC\_Type \*base)  
*Starts the RTC time counter.*
- static void [RTC\\_StopTimer](#) (RTC\_Type \*base)  
*Stops the RTC time counter.*

## 22.4 Data Structure Documentation

### 22.4.1 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.*

## Enumeration Type Documentation

- uint8\_t [second](#)  
*Range from 0 to 59.*

### 22.4.1.0.0.34 Field Documentation

22.4.1.0.0.34.1 uint16\_t rtc\_datetime\_t::year

22.4.1.0.0.34.2 uint8\_t rtc\_datetime\_t::month

22.4.1.0.0.34.3 uint8\_t rtc\_datetime\_t::day

22.4.1.0.0.34.4 uint8\_t rtc\_datetime\_t::hour

22.4.1.0.0.34.5 uint8\_t rtc\_datetime\_t::minute

22.4.1.0.0.34.6 uint8\_t rtc\_datetime\_t::second

### 22.4.2 struct rtc\_config\_t

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the [RTC\\_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 [wakeupSelect](#)  
*true: Wakeup pin outputs the 32 KHz clock; false: Wakeup pin used to wakeup the chip*
- bool [updateMode](#)  
*true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked*
- bool [supervisorAccess](#)  
*true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported*
- uint32\_t [compensationInterval](#)  
*Compensation interval that is written to the CIR field in RTC TCR Register.*
- uint32\_t [compensationTime](#)  
*Compensation time that is written to the TCR field in RTC TCR Register.*

## 22.5 Enumeration Type Documentation

### 22.5.1 enum rtc\_interrupt\_enable\_t

Enumerator

***kRTC\_TimeInvalidInterruptEnable*** Time invalid interrupt.  
***kRTC\_TimeOverflowInterruptEnable*** Time overflow interrupt.  
***kRTC\_AlarmInterruptEnable*** Alarm interrupt.



***kRTC\_SecondsInterruptEnable*** Seconds interrupt.

## 22.5.2 enum rtc\_status\_flags\_t

Enumerator

***kRTC\_TimeInvalidFlag*** Time invalid flag.

***kRTC\_TimeOverflowFlag*** Time overflow flag.

***kRTC\_AlarmFlag*** Alarm flag.

## 22.6 Function Documentation

### 22.6.1 void RTC\_Init ( RTC\_Type \* *base*, const rtc\_config\_t \* *config* )

This function will issue a software reset if the timer invalid flag is set.

Note

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

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | RTC peripheral base address             |
| <i>config</i> | Pointer to user's RTC config structure. |

### 22.6.2 static void RTC\_Deinit ( RTC\_Type \* *base* ) [inline], [static]

Parameters

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

### 22.6.3 void RTC\_GetDefaultConfig ( rtc\_config\_t \* *config* )

The default values are:

```
* config->wakeupSelect = false;
* config->updateMode = false;
* config->supervisorAccess = false;
* config->compensationInterval = 0;
* config->compensationTime = 0;
*
```

## Function Documentation

### Parameters

|               |   |
|---------------|---|
| <i>config</i> | Pointer to user's RTC config structure. |
|---------------|---|

### 22.6.4 **status\_t RTC\_SetDatetime ( RTC\_Type \* *base*, const rtc\_datetime\_t \* *datetime* )**

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

### Parameters

|                 |  |
|-----------------|--|
| <i>base</i>     | RTC peripheral base address  |
| <i>datetime</i> | Pointer to structure where the date and time details to set are stored |

### Returns

kStatus\_Success: Success in setting the time and starting the RTC  
kStatus\_InvalidArgument: Error because the datetime format is incorrect

### 22.6.5 **void RTC\_GetDatetime ( RTC\_Type \* *base*, rtc\_datetime\_t \* *datetime* )**

### Parameters

|                 |  |
|-----------------|--|
| <i>base</i>     | RTC peripheral base address                                      |
| <i>datetime</i> | Pointer to structure where the date and time details are stored. |

### 22.6.6 **status\_t RTC\_SetAlarm ( RTC\_Type \* *base*, const rtc\_datetime\_t \* *alarmTime* )**

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

### Parameters

---

|                  |  |
|------------------|--|
| <i>base</i>      | RTC peripheral base address                          |
| <i>alarmTime</i> | Pointer to structure where the alarm time is stored. |

Returns

kStatus\_Success: success in setting the RTC alarm  
 kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
 kStatus\_Fail: Error because the alarm time has already passed

### 22.6.7 void RTC\_GetAlarm ( RTC\_Type \* *base*, rtc\_datetime\_t \* *datetime* )

Parameters

|                 |  |
|-----------------|--|
| <i>base</i>     | RTC peripheral base address  |
| <i>datetime</i> | Pointer to structure where the alarm date and time details are stored. |

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

Parameters

|             |   |
|-------------|---|
| <i>base</i> | RTC peripheral base address   |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">rtc_interrupt_enable_t</a> |

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

Parameters

|             |   |
|-------------|---|
| <i>base</i> | RTC peripheral base address   |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">rtc_interrupt_enable_t</a> |

### 22.6.10 static uint32\_t RTC\_GetEnabledInterrupts ( RTC\_Type \* *base* ) [inline], [static]

## Function Documentation

### Parameters

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

### Returns

The enabled interrupts. This is the logical OR of members of the enumeration [rtc\\_interrupt\\_enable\\_t](#)

### 22.6.11 static uint32\_t RTC\_GetStatusFlags ( RTC\_Type \* *base* ) [inline], [static]

### Parameters

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

### Returns

The status flags. This is the logical OR of members of the enumeration [rtc\\_status\\_flags\\_t](#)

### 22.6.12 void RTC\_ClearStatusFlags ( RTC\_Type \* *base*, uint32\_t *mask* )

### Parameters

|             |  |
|-------------|--|
| <i>base</i> | RTC peripheral base address  |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">rtc_status_flags_t</a> |

### 22.6.13 static void RTC\_StartTimer ( RTC\_Type \* *base* ) [inline], [static]

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

### Parameters

---

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

#### 22.6.14 static void RTC\_StopTimer ( RTC\_Type \* *base* ) [inline], [static]

RTC's seconds register can be written to only when the timer is stopped.

Parameters

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

#### 22.6.15 static void RTC\_Reset ( RTC\_Type \* *base* ) [inline], [static]

This resets all RTC registers except for the SWR bit and the RTC\_WAR and RTC\_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

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



## Chapter 23

# SAI: Serial Audio Interface

### 23.1 Overview

The KSDK provides a peripheral driver for the Serial Audio Interface (SAI) module of Kinetis devices.

SAI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low-level APIs. Functional APIs can be used for SAI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SAI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SAI functional operation groups provide the functional API set.

Transactional APIs 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 `sai_handle_t` as the first parameter. Initialize the handle by calling the [SAI\\_TransferTxCreateHandle\(\)](#) or [SAI\\_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SAI\\_TransferSendNonBlocking\(\)](#) and [SAI\\_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SAI_TxIdle` and `kStatus_SAI_RxIdle` status.

### 23.2 Typical use case

#### 23.2.1 SAI Send/Receive using an interrupt method

```
sai_handle_t g_saiTxHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
volatile bool rxFinished;
const uint8_t sendData[] = {.....};

void SAI_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_SAI_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    SAI_TxGetDefaultConfig(&user_config);
```

## Typical use case

```
SAI_TxInit(SAI0, &user_config);
SAI_TransferTxCreateHandle(SAI0, &g_saiHandle, SAI_UserCallback, NULL);

//Configure sai format
SAI_TransferTxSetTransferFormat(SAI0, &g_saiHandle, mclkSource, mclk);

// Prepare to send.
sendXfer.data = sendData
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Send out.
SAI_TransferSendNonBlocking(SAI0, &g_saiHandle, &sendXfer);

// Wait send finished.
while (!txFinished)
{
}

// ...
}
```

### 23.2.2 SAI Send/receive using a DMA method

```
sai_handle_t g_saiHandle;
dma_handle_t g_saiTxDmaHandle;
dma_handle_t g_saiRxDmaHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
uint8_t sendData[] = ...;

void SAI_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_SAI_TxIdle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    //...

    SAI_TxGetDefaultConfig(&user_config);
    SAI_TxInit(SAI0, &user_config);

    // Sets up the DMA.
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, SAI_TX_DMA_CHANNEL, SAI_TX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, SAI_TX_DMA_CHANNEL);

    DMA_Init(DMA0);

    /* Creates the DMA handle. */
    DMA_CreateHandle(&g_saiTxDmaHandle, DMA0, SAI_TX_DMA_CHANNEL);

    SAI_TransferTxCreateHandleDMA(SAI0, &g_saiTxDmaHandle, SAI_UserCallback,
        NULL);

    // Prepares to send.
    sendXfer.data = sendData
```



```

sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
SAI_TransferSendDMA(&g_saiHandle, &sendXfer);

// Waits for send to complete.
while (!txFinished)
{
}

// ...
}

```

## Modules

- [SAI DMA Driver](#)
- [SAI eDMA Driver](#)

## Data Structures

- struct [sai\\_config\\_t](#)  
*SAI user configuration structure. [More...](#)*
- struct [sai\\_transfer\\_format\\_t](#)  
*sai transfer format [More...](#)*
- struct [sai\\_transfer\\_t](#)  
*SAI transfer structure. [More...](#)*
- struct [sai\\_handle\\_t](#)  
*SAI handle structure. [More...](#)*

## Macros

- #define [SAI\\_XFER\\_QUEUE\\_SIZE](#) (4)  
*SAI transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(\* [sai\\_transfer\\_callback\\_t](#))(I2S\_Type \*base, sai\_handle\_t \*handle, status\_t status, void \*userData)  
*SAI transfer callback prototype.*

## Enumerations

- enum [\\_sai\\_status\\_t](#) {  
[kStatus\\_SAI\\_TxBusy](#) = MAKE\_STATUS(kStatusGroup\_SAI, 0),  
[kStatus\\_SAI\\_RxBusy](#) = MAKE\_STATUS(kStatusGroup\_SAI, 1),  
[kStatus\\_SAI\\_TxError](#) = MAKE\_STATUS(kStatusGroup\_SAI, 2),  
[kStatus\\_SAI\\_RxError](#) = MAKE\_STATUS(kStatusGroup\_SAI, 3),  
[kStatus\\_SAI\\_QueueFull](#) = MAKE\_STATUS(kStatusGroup\_SAI, 4),  
[kStatus\\_SAI\\_TxIdle](#) = MAKE\_STATUS(kStatusGroup\_SAI, 5),  
[kStatus\\_SAI\\_RxIdle](#) = MAKE\_STATUS(kStatusGroup\_SAI, 6) }  
*SAI return status.*

## Typical use case

- enum `sai_protocol_t` {  
    `kSAI_BusLeftJustified` = 0x0U,  
    `kSAI_BusRightJustified`,  
    `kSAI_BusI2S`,  
    `kSAI_BusPCMA`,  
    `kSAI_BusPCMB` }  
    *Define the SAI bus type.*
- enum `sai_master_slave_t` {  
    `kSAI_Master` = 0x0U,  
    `kSAI_Slave` = 0x1U }  
    *Master or slave mode.*
- enum `sai_mono_stereo_t` {  
    `kSAI_Stereo` = 0x0U,  
    `kSAI_MonoLeft`,  
    `kSAI_MonoRight` }  
    *Mono or stereo audio format.*
- enum `sai_sync_mode_t` {  
    `kSAI_ModeAsync` = 0x0U,  
    `kSAI_ModeSync`,  
    `kSAI_ModeSyncWithOtherTx`,  
    `kSAI_ModeSyncWithOtherRx` }  
    *Synchronous or asynchronous mode.*
- enum `sai_mclk_source_t` {  
    `kSAI_MclkSourceSysclk` = 0x0U,  
    `kSAI_MclkSourceSelect1`,  
    `kSAI_MclkSourceSelect2`,  
    `kSAI_MclkSourceSelect3` }  
    *Master clock source.*
- enum `sai_bclk_source_t` {  
    `kSAI_BclkSourceBusclk` = 0x0U,  
    `kSAI_BclkSourceMclkDiv`,  
    `kSAI_BclkSourceOtherSai0`,  
    `kSAI_BclkSourceOtherSai1` }  
    *Bit clock source.*
- enum `_sai_interrupt_enable_t` {  
    `kSAI_WordStartInterruptEnable`,  
    `kSAI_SyncErrorInterruptEnable` = I2S\_TCSR\_SEIE\_MASK,  
    `kSAI_FIFOWarningInterruptEnable` = I2S\_TCSR\_FWIE\_MASK,  
    `kSAI_FIFOErrorInterruptEnable` = I2S\_TCSR\_FEIE\_MASK }  
    *The SAI interrupt enable flag.*
- enum `_sai_dma_enable_t` { `kSAI_FIFOWarningDMAEnable` = I2S\_TCSR\_FWDE\_MASK }  
    *The DMA request sources.*
- enum `_sai_flags` {  
    `kSAI_WordStartFlag` = I2S\_TCSR\_WSF\_MASK,  
    `kSAI_SyncErrorFlag` = I2S\_TCSR\_SEF\_MASK,  
    `kSAI_FIFOErrorFlag` = I2S\_TCSR\_FEF\_MASK,  
    `kSAI_FIFOWarningFlag` = I2S\_TCSR\_FWF\_MASK }

*The SAI status flag.*

- enum `sai_reset_type_t` {  
`kSAI_ResetTypeSoftware` = `I2S_TCSR_SR_MASK`,  
`kSAI_ResetTypeFIFO` = `I2S_TCSR_FR_MASK`,  
`kSAI_ResetAll` = `I2S_TCSR_SR_MASK | I2S_TCSR_FR_MASK` }

*The reset type.*

- enum `sai_sample_rate_t` {  
`kSAI_SampleRate8KHz` = `8000U`,  
`kSAI_SampleRate11025Hz` = `11025U`,  
`kSAI_SampleRate12KHz` = `12000U`,  
`kSAI_SampleRate16KHz` = `16000U`,  
`kSAI_SampleRate22050Hz` = `22050U`,  
`kSAI_SampleRate24KHz` = `24000U`,  
`kSAI_SampleRate32KHz` = `32000U`,  
`kSAI_SampleRate44100Hz` = `44100U`,  
`kSAI_SampleRate48KHz` = `48000U`,  
`kSAI_SampleRate96KHz` = `96000U` }

*Audio sample rate.*

- enum `sai_word_width_t` {  
`kSAI_WordWidth8bits` = `8U`,  
`kSAI_WordWidth16bits` = `16U`,  
`kSAI_WordWidth24bits` = `24U`,  
`kSAI_WordWidth32bits` = `32U` }

*Audio word width.*

## Driver version

- #define `FSL_SAI_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)  
*Version 2.1.1.*

## Initialization and deinitialization

- void `SAI_TxInit` (`I2S_Type *base`, const `sai_config_t *config`)  
*Initializes the SAI Tx peripheral.*
- void `SAI_RxInit` (`I2S_Type *base`, const `sai_config_t *config`)  
*Initializes the the SAI Rx peripheral.*
- void `SAI_TxGetDefaultConfig` (`sai_config_t *config`)  
*Sets the SAI Tx configuration structure to default values.*
- void `SAI_RxGetDefaultConfig` (`sai_config_t *config`)  
*Sets the SAI Rx configuration structure to default values.*
- void `SAI_Deinit` (`I2S_Type *base`)  
*De-initializes the SAI peripheral.*
- void `SAI_TxReset` (`I2S_Type *base`)  
*Resets the SAI Tx.*
- void `SAI_RxReset` (`I2S_Type *base`)  
*Resets the SAI Rx.*
- void `SAI_TxEnable` (`I2S_Type *base`, bool enable)  
*Enables/disables SAI Tx.*
- void `SAI_RxEnable` (`I2S_Type *base`, bool enable)

## Typical use case

*Enables/disables SAI Rx.*

## Status

- static uint32\_t [SAI\\_TxGetStatusFlag](#) (I2S\_Type \*base)  
*Gets the SAI Tx status flag state.*
- static void [SAI\\_TxClearStatusFlags](#) (I2S\_Type \*base, uint32\_t mask)  
*Clears the SAI Tx status flag state.*
- static uint32\_t [SAI\\_RxGetStatusFlag](#) (I2S\_Type \*base)  
*Gets the SAI Rx status flag state.*
- static void [SAI\\_RxClearStatusFlags](#) (I2S\_Type \*base, uint32\_t mask)  
*Clears the SAI Rx status flag state.*

## Interrupts

- static void [SAI\\_TxEnableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Enables SAI Tx interrupt requests.*
- static void [SAI\\_RxEnableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Enables SAI Rx interrupt requests.*
- static void [SAI\\_TxDisableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Disables SAI Tx interrupt requests.*
- static void [SAI\\_RxDisableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Disables SAI Rx interrupt requests.*

## DMA Control

- static void [SAI\\_TxEnableDMA](#) (I2S\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables SAI Tx DMA requests.*
- static void [SAI\\_RxEnableDMA](#) (I2S\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables SAI Rx DMA requests.*
- static uint32\_t [SAI\\_TxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Tx data register address.*
- static uint32\_t [SAI\\_RxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Rx data register address.*

## Bus Operations

- void [SAI\\_TxSetFormat](#) (I2S\_Type \*base, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_RxSetFormat](#) (I2S\_Type \*base, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- void [SAI\\_WriteBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Sends data using a blocking method.*
- static void [SAI\\_WriteData](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t data)  
*Writes data into SAI FIFO.*
- void [SAI\\_ReadBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)

*Receives data using a blocking method.*

- static uint32\_t [SAI\\_ReadData](#) (I2S\_Type \*base, uint32\_t channel)  
*Reads data from SAI FIFO.*

## Transactional

- void [SAI\\_TransferTxCreateHandle](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_callback\_t callback, void \*userData)  
*Initializes the SAI Tx handle.*
- void [SAI\\_TransferRxCreateHandle](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_callback\_t callback, void \*userData)  
*Initializes the SAI Rx handle.*
- status\_t [SAI\\_TransferTxSetFormat](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_format\_t \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- status\_t [SAI\\_TransferRxSetFormat](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_format\_t \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- status\_t [SAI\\_TransferSendNonBlocking](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_t \*xfer)  
*Performs an interrupt non-blocking send transfer on SAI.*
- status\_t [SAI\\_TransferReceiveNonBlocking](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_t \*xfer)  
*Performs an interrupt non-blocking receive transfer on SAI.*
- status\_t [SAI\\_TransferGetSendCount](#) (I2S\_Type \*base, sai\_handle\_t \*handle, size\_t \*count)  
*Gets a set byte count.*
- status\_t [SAI\\_TransferGetReceiveCount](#) (I2S\_Type \*base, sai\_handle\_t \*handle, size\_t \*count)  
*Gets a received byte count.*
- void [SAI\\_TransferAbortSend](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Aborts the current send.*
- void [SAI\\_TransferAbortReceive](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Aborts the the current IRQ receive.*
- void [SAI\\_TransferTxHandleIRQ](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Tx interrupt handler.*
- void [SAI\\_TransferRxHandleIRQ](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Rx interrupt handler.*

## 23.3 Data Structure Documentation

### 23.3.1 struct sai\_config\_t

#### Data Fields

- [sai\\_protocol\\_t](#) protocol  
*Audio bus protocol in SAI.*
- [sai\\_sync\\_mode\\_t](#) syncMode  
*SAI sync mode, control Tx/Rx clock sync.*
- [sai\\_mclk\\_source\\_t](#) mclkSource  
*Master Clock source.*

## Data Structure Documentation

- [sai\\_bclk\\_source\\_t](#) bclkSource  
*Bit Clock source.*
- [sai\\_master\\_slave\\_t](#) masterSlave  
*Master or slave.*

### 23.3.2 struct sai\_transfer\_format\_t

#### Data Fields

- uint32\_t [sampleRate\\_Hz](#)  
*Sample rate of audio data.*
- uint32\_t [bitWidth](#)  
*Data length of audio data, usually 8/16/24/32 bits.*
- [sai\\_mono\\_stereo\\_t](#) stereo  
*Mono or stereo.*
- uint32\_t [masterClockHz](#)  
*Master clock frequency in Hz.*
- uint8\_t [channel](#)  
*Data channel used in transfer.*
- [sai\\_protocol\\_t](#) protocol  
*Which audio protocol used.*

#### 23.3.2.0.0.35 Field Documentation

##### 23.3.2.0.0.35.1 uint8\_t sai\_transfer\_format\_t::channel

### 23.3.3 struct sai\_transfer\_t

#### Data Fields

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

#### 23.3.3.0.0.36 Field Documentation

##### 23.3.3.0.0.36.1 uint8\_t\* sai\_transfer\_t::data

##### 23.3.3.0.0.36.2 size\_t sai\_transfer\_t::dataSize

### 23.3.4 struct \_sai\_handle

#### Data Fields

- uint32\_t [state](#)  
*Transfer status.*

- [sai\\_transfer\\_callback\\_t](#) callback  
*Callback function called at transfer event.*
- void \* [userData](#)  
*Callback parameter passed to callback function.*
- uint8\_t [bitWidth](#)  
*Bit width for transfer, 8/16/24/32 bits.*
- uint8\_t [channel](#)  
*Transfer channel.*
- [sai\\_transfer\\_t](#) [saiQueue](#) [SAI\_XFER\_QUEUE\_SIZE]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [SAI\_XFER\_QUEUE\_SIZE]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*

## 23.4 Macro Definition Documentation

### 23.4.1 #define SAI\_XFER\_QUEUE\_SIZE (4)

## 23.5 Enumeration Type Documentation

### 23.5.1 enum \_sai\_status\_t

Enumerator

***kStatus\_SAI\_TxBusy*** SAI Tx is busy.  
***kStatus\_SAI\_RxBusy*** SAI Rx is busy.  
***kStatus\_SAI\_TxError*** SAI Tx FIFO error.  
***kStatus\_SAI\_RxError*** SAI Rx FIFO error.  
***kStatus\_SAI\_QueueFull*** SAI transfer queue is full.  
***kStatus\_SAI\_TxIdle*** SAI Tx is idle.  
***kStatus\_SAI\_RxIdle*** SAI Rx is idle.

### 23.5.2 enum sai\_protocol\_t

Enumerator

***kSAI\_BusLeftJustified*** Uses left justified format.  
***kSAI\_BusRightJustified*** Uses right justified format.  
***kSAI\_BusI2S*** Uses I2S format.  
***kSAI\_BusPCMA*** Uses I2S PCM A format.  
***kSAI\_BusPCMB*** Uses I2S PCM B format.

## Enumeration Type Documentation

### 23.5.3 enum sai\_master\_slave\_t

Enumerator

*kSAI\_Master* Master mode.

*kSAI\_Slave* Slave mode.

### 23.5.4 enum sai\_mono\_stereo\_t

Enumerator

*kSAI\_Stereo* Stereo sound.

*kSAI\_MonoLeft* Only left channel have sound.

*kSAI\_MonoRight* Only Right channel have sound.

### 23.5.5 enum sai\_sync\_mode\_t

Enumerator

*kSAI\_ModeAsync* Asynchronous mode.

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

*kSAI\_ModeSyncWithOtherTx* Synchronous with another SAI transmit.

*kSAI\_ModeSyncWithOtherRx* Synchronous with another SAI receiver.

### 23.5.6 enum sai\_mclk\_source\_t

Enumerator

*kSAI\_MclkSourceSysclk* Master clock from the system clock.

*kSAI\_MclkSourceSelect1* Master clock from source 1.

*kSAI\_MclkSourceSelect2* Master clock from source 2.

*kSAI\_MclkSourceSelect3* Master clock from source 3.

### 23.5.7 enum sai\_bclk\_source\_t

Enumerator

*kSAI\_BclkSourceBusclk* Bit clock using bus clock.

*kSAI\_BclkSourceMclkDiv* Bit clock using master clock divider.

*kSAI\_BclkSourceOtherSai0* Bit clock from other SAI device.

*kSAI\_BclkSourceOtherSai1* Bit clock from other SAI device.



### 23.5.8 enum \_sai\_interrupt\_enable\_t

Enumerator

*kSAI\_WordStartInterruptEnable* Word start flag, means the first word in a frame detected.

*kSAI\_SyncErrorInterruptEnable* Sync error flag, means the sync error is detected.

*kSAI\_FIFOWarningInterruptEnable* FIFO warning flag, means the FIFO is empty.

*kSAI\_FIFOErrorInterruptEnable* FIFO error flag.

### 23.5.9 enum \_sai\_dma\_enable\_t

Enumerator

*kSAI\_FIFOWarningDMAEnable* FIFO warning caused by the DMA request.

### 23.5.10 enum \_sai\_flags

Enumerator

*kSAI\_WordStartFlag* Word start flag, means the first word in a frame detected.

*kSAI\_SyncErrorFlag* Sync error flag, means the sync error is detected.

*kSAI\_FIFOErrorFlag* FIFO error flag.

*kSAI\_FIFOWarningFlag* FIFO warning flag.

### 23.5.11 enum sai\_reset\_type\_t

Enumerator

*kSAI\_ResetTypeSoftware* Software reset, reset the logic state.

*kSAI\_ResetTypeFIFO* FIFO reset, reset the FIFO read and write pointer.

*kSAI\_ResetAll* All reset.

### 23.5.12 enum sai\_sample\_rate\_t

Enumerator

*kSAI\_SampleRate8KHz* Sample rate 8000 Hz.

*kSAI\_SampleRate11025Hz* Sample rate 11025 Hz.

*kSAI\_SampleRate12KHz* Sample rate 12000 Hz.

*kSAI\_SampleRate16KHz* Sample rate 16000 Hz.

## Function Documentation

*kSAI\_SampleRate22050Hz* Sample rate 22050 Hz.  
*kSAI\_SampleRate24KHz* Sample rate 24000 Hz.  
*kSAI\_SampleRate32KHz* Sample rate 32000 Hz.  
*kSAI\_SampleRate44100Hz* Sample rate 44100 Hz.  
*kSAI\_SampleRate48KHz* Sample rate 48000 Hz.  
*kSAI\_SampleRate96KHz* Sample rate 96000 Hz.

### 23.5.13 enum sai\_word\_width\_t

Enumerator

*kSAI\_WordWidth8bits* Audio data width 8 bits.  
*kSAI\_WordWidth16bits* Audio data width 16 bits.  
*kSAI\_WordWidth24bits* Audio data width 24 bits.  
*kSAI\_WordWidth32bits* Audio data width 32 bits.

## 23.6 Function Documentation

### 23.6.1 void SAI\_TxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )

Ungates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI\\_TxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

### 23.6.2 void SAI\_RxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI\\_RxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

**23.6.3 void SAI\_TxGetDefaultConfig ( sai\_config\_t \* *config* )**

This API initializes the configuration structure for use in SAI\_TxConfig(). The initialized structure can remain unchanged in SAI\_TxConfig(), or it can be modified before calling SAI\_TxConfig(). Example:

```
sai_config_t config;
SAI_TxGetDefaultConfig(&config);
```

## Parameters

|               |   |
|---------------|---|
| <i>config</i> | pointer to master configuration structure |
|---------------|---|

**23.6.4 void SAI\_RxGetDefaultConfig ( sai\_config\_t \* *config* )**

This API initializes the configuration structure for use in SAI\_RxConfig(). The initialized structure can remain unchanged in SAI\_RxConfig() or it can be modified before calling SAI\_RxConfig(). Example:

```
sai_config_t config;
SAI_RxGetDefaultConfig(&config);
```

## Parameters

|               |   |
|---------------|---|
| <i>config</i> | pointer to master configuration structure |
|---------------|---|

**23.6.5 void SAI\_Deinit ( I2S\_Type \* *base* )**

This API gates the SAI clock. The SAI module can't operate unless SAI\_TxInit or SAI\_RxInit is called to enable the clock.

## Parameters

---

## Function Documentation

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

### 23.6.6 void SAI\_TxReset ( I2S\_Type \* *base* )

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

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

### 23.6.7 void SAI\_RxReset ( I2S\_Type \* *base* )

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

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

### 23.6.8 void SAI\_TxEnable ( I2S\_Type \* *base*, bool *enable* )

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SAI base pointer                               |
| <i>enable</i> | True means enable SAI Tx, false means disable. |

### 23.6.9 void SAI\_RxEnable ( I2S\_Type \* *base*, bool *enable* )

Parameters

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

### 23.6.10 static uint32\_t SAI\_TxGetStatusFlag ( I2S\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

SAI Tx status flag value. Use the Status Mask to get the status value needed.

**23.6.11 static void SAI\_TxClearStatusFlags ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | SAI base pointer   |
| <i>mask</i> | State mask. It can be a combination of the following source if defined: <ul style="list-style-type: none"> <li>• kSAI_WordStartFlag</li> <li>• kSAI_SyncErrorFlag</li> <li>• kSAI_FIFOErrorFlag</li> </ul> |

**23.6.12 static uint32\_t SAI\_RxGetStatusFlag ( I2S\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

SAI Rx status flag value. Use the Status Mask to get the status value needed.

**23.6.13 static void SAI\_RxClearStatusFlags ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Function Documentation

### Parameters

|             |  |
|-------------|--|
| <i>base</i> | SAI base pointer   |
| <i>mask</i> | State mask. It can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kSAI_WordStartFlag</li><li>• kSAI_SyncErrorFlag</li><li>• kSAI_FIFOErrorFlag</li></ul> |

### 23.6.14 static void SAI\_TxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | SAI base pointer  |
| <i>mask</i> | interrupt source The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFORequestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

### 23.6.15 static void SAI\_RxEnableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | SAI base pointer  |
| <i>mask</i> | interrupt source The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFORequestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

**23.6.16** `static void SAI_TxDisableInterrupts ( l2s_Type * base, uint32_t mask )`  
`[inline], [static]`

## Function Documentation

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | SAI base pointer  |
| <i>mask</i> | interrupt source The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFORequestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

### 23.6.17 static void SAI\_RxDisableInterrupts ( I2S\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | SAI base pointer  |
| <i>mask</i> | interrupt source The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kSAI_WordStartInterruptEnable</li><li>• kSAI_SyncErrorInterruptEnable</li><li>• kSAI_FIFOWarningInterruptEnable</li><li>• kSAI_FIFORequestInterruptEnable</li><li>• kSAI_FIFOErrorInterruptEnable</li></ul> |

### 23.6.18 static void SAI\_TxEnableDMA ( I2S\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SAI base pointer  |
| <i>mask</i>   | DMA source The parameter can be combination of the following source if defined: <ul style="list-style-type: none"><li>• kSAI_FIFOWarningDMAEnable</li><li>• kSAI_FIFORequestDMAEnable</li></ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA.   |



**23.6.19**   **static void SAI\_RxEnableDMA ( I2S\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

## Function Documentation

### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SAI base pointer   |
| <i>mask</i>   | DMA source The parameter can be a combination of the following source if defined: <ul style="list-style-type: none"><li>• kSAI_FIFOWarningDMAEnable</li><li>• kSAI_FIFOResourceDMAEnable</li></ul> |
| <i>enable</i> | True means enable DMA, false means disable DMA.  |

### 23.6.20 static uint32\_t SAI\_TxGetDataRegisterAddress ( I2S\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This API is used to provide a transfer address for SAI DMA transfer configuration.

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | SAI base pointer.        |
| <i>channel</i> | Which data channel used. |

### Returns

data register address.

### 23.6.21 static uint32\_t SAI\_RxGetDataRegisterAddress ( I2S\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This API is used to provide a transfer address for SAI DMA transfer configuration.

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | SAI base pointer.        |
| <i>channel</i> | Which data channel used. |

### Returns

data register address.

**23.6.22** void SAI\_TxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*,  
uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Function Documentation

### Parameters

|                           |   |
|---------------------------|---|
| <i>base</i>               | SAI base pointer.   |
| <i>format</i>             | Pointer to SAI audio data format structure.   |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.  |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

**23.6.23 void SAI\_RxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

### Parameters

|                           |   |
|---------------------------|---|
| <i>base</i>               | SAI base pointer.   |
| <i>format</i>             | Pointer to SAI audio data format structure.   |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.  |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

**23.6.24 void SAI\_WriteBlocking ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *bitWidth*, uint8\_t \* *buffer*, uint32\_t *size* )**

### Note

This function blocks by polling until data is ready to be sent.

### Parameters

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

|                 |   |
|-----------------|---|
| <i>channel</i>  | Data channel used.                                      |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>buffer</i>   | Pointer to the data to be written.                      |
| <i>size</i>     | Bytes to be written.                                    |

**23.6.25 static void SAI\_WriteData ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *data* ) [inline], [static]**

Parameters

|                |                           |
|----------------|---------------------------|
| <i>base</i>    | SAI base pointer.         |
| <i>channel</i> | Data channel used.        |
| <i>data</i>    | Data needs to be written. |

**23.6.26 void SAI\_ReadBlocking ( I2S\_Type \* *base*, uint32\_t *channel*, uint32\_t *bitWidth*, uint8\_t \* *buffer*, uint32\_t *size* )**

Note

This function blocks by polling until data is ready to be sent.

Parameters

|                 |   |
|-----------------|---|
| <i>base</i>     | SAI base pointer.                                       |
| <i>channel</i>  | Data channel used.                                      |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>buffer</i>   | Pointer to the data to be read.                         |
| <i>size</i>     | Bytes to be read.                                       |

**23.6.27 static uint32\_t SAI\_ReadData ( I2S\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

## Function Documentation

### Parameters

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

### Returns

Data in SAI FIFO.

### 23.6.28 void SAI\_TransferTxCreateHandle ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the Tx handle for SAI Tx transactional APIs. Call this function one time to get the handle initialized.

### Parameters

|                 |  |
|-----------------|--|
| <i>base</i>     | SAI base pointer                               |
| <i>handle</i>   | SAI handle pointer.                            |
| <i>callback</i> | pointer to user callback function              |
| <i>userData</i> | user parameter passed to the callback function |

### 23.6.29 void SAI\_TransferRxCreateHandle ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the Rx handle for SAI Rx transactional APIs. Call this function one time to get the handle initialized.

### Parameters

|                 |  |
|-----------------|--|
| <i>base</i>     | SAI base pointer.                              |
| <i>handle</i>   | SAI handle pointer.                            |
| <i>callback</i> | pointer to user callback function              |
| <i>userData</i> | user parameter passed to the callback function |

**23.6.30** `status_t SAI_TransferTxSetFormat ( I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz )`

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Function Documentation

### Parameters

|                           |  |
|---------------------------|--|
| <i>base</i>               | SAI base pointer.  |
| <i>handle</i>             | SAI handle pointer.  |
| <i>format</i>             | Pointer to SAI audio data format structure.  |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.   |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal to masterClockHz in format. |

### Returns

Status of this function. Return value is one of status\_t.

**23.6.31** **status\_t SAI\_TransferRxSetFormat ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

### Parameters

|                           |   |
|---------------------------|---|
| <i>base</i>               | SAI base pointer.   |
| <i>handle</i>             | SAI handle pointer.   |
| <i>format</i>             | Pointer to SAI audio data format structure.   |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.  |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

### Returns

Status of this function. Return value is one of status\_t.

**23.6.32** **status\_t SAI\_TransferSendNonBlocking ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**



## Note

This API returns immediately after the transfer initiates. Call the SAI\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SAI\_Busy, the transfer is finished.

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SAI base pointer  |
| <i>handle</i> | pointer to sai_handle_t structure which stores the transfer state |
| <i>xfer</i>   | pointer to sai_transfer_t structure                               |

## Return values

|                                |  |
|--------------------------------|--|
| <i>kStatus_Success</i>         | Successfully started the data receive. |
| <i>kStatus_SAI_TxBusy</i>      | Previous receive still not finished.   |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.        |

### 23.6.33 status\_t SAI\_TransferReceiveNonBlocking ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## Note

This API returns immediately after the transfer initiates. Call the SAI\_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SAI\_Busy, the transfer is finished.

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SAI base pointer  |
| <i>handle</i> | pointer to sai_handle_t structure which stores the transfer state |
| <i>xfer</i>   | pointer to sai_transfer_t structure                               |

## Return values

|                        |  |
|------------------------|--|
| <i>kStatus_Success</i> | Successfully started the data receive. |
|------------------------|--|

## Function Documentation

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_SAI_RxBusy</i>      | Previous receive still not finished. |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.      |

### 23.6.34 **status\_t SAI\_TransferGetSendCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )**

#### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SAI base pointer.  |
| <i>handle</i> | pointer to sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count sent.  |

#### Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

### 23.6.35 **status\_t SAI\_TransferGetReceiveCount ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, size\_t \* *count* )**

#### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SAI base pointer.  |
| <i>handle</i> | pointer to sai_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count received.  |

#### Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

### 23.6.36 **void SAI\_TransferAbortSend ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )**

## Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

## Parameters

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

### 23.6.37 void SAI\_TransferAbortReceive ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

## Note

This API can be called any time when an interrupt non-blocking transfer initiates to abort the transfer early.

## Parameters

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

### 23.6.38 void SAI\_TransferTxHandleIRQ ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer.                  |
| <i>handle</i> | pointer to sai_handle_t structure. |

### 23.6.39 void SAI\_TransferRxHandleIRQ ( I2S\_Type \* *base*, sai\_handle\_t \* *handle* )

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer.                  |
| <i>handle</i> | pointer to sai_handle_t structure. |

### 23.7 SAI DMA Driver

#### 23.7.1 Overview

##### Data Structures

- struct [sai\\_dma\\_handle\\_t](#)  
*SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

##### Typedefs

- typedef void(\* [sai\\_dma\\_callback\\_t](#))(I2S\_Type \*base, sai\_dma\_handle\_t \*handle, status\_t status, void \*userData)  
*Define SAI DMA callback.*

##### DMA Transactional

- void [SAI\\_TransferTxCreateHandleDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_dma\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*dmaHandle)  
*Initializes the SAI master DMA handle.*
- void [SAI\\_TransferRxCreateHandleDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_dma\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*dmaHandle)  
*Initializes the SAI slave DMA handle.*
- void [SAI\\_TransferTxSetFormatDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_TransferRxSetFormatDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- status\_t [SAI\\_TransferSendDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI transfer using DMA.*
- status\_t [SAI\\_TransferReceiveDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI transfer using DMA.*
- void [SAI\\_TransferAbortSendDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle)  
*Aborts a SAI transfer using DMA.*
- void [SAI\\_TransferAbortReceiveDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle)  
*Aborts a SAI transfer using DMA.*
- status\_t [SAI\\_TransferGetSendCountDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets byte count sent by SAI.*
- status\_t [SAI\\_TransferGetReceiveCountDMA](#) (I2S\_Type \*base, sai\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets byte count received by SAI.*

## 23.7.2 Data Structure Documentation

### 23.7.2.1 struct \_sai\_dma\_handle

#### Data Fields

- [dma\\_handle\\_t](#) \* [dmaHandle](#)  
*DMA handler for SAI send.*
- [uint8\\_t](#) [bytesPerFrame](#)  
*Bytes in a frame.*
- [uint8\\_t](#) [channel](#)  
*Which Data channel SAI use.*
- [uint32\\_t](#) [state](#)  
*SAI DMA transfer internal state.*
- [sai\\_dma\\_callback\\_t](#) [callback](#)  
*Callback for users while transfer finish or error occurred.*
- [void](#) \* [userData](#)  
*User callback parameter.*
- [sai\\_transfer\\_t](#) [saiQueue](#) [[SAI\\_XFER\\_QUEUE\\_SIZE](#)]  
*Transfer queue storing queued transfer.*
- [size\\_t](#) [transferSize](#) [[SAI\\_XFER\\_QUEUE\\_SIZE](#)]  
*Data bytes need to transfer.*
- [volatile uint8\\_t](#) [queueUser](#)  
*Index for user to queue transfer.*
- [volatile uint8\\_t](#) [queueDriver](#)  
*Index for driver to get the transfer data and size.*

#### 23.7.2.1.0.37 Field Documentation

23.7.2.1.0.37.1 [sai\\_transfer\\_t](#) [sai\\_dma\\_handle\\_t::saiQueue](#)[[SAI\\_XFER\\_QUEUE\\_SIZE](#)]

23.7.2.1.0.37.2 [volatile uint8\\_t](#) [sai\\_dma\\_handle\\_t::queueUser](#)

## 23.7.3 Function Documentation

23.7.3.1 **`void SAI_TransferTxCreateHandleDMA ( I2S_Type * base, sai\_dma\_handle\_t * handle, sai\_dma\_callback\_t callback, void * userData, dma\_handle\_t * dmaHandle )`**

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

---

## SAI DMA Driver

|                  |   |
|------------------|---|
| <i>base</i>      | SAI base pointer.   |
| <i>handle</i>    | SAI DMA handle pointer.   |
| <i>base</i>      | SAI peripheral base address.  |
| <i>callback</i>  | Pointer to user callback function.                                  |
| <i>userData</i>  | User parameter passed to the callback function.                     |
| <i>dmaHandle</i> | DMA handle pointer, this handle shall be static allocated by users. |

**23.7.3.2 void SAI\_TransferRxCreateHandleDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_dma\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *dmaHandle* )**

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

Parameters

|                  |   |
|------------------|---|
| <i>base</i>      | SAI base pointer.   |
| <i>handle</i>    | SAI DMA handle pointer.   |
| <i>base</i>      | SAI peripheral base address.  |
| <i>callback</i>  | Pointer to user callback function.                                  |
| <i>userData</i>  | User parameter passed to the callback function.                     |
| <i>dmaHandle</i> | DMA handle pointer, this handle shall be static allocated by users. |

**23.7.3.3 void SAI\_TransferTxSetFormatDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to the format.

Parameters

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

|                           |  |
|---------------------------|--|
| <i>format</i>             | Pointer to SAI audio data format structure.  |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.   |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master. clock, this value should equals to masterClockHz in format. |

Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully.  |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid. |

**23.7.3.4 void SAI\_TransferRxSetFormatDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets eDMA parameter according to format.

Parameters

|                           |  |
|---------------------------|--|
| <i>base</i>               | SAI base pointer.  |
| <i>handle</i>             | SAI DMA handle pointer.  |
| <i>format</i>             | Pointer to SAI audio data format structure.  |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.   |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master. clock, this value should equals to masterClockHz in format. |

Return values

|                                |                                 |
|--------------------------------|---------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully.  |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid. |

**23.7.3.5 status\_t SAI\_TransferSendDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

Note

This interface returns immediately after the transfer initiates. Call the SAI\_GetTransferStatus to poll the transfer status to check whether the SAI transfer finished.

## SAI DMA Driver

### Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer.                  |
| <i>handle</i> | SAI DMA handle pointer.            |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

### Return values

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data receive. |
| <i>kStatus_SAI_TxBusy</i>      | Previous receive still not finished. |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.      |

### 23.7.3.6 **status\_t SAI\_TransferReceiveDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

#### Note

This interface returns immediately after transfer initiates. Call SAI\_GetTransferStatus to poll the transfer status to check whether the SAI transfer is finished.

### Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer                   |
| <i>handle</i> | SAI DMA handle pointer.            |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

### Return values

|                                |                                      |
|--------------------------------|--------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data receive. |
| <i>kStatus_SAI_RxBusy</i>      | Previous receive still not finished. |
| <i>kStatus_InvalidArgument</i> | The input parameter is invalid.      |

### 23.7.3.7 **void SAI\_TransferAbortSendDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle* )**



## Parameters

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

### 23.7.3.8 void SAI\_TransferAbortReceiveDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle* )

## Parameters

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

### 23.7.3.9 status\_t SAI\_TransferGetSendCountDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI DMA handle pointer.  |
| <i>count</i>  | Bytes count sent by SAI. |

## Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

### 23.7.3.10 status\_t SAI\_TransferGetReceiveCountDMA ( I2S\_Type \* *base*, sai\_dma\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

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

## SAI DMA Driver

|               |                              |
|---------------|------------------------------|
| <i>handle</i> | SAI DMA handle pointer.      |
| <i>count</i>  | Bytes count received by SAI. |

### Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

## 23.8 SAI eDMA Driver

### 23.8.1 Overview

#### Data Structures

- struct [sai\\_edma\\_handle\\_t](#)  
SAI DMA transfer handle, users should not touch the content of the handle. [More...](#)

#### Typedefs

- typedef void(\* [sai\\_edma\\_callback\\_t](#))(I2S\_Type \*base, sai\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
SAI eDMA transfer callback function for finish and error.

#### eDMA Transactional

- void [SAI\\_TransferTxCreateHandleEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_edma\\_callback\\_t](#) callback, void \*userData, edma\_handle\_t \*dmaHandle)  
*Initializes the SAI eDMA handle.*
- void [SAI\\_TransferRxCreateHandleEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_edma\\_callback\\_t](#) callback, void \*userData, edma\_handle\_t \*dmaHandle)  
*Initializes the SAI Rx eDMA handle.*
- void [SAI\\_TransferTxSetFormatEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_TransferRxSetFormatEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- status\_t [SAI\\_TransferSendEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI transfer using DMA.*
- status\_t [SAI\\_TransferReceiveEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SAI receive using eDMA.*
- void [SAI\\_TransferAbortSendEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle)  
*Aborts a SAI transfer using eDMA.*
- void [SAI\\_TransferAbortReceiveEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle)  
*Aborts a SAI receive using eDMA.*
- status\_t [SAI\\_TransferGetSendCountEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets byte count sent by SAI.*
- status\_t [SAI\\_TransferGetReceiveCountEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, size\_t \*count)  
*Gets byte count received by SAI.*

### 23.8.2 Data Structure Documentation

#### 23.8.2.1 struct \_sai\_edma\_handle

##### Data Fields

- `edma_handle_t * dmaHandle`  
*DMA handler for SAI send.*
- `uint8_t bytesPerFrame`  
*Bytes in a frame.*
- `uint8_t channel`  
*Which data channel.*
- `uint8_t count`  
*The transfer data count in a DMA request.*
- `uint32_t state`  
*Internal state for SAI eDMA transfer.*
- `sai_edma_callback_t callback`  
*Callback for users while transfer finish or error occurs.*
- `void * userData`  
*User callback parameter.*
- `edma_tcd_t tcd [SAI_XFER_QUEUE_SIZE+1U]`  
*TCD pool for eDMA transfer.*
- `sai_transfer_t saiQueue [SAI_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*
- `size_t transferSize [SAI_XFER_QUEUE_SIZE]`  
*Data bytes need to transfer.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*

##### 23.8.2.1.0.38 Field Documentation

**23.8.2.1.0.38.1** `edma_tcd_t sai_edma_handle_t::tcd[SAI_XFER_QUEUE_SIZE+1U]`

**23.8.2.1.0.38.2** `sai_transfer_t sai_edma_handle_t::saiQueue[SAI_XFER_QUEUE_SIZE]`

**23.8.2.1.0.38.3** `volatile uint8_t sai_edma_handle_t::queueUser`

### 23.8.3 Function Documentation

**23.8.3.1** `void SAI_TransferTxCreateHandleEDMA ( I2S_Type * base, sai_edma_handle_t * handle, sai_edma_callback_t callback, void * userData, edma_handle_t * dmaHandle )`

This function initializes the SAI master DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

## Parameters

|                  |  |
|------------------|--|
| <i>base</i>      | SAI base pointer.  |
| <i>handle</i>    | SAI eDMA handle pointer.   |
| <i>base</i>      | SAI peripheral base address.   |
| <i>callback</i>  | Pointer to user callback function.                                   |
| <i>userData</i>  | User parameter passed to the callback function.                      |
| <i>dmaHandle</i> | eDMA handle pointer, this handle shall be static allocated by users. |

**23.8.3.2 void SAI\_TransferRxCreateHandleEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_edma\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *dmaHandle* )**

This function initializes the SAI slave DMA handle, which can be used for other SAI master transactional APIs. Usually, for a specified SAI instance, call this API once to get the initialized handle.

## Parameters

|                  |  |
|------------------|--|
| <i>base</i>      | SAI base pointer.  |
| <i>handle</i>    | SAI eDMA handle pointer.   |
| <i>base</i>      | SAI peripheral base address.   |
| <i>callback</i>  | Pointer to user callback function.                                   |
| <i>userData</i>  | User parameter passed to the callback function.                      |
| <i>dmaHandle</i> | eDMA handle pointer, this handle shall be static allocated by users. |

**23.8.3.3 void SAI\_TransferTxSetFormatEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

## Parameters

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

## SAI eDMA Driver

|                           |   |
|---------------------------|---|
| <i>handle</i>             | SAI eDMA handle pointer.  |
| <i>format</i>             | Pointer to SAI audio data format structure.   |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.  |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**23.8.3.4 void SAI\_TransferRxSetFormatEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

Parameters

|                           |  |
|---------------------------|--|
| <i>base</i>               | SAI base pointer.  |
| <i>handle</i>             | SAI eDMA handle pointer.   |
| <i>format</i>             | Pointer to SAI audio data format structure.  |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.   |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format. |

Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**23.8.3.5 status\_t SAI\_TransferSendEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Note

This interface returns immediately after the transfer initiates. Call SAI\_GetTransferStatus to poll the transfer status and check whether the SAI transfer is finished.

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SAI base pointer.                      |
| <i>handle</i> | SAI eDMA handle pointer.               |
| <i>xfer</i>   | Pointer to the DMA transfer structure. |

## Return values

|                                |                                     |
|--------------------------------|-------------------------------------|
| <i>kStatus_Success</i>         | Start a SAI eDMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.      |
| <i>kStatus_TxBusy</i>          | SAI is busy sending data.           |

### 23.8.3.6 status\_t SAI\_TransferReceiveEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## Note

This interface returns immediately after the transfer initiates. Call the SAI\_GetReceiveRemaining-Bytes to poll the transfer status and check whether the SAI transfer is finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SAI base pointer                   |
| <i>handle</i> | SAI eDMA handle pointer.           |
| <i>xfer</i>   | Pointer to DMA transfer structure. |

## Return values

|                                |  |
|--------------------------------|--|
| <i>kStatus_Success</i>         | Start a SAI eDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.         |
| <i>kStatus_RxBusy</i>          | SAI is busy receiving data.            |

### 23.8.3.7 void SAI\_TransferAbortSendEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle* )

## SAI eDMA Driver

### Parameters

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

### 23.8.3.8 void SAI\_TransferAbortReceiveEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle* )

### Parameters

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

### 23.8.3.9 status\_t SAI\_TransferGetSendCountEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | SAI base pointer.        |
| <i>handle</i> | SAI eDMA handle pointer. |
| <i>count</i>  | Bytes count sent by SAI. |

### Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                   |
| <i>kStatus_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |

### 23.8.3.10 status\_t SAI\_TransferGetReceiveCountEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

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



|               |                              |
|---------------|------------------------------|
| <i>handle</i> | SAI eDMA handle pointer.     |
| <i>count</i>  | Bytes count received by SAI. |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                   |
| <i>kStatus_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |



## Chapter 24

# SIM: System Integration Module Driver

### 24.1 Overview

The KSDK provides a peripheral driver for the System Integration Module (SIM) of Kinetis devices.

### Data Structures

- struct [sim\\_uid\\_t](#)  
*Unique ID. [More...](#)*

### Enumerations

- enum [\\_sim\\_flash\\_mode](#) {  
    [kSIM\\_FlashDisableInWait](#) = SIM\_FCFG1\_FLASHDOZE\_MASK,  
    [kSIM\\_FlashDisable](#) = SIM\_FCFG1\_FLASHDIS\_MASK }  
*Flash enable mode.*

### Functions

- void [SIM\\_GetUniqueId](#) ([sim\\_uid\\_t](#) \*uid)  
*Get the unique identification register value.*
- static void [SIM\\_SetFlashMode](#) (uint8\_t mode)  
*Set the flash enable mode.*

### Driver version

- #define [FSL\\_SIM\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 0))  
*Driver version 2.0.0.*

## 24.2 Data Structure Documentation

### 24.2.1 struct [sim\\_uid\\_t](#)

#### Data Fields

- uint32\_t [MH](#)  
*UIDMH.*
- uint32\_t [ML](#)  
*UIDML.*
- uint32\_t [L](#)  
*UIDL.*

## Function Documentation

### 24.2.1.0.0.39 Field Documentation

24.2.1.0.0.39.1 uint32\_t sim\_uid\_t::MH

24.2.1.0.0.39.2 uint32\_t sim\_uid\_t::ML

24.2.1.0.0.39.3 uint32\_t sim\_uid\_t::L

## 24.3 Enumeration Type Documentation

### 24.3.1 enum \_sim\_flash\_mode

Enumerator

*kSIM\_FlashDisableInWait* Disable flash in wait mode.

*kSIM\_FlashDisable* Disable flash in normal mode.

## 24.4 Function Documentation

### 24.4.1 void SIM\_GetUniqueld ( sim\_uid\_t \* uid )

Parameters

|            |   |
|------------|---|
| <i>uid</i> | Pointer to the structure to save the UID value. |
|------------|---|

### 24.4.2 static void SIM\_SetFlashMode ( uint8\_t mode ) [inline], [static]

Parameters

|             |  |
|-------------|--|
| <i>mode</i> | The mode to set, see <a href="#">_sim_flash_mode</a> for mode details. |
|-------------|--|

## Chapter 25

# SLCD: Segment LCD Driver

### 25.1 Overview

The KSDK provides a peripheral driver for the Segment LCD (SLCD) module of Kinetis devices. The SLCD module is a CMOS charge pump voltage inverter that is designed for low voltage and low-power operation. SLCD is designed to generate the appropriate waveforms to drive multiplexed numeric, alphanumeric, or custom segment LCD panels. SLCD also has several timing and control settings that can be software-configured depending on the application's requirements. Timing and control consists of registers and control logic for the following:

1. LCD frame frequency
2. Duty cycle selection
3. Front plane/back plane selection and enabling
4. Blink modes and frequency
5. Operation in low-power modes

After the SLCD general initialization, the [SLCD\\_SetBackPlanePhase\(\)](#), [SLCD\\_SetFrontPlaneSegments\(\)](#), and [SLCD\\_SetFrontPlaneOnePhase\(\)](#) are used to set the special back/front Plane to make SLCD display correctly. Then, the independent display control APIs, [SLCD\\_StartDisplay\(\)](#) and [SLCD\\_StopDisplay\(\)](#), start and stop the SLCD display.

The [SLCD\\_StartBlinkMode\(\)](#) and [SLCD\\_StopBlinkMode\(\)](#) are provided for the runtime special blink mode control. To get the SLCD fault detection result, call the [SLCD\\_GetFaultDetectCounter\(\)](#).

### 25.2 Typical use case

#### 25.2.1 SLCD Initialization operation

```
slcd_config_t configure = 0;
slcd_clock_config_t clkConfig =
{
    kSLCD_AlternateClk1,
    kSLCD_AltClkDivFactor1,
    kSLCD_ClkPrescaler00
#ifdef FSL_FEATURE_SLCD_HAS_FAST_FRAME_RATE
    ,
    false
#endif
};
SLCD_GetDefaultConfig(&configure);
configure.clkConfig      = &clkConfig;
configure.loadAdjust     = kSLCD_LowLoadOrIntermediateClkSrc;
configure.dutyCycle      = kSLCD_1Div4DutyCycle;
configure.slcdlowPinEnabled = 0x1a44;
configure.backPlaneLowPin = 0x0822;
configure.faultConfig    = NULL;

SLCD_Init(base, &configure);
```

## Typical use case

```
SLCD_SetBackPlanePhase(base, 1, kSLCD_PhaseAActivate);
SLCD_SetBackPlanePhase(base, 5, kSLCD_PhaseBActivate);
SLCD_SetBackPlanePhase(base, 11, kSLCD_PhaseCActivate);

SLCD_SetFrontPlaneSegments(base, 0, (
    kSLCD_PhaseAActivate | kSLCD_PhaseBActivate));
SLCD_SetFrontPlaneSegments(base, 9, (
    kSLCD_PhaseBActivate | kSLCD_PhaseCActivate));

SLCD_StartDisplay(base);
```

## Data Structures

- struct `slcd_fault_detect_config_t`  
*SLCD fault frame detection configure structure. [More...](#)*
- struct `slcd_clock_config_t`  
*SLCD clock configure structure. [More...](#)*
- struct `slcd_config_t`  
*SLCD configure structure. [More...](#)*

## Enumerations

- enum `slcd_power_supply_option_t` {  
    `kSLCD_InternalVil3UseChargePump`,  
    `kSLCD_ExternalVil3UseResistorBiasNetwork`,  
    `kSLCD_ExtetanlVil3UseChargePump`,  
    `kSLCD_InternalVil1UseChargePump` }  
*SLCD power supply option.*
- enum `slcd_regulated_voltage_trim_t` {  
    `kSLCD_RegulatedVolatgeTrim00` = 0U,  
    `kSLCD_RegulatedVolatgeTrim01`,  
    `kSLCD_RegulatedVolatgeTrim02`,  
    `kSLCD_RegulatedVolatgeTrim03`,  
    `kSLCD_RegulatedVolatgeTrim04`,  
    `kSLCD_RegulatedVolatgeTrim05`,  
    `kSLCD_RegulatedVolatgeTrim06`,  
    `kSLCD_RegulatedVolatgeTrim07`,  
    `kSLCD_RegulatedVolatgeTrim08`,  
    `kSLCD_RegulatedVolatgeTrim09`,  
    `kSLCD_RegulatedVolatgeTrim10`,  
    `kSLCD_RegulatedVolatgeTrim11`,  
    `kSLCD_RegulatedVolatgeTrim12`,  
    `kSLCD_RegulatedVolatgeTrim13`,  
    `kSLCD_RegulatedVolatgeTrim14`,  
    `kSLCD_RegulatedVolatgeTrim15` }  
*SLCD regulated voltage trim parameter, be used to meet the desired contrast.*
- enum `slcd_load_adjust_t` {  
    `kSLCD_LowLoadOrFastestClkSrc` = 0U,  
    `kSLCD_LowLoadOrIntermediateClkSrc`,  
    `kSLCD_HighLoadOrIntermediateClkSrc`,

`kSLCD_HighLoadOrSlowestClkSrc }`

*SLCD load adjust to handle different LCD glass capacitance or configure the LCD charge pump clock source.*

- enum `slcd_clock_src_t` {  
`kSLCD_DefaultClk` = 0U,  
`kSLCD_AlternateClk1` = 1U }

*SLCD clock source.*

- enum `slcd_alt_clock_div_t` {  
`kSLCD_AltClkDivFactor1` = 0U,  
`kSLCD_AltClkDivFactor64`,  
`kSLCD_AltClkDivFactor256`,  
`kSLCD_AltClkDivFactor512` }

*SLCD alternate clock divider.*

- enum `slcd_clock_prescaler_t` {  
`kSLCD_ClkPrescaler00` = 0U,  
`kSLCD_ClkPrescaler01`,  
`kSLCD_ClkPrescaler02`,  
`kSLCD_ClkPrescaler03`,  
`kSLCD_ClkPrescaler04`,  
`kSLCD_ClkPrescaler05`,  
`kSLCD_ClkPrescaler06`,  
`kSLCD_ClkPrescaler07` }

*SLCD clock prescaler to generate frame frequency.*

- enum `slcd_duty_cycle_t` {  
`kSLCD_1Div1DutyCycle` = 0U,  
`kSLCD_1Div2DutyCycle`,  
`kSLCD_1Div3DutyCycle`,  
`kSLCD_1Div4DutyCycle`,  
`kSLCD_1Div5DutyCycle`,  
`kSLCD_1Div6DutyCycle`,  
`kSLCD_1Div7DutyCycle`,  
`kSLCD_1Div8DutyCycle` }

*SLCD duty cycle.*

- enum `slcd_phase_type_t` {  
`kSLCD_NoPhaseActivate` = 0x00U,  
`kSLCD_PhaseAActivate` = 0x01U,  
`kSLCD_PhaseBActivate` = 0x02U,  
`kSLCD_PhaseCActivate` = 0x04U,  
`kSLCD_PhaseDActivate` = 0x08U,  
`kSLCD_PhaseEActivate` = 0x10U,  
`kSLCD_PhaseFActivate` = 0x20U,  
`kSLCD_PhaseGActivate` = 0x40U,  
`kSLCD_PhaseHActivate` = 0x80U }

*SLCD segment phase type.*

- enum `slcd_phase_index_t` {

## Typical use case

```
kSLCD_PhaseAIndex = 0x0U,  
kSLCD_PhaseBIndex = 0x1U,  
kSLCD_PhaseCIndex = 0x2U,  
kSLCD_PhaseDIndex = 0x3U,  
kSLCD_PhaseEIndex = 0x4U,  
kSLCD_PhaseFIndex = 0x5U,  
kSLCD_PhaseGIndex = 0x6U,  
kSLCD_PhaseHIndex = 0x7U }
```

*SLCD segment phase bit index.*

- enum `slcd_display_mode_t` {  
    kSLCD\_NormalMode = 0U,  
    kSLCD\_AlternateMode,  
    kSLCD\_BlankMode }

*SLCD display mode.*

- enum `slcd_blink_mode_t` {  
    kSLCD\_BlankDisplayBlink = 0U,  
    kSLCD\_AltDisplayBlink }

*SLCD blink mode.*

- enum `slcd_blink_rate_t` {  
    kSLCD\_BlinkRate00 = 0U,  
    kSLCD\_BlinkRate01,  
    kSLCD\_BlinkRate02,  
    kSLCD\_BlinkRate03,  
    kSLCD\_BlinkRate04,  
    kSLCD\_BlinkRate05,  
    kSLCD\_BlinkRate06,  
    kSLCD\_BlinkRate07 }

*SLCD blink rate.*

- enum `slcd_fault_detect_clock_prescaler_t` {  
    kSLCD\_FaultSampleFreqDivider1 = 0U,  
    kSLCD\_FaultSampleFreqDivider2,  
    kSLCD\_FaultSampleFreqDivider4,  
    kSLCD\_FaultSampleFreqDivider8,  
    kSLCD\_FaultSampleFreqDivider16,  
    kSLCD\_FaultSampleFreqDivider32,  
    kSLCD\_FaultSampleFreqDivider64,  
    kSLCD\_FaultSampleFreqDivider128 }

*SLCD fault detect clock prescaler.*

- enum `slcd_fault_detect_sample_window_width_t` {  
    kSLCD\_FaultDetectWindowWidth4SampleClk = 0U,  
    kSLCD\_FaultDetectWindowWidth8SampleClk,  
    kSLCD\_FaultDetectWindowWidth16SampleClk,  
    kSLCD\_FaultDetectWindowWidth32SampleClk,  
    kSLCD\_FaultDetectWindowWidth64SampleClk,  
    kSLCD\_FaultDetectWindowWidth128SampleClk,  
    kSLCD\_FaultDetectWindowWidth256SampleClk,



- `kSLCD_FaultDetectWindowWidth512SampleClk }`  
*SLCD fault detect sample window width.*
- enum `slcd_interrupt_enable_t` { `kSLCD_FaultDetectCompleteInterrupt = 1U` }  
*SLCD interrupt source.*
- enum `slcd_lowpower_behavior` {  
    `kSLCD_EnabledInWaitStop = 0`,  
    `kSLCD_EnabledInWaitOnly`,  
    `kSLCD_EnabledInStopOnly`,  
    `kSLCD_DisabledInWaitStop` }  
*SLCD behavior in low power mode.*

## Driver version

- #define `FSL_SLCD_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)  
*SLCD driver version 2.0.1.*

## Initialization and deinitialization

- void `SLCD_Init` (`LCD_Type *base`, `slcd_config_t *configure`)  
*Initializes the SLCD, ungates the module clock, initializes the power setting, enables all used plane pins, and sets with interrupt and work mode with configuration.*
- void `SLCD_Deinit` (`LCD_Type *base`)  
*Deinitializes the SLCD module, gates the module clock, disables an interrupt, and displays the SLCD.*
- void `SLCD_GetDefaultConfig` (`slcd_config_t *configure`)  
*Gets the SLCD default configuration structure.*

## Plane Setting and Display Control

- static void `SLCD_StartDisplay` (`LCD_Type *base`)  
*Enables the SLCD controller, starts generate, and displays the front plane and back plane waveform.*
- static void `SLCD_StopDisplay` (`LCD_Type *base`)  
*Stops the SLCD controller.*
- void `SLCD_StartBlinkMode` (`LCD_Type *base`, `slcd_blink_mode_t mode`, `slcd_blink_rate_t rate`)  
*Starts the SLCD blink mode.*
- static void `SLCD_StopBlinkMode` (`LCD_Type *base`)  
*Stops the SLCD blink mode.*
- static void `SLCD_SetBackPlanePhase` (`LCD_Type *base`, `uint32_t pinIdx`, `slcd_phase_type_t phase`)  
*Sets the SLCD back plane pin phase.*
- static void `SLCD_SetFrontPlaneSegments` (`LCD_Type *base`, `uint32_t pinIdx`, `uint8_t operation`)  
*Sets the SLCD front plane segment operation for a front plane pin.*
- static void `SLCD_SetFrontPlaneOnePhase` (`LCD_Type *base`, `uint32_t pinIdx`, `slcd_phase_index_t phaseIdx`, `bool enable`)  
*Sets one SLCD front plane pin for one phase.*
- static `uint32_t SLCD_GetFaultDetectCounter` (`LCD_Type *base`)  
*Gets the SLCD fault detect counter.*

## Interrupts.

- void `SLCD_EnableInterrupts` (`LCD_Type *base`, `uint32_t mask`)

## Data Structure Documentation

- *Enables the SLCD interrupt.*  
void [SLCD\\_DisableInterrupts](#) (LCD\_Type \*base, uint32\_t mask)
- *Disables the SLCD interrupt.*  
uint32\_t [SLCD\\_GetInterruptStatus](#) (LCD\_Type \*base)
- *Gets the SLCD interrupt status flag.*  
void [SLCD\\_ClearInterruptStatus](#) (LCD\_Type \*base, uint32\_t mask)
- *Clears the SLCD interrupt events status flag.*

## 25.3 Data Structure Documentation

### 25.3.1 struct slcd\_fault\_detect\_config\_t

#### Data Fields

- bool [faultDetectIntEnable](#)  
*Fault frame detection interrupt enable flag.*
- bool [faultDetectBackPlaneEnable](#)  
*True means the pin id fault detected is back plane otherwise front plane.*
- uint8\_t [faultDetectPinIndex](#)  
*Fault detected pin id from 0 to 63.*
- [slcd\\_fault\\_detect\\_clock\\_prescaler\\_t](#) [faultPrescaler](#)  
*Fault detect clock prescaler.*
- [slcd\\_fault\\_detect\\_sample\\_window\\_width\\_t](#) [width](#)  
*Fault detect sample window width.*

#### 25.3.1.0.0.40 Field Documentation

25.3.1.0.0.40.1 bool slcd\_fault\_detect\_config\_t::faultDetectIntEnable

25.3.1.0.0.40.2 bool slcd\_fault\_detect\_config\_t::faultDetectBackPlaneEnable

25.3.1.0.0.40.3 uint8\_t slcd\_fault\_detect\_config\_t::faultDetectPinIndex

25.3.1.0.0.40.4 slcd\_fault\_detect\_clock\_prescaler\_t slcd\_fault\_detect\_config\_t::faultPrescaler

25.3.1.0.0.40.5 slcd\_fault\_detect\_sample\_window\_width\_t slcd\_fault\_detect\_config\_t::width

### 25.3.2 struct slcd\_clock\_config\_t

#### Data Fields

- [slcd\\_clock\\_src\\_t](#) [clkSource](#)  
*Clock source.*
- [slcd\\_alt\\_clock\\_div\\_t](#) [altClkDivider](#)  
*The divider to divide the alternate clock used for alternate clock source.*
- [slcd\\_clock\\_prescaler\\_t](#) [clkPrescaler](#)  
*Clock prescaler.*

### 25.3.2.0.0.41 Field Documentation

#### 25.3.2.0.0.41.1 `slcd_clock_src_t slcd_clock_config_t::clkSource`

"slcd\_clock\_src\_t" is recommended to be used. The SLCD is optimized to operate using a 32.768kHz clock input.

#### 25.3.2.0.0.41.2 `slcd_alt_clock_div_t slcd_clock_config_t::altClkDivider`

#### 25.3.2.0.0.41.3 `slcd_clock_prescaler_t slcd_clock_config_t::clkPrescaler`

### 25.3.3 `struct slcd_config_t`

#### Data Fields

- [slcd\\_power\\_supply\\_option\\_t](#) `powerSupply`  
*Power supply option.*
- [slcd\\_regulated\\_voltage\\_trim\\_t](#) `voltageTrim`  
*Regulated voltage trim used for the internal regulator VIREG to adjust to facilitate contrast control.*
- [slcd\\_clock\\_config\\_t](#) \* `clkConfig`  
*Clock configure.*
- [slcd\\_display\\_mode\\_t](#) `displayMode`  
*SLCD display mode.*
- [slcd\\_load\\_adjust\\_t](#) `loadAdjust`  
*Load adjust to handle glass capacitance.*
- [slcd\\_duty\\_cycle\\_t](#) `dutyCycle`  
*Duty cycle.*
- [slcd\\_lowpower\\_behavior](#) `lowPowerBehavior`  
*SLCD behavior in low power mode.*
- `uint32_t` [slcdLowPinEnabled](#)  
*Setting enabled SLCD pin 0 ~ pin 31.*
- `uint32_t` [slcdHighPinEnabled](#)  
*Setting enabled SLCD pin 32 ~ pin 63.*
- `uint32_t` [backPlaneLowPin](#)  
*Setting back plane pin 0 ~ pin 31.*
- `uint32_t` [backPlaneHighPin](#)  
*Setting back plane pin 32 ~ pin 63.*
- [slcd\\_fault\\_detect\\_config\\_t](#) \* `faultConfig`  
*Fault frame detection configure.*

## Enumeration Type Documentation

### 25.3.3.0.0.42 Field Documentation

**25.3.3.0.0.42.1** `slcd_power_supply_option_t slcd_config_t::powerSupply`

**25.3.3.0.0.42.2** `slcd_regulated_voltage_trim_t slcd_config_t::voltageTrim`

**25.3.3.0.0.42.3** `slcd_clock_config_t* slcd_config_t::clkConfig`

**25.3.3.0.0.42.4** `slcd_display_mode_t slcd_config_t::displayMode`

**25.3.3.0.0.42.5** `slcd_load_adjust_t slcd_config_t::loadAdjust`

**25.3.3.0.0.42.6** `slcd_duty_cycle_t slcd_config_t::dutyCycle`

**25.3.3.0.0.42.7** `slcd_lowpower_behavior slcd_config_t::lowPowerBehavior`

**25.3.3.0.0.42.8** `uint32_t slcd_config_t::slcdLowPinEnabled`

Setting bit *n* to 1 means enable pin *n*.

**25.3.3.0.0.42.9** `uint32_t slcd_config_t::slcdHighPinEnabled`

Setting bit *n* to 1 means enable pin (*n* + 32).

**25.3.3.0.0.42.10** `uint32_t slcd_config_t::backPlaneLowPin`

Setting bit *n* to 1 means setting pin *n* as back plane. It should never have the same bit setting as the frontPlane Pin.

**25.3.3.0.0.42.11** `uint32_t slcd_config_t::backPlaneHighPin`

Setting bit *n* to 1 means setting pin (*n* + 32) as back plane. It should never have the same bit setting as the frontPlane Pin.

**25.3.3.0.0.42.12** `slcd_fault_detect_config_t* slcd_config_t::faultConfig`

If not requirement, set to NULL.

## 25.4 Macro Definition Documentation

**25.4.1** `#define FSL_SLCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`

## 25.5 Enumeration Type Documentation

**25.5.1** `enum slcd_power_supply_option_t`

Enumerator

*kSLCD\_InternalVll3UseChargePump* VLL3 connected to VDD internally, charge pump is used to generate VLL1 and VLL2.

***kSLCD\_ExternalVll3UseResistorBiasNetwork*** VLL3 is driven externally and resistor bias network is used to generate VLL1 and VLL2.

***kSLCD\_ExteranlVll3UseChargePump*** VLL3 is driven externally and charge pump is used to generate VLL1 and VLL2.

***kSLCD\_InternalVll1UseChargePump*** VIREG is connected to VLL1 internally and charge pump is used to generate VLL2 and VLL3.

## 25.5.2 enum slcd\_regulated\_voltage\_trim\_t

Enumerator

|  |                                 |
|--|---------------------------------|
| <b><i>kSLCD_RegulatedVolatgeTrim00</i></b> | Increase the voltage to 0.91 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim01</i></b> | Increase the voltage to 1.01 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim02</i></b> | Increase the voltage to 0.96 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim03</i></b> | Increase the voltage to 1.06 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim04</i></b> | Increase the voltage to 0.93 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim05</i></b> | Increase the voltage to 1.02 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim06</i></b> | Increase the voltage to 0.98 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim07</i></b> | Increase the voltage to 1.08 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim08</i></b> | Increase the voltage to 0.92 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim09</i></b> | Increase the voltage to 1.02 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim10</i></b> | Increase the voltage to 0.97 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim11</i></b> | Increase the voltage to 1.07 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim12</i></b> | Increase the voltage to 0.94 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim13</i></b> | Increase the voltage to 1.05 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim14</i></b> | Increase the voltage to 0.99 V. |
| <b><i>kSLCD_RegulatedVolatgeTrim15</i></b> | Increase the voltage to 1.09 V. |

## 25.5.3 enum slcd\_load\_adjust\_t

Adjust the LCD glass capacitance if resistor bias network is enabled: ***kSLCD\_LowLoadOrFastestClkSrc*** - Low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,VLL2,Vcap1 and Vcap2 pins) ***kSLCD\_LowLoadOrIntermediateClkSrc*** - low load (LCD glass capacitance 2000pF or lower. LCD or GPIO function can be used on VLL1,VLL2,Vcap1 and Vcap2 pins) ***kSLCD\_HighLoadOrIntermediateClkSrc*** - high load (LCD glass capacitance 8000pF or lower. LCD or GPIO function can be used on Vcap1 and Vcap2 pins) ***kSLCD\_HighLoadOrSlowestClkSrc*** - high load (LCD glass capacitance 8000pF or lower LCD or GPIO function can be used on Vcap1 and Vcap2 pins) Adjust clock for charge pump if charge pump is enabled: ***kSLCD\_LowLoadOrFastestClkSrc*** - Fasten clock source (LCD glass capacitance 8000pF or 4000pF or lower if Fast Frame Rate is set) ***kSLCD\_LowLoadOrIntermediateClkSrc*** - Intermediate clock source (LCD glass capacitance 4000pF or 2000pF or lower if Fast Frame Rate is set) ***kSLCD\_HighLoadOrIntermediateClkSrc*** - Intermediate clock source

## Enumeration Type Documentation

(LCD glass capacitance 2000pF or 1000pF or lower if Fast Frame Rate is set) *kSLCD\_HighLoadOrSlowestClkSrc* - slowest clock source (LCD glass capacitance 1000pF or 500pF or lower if Fast Frame Rate is set)

Enumerator

*kSLCD\_LowLoadOrFastestClkSrc* Adjust in low load or selects fastest clock.  
*kSLCD\_LowLoadOrIntermediateClkSrc* Adjust in low load or selects intermediate clock.  
*kSLCD\_HighLoadOrIntermediateClkSrc* Adjust in high load or selects intermediate clock.  
*kSLCD\_HighLoadOrSlowestClkSrc* Adjust in high load or selects slowest clock.

### 25.5.4 enum slcd\_clock\_src\_t

Enumerator

*kSLCD\_DefaultClk* Select default clock ERCLK32K.  
*kSLCD\_AlternateClk1* Select alternate clock source 1 : MCGIRCLK.

### 25.5.5 enum slcd\_alt\_clock\_div\_t

Enumerator

*kSLCD\_AltClkDivFactor1* No divide for alternate clock.  
*kSLCD\_AltClkDivFactor64* Divide alternate clock with factor 64.  
*kSLCD\_AltClkDivFactor256* Divide alternate clock with factor 256.  
*kSLCD\_AltClkDivFactor512* Divide alternate clock with factor 512.

### 25.5.6 enum slcd\_clock\_prescaler\_t

Enumerator

*kSLCD\_ClkPrescaler00* Prescaler 0.  
*kSLCD\_ClkPrescaler01* Prescaler 1.  
*kSLCD\_ClkPrescaler02* Prescaler 2.  
*kSLCD\_ClkPrescaler03* Prescaler 3.  
*kSLCD\_ClkPrescaler04* Prescaler 4.  
*kSLCD\_ClkPrescaler05* Prescaler 5.  
*kSLCD\_ClkPrescaler06* Prescaler 6.  
*kSLCD\_ClkPrescaler07* Prescaler 7.

### 25.5.7 enum slcd\_duty\_cycle\_t

Enumerator

*kSLCD\_1Div1DutyCycle* LCD use 1 BP 1/1 duty cycle.  
*kSLCD\_1Div2DutyCycle* LCD use 2 BP 1/2 duty cycle.  
*kSLCD\_1Div3DutyCycle* LCD use 3 BP 1/3 duty cycle.  
*kSLCD\_1Div4DutyCycle* LCD use 4 BP 1/4 duty cycle.  
*kSLCD\_1Div5DutyCycle* LCD use 5 BP 1/5 duty cycle.  
*kSLCD\_1Div6DutyCycle* LCD use 6 BP 1/6 duty cycle.  
*kSLCD\_1Div7DutyCycle* LCD use 7 BP 1/7 duty cycle.  
*kSLCD\_1Div8DutyCycle* LCD use 8 BP 1/8 duty cycle.

### 25.5.8 enum slcd\_phase\_type\_t

Enumerator

*kSLCD\_NoPhaseActivate* LCD waveform no phase activates.  
*kSLCD\_PhaseAActivate* LCD waveform phase A activates.  
*kSLCD\_PhaseBActivate* LCD waveform phase B activates.  
*kSLCD\_PhaseCAActivate* LCD waveform phase C activates.  
*kSLCD\_PhaseDAActivate* LCD waveform phase D activates.  
*kSLCD\_PhaseEActivate* LCD waveform phase E activates.  
*kSLCD\_PhaseFAActivate* LCD waveform phase F activates.  
*kSLCD\_PhaseGActivate* LCD waveform phase G activates.  
*kSLCD\_PhaseHActivate* LCD waveform phase H activates.

### 25.5.9 enum slcd\_phase\_index\_t

Enumerator

*kSLCD\_PhaseAIndex* LCD phase A bit index.  
*kSLCD\_PhaseBIndex* LCD phase B bit index.  
*kSLCD\_PhaseCIndex* LCD phase C bit index.  
*kSLCD\_PhaseDIndex* LCD phase D bit index.  
*kSLCD\_PhaseEIndex* LCD phase E bit index.  
*kSLCD\_PhaseFIndex* LCD phase F bit index.  
*kSLCD\_PhaseGIndex* LCD phase G bit index.  
*kSLCD\_PhaseHIndex* LCD phase H bit index.

### 25.5.10 enum slcd\_display\_mode\_t

Enumerator

*kSLCD\_NormalMode* LCD Normal display mode.

*kSLCD\_AlternateMode* LCD Alternate display mode. For four back planes or less.

*kSLCD\_BlankMode* LCD Blank display mode.

### 25.5.11 enum slcd\_blink\_mode\_t

Enumerator

*kSLCD\_BlankDisplayBlink* Display blank during the blink period.

*kSLCD\_AltDisplayBlink* Display alternate display during the blink period if duty cycle is lower than 5.

### 25.5.12 enum slcd\_blink\_rate\_t

Enumerator

*kSLCD\_BlinkRate00* SLCD blink rate is LCD clock/ $((2^{12}))$ .

*kSLCD\_BlinkRate01* SLCD blink rate is LCD clock/ $((2^{13}))$ .

*kSLCD\_BlinkRate02* SLCD blink rate is LCD clock/ $((2^{14}))$ .

*kSLCD\_BlinkRate03* SLCD blink rate is LCD clock/ $((2^{15}))$ .

*kSLCD\_BlinkRate04* SLCD blink rate is LCD clock/ $((2^{16}))$ .

*kSLCD\_BlinkRate05* SLCD blink rate is LCD clock/ $((2^{17}))$ .

*kSLCD\_BlinkRate06* SLCD blink rate is LCD clock/ $((2^{18}))$ .

*kSLCD\_BlinkRate07* SLCD blink rate is LCD clock/ $((2^{19}))$ .

### 25.5.13 enum slcd\_fault\_detect\_clock\_prescaler\_t

Enumerator

*kSLCD\_FaultSampleFreqDivider1* Fault detect sample clock frequency is 1/1 bus clock.

*kSLCD\_FaultSampleFreqDivider2* Fault detect sample clock frequency is 1/2 bus clock.

*kSLCD\_FaultSampleFreqDivider4* Fault detect sample clock frequency is 1/4 bus clock.

*kSLCD\_FaultSampleFreqDivider8* Fault detect sample clock frequency is 1/8 bus clock.

*kSLCD\_FaultSampleFreqDivider16* Fault detect sample clock frequency is 1/16 bus clock.

*kSLCD\_FaultSampleFreqDivider32* Fault detect sample clock frequency is 1/32 bus clock.

*kSLCD\_FaultSampleFreqDivider64* Fault detect sample clock frequency is 1/64 bus clock.

*kSLCD\_FaultSampleFreqDivider128* Fault detect sample clock frequency is 1/128 bus clock.



### 25.5.14 enum slcd\_fault\_detect\_sample\_window\_width\_t

Enumerator

- kSLCD\_FaultDetectWindowWidth4SampleClk* Sample window width is 4 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth8SampleClk* Sample window width is 8 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth16SampleClk* Sample window width is 16 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth32SampleClk* Sample window width is 32 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth64SampleClk* Sample window width is 64 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth128SampleClk* Sample window width is 128 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth256SampleClk* Sample window width is 256 sample clock cycles.
- kSLCD\_FaultDetectWindowWidth512SampleClk* Sample window width is 512 sample clock cycles.

### 25.5.15 enum slcd\_interrupt\_enable\_t

Enumerator

- kSLCD\_FaultDetectCompleteInterrupt* SLCD fault detection complete interrupt source.

### 25.5.16 enum slcd\_lowpower\_behavior

Enumerator

- kSLCD\_EnabledInWaitStop* SLCD works in wait and stop mode.
- kSLCD\_EnabledInWaitOnly* SLCD works in wait mode and is disabled in stop mode.
- kSLCD\_EnabledInStopOnly* SLCD works in stop mode and is disabled in wait mode.
- kSLCD\_DisabledInWaitStop* SLCD is disabled in stop mode and wait mode.

## 25.6 Function Documentation

### 25.6.1 void SLCD\_Init ( LCD\_Type \* *base*, slcd\_config\_t \* *configure* )

Parameters

---

## Function Documentation

|                  |  |
|------------------|--|
| <i>base</i>      | SLCD peripheral base address.  |
| <i>configure</i> | SLCD configuration pointer. For the configuration structure, many parameters have the default setting and the SLCD_Getdefaultconfig() is provided to get them. Use it verified for their applications. The others have no default settings such as "clkConfig" and must be provided by the application before calling the <a href="#">SLCD_Init()</a> API. |

### 25.6.2 void SLCD\_Deinit ( LCD\_Type \* *base* )

Parameters

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

### 25.6.3 void SLCD\_GetDefaultConfig ( slcd\_config\_t \* *configure* )

The purpose of this API is to get default parameters of the configuration structure for the [SLCD\\_Init\(\)](#). Use these initialized parameters unchanged in [SLCD\\_Init\(\)](#), or modify some fields of the structure before the calling [SLCD\\_Init\(\)](#). All default parameters of the configure structure are listed:

```
config.displayMode      = kSLCD_NormalMode; // SLCD normal mode
config.powerSupply      = kSLCD_InternalV113UseChargePump; // Use charge
                        pump internal VLL3
config.voltageTrim      = kSLCD_RegulatedVolatgeTrim00;
config.lowPowerBehavior = kSLCD_EnabledInWaitStop; // Work on low power mode
config.interruptSrc     = 0; // No interrupt source is enabled
config.faultConfig      = NULL; // Fault detection is disabled
config.frameFreqIntEnable = false;
```

Parameters

|                  |   |
|------------------|---|
| <i>configure</i> | The SLCD configuration structure pointer. |
|------------------|---|

### 25.6.4 static void SLCD\_StartDisplay ( LCD\_Type \* *base* ) [inline], [static]

Parameters

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

### 25.6.5 static void SLCD\_StopDisplay ( LCD\_Type \* *base* ) [inline], [static]

There is no waveform generator and all enabled pins only output a low value.

Parameters

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

### 25.6.6 void SLCD\_StartBlinkMode ( LCD\_Type \* *base*, slcd\_blink\_mode\_t *mode*, slcd\_blink\_rate\_t *rate* )

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | SLCD peripheral base address. |
| <i>mode</i> | SLCD blink mode.              |
| <i>rate</i> | SLCD blink rate.              |

### 25.6.7 static void SLCD\_StopBlinkMode ( LCD\_Type \* *base* ) [inline], [static]

Parameters

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

### 25.6.8 static void SLCD\_SetBackPlanePhase ( LCD\_Type \* *base*, uint32\_t *pinIdx*, slcd\_phase\_type\_t *phase* ) [inline], [static]

This function sets the SLCD back plane pin phase. "kSLCD\_PhaseXActivate" setting means the phase X is active for the back plane pin. "kSLCD\_NoPhaseActivate" setting means there is no phase active for the back plane pin. register value. For example, set the back plane pin 20 for phase A:

```
* SLCD_SetBackPlanePhase(LCD, 20, kSLCD_PhaseAActivate);
*
```

## Function Documentation

### Parameters

|                |  |
|----------------|--|
| <i>base</i>    | SLCD peripheral base address.                  |
| <i>pinIndx</i> | SLCD back plane pin index. Range from 0 to 63. |
| <i>phase</i>   | The phase activates for the back plane pin.    |

### 25.6.9 static void SLCD\_SetFrontPlaneSegments ( LCD\_Type \* *base*, uint32\_t *pinIndx*, uint8\_t *operation* ) [inline], [static]

This function sets the SLCD front plane segment on or off operation. Each bit turns on or off the segments associated with the front plane pin in the following pattern: HGFEDCBA (most significant bit controls segment H and least significant bit controls segment A). For example, turn on the front plane pin 20 for phase B and phase C:

```
* SLCD_SetFrontPlaneSegments(LCD, 20, (  
    kSLCD_PhaseBActivate | kSLCD_PhaseCActivate));  
*
```

### Parameters

|                  |   |
|------------------|---|
| <i>base</i>      | SLCD peripheral base address.   |
| <i>pinIndx</i>   | SLCD back plane pin index. Range from 0 to 63.  |
| <i>operation</i> | The operation for the segment on the front plane pin. This is a logical OR of the enumeration :: slcd_phase_type_t. |

### 25.6.10 static void SLCD\_SetFrontPlaneOnePhase ( LCD\_Type \* *base*, uint32\_t *pinIndx*, slcd\_phase\_index\_t *phaseIndx*, bool *enable* ) [inline], [static]

This function can be used to set one phase on or off for the front plane pin. It can be call many times to set the plane pin for different phase indexes. For example, turn on the front plane pin 20 for phase B and phase C:

```
* SLCD_SetFrontPlaneOnePhase(LCD, 20,  
    kSLCD_PhaseBIndex, true);  
* SLCD_SetFrontPlaneOnePhase(LCD, 20,  
    kSLCD_PhaseCIndex, true);  
*
```

## Parameters

|                  |  |
|------------------|--|
| <i>base</i>      | SLCD peripheral base address.  |
| <i>pinIndx</i>   | SLCD back plane pin index. Range from 0 to 63.   |
| <i>phaseIndx</i> | The phase bit index <a href="#">slcd_phase_index_t</a> .   |
| <i>enable</i>    | True to turn on the segment for phaseIndx phase false to turn off the segment for phaseIndx phase. |

### 25.6.11 static uint32\_t SLCD\_GetFaultDetectCounter ( LCD\_Type \* *base* ) [inline], [static]

This function gets the number of samples inside the fault detection sample window.

## Parameters

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

## Returns

The fault detect counter. The maximum return value is 255. If the maximum 255 returns, the overflow may happen. Reconfigure the fault detect sample window and fault detect clock prescaler for proper sampling.

### 25.6.12 void SLCD\_EnableInterrupts ( LCD\_Type \* *base*, uint32\_t *mask* )

For example, to enable fault detect complete interrupt and frame frequency interrupt, for FSL\_FEATURE\_SLCD\_HAS\_FRAME\_FREQUENCY\_INTERRUPT enabled case, do the following.

```
* SLCD_EnableInterrupts(LCD,  
* kSLCD_FaultDetectCompleteInterrupt | kSLCD_FrameFreqInterrupt);  
*
```

## Parameters

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

## Function Documentation

|             |  |
|-------------|--|
| <i>mask</i> | SLCD interrupts to enable. This is a logical OR of the enumeration :: slcd_interrupt_enable_t. |
|-------------|--|

### 25.6.13 void SLCD\_DisableInterrupts ( LCD\_Type \* *base*, uint32\_t *mask* )

For example, to disable fault detect complete interrupt and frame frequency interrupt, for FSL\_FEATURE\_SLCD\_HAS\_FRAME\_FREQUENCY\_INTERRUPT enabled case, do the following.

```
* SLCD_DisableInterrupts (LCD,  
* kSLCD_FaultDetectCompleteInterrupt | kSLCD_FrameFreqInterrupt);  
*
```

#### Parameters

|             |   |
|-------------|---|
| <i>base</i> | SLCD peripheral base address.   |
| <i>mask</i> | SLCD interrupts to disable. This is a logical OR of the enumeration :: slcd_interrupt_enable_t. |

### 25.6.14 uint32\_t SLCD\_GetInterruptStatus ( LCD\_Type \* *base* )

#### Parameters

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

#### Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: slcd\_interrupt\_enable\_t.

### 25.6.15 void SLCD\_ClearInterruptStatus ( LCD\_Type \* *base*, uint32\_t *mask* )

#### Parameters

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

|             |   |
|-------------|---|
| <i>mask</i> | SLCD interrupt source to be cleared. This is the logical OR of members of the enumeration :: <code>slcd_interrupt_enable_t</code> . |
|-------------|---|





## Chapter 26

# SMC: System Mode Controller Driver

### 26.1 Overview

The KSDK provides a Peripheral driver for the System Mode Controller (SMC) module of Kinetis devices. The SMC module is responsible for sequencing the system into and out of all low-power Stop and Run modes

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

### Enumerations

- enum `smc_power_mode_protection_t` {  
    `kSMC_AllowPowerModeVlp` = `SMC_PMPROT_AVLP_MASK`,  
    `kSMC_AllowPowerModeAll` }  
    *Power Modes Protection.*
- enum `smc_power_state_t` {  
    `kSMC_PowerStateRun` = `0x01U << 0U`,  
    `kSMC_PowerStateStop` = `0x01U << 1U`,  
    `kSMC_PowerStateVlpr` = `0x01U << 2U`,  
    `kSMC_PowerStateVlpw` = `0x01U << 3U`,  
    `kSMC_PowerStateVlps` = `0x01U << 4U` }  
    *Power Modes in PMSTAT.*
- enum `smc_run_mode_t` {  
    `kSMC_RunNormal` = `0U`,  
    `kSMC_RunVlpr` = `2U` }  
    *Run mode definition.*
- enum `smc_stop_mode_t` {  
    `kSMC_StopNormal` = `0U`,  
    `kSMC_StopVlps` = `2U` }  
    *Stop mode definition.*
- enum `smc_partial_stop_option_t` {  
    `kSMC_PartialStop` = `0U`,  
    `kSMC_PartialStop1` = `1U`,  
    `kSMC_PartialStop2` = `2U` }  
    *Partial STOP option.*
- enum `_smc_status` { `kStatus_SMC_StopAbort` = `MAKE_STATUS(kStatusGroup_POWER, 0)` }  
    *SMC configuration status.*

## Enumeration Type Documentation

### Driver version

- #define **FSL\_SMC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 2))  
*SMC driver version 2.0.2.*

### System mode controller APIs

- static void **SMC\_SetPowerModeProtection** (SMC\_Type \*base, uint8\_t allowedModes)  
*Configures all power mode protection settings.*
- static **smc\_power\_state\_t** **SMC\_GetPowerModeState** (SMC\_Type \*base)  
*Gets the current power mode status.*
- status\_t **SMC\_SetPowerModeRun** (SMC\_Type \*base)  
*Configure the system to RUN power mode.*
- status\_t **SMC\_SetPowerModeWait** (SMC\_Type \*base)  
*Configure the system to WAIT power mode.*
- status\_t **SMC\_SetPowerModeStop** (SMC\_Type \*base, **smc\_partial\_stop\_option\_t** option)  
*Configure the system to Stop power mode.*
- status\_t **SMC\_SetPowerModeVlpr** (SMC\_Type \*base)  
*Configure the system to VLPR power mode.*
- status\_t **SMC\_SetPowerModeVlpw** (SMC\_Type \*base)  
*Configure the system to VLPW power mode.*
- status\_t **SMC\_SetPowerModeVlps** (SMC\_Type \*base)  
*Configure the system to VLPS power mode.*

## 26.2 Macro Definition Documentation

### 26.2.1 #define FSL\_SMC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

## 26.3 Enumeration Type Documentation

### 26.3.1 enum smc\_power\_mode\_protection\_t

Enumerator

**kSMC\_AllowPowerModeVlp** Allow Very-Low-Power Mode.  
**kSMC\_AllowPowerModeAll** Allow all power mode.

### 26.3.2 enum smc\_power\_state\_t

Enumerator

**kSMC\_PowerStateRun** 0000\_0001 - Current power mode is RUN  
**kSMC\_PowerStateStop** 0000\_0010 - Current power mode is STOP  
**kSMC\_PowerStateVlpr** 0000\_0100 - Current power mode is VLPR  
**kSMC\_PowerStateVlpw** 0000\_1000 - Current power mode is VLPW  
**kSMC\_PowerStateVlps** 0001\_0000 - Current power mode is VLPS

### 26.3.3 enum smc\_run\_mode\_t

Enumerator

*kSMC\_RunNormal* normal RUN mode.  
*kSMC\_RunVlpr* Very-Low-Power RUN mode.

### 26.3.4 enum smc\_stop\_mode\_t

Enumerator

*kSMC\_StopNormal* Normal STOP mode.  
*kSMC\_StopVlps* Very-Low-Power STOP mode.

### 26.3.5 enum smc\_partial\_stop\_option\_t

Enumerator

*kSMC\_PartialStop* STOP - Normal Stop mode.  
*kSMC\_PartialStop1* Partial Stop with both system and bus clocks disabled.  
*kSMC\_PartialStop2* Partial Stop with system clock disabled and bus clock enabled.

### 26.3.6 enum \_smc\_status

Enumerator

*kStatus\_SMC\_StopAbort* Entering Stop mode is abort.

## 26.4 Function Documentation

### 26.4.1 static void SMC\_SetPowerModeProtection ( SMC\_Type \* *base*, uint8\_t *allowedModes* ) [inline], [static]

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

The allowed modes are passed as bit map, for example, to allow LLS and VLLS, use `SMC_SetPowerModeProtection(kSMC_AllowPowerModeVlls | kSMC_AllowPowerModeVlps)`. To allow all modes, use `SMC_SetPowerModeProtection(kSMC_AllowPowerModeAll)`.

## Function Documentation

### Parameters

|                     |                                    |
|---------------------|------------------------------------|
| <i>base</i>         | SMC peripheral base address.       |
| <i>allowedModes</i> | Bitmap of the allowed power modes. |

### 26.4.2 static smc\_power\_state\_t SMC\_GetPowerModeState ( SMC\_Type \* *base* ) [inline], [static]

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

### Parameters

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

### Returns

Current power mode status.

### 26.4.3 status\_t SMC\_SetPowerModeRun ( SMC\_Type \* *base* )

### Parameters

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

### Returns

SMC configuration error code.

### 26.4.4 status\_t SMC\_SetPowerModeWait ( SMC\_Type \* *base* )

## Parameters

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

## Returns

SMC configuration error code.

### 26.4.5 **status\_t SMC\_SetPowerModeStop ( SMC\_Type \* *base*, smc\_partial\_stop\_option\_t *option* )**

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SMC peripheral base address. |
| <i>option</i> | Partial Stop mode option.    |

## Returns

SMC configuration error code.

### 26.4.6 **status\_t SMC\_SetPowerModeVlpr ( SMC\_Type \* *base* )**

## Parameters

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

## Returns

SMC configuration error code.

### 26.4.7 **status\_t SMC\_SetPowerModeVlprw ( SMC\_Type \* *base* )**

## Parameters

---

## Function Documentation

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

Returns

SMC configuration error code.

### 26.4.8 **status\_t** SMC\_SetPowerModeVlps ( SMC\_Type \* *base* )

Parameters

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

Returns

SMC configuration error code.



## Chapter 27

### SPI: Serial Peripheral Interface Driver

#### 27.1 Overview

##### Modules

- [SPI DMA Driver](#)
- [SPI Driver](#)
- [SPI FreeRTOS driver](#)
- [SPI  \$\mu\$ COS/II driver](#)
- [SPI  \$\mu\$ COS/III driver](#)

## 27.2 SPI Driver

### 27.2.1 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.

### 27.2.2 Typical use case

#### 27.2.2.1 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);
    SPI_MasterTransferCreateHandle(SPI0, &spiHandle, SPI_UserCallback, NULL);

    // Prepare to send.
    xfer.txData = sendData;
    xfer.rxData = receiveBuff;
    xfer.dataSize = BUFFER_LEN;

    // Send out.
    SPI_MasterTransferNonBlocking(SPI0, &spiHandle, &xfer);
```



```

    // Wait send finished.
    while (!isFinished)
    {
    }

    // ...
}

```

### 27.2.2.2 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)
{
    //...

    SPI_MasterGetDefaultConfig(&masterConfig);
    SPI_MasterInit(SPI0, &masterConfig);

    // Sets up the DMA.
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, SPI_TX_DMA_CHANNEL, SPI_TX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, SPI_TX_DMA_CHANNEL);
    DMAMUX_SetSource(DMAMUX0, SPI_RX_DMA_CHANNEL, SPI_RX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, SPI_RX_DMA_CHANNEL);

    DMA_Init(DMA0);

    /* Creates the DMA handle. */
    DMA_CreateHandle(&g_spiTxDmaHandle, DMA0, SPI_TX_DMA_CHANNEL);
    DMA_CreateHandle(&g_spiRxDmaHandle, DMA0, SPI_RX_DMA_CHANNEL);

    SPI_MasterTransferCreateHandleDMA(SPI0, spiHandle, &g_spiTxDmaHandle,
        &g_spiRxDmaHandle, SPI_UserCallback, NULL);

    // Prepares to send.
    xfer.txData = sendData;
    xfer.rxData = receiveBuff;
    xfer.dataSize = BUFFER_LEN;

    // Sends out.
    SPI_MasterTransferDMA(SPI0, &spiHandle, &xfer);

    // Waits for send to complete.
    while (!isFinished)
    {
    }

    // ...
}

```

## SPI Driver

### Data Structures

- 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\\_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.*

### Typedefs

- typedef [spi\\_master\\_handle\\_t](#) [spi\\_slave\\_handle\\_t](#)  
*Slave handle is the same with master handle.*
- 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 master callback for finished transmit.*

### Enumerations

- enum [\\_spi\\_status](#) {  
    [kStatus\\_SPI\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_SPI, 0),  
    [kStatus\\_SPI\\_Idle](#) = MAKE\_STATUS(kStatusGroup\_SPI, 1),  
    [kStatus\\_SPI\\_Error](#) = MAKE\_STATUS(kStatusGroup\_SPI, 2) }  
*Return status for the SPI driver.*
- 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\\_shift\\_direction\\_t](#) {  
    [kSPI\\_MsbFirst](#) = 0x0U,  
    [kSPI\\_LsbFirst](#) }

- SPI data shifter direction options.*
  - enum `spi_ss_output_mode_t` {  
`kSPI_SlaveSelectAsGpio` = 0x0U,  
`kSPI_SlaveSelectFaultInput` = 0x2U,  
`kSPI_SlaveSelectAutomaticOutput` = 0x3U }
- SPI slave select output mode options.*
  - enum `spi_pin_mode_t` {  
`kSPI_PinModeNormal` = 0x0U,  
`kSPI_PinModeInput` = 0x1U,  
`kSPI_PinModeOutput` = 0x3U }
- SPI pin mode options.*
  - enum `spi_data_bitcount_mode_t` {  
`kSPI_8BitMode` = 0x0U,  
`kSPI_16BitMode` }
- SPI data length mode options.*
  - enum `_spi_interrupt_enable` {  
`kSPI_RxFullAndModfInterruptEnable` = 0x1U,  
`kSPI_TxEmptyInterruptEnable` = 0x2U,  
`kSPI_MatchInterruptEnable` = 0x4U }
- SPI interrupt sources.*
  - enum `_spi_flags` {  
`kSPI_RxBufferFullFlag` = SPI\_S\_SPRF\_MASK,  
`kSPI_MatchFlag` = SPI\_S\_SPMF\_MASK,  
`kSPI_TxBufferEmptyFlag` = SPI\_S\_SPTEF\_MASK,  
`kSPI_ModeFaultFlag` = SPI\_S\_MODF\_MASK }
- SPI status flags.*

## Driver version

- #define `FSL_SPI_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 1))  
*SPI driver version 2.0.1.*

## Initialization and deinitialization

- void `SPI_MasterGetDefaultConfig` (`spi_master_config_t` \*config)  
*Sets the SPI master configuration structure to default values.*
- void `SPI_MasterInit` (`SPI_Type` \*base, const `spi_master_config_t` \*config, `uint32_t` srcClock\_Hz)  
*Initializes the SPI with master configuration.*
- void `SPI_SlaveGetDefaultConfig` (`spi_slave_config_t` \*config)  
*Sets the SPI slave configuration structure to default values.*
- void `SPI_SlaveInit` (`SPI_Type` \*base, const `spi_slave_config_t` \*config)  
*Initializes the SPI with slave configuration.*
- void `SPI_Deinit` (`SPI_Type` \*base)  
*De-initializes the SPI.*
- static void `SPI_Enable` (`SPI_Type` \*base, bool enable)  
*Enables or disables the SPI.*

## SPI Driver

### Status

- uint32\_t [SPI\\_GetStatusFlags](#) (SPI\_Type \*base)  
*Gets the status flag.*

### Interrupts

- void [SPI\\_EnableInterrupts](#) (SPI\_Type \*base, uint32\_t mask)  
*Enables the interrupt for the SPI.*
- void [SPI\\_DisableInterrupts](#) (SPI\_Type \*base, uint32\_t mask)  
*Disables the interrupt for the SPI.*

### DMA Control

- static uint32\_t [SPI\\_GetDataRegisterAddress](#) (SPI\_Type \*base)  
*Gets the SPI tx/rx data register address.*

### Bus Operations

- void [SPI\\_MasterSetBaudRate](#) (SPI\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the baud rate for SPI transfer.*
- static void [SPI\\_SetMatchData](#) (SPI\_Type \*base, uint32\_t matchData)  
*Sets the match data for SPI.*
- void [SPI\\_WriteBlocking](#) (SPI\_Type \*base, uint8\_t \*buffer, size\_t size)  
*Sends a buffer of data bytes using a blocking method.*
- void [SPI\\_WriteData](#) (SPI\_Type \*base, uint16\_t data)  
*Writes a data into the SPI data register.*
- uint16\_t [SPI\\_ReadData](#) (SPI\_Type \*base)  
*Gets a data from the SPI data register.*

### Transactional

- void [SPI\\_MasterTransferCreateHandle](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle, [spi\\_master\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SPI master handle.*
- status\_t [SPI\\_MasterTransferBlocking](#) (SPI\_Type \*base, [spi\\_transfer\\_t](#) \*xfer)  
*Transfers a block of data using a polling method.*
- status\_t [SPI\\_MasterTransferNonBlocking](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle, [spi\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SPI interrupt transfer.*
- status\_t [SPI\\_MasterTransferGetCount](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the bytes of the SPI interrupt transferred.*
- void [SPI\\_MasterTransferAbort](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle)  
*Aborts an SPI transfer using interrupt.*

- void [SPI\\_MasterTransferHandleIRQ](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle)  
*Interrupts the handler for the SPI.*
- void [SPI\\_SlaveTransferCreateHandle](#) (SPI\_Type \*base, spi\_slave\_handle\_t \*handle, spi\_slave\_callback\_t callback, void \*userData)  
*Initializes the SPI slave handle.*
- static status\_t [SPI\\_SlaveTransferNonBlocking](#) (SPI\_Type \*base, spi\_slave\_handle\_t \*handle, spi\_transfer\_t \*xfer)  
*Performs a non-blocking SPI slave interrupt transfer.*
- static status\_t [SPI\\_SlaveTransferGetCount](#) (SPI\_Type \*base, spi\_slave\_handle\_t \*handle, size\_t \*count)  
*Gets the bytes of the SPI interrupt transferred.*
- static void [SPI\\_SlaveTransferAbort](#) (SPI\_Type \*base, spi\_slave\_handle\_t \*handle)  
*Aborts an SPI slave transfer using interrupt.*
- void [SPI\\_SlaveTransferHandleIRQ](#) (SPI\_Type \*base, spi\_slave\_handle\_t \*handle)  
*Interrupts a handler for the SPI slave.*

## 27.2.3 Data Structure Documentation

### 27.2.3.1 struct spi\_master\_config\_t

#### Data Fields

- bool [enableMaster](#)  
*Enable SPI at initialization time.*
- bool [enableStopInWaitMode](#)  
*SPI stop in wait mode.*
- [spi\\_clock\\_polarity\\_t](#) polarity  
*Clock polarity.*
- [spi\\_clock\\_phase\\_t](#) phase  
*Clock phase.*
- [spi\\_shift\\_direction\\_t](#) direction  
*MSB or LSB.*
- [spi\\_ss\\_output\\_mode\\_t](#) outputMode  
*SS pin setting.*
- [spi\\_pin\\_mode\\_t](#) pinMode  
*SPI pin mode select.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud Rate for SPI in Hz.*

### 27.2.3.2 struct spi\_slave\_config\_t

#### Data Fields

- bool [enableSlave](#)  
*Enable SPI at initialization time.*
- bool [enableStopInWaitMode](#)  
*SPI stop in wait mode.*
- [spi\\_clock\\_polarity\\_t](#) polarity

## SPI Driver

- *Clock polarity.*  
[spi\\_clock\\_phase\\_t](#) phase
- *Clock phase.*  
[spi\\_shift\\_direction\\_t](#) direction  
*MSB or LSB.*

### 27.2.3.3 struct spi\_transfer\_t

#### Data Fields

- [uint8\\_t](#) \* [txData](#)  
*Send buffer.*
- [uint8\\_t](#) \* [rxData](#)  
*Receive buffer.*
- [size\\_t](#) [dataSize](#)  
*Transfer bytes.*
- [uint32\\_t](#) [flags](#)  
*SPI control flag, useless to SPI.*

#### 27.2.3.3.0.43 Field Documentation

##### 27.2.3.3.0.43.1 [uint32\\_t spi\\_transfer\\_t::flags](#)

### 27.2.3.4 struct \_spi\_master\_handle

#### Data Fields

- [uint8\\_t](#) \*volatile [txData](#)  
*Transfer buffer.*
- [uint8\\_t](#) \*volatile [rxData](#)  
*Receive buffer.*
- volatile [size\\_t](#) [txRemainingBytes](#)  
*Send data remaining in bytes.*
- volatile [size\\_t](#) [rxRemainingBytes](#)  
*Receive data remaining in bytes.*
- volatile [uint32\\_t](#) [state](#)  
*SPI internal state.*
- [size\\_t](#) [transferSize](#)  
*Bytes to be transferred.*
- [uint8\\_t](#) [bytePerFrame](#)  
*SPI mode, 2bytes or 1byte in a frame.*
- [uint8\\_t](#) [watermark](#)  
*Watermark value for SPI transfer.*
- [spi\\_master\\_callback\\_t](#) [callback](#)  
*SPI callback.*
- void \* [userData](#)  
*Callback parameter.*

## 27.2.4 Macro Definition Documentation

27.2.4.1 **#define FSL\_SPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))**

27.2.4.2 **#define SPI\_DUMMYDATA (0xFFU)**

## 27.2.5 Enumeration Type Documentation

### 27.2.5.1 enum \_spi\_status

Enumerator

*kStatus\_SPI\_Busy* SPI bus is busy.

*kStatus\_SPI\_Idle* SPI is idle.

*kStatus\_SPI\_Error* SPI error.

### 27.2.5.2 enum spi\_clock\_polarity\_t

Enumerator

*kSPI\_ClockPolarityActiveHigh* Active-high SPI clock (idles low).

*kSPI\_ClockPolarityActiveLow* Active-low SPI clock (idles high).

### 27.2.5.3 enum spi\_clock\_phase\_t

Enumerator

*kSPI\_ClockPhaseFirstEdge* First edge on SPSCCK occurs at the middle of the first cycle of a data transfer.

*kSPI\_ClockPhaseSecondEdge* First edge on SPSCCK occurs at the start of the first cycle of a data transfer.

### 27.2.5.4 enum spi\_shift\_direction\_t

Enumerator

*kSPI\_MsbFirst* Data transfers start with most significant bit.

*kSPI\_LsbFirst* Data transfers start with least significant bit.

## SPI Driver

### 27.2.5.5 enum spi\_ss\_output\_mode\_t

Enumerator

*kSPI\_SlaveSelectAsGpio* Slave select pin configured as GPIO.

*kSPI\_SlaveSelectFaultInput* Slave select pin configured for fault detection.

*kSPI\_SlaveSelectAutomaticOutput* Slave select pin configured for automatic SPI output.

### 27.2.5.6 enum spi\_pin\_mode\_t

Enumerator

*kSPI\_PinModeNormal* Pins operate in normal, single-direction mode.

*kSPI\_PinModeInput* Bidirectional mode. Master: MOSI pin is input; Slave: MISO pin is input.

*kSPI\_PinModeOutput* Bidirectional mode. Master: MOSI pin is output; Slave: MISO pin is output.

### 27.2.5.7 enum spi\_data\_bitcount\_mode\_t

Enumerator

*kSPI\_8BitMode* 8-bit data transmission mode

*kSPI\_16BitMode* 16-bit data transmission mode

### 27.2.5.8 enum \_spi\_interrupt\_enable

Enumerator

*kSPI\_RxFullAndModfInterruptEnable* Receive buffer full (SPRF) and mode fault (MODF) interrupt.

*kSPI\_TxEmptyInterruptEnable* Transmit buffer empty interrupt.

*kSPI\_MatchInterruptEnable* Match interrupt.

### 27.2.5.9 enum \_spi\_flags

Enumerator

*kSPI\_RxBufferFullFlag* Read buffer full flag.

*kSPI\_MatchFlag* Match flag.

*kSPI\_TxBufferEmptyFlag* Transmit buffer empty flag.

*kSPI\_ModeFaultFlag* Mode fault flag.



## 27.2.6 Function Documentation

### 27.2.6.1 void SPI\_MasterGetDefaultConfig ( spi\_master\_config\_t \* *config* )

The purpose of this API is to get the configuration structure initialized for use in [SPI\\_MasterInit\(\)](#). User may use the initialized structure unchanged in [SPI\\_MasterInit\(\)](#), or modify some fields of the structure before calling [SPI\\_MasterInit\(\)](#). After calling this API, the master is ready to transfer. Example:

```
spi_master_config_t config;
SPI_MasterGetDefaultConfig(&config);
```

Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>config</i> | pointer to master config structure |
|---------------|------------------------------------|

### 27.2.6.2 void SPI\_MasterInit ( SPI\_Type \* *base*, const spi\_master\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

The configuration structure can be filled by user from scratch, or be set with default values by [SPI\\_MasterGetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
spi_master_config_t config = {
    .baudRate_Bps = 400000,
    ...
};
SPI_MasterInit(SPI0, &config);
```

Parameters

|                    |   |
|--------------------|---|
| <i>base</i>        | SPI base pointer                          |
| <i>config</i>      | pointer to master configuration structure |
| <i>srcClock_Hz</i> | Source clock frequency.                   |

### 27.2.6.3 void SPI\_SlaveGetDefaultConfig ( spi\_slave\_config\_t \* *config* )

The purpose of this API is to get the configuration structure initialized for use in [SPI\\_SlaveInit\(\)](#). Modify some fields of the structure before calling [SPI\\_SlaveInit\(\)](#). Example:

```
spi_slave_config_t config;
SPI_SlaveGetDefaultConfig(&config);
```

## SPI Driver

### Parameters

|               |  |
|---------------|--|
| <i>config</i> | pointer to slave configuration structure |
|---------------|--|

#### 27.2.6.4 void SPI\_SlaveInit ( SPI\_Type \* *base*, const spi\_slave\_config\_t \* *config* )

The configuration structure can be filled by user from scratch or be set with default values by [SPI\\_Slave-GetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
spi_slave_config_t config = {  
.polarity = kSPIClockPolarity_ActiveHigh;  
.phase = kSPIClockPhase_FirstEdge;  
.direction = kSPIMsbFirst;  
...  
};  
SPI_MasterInit(SPI0, &config);
```

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI base pointer                          |
| <i>config</i> | pointer to master configuration structure |

#### 27.2.6.5 void SPI\_Deinit ( SPI\_Type \* *base* )

Calling this API resets the SPI module, gates the SPI clock. The SPI module can't work unless calling the SPI\_MasterInit/SPI\_SlaveInit to initialize module.

### Parameters

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

#### 27.2.6.6 static void SPI\_Enable ( SPI\_Type \* *base*, bool *enable* ) [inline], [static]

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI base pointer                                    |
| <i>enable</i> | pass true to enable module, false to disable module |

#### 27.2.6.7 uint32\_t SPI\_GetStatusFlags ( SPI\_Type \* *base* )

## Parameters

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

## Returns

SPI Status, use status flag to AND [\\_spi\\_flags](#) could get the related status.

### 27.2.6.8 void SPI\_EnableInterrupts ( SPI\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | SPI base pointer   |
| <i>mask</i> | SPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kSPI_RxFullAndModfInterruptEnable</li> <li>• kSPI_TxEmptyInterruptEnable</li> <li>• kSPI_MatchInterruptEnable</li> <li>• kSPI_RxFifoNearFullInterruptEnable</li> <li>• kSPI_TxFifoNearEmptyInterruptEnable</li> </ul> |

### 27.2.6.9 void SPI\_DisableInterrupts ( SPI\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | SPI base pointer   |
| <i>mask</i> | SPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kSPI_RxFullAndModfInterruptEnable</li> <li>• kSPI_TxEmptyInterruptEnable</li> <li>• kSPI_MatchInterruptEnable</li> <li>• kSPI_RxFifoNearFullInterruptEnable</li> <li>• kSPI_TxFifoNearEmptyInterruptEnable</li> </ul> |

### 27.2.6.10 static uint32\_t SPI\_GetDataRegisterAddress ( SPI\_Type \* *base* ) [inline], [static]

This API is used to provide a transfer address for the SPI DMA transfer configuration.

## SPI Driver

### Parameters

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

### Returns

data register address

#### 27.2.6.11 void SPI\_MasterSetBaudRate ( SPI\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

This is only used in master.

### Parameters

|                     |                                   |
|---------------------|-----------------------------------|
| <i>base</i>         | SPI base pointer                  |
| <i>baudRate_Bps</i> | baud rate needed in Hz.           |
| <i>srcClock_Hz</i>  | SPI source clock frequency in Hz. |

#### 27.2.6.12 static void SPI\_SetMatchData ( SPI\_Type \* *base*, uint32\_t *matchData* ) [inline], [static]

The match data is a hardware comparison value. When the value received in the SPI receive data buffer equals the hardware comparison value, the SPI Match Flag in the S register (S[SPMF]) sets. This can also generate an interrupt if the enable bit sets.

### Parameters

|                  |                  |
|------------------|------------------|
| <i>base</i>      | SPI base pointer |
| <i>matchData</i> | Match data.      |

#### 27.2.6.13 void SPI\_WriteBlocking ( SPI\_Type \* *base*, uint8\_t \* *buffer*, size\_t *size* )

### Note

This function blocks via polling until all bytes have been sent.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | SPI base pointer                 |
| <i>buffer</i> | The data bytes to send           |
| <i>size</i>   | The number of data bytes to send |

**27.2.6.14 void SPI\_WriteData ( SPI\_Type \* *base*, uint16\_t *data* )**

## Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | SPI base pointer   |
| <i>data</i> | needs to be write. |

**27.2.6.15 uint16\_t SPI\_ReadData ( SPI\_Type \* *base* )**

## Parameters

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

## Returns

Data in the register.

**27.2.6.16 void SPI\_MasterTransferCreateHandle ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle*, spi\_master\_callback\_t *callback*, void \* *userData* )**

This function initializes the SPI master handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

## Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | SPI peripheral base address. |
| <i>handle</i>   | SPI handle pointer.          |
| <i>callback</i> | Callback function.           |

## SPI Driver

|                 |            |
|-----------------|------------|
| <i>userData</i> | User data. |
|-----------------|------------|

### 27.2.6.17 **status\_t SPI\_MasterTransferBlocking ( SPI\_Type \* *base*, spi\_transfer\_t \* *xfer* )**

#### Parameters

|             |  |
|-------------|--|
| <i>base</i> | SPI base pointer                       |
| <i>xfer</i> | pointer to spi_xfer_config_t structure |

#### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer. |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.     |

### 27.2.6.18 **status\_t SPI\_MasterTransferNonBlocking ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* )**

#### Note

The API immediately returns after transfer initialization is finished. Call SPI\_GetStatusIRQ() to get the transfer status.

If using the SPI with FIFO for the interrupt transfer, the transfer size is the integer times of the watermark. Otherwise, the last data may be lost because it cannot generate an interrupt request.

Users can also call the functional API to get the last received data.

#### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI peripheral base address.   |
| <i>handle</i> | pointer to spi_master_handle_t structure which stores the transfer state |
| <i>xfer</i>   | pointer to spi_xfer_config_t structure                                   |

#### Return values

|                        |                                |
|------------------------|--------------------------------|
| <i>kStatus_Success</i> | Successfully start a transfer. |
|------------------------|--------------------------------|

|                                |   |
|--------------------------------|---|
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_SPI_Busy</i>        | SPI is not idle, is running another transfer. |

#### 27.2.6.19 **status\_t SPI\_MasterTransferGetCount ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI peripheral base address.                                      |
| <i>handle</i> | Pointer to SPI transfer handle, this should be a static variable. |
| <i>count</i>  | Transferred bytes of SPI master.                                  |

Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_SPI_Success</i>          | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

#### 27.2.6.20 **void SPI\_MasterTransferAbort ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle* )**

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI peripheral base address.                                      |
| <i>handle</i> | Pointer to SPI transfer handle, this should be a static variable. |

#### 27.2.6.21 **void SPI\_MasterTransferHandleIRQ ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle* )**

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI peripheral base address.  |
| <i>handle</i> | pointer to spi_master_handle_t structure which stores the transfer state. |

**27.2.6.22 void SPI\_SlaveTransferCreateHandle ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle*, spi\_slave\_callback\_t *callback*, void \* *userData* )**

This function initializes the SPI slave handle which can be used for other SPI slave transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.



## Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | SPI peripheral base address. |
| <i>handle</i>   | SPI handle pointer.          |
| <i>callback</i> | Callback function.           |
| <i>userData</i> | User data.                   |

**27.2.6.23 static status\_t SPI\_SlaveTransferNonBlocking ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* ) [inline], [static]**

## Note

The API returns immediately after the transfer initialization is finished. Call SPI\_GetStatusIRQ() to get the transfer status.

If using the SPI with FIFO for the interrupt transfer, the transfer size is the integer times the watermark. Otherwise, the last data may be lost because it cannot generate an interrupt request. Call the functional API to get the last several receive data.

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI peripheral base address.   |
| <i>handle</i> | pointer to spi_master_handle_t structure which stores the transfer state |
| <i>xfer</i>   | pointer to spi_xfer_config_t structure                                   |

## Return values

|                                |   |
|--------------------------------|---|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_SPI_Busy</i>        | SPI is not idle, is running another transfer. |

**27.2.6.24 static status\_t SPI\_SlaveTransferGetCount ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle*, size\_t \* *count* ) [inline], [static]**

## Parameters

## SPI Driver

|               |   |
|---------------|---|
| <i>base</i>   | SPI peripheral base address.                                      |
| <i>handle</i> | Pointer to SPI transfer handle, this should be a static variable. |
| <i>count</i>  | Transferred bytes of SPI slave.                                   |

### Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_SPI_Success</i>          | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

**27.2.6.25 static void SPI\_SlaveTransferAbort ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle* ) [inline], [static]**

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI peripheral base address.                                      |
| <i>handle</i> | Pointer to SPI transfer handle, this should be a static variable. |

**27.2.6.26 void SPI\_SlaveTransferHandleIRQ ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle* )**

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI peripheral base address.  |
| <i>handle</i> | pointer to spi_slave_handle_t structure which stores the transfer state |

## 27.3 SPI DMA Driver

### 27.3.1 Overview

This section describes the programming interface of the SPI DMA driver.

#### 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.

#### DMA Transactional

- void [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.
- 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)  
Get the transferred bytes for SPI slave DMA.
- static void [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.
- 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)  
Get the transferred bytes for SPI slave DMA.

### 27.3.2 Data Structure Documentation

#### 27.3.2.1 struct \_spi\_dma\_handle

##### Data Fields

- bool [txInProgress](#)  
*Send transfer finished.*
- 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.*

### 27.3.3 Typedef Documentation

**27.3.3.1** `typedef void(* spi_dma_callback_t)(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)`

### 27.3.4 Function Documentation

**27.3.4.1** `void SPI_MasterTransferCreateHandleDMA ( SPI_Type * base, spi_dma_handle_t * handle, spi_dma_callback_t callback, void * userData, dma_handle_t * txHandle, dma_handle_t * rxHandle )`

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

Parameters

---

|                 |   |
|-----------------|---|
| <i>base</i>     | SPI peripheral base address.  |
| <i>handle</i>   | SPI handle pointer.   |
| <i>callback</i> | User callback function called at the end of a transfer.                       |
| <i>userData</i> | User data for callback.   |
| <i>txHandle</i> | DMA handle pointer for SPI Tx, the handle shall be static allocated by users. |
| <i>rxHandle</i> | DMA handle pointer for SPI Rx, the handle shall be static allocated by users. |

#### 27.3.4.2 **status\_t SPI\_MasterTransferDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* )**

##### Note

This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

##### Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SPI peripheral base address.       |
| <i>handle</i> | SPI DMA handle pointer.            |
| <i>xfer</i>   | Pointer to dma transfer structure. |

##### Return values

|                                |   |
|--------------------------------|---|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_SPI_Busy</i>        | SPI is not idle, is running another transfer. |

#### 27.3.4.3 **void SPI\_MasterTransferAbortDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle* )**

##### Parameters

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

## SPI DMA Driver

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

**27.3.4.4** `status_t SPI_MasterTransferGetCountDMA ( SPI_Type * base, spi_dma_handle_t * handle, size_t * count )`

### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SPI peripheral base address. |
| <i>handle</i> | SPI DMA handle pointer.      |
| <i>count</i>  | Transferred bytes.           |

### Return values

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_SPI_Success</i>          | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

**27.3.4.5** `static void SPI_SlaveTransferCreateHandleDMA ( SPI_Type * base, spi_dma_handle_t * handle, spi_dma_callback_t callback, void * userData, dma_handle_t * txHandle, dma_handle_t * rxHandle ) [inline], [static]`

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

### Parameters

|                 |   |
|-----------------|---|
| <i>base</i>     | SPI peripheral base address.  |
| <i>handle</i>   | SPI handle pointer.   |
| <i>callback</i> | User callback function called at the end of a transfer.                       |
| <i>userData</i> | User data for callback.   |
| <i>txHandle</i> | DMA handle pointer for SPI Tx, the handle shall be static allocated by users. |
| <i>rxHandle</i> | DMA handle pointer for SPI Rx, the handle shall be static allocated by users. |

**27.3.4.6** `static status_t SPI_SlaveTransferDMA ( SPI_Type * base, spi_dma_handle_t * handle, spi_transfer_t * xfer ) [inline], [static]`

## Note

This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SPI peripheral base address.       |
| <i>handle</i> | SPI DMA handle pointer.            |
| <i>xfer</i>   | Pointer to dma transfer structure. |

## Return values

|                                |   |
|--------------------------------|---|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_SPI_Busy</i>        | SPI is not idle, is running another transfer. |

**27.3.4.7 static void SPI\_SlaveTransferAbortDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle* ) [inline], [static]**

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SPI peripheral base address. |
| <i>handle</i> | SPI DMA handle pointer.      |

**27.3.4.8 static status\_t SPI\_SlaveTransferGetCountDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, size\_t \* *count* ) [inline], [static]**

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SPI peripheral base address. |
| <i>handle</i> | SPI DMA handle pointer.      |
| <i>count</i>  | Transferred bytes.           |

## Return values

## SPI DMA Driver

|                                     |  |
|-------------------------------------|--|
| <i>kStatus_SPI_Success</i>          | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |



## 27.4 SPI FreeRTOS driver

### 27.4.1 Overview

This section describes the programming interface of the SPI FreeRTOS driver.

#### Data Structures

- struct [spi\\_rtos\\_handle\\_t](#)  
*SPI FreeRTOS handle. [More...](#)*

#### 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.*

### 27.4.2 Data Structure Documentation

#### 27.4.2.1 struct spi\_rtos\_handle\_t

SPI RTOS handle.

#### 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 transfer.*
- SemaphoreHandle\_t [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_EVENT \* [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_SEM [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 27.4.3 Function Documentation

**27.4.3.1** `status_t SPI_RTOS_Init ( spi_rtos_handle_t * handle, SPI_Type * base, const spi_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the SPI module and related RTOS context.

## Parameters

|                     |  |
|---------------------|--|
| <i>handle</i>       | The RTOS SPI handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the SPI instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up SPI in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the SPI module.                              |

## Returns

status of the operation.

### 27.4.3.2 **status\_t SPI\_RTOS\_Deinit ( spi\_rtos\_handle\_t \* *handle* )**

This function deinitializes the SPI module and related RTOS context.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | The RTOS SPI handle. |
|---------------|----------------------|

### 27.4.3.3 **status\_t SPI\_RTOS\_Transfer ( spi\_rtos\_handle\_t \* *handle*, spi\_transfer\_t \* *transfer* )**

This function performs an SPI transfer according to data given in the transfer structure.

## Parameters

|                 |   |
|-----------------|---|
| <i>handle</i>   | The RTOS SPI handle.                          |
| <i>transfer</i> | Structure specifying the transfer parameters. |

## Returns

status of the operation.

### 27.5 SPI $\mu$ COS/II driver

#### 27.5.1 Overview

This section describes the programming interface of the SPI  $\mu$ COS/II driver.

#### Data Structures

- struct [spi\\_rtos\\_handle\\_t](#)  
*SPI FreeRTOS handle. [More...](#)*

#### 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.*

#### 27.5.2 Data Structure Documentation

##### 27.5.2.1 struct [spi\\_rtos\\_handle\\_t](#)

SPI RTOS handle.

#### 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 transfer.*
- SemaphoreHandle\_t [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_EVENT \* [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_SEM [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 27.5.3 Function Documentation

**27.5.3.1** `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.

## SPI $\mu$ COS/II driver

### Parameters

|                     |  |
|---------------------|--|
| <i>handle</i>       | The RTOS SPI handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the SPI instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up SPI in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the SPI module.                              |

### Returns

status of the operation.

#### 27.5.3.2 **status\_t SPI\_RTOS\_Deinit ( spi\_rtos\_handle\_t \* *handle* )**

This function deinitializes the SPI module and related RTOS context.

### Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | The RTOS SPI handle. |
|---------------|----------------------|

#### 27.5.3.3 **status\_t SPI\_RTOS\_Transfer ( spi\_rtos\_handle\_t \* *handle*, spi\_transfer\_t \* *transfer* )**

This function performs an SPI transfer according to data given in the transfer structure.

### Parameters

|                 |   |
|-----------------|---|
| <i>handle</i>   | The RTOS SPI handle.                          |
| <i>transfer</i> | Structure specifying the transfer parameters. |

### Returns

status of the operation.

## 27.6 SPI $\mu$ COS/III driver

### 27.6.1 Overview

This section describes the programming interface of the SPI  $\mu$ COS/III driver.

#### Data Structures

- struct [spi\\_rtos\\_handle\\_t](#)  
*SPI FreeRTOS handle. [More...](#)*

#### 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.*

### 27.6.2 Data Structure Documentation

#### 27.6.2.1 struct spi\_rtos\_handle\_t

SPI RTOS handle.

#### 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 transfer.*
- SemaphoreHandle\_t [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_EVENT \* [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP \* [event](#)  
*Semaphore to notify and unblock task when transfer ends.*
- OS\_SEM [mutex](#)  
*Mutex to lock the handle during a transfer.*
- OS\_FLAG\_GRP [event](#)  
*Semaphore to notify and unblock task when transfer ends.*

### 27.6.3 Function Documentation

**27.6.3.1** `status_t SPI_RTOS_Init ( spi_rtos_handle_t * handle, SPI_Type * base, const spi_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function initializes the SPI module and related RTOS context.



## Parameters

|                     |  |
|---------------------|--|
| <i>handle</i>       | The RTOS SPI handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the SPI instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up SPI in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the SPI module.                              |

## Returns

status of the operation.

### 27.6.3.2 **status\_t SPI\_RTOS\_Deinit ( spi\_rtos\_handle\_t \* *handle* )**

This function deinitializes the SPI module and related RTOS context.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | The RTOS SPI handle. |
|---------------|----------------------|

### 27.6.3.3 **status\_t SPI\_RTOS\_Transfer ( spi\_rtos\_handle\_t \* *handle*, spi\_transfer\_t \* *transfer* )**

This function performs an SPI transfer according to data given in the transfer structure.

## Parameters

|                 |   |
|-----------------|---|
| <i>handle</i>   | The RTOS SPI handle.                          |
| <i>transfer</i> | Structure specifying the transfer parameters. |

## Returns

status of the operation.



## Chapter 28

### TPM: Timer PWM Module

#### 28.1 Overview

The KSDK provides a driver for the Timer PWM Module (TPM) of Kinetis devices.

The KSDK TPM driver supports the generation of PWM signals, input capture, and output compare modes. On some SoC's, the driver supports the generation of combined PWM signals, dual-edge capture, and quadrature decode modes. The driver also supports configuring each of the TPM fault inputs. The fault input is available only on some SoC's.

The function [TPM\\_Init\(\)](#) initializes the TPM with specified configurations. The function [TPM\\_GetDefaultConfig\(\)](#) gets the default configurations. On some SoC's, the initialization function issues a software reset to reset the TPM internal logic. The initialization function configures the TPM's behavior when it receives a trigger input and its operation in doze and debug modes.

The function [TPM\\_Deinit\(\)](#) disables the TPM counter and turns off the module clock.

The function [TPM\\_SetupPwm\(\)](#) sets up TPM channels for the PWM output. The function can set up the PWM signal properties for multiple channels. Each channel has its own [tpm\\_chnl\\_pwm\\_signal\\_param\\_t](#) structure that is used to specify the output signals duty cycle and level-mode. However, the same PWM period and PWM mode is applied to all channels requesting a PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 where 0=inactive signal (0% duty cycle) and 100=always active signal (100% duty cycle). When generating a combined PWM signal, the channel number passed refers to a channel pair number, for example 0 refers to channel 0 and 1, 1 refers to channels 2 and 3.

The function [TPM\\_UpdatePwmDutycycle\(\)](#) updates the PWM signal duty cycle of a particular TPM channel.

The function [TPM\\_UpdateChnlEdgeLevelSelect\(\)](#) updates the level select bits of a particular TPM channel. This can be used to disable the PWM output when making changes to the PWM signal.

The function [TPM\\_SetupInputCapture\(\)](#) sets up a TPM channel for input capture. The user can specify the capture edge.

The function [TPM\\_SetupDualEdgeCapture\(\)](#) can be used to measure the pulse width of a signal. This is available only for certain SoC's. A channel pair is used during the capture with the input signal coming through a channel that can be configured. The user can specify the capture edge for each channel and any filter value to be used when processing the input signal.

The function [TPM\\_SetupOutputCompare\(\)](#) sets up a TPM channel for output comparison. The user can specify the channel output on a successful comparison and a comparison value.

The function [TPM\\_SetupQuadDecode\(\)](#) sets up TPM channels 0 and 1 for quad decode, which is available only for certain SoC's. The user can specify the quad decode mode, polarity, and filter properties for each input signal.

## Typical use case

The function `TPM_SetupFault()` sets up the properties for each fault, which is available only for certain SoC's. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

Provides functions to get and clear the TPM status.

Provides functions to enable/disable TPM interrupts and get current enabled interrupts.

## 28.2 Typical use case

### 28.2.1 PWM output

Output the PWM signal on 2 TPM channels with different duty cycles. Periodically update the PWM signal duty cycle.

```
int main(void)
{
    bool brightnessUp = true; /* Indicates whether the LED is brighter or dimmer. */
    tpm_config_t tpmInfo;
    uint8_t updatedDutycycle = 0U;
    tpm_chnl_pwm_signal_param_t tpmParam[2];

    /* Configures the TPM parameters with frequency 24 kHz. */
    tpmParam[0].chnlNumber = (tpm_chnl_t)BOARD_FIRST_TPM_CHANNEL;
    tpmParam[0].level = kTPM_LowTrue;
    tpmParam[0].dutyCyclePercent = 0U;

    tpmParam[1].chnlNumber = (tpm_chnl_t)BOARD_SECOND_TPM_CHANNEL;
    tpmParam[1].level = kTPM_LowTrue;
    tpmParam[1].dutyCyclePercent = 0U;

    /* Board pin, clock, and debug console initialization. */
    BOARD_InitHardware();

    TPM_GetDefaultConfig(&tpmInfo);
    /* Initializes the TPM module. */
    TPM_Init(BOARD_TPM_BASEADDR, &tpmInfo);

    TPM_SetupPwm(BOARD_TPM_BASEADDR, tpmParam, 2U,
        kTPM_EdgeAlignedPwm, 24000U, TPM_SOURCE_CLOCK);
    TPM_StartTimer(BOARD_TPM_BASEADDR, kTPM_SystemClock);
    while (1)
    {
        /* Delays to see the change of LED brightness. */
        delay();

        if (brightnessUp)
        {
            /* Increases a duty cycle until it reaches a limited value. */
            if (++updatedDutycycle == 100U)
            {
                brightnessUp = false;
            }
        }
        else
        {
            /* Decreases a duty cycle until it reaches a limited value. */
            if (--updatedDutycycle == 0U)
            {
                brightnessUp = true;
            }
        }
    }
}
```

```

    /* Starts PWM mode with an updated duty cycle. */
    TPM_UpdatePwmDutycycle(BOARD_TPM_BASEADDR, (
tpm_chnl_t)BOARD_FIRST_TPM_CHANNEL, kTPM_EdgeAlignedPwm,
                        updatedDutycycle);
    TPM_UpdatePwmDutycycle(BOARD_TPM_BASEADDR, (
tpm_chnl_t)BOARD_SECOND_TPM_CHANNEL, kTPM_EdgeAlignedPwm,
                        updatedDutycycle);
}

```

## Data Structures

- struct `tpm_chnl_pwm_signal_param_t`  
Options to configure a TPM channel's PWM signal. [More...](#)
- struct `tpm_config_t`  
TPM config structure. [More...](#)

## Enumerations

- enum `tpm_chnl_t` {  
kTPM\_Chnl\_0 = 0U,  
kTPM\_Chnl\_1,  
kTPM\_Chnl\_2,  
kTPM\_Chnl\_3,  
kTPM\_Chnl\_4,  
kTPM\_Chnl\_5,  
kTPM\_Chnl\_6,  
kTPM\_Chnl\_7 }  
List of TPM channels.
- enum `tpm_pwm_mode_t` {  
kTPM\_EdgeAlignedPwm = 0U,  
kTPM\_CenterAlignedPwm }  
TPM PWM operation modes.
- enum `tpm_pwm_level_select_t` {  
kTPM\_NoPwmSignal = 0U,  
kTPM\_LowTrue,  
kTPM\_HighTrue }  
TPM PWM output pulse mode: high-true, low-true or no output.
- enum `tpm_trigger_select_t`  
Trigger options available.
- enum `tpm_output_compare_mode_t` {  
kTPM\_NoOutputSignal = (1U << TPM\_CnSC\_MSA\_SHIFT),  
kTPM\_ToggleOnMatch = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (1U << TPM\_CnSC\_ELSA\_S-  
HIFT)),  
kTPM\_ClearOnMatch = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (2U << TPM\_CnSC\_ELSA\_SH-  
IFT)),  
kTPM\_SetOnMatch = ((1U << TPM\_CnSC\_MSA\_SHIFT) | (3U << TPM\_CnSC\_ELSA\_SHIF-  
T)),  
kTPM\_HighPulseOutput = ((3U << TPM\_CnSC\_MSA\_SHIFT) | (1U << TPM\_CnSC\_ELSA\_-

## Typical use case

SHIFT)),

kTPM\_LowPulseOutput = ((3U << TPM\_CnSC\_MSA\_SHIFT) | (2U << TPM\_CnSC\_ELSA\_SHIFT)) }

*TPM output compare modes.*

- enum tpm\_input\_capture\_edge\_t {  
kTPM\_RisingEdge = (1U << TPM\_CnSC\_ELSA\_SHIFT),  
kTPM\_FallingEdge = (2U << TPM\_CnSC\_ELSA\_SHIFT),  
kTPM\_RiseAndFallEdge = (3U << TPM\_CnSC\_ELSA\_SHIFT) }

*TPM input capture edge.*

- enum tpm\_clock\_source\_t {  
kTPM\_SystemClock = 1U,  
kTPM\_ExternalClock }

*TPM clock source selection.*

- enum tpm\_clock\_prescale\_t {  
kTPM\_Prescale\_Divide\_1 = 0U,  
kTPM\_Prescale\_Divide\_2,  
kTPM\_Prescale\_Divide\_4,  
kTPM\_Prescale\_Divide\_8,  
kTPM\_Prescale\_Divide\_16,  
kTPM\_Prescale\_Divide\_32,  
kTPM\_Prescale\_Divide\_64,  
kTPM\_Prescale\_Divide\_128 }

*TPM prescale value selection for the clock source.*

- enum tpm\_interrupt\_enable\_t {  
kTPM\_Chnl0InterruptEnable = (1U << 0),  
kTPM\_Chnl1InterruptEnable = (1U << 1),  
kTPM\_Chnl2InterruptEnable = (1U << 2),  
kTPM\_Chnl3InterruptEnable = (1U << 3),  
kTPM\_Chnl4InterruptEnable = (1U << 4),  
kTPM\_Chnl5InterruptEnable = (1U << 5),  
kTPM\_Chnl6InterruptEnable = (1U << 6),  
kTPM\_Chnl7InterruptEnable = (1U << 7),  
kTPM\_TimeOverflowInterruptEnable = (1U << 8) }

*List of TPM interrupts.*

- enum tpm\_status\_flags\_t {  
kTPM\_Chnl0Flag = (1U << 0),  
kTPM\_Chnl1Flag = (1U << 1),  
kTPM\_Chnl2Flag = (1U << 2),  
kTPM\_Chnl3Flag = (1U << 3),  
kTPM\_Chnl4Flag = (1U << 4),  
kTPM\_Chnl5Flag = (1U << 5),  
kTPM\_Chnl6Flag = (1U << 6),  
kTPM\_Chnl7Flag = (1U << 7),  
kTPM\_TimeOverflowFlag = (1U << 8) }

*List of TPM flags.*

## Driver version

- #define `FSL_TPM_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 2)`)  
Version 2.0.2.

## Initialization and deinitialization

- void `TPM_Init` (`TPM_Type *base`, const `tpm_config_t *config`)  
*Ungates the TPM clock and configures the peripheral for basic operation.*
- void `TPM_Deinit` (`TPM_Type *base`)  
*Stops the counter and gates the TPM clock.*
- void `TPM_GetDefaultConfig` (`tpm_config_t *config`)  
*Fill in the TPM config struct with the default settings.*

## Channel mode operations

- status\_t `TPM_SetupPwm` (`TPM_Type *base`, const `tpm_chnl_pwm_signal_param_t *chnlParams`, `uint8_t numOfChnls`, `tpm_pwm_mode_t mode`, `uint32_t pwmFreq_Hz`, `uint32_t srcClock_Hz`)  
*Configures the PWM signal parameters.*
- void `TPM_UpdatePwmDutycycle` (`TPM_Type *base`, `tpm_chnl_t chnlNumber`, `tpm_pwm_mode_t currentPwmMode`, `uint8_t dutyCyclePercent`)  
*Update the duty cycle of an active PWM signal.*
- void `TPM_UpdateChnlEdgeLevelSelect` (`TPM_Type *base`, `tpm_chnl_t chnlNumber`, `uint8_t level`)  
*Update the edge level selection for a channel.*
- void `TPM_SetupInputCapture` (`TPM_Type *base`, `tpm_chnl_t chnlNumber`, `tpm_input_capture_edge_t captureMode`)  
*Enables capturing an input signal on the channel using the function parameters.*
- void `TPM_SetupOutputCompare` (`TPM_Type *base`, `tpm_chnl_t chnlNumber`, `tpm_output_compare_mode_t compareMode`, `uint32_t compareValue`)  
*Configures the TPM to generate timed pulses.*

## Interrupt Interface

- void `TPM_EnableInterrupts` (`TPM_Type *base`, `uint32_t mask`)  
*Enables the selected TPM interrupts.*
- void `TPM_DisableInterrupts` (`TPM_Type *base`, `uint32_t mask`)  
*Disables the selected TPM interrupts.*
- `uint32_t` `TPM_GetEnabledInterrupts` (`TPM_Type *base`)  
*Gets the enabled TPM interrupts.*

## Status Interface

- static `uint32_t` `TPM_GetStatusFlags` (`TPM_Type *base`)  
*Gets the TPM status flags.*
- static void `TPM_ClearStatusFlags` (`TPM_Type *base`, `uint32_t mask`)  
*Clears the TPM status flags.*

## Timer Start and Stop

- static void `TPM_StartTimer` (`TPM_Type *base`, `tpm_clock_source_t clockSource`)

## Data Structure Documentation

- Starts the TPM counter.*
- static void [TPM\\_StopTimer](#) (TPM\_Type \*base)  
*Stops the TPM counter.*

## 28.3 Data Structure Documentation

### 28.3.1 struct tpm\_chnl\_pwm\_signal\_param\_t

#### Data Fields

- [tpm\\_chnl\\_t chnlNumber](#)  
*TPM channel to configure.*
- [tpm\\_pwm\\_level\\_select\\_t level](#)  
*PWM output active level select.*
- [uint8\\_t dutyCyclePercent](#)  
*PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...*

#### 28.3.1.0.0.44 Field Documentation

##### 28.3.1.0.0.44.1 tpm\_chnl\_t tpm\_chnl\_pwm\_signal\_param\_t::chnlNumber

In combined mode (available in some SoC's, this represents the channel pair number

##### 28.3.1.0.0.44.2 uint8\_t tpm\_chnl\_pwm\_signal\_param\_t::dutyCyclePercent

100=always active signal (100% duty cycle)

### 28.3.2 struct tpm\_config\_t

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the [TPM\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

#### Data Fields

- [tpm\\_clock\\_prescale\\_t prescale](#)  
*Select TPM clock prescale value.*
- bool [useGlobalTimeBase](#)  
*true: Use of an external global time base is enabled; false: disabled*
- [tpm\\_trigger\\_select\\_t triggerSelect](#)  
*Input trigger to use for controlling the counter operation.*
- bool [enableDoze](#)  
*true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode*
- bool [enableDebugMode](#)  
*true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode*



- bool [enableReloadOnTrigger](#)  
*true: TPM counter is reloaded on trigger; false: TPM counter not reloaded*
- bool [enableStopOnOverflow](#)  
*true: TPM counter stops after overflow; false: TPM counter continues running after overflow*
- bool [enableStartOnTrigger](#)  
*true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately*

## 28.4 Enumeration Type Documentation

### 28.4.1 enum tpm\_chnl\_t

Note

Actual number of available channels is SoC dependent

Enumerator

***kTPM\_Chnl\_0*** TPM channel number 0.  
***kTPM\_Chnl\_1*** TPM channel number 1.  
***kTPM\_Chnl\_2*** TPM channel number 2.  
***kTPM\_Chnl\_3*** TPM channel number 3.  
***kTPM\_Chnl\_4*** TPM channel number 4.  
***kTPM\_Chnl\_5*** TPM channel number 5.  
***kTPM\_Chnl\_6*** TPM channel number 6.  
***kTPM\_Chnl\_7*** TPM channel number 7.

### 28.4.2 enum tpm\_pwm\_mode\_t

Enumerator

***kTPM\_EdgeAlignedPwm*** Edge aligned PWM.  
***kTPM\_CenterAlignedPwm*** Center aligned PWM.

### 28.4.3 enum tpm\_pwm\_level\_select\_t

Enumerator

***kTPM\_NoPwmSignal*** No PWM output on pin.  
***kTPM\_LowTrue*** Low true pulses.  
***kTPM\_HighTrue*** High true pulses.

## Enumeration Type Documentation

### 28.4.4 enum tpm\_trigger\_select\_t

This is used for both internal & external trigger sources (external option available in certain SoC's)

Note

The actual trigger options available is SoC-specific.

### 28.4.5 enum tpm\_output\_compare\_mode\_t

Enumerator

*kTPM\_NoOutputSignal* No channel output when counter reaches CnV.  
*kTPM\_ToggleOnMatch* Toggle output.  
*kTPM\_ClearOnMatch* Clear output.  
*kTPM\_SetOnMatch* Set output.  
*kTPM\_HighPulseOutput* Pulse output high.  
*kTPM\_LowPulseOutput* Pulse output low.

### 28.4.6 enum tpm\_input\_capture\_edge\_t

Enumerator

*kTPM\_RisingEdge* Capture on rising edge only.  
*kTPM\_FallingEdge* Capture on falling edge only.  
*kTPM\_RiseAndFallEdge* Capture on rising or falling edge.

### 28.4.7 enum tpm\_clock\_source\_t

Enumerator

*kTPM\_SystemClock* System clock.  
*kTPM\_ExternalClock* External clock.

### 28.4.8 enum tpm\_clock\_prescale\_t

Enumerator

*kTPM\_Prescale\_Divide\_1* Divide by 1.  
*kTPM\_Prescale\_Divide\_2* Divide by 2.

*kTPM\_Prescale\_Divide\_4* Divide by 4.  
*kTPM\_Prescale\_Divide\_8* Divide by 8.  
*kTPM\_Prescale\_Divide\_16* Divide by 16.  
*kTPM\_Prescale\_Divide\_32* Divide by 32.  
*kTPM\_Prescale\_Divide\_64* Divide by 64.  
*kTPM\_Prescale\_Divide\_128* Divide by 128.

### 28.4.9 enum tpm\_interrupt\_enable\_t

Enumerator

*kTPM\_Chnl0InterruptEnable* Channel 0 interrupt.  
*kTPM\_Chnl1InterruptEnable* Channel 1 interrupt.  
*kTPM\_Chnl2InterruptEnable* Channel 2 interrupt.  
*kTPM\_Chnl3InterruptEnable* Channel 3 interrupt.  
*kTPM\_Chnl4InterruptEnable* Channel 4 interrupt.  
*kTPM\_Chnl5InterruptEnable* Channel 5 interrupt.  
*kTPM\_Chnl6InterruptEnable* Channel 6 interrupt.  
*kTPM\_Chnl7InterruptEnable* Channel 7 interrupt.  
*kTPM\_TimeOverflowInterruptEnable* Time overflow interrupt.

### 28.4.10 enum tpm\_status\_flags\_t

Enumerator

*kTPM\_Chnl0Flag* Channel 0 flag.  
*kTPM\_Chnl1Flag* Channel 1 flag.  
*kTPM\_Chnl2Flag* Channel 2 flag.  
*kTPM\_Chnl3Flag* Channel 3 flag.  
*kTPM\_Chnl4Flag* Channel 4 flag.  
*kTPM\_Chnl5Flag* Channel 5 flag.  
*kTPM\_Chnl6Flag* Channel 6 flag.  
*kTPM\_Chnl7Flag* Channel 7 flag.  
*kTPM\_TimeOverflowFlag* Time overflow flag.

## 28.5 Function Documentation

### 28.5.1 void TPM\_Init ( TPM\_Type \* *base*, const tpm\_config\_t \* *config* )

Note

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

## Function Documentation

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | TPM peripheral base address             |
| <i>config</i> | Pointer to user's TPM config structure. |

### 28.5.2 void TPM\_Deinit ( TPM\_Type \* *base* )

### Parameters

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

### 28.5.3 void TPM\_GetDefaultConfig ( tpm\_config\_t \* *config* )

The default values are:

```
* config->prescale = kTPM_Prescale_Divide_1;
* config->useGlobalTimeBase = false;
* config->dozeEnable = false;
* config->dbgMode = false;
* config->enableReloadOnTrigger = false;
* config->enableStopOnOverflow = false;
* config->enableStartOnTrigger = false;
*#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
* config->enablePauseOnTrigger = false;
*#endif
* config->triggerSelect = kTPM_Trigger_Select_0;
*#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
* config->triggerSource = kTPM_TriggerSource_External;
*#endif
*
```

### Parameters

|               |   |
|---------------|---|
| <i>config</i> | Pointer to user's TPM config structure. |
|---------------|---|

### 28.5.4 status\_t TPM\_SetupPwm ( TPM\_Type \* *base*, const tpm\_chnl\_pwm\_signal\_param\_t \* *chnlParams*, uint8\_t *numOfChnls*, tpm\_pwm\_mode\_t *mode*, uint32\_t *pwmFreq\_Hz*, uint32\_t *srcClock\_Hz* )

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

## Parameters

|                    |   |
|--------------------|---|
| <i>base</i>        | TPM peripheral base address   |
| <i>chnlParams</i>  | Array of PWM channel parameters to configure the channel(s)                         |
| <i>numOfChnls</i>  | Number of channels to configure, this should be the size of the array passed in     |
| <i>mode</i>        | PWM operation mode, options available in enumeration <a href="#">tpm_pwm_mode_t</a> |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz  |
| <i>srcClock_Hz</i> | TPM counter clock in Hz   |

## Returns

kStatus\_Success if the PWM setup was successful, kStatus\_Error on failure

**28.5.5 void TPM\_UpdatePwmDutycycle ( TPM\_Type \* *base*, tpm\_chnl\_t *chnlNumber*, tpm\_pwm\_mode\_t *currentPwmMode*, uint8\_t *dutyCyclePercent* )**

## Parameters

|                          |   |
|--------------------------|---|
| <i>base</i>              | TPM peripheral base address   |
| <i>chnlNumber</i>        | The channel number. In combined mode, this represents the channel pair number   |
| <i>currentPwm-Mode</i>   | The current PWM mode set during PWM setup   |
| <i>dutyCycle-Percent</i> | New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle) |

**28.5.6 void TPM\_UpdateChnlEdgeLevelSelect ( TPM\_Type \* *base*, tpm\_chnl\_t *chnlNumber*, uint8\_t *level* )**

## Parameters

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

## Function Documentation

|                   |   |
|-------------------|---|
| <i>chnlNumber</i> | The channel number  |
| <i>level</i>      | The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field. |

### 28.5.7 void TPM\_SetupInputCapture ( TPM\_Type \* *base*, tpm\_chnl\_t *chnlNumber*, tpm\_input\_capture\_edge\_t *captureMode* )

When the edge specified in the *captureMode* argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>base</i>        | TPM peripheral base address     |
| <i>chnlNumber</i>  | The channel number              |
| <i>captureMode</i> | Specifies which edge to capture |

### 28.5.8 void TPM\_SetupOutputCompare ( TPM\_Type \* *base*, tpm\_chnl\_t *chnlNumber*, tpm\_output\_compare\_mode\_t *compareMode*, uint32\_t *compareValue* )

When the TPM counter matches the value of *compareVal* argument (this is written into CnV reg), the channel output is changed based on what is specified in the *compareMode* argument.

Parameters

|                     |  |
|---------------------|--|
| <i>base</i>         | TPM peripheral base address  |
| <i>chnlNumber</i>   | The channel number   |
| <i>compareMode</i>  | Action to take on the channel output when the compare condition is met |
| <i>compareValue</i> | Value to be programmed in the CnV register.                            |

### 28.5.9 void TPM\_EnableInterrupts ( TPM\_Type \* *base*, uint32\_t *mask* )

Parameters

---

|             |   |
|-------------|---|
| <i>base</i> | TPM peripheral base address   |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">tpm_interrupt_enable_t</a> |

### 28.5.10 void TPM\_DisableInterrupts ( TPM\_Type \* *base*, uint32\_t *mask* )

Parameters

|             |  |
|-------------|--|
| <i>base</i> | TPM peripheral base address  |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">tpm_interrupt_enable_t</a> |

### 28.5.11 uint32\_t TPM\_GetEnabledInterrupts ( TPM\_Type \* *base* )

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [tpm\\_interrupt\\_enable\\_t](#)

### 28.5.12 static uint32\_t TPM\_GetStatusFlags ( TPM\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [tpm\\_status\\_flags\\_t](#)

### 28.5.13 static void TPM\_ClearStatusFlags ( TPM\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Function Documentation

### Parameters

|             |   |
|-------------|---|
| <i>base</i> | TPM peripheral base address   |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">tpm_-status_flags_t</a> |

**28.5.14 static void TPM\_StartTimer ( TPM\_Type \* *base*, tpm\_clock\_source\_t *clockSource* ) [inline], [static]**

### Parameters

|                    |   |
|--------------------|---|
| <i>base</i>        | TPM peripheral base address   |
| <i>clockSource</i> | TPM clock source; once clock source is set the counter will start running |

**28.5.15 static void TPM\_StopTimer ( TPM\_Type \* *base* ) [inline], [static]**

### Parameters

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





## Chapter 29

# UART: Universal Asynchronous Receiver/Transmitter Driver

## 29.1 Overview

### Modules

- [UART DMA Driver](#)
- [UART Driver](#)
- [UART FreeRTOS Driver](#)
- [UART eDMA Driver](#)
- [UART  \$\mu\$ COS/II Driver](#)
- [UART  \$\mu\$ COS/III Driver](#)

### 29.2 UART Driver

#### 29.2.1 Overview

The KSDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) module of Kinetis devices.

The UART driver includes two parts: functional APIs and transactional APIs.

Functional APIs are used for UART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the UART 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. UART functional operation groups provide the functional APIs set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `uart_handle_t` as the first parameter. Initialize the handle by calling the `UART_CreateHandle()` API.

Transactional APIs support asynchronous transfer, which means that the functions `UART_SendNonBlocking()` and `UART_ReceiveNonBlocking()` set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_UART_TxIdle` and `kStatus_UART_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 `UART_CreateHandle()`. 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 `UART_ReceiveNonBlocking()` 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_UART_RxIdle`.

If the receive ring buffer is full, the upper layer is informed through a callback with the `kStatus_UART_RxRingBufferOverflow`. 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:

```
UART_CreateHandle(&handle, UART0, &ringBuffer, 32);
```

In this example, the buffer size is 32, but only 31 bytes are used for saving data.

#### 29.2.2 Typical use case

##### 29.2.2.1 UART Send/receive using a polling method

```
uint8_t ch;
```

```

UART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableTx = true;
user_config.enableRx = true;

UART_Init(UART1, &user_config, 120000000U);

while(1)
{
    UART_TransferReceiveBlocking(UART1, &ch, 1);
    UART_TransferSendBlocking(UART1, &ch, 1);
}

```

### 29.2.2.2 UART Send/receive using an interrupt method

```

uart_handle_t g_uartHandle;
uart_config_t user_config;
uart_transfer_t sendXfer;
uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void UART_UserCallback(uart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    UART_Init(UART1, &user_config, 120000000U);
    UART_CreateHandle(&g_uartHandle, UART1, NULL, 0);
    UART_SetTransferCallback(&g_uartHandle, UART_UserCallback, NULL);

    // Prepare to send.
    sendXfer.data = sendData;
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Send out.
    UART_SendNonBlocking(&g_uartHandle, &sendXfer);

    // Wait send finished.
    while (!txFinished)
    {
    }
}

```

## UART Driver

```
// Prepare to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
rxFinished = false;

// Receive.
UART_ReceiveNonBlocking(&g_uartHandle, &receiveXfer, NULL);

// Wait receive finished.
while (!rxFinished)
{
}

// ...
}
```

### 29.2.2.3 UART Receive using the ringbuffer feature

```
#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE 32

uart_handle_t g_uartHandle;
uart_config_t user_config;
uart_transfer_t sendXfer;
uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void UART_UserCallback(uart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    UART_Init(UART1, &user_config, 120000000U);
    UART_CreateHandle(&g_uartHandle, UART1, &ringBuffer, RING_BUFFER_SIZE);
    UART_SetTransferCallback(&g_uartHandle, UART_UserCallback, NULL);

    // Now the RX is working in background, receive in to ring buffer.

    // Prepare to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = RX_DATA_SIZE;
    rxFinished = false;

    // Receive.
    UART_ReceiveNonBlocking(&g_uartHandle, &receiveXfer, &bytesRead);

    if (bytesRead == RX_DATA_SIZE) /* Have read enough data. */
}
```

```

    {
        ;
    }
    else
    {
        if (bytesRead) /* Received some data, process first. */
        {
            ;
        }

        // Wait receive finished.
        while (!rxFinished)
        {
        }
    }

    // ...
}

```

### 29.2.2.4 UART Send/Receive using the DMA method

```

uart_handle_t g_uartHandle;
dma_handle_t g_uartTxDmaHandle;
dma_handle_t g_uartRxDmaHandle;
uart_config_t user_config;
uart_transfer_t sendXfer;
uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void UART_UserCallback(uart_handle_t *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableTx = true;
    user_config.enableRx = true;

    UART_Init(UART1, &user_config, 120000000U);

    // Set up the DMA
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, UART_TX_DMA_CHANNEL, UART_TX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, UART_TX_DMA_CHANNEL);
    DMAMUX_SetSource(DMAMUX0, UART_RX_DMA_CHANNEL, UART_RX_DMA_REQUEST);
    DMAMUX_EnableChannel(DMAMUX0, UART_RX_DMA_CHANNEL);
}

```

## UART Driver

```
DMA_Init(DMA0);

/* Create DMA handle. */
DMA_CreateHandle(&g_uartTxDmaHandle, DMA0, UART_TX_DMA_CHANNEL);
DMA_CreateHandle(&g_uartRxDmaHandle, DMA0, UART_RX_DMA_CHANNEL);

UART_CreateHandleDMA(&g_uartHandle, UART1, &g_uartTxDmaHandle, &g_uartRxDmaHandle);
UART_SetTransferCallbackDMA(&g_uartDmaHandle, UART_UserCallback, NULL);

// Prepare to send.
sendXfer.data = sendData;
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Send out.
UART_SendDMA(&g_uartHandle, &sendXfer);

// Wait send finished.
while (!txFinished)
{
}

// Prepare to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
rxFinished = false;

// Receive.
UART_ReceiveDMA(&g_uartHandle, &receiveXfer, NULL);

// Wait receive finished.
while (!rxFinished)
{
}

// ...
}
```

## Data Structures

- struct `uart_config_t`  
*UART configuration structure. [More...](#)*
- struct `uart_transfer_t`  
*UART transfer structure. [More...](#)*
- struct `uart_handle_t`  
*UART handle structure. [More...](#)*

## Typedefs

- typedef void(\* `uart_transfer_callback_t`)(UART\_Type \*base, uart\_handle\_t \*handle, status\_t status, void \*userData)  
*UART transfer callback function.*

## Enumerations

- enum `_uart_status` {
  - `kStatus_UART_TxBusy` = MAKE\_STATUS(kStatusGroup\_UART, 0),
  - `kStatus_UART_RxBusy` = MAKE\_STATUS(kStatusGroup\_UART, 1),
  - `kStatus_UART_TxIdle` = MAKE\_STATUS(kStatusGroup\_UART, 2),
  - `kStatus_UART_RxIdle` = MAKE\_STATUS(kStatusGroup\_UART, 3),
  - `kStatus_UART_TxWatermarkTooLarge` = MAKE\_STATUS(kStatusGroup\_UART, 4),
  - `kStatus_UART_RxWatermarkTooLarge` = MAKE\_STATUS(kStatusGroup\_UART, 5),
  - `kStatus_UART_FlagCannotClearManually`,
  - `kStatus_UART_Error` = MAKE\_STATUS(kStatusGroup\_UART, 7),
  - `kStatus_UART_RxRingBufferOverflow` = MAKE\_STATUS(kStatusGroup\_UART, 8),
  - `kStatus_UART_RxHardwareOverflow` = MAKE\_STATUS(kStatusGroup\_UART, 9),
  - `kStatus_UART_NoiseError` = MAKE\_STATUS(kStatusGroup\_UART, 10),
  - `kStatus_UART_FramingError` = MAKE\_STATUS(kStatusGroup\_UART, 11),
  - `kStatus_UART_ParityError` = MAKE\_STATUS(kStatusGroup\_UART, 12),
  - `kStatus_UART_BaudrateNotSupport` = MAKE\_STATUS(kStatusGroup\_UART, 13) }

*Error codes for the UART driver.*
- enum `uart_parity_mode_t` {
  - `kUART_ParityDisabled` = 0x0U,
  - `kUART_ParityEven` = 0x2U,
  - `kUART_ParityOdd` = 0x3U }

*UART parity mode.*
- enum `uart_stop_bit_count_t` {
  - `kUART_OneStopBit` = 0U,
  - `kUART_TwoStopBit` = 1U }

*UART stop bit count.*
- enum `_uart_interrupt_enable` {
  - `kUART_RxActiveEdgeInterruptEnable` = (UART\_BDH\_RXEDGIE\_MASK),
  - `kUART_TxDataRegEmptyInterruptEnable` = (UART\_C2\_TIE\_MASK << 8),
  - `kUART_TransmissionCompleteInterruptEnable` = (UART\_C2\_TCIE\_MASK << 8),
  - `kUART_RxDataRegFullInterruptEnable` = (UART\_C2\_RIE\_MASK << 8),
  - `kUART_IdleLineInterruptEnable` = (UART\_C2\_ILIE\_MASK << 8),
  - `kUART_RxOverflowInterruptEnable` = (UART\_C3\_ORIE\_MASK << 16),
  - `kUART_NoiseErrorInterruptEnable` = (UART\_C3\_NEIE\_MASK << 16),
  - `kUART_FramingErrorInterruptEnable` = (UART\_C3\_FEIE\_MASK << 16),
  - `kUART_ParityErrorInterruptEnable` = (UART\_C3\_PEIE\_MASK << 16) }

*UART interrupt configuration structure, default settings all disabled.*
- enum `_uart_flags` {

## UART Driver

```
kUART_TxDataRegEmptyFlag = (UART_S1_TDRE_MASK),
kUART_TransmissionCompleteFlag = (UART_S1_TC_MASK),
kUART_RxDataRegFullFlag = (UART_S1_RDRF_MASK),
kUART_IdleLineFlag = (UART_S1_IDLE_MASK),
kUART_RxOverrunFlag = (UART_S1_OR_MASK),
kUART_NoiseErrorFlag = (UART_S1_NF_MASK),
kUART_FramingErrorFlag = (UART_S1_FE_MASK),
kUART_ParityErrorFlag = (UART_S1_PF_MASK),
kUART_RxActiveEdgeFlag = (UART_S2_RXEDGIF_MASK << 8),
kUART_RxActiveFlag = (UART_S2_RAF_MASK << 8) }
UART status flags.
```

## Driver version

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

## Initialization and deinitialization

- status\_t **UART\_Init** (UART\_Type \*base, const **uart\_config\_t** \*config, uint32\_t srcClock\_Hz)  
*Initializes a UART instance with user configuration structure and peripheral clock.*
- void **UART\_Deinit** (UART\_Type \*base)  
*Deinitializes a UART instance.*
- void **UART\_GetDefaultConfig** (**uart\_config\_t** \*config)  
*Gets the default configuration structure.*
- status\_t **UART\_SetBaudRate** (UART\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the UART instance baud rate.*

## Status

- uint32\_t **UART\_GetStatusFlags** (UART\_Type \*base)  
*Get UART status flags.*
- status\_t **UART\_ClearStatusFlags** (UART\_Type \*base, uint32\_t mask)  
*Clears status flags with the provided mask.*

## Interrupts

- void **UART\_EnableInterrupts** (UART\_Type \*base, uint32\_t mask)  
*Enables UART interrupts according to the provided mask.*
- void **UART\_DisableInterrupts** (UART\_Type \*base, uint32\_t mask)  
*Disables the UART interrupts according to the provided mask.*
- uint32\_t **UART\_GetEnabledInterrupts** (UART\_Type \*base)  
*Gets the enabled UART interrupts.*



## Bus Operations

- static void [UART\\_EnableTx](#) (UART\_Type \*base, bool enable)  
*Enables or disables the UART transmitter.*
- static void [UART\\_EnableRx](#) (UART\_Type \*base, bool enable)  
*Enables or disables the UART receiver.*
- static void [UART\\_WriteByte](#) (UART\_Type \*base, uint8\_t data)  
*Writes to the TX register.*
- static uint8\_t [UART\\_ReadByte](#) (UART\_Type \*base)  
*Reads the RX register directly.*
- void [UART\\_WriteBlocking](#) (UART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*
- status\_t [UART\\_ReadBlocking](#) (UART\_Type \*base, uint8\_t \*data, size\_t length)  
*Read RX data register using a blocking method.*

## Transactional

- void [UART\\_TransferCreateHandle](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the UART handle.*
- void [UART\\_TransferStartRingBuffer](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void [UART\\_TransferStopRingBuffer](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- status\_t [UART\\_TransferSendNonBlocking](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_t](#) \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void [UART\\_TransferAbortSend](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the interrupt driven data transmit.*
- status\_t [UART\\_TransferGetSendCount](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been written to UART TX register.*
- status\_t [UART\\_TransferReceiveNonBlocking](#) (UART\_Type \*base, uart\_handle\_t \*handle, [uart\\_transfer\\_t](#) \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using an interrupt method.*
- void [UART\\_TransferAbortReceive](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*Aborts the interrupt-driven data receiving.*
- status\_t [UART\\_TransferGetReceiveCount](#) (UART\_Type \*base, uart\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*
- void [UART\\_TransferHandleIRQ](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*UART IRQ handle function.*
- void [UART\\_TransferHandleErrorIRQ](#) (UART\_Type \*base, uart\_handle\_t \*handle)  
*UART Error IRQ handle function.*

### 29.2.3 Data Structure Documentation

#### 29.2.3.1 struct uart\_config\_t

##### Data Fields

- uint32\_t [baudRate\\_Bps](#)  
*UART baud rate.*
- [uart\\_parity\\_mode\\_t](#) [parityMode](#)  
*Parity mode, disabled (default), even, odd.*
- bool [enableTx](#)  
*Enable TX.*
- bool [enableRx](#)  
*Enable RX.*

#### 29.2.3.2 struct uart\_transfer\_t

##### Data Fields

- uint8\_t \* [data](#)  
*The buffer of data to be transfer.*
- size\_t [dataSize](#)  
*The byte count to be transfer.*

##### 29.2.3.2.0.45 Field Documentation

###### 29.2.3.2.0.45.1 uint8\_t\* uart\_transfer\_t::data

###### 29.2.3.2.0.45.2 size\_t uart\_transfer\_t::dataSize

#### 29.2.3.3 struct \_uart\_handle

##### Data Fields

- uint8\_t \*volatile [txData](#)  
*Address of remaining data to send.*
- volatile size\_t [txDataSize](#)  
*Size of the remaining data to send.*
- size\_t [txDataSizeAll](#)  
*Size of the data to send out.*
- uint8\_t \*volatile [rxData](#)  
*Address of remaining data to receive.*
- volatile size\_t [rxDataSize](#)  
*Size of the remaining data to receive.*
- size\_t [rxDataSizeAll](#)  
*Size of the data to receive.*
- uint8\_t \* [rxRingBuffer](#)  
*Start address of the receiver ring buffer.*
- size\_t [rxRingBufferSize](#)

- *Size of the ring buffer.*  
volatile uint16\_t [rxRingBufferHead](#)
- *Index for the driver to store received data into ring buffer.*  
volatile uint16\_t [rxRingBufferTail](#)
- *Index for the user to get data from the ring buffer.*  
[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.*

## UART Driver

### 29.2.3.3.0.46 Field Documentation

- 29.2.3.3.0.46.1 `uint8_t* volatile uart_handle_t::txData`
- 29.2.3.3.0.46.2 `volatile size_t uart_handle_t::txDataSize`
- 29.2.3.3.0.46.3 `size_t uart_handle_t::txDataSizeAll`
- 29.2.3.3.0.46.4 `uint8_t* volatile uart_handle_t::rxData`
- 29.2.3.3.0.46.5 `volatile size_t uart_handle_t::rxDataSize`
- 29.2.3.3.0.46.6 `size_t uart_handle_t::rxDataSizeAll`
- 29.2.3.3.0.46.7 `uint8_t* uart_handle_t::rxRingBuffer`
- 29.2.3.3.0.46.8 `size_t uart_handle_t::rxRingBufferSize`
- 29.2.3.3.0.46.9 `volatile uint16_t uart_handle_t::rxRingBufferHead`
- 29.2.3.3.0.46.10 `volatile uint16_t uart_handle_t::rxRingBufferTail`
- 29.2.3.3.0.46.11 `uart_transfer_callback_t uart_handle_t::callback`
- 29.2.3.3.0.46.12 `void* uart_handle_t::userData`
- 29.2.3.3.0.46.13 `volatile uint8_t uart_handle_t::txState`

### 29.2.4 Macro Definition Documentation

- 29.2.4.1 `#define FSL_UART_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

### 29.2.5 Typedef Documentation

- 29.2.5.1 `typedef void(* uart_transfer_callback_t)(UART_Type *base, uart_handle_t *handle, status_t status, void *userData)`

### 29.2.6 Enumeration Type Documentation

#### 29.2.6.1 `enum _uart_status`

Enumerator

- kStatus\_UART\_TxBusy* Transmitter is busy.
- kStatus\_UART\_RxBusy* Receiver is busy.
- kStatus\_UART\_TxIdle* UART transmitter is idle.
- kStatus\_UART\_RxIdle* UART receiver is idle.
- kStatus\_UART\_TxWatermarkTooLarge* TX FIFO watermark too large.

*kStatus\_UART\_RxWatermarkTooLarge* RX FIFO watermark too large.  
*kStatus\_UART\_FlagCannotClearManually* UART flag can't be manually cleared.  
*kStatus\_UART\_Error* Error happens on UART.  
*kStatus\_UART\_RxRingBufferOverflow* UART RX software ring buffer overrun.  
*kStatus\_UART\_RxHardwareOverflow* UART RX receiver overrun.  
*kStatus\_UART\_NoiseError* UART noise error.  
*kStatus\_UART\_FramingError* UART framing error.  
*kStatus\_UART\_ParityError* UART parity error.  
*kStatus\_UART\_BaudrateNotSupport* Baudrate is not support in current clock source.

### 29.2.6.2 enum uart\_parity\_mode\_t

Enumerator

*kUART\_ParityDisabled* Parity disabled.  
*kUART\_ParityEven* Parity enabled, type even, bit setting: PE|PT = 10.  
*kUART\_ParityOdd* Parity enabled, type odd, bit setting: PE|PT = 11.

### 29.2.6.3 enum uart\_stop\_bit\_count\_t

Enumerator

*kUART\_OneStopBit* One stop bit.  
*kUART\_TwoStopBit* Two stop bits.

### 29.2.6.4 enum \_uart\_interrupt\_enable

This structure contains the settings for all of the UART interrupt configurations.

Enumerator

*kUART\_RxActiveEdgeInterruptEnable* RX active edge interrupt.  
*kUART\_TxDataRegEmptyInterruptEnable* Transmit data register empty interrupt.  
*kUART\_TransmissionCompleteInterruptEnable* Transmission complete interrupt.  
*kUART\_RxDataRegFullInterruptEnable* Receiver data register full interrupt.  
*kUART\_IdleLineInterruptEnable* Idle line interrupt.  
*kUART\_RxOverflowInterruptEnable* Receiver overrun interrupt.  
*kUART\_NoiseErrorInterruptEnable* Noise error flag interrupt.  
*kUART\_FramingErrorInterruptEnable* Framing error flag interrupt.  
*kUART\_ParityErrorInterruptEnable* Parity error flag interrupt.

## UART Driver

### 29.2.6.5 enum \_uart\_flags

This provides constants for the UART status flags for use in the UART functions.

Enumerator

***kUART\_TxDataRegEmptyFlag*** TX data register empty flag.  
***kUART\_TransmissionCompleteFlag*** Transmission complete flag.  
***kUART\_RxDataRegFullFlag*** RX data register full flag.  
***kUART\_IdleLineFlag*** Idle line detect flag.  
***kUART\_RxOverrunFlag*** RX overrun flag.  
***kUART\_NoiseErrorFlag*** RX takes 3 samples of each received bit. If any of these samples differ, noise flag sets  
***kUART\_FramingErrorFlag*** Frame error flag, sets if logic 0 was detected where stop bit expected.  
***kUART\_ParityErrorFlag*** If parity enabled, sets upon parity error detection.  
***kUART\_RxActiveEdgeFlag*** RX pin active edge interrupt flag, sets when active edge detected.  
***kUART\_RxActiveFlag*** Receiver Active Flag (RAF), sets at beginning of valid start bit.

### 29.2.7 Function Documentation

#### 29.2.7.1 status\_t UART\_Init ( UART\_Type \* *base*, const uart\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

This function configures the UART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the [UART\\_GetDefaultConfig\(\)](#) function. Example below shows how to use this API to configure UART.

```
* uart_config_t uartConfig;  
* uartConfig.baudRate_Bps = 115200U;  
* uartConfig.parityMode = kUART_ParityDisabled;  
* uartConfig.stopBitCount = kUART_OneStopBit;  
* uartConfig.txFifoWatermark = 0;  
* uartConfig.rxFifoWatermark = 1;  
* UART_Init(UART1, &uartConfig, 200000000U);  
*
```

Parameters

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

|                    |                                    |
|--------------------|------------------------------------|
| <i>srcClock_Hz</i> | UART clock source frequency in HZ. |
|--------------------|------------------------------------|

Return values

|   |  |
|---|--|
| <i>kStatus_UART_Baudrate-NotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                  | Status UART initialize succeed                   |

### 29.2.7.2 void UART\_Deinit ( UART\_Type \* *base* )

This function waits for TX complete, disables TX and RX, and disables the UART clock.

Parameters

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

### 29.2.7.3 void UART\_GetDefaultConfig ( uart\_config\_t \* *config* )

This function initializes the UART configuration structure to a default value. The default values are: uartConfig->baudRate\_Bps = 115200U; uartConfig->bitCountPerChar = kUART\_8BitsPerChar; uartConfig->parityMode = kUART\_ParityDisabled; uartConfig->stopBitCount = kUART\_OneStopBit; uartConfig->txFifoWatermark = 0; uartConfig->rxFifoWatermark = 1; uartConfig->enableTx = false; uartConfig->enableRx = false;

Parameters

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

### 29.2.7.4 status\_t UART\_SetBaudRate ( UART\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the UART module baud rate. This function is used to update the UART module baud rate after the UART module is initialized by the UART\_Init.

```
* UART_SetBaudRate(UART1, 115200U, 200000000U);
*
```

## UART Driver

### Parameters

|                     |                                    |
|---------------------|------------------------------------|
| <i>base</i>         | UART peripheral base address.      |
| <i>baudRate_Bps</i> | UART baudrate to be set.           |
| <i>srcClock_Hz</i>  | UART clock source frequency in HZ. |

### Return values

|   |  |
|---|--|
| <i>kStatus_UART_Baudrate-NotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                  | Set baudrate succeed                             |

### 29.2.7.5 uint32\_t UART\_GetStatusFlags ( UART\_Type \* *base* )

This function get all UART status flags, the flags are returned as the logical OR value of the enumerators [\\_uart\\_flags](#). To check a specific status, compare the return value with enumerators in [\\_uart\\_flags](#). For example, to check whether the TX is empty:

```
*      if (kUART_TxDataRegEmptyFlag & UART_GetStatusFlags(UART1))
*      {
*          ...
*      }
*
```

### Parameters

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

### Returns

UART status flags which are ORed by the enumerators in the [\\_uart\\_flags](#).

### 29.2.7.6 status\_t UART\_ClearStatusFlags ( UART\_Type \* *base*, uint32\_t *mask* )

This function clears UART status flags with a provided mask. Automatically cleared flag can't be cleared by this function. Some flags can only be cleared or set by hardware itself. These flags are: kUART\_TxDataRegEmptyFlag, kUART\_TransmissionCompleteFlag, kUART\_RxDataRegFullFlag, kUART\_RxActiveFlag, kUART\_NoiseErrorInRxDataRegFlag, kUART\_ParityErrorInRxDataRegFlag, kUART\_TxFifoEmptyFlag, kUART\_RxFifoEmptyFlag. Note: This API should be called when the Tx/Rx is idle, otherwise it takes no effects.



## Parameters

|             |   |
|-------------|---|
| <i>base</i> | UART peripheral base address.   |
| <i>mask</i> | The status flags to be cleared, it is logical OR value of <a href="#">_uart_flags</a> . |

## Return values

|   |   |
|---|---|
| <i>kStatus_UART_Flag-<br/>CannotClearManually</i> | The flag can't be cleared by this function but it is cleared automatically by hardware. |
| <i>kStatus_Success</i>                            | Status in the mask are cleared.   |

**29.2.7.7 void UART\_EnableInterrupts ( UART\_Type \* *base*, uint32\_t *mask* )**

This function enables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_uart\\_interrupt\\_enable](#). For example, to enable TX empty interrupt and RX full interrupt:

```
*  UART_EnableInterrupts(UART1,
    kUART_TxDataRegEmptyInterruptEnable |
    kUART_RxDataRegFullInterruptEnable);
*
```

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | UART peripheral base address.  |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_uart_interrupt_enable</a> . |

**29.2.7.8 void UART\_DisableInterrupts ( UART\_Type \* *base*, uint32\_t *mask* )**

This function disables the UART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_uart\\_interrupt\\_enable](#). For example, to disable TX empty interrupt and RX full interrupt:

```
*  UART_DisableInterrupts(UART1,
    kUART_TxDataRegEmptyInterruptEnable |
    kUART_RxDataRegFullInterruptEnable);
*
```

## UART Driver

### Parameters

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

### 29.2.7.9 uint32\_t UART\_GetEnabledInterrupts ( UART\_Type \* *base* )

This function gets the enabled UART interrupts. The enabled interrupts are returned as the logical OR value of the enumerators [\\_uart\\_interrupt\\_enable](#). To check a specific interrupts enable status, compare the return value with enumerators in [\\_uart\\_interrupt\\_enable](#). For example, to check whether TX empty interrupt is enabled:

```
*    uint32_t enabledInterrupts = UART_GetEnabledInterrupts(UART1);  
*  
*    if (kUART_TxDataRegEmptyInterruptEnable & enabledInterrupts)  
*    {  
*        ...  
*    }  
*
```

### Parameters

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

### Returns

UART interrupt flags which are logical OR of the enumerators in [\\_uart\\_interrupt\\_enable](#).

### 29.2.7.10 static void UART\_EnableTx ( UART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the UART transmitter.

### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | UART peripheral base address.     |
| <i>enable</i> | True to enable, false to disable. |

### 29.2.7.11 static void UART\_EnableRx ( UART\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the UART receiver.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | UART peripheral base address.     |
| <i>enable</i> | True to enable, false to disable. |

### 29.2.7.12 static void UART\_WriteByte ( UART\_Type \* *base*, uint8\_t *data* ) [inline], [static]

This function writes data to the TX register directly. The upper layer must ensure that the TX register is empty or TX FIFO has empty room before calling this function.

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | UART peripheral base address. |
| <i>data</i> | The byte to write.            |

### 29.2.7.13 static uint8\_t UART\_ReadByte ( UART\_Type \* *base* ) [inline], [static]

This function reads data from the TX register directly. The upper layer must ensure that the RX register is full or that the TX FIFO has data before calling this function.

## Parameters

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

## Returns

The byte read from UART data register.

### 29.2.7.14 void UART\_WriteBlocking ( UART\_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.

## Note

This function does not check whether all the data has been sent out to the bus. Before disabling the TX, check kUART\_TransmissionCompleteFlag to ensure that the TX is finished.

## UART Driver

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | UART peripheral base address.       |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

### 29.2.7.15 **status\_t UART\_ReadBlocking ( UART\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )**

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data and read data from the TX register.

### Parameters

|               |   |
|---------------|---|
| <i>base</i>   | UART peripheral base address.                           |
| <i>data</i>   | Start address of the buffer to store the received data. |
| <i>length</i> | Size of the buffer.                                     |

### Return values

|  |   |
|--|---|
| <i>kStatus_UART_Rx-HardwareOverrun</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_UART_Noise-Error</i>        | Noise error happened while receiving data.      |
| <i>kStatus_UART_Framing-Error</i>      | Framing error happened while receiving data.    |
| <i>kStatus_UART_Parity-Error</i>       | Parity error happened while receiving data.     |
| <i>kStatus_Success</i>                 | Successfully received all data.                 |

### 29.2.7.16 **void UART\_TransferCreateHandle ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, uart\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the UART handle which can be used for other UART transactional APIs. Usually, for a specified UART instance, call this API once to get the initialized handle.

## Parameters

|                 |   |
|-----------------|---|
| <i>base</i>     | UART peripheral base address.           |
| <i>handle</i>   | UART handle pointer.                    |
| <i>callback</i> | The callback function.                  |
| <i>userData</i> | The parameter of the callback function. |

### 29.2.7.17 void UART\_TransferStartRingBuffer ( UART\_Type \* *base*, 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 are stored into the ring buffer even when the user doesn't call the [UART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

## Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

## Parameters

|                       |  |
|-----------------------|--|
| <i>base</i>           | UART peripheral base address.  |
| <i>handle</i>         | UART handle pointer.   |
| <i>ringBuffer</i>     | Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.   |

### 29.2.7.18 void UART\_TransferStopRingBuffer ( UART\_Type \* *base*, uart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

## Parameters

---

## UART Driver

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |

### 29.2.7.19 **status\_t UART\_TransferSendNonBlocking ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, 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 the ISR, the UART driver calls the callback function and passes the [kStatus\\_UART\\_TxIdle](#) as status parameter.

#### Note

The [kStatus\\_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 are sent out. Before disabling the TX, check the [kUART\\_TransmissionCompleteFlag](#) to ensure that the TX is finished.

#### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | UART peripheral base address.                                  |
| <i>handle</i> | UART handle pointer.   |
| <i>xfer</i>   | UART transfer structure. See <a href="#">uart_transfer_t</a> . |

#### Return values

|                                |  |
|--------------------------------|--|
| <i>kStatus_Success</i>         | Successfully start the data transmission.  |
| <i>kStatus_UART_TxBusy</i>     | Previous transmission still not finished, data not all written to TX register yet. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.  |

### 29.2.7.20 **void UART\_TransferAbortSend ( UART\_Type \* *base*, uart\_handle\_t \* *handle* )**

This function aborts the interrupt driven data sending. The user can get the `remainBytes` to find out how many bytes are still not sent out.

#### Parameters

---

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |

#### 29.2.7.21 **status\_t UART\_TransferGetSendCount ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been written to UART TX register by interrupt method.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |
| <i>count</i>  | Send bytes count.             |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

#### 29.2.7.22 **status\_t UART\_TransferReceiveNonBlocking ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, uart\_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 UART driver. When the 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, 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 UART 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.

## UART Driver

### Parameters

|                      |  |
|----------------------|--|
| <i>base</i>          | UART peripheral base address.                                  |
| <i>handle</i>        | UART handle pointer.   |
| <i>xfer</i>          | UART transfer structure, see <a href="#">uart_transfer_t</a> . |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.                  |

### Return values

|                                |  |
|--------------------------------|--|
| <i>kStatus_Success</i>         | Successfully queue the transfer into transmit queue. |
| <i>kStatus_UART_RxBusy</i>     | Previous receive request is not finished.            |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                    |

#### 29.2.7.23 void UART\_TransferAbortReceive ( UART\_Type \* *base*, uart\_handle\_t \* *handle* )

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to know how many bytes not received yet.

### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |

#### 29.2.7.24 status\_t UART\_TransferGetReceiveCount ( UART\_Type \* *base*, uart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |
| <i>count</i>  | Receive bytes count.          |



Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

#### 29.2.7.25 void UART\_TransferHandleIRQ ( UART\_Type \* *base*, uart\_handle\_t \* *handle* )

This function handles the UART transmit and receive IRQ request.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |

#### 29.2.7.26 void UART\_TransferHandleErrorIRQ ( UART\_Type \* *base*, uart\_handle\_t \* *handle* )

This function handle the UART error IRQ request.

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |

### 29.3 UART DMA Driver

#### 29.3.1 Overview

##### Data Structures

- struct `uart_dma_handle_t`  
*UART DMA handle. [More...](#)*

##### Typedefs

- typedef void(\* `uart_dma_transfer_callback_t`)(UART\_Type \*base, uart\_dma\_handle\_t \*handle, status\_t status, void \*userData)  
*UART transfer callback function.*

##### eDMA transactional

- void `UART_TransferCreateHandleDMA` (UART\_Type \*base, uart\_dma\_handle\_t \*handle, `uart_dma_transfer_callback_t` callback, void \*userData, `dma_handle_t` \*txDmaHandle, `dma_handle_t` \*rxDmaHandle)  
*Initializes the UART handle which is used in transactional functions and sets the callback.*
- status\_t `UART_TransferSendDMA` (UART\_Type \*base, uart\_dma\_handle\_t \*handle, `uart_transfer_t` \*xfer)  
*Sends data using DMA.*
- status\_t `UART_TransferReceiveDMA` (UART\_Type \*base, uart\_dma\_handle\_t \*handle, `uart_transfer_t` \*xfer)  
*Receives data using DMA.*
- void `UART_TransferAbortSendDMA` (UART\_Type \*base, uart\_dma\_handle\_t \*handle)  
*Aborts the send data using DMA.*
- void `UART_TransferAbortReceiveDMA` (UART\_Type \*base, uart\_dma\_handle\_t \*handle)  
*Aborts the received data using DMA.*
- status\_t `UART_TransferGetSendCountDMA` (UART\_Type \*base, uart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been written to UART TX register.*
- status\_t `UART_TransferGetReceiveCountDMA` (UART\_Type \*base, uart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*

#### 29.3.2 Data Structure Documentation

##### 29.3.2.1 struct `_uart_dma_handle`

##### Data Fields

- UART\_Type \* `base`

- *UART peripheral base address.*  
• `uart_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.*

#### 29.3.2.1.0.47 Field Documentation

29.3.2.1.0.47.1 `UART_Type* uart_dma_handle_t::base`

29.3.2.1.0.47.2 `uart_dma_transfer_callback_t uart_dma_handle_t::callback`

29.3.2.1.0.47.3 `void* uart_dma_handle_t::userData`

29.3.2.1.0.47.4 `size_t uart_dma_handle_t::rxDataSizeAll`

29.3.2.1.0.47.5 `size_t uart_dma_handle_t::txDataSizeAll`

29.3.2.1.0.47.6 `dma_handle_t* uart_dma_handle_t::txDmaHandle`

29.3.2.1.0.47.7 `dma_handle_t* uart_dma_handle_t::rxDmaHandle`

29.3.2.1.0.47.8 `volatile uint8_t uart_dma_handle_t::txState`

#### 29.3.3 Typedef Documentation

29.3.3.1 `typedef void(* uart_dma_transfer_callback_t)(UART_Type *base,  
uart_dma_handle_t *handle, status_t status, void *userData)`

#### 29.3.4 Function Documentation

29.3.4.1 `void UART_TransferCreateHandleDMA ( UART_Type * base, uart_dma_handle_t  
* handle, uart_dma_transfer_callback_t callback, void * userData,  
dma_handle_t * txDmaHandle, dma_handle_t * rxDmaHandle )`

## UART DMA Driver

### Parameters

|                    |  |
|--------------------|--|
| <i>base</i>        | UART peripheral base address.                        |
| <i>handle</i>      | Pointer to <code>uart_dma_handle_t</code> structure. |
| <i>callback</i>    | UART callback, NULL means no callback.               |
| <i>userData</i>    | User callback function data.                         |
| <i>rxDmaHandle</i> | User requested DMA handle for RX DMA transfer.       |
| <i>txDmaHandle</i> | User requested DMA handle for TX DMA transfer.       |

### 29.3.4.2 **status\_t UART\_TransferSendDMA ( UART\_Type \* *base*, `uart_dma_handle_t` \* *handle*, `uart_transfer_t` \* *xfer* )**

This function sends data using DMA. This is non-blocking function, which returns right away. When all data is sent, the send callback function is called.

### Parameters

|               |  |
|---------------|--|
| <i>base</i>   | UART peripheral base address.                                      |
| <i>handle</i> | UART handle pointer.   |
| <i>xfer</i>   | UART DMA transfer structure. See <a href="#">uart_transfer_t</a> . |

### Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_UART_TxBusy</i>     | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

### 29.3.4.3 **status\_t UART\_TransferReceiveDMA ( UART\_Type \* *base*, `uart_dma_handle_t` \* *handle*, `uart_transfer_t` \* *xfer* )**

This function receives data using DMA. This is non-blocking function, which returns right away. When all data is received, the receive callback function is called.

### Parameters

---

|               |  |
|---------------|--|
| <i>base</i>   | UART peripheral base address.                                      |
| <i>handle</i> | Pointer to <code>uart_dma_handle_t</code> structure.               |
| <i>xfer</i>   | UART DMA transfer structure. See <a href="#">uart_transfer_t</a> . |

Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_UART_RxBusy</i>     | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

#### 29.3.4.4 void UART\_TransferAbortSendDMA ( UART\_Type \* *base*, `uart_dma_handle_t` \* *handle* )

This function aborts the sent data using DMA.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | UART peripheral base address.                        |
| <i>handle</i> | Pointer to <code>uart_dma_handle_t</code> structure. |

#### 29.3.4.5 void UART\_TransferAbortReceiveDMA ( UART\_Type \* *base*, `uart_dma_handle_t` \* *handle* )

This function abort receive data which using DMA.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | UART peripheral base address.                        |
| <i>handle</i> | Pointer to <code>uart_dma_handle_t</code> structure. |

#### 29.3.4.6 status\_t UART\_TransferGetSendCountDMA ( UART\_Type \* *base*, `uart_dma_handle_t` \* *handle*, `uint32_t` \* *count* )

This function gets the number of bytes that have been written to UART TX register by DMA.

## UART DMA Driver

### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |
| <i>count</i>  | Send bytes count.             |

### Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

#### 29.3.4.7 **status\_t** UART\_TransferGetReceiveCountDMA ( **UART\_Type** \* *base*, **uart\_dma\_handle\_t** \* *handle*, **uint32\_t** \* *count* )

This function gets the number of bytes that have been received.

### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |
| <i>count</i>  | Receive bytes count.          |

### Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                                     |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

## 29.4 UART eDMA Driver

### 29.4.1 Overview

#### Data Structures

- struct [uart\\_edma\\_handle\\_t](#)  
UART eDMA handle. [More...](#)

#### Typedefs

- typedef void(\* [uart\\_edma\\_transfer\\_callback\\_t](#))(UART\_Type \*base, uart\_edma\_handle\_t \*handle, status\_t status, void \*userData)  
UART transfer callback function.

#### eDMA transactional

- void [UART\\_TransferCreateHandleEDMA](#) (UART\_Type \*base, uart\_edma\_handle\_t \*handle, [uart\\_edma\\_transfer\\_callback\\_t](#) callback, void \*userData, edma\_handle\_t \*txEdmaHandle, edma\_handle\_t \*rxEdmaHandle)  
*Initializes the UART handle which is used in transactional functions.*
- status\_t [UART\\_SendEDMA](#) (UART\_Type \*base, uart\_edma\_handle\_t \*handle, [uart\\_transfer\\_t](#) \*xfer)  
*Sends data using eDMA.*
- status\_t [UART\\_ReceiveEDMA](#) (UART\_Type \*base, uart\_edma\_handle\_t \*handle, [uart\\_transfer\\_t](#) \*xfer)  
*Receive data using eDMA.*
- void [UART\\_TransferAbortSendEDMA](#) (UART\_Type \*base, uart\_edma\_handle\_t \*handle)  
*Aborts the sent data using eDMA.*
- void [UART\\_TransferAbortReceiveEDMA](#) (UART\_Type \*base, uart\_edma\_handle\_t \*handle)  
*Aborts the receive data using eDMA.*
- status\_t [UART\\_TransferGetSendCountEDMA](#) (UART\_Type \*base, uart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been written to UART TX register.*
- status\_t [UART\\_TransferGetReceiveCountEDMA](#) (UART\_Type \*base, uart\_edma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*

### 29.4.2 Data Structure Documentation

#### 29.4.2.1 struct \_uart\_edma\_handle

##### Data Fields

- [uart\\_edma\\_transfer\\_callback\\_t](#) callback

## UART eDMA Driver

- 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.*
- edma\_handle\_t \* [txEdmaHandle](#)  
*The eDMA TX channel used.*
- edma\_handle\_t \* [rxEdmaHandle](#)  
*The eDMA RX channel used.*
- volatile uint8\_t [txState](#)  
*TX transfer state.*
- volatile uint8\_t [rxState](#)  
*RX transfer state.*

### 29.4.2.1.0.48 Field Documentation

29.4.2.1.0.48.1 `uart_edma_transfer_callback_t` `uart_edma_handle_t::callback`

29.4.2.1.0.48.2 `void*` `uart_edma_handle_t::userData`

29.4.2.1.0.48.3 `size_t` `uart_edma_handle_t::rxDataSizeAll`

29.4.2.1.0.48.4 `size_t` `uart_edma_handle_t::txDataSizeAll`

29.4.2.1.0.48.5 `edma_handle_t*` `uart_edma_handle_t::txEdmaHandle`

29.4.2.1.0.48.6 `edma_handle_t*` `uart_edma_handle_t::rxEdmaHandle`

29.4.2.1.0.48.7 `volatile uint8_t` `uart_edma_handle_t::txState`

### 29.4.3 Typedef Documentation

29.4.3.1 `typedef void(* uart_edma_transfer_callback_t)(UART_Type *base,  
uart_edma_handle_t *handle, status_t status, void *userData)`

### 29.4.4 Function Documentation

29.4.4.1 `void UART_TransferCreateHandleEDMA ( UART_Type * base,  
uart_edma_handle_t * handle, uart_edma_transfer_callback_t callback, void *  
userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle )`



## Parameters

|                     |   |
|---------------------|---|
| <i>base</i>         | UART peripheral base address.                         |
| <i>handle</i>       | Pointer to <code>uart_edma_handle_t</code> structure. |
| <i>callback</i>     | UART callback, NULL means no callback.                |
| <i>userData</i>     | User callback function data.                          |
| <i>rxEdmaHandle</i> | User requested DMA handle for RX DMA transfer.        |
| <i>txEdmaHandle</i> | User requested DMA handle for TX DMA transfer.        |

#### 29.4.4.2 **status\_t UART\_SendEDMA ( UART\_Type \* *base*, uart\_edma\_handle\_t \* *handle*, uart\_transfer\_t \* *xfer* )**

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | UART peripheral base address.                                       |
| <i>handle</i> | UART handle pointer.  |
| <i>xfer</i>   | UART eDMA transfer structure. See <a href="#">uart_transfer_t</a> . |

## Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_UART_TxBusy</i>     | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

#### 29.4.4.3 **status\_t UART\_ReceiveEDMA ( UART\_Type \* *base*, uart\_edma\_handle\_t \* *handle*, uart\_transfer\_t \* *xfer* )**

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

## Parameters

---

## UART eDMA Driver

|               |   |
|---------------|---|
| <i>base</i>   | UART peripheral base address.                                       |
| <i>handle</i> | Pointer to uart_edma_handle_t structure.                            |
| <i>xfer</i>   | UART eDMA transfer structure. See <a href="#">uart_transfer_t</a> . |

Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_UART_RxBusy</i>     | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

### 29.4.4.4 void UART\_TransferAbortSendEDMA ( UART\_Type \* *base*, uart\_edma\_handle\_t \* *handle* )

This function aborts sent data using eDMA.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | UART peripheral base address.            |
| <i>handle</i> | Pointer to uart_edma_handle_t structure. |

### 29.4.4.5 void UART\_TransferAbortReceiveEDMA ( UART\_Type \* *base*, uart\_edma\_handle\_t \* *handle* )

This function aborts receive data using eDMA.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | UART peripheral base address.            |
| <i>handle</i> | Pointer to uart_edma_handle_t structure. |

### 29.4.4.6 status\_t UART\_TransferGetSendCountEDMA ( UART\_Type \* *base*, uart\_edma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been written to UART TX register by DMA.

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |
| <i>count</i>  | Send bytes count.             |

## Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

#### 29.4.4.7 **status\_t** UART\_TransferGetReceiveCountEDMA ( **UART\_Type** \* *base*, **uart\_edma\_handle\_t** \* *handle*, **uint32\_t** \* *count* )

This function gets the number of bytes that have been received.

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | UART peripheral base address. |
| <i>handle</i> | UART handle pointer.          |
| <i>count</i>  | Receive bytes count.          |

## Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                                     |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                       |
| <i>kStatus_Success</i>              | Get successfully through the parameter <code>count</code> ; |

### 29.5 UART FreeRTOS Driver

#### 29.5.1 Overview

##### Data Structures

- struct [rtos\\_uart\\_config](#)  
*UART configuration structure. [More...](#)*
- struct [uart\\_rtos\\_handle\\_t](#)  
*UART FreeRTOS handle. [More...](#)*

##### UART RTOS Operation

- int [UART\\_RTOS\\_Init](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, [uart\\_handle\\_t](#) \*t\_handle, const struct [rtos\\_uart\\_config](#) \*cfg)  
*Initializes a UART instance for operation in RTOS.*
- int [UART\\_RTOS\\_Deinit](#) ([uart\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes a UART instance for operation.*

##### UART transactional Operation

- int [UART\\_RTOS\\_Send](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, const [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
*Sends data in the background.*
- int [UART\\_RTOS\\_Receive](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length, [size\\_t](#) \*received)  
*Receives data.*

#### 29.5.2 Data Structure Documentation

##### 29.5.2.1 struct [rtos\\_uart\\_config](#)

###### Data Fields

- [UART\\_Type](#) \* [base](#)  
*UART base address.*
- [uint32\\_t](#) [srcclk](#)  
*UART source clock in Hz.*
- [uint32\\_t](#) [baudrate](#)  
*Desired communication speed.*
- [uart\\_parity\\_mode\\_t](#) [parity](#)  
*Parity setting.*
- [uart\\_stop\\_bit\\_count\\_t](#) [stopbits](#)  
*Number of stop bits to use.*
- [uint8\\_t](#) \* [buffer](#)  
*Buffer for background reception.*
- [uint32\\_t](#) [buffer\\_size](#)

*Size of buffer for background reception.*

### 29.5.2.2 struct uart\_rtos\_handle\_t

#### Data Fields

- UART\_Type \* [base](#)  
*UART base address.*
- struct \_uart\_transfer [tx\\_xfer](#)  
*TX transfer structure.*
- struct \_uart\_transfer [rx\\_xfer](#)  
*RX transfer structure.*
- SemaphoreHandle\_t [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- SemaphoreHandle\_t [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- EventGroupHandle\_t [rx\\_event](#)  
*RX completion event.*
- EventGroupHandle\_t [tx\\_event](#)  
*TX completion event.*
- void \* [t\\_state](#)  
*Transactional state of the underlying driver.*
- OS\_EVENT \* [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- OS\_EVENT \* [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- OS\_FLAG\_GRP \* [rx\\_event](#)  
*RX completion event.*
- OS\_FLAG\_GRP \* [tx\\_event](#)  
*TX completion event.*
- OS\_SEM [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- OS\_SEM [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- OS\_FLAG\_GRP [rx\\_event](#)  
*RX completion event.*
- OS\_FLAG\_GRP [tx\\_event](#)  
*TX completion event.*

### 29.5.3 Function Documentation

**29.5.3.1** int UART\_RTOS\_Init ( uart\_rtos\_handle\_t \* *handle*, uart\_handle\_t \* *t\_handle*,  
const struct rtos\_uart\_config \* *cfg* )

## UART FreeRTOS Driver

### Parameters

|                 |  |
|-----------------|--|
| <i>handle</i>   | The RTOS UART handle, the pointer to allocated space for RTOS context.             |
| <i>t_handle</i> | The pointer to allocated space where to store transactional layer internal state.  |
| <i>cfg</i>      | The pointer to the parameters required to configure the UART after initialization. |

### Returns

0 succeed, others fail.

#### 29.5.3.2 int UART\_RTOS\_Deinit ( uart\_rtos\_handle\_t \* *handle* )

This function deinitializes the UART module, sets all register values to reset value, and releases the resources.

### Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | The RTOS UART handle. |
|---------------|-----------------------|

#### 29.5.3.3 int UART\_RTOS\_Send ( uart\_rtos\_handle\_t \* *handle*, const uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS UART handle.          |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 29.5.3.4 int UART\_RTOS\_Receive ( uart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from UART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

## Parameters

|                 |  |
|-----------------|--|
| <i>handle</i>   | The RTOS UART handle.  |
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.  |
| <i>received</i> | The pointer to a variable of size_t where the number of received data is filled. |

### 29.6 UART $\mu$ COS/II Driver

#### 29.6.1 Overview

##### Data Structures

- struct [rtos\\_uart\\_config](#)  
*UART configuration structure. [More...](#)*
- struct [uart\\_rtos\\_handle\\_t](#)  
*UART FreeRTOS handle. [More...](#)*

##### UART RTOS Operation

- int [UART\\_RTOS\\_Init](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, [uart\\_handle\\_t](#) \*t\_handle, const struct [rtos\\_uart\\_config](#) \*cfg)  
*Initializes a UART instance for operation in RTOS.*
- int [UART\\_RTOS\\_Deinit](#) ([uart\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes a UART instance for operation.*

##### UART transactional Operation

- int [UART\\_RTOS\\_Send](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, const [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
*Sends data in the background.*
- int [UART\\_RTOS\\_Receive](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length, [size\\_t](#) \*received)  
*Receives data.*

#### 29.6.2 Data Structure Documentation

##### 29.6.2.1 struct [rtos\\_uart\\_config](#)

###### Data Fields

- [UART\\_Type](#) \* [base](#)  
*UART base address.*
- [uint32\\_t](#) [srcclk](#)  
*UART source clock in Hz.*
- [uint32\\_t](#) [baudrate](#)  
*Desired communication speed.*
- [uart\\_parity\\_mode\\_t](#) [parity](#)  
*Parity setting.*
- [uart\\_stop\\_bit\\_count\\_t](#) [stopbits](#)  
*Number of stop bits to use.*
- [uint8\\_t](#) \* [buffer](#)  
*Buffer for background reception.*
- [uint32\\_t](#) [buffer\\_size](#)



*Size of buffer for background reception.*

### 29.6.2.2 struct uart\_rtos\_handle\_t

#### Data Fields

- UART\_Type \* [base](#)  
*UART base address.*
- struct \_uart\_transfer [tx\\_xfer](#)  
*TX transfer structure.*
- struct \_uart\_transfer [rx\\_xfer](#)  
*RX transfer structure.*
- SemaphoreHandle\_t [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- SemaphoreHandle\_t [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- EventGroupHandle\_t [rx\\_event](#)  
*RX completion event.*
- EventGroupHandle\_t [tx\\_event](#)  
*TX completion event.*
- void \* [t\\_state](#)  
*Transactional state of the underlying driver.*
- OS\_EVENT \* [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- OS\_EVENT \* [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- OS\_FLAG\_GRP \* [rx\\_event](#)  
*RX completion event.*
- OS\_FLAG\_GRP \* [tx\\_event](#)  
*TX completion event.*
- OS\_SEM [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- OS\_SEM [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- OS\_FLAG\_GRP [rx\\_event](#)  
*RX completion event.*
- OS\_FLAG\_GRP [tx\\_event](#)  
*TX completion event.*

### 29.6.3 Function Documentation

**29.6.3.1** int UART\_RTOS\_Init ( uart\_rtos\_handle\_t \* *handle*, uart\_handle\_t \* *t\_handle*,  
const struct rtos\_uart\_config \* *cfg* )

## UART $\mu$ COS/II Driver

### Parameters

|                      |  |
|----------------------|--|
| <i>handle</i>        | The RTOS UART handle, the pointer to allocated space for RTOS context.             |
| <i>uart_t_handle</i> | The pointer to allocated space where to store transactional layer internal state.  |
| <i>cfg</i>           | The pointer to the parameters required to configure the UART after initialization. |

### Returns

0 Succeed, others fail.

#### 29.6.3.2 int UART\_RTOS\_Deinit ( uart\_rtos\_handle\_t \* *handle* )

This function deinitializes the UART module, sets all register values to reset value, and releases the resources.

### Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | The RTOS UART handle. |
|---------------|-----------------------|

#### 29.6.3.3 int UART\_RTOS\_Send ( uart\_rtos\_handle\_t \* *handle*, const uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS UART handle.          |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 29.6.3.4 int UART\_RTOS\_Receive ( uart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from UART. It is a synchronous API. If any data is immediately available it is returned immediately and the number of bytes received.

## Parameters

|                 |  |
|-----------------|--|
| <i>handle</i>   | The RTOS UART handle.  |
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.  |
| <i>received</i> | The pointer to a variable of size_t where the number of received data is filled. |

### 29.7 UART $\mu$ COS/III Driver

#### 29.7.1 Overview

##### Data Structures

- struct [rtos\\_uart\\_config](#)  
*UART configuration structure. [More...](#)*
- struct [uart\\_rtos\\_handle\\_t](#)  
*UART FreeRTOS handle. [More...](#)*

##### UART RTOS Operation

- int [UART\\_RTOS\\_Init](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, [uart\\_handle\\_t](#) \*t\_handle, const struct [rtos\\_uart\\_config](#) \*cfg)  
*Initializes a UART instance for operation in RTOS.*
- int [UART\\_RTOS\\_Deinit](#) ([uart\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes a UART instance for operation.*

##### UART transactional Operation

- int [UART\\_RTOS\\_Send](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, const [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
*Sends data in the background.*
- int [UART\\_RTOS\\_Receive](#) ([uart\\_rtos\\_handle\\_t](#) \*handle, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length, [size\\_t](#) \*received)  
*Receives data.*

#### 29.7.2 Data Structure Documentation

##### 29.7.2.1 struct [rtos\\_uart\\_config](#)

###### Data Fields

- [UART\\_Type](#) \* [base](#)  
*UART base address.*
- [uint32\\_t](#) [srcclk](#)  
*UART source clock in Hz.*
- [uint32\\_t](#) [baudrate](#)  
*Desired communication speed.*
- [uart\\_parity\\_mode\\_t](#) [parity](#)  
*Parity setting.*
- [uart\\_stop\\_bit\\_count\\_t](#) [stopbits](#)  
*Number of stop bits to use.*
- [uint8\\_t](#) \* [buffer](#)  
*Buffer for background reception.*
- [uint32\\_t](#) [buffer\\_size](#)

*Size of buffer for background reception.*

### 29.7.2.2 struct uart\_rtos\_handle\_t

#### Data Fields

- UART\_Type \* [base](#)  
*UART base address.*
- struct \_uart\_transfer [tx\\_xfer](#)  
*TX transfer structure.*
- struct \_uart\_transfer [rx\\_xfer](#)  
*RX transfer structure.*
- SemaphoreHandle\_t [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- SemaphoreHandle\_t [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- EventGroupHandle\_t [rx\\_event](#)  
*RX completion event.*
- EventGroupHandle\_t [tx\\_event](#)  
*TX completion event.*
- void \* [t\\_state](#)  
*Transactional state of the underlying driver.*
- OS\_EVENT \* [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- OS\_EVENT \* [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- OS\_FLAG\_GRP \* [rx\\_event](#)  
*RX completion event.*
- OS\_FLAG\_GRP \* [tx\\_event](#)  
*TX completion event.*
- OS\_SEM [rx\\_sem](#)  
*RX semaphore for resource sharing.*
- OS\_SEM [tx\\_sem](#)  
*TX semaphore for resource sharing.*
- OS\_FLAG\_GRP [rx\\_event](#)  
*RX completion event.*
- OS\_FLAG\_GRP [tx\\_event](#)  
*TX completion event.*

### 29.7.3 Function Documentation

**29.7.3.1** int UART\_RTOS\_Init ( uart\_rtos\_handle\_t \* *handle*, uart\_handle\_t \* *t\_handle*,  
const struct rtos\_uart\_config \* *cfg* )

## UART $\mu$ COS/III Driver

### Parameters

|                      |  |
|----------------------|--|
| <i>handle</i>        | The RTOS UART handle, the pointer to allocated space for RTOS context.               |
| <i>uart_t_handle</i> | The pointer to an allocated space where to store transactional layer internal state. |
| <i>cfg</i>           | The pointer to the parameters required to configure the UART after initialization.   |

### Returns

0 Succeed, others fail.

#### 29.7.3.2 int UART\_RTOS\_Deinit ( uart\_rtos\_handle\_t \* *handle* )

This function deinitializes the UART module, sets all register values to reset value, and releases the resources.

### Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | The RTOS UART handle. |
|---------------|-----------------------|

#### 29.7.3.3 int UART\_RTOS\_Send ( uart\_rtos\_handle\_t \* *handle*, const uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS UART handle.          |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 29.7.3.4 int UART\_RTOS\_Receive ( uart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from UART. It is a synchronous API. If any data is immediately available, it is returned immediately and the number of bytes received.

## Parameters

|                 |  |
|-----------------|--|
| <i>handle</i>   | The RTOS UART handle.  |
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.  |
| <i>received</i> | The pointer to variable of a size_t where the number of received data is filled. |





## Chapter 30 Debug Console

### 30.1 Overview

This part 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.

### 30.2 Function groups

#### 30.2.1 Initialization

To initialize the debug console, call the `DbgConsole_Init()` function with these parameters. This function automatically enables the module and the clock.

```
/*
 * @brief Initializes the the peripheral used to debug messages.
 *
 * @param baseAddr      Indicates which address of the peripheral is used to send debug messages.
 * @param baudRate      The desired baud rate in bits per second.
 * @param device        Low level device type for the debug console, can be one of:
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_UART,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_LPUART,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_LPSCI,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_USBCDC.
 * @param clkSrcFreq    Frequency of peripheral source clock.
 *
 * @return              Whether initialization was successful or not.
 */
status_t DbgConsole_Init(uint32_t baseAddr, uint32_t baudRate, uint8_t device, uint32_t clkSrcFreq)
```

Selects the supported debug console hardware device type, such as

```
DEBUG_CONSOLE_DEVICE_TYPE_NONE
DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
DEBUG_CONSOLE_DEVICE_TYPE_UART
DEBUG_CONSOLE_DEVICE_TYPE_LPUART
DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
```

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          type;
    void*            base;
    debug_console_ops_t ops;
} debug_console_state_t;
```

## Function groups

This example shows how to call the DbgConsole\_Init() given the user configuration structure:

```
uint32_t uartClkSrcFreq = CLOCK_GetFreq (BOARD_DEBUG_UART_CLKSRC);  
  
DbgConsole_Init (BOARD_DEBUG_UART_BASEADDR, BOARD_DEBUG_UART_BAUDRATE, DEBUG_CONSOLE_DEVICE_TYPE_UART,  
                uartClkSrcFreq);
```

### 30.2.2 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 going to be 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.  |

| <b>.precision</b> | <b>Description</b>   |
|-------------------|--|
| .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.  |

| <b>length</b>  | <b>Description</b> |
|----------------|--------------------|
| Do not support |                    |

| <b>specifier</b> | <b>Description</b>                           |
|------------------|--|
| d or i           | Signed decimal integer                       |
| f                | Decimal floating point                       |
| F                | Decimal floating point capital letters       |
| x                | Unsigned hexadecimal integer                 |
| X                | Unsigned hexadecimal integer capital letters |
| o                | Signed octal                                 |
| b                | Binary value                                 |
| p                | Pointer address                              |
| u                | Unsigned decimal integer                     |
| c                | Character                                    |
| s                | String of characters                         |
| n                | Nothing printed                              |

## Function groups

- Support a format specifier for SCANF following this prototype " %[\*][width][length]specifier", which is explained below

| *  | Description |
|--|-------------|
| An optional starting asterisk indicates that the data is to be read from the stream but ignored, i.e., it is not stored in the corresponding argument. |             |

| width  | Description |
|--|-------------|
| This specifies the maximum number of characters to be read in the current reading operation. |             |

| length      | Description  |
|-------------|--|
| hh          | The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).  |
| h           | The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).   |
| l           | The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X), and as a wide character or wide character string for specifiers c and s.           |
| ll          | The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X), and as a wide character or wide character string for specifiers c and s. |
| L           | The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).   |
| j or z or t | Not supported  |

| specifier | Qualifying Input   | Type of argument |
|-----------|--|------------------|
| c         | Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end. | char *           |

| specifier              | Qualifying Input  | Type of argument |
|------------------------|---|------------------|
| i                      | Integer: : Number optionally preceded with a + or - sign  | int *            |
| d                      | Decimal integer: Number optionally preceded with a + or - sign  | int *            |
| a, A, e, E, f, F, g, G | Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4 | float *          |
| o                      | Octal Integer:  | int *            |
| s                      | String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).  | char *           |
| u                      | Unsigned decimal integer.   | unsigned int *   |

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file:

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(const char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the KSDK printf/scanf:

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

## 30.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

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

## Typical use case

### Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalent 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 KSDK \_\_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:

If you want to use 'printf' and 'scanf' for GNUC Base, you should add file '**fsl\_sbrk.c**' in path: **..\{package}\devices\{subset}\utilities\fsl\_sbrk.c** to your project.

## Modules

- [Semihosting](#)

## 30.4 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 could 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

### 30.4.1 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 will ensure that the debug session will start by running to 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. Start the project by choosing Project>Download and Debug.
4. Choose View>Terminal I/O to display the output from the I/O operations.

### 30.4.2 Guide Semihosting for Keil $\mu$ Vision

**NOTE:** Keil supports Semihosting only for M3/M4 cores.

#### Step 1: Prepare code

Remove function `fputc` and `fgetc` is used to support KEIL in "fsl\_debug\_console.c" then add the following code to project:

```
#pragma import(__use_no_semihosting_swi)

volatile int ITM_RxBuffer = ITM_RXBUFFER_EMPTY;          /* used for Debug Input */
```

## Semihosting

```
struct __FILE
{
    int handle;
};
FILE __stdout;
FILE __stdin;

int fputc(int ch, FILE *f)
{
    return (ITM_SendChar(ch));
}

int fgetc(FILE *f)
{
    /* blocking */
    while (ITM_CheckChar() != 1)
        ;
    return (ITM_ReceiveChar());
}

int ferror(FILE *f)
{
    /* Your implementation of ferror */
    return EOF;
}

void _ttywrch(int ch)
{
    ITM_SendChar(ch);
}

void _sys_exit(int return_code)
{
label:
    goto label; /* endless loop */
}
```

### Step 2: Setting up the environment

1. In menu bar, choose Project>Options for target or using Alt+F7 or click
2. Next, select "Target" tab and not select "Use MicroLIB".
3. Next, select "Debug" tab, select "J-LINK/J-TRACE Cortex" and click "Setting button".
4. Next, select "Debug" tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click OK

### Step 3: Building the project

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

### Step 4: Building 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.



### 30.4.3 Guide Semihosting for KDS

**NOTE:** After the setting we can use "printf" for debugging

#### Step 1: Setting up the environment

1. In menu bar, choose Project>Properties>C/C++ Build>Settings>Tool Settings.
2. Select "Libraries" on "Cross ARM C Linker" and delete "nosys".
3. Select "Miscellaneous" on "Cross ARM C Linker", add "-specs=rdimon.specs" to "Other link flages" and tick "Use newlib-nano" and click OK.

#### Step 2: Building the project

1. In menu bar, choose Project>Build Project.

#### Step 3: Starting semihosting

1. In Debug configurations, choose "Startup" tab, tick "Enable semihosting and Telnet". Press "Apply" and "Debug".
2. After click Debug, the Window same as below, run line by line to see result in Console Window.

### 30.4.4 Guide Semihosting for ATL

**NOTE:** Hardware jlink have to be used to enable semihosting

#### Step 1: Prepare code

Add the following code to project:

```
int _write(int file, char *ptr, int len)
{
    /* Implement your write code here, this is used by puts and printf for example */
    int i=0;
    for(i=0 ; i<len ; i++)
        ITM_SendChar((*ptr++));
    return len;
}
```

#### Step 2: Setting up the environment

1. In menu bar, choose Debug Configurations. In tab "Embedded C/C++ Application" choose "- Semihosting\_ATL\_xxx debug jlink".
2. In tab "Debugger" setup like that:
  - JTAG mode must be selected
  - SWV tracing must be enabled

## Semihosting

- Enter the Core Clock frequency. This is H/W board specific.
  - Enter the desired SWO Clock frequency. The latter depends on the JTAG Probe and must be a multiple of the Core Clock value.
3. Click "Apply" and "Debug".

### Step 3: Starting semihosting

1. In the Views menu, expand the submenu SWV and open the docking view "SWV Console".
2. Open the SWV settings panel by clicking on the Configure Serial Wire Viewer button in the SWV Console view toolbar.
3. Configure the data ports to be traced by enabling the ITM channel 0 check-box in the ITM stimulus ports group: Choose "EXETRC: Trace Exceptions" and In tab "ITM Stimulus Ports" choose "Enable Port" 0. Then click "OK".
4. Recommend not enabling other SWV trace functionalities at the same time, as this may over-use the SWO pin causing packet loss due to limited bandwidth (certain other SWV tracing capabilities can send a lot of data at very high speed). Save the SWV configuration by clicking the OK button. The configuration is saved together with other debug configurations and will remain effective until changed.
5. Press the red Start/Stop Trace button to send the SWV configuration to the target board and enable SWV trace recoding. The board will not send any SWV packages until it is properly configured. The SWV Configuration must be resent, if the configuration registers on the target board are reset. Also, actual tracing will not start until the target starts to execute
6. Start the target execution again by pressing the green Resume Debug button.
7. The SWV console will now show the printf() output

## 30.4.5 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". Setup like this :
  - "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}
```

## Semihosting

```
G} --specs=rdimon.specs ")
```

### Remove

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

2. Run "build\_debug.bat" to build project

## Step 3: Starting semihosting

- (a) Download the image and set like this:

```
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
```

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

## Chapter 31 Notification Framework

### 31.1 Overview

This section describes the programming interface of the Notifier driver.

### 31.2 Notifier Overview

The Notifier provides a configuration dynamic change service. Based on this service, applications can switch between pre-defined configurations. The Notifier enables drivers and applications to register callback functions to this framework. Each time that the configuration is changed, drivers and applications receive a notification and change their settings. To simplify, the Notifier only supports the static callback registration. This means that, for applications, all callback functions are collected into a static table and passed to the Notifier.

The configuration transition includes 3 steps:

1. Before configuration transition, the Notifier sends a "BEFORE" message to the callback table. When this message is received, IP drivers should check whether any current processes can be stopped and stop them. If the processes cannot be stopped, the callback function returns an error.  
The Notifier supports two types of transition policies, a graceful policy and a forceful policy. When the graceful policy is used, if some callbacks return an error while sending "BEFORE" message, the configuration transition stops and the Notifier sends a "RECOVER" message to all drivers that have stopped. Then, these drivers can recover the previous status and continue to work. When the forceful policy is used, drivers are stopped forcefully.
2. After the "BEFORE" message is processed successfully, the system changes to the new configuration.
3. After the configuration changes, the Notifier sends an "AFTER" message to the callback table to notify drivers that the configuration transition is finished.

This example shows how to use the Notifier in the Power Manager application:

```
#include "fsl_notifier.h"

/* Definition of the Power Manager callback */
status_t callback0(notifier_notification_block_t *notify, void *data)
{
    status_t ret = kStatus_Success;

    ...
    ...
    ...

    return ret;
}

/* Definition of the Power Manager user function */
status_t APP_PowerModeSwitch(notifier_user_config_t *targetConfig, void *userData)
{

```

## Notifier Overview

```
...
...
...
}
...
...
...
...
...
/* Main function */
int main(void)
{
    /* Define a notifier handle */
    notifier_handle_t powerModeHandle;

    /* Callback configuration */
    user_callback_data_t callbackData0;

    notifier_callback_config_t callbackCfg0 = {callback0,
        kNOTIFIER_CallbackBeforeAfter,
        (void *)&callbackData0};

    notifier_callback_config_t callbacks[] = {callbackCfg0};

    /* Power mode configurations */
    power_user_config_t vlprConfig;
    power_user_config_t stopConfig;

    notifier_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

    /* Definition of a transition to and out the power modes */
    vlprConfig.mode = kAPP_PowerModeVlpr;
    vlprConfig.enableLowPowerWakeUpOnInterrupt = false;

    stopConfig = vlprConfig;
    stopConfig.mode = kAPP_PowerModeStop;

    /* Create Notifier handle */
    NOTIFIER_CreateHandle(&powerModeHandle, powerConfigs, 2U, callbacks, 1U,
        APP_PowerModeSwitch, NULL);
    ...
    ...
    /* Power mode switch */
    NOTIFIER_switchConfig(&powerModeHandle, targetConfigIndex,
        kNOTIFIER_PolicyAgreement);
}
```

## Data Structures

- struct `notifier_notification_block_t`  
*notification block passed to the registered callback function. [More...](#)*
- struct `notifier_callback_config_t`  
*Callback configuration structure. [More...](#)*
- struct `notifier_handle_t`  
*Notifier handle structure. [More...](#)*

## Typedefs

- typedef void `notifier_user_config_t`  
*Notifier user configuration type.*
- typedef status\_t(\* `notifier_user_function_t`)(`notifier_user_config_t` \*targetConfig, void \*userData)  
*Notifier user function prototype Use this function to execute specific operations in configuration switch.*

- typedef status\_t(\* [notifier\\_callback\\_t](#))([notifier\\_notification\\_block\\_t](#) \*notify, void \*data)  
*Callback prototype.*

## Enumerations

- enum [\\_notifier\\_status](#) {  
    [kStatus\\_NOTIFIER\\_ErrorNotificationBefore](#),  
    [kStatus\\_NOTIFIER\\_ErrorNotificationAfter](#) }  
*Notifier error codes.*
- enum [notifier\\_policy\\_t](#) {  
    [kNOTIFIER\\_PolicyAgreement](#),  
    [kNOTIFIER\\_PolicyForcible](#) }  
*Notifier policies.*
- enum [notifier\\_notification\\_type\\_t](#) {  
    [kNOTIFIER\\_NotifyRecover](#) = 0x00U,  
    [kNOTIFIER\\_NotifyBefore](#) = 0x01U,  
    [kNOTIFIER\\_NotifyAfter](#) = 0x02U }  
*Notification type.*
- enum [notifier\\_callback\\_type\\_t](#) {  
    [kNOTIFIER\\_CallbackBefore](#) = 0x01U,  
    [kNOTIFIER\\_CallbackAfter](#) = 0x02U,  
    [kNOTIFIER\\_CallbackBeforeAfter](#) = 0x03U }  
*The callback type, indicates what kinds of notification the callback handles.*

## Functions

- status\_t [NOTIFIER\\_CreateHandle](#) ([notifier\\_handle\\_t](#) \*notifierHandle, [notifier\\_user\\_config\\_t](#) \*\*configs, uint8\_t configsNumber, [notifier\\_callback\\_config\\_t](#) \*callbacks, uint8\_t callbacksNumber, [notifier\\_user\\_function\\_t](#) userFunction, void \*userData)  
*Create Notifier handle.*
- status\_t [NOTIFIER\\_SwitchConfig](#) ([notifier\\_handle\\_t](#) \*notifierHandle, uint8\_t configIndex, [notifier-\\_policy\\_t](#) policy)  
*Switch configuration according to a pre-defined structure.*
- uint8\_t [NOTIFIER\\_GetErrorCallbackIndex](#) ([notifier\\_handle\\_t](#) \*notifierHandle)  
*This function returns the last failed notification callback.*

## 31.3 Data Structure Documentation

### 31.3.1 struct [notifier\\_notification\\_block\\_t](#)

#### Data Fields

- [notifier\\_user\\_config\\_t](#) \* [targetConfig](#)  
*Pointer to target configuration.*
- [notifier\\_policy\\_t](#) [policy](#)  
*Configure transition policy.*
- [notifier\\_notification\\_type\\_t](#) [notifyType](#)  
*Configure notification type.*

### 31.3.1.0.0.49 Field Documentation

31.3.1.0.0.49.1 `notifier_user_config_t* notifier_notification_block_t::targetConfig`

31.3.1.0.0.49.2 `notifier_policy_t notifier_notification_block_t::policy`

31.3.1.0.0.49.3 `notifier_notification_type_t notifier_notification_block_t::notifyType`

### 31.3.2 struct `notifier_callback_config_t`

This structure holds configuration of callbacks. Callbacks of this type are expected to be statically allocated. This structure contains following application-defined data: `callback` - pointer to the callback function `callbackType` - specifies when the callback is called `callbackData` - pointer to the data passed to the callback.

#### Data Fields

- [notifier\\_callback\\_t callback](#)  
*Pointer to the callback function.*
- [notifier\\_callback\\_type\\_t callbackType](#)  
*Callback type.*
- `void *` [callbackData](#)  
*Pointer to the data passed to the callback.*

### 31.3.2.0.0.50 Field Documentation

31.3.2.0.0.50.1 `notifier_callback_t notifier_callback_config_t::callback`

31.3.2.0.0.50.2 `notifier_callback_type_t notifier_callback_config_t::callbackType`

31.3.2.0.0.50.3 `void* notifier_callback_config_t::callbackData`

### 31.3.3 struct `notifier_handle_t`

Notifier handle structure. Contains data necessary for Notifier proper function. Stores references to registered configurations, callbacks, information about their numbers, user function, user data and other internal data. [NOTIFIER\\_CreateHandle\(\)](#) must be called to initialize this handle.

#### Data Fields

- `notifier_user_config_t *` [configsTable](#)  
*Pointer to configure table.*
- `uint8_t` [configsNumber](#)  
*Number of configurations.*
- `notifier_callback_config_t *` [callbacksTable](#)  
*Pointer to callback table.*



- `uint8_t callbacksNumber`  
*Maximum number of callback configurations.*
- `uint8_t errorCallbackIndex`  
*Index of callback returns error.*
- `uint8_t currentConfigIndex`  
*Index of current configuration.*
- `notifier_user_function_t userFunction`  
*user function.*
- `void * userData`  
*user data passed to user function.*

### 31.3.3.0.0.51 Field Documentation

31.3.3.0.0.51.1 `notifier_user_config_t** notifier_handle_t::configsTable`

31.3.3.0.0.51.2 `uint8_t notifier_handle_t::configsNumber`

31.3.3.0.0.51.3 `notifier_callback_config_t* notifier_handle_t::callbacksTable`

31.3.3.0.0.51.4 `uint8_t notifier_handle_t::callbacksNumber`

31.3.3.0.0.51.5 `uint8_t notifier_handle_t::errorCallbackIndex`

31.3.3.0.0.51.6 `uint8_t notifier_handle_t::currentConfigIndex`

31.3.3.0.0.51.7 `notifier_user_function_t notifier_handle_t::userFunction`

31.3.3.0.0.51.8 `void* notifier_handle_t::userData`

## 31.4 Typedef Documentation

### 31.4.1 `typedef void notifier_user_config_t`

Reference of user defined configuration is stored in an array; the notifier switches between these configurations based on this array.

### 31.4.2 `typedef status_t(* notifier_user_function_t)(notifier_user_config_t *targetConfig, void *userData)`

Before and after this function execution, different notification is sent to registered callbacks. If this function returns any error code, `NOTIFIER_SwitchConfig()` exits.

Parameters

---

## Enumeration Type Documentation

|                     |  |
|---------------------|--|
| <i>targetConfig</i> | target Configuration.                                  |
| <i>userData</i>     | Refers to other specific data passed to user function. |

### Returns

An error code or `kStatus_Success`.

### 31.4.3 `typedef status_t(* notifier_callback_t)(notifier_notification_block_t *notify, void *data)`

Declaration of callback. It is common for registered callbacks. Reference to function of this type is part of `notifier_callback_config_t` callback configuration structure. Depending on callback type, function of this prototype is called (see `NOTIFIER_SwitchConfig()`) before configuration switch, after it or in both use cases to notify about the switch progress (see `notifier_callback_type_t`). When called, type of the notification is passed as parameter along with reference to the target configuration structure (see `notifier_notification_block_t`) and any data passed during the callback registration. When notified before configuration switch, depending on the configuration switch policy (see `notifier_policy_t`) the callback may deny the execution of user function by returning any error code different from `kStatus_Success` (see `NOTIFIER_SwitchConfig()`).

### Parameters

|               |  |
|---------------|--|
| <i>notify</i> | Notification block.  |
| <i>data</i>   | Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information. |

### Returns

An error code or `kStatus_Success`.

## 31.5 Enumeration Type Documentation

### 31.5.1 `enum _notifier_status`

Used as return value of Notifier functions.

### Enumerator

***kStatus\_NOTIFIER\_ErrorNotificationBefore*** Error occurs during send "BEFORE" notification.  
***kStatus\_NOTIFIER\_ErrorNotificationAfter*** Error occurs during send "AFTER" notification.

### 31.5.2 enum notifier\_policy\_t

Defines whether user function execution is forced or not. For `kNOTIFIER_PolicyForcible`, the user function is executed regardless of the callback results, while `kNOTIFIER_PolicyAgreement` policy is used to exit `NOTIFIER_SwitchConfig()` when any of the callbacks returns error code. See also `NOTIFIER_SwitchConfig()` description.

Enumerator

***kNOTIFIER\_PolicyAgreement*** `NOTIFIER_SwitchConfig()` method is exited when any of the callbacks returns error code.

***kNOTIFIER\_PolicyForcible*** user function is executed regardless of the results.

### 31.5.3 enum notifier\_notification\_type\_t

Used to notify registered callbacks

Enumerator

***kNOTIFIER\_NotifyRecover*** Notify IP to recover to previous work state.

***kNOTIFIER\_NotifyBefore*** Notify IP that configuration setting is going to change.

***kNOTIFIER\_NotifyAfter*** Notify IP that configuration setting has been changed.

### 31.5.4 enum notifier\_callback\_type\_t

Used in the callback configuration structure (`notifier_callback_config_t`) to specify when the registered callback is called during configuration switch initiated by `NOTIFIER_SwitchConfig()`. Callback can be invoked in following situations:

- before the configuration switch (Callback return value can affect `NOTIFIER_SwitchConfig()` execution. See the `NOTIFIER_SwitchConfig()` and `notifier_policy_t` documentation).
- after unsuccessful attempt to switch configuration
- after successful configuration switch

Enumerator

***kNOTIFIER\_CallbackBefore*** Callback handles BEFORE notification.

***kNOTIFIER\_CallbackAfter*** Callback handles AFTER notification.

***kNOTIFIER\_CallbackBeforeAfter*** Callback handles BEFORE and AFTER notification.

## 31.6 Function Documentation

**31.6.1** `status_t NOTIFIER_CreateHandle ( notifier_handle_t * notifierHandle,  
notifier_user_config_t ** configs, uint8_t configsNumber, notifier_callback-  
_config_t * callbacks, uint8_t callbacksNumber, notifier_user_function_t  
userFunction, void * userData )`

## Parameters

|                         |   |
|-------------------------|---|
| <i>notifierHandle</i>   | A pointer to notifier handle  |
| <i>configs</i>          | A pointer to an array with references to all configurations which is handled by the Notifier.   |
| <i>configsNumber</i>    | Number of configurations. Size of the configuration array.  |
| <i>callbacks</i>        | A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value. |
| <i>callbacks-Number</i> | Number of registered callbacks. Size of callbacks array.  |
| <i>userFunction</i>     | user function.  |
| <i>userData</i>         | user data passed to user function.  |

## Returns

An error code or kStatus\_Success.

### 31.6.2 **status\_t NOTIFIER\_SwitchConfig ( notifier\_handle\_t \* *notifierHandle*, uint8\_t *configIndex*, notifier\_policy\_t *policy* )**

This function sets the system to the target configuration. Before transition, the Notifier sends notifications to all callbacks registered to the callback table. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array. The same order is used for before and after switch notifications. The notifications before the configuration switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the configuration change, further execution of this function depends on the notifier policy: the configuration change is either forced (kNOTIFIER\_PolicyForcible) or exited (kNOTIFIER\_PolicyAgreement). When configuration change is forced, the result of the before switch notifications are ignored. If agreement is required, if any callback returns an error code then further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked. The index of the callback which returned error code during pre-switch notifications is stored (any error codes during callbacks re-invocation are ignored) and NOTIFIER\_GetErrorCallback() can be used to get it. Regardless of the policies, if any callback returned an error code, an error code denoting in which phase the error occurred is returned when [NOTIFIER\\_SwitchConfig\(\)](#) exits.

## Parameters

## Function Documentation

|                       |  |
|-----------------------|--|
| <i>notifierHandle</i> | pointer to notifier handle   |
| <i>configIndex</i>    | Index of the target configuration.   |
| <i>policy</i>         | Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible. |

### Returns

An error code or kStatus\_Success.

### 31.6.3 uint8\_t NOTIFIER\_GetErrorCallbackIndex ( notifier\_handle\_t \* *notifierHandle* )

This function returns index of the last callback that failed during the configuration switch while the last [NOTIFIER\\_SwitchConfig\(\)](#) was called. If the last [NOTIFIER\\_SwitchConfig\(\)](#) call ended successfully value equal to callbacks number is returned. Returned value represents index in the array of static call-backs.

### Parameters

|                       |                            |
|-----------------------|----------------------------|
| <i>notifierHandle</i> | pointer to notifier handle |
|-----------------------|----------------------------|

### Returns

Callback index of last failed callback or value equal to callbacks count.

## Chapter 32

### Shell

#### 32.1 Overview

This part describes the programming interface of the Shell middleware. Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

#### 32.2 Function groups

##### 32.2.1 Initialization

To initialize the Shell middleware, call the [SHELL\\_Init\(\)](#) function with these parameters. This function automatically enables the middleware.

```
void SHELL_Init(p_shell_context_t context, send_data_cb_t send_cb,
               recv_data_cb_t recv_cb, char *prompt);
```

Then, after the initialization was successful, call a command to control MCUs.

This example shows how to call the [SHELL\\_Init\(\)](#) given the user configuration structure.

```
SHELL_Init(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
```

##### 32.2.2 Advanced Feature

- Support to get a character from standard input devices.

```
static uint8_t GetChar(p_shell_context_t context);
```

| Commands   | Description                                      |
|------------|--|
| Help       | Lists all commands which are supported by Shell. |
| Exit       | Exits the Shell program.                         |
| strCompare | Compares the two input strings.                  |

| Input character | Description   |
|-----------------|---|
| A               | Gets the latest command in the history.             |
| B               | Gets the first command in the history.              |
| C               | Replaces one character at the right of the pointer. |

## Function groups

| Input character | Description  |
|-----------------|--|
| D               | Replaces one character at the left of the pointer. |
|                 | Run AutoComplete function                          |
|                 | Run cmdProcess function                            |
|                 | Clears a command.                                  |

### 32.2.3 Shell Operation

```
SHELL_Init(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");  
SHELL_Main(&user_context);
```

## Data Structures

- struct [p\\_shell\\_context\\_t](#)  
*Data structure for Shell environment. [More...](#)*
- struct [shell\\_command\\_context\\_t](#)  
*User command data structure. [More...](#)*
- struct [shell\\_command\\_context\\_list\\_t](#)  
*Structure list command. [More...](#)*

## Macros

- #define [SHELL\\_USE\\_HISTORY](#) (0U)  
*Macro to set on/off history feature.*
- #define [SHELL\\_SEARCH\\_IN\\_HIST](#) (1U)  
*Macro to set on/off history feature.*
- #define [SHELL\\_USE\\_FILE\\_STREAM](#) (0U)  
*Macro to select method stream.*
- #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\\_HIST\\_MAX](#) (3U)  
*Macro to set maximum count of history commands.*
- #define [SHELL\\_MAX\\_CMD](#) (6U)  
*Macro to set maximum count of commands.*

## Typedefs

- typedef void(\* [send\\_data\\_cb\\_t](#))(uint8\_t \*buf, uint32\_t len)  
*Shell user send data callback prototype.*
- typedef void(\* [recv\\_data\\_cb\\_t](#))(uint8\_t \*buf, uint32\_t len)  
*Shell user receiver data callback prototype.*
- typedef int(\* [printf\\_data\\_t](#))(const char \*format,...)



- *Shell user printf data prototype.*  
typedef int32\_t(\* [cmd\\_function\\_t](#))(p\_shell\_context\_t context, int32\_t argc, char \*\*argv)  
*User command function prototype.*

## Enumerations

- enum [fun\\_key\\_status\\_t](#) {  
    [kSHELL\\_Normal](#) = 0U,  
    [kSHELL\\_Special](#) = 1U,  
    [kSHELL\\_Function](#) = 2U }  
*A type for the handle special key.*

## Shell functional Operation

- void [SHELL\\_Init](#) (p\_shell\_context\_t context, [send\\_data\\_cb\\_t](#) send\_cb, [recv\\_data\\_cb\\_t](#) recv\_cb, [printf\\_data\\_t](#) shell\_printf, char \*prompt)  
*Enables the clock gate and configure the Shell module according to the configuration structure.*
- int32\_t [SHELL\\_RegisterCommand](#) (const [shell\\_command\\_context\\_t](#) \*command\_context)  
*Shell register command.*
- int32\_t [SHELL\\_Main](#) (p\_shell\_context\_t context)  
*Main loop for Shell.*

## 32.3 Data Structure Documentation

### 32.3.1 struct shell\_context\_struct

#### Data Fields

- char \* [prompt](#)  
*Prompt string.*
- enum [\\_fun\\_key\\_status](#) [stat](#)  
*Special key status.*
- char [line](#) [[SHELL\\_BUFFER\\_SIZE](#)]  
*Consult buffer.*
- uint8\_t [cmd\\_num](#)  
*Number of user commands.*
- uint8\_t [l\\_pos](#)  
*Total line position.*
- uint8\_t [c\\_pos](#)  
*Current line position.*
- [send\\_data\\_cb\\_t](#) [send\\_data\\_func](#)  
*Send data interface operation.*
- [recv\\_data\\_cb\\_t](#) [recv\\_data\\_func](#)  
*Receive data interface operation.*
- uint16\_t [hist\\_current](#)  
*Current history command in hist buff.*
- uint16\_t [hist\\_count](#)  
*Total history command in hist buff.*
- char [hist\\_buf](#) [[SHELL\\_HIST\\_MAX](#)][[SHELL\\_BUFFER\\_SIZE](#)]

## Data Structure Documentation

- History buffer.*
- bool [exit](#)  
*Exit Flag.*

### 32.3.2 struct shell\_command\_context\_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.*

#### 32.3.2.0.0.52 Field Documentation

##### 32.3.2.0.0.52.1 const char\* shell\_command\_context\_t::pcCommand

For example "help". It must be all lower case.

##### 32.3.2.0.0.52.2 char\* shell\_command\_context\_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".

##### 32.3.2.0.0.52.3 const cmd\_function\_t shell\_command\_context\_t::pFuncCallBack

##### 32.3.2.0.0.52.4 uint8\_t shell\_command\_context\_t::cExpectedNumberOfParameters

### 32.3.3 struct shell\_command\_context\_list\_t

#### Data Fields

- const [shell\\_command\\_context\\_t](#) \* [CommandList](#) [[SHELL\\_MAX\\_CMD](#)]  
*The command table list.*
- uint8\_t [numberOfCommandInList](#)  
*The total command in list.*

## 32.4 Macro Definition Documentation

32.4.1 `#define SHELL_USE_HISTORY (0U)`

32.4.2 `#define SHELL_SEARCH_IN_HIST (1U)`

32.4.3 `#define SHELL_USE_FILE_STREAM (0U)`

32.4.4 `#define SHELL_AUTO_COMPLETE (1U)`

32.4.5 `#define SHELL_BUFFER_SIZE (64U)`

32.4.6 `#define SHELL_MAX_ARGS (8U)`

32.4.7 `#define SHELL_HIST_MAX (3U)`

32.4.8 `#define SHELL_MAX_CMD (6U)`

## 32.5 Typedef Documentation

32.5.1 `typedef void(* send_data_cb_t)(uint8_t *buf, uint32_t len)`

32.5.2 `typedef void(* recv_data_cb_t)(uint8_t *buf, uint32_t len)`

32.5.3 `typedef int(* printf_data_t)(const char *format,...)`

32.5.4 `typedef int32_t(* cmd_function_t)(p_shell_context_t context, int32_t argc, char **argv)`

## 32.6 Enumeration Type Documentation

32.6.1 `enum fun_key_status_t`

Enumerator

*kSHELL\_Normal* Normal key.

*kSHELL\_Special* Special key.

*kSHELL\_Function* Function key.

### 32.7 Function Documentation

#### 32.7.1 void SHELL\_Init ( p\_shell\_context\_t *context*, send\_data\_cb\_t *send\_cb*, recv\_data\_cb\_t *recv\_cb*, printf\_data\_t *shell\_printf*, 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 middleware Shell and how to call the SHELL\_Init function by passing in these parameters: Example:

```
*  shell_context_struct user_context;
*  SHELL_Init(&user_context, SendDataFunc, ReceiveDataFunc, "SHELL>> ");
*
```

##### Parameters

|                |  |
|----------------|--|
| <i>context</i> | The pointer to the Shell environment and runtime states. |
| <i>send_cb</i> | The pointer to call back send data function.             |
| <i>recv_cb</i> | The pointer to call back receive data function.          |
| <i>prompt</i>  | The string prompt of Shell                               |

#### 32.7.2 int32\_t SHELL\_RegisterCommand ( const shell\_command\_context\_t \* *command\_context* )

##### Parameters

|                        |  |
|------------------------|--|
| <i>command_context</i> | The pointer to the command data structure. |
|------------------------|--|

##### Returns

-1 if error or 0 if success

#### 32.7.3 int32\_t SHELL\_Main ( p\_shell\_context\_t *context* )

Main loop for Shell; After this function is called, Shell begins to initialize the basic variables and starts to work.

## Parameters

|                |  |
|----------------|--|
| <i>context</i> | The pointer to the Shell environment and runtime states. |
|----------------|--|

## Returns

this function does not return until Shell command exit was called.



## Chapter 33 DMA Manager

### 33.1 Overview

DMA Manager provides a series of functions to manage the DMAMUX channels.

### 33.2 Function groups

#### 33.2.1 DMAMGR Initialization and De-initialization

This function group initializes and deinitializes the DMA Manager.

#### 33.2.2 DMAMGR Operation

This function group requests/releases the DMAMUX channel and configures the channel request source.

### 33.3 Typical use case

#### 33.3.1 DMAMGR static channel allocate

```
DMAMUX_Type *dmamux_base;
uint8_t channel;

/* Initialize DMAMGR */
DMAMGR_Init();
/* Request a DMAMUX channel by static allocate mechanism */
dmamux_base = DMAMUX0;
channel = 0;
DMAMGR_RequestChannel(kDmaRequestMux0AlwaysOn63, &dmamux_base, &channel,
    kDMAMGR_STATIC_ALLOCATE);
```

#### 33.3.2 DMAMGR dynamic channel allocate

```
DMAMUX_Type *dmamux_base;
uint8_t channel;

/* Initialize DMAMGR */
DMAMGR_Init();
/* Request a DMAMUX channel by static allocate mechanism */
dmamux_base = DMAMUX0;
channel = 0;
DMAMGR_RequestChannel(kDmaRequestMux0AlwaysOn63, &dmamux_base, &channel,
    kDMAMGR_DYNAMIC_ALLOCATE);
```

### Macros

- #define [DMAMGR\\_DYNAMIC\\_ALLOCATE](#) 0xFFU

## Function Documentation

*Dynamic channel allocate mechanism.*

## Enumerations

- enum `_dma_manager_status` {  
    [kStatus\\_DMAMGR\\_ChannelOccupied](#) = MAKE\_STATUS(kStatusGroup\_DMAMGR, 0),  
    [kStatus\\_DMAMGR\\_ChannelNotUsed](#) = MAKE\_STATUS(kStatusGroup\_DMAMGR, 1),  
    [kStatus\\_DMAMGR\\_NoFreeChannel](#) = MAKE\_STATUS(kStatusGroup\_DMAMGR, 2),  
    [kStatus\\_DMAMGR\\_ChannelNotMatchSource](#) = MAKE\_STATUS(kStatusGroup\_DMAMGR, 3)  
}

*DMA manager status.*

## DMAMGR Initialize and De-initialize

- void [DMAMGR\\_Init](#) (void)  
    *Initializes the DAM manager.*
- void [DMAMGR\\_Deinit](#) (void)  
    *Deinitializes the DMA manager.*

## DMAMGR Operation

- status\_t [DMAMGR\\_RequestChannel](#) (dma\_request\_source\_t requestSource, uint8\_t virtual-Channel, void \*handle)  
    *Requests a DMA channel.*
- status\_t [DMAMGR\\_ReleaseChannel](#) (void \*handle)  
    *Releases a DMA channel.*

## 33.4 Macro Definition Documentation

### 33.4.1 #define DMAMGR\_DYNAMIC\_ALLOCATE 0xFFU

## 33.5 Enumeration Type Documentation

### 33.5.1 enum \_dma\_manager\_status

Enumerator

*kStatus\_DMAMGR\_ChannelOccupied* Channel has been occupied.  
*kStatus\_DMAMGR\_ChannelNotUsed* Channel has not been used.  
*kStatus\_DMAMGR\_NoFreeChannel* All channel has been occupied.  
*kStatus\_DMAMGR\_ChannelNotMatchSource* Channel do not match the request source.

## 33.6 Function Documentation

### 33.6.1 void DMAMGR\_Init ( void )

This function initializes the DMA manager, ungates all DMAMUX clocks, and initializes the eDMA or DMA peripheral.



### 33.6.2 void DMAMGR\_Deinit ( void )

This function deinitializes the DMA manager, disables all DMAMUX channel, gates all DMAMUX clock, and deinitializes the eDMA or DMA peripheral.

### 33.6.3 status\_t DMAMGR\_RequestChannel ( dma\_request\_source\_t requestSource, uint8\_t virtualChannel, void \* handle )

This function request a DMA channel which is not occupied. There are two channels to allocate the mechanism dynamic and static. For the dynamic allocation mechanism (virtualChannel = DMAMGR\_DYNAMIC\_ALLOCATE), DMAMGR allocates a DMA channel according to the given request source and then configure it. For static allocation mechanism, DMAMGR configures the given channel according to the given request source and channel number.

Parameters

|                       |  |
|-----------------------|--|
| <i>requestSource</i>  | DMA channel request source number. See the soc.h.  |
| <i>virtualChannel</i> | The channel number user wants to occupy. If using the dynamic channel allocate mechanism, set the virtualChannel equal to DMAMGR_DYNAMIC_ALLOCATE. |
| <i>handle</i>         | DMA or eDMA handle pointer.  |

Return values

|   |   |
|---|---|
| <i>kStatus_Success</i>                      | In dynamic/static channel allocate mechanism, allocate DMAMUX channel successfully.     |
| <i>kStatus_DMAMGR_NoFreeChannel</i>         | In dynamic channel allocate mechanism, all DMAMUX channels has been occupied.           |
| <i>kStatus_DMAMGR_ChannelNotMatchSource</i> | In static channel allocate mechanism, the given channel do not match the given request. |
| <i>kStatus_DMAMGR_ChannelOccupied</i>       | In static channel allocate mechanism, the given channel has been occupied.              |

### 33.6.4 status\_t DMAMGR\_ReleaseChannel ( void \* handle )

This function releases an occupied DMA channel.

## Function Documentation

### Parameters

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

### Return values

|                                      |   |
|--------------------------------------|---|
| <i>kStatus_Success</i>               | Release the given channel successfully.                         |
| <i>kStatus_DMAMGR_ChannelNotUsed</i> | The given channel which to be released is not been used before. |

## Chapter 34

# Secured Digital Card/Embedded MultiMedia Card (CARD)

### 34.1 Overview

The Kinetis SDK provides a driver to access the Secured Digital Card and Embedded MultiMedia Card based on the SDHC driver.

### Function groups

This function group implements the SD card functional API.

This function group implements the MMC card functional API.

### Typical use case

```
/* Initialize SDHC. */
sdhcConfig->cardDetectDat3 = false;
sdhcConfig->endianMode = kSDHC_EndianModeLittle;
sdhcConfig->dmaMode = kSDHC_DmaModeAdma2;
sdhcConfig->readWatermarkLevel = 0x80U;
sdhcConfig->writeWatermarkLevel = 0x80U;
SDHC_Init(BOARD_SDHC_BASEADDR, sdhcConfig);

/* Save host information. */
card->host.base = BOARD_SDHC_BASEADDR;
card->host.sourceClock_Hz = CLOCK_GetFreq(BOARD_SDHC_CLKSRC);
card->host.transfer = SDHC_TransferFunction;

/* Init card. */
if (SD_Init(card))
{
    PRINTF("\r\nSD card init failed.\r\n");
}

while (true)
{
    if (kStatus_Success != SD_WriteBlocks(card, g_dataWrite, DATA_BLOCK_START,
        DATA_BLOCK_COUNT))
    {
        PRINTF("Write multiple data blocks failed.\r\n");
    }
    if (kStatus_Success != SD_ReadBlocks(card, g_dataRead, DATA_BLOCK_START, DATA_BLOCK_COUNT)
    )
    {
        PRINTF("Read multiple data blocks failed.\r\n");
    }

    if (kStatus_Success != SD_EraseBlocks(card, DATA_BLOCK_START, DATA_BLOCK_COUNT))
    {
        PRINTF("Erase multiple data blocks failed.\r\n");
    }
}

SD_Deinit(card);

/* Initialize SDHC. */
```

## Overview

```
sdhcConfig->cardDetectDat3 = false;
sdhcConfig->endianMode = kSDHC_EndianModeLittle;
sdhcConfig->dmaMode = kSDHC_DmaModeAdma2;
sdhcConfig->readWatermarkLevel = 0x80U;
sdhcConfig->writeWatermarkLevel = 0x80U;
SDHC_Init(BOARD_SDHC_BASEADDR, sdhcConfig);

/* Save host information. */
card->host.base = BOARD_SDHC_BASEADDR;
card->host.sourceClock_Hz = CLOCK_GetFreq(BOARD_SDHC_CLKSRC);
card->host.transfer = SDHC_TransferFunction;

/* Init card. */
if (MMC_Init(card))
{
    PRINTF("\n MMC card init failed \n");
}

while (true)
{
    if (kStatus_Success != MMC_WriteBlocks(card, g_dataWrite, DATA_BLOCK_START,
        DATA_BLOCK_COUNT))
    {
        PRINTF("Write multiple data blocks failed.\r\n");
    }
    if (kStatus_Success != MMC_ReadBlocks(card, g_dataRead, DATA_BLOCK_START,
        DATA_BLOCK_COUNT))
    {
        PRINTF("Read multiple data blocks failed.\r\n");
    }
}

MMC_Deinit(card);
```

## Data Structures

- struct [sd\\_card\\_t](#)  
*SD card state. [More...](#)*
- struct [mmc\\_card\\_t](#)  
*SD card state. [More...](#)*
- struct [mmc\\_boot\\_config\\_t](#)  
*MMC card boot configuration definition. [More...](#)*

## Macros

- #define [FSL\\_SDMMC\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2U, 1U, 1U)) /\*2.1.1\*/  
*Driver version.*
- #define [FSL\\_SDMMC\\_DEFAULT\\_BLOCK\\_SIZE](#) (512U)  
*Default block size.*

## Enumerations

- enum `_sdmmc_status` {  
`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_SwitchHighSpeedFailed` = 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) }

*SD/MMC card API's running status.*

- enum `_sd_card_flag` {  
`kSD_SupportHighCapacityFlag` = (1U << 1U),  
`kSD_Support4BitWidthFlag` = (1U << 2U),  
`kSD_SupportSdhcFlag` = (1U << 3U),  
`kSD_SupportSdxcFlag` = (1U << 4U) }

*SD card flags.*

- enum `_mmc_card_flag` {  
`kMMC_SupportHighCapacityFlag` = (1U << 0U),  
`kMMC_SupportHighSpeedFlag` = (1U << 1U),  
`kMMC_SupportHighSpeed52MHZFlag` = (1U << 2U),  
`kMMC_SupportHighSpeed26MHZFlag` = (1U << 3U),  
`kMMC_SupportAlternateBootFlag` = (1U << 4U) }

*MMC card flags.*

### SDCARD Function

- status\_t [SD\\_Init](#) (sd\_card\_t \*card)  
*Initialize the card on a specific host controller.*
- void [SD\\_Deinit](#) (sd\_card\_t \*card)  
*Deinitialize the card.*
- bool [SD\\_CheckReadOnly](#) (sd\_card\_t \*card)  
*Check whether the card is write-protected.*
- status\_t [SD\\_ReadBlocks](#) (sd\_card\_t \*card, uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)  
*Read blocks from the specific card.*
- status\_t [SD\\_WriteBlocks](#) (sd\_card\_t \*card, const uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)  
*Write blocks of data to the specific card.*
- status\_t [SD\\_EraseBlocks](#) (sd\_card\_t \*card, uint32\_t startBlock, uint32\_t blockCount)  
*Erase blocks of the specific card.*

### MMCCARD Function

- status\_t [MMC\\_Init](#) (mmc\_card\_t \*card)  
*Initialize the MMC card.*
- void [MMC\\_Deinit](#) (mmc\_card\_t \*card)  
*Deinitialize the card.*
- bool [MMC\\_CheckReadOnly](#) (mmc\_card\_t \*card)  
*Check if the card is read only.*
- status\_t [MMC\\_ReadBlocks](#) (mmc\_card\_t \*card, uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)  
*Read data blocks from the card.*
- status\_t [MMC\\_WriteBlocks](#) (mmc\_card\_t \*card, const uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)  
*Write data blocks to the card.*
- status\_t [MMC\\_EraseGroups](#) (mmc\_card\_t \*card, uint32\_t startGroup, uint32\_t endGroup)  
*Erase groups of the card.*
- status\_t [MMC\\_SelectPartition](#) (mmc\_card\_t \*card, mmc\_access\_partition\_t partitionNumber)  
*Select the partition to access.*
- status\_t [MMC\\_SetBootConfig](#) (mmc\_card\_t \*card, const mmc\_boot\_config\_t \*config)  
*Configure boot activity of the card.*

## 34.2 Data Structure Documentation

### 34.2.1 struct sd\_card\_t

Define the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- sdhc\_host\_t [host](#)  
*Host information.*
- 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.*
- uint32\_t [rawCid](#) [4U]  
*Raw CID content.*
- uint32\_t [rawCsd](#) [4U]  
*Raw CSD content.*
- uint32\_t [rawScr](#) [2U]  
*Raw CSD content.*
- uint32\_t [ocr](#)  
*Raw OCR content.*
- sd\_cid\_t [cid](#)  
*CID.*
- sd\_csd\_t [csd](#)  
*CSD.*
- sd\_scr\_t [scr](#)  
*SCR.*
- uint32\_t [blockCount](#)  
*Card total block number.*
- uint32\_t [blockSize](#)  
*Card block size.*

### 34.2.2 struct mmc\_card\_t

Define the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- sdhc\_host\_t [host](#)  
*Host information.*
- 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.*
- uint32\_t [rawCid](#) [4U]  
*Raw CID content.*
- uint32\_t [rawCsd](#) [4U]  
*Raw CSD content.*
- uint32\_t [rawExtendedCsd](#) [MMC\_EXTENDED\_CSD\_BYTES/4U]  
*Raw MMC Extended CSD content.*

## Enumeration Type Documentation

- 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 [hostVoltageWindow](#)  
*Host voltage window.*

### 34.2.3 struct mmc\_boot\_config\_t

#### Data Fields

- bool [enableBootAck](#)  
*Enable boot ACK.*
- mmc\_boot\_partition\_enable\_t [bootPartition](#)  
*Boot partition.*
- bool [retainBootBusWidth](#)  
*If retain boot bus width.*
- mmc\_data\_bus\_width\_t [bootDataBusWidth](#)  
*Boot data bus width.*

## 34.3 Macro Definition Documentation

**34.3.1 #define FSL\_SDMMC\_DRIVER\_VERSION (MAKE\_VERSION(2U, 1U, 1U))**  
*/\*2.1.1\*/*

## 34.4 Enumeration Type Documentation

### 34.4.1 enum \_sdmmc\_status

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\_SwitchHighSpeedFailed*** 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.

### 34.4.2 enum\_sd\_card\_flag

Enumerator

***kSD\_SupportHighCapacityFlag*** Support high capacity.  
***kSD\_Support4BitWidthFlag*** Support 4-bit data width.  
***kSD\_SupportSdhcFlag*** Card is SDHC.  
***kSD\_SupportSdxcFlag*** Card is SDXC.

### 34.4.3 enum\_mmc\_card\_flag

Enumerator

***kMMC\_SupportHighCapacityFlag*** Support high capacity.  
***kMMC\_SupportHighSpeedFlag*** Support high speed.  
***kMMC\_SupportHighSpeed52MHZFlag*** Support high speed 52MHZ.  
***kMMC\_SupportHighSpeed26MHZFlag*** Support high speed 26MHZ.  
***kMMC\_SupportAlternateBootFlag*** Support alternate boot.

### 34.5 Function Documentation

#### 34.5.1 `status_t SD_Init ( sd_card_t * card )`

This function initializes the card on a specific host controller.

## Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

## Return values

|   |                                  |
|---|----------------------------------|
| <i>kStatus_SDMMC_GoIdleFailed</i>                 | Go idle failed.                  |
| <i>kStatus_SDMMC_NotSupportYet</i>                | Card not support.                |
| <i>kStatus_SDMMC_SendOperationConditionFailed</i> | Send operation condition failed. |
| <i>kStatus_SDMMC_AllSendCidFailed</i>             | Send CID failed.                 |
| <i>kStatus_SDMMC_SendRelativeAddressFailed</i>    | Send relative address failed.    |
| <i>kStatus_SDMMC_SendCsdFailed</i>                | Send CSD failed.                 |
| <i>kStatus_SDMMC_SelectCardFailed</i>             | Send SELECT_CARD command failed. |
| <i>kStatus_SDMMC_SendScrFailed</i>                | Send SCR failed.                 |
| <i>kStatus_SDMMC_SetBusWidthFailed</i>            | Set bus width failed.            |
| <i>kStatus_SDMMC_SwitchHighSpeedFailed</i>        | Switch high speed failed.        |
| <i>kStatus_SDMMC_SetCardBlockSizeFailed</i>       | Set card block size failed.      |
| <i>kStatus_Success</i>                            | Operate successfully.            |

### 34.5.2 void SD\_Deinit ( sd\_card\_t \* *card* )

This function deinitializes the specific card.

## Function Documentation

### Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

### 34.5.3 bool SD\_CheckReadOnly ( sd\_card\_t \* *card* )

This function checks if the card is write-protected via CSD register.

### Parameters

|             |                    |
|-------------|--------------------|
| <i>card</i> | The specific card. |
|-------------|--------------------|

### Return values

|              |                       |
|--------------|-----------------------|
| <i>true</i>  | Card is read only.    |
| <i>false</i> | Card isn't read only. |

### 34.5.4 status\_t SD\_ReadBlocks ( sd\_card\_t \* *card*, uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )

This function reads blocks from specific card, with default block size defined by SDHC\_CARD\_DEFAULT\_BLOCK\_SIZE.

### Parameters

|                   |   |
|-------------------|---|
| <i>card</i>       | Card descriptor.                            |
| <i>buffer</i>     | The buffer to save the data read from card. |
| <i>startBlock</i> | The start block index.                      |
| <i>blockCount</i> | The number of blocks to read.               |

### Return values

|                                      |                   |
|--------------------------------------|-------------------|
| <i>kStatus_InvalidArgument</i>       | Invalid argument. |
| <i>kStatus_SDMMC_Card-NotSupport</i> | Card not support. |

|   |                           |
|---|---------------------------|
| <i>kStatus_SDMMC_Not-SupportYet</i>           | Not support now.          |
| <i>kStatus_SDMMC_Wait-WriteCompleteFailed</i> | Send status failed.       |
| <i>kStatus_SDMMC_-TransferFailed</i>          | Transfer failed.          |
| <i>kStatus_SDMMC_Stop-TransmissionFailed</i>  | Stop transmission failed. |
| <i>kStatus_Success</i>                        | Operate successfully.     |

### 34.5.5 **status\_t SD\_WriteBlocks ( sd\_card\_t \* *card*, const uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )**

This function writes blocks to specific card, with default block size 512 bytes.

Parameters

|                   |  |
|-------------------|--|
| <i>card</i>       | Card descriptor.                                       |
| <i>buffer</i>     | The buffer holding the data to be written to the card. |
| <i>startBlock</i> | The start block index.                                 |
| <i>blockCount</i> | The number of blocks to write.                         |

Return values

|   |                     |
|---|---------------------|
| <i>kStatus_InvalidArgument</i>                | Invalid argument.   |
| <i>kStatus_SDMMC_Not-SupportYet</i>           | Not support now.    |
| <i>kStatus_SDMMC_Card-NotSupport</i>          | Card not support.   |
| <i>kStatus_SDMMC_Wait-WriteCompleteFailed</i> | Send status failed. |
| <i>kStatus_SDMMC_-TransferFailed</i>          | Transfer failed.    |

## Function Documentation

|  |                           |
|--|---------------------------|
| <i>kStatus_SDMMC_Stop-TransmissionFailed</i> | Stop transmission failed. |
| <i>kStatus_Success</i>                       | Operate successfully.     |

### 34.5.6 status\_t SD\_EraseBlocks ( sd\_card\_t \* *card*, uint32\_t *startBlock*, uint32\_t *blockCount* )

This function erases blocks of a specific card, with default block size 512 bytes.

Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>card</i>       | Card descriptor.               |
| <i>startBlock</i> | The start block index.         |
| <i>blockCount</i> | The number of blocks to erase. |

Return values

|   |                       |
|---|-----------------------|
| <i>kStatus_InvalidArgument</i>                | Invalid argument.     |
| <i>kStatus_SDMMC_Wait-WriteCompleteFailed</i> | Send status failed.   |
| <i>kStatus_SDMMC_-TransferFailed</i>          | Transfer failed.      |
| <i>kStatus_SDMMC_Wait-WriteCompleteFailed</i> | Send status failed.   |
| <i>kStatus_Success</i>                        | Operate successfully. |

### 34.5.7 status\_t MMC\_Init ( mmc\_card\_t \* *card* )

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

Return values

|   |                                  |
|---|----------------------------------|
| <i>kStatus_SDMMC_Go-IdleFailed</i>                  | Go idle failed.                  |
| <i>kStatus_SDMMC_Send-OperationCondition-Failed</i> | Send operation condition failed. |
| <i>kStatus_SDMMC_All-SendCidFailed</i>              | Send CID failed.                 |
| <i>kStatus_SDMMC_Set-RelativeAddressFailed</i>      | Set relative address failed.     |
| <i>kStatus_SDMMC_Send-CsdFailed</i>                 | Send CSD failed.                 |
| <i>kStatus_SDMMC_Card-NotSupport</i>                | Card not support.                |
| <i>kStatus_SDMMC_Select-CardFailed</i>              | Send SELECT_CARD command failed. |
| <i>kStatus_SDMMC_Send-ExtendedCsdFailed</i>         | Send EXT_CSD failed.             |
| <i>kStatus_SDMMC_SetBus-WidthFailed</i>             | Set bus width failed.            |
| <i>kStatus_SDMMC_Switch-HighSpeedFailed</i>         | Switch high speed failed.        |
| <i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>        | Set card block size failed.      |
| <i>kStatus_Success</i>                              | Operate successfully.            |

### 34.5.8 void MMC\_Deinit ( mmc\_card\_t \* *card* )

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

### 34.5.9 bool MMC\_CheckReadOnly ( mmc\_card\_t \* *card* )

## Function Documentation

### Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

### Return values

|              |                       |
|--------------|-----------------------|
| <i>true</i>  | Card is read only.    |
| <i>false</i> | Card isn't read only. |

### 34.5.10 **status\_t MMC\_ReadBlocks ( mmc\_card\_t \* *card*, uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )**

### Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>card</i>       | Card descriptor.              |
| <i>buffer</i>     | The buffer to save data.      |
| <i>startBlock</i> | The start block index.        |
| <i>blockCount</i> | The number of blocks to read. |

### Return values

|  |                           |
|--|---------------------------|
| <i>kStatus_InvalidArgument</i>               | Invalid argument.         |
| <i>kStatus_SDMMC_Card-NotSupport</i>         | Card not support.         |
| <i>kStatus_SDMMC_Set-BlockCountFailed</i>    | Set block count failed.   |
| <i>kStatus_SDMMC_-TransferFailed</i>         | Transfer failed.          |
| <i>kStatus_SDMMC_Stop-TransmissionFailed</i> | Stop transmission failed. |
| <i>kStatus_Success</i>                       | Operate successfully.     |

### 34.5.11 **status\_t MMC\_WriteBlocks ( mmc\_card\_t \* *card*, const uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )**



## Parameters

|                   |                                 |
|-------------------|---------------------------------|
| <i>card</i>       | Card descriptor.                |
| <i>buffer</i>     | The buffer to save data blocks. |
| <i>startBlock</i> | Start block number to write.    |
| <i>blockCount</i> | Block count.                    |

## Return values

|   |                           |
|---|---------------------------|
| <i>kStatus_InvalidArgument</i>                | Invalid argument.         |
| <i>kStatus_SDMMC_Not-SupportYet</i>           | Not support now.          |
| <i>kStatus_SDMMC_Set-BlockCountFailed</i>     | Set block count failed.   |
| <i>kStatus_SDMMC_Wait-WriteCompleteFailed</i> | Send status failed.       |
| <i>kStatus_SDMMC_-TransferFailed</i>          | Transfer failed.          |
| <i>kStatus_SDMMC_Stop-TransmissionFailed</i>  | Stop transmission failed. |
| <i>kStatus_Success</i>                        | Operate successfully.     |

### 34.5.12 **status\_t MMC\_EraseGroups ( mmc\_card\_t \* *card*, uint32\_t *startGroup*, uint32\_t *endGroup* )**

Erase group is the smallest erase unit in MMC card. The erase range is [startGroup, endGroup].

## Parameters

|                   |                     |
|-------------------|---------------------|
| <i>card</i>       | Card descriptor.    |
| <i>startGroup</i> | Start group number. |
| <i>endGroup</i>   | End group number.   |

## Return values

## Function Documentation

|   |                       |
|---|-----------------------|
| <i>kStatus_InvalidArgument</i>                | Invalid argument.     |
| <i>kStatus_SDMMC_Wait-WriteCompleteFailed</i> | Send status failed.   |
| <i>kStatus_SDMMC_-TransferFailed</i>          | Transfer failed.      |
| <i>kStatus_Success</i>                        | Operate successfully. |

### 34.5.13 **status\_t MMC\_SelectPartition ( mmc\_card\_t \* *card*, mmc\_access\_partition\_t *partitionNumber* )**

#### Parameters

|                         |                       |
|-------------------------|-----------------------|
| <i>card</i>             | Card descriptor.      |
| <i>partition-Number</i> | The partition number. |

#### Return values

|   |                           |
|---|---------------------------|
| <i>kStatus_SDMMC_-ConfigureExtendedCsd-Failed</i> | Configure EXT_CSD failed. |
| <i>kStatus_Success</i>                            | Operate successfully.     |

### 34.5.14 **status\_t MMC\_SetBootConfig ( mmc\_card\_t \* *card*, const mmc\_boot\_config\_t \* *config* )**

#### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>card</i>   | Card descriptor.              |
| <i>config</i> | Boot configuration structure. |

#### Return values

---

|   |                           |
|---|---------------------------|
| <i>kStatus_SDMMC_Not-SupportYet</i>               | Not support now.          |
| <i>kStatus_SDMMC_-ConfigureExtendedCsd-Failed</i> | Configure EXT_CSD failed. |
| <i>kStatus_SDMMC_-ConfigureBootFailed</i>         | Configure boot failed.    |
| <i>kStatus_Success</i>                            | Operate successfully.     |



## Chapter 35

# SPI based Secured Digital Card (SDSPI)

### 35.1 Overview

The KSDK provides a driver to access the Secured Digital Card based on the SPI driver.

### Function groups

This function group implements the SD card functional API in the SPI mode.

### Typical use case

```
/* SPI_Init(). */

/* Register the SDSPI driver callback. */

/* Initializes card. */
if (kStatus_Success != SDSPI_Init(card))
{
    SDSPI_Deinit(card)
    return;
}

/* Read/Write card */
memset(g_testWriteBuffer, 0x17U, sizeof(g_testWriteBuffer));

while (true)
{
    memset(g_testReadBuffer, 0U, sizeof(g_testReadBuffer));

    SDSPI_WriteBlocks(card, g_testWriteBuffer, TEST_START_BLOCK, TEST_BLOCK_COUNT);

    SDSPI_ReadBlocks(card, g_testReadBuffer, TEST_START_BLOCK, TEST_BLOCK_COUNT);

    if (memcmp(g_testReadBuffer, g_testReadBuffer, sizeof(g_testWriteBuffer)))
    {
        break;
    }
}
```

### Data Structures

- struct [sdspi\\_command\\_t](#)  
SDSPI command. [More...](#)
- struct [sdspi\\_host\\_t](#)  
SDSPI host state. [More...](#)
- struct [sdspi\\_card\\_t](#)  
SD Card Structure. [More...](#)

### Enumerations

- enum `_sdspi_status` {  
`kStatus_SDSPI_SetFrequencyFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 0U),  
`kStatus_SDSPI_ExchangeFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 1U),  
`kStatus_SDSPI_WaitReadyFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 2U),  
`kStatus_SDSPI_ResponseError` = MAKE\_STATUS(kStatusGroup\_SDSPI, 3U),  
`kStatus_SDSPI_WriteProtected` = MAKE\_STATUS(kStatusGroup\_SDSPI, 4U),  
`kStatus_SDSPI_GoIdleFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 5U),  
`kStatus_SDSPI_SendCommandFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 6U),  
`kStatus_SDSPI_ReadFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 7U),  
`kStatus_SDSPI_WriteFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 8U),  
`kStatus_SDSPI_SendInterfaceConditionFailed`,  
`kStatus_SDSPI_SendOperationConditionFailed`,  
`kStatus_SDSPI_ReadOcrFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 11U),  
`kStatus_SDSPI_SetBlockSizeFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 12U),  
`kStatus_SDSPI_SendCsdFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 13U),  
`kStatus_SDSPI_SendCidFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 14U),  
`kStatus_SDSPI_StopTransmissionFailed` = MAKE\_STATUS(kStatusGroup\_SDSPI, 15U),  
`kStatus_SDSPI_SendApplicationCommandFailed` }  
*SDSPI API status.*
- enum `_sdspi_card_flag` {  
`kSDSPI_SupportHighCapacityFlag` = (1U << 0U),  
`kSDSPI_SupportSdhcFlag` = (1U << 1U),  
`kSDSPI_SupportSdxcFlag` = (1U << 2U),  
`kSDSPI_SupportSdscFlag` = (1U << 3U) }  
*SDSPI card flag.*
- enum `sdspi_response_type_t` {  
`kSDSPI_ResponseTypeR1` = 0U,  
`kSDSPI_ResponseTypeR1b` = 1U,  
`kSDSPI_ResponseTypeR2` = 2U,  
`kSDSPI_ResponseTypeR3` = 3U,  
`kSDSPI_ResponseTypeR7` = 4U }  
*SDSPI response type.*

### SDSPI Function

- status\_t `SDSPI_Init` (`sdspi_card_t` \*card)  
*Initialize the card on a specific SPI instance.*
- void `SDSPI_Deinit` (`sdspi_card_t` \*card)  
*Deinitialize the card.*
- bool `SDSPI_CheckReadOnly` (`sdspi_card_t` \*card)  
*Check whether the card is write-protected.*
- status\_t `SDSPI_ReadBlocks` (`sdspi_card_t` \*card, uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)  
*Read blocks from the specific card.*
- status\_t `SDSPI_WriteBlocks` (`sdspi_card_t` \*card, uint8\_t \*buffer, uint32\_t startBlock, uint32\_t blockCount)

*Write blocks of data to the specific card.*

## 35.2 Data Structure Documentation

### 35.2.1 struct sdsapi\_command\_t

#### Data Fields

- uint8\_t [index](#)  
*Command index.*
- uint32\_t [argument](#)  
*Command argument.*
- uint8\_t [responseType](#)  
*Response type.*
- uint8\_t [response](#) [5U]  
*Response content.*

### 35.2.2 struct sdsapi\_host\_t

#### Data Fields

- uint32\_t [busBaudRate](#)  
*Bus baud rate.*
- status\_t(\* [setFrequency](#) )(uint32\_t frequency)  
*Set frequency of SPI.*
- status\_t(\* [exchange](#) )(uint8\_t \*in, uint8\_t \*out, uint32\_t size)  
*Exchange data over SPI.*
- uint32\_t(\* [getCurrentMilliseconds](#) )(void)  
*Get current time in milliseconds.*

### 35.2.3 struct sdsapi\_card\_t

Define the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- [sdsapi\\_host\\_t](#) \* [host](#)  
*Host state information.*
- uint32\_t [relativeAddress](#)  
*Relative address of the card.*
- uint32\_t [flags](#)  
*Flags defined in \_sdsapi\_card\_flag.*
- uint8\_t [rawCid](#) [16U]  
*Raw CID content.*
- uint8\_t [rawCsd](#) [16U]

## Enumeration Type Documentation

- *Raw CSD content.*  
uint8\_t [rawScr](#) [8U]
- *Raw SCR content.*  
uint32\_t [ocr](#)
- *Raw OCR content.*  
sd\_cid\_t [cid](#)
- *CID.*  
sd\_csd\_t [csd](#)
- *CSD.*  
sd\_scr\_t [scr](#)
- *SCR.*  
uint32\_t [blockCount](#)
- *Card total block number.*  
uint32\_t [blockSize](#)
- *Card block size.*

### 35.2.3.0.0.53 Field Documentation

#### 35.2.3.0.0.53.1 uint32\_t sdspi\_card\_t::flags

## 35.3 Enumeration Type Documentation

### 35.3.1 enum \_sdspi\_status

Enumerator

*kStatus\_SDSPI\_SetFrequencyFailed* Set frequency failed.  
*kStatus\_SDSPI\_ExchangeFailed* Exchange data on SPI bus failed.  
*kStatus\_SDSPI\_WaitReadyFailed* Wait card ready failed.  
*kStatus\_SDSPI\_ResponseError* Response is error.  
*kStatus\_SDSPI\_WriteProtected* Write protected.  
*kStatus\_SDSPI\_GoIdleFailed* Go idle failed.  
*kStatus\_SDSPI\_SendCommandFailed* Send command failed.  
*kStatus\_SDSPI\_ReadFailed* Read data failed.  
*kStatus\_SDSPI\_WriteFailed* Write data failed.  
*kStatus\_SDSPI\_SendInterfaceConditionFailed* Send interface condition failed.  
*kStatus\_SDSPI\_SendOperationConditionFailed* Send operation condition failed.  
*kStatus\_SDSPI\_ReadOcrFailed* Read OCR failed.  
*kStatus\_SDSPI\_SetBlockSizeFailed* Set block size failed.  
*kStatus\_SDSPI\_SendCsdFailed* Send CSD failed.  
*kStatus\_SDSPI\_SendCidFailed* Send CID failed.  
*kStatus\_SDSPI\_StopTransmissionFailed* Stop transmission failed.  
*kStatus\_SDSPI\_SendApplicationCommandFailed* Send application command failed.



### 35.3.2 enum \_sdspi\_card\_flag

Enumerator

***kSDSPI\_SupportHighCapacityFlag*** Card is high capacity.

***kSDSPI\_SupportSdhcFlag*** Card is SDHC.

***kSDSPI\_SupportSdxcFlag*** Card is SDXC.

***kSDSPI\_SupportSdscFlag*** Card is SDSC.

### 35.3.3 enum sdspi\_response\_type\_t

Enumerator

***kSDSPI\_ResponseTypeR1*** Response 1.

***kSDSPI\_ResponseTypeR1b*** Response 1 with busy.

***kSDSPI\_ResponseTypeR2*** Response 2.

***kSDSPI\_ResponseTypeR3*** Response 3.

***kSDSPI\_ResponseTypeR7*** Response 7.

## 35.4 Function Documentation

### 35.4.1 status\_t SDSPI\_Init ( sdspi\_card\_t \* *card* )

This function initializes the card on a specific SPI instance.

Parameters

|             |                 |
|-------------|-----------------|
| <i>card</i> | Card descriptor |
|-------------|-----------------|

Return values

|  |                                  |
|--|----------------------------------|
| <i>kStatus_SDSPI_Set-FrequencyFailed</i>           | Set frequency failed.            |
| <i>kStatus_SDSPI_GoIdle-Failed</i>                 | Go idle failed.                  |
| <i>kStatus_SDSPI_Send-InterfaceConditionFailed</i> | Send interface condition failed. |

## Function Documentation

|   |                                  |
|---|----------------------------------|
| <i>kStatus_SDSPI_Send-OperationCondition-Failed</i> | Send operation condition failed. |
| <i>kStatus_Timeout</i>                              | Send command timeout.            |
| <i>kStatus_SDSPI_Not-SupportYet</i>                 | Not support yet.                 |
| <i>kStatus_SDSPI_ReadOcr-Failed</i>                 | Read OCR failed.                 |
| <i>kStatus_SDSPI_SetBlock-SizeFailed</i>            | Set block size failed.           |
| <i>kStatus_SDSPI_SendCsd-Failed</i>                 | Send CSD failed.                 |
| <i>kStatus_SDSPI_SendCid-Failed</i>                 | Send CID failed.                 |
| <i>kStatus_Success</i>                              | Operate successfully.            |

### 35.4.2 void SDSPI\_Deinit ( sdspi\_card\_t \* *card* )

This function deinitializes the specific card.

Parameters

|             |                 |
|-------------|-----------------|
| <i>card</i> | Card descriptor |
|-------------|-----------------|

### 35.4.3 bool SDSPI\_CheckReadOnly ( sdspi\_card\_t \* *card* )

This function checks if the card is write-protected via CSD register.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

Return values

---

|              |                       |
|--------------|-----------------------|
| <i>true</i>  | Card is read only.    |
| <i>false</i> | Card isn't read only. |

#### 35.4.4 **status\_t SDSPI\_ReadBlocks ( sdspi\_card\_t \* *card*, uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )**

This function reads blocks from specific card.

Parameters

|                   |  |
|-------------------|--|
| <i>card</i>       | Card descriptor.                           |
| <i>buffer</i>     | the buffer to hold the data read from card |
| <i>startBlock</i> | the start block index                      |
| <i>blockCount</i> | the number of blocks to read               |

Return values

|  |                           |
|--|---------------------------|
| <i>kStatus_SDSPI_Send-CommandFailed</i>      | Send command failed.      |
| <i>kStatus_SDSPI_Read-Failed</i>             | Read data failed.         |
| <i>kStatus_SDSPI_Stop-TransmissionFailed</i> | Stop transmission failed. |
| <i>kStatus_Success</i>                       | Operate successfully.     |

#### 35.4.5 **status\_t SDSPI\_WriteBlocks ( sdspi\_card\_t \* *card*, uint8\_t \* *buffer*, uint32\_t *startBlock*, uint32\_t *blockCount* )**

This function writes blocks to specific card

Parameters

|               |   |
|---------------|---|
| <i>card</i>   | Card descriptor.                                      |
| <i>buffer</i> | the buffer holding the data to be written to the card |

## Function Documentation

|                   |                               |
|-------------------|-------------------------------|
| <i>startBlock</i> | the start block index         |
| <i>blockCount</i> | the number of blocks to write |

### Return values

|   |                                      |
|---|--------------------------------------|
| <i>kStatus_SDSPI_Write-Protected</i>    | Card is write protected.             |
| <i>kStatus_SDSPI_Send-CommandFailed</i> | Send command failed.                 |
| <i>kStatus_SDSPI-ResponseError</i>      | Response is error.                   |
| <i>kStatus_SDSPI_Write-Failed</i>       | Write data failed.                   |
| <i>kStatus_SDSPI-ExchangeFailed</i>     | Exchange data over SPI failed.       |
| <i>kStatus_SDSPI_Wait-ReadyFailed</i>   | Wait card to be ready status failed. |
| <i>kStatus_Success</i>                  | Operate successfully.                |

**How to Reach Us:****Home Page:**[nxp.com](http://nxp.com)**Web Support:**[nxp.com/support](http://nxp.com/support)

Information in this document is provided solely to enable system and software implementers to use Freescale 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.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 Freescale 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. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address:

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

Freescale, the Freescale logo, Kinetis, Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Tower is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM, ARM Powered logo,  $\mu$ Vision, Keil and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016 Freescale Semiconductor, Inc.

Document Number: KSDKL4K20APIRM

Rev. 0

Jun 2016

