

---

Document Number: MCUXSDKAPIRM  
Rev 2.15.000  
Jan 2024

# MCUXpresso SDK API Reference Manual

**NXP Semiconductors**



# Contents

## Chapter 1 Introduction

## Chapter 2 Trademarks

## Chapter 3 Architectural Overview

## Chapter 4 Clock Driver

|            |   |           |
|------------|---|-----------|
| <b>4.1</b> | <b>Overview</b> .....                       | <b>7</b>  |
| <b>4.2</b> | <b>Function description</b> .....           | <b>7</b>  |
| 4.2.1      | SYSCON Clock frequency functions .....      | 7         |
| 4.2.2      | SYSCON clock Selection Muxes .....          | 7         |
| 4.2.3      | SYSCON clock dividers .....                 | 8         |
| 4.2.4      | SYSCON flash wait states .....              | 8         |
| <b>4.3</b> | <b>Typical use case</b> .....               | <b>8</b>  |
| <b>4.4</b> | <b>Macro Definition Documentation</b> ..... | <b>13</b> |
| 4.4.1      | FSL_CLOCK_DRIVER_VERSION .....              | 13        |
| 4.4.2      | CLOCK_FRO_SETTING_API_ROM_ADDRESS .....     | 13        |
| 4.4.3      | ADC_CLOCKS .....                            | 13        |
| 4.4.4      | ACMP_CLOCKS .....                           | 14        |
| 4.4.5      | DAC_CLOCKS .....                            | 14        |
| 4.4.6      | SWM_CLOCKS .....                            | 14        |
| 4.4.7      | ROM_CLOCKS .....                            | 14        |
| 4.4.8      | SRAM_CLOCKS .....                           | 14        |
| 4.4.9      | IOCON_CLOCKS .....                          | 15        |
| 4.4.10     | GPIO_CLOCKS .....                           | 15        |
| 4.4.11     | GPIO_INT_CLOCKS .....                       | 15        |
| 4.4.12     | CRC_CLOCKS .....                            | 15        |
| 4.4.13     | WWDT_CLOCKS .....                           | 15        |
| 4.4.14     | SCT_CLOCKS .....                            | 16        |
| 4.4.15     | I2C_CLOCKS .....                            | 16        |
| 4.4.16     | USART_CLOCKS .....                          | 16        |
| 4.4.17     | SPI_CLOCKS .....                            | 16        |
| 4.4.18     | CAPT_CLOCKS .....                           | 16        |
| 4.4.19     | CTIMER_CLOCKS .....                         | 17        |

| Section No.                       | Title                                       | Page No.  |
|-----------------------------------|---|-----------|
| 4.4.20                            | MRT_CLOCKS .....                            | 17        |
| 4.4.21                            | WKT_CLOCKS .....                            | 17        |
| 4.4.22                            | PLU_CLOCKS .....                            | 17        |
| 4.4.23                            | CLK_GATE_DEFINE .....                       | 17        |
| <b>4.5</b>                        | <b>Typedef Documentation</b> .....          | <b>18</b> |
| 4.5.1                             | clock_ip_name_t .....                       | 18        |
| 4.5.2                             | clock_name_t .....                          | 18        |
| 4.5.3                             | clock_select_t .....                        | 18        |
| 4.5.4                             | clock_fro_osc_freq_t .....                  | 18        |
| 4.5.5                             | clock_main_clk_src_t .....                  | 18        |
| <b>4.6</b>                        | <b>Enumeration Type Documentation</b> ..... | <b>18</b> |
| 4.6.1                             | _clock_ip_name .....                        | 18        |
| 4.6.2                             | _clock_name .....                           | 19        |
| 4.6.3                             | _clock_select .....                         | 19        |
| 4.6.4                             | _clock_divider .....                        | 20        |
| 4.6.5                             | _clock_fro_osc_freq .....                   | 20        |
| 4.6.6                             | _clock_main_clk_src .....                   | 20        |
| <b>4.7</b>                        | <b>Function Documentation</b> .....         | <b>20</b> |
| 4.7.1                             | CLOCK_SetMainClkSrc .....                   | 20        |
| 4.7.2                             | CLOCK_GetFRG0ClkFreq .....                  | 21        |
| 4.7.3                             | CLOCK_GetMainClkFreq .....                  | 21        |
| 4.7.4                             | CLOCK_GetFroFreq .....                      | 21        |
| 4.7.5                             | CLOCK_GetCoreSysClkFreq .....               | 21        |
| 4.7.6                             | CLOCK_GetClockOutClkFreq .....              | 21        |
| 4.7.7                             | CLOCK_GetUart0ClkFreq .....                 | 21        |
| 4.7.8                             | CLOCK_GetUart1ClkFreq .....                 | 22        |
| 4.7.9                             | CLOCK_GetFreq .....                         | 22        |
| 4.7.10                            | CLOCK_GetLPOscFreq .....                    | 22        |
| 4.7.11                            | CLOCK_GetExtClkFreq .....                   | 22        |
| 4.7.12                            | CLOCK_SetFRG0ClkFreq .....                  | 22        |
| 4.7.13                            | CLOCK_InitExtClkin .....                    | 23        |
| 4.7.14                            | CLOCK_SetFroOscFreq .....                   | 23        |
| <b>4.8</b>                        | <b>Variable Documentation</b> .....         | <b>23</b> |
| 4.8.1                             | g_LP_Osc_Freq .....                         | 23        |
| 4.8.2                             | g_Ext_Clk_Freq .....                        | 23        |
| 4.8.3                             | g_Fro_Osc_Freq .....                        | 23        |
| <br><b>Chapter 5 Power Driver</b> |   |           |
| <b>5.1</b>                        | <b>Overview</b> .....                       | <b>24</b> |

| Section No.                       | Title                                 | Page No.  |
|-----------------------------------|---------------------------------------|-----------|
| <b>5.2</b>                        | <b>Macro Definition Documentation</b> | <b>26</b> |
| 5.2.1                             | FSL_POWER_DRIVER_VERSION              | 26        |
| <b>5.3</b>                        | <b>Enumeration Type Documentation</b> | <b>26</b> |
| 5.3.1                             | _power_gen_reg                        | 26        |
| 5.3.2                             | _power_bod_reset_level                | 26        |
| 5.3.3                             | _power_bod_interrupt_level            | 26        |
| <b>5.4</b>                        | <b>Function Documentation</b>         | <b>27</b> |
| 5.4.1                             | POWER_EnablePD                        | 27        |
| 5.4.2                             | POWER_DisablePD                       | 27        |
| 5.4.3                             | POWER_EnableDeepSleep                 | 27        |
| 5.4.4                             | POWER_DisableDeepSleep                | 27        |
| 5.4.5                             | POWER_EnterSleep                      | 28        |
| 5.4.6                             | POWER_EnterDeepSleep                  | 28        |
| 5.4.7                             | POWER_EnterPowerDown                  | 28        |
| 5.4.8                             | POWER_EnterDeepPowerDownMode          | 28        |
| 5.4.9                             | POWER_GetSleepModeFlag                | 28        |
| 5.4.10                            | POWER_GetDeepPowerDownModeFlag        | 29        |
| 5.4.11                            | POWER_EnableNonDpd                    | 29        |
| 5.4.12                            | POWER_EnableLPO                       | 29        |
| 5.4.13                            | POWER_WakeUpConfig                    | 29        |
| 5.4.14                            | POWER_DeepSleepConfig                 | 29        |
| 5.4.15                            | POWER_SetRetainData                   | 30        |
| 5.4.16                            | POWER_GetRetainData                   | 30        |
| 5.4.17                            | POWER_SetBodLevel                     | 30        |
| <br><b>Chapter 6 Reset Driver</b> |                                       |           |
| <b>6.1</b>                        | <b>Overview</b>                       | <b>32</b> |
| <b>6.2</b>                        | <b>Macro Definition Documentation</b> | <b>33</b> |
| 6.2.1                             | FLASH_RSTS_N                          | 33        |
| <b>6.3</b>                        | <b>Typedef Documentation</b>          | <b>33</b> |
| 6.3.1                             | SYSCON_RSTn_t                         | 33        |
| <b>6.4</b>                        | <b>Enumeration Type Documentation</b> | <b>33</b> |
| 6.4.1                             | _SYSCON_RSTn                          | 33        |
| <b>6.5</b>                        | <b>Function Documentation</b>         | <b>34</b> |
| 6.5.1                             | RESET_SetPeripheralReset              | 34        |
| 6.5.2                             | RESET_ClearPeripheralReset            | 34        |
| 6.5.3                             | RESET_PeripheralReset                 | 34        |
| 6.5.4                             | RESET_ReleasePeripheralReset          | 35        |

| Section No.                             | Title                                 | Page No.  |
|---|---------------------------------------|-----------|
| <b>Chapter 7 CAPT: Capacitive Touch</b> |                                       |           |
| <b>7.1</b>                              | <b>Overview</b>                       | <b>36</b> |
| <b>7.2</b>                              | <b>Typical use case</b>               | <b>36</b> |
| 7.2.1                                   | Normal Configuration                  | 36        |
| <b>7.3</b>                              | <b>Data Structure Documentation</b>   | <b>39</b> |
| 7.3.1                                   | struct_capt_config                    | 39        |
| 7.3.2                                   | struct_capt_touch_data                | 40        |
| <b>7.4</b>                              | <b>Macro Definition Documentation</b> | <b>41</b> |
| 7.4.1                                   | FSL_CAPT_DRIVER_VERSION               | 41        |
| <b>7.5</b>                              | <b>Enumeration Type Documentation</b> | <b>41</b> |
| 7.5.1                                   | _capt_xpins                           | 41        |
| 7.5.2                                   | _capt_interrupt_enable                | 41        |
| 7.5.3                                   | _capt_interrupt_status_flags          | 42        |
| 7.5.4                                   | _capt_status_flags                    | 42        |
| 7.5.5                                   | _capt_trigger_mode                    | 42        |
| 7.5.6                                   | _capt_inactive_xpins_mode             | 42        |
| 7.5.7                                   | _capt_measurement_delay               | 43        |
| 7.5.8                                   | _capt_reset_delay                     | 43        |
| 7.5.9                                   | _capt_polling_mode                    | 43        |
| 7.5.10                                  | _capt_dma_mode                        | 43        |
| <b>7.6</b>                              | <b>Function Documentation</b>         | <b>43</b> |
| 7.6.1                                   | CAPT_Init                             | 43        |
| 7.6.2                                   | CAPT_Deinit                           | 44        |
| 7.6.3                                   | CAPT_GetDefaultConfig                 | 44        |
| 7.6.4                                   | CAPT_SetThreshold                     | 44        |
| 7.6.5                                   | CAPT_SetPollMode                      | 45        |
| 7.6.6                                   | CAPT_EnableInterrupts                 | 45        |
| 7.6.7                                   | CAPT_DisableInterrupts                | 45        |
| 7.6.8                                   | CAPT_GetInterruptStatusFlags          | 45        |
| 7.6.9                                   | CAPT_ClearInterruptStatusFlags        | 46        |
| 7.6.10                                  | CAPT_GetStatusFlags                   | 46        |
| 7.6.11                                  | CAPT_GetTouchData                     | 46        |
| 7.6.12                                  | CAPT_PollNow                          | 47        |
| <b>Chapter 8 Common Driver</b>          |                                       |           |
| <b>8.1</b>                              | <b>Overview</b>                       | <b>48</b> |
| <b>8.2</b>                              | <b>Macro Definition Documentation</b> | <b>54</b> |
| 8.2.1                                   | FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ   | 54        |

| Section No. | Title                                       | Page No.  |
|-------------|---|-----------|
| 8.2.2       | MAKE_STATUS .....                           | 54        |
| 8.2.3       | MAKE_VERSION .....                          | 54        |
| 8.2.4       | FSL_COMMON_DRIVER_VERSION .....             | 55        |
| 8.2.5       | DEBUG_CONSOLE_DEVICE_TYPE_NONE .....        | 55        |
| 8.2.6       | DEBUG_CONSOLE_DEVICE_TYPE_UART .....        | 55        |
| 8.2.7       | DEBUG_CONSOLE_DEVICE_TYPE_LPUART .....      | 55        |
| 8.2.8       | DEBUG_CONSOLE_DEVICE_TYPE_LPSCI .....       | 55        |
| 8.2.9       | DEBUG_CONSOLE_DEVICE_TYPE_USBCDC .....      | 55        |
| 8.2.10      | DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM .....    | 55        |
| 8.2.11      | DEBUG_CONSOLE_DEVICE_TYPE_IUART .....       | 55        |
| 8.2.12      | DEBUG_CONSOLE_DEVICE_TYPE_VUSART .....      | 55        |
| 8.2.13      | DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART .....  | 55        |
| 8.2.14      | DEBUG_CONSOLE_DEVICE_TYPE_SWO .....         | 55        |
| 8.2.15      | DEBUG_CONSOLE_DEVICE_TYPE_QSCI .....        | 55        |
| 8.2.16      | MIN .....                                   | 55        |
| 8.2.17      | MAX .....                                   | 55        |
| 8.2.18      | ARRAY_SIZE .....                            | 55        |
| 8.2.19      | UINT16_MAX .....                            | 55        |
| 8.2.20      | UINT32_MAX .....                            | 55        |
| 8.2.21      | SUPPRESS_FALL_THROUGH_WARNING .....         | 55        |
| 8.2.22      | SDK_SIZEALIGN .....                         | 56        |
| <b>8.3</b>  | <b>Typedef Documentation .....</b>          | <b>56</b> |
| 8.3.1       | status_t .....                              | 56        |
| <b>8.4</b>  | <b>Enumeration Type Documentation .....</b> | <b>56</b> |
| 8.4.1       | _status_groups .....                        | 56        |
| 8.4.2       | anonymous enum .....                        | 59        |
| <b>8.5</b>  | <b>Function Documentation .....</b>         | <b>59</b> |
| 8.5.1       | SDK_Malloc .....                            | 59        |
| 8.5.2       | SDK_Free .....                              | 59        |
| 8.5.3       | SDK_DelayAtLeastUs .....                    | 59        |
| 8.5.4       | EnableIRQ .....                             | 60        |
| 8.5.5       | DisableIRQ .....                            | 60        |
| 8.5.6       | EnableIRQWithPriority .....                 | 61        |
| 8.5.7       | IRQ_SetPriority .....                       | 61        |
| 8.5.8       | IRQ_ClearPendingIRQ .....                   | 62        |
| 8.5.9       | DisableGlobalIRQ .....                      | 62        |
| 8.5.10      | EnableGlobalIRQ .....                       | 63        |
| 8.5.11      | EnableDeepSleepIRQ .....                    | 63        |
| 8.5.12      | DisableDeepSleepIRQ .....                   | 63        |

| Section No.                                      | Title                                 | Page No.  |
|--|---------------------------------------|-----------|
| <b>Chapter 9 CTIMER: Standard counter/timers</b> |                                       |           |
| <b>9.1</b>                                       | <b>Overview</b>                       | <b>65</b> |
| <b>9.2</b>                                       | <b>Function groups</b>                | <b>65</b> |
| 9.2.1  | Initialization and deinitialization   | 65        |
| 9.2.2  | PWM Operations                        | 65        |
| 9.2.3  | Match Operation                       | 65        |
| 9.2.4  | Input capture operations              | 65        |
| <b>9.3</b>                                       | <b>Typical use case</b>               | <b>66</b> |
| 9.3.1  | Match example                         | 66        |
| 9.3.2  | PWM output example                    | 66        |
| <b>9.4</b>                                       | <b>Data Structure Documentation</b>   | <b>70</b> |
| 9.4.1  | struct_ctimer_match_config            | 70        |
| 9.4.2  | struct_ctimer_config                  | 70        |
| <b>9.5</b>                                       | <b>Typedef Documentation</b>          | <b>71</b> |
| 9.5.1  | ctimer_match_config_t                 | 71        |
| 9.5.2  | ctimer_config_t                       | 71        |
| <b>9.6</b>                                       | <b>Enumeration Type Documentation</b> | <b>71</b> |
| 9.6.1  | _ctimer_capture_channel               | 71        |
| 9.6.2  | _ctimer_capture_edge                  | 71        |
| 9.6.3  | _ctimer_match                         | 71        |
| 9.6.4  | _ctimer_external_match                | 72        |
| 9.6.5  | _ctimer_match_output_control          | 72        |
| 9.6.6  | _ctimer_interrupt_enable              | 72        |
| 9.6.7  | _ctimer_status_flags                  | 72        |
| 9.6.8  | ctimer_callback_type_t                | 73        |
| <b>9.7</b>                                       | <b>Function Documentation</b>         | <b>73</b> |
| 9.7.1  | CTIMER_Init                           | 73        |
| 9.7.2  | CTIMER_Deinit                         | 73        |
| 9.7.3  | CTIMER_GetDefaultConfig               | 73        |
| 9.7.4  | CTIMER_SetupPwmPeriod                 | 74        |
| 9.7.5  | CTIMER_SetupPwm                       | 74        |
| 9.7.6  | CTIMER_UpdatePwmPulsePeriod           | 75        |
| 9.7.7  | CTIMER_UpdatePwmDutycycle             | 75        |
| 9.7.8  | CTIMER_SetupMatch                     | 76        |
| 9.7.9  | CTIMER_GetOutputMatchStatus           | 76        |
| 9.7.10   | CTIMER_SetupCapture                   | 76        |
| 9.7.11   | CTIMER_GetTimerCountValue             | 77        |
| 9.7.12   | CTIMER_RegisterCallBack               | 77        |
| 9.7.13   | CTIMER_EnableInterrupts               | 77        |

| <b>Section No.</b>                                       | <b>Title</b>                          | <b>Page No.</b> |
|--|---------------------------------------|-----------------|
| 9.7.14   | CTIMER_DisableInterrupts              | 78              |
| 9.7.15   | CTIMER_GetEnabledInterrupts           | 78              |
| 9.7.16   | CTIMER_GetStatusFlags                 | 78              |
| 9.7.17   | CTIMER_ClearStatusFlags               | 79              |
| 9.7.18   | CTIMER_StartTimer                     | 80              |
| 9.7.19   | CTIMER_StopTimer                      | 80              |
| 9.7.20   | CTIMER_Reset                          | 80              |
| 9.7.21   | CTIMER_SetPrescale                    | 80              |
| 9.7.22   | CTIMER_GetCaptureValue                | 81              |
| 9.7.23   | CTIMER_EnableResetMatchChannel        | 81              |
| 9.7.24   | CTIMER_EnableStopMatchChannel         | 81              |
| 9.7.25   | CTIMER_EnableMatchChannelReload       | 82              |
| 9.7.26   | CTIMER_EnableRisingEdgeCapture        | 82              |
| 9.7.27   | CTIMER_EnableFallingEdgeCapture       | 82              |
| 9.7.28   | CTIMER_SetShadowValue                 | 83              |
| <br>   |                                       |                 |
| <b>Chapter 10 IAP: In Application Programming Driver</b> |                                       |                 |
| <b>10.1</b>  | <b>Overview</b>                       | <b>84</b>       |
| <b>10.2</b>  | <b>Function groups</b>                | <b>84</b>       |
| 10.2.1   | Basic operations                      | 84              |
| 10.2.2   | Flash operations                      | 84              |
| 10.2.3   | EEPROM operations                     | 85              |
| 10.2.4   | FAIM operations                       | 85              |
| <b>10.3</b>  | <b>Typical use case</b>               | <b>85</b>       |
| 10.3.1   | IAP Basic Operations                  | 85              |
| 10.3.2   | IAP Flash Operations                  | 85              |
| 10.3.3   | IAP EEPROM Operations                 | 85              |
| 10.3.4   | IAP FAIM Operations                   | 85              |
| <b>10.4</b>  | <b>Enumeration Type Documentation</b> | <b>87</b>       |
| 10.4.1   | anonymous enum                        | 87              |
| 10.4.2   | _iap_commands                         | 88              |
| 10.4.3   | _flash_access_time                    | 88              |
| <b>10.5</b>  | <b>Function Documentation</b>         | <b>88</b>       |
| 10.5.1   | IAP_ReadPartID                        | 88              |
| 10.5.2   | IAP_ReadBootCodeVersion               | 89              |
| 10.5.3   | IAP_ReinvokeISP                       | 89              |
| 10.5.4   | IAP_ReadUniqueID                      | 89              |
| 10.5.5   | IAP_PrepareSectorForWrite             | 90              |
| 10.5.6   | IAP_CopyRamToFlash                    | 90              |
| 10.5.7   | IAP_EraseSector                       | 91              |

| <b>Section No.</b>   | <b>Title</b>                                | <b>Page No.</b> |
|--|---|-----------------|
| 10.5.8   | IAP_ErasePage .....                         | 92              |
| 10.5.9   | IAP_BlankCheckSector .....                  | 93              |
| 10.5.10  | IAP_Compare .....                           | 93              |
| <br>   |   |                 |
| <b>Chapter 11 LPC_ACOMP: Analog comparator Driver</b>                |   |                 |
| <b>11.1</b>  | <b>Overview .....</b>                       | <b>95</b>       |
| <b>11.2</b>  | <b>Typical use case .....</b>               | <b>95</b>       |
| 11.2.1   | Polling Configuration .....                 | 95              |
| 11.2.2   | Interrupt Configuration .....               | 95              |
| <b>11.3</b>  | <b>Data Structure Documentation .....</b>   | <b>97</b>       |
| 11.3.1   | struct _acomp_config .....                  | 97              |
| 11.3.2   | struct _acomp_ladder_config .....           | 97              |
| <b>11.4</b>  | <b>Macro Definition Documentation .....</b> | <b>97</b>       |
| 11.4.1   | FSL_ACOMP_DRIVER_VERSION .....              | 97              |
| <b>11.5</b>  | <b>Enumeration Type Documentation .....</b> | <b>97</b>       |
| 11.5.1   | _acomp_ladder_reference_voltage .....       | 97              |
| 11.5.2   | _acomp_interrupt_enable .....               | 98              |
| 11.5.3   | _acomp_hysteresis_selection .....           | 98              |
| <b>11.6</b>  | <b>Function Documentation .....</b>         | <b>98</b>       |
| 11.6.1   | ACOMP_Init .....                            | 98              |
| 11.6.2   | ACOMP_Deinit .....                          | 98              |
| 11.6.3   | ACOMP_GetDefaultConfig .....                | 98              |
| 11.6.4   | ACOMP_EnableInterrupts .....                | 99              |
| 11.6.5   | ACOMP_GetInterruptsStatusFlags .....        | 99              |
| 11.6.6   | ACOMP_ClearInterruptsStatusFlags .....      | 99              |
| 11.6.7   | ACOMP_GetOutputStatusFlags .....            | 99              |
| 11.6.8   | ACOMP_SetInputChannel .....                 | 100             |
| 11.6.9   | ACOMP_SetLadderConfig .....                 | 100             |
| <br>   |   |                 |
| <b>Chapter 12 ADC: 12-bit SAR Analog-to-Digital Converter Driver</b> |   |                 |
| <b>12.1</b>  | <b>Overview .....</b>                       | <b>101</b>      |
| <b>12.2</b>  | <b>Typical use case .....</b>               | <b>101</b>      |
| 12.2.1   | Polling Configuration .....                 | 101             |
| 12.2.2   | Interrupt Configuration .....               | 101             |
| <b>12.3</b>  | <b>Data Structure Documentation .....</b>   | <b>106</b>      |
| 12.3.1   | struct _adc_config .....                    | 106             |
| 12.3.2   | struct _adc_conv_seq_config .....           | 107             |

| Section No. | Title  | Page No.   |
|-------------|--|------------|
| 12.3.3      | <code>struct_adc_result_info</code> .....                | 107        |
| <b>12.4</b> | <b>Macro Definition Documentation</b> .....              | <b>108</b> |
| 12.4.1      | <code>FSL_ADC_DRIVER_VERSION</code> .....                | 108        |
| <b>12.5</b> | <b>Enumeration Type Documentation</b> .....              | <b>108</b> |
| 12.5.1      | <code>_adc_status_flags</code> .....                     | 108        |
| 12.5.2      | <code>_adc_interrupt_enable</code> .....                 | 109        |
| 12.5.3      | <code>_adc_trigger_polarity</code> .....                 | 110        |
| 12.5.4      | <code>_adc_priority</code> .....                         | 110        |
| 12.5.5      | <code>_adc_seq_interrupt_mode</code> .....               | 110        |
| 12.5.6      | <code>_adc_threshold_compare_status</code> .....         | 110        |
| 12.5.7      | <code>_adc_threshold_crossing_status</code> .....        | 110        |
| 12.5.8      | <code>_adc_threshold_interrupt_mode</code> .....         | 111        |
| 12.5.9      | <code>_adc_inforeultshift</code> .....                   | 111        |
| 12.5.10     | <code>_adc_tempsensor_common_mode</code> .....           | 111        |
| 12.5.11     | <code>_adc_second_control</code> .....                   | 111        |
| <b>12.6</b> | <b>Function Documentation</b> .....                      | <b>112</b> |
| 12.6.1      | <code>ADC_Init</code> .....                              | 112        |
| 12.6.2      | <code>ADC_Deinit</code> .....                            | 113        |
| 12.6.3      | <code>ADC_GetDefaultConfig</code> .....                  | 113        |
| 12.6.4      | <code>ADC_EnableConvSeqA</code> .....                    | 113        |
| 12.6.5      | <code>ADC_SetConvSeqAConfig</code> .....                 | 114        |
| 12.6.6      | <code>ADC_DoSoftwareTriggerConvSeqA</code> .....         | 114        |
| 12.6.7      | <code>ADC_EnableConvSeqABurstMode</code> .....           | 114        |
| 12.6.8      | <code>ADC_SetConvSeqAHighPriority</code> .....           | 114        |
| 12.6.9      | <code>ADC_EnableConvSeqB</code> .....                    | 115        |
| 12.6.10     | <code>ADC_SetConvSeqBConfig</code> .....                 | 115        |
| 12.6.11     | <code>ADC_DoSoftwareTriggerConvSeqB</code> .....         | 115        |
| 12.6.12     | <code>ADC_EnableConvSeqBBurstMode</code> .....           | 115        |
| 12.6.13     | <code>ADC_SetConvSeqBHighPriority</code> .....           | 117        |
| 12.6.14     | <code>ADC_GetConvSeqAGlobalConversionResult</code> ..... | 117        |
| 12.6.15     | <code>ADC_GetConvSeqBGlobalConversionResult</code> ..... | 117        |
| 12.6.16     | <code>ADC_GetChannelConversionResult</code> .....        | 118        |
| 12.6.17     | <code>ADC_SetThresholdPair0</code> .....                 | 118        |
| 12.6.18     | <code>ADC_SetThresholdPair1</code> .....                 | 118        |
| 12.6.19     | <code>ADC_SetChannelWithThresholdPair0</code> .....      | 119        |
| 12.6.20     | <code>ADC_SetChannelWithThresholdPair1</code> .....      | 119        |
| 12.6.21     | <code>ADC_EnableInterrupts</code> .....                  | 119        |
| 12.6.22     | <code>ADC_DisableInterrupts</code> .....                 | 119        |
| 12.6.23     | <code>ADC_EnableThresholdCompareInterrupt</code> .....   | 120        |
| 12.6.24     | <code>ADC_GetStatusFlags</code> .....                    | 120        |
| 12.6.25     | <code>ADC_ClearStatusFlags</code> .....                  | 120        |

| Section No.  | Title  | Page No.   |
|--|--|------------|
| <b>Chapter 13 CRC: Cyclic Redundancy Check Driver</b>            |  |            |
| <b>13.1</b>  | <b>Overview</b>                                    | <b>122</b> |
| <b>13.2</b>  | <b>CRC Driver Initialization and Configuration</b> | <b>122</b> |
| <b>13.3</b>  | <b>CRC Write Data</b>                              | <b>122</b> |
| <b>13.4</b>  | <b>CRC Get Checksum</b>                            | <b>122</b> |
| <b>13.5</b>  | <b>Comments about API usage in RTOS</b>            | <b>123</b> |
| <b>13.6</b>  | <b>Data Structure Documentation</b>                | <b>124</b> |
| 13.6.1   | struct_crc_config                                  | 124        |
| <b>13.7</b>  | <b>Macro Definition Documentation</b>              | <b>125</b> |
| 13.7.1   | FSL_CRC_DRIVER_VERSION                             | 125        |
| 13.7.2   | CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT        | 125        |
| <b>13.8</b>  | <b>Typedef Documentation</b>                       | <b>126</b> |
| 13.8.1   | crc_polynomial_t                                   | 126        |
| 13.8.2   | crc_config_t                                       | 126        |
| <b>13.9</b>  | <b>Enumeration Type Documentation</b>              | <b>126</b> |
| 13.9.1   | _crc_polynomial                                    | 126        |
| <b>13.10</b>   | <b>Function Documentation</b>                      | <b>126</b> |
| 13.10.1  | CRC_Init   | 126        |
| 13.10.2  | CRC_Deinit   | 126        |
| 13.10.3  | CRC_Reset  | 126        |
| 13.10.4  | CRC_WriteSeed                                      | 127        |
| 13.10.5  | CRC_GetDefaultConfig                               | 127        |
| 13.10.6  | CRC_GetConfig                                      | 127        |
| 13.10.7  | CRC_WriteData                                      | 127        |
| 13.10.8  | CRC_Get32bitResult                                 | 128        |
| 13.10.9  | CRC_Get16bitResult                                 | 128        |
| <b>Chapter 14 DAC: 10-bit Digital To Analog Converter Driver</b> |  |            |
| <b>14.1</b>  | <b>Overview</b>                                    | <b>129</b> |
| <b>14.2</b>  | <b>Typical use case</b>                            | <b>129</b> |
| 14.2.1   | Polling Configuration                              | 129        |
| 14.2.2   | Interrupt Configuration                            | 129        |
| <b>14.3</b>  | <b>Data Structure Documentation</b>                | <b>130</b> |
| 14.3.1   | struct_dac_config                                  | 130        |

| Section No.                                     | Title                                 | Page No.   |
|---|---------------------------------------|------------|
| <b>14.4</b>                                     | <b>Macro Definition Documentation</b> | <b>130</b> |
| 14.4.1  | LPC_DAC_DRIVER_VERSION                | 130        |
| <b>14.5</b>                                     | <b>Enumeration Type Documentation</b> | <b>131</b> |
| 14.5.1  | _dac_settling_time                    | 131        |
| <b>14.6</b>                                     | <b>Function Documentation</b>         | <b>131</b> |
| 14.6.1  | DAC_Init                              | 131        |
| 14.6.2  | DAC_Deinit                            | 131        |
| 14.6.3  | DAC_GetDefaultConfig                  | 131        |
| 14.6.4  | DAC_EnableDoubleBuffering             | 132        |
| 14.6.5  | DAC_SetBufferValue                    | 132        |
| 14.6.6  | DAC_SetCounterValue                   | 132        |
| 14.6.7  | DAC_EnableCounter                     | 133        |
| 14.6.8  | DAC_GetDMAInterruptRequestFlag        | 133        |
| <br><b>Chapter 15 GPIO: General Purpose I/O</b> |                                       |            |
| <b>15.1</b>                                     | <b>Overview</b>                       | <b>134</b> |
| <b>15.2</b>                                     | <b>Function groups</b>                | <b>134</b> |
| 15.2.1  | Initialization and deinitialization   | 134        |
| 15.2.2  | Pin manipulation                      | 134        |
| 15.2.3  | Port manipulation                     | 134        |
| 15.2.4  | Port masking                          | 134        |
| <b>15.3</b>                                     | <b>Typical use case</b>               | <b>134</b> |
| <b>15.4</b>                                     | <b>Data Structure Documentation</b>   | <b>136</b> |
| 15.4.1  | struct_gpio_pin_config                | 136        |
| <b>15.5</b>                                     | <b>Macro Definition Documentation</b> | <b>136</b> |
| 15.5.1  | FSL_GPIO_DRIVER_VERSION               | 136        |
| <b>15.6</b>                                     | <b>Typedef Documentation</b>          | <b>136</b> |
| 15.6.1  | gpio_pin_config_t                     | 136        |
| <b>15.7</b>                                     | <b>Enumeration Type Documentation</b> | <b>136</b> |
| 15.7.1  | _gpio_pin_direction                   | 136        |
| <b>15.8</b>                                     | <b>Function Documentation</b>         | <b>136</b> |
| 15.8.1  | GPIO_PortInit                         | 136        |
| 15.8.2  | GPIO_PinInit                          | 137        |
| 15.8.3  | GPIO_PinWrite                         | 137        |
| 15.8.4  | GPIO_PinRead                          | 138        |
| 15.8.5  | GPIO_PortSet                          | 138        |

| <b>Section No.</b>   | <b>Title</b>  | <b>Page No.</b> |
|--|---|-----------------|
| 15.8.6   | GPIO_PortClear .....                                    | 138             |
| 15.8.7   | GPIO_PortToggle .....                                   | 139             |
| <br><b>Chapter 16 I2C: Inter-Integrated Circuit Driver</b> |   |                 |
| <b>16.1</b>  | <b>Overview .....</b>                                   | <b>140</b>      |
| <b>16.2</b>  | <b>Typical use case .....</b>                           | <b>140</b>      |
| 16.2.1   | Master Operation in functional method .....             | 140             |
| 16.2.2   | Master Operation in DMA transactional method .....      | 140             |
| 16.2.3   | Slave Operation in functional method .....              | 140             |
| 16.2.4   | Slave Operation in interrupt transactional method ..... | 141             |
| <b>16.3</b>  | <b>I2C Driver .....</b>                                 | <b>142</b>      |
| 16.3.1   | Overview .....  | 142             |
| 16.3.2   | Macro Definition Documentation .....                    | 143             |
| 16.3.3   | Enumeration Type Documentation .....                    | 143             |
| <b>16.4</b>  | <b>I2C Master Driver .....</b>                          | <b>144</b>      |
| 16.4.1   | Overview .....  | 144             |
| 16.4.2   | Data Structure Documentation .....                      | 146             |
| 16.4.3   | Typedef Documentation .....                             | 148             |
| 16.4.4   | Enumeration Type Documentation .....                    | 149             |
| 16.4.5   | Function Documentation .....                            | 150             |
| <b>16.5</b>  | <b>I2C Slave Driver .....</b>                           | <b>161</b>      |
| 16.5.1   | Overview .....  | 161             |
| 16.5.2   | Data Structure Documentation .....                      | 164             |
| 16.5.3   | Typedef Documentation .....                             | 167             |
| 16.5.4   | Enumeration Type Documentation .....                    | 169             |
| 16.5.5   | Function Documentation .....                            | 170             |
| <br><b>Chapter 17 IOCON: I/O pin configuration</b>         |   |                 |
| <b>17.1</b>  | <b>Overview .....</b>                                   | <b>178</b>      |
| <b>17.2</b>  | <b>Function groups .....</b>                            | <b>178</b>      |
| 17.2.1   | Pin mux set .....                                       | 178             |
| 17.2.2   | Pin mux set .....                                       | 178             |
| <b>17.3</b>  | <b>Typical use case .....</b>                           | <b>178</b>      |
| <b>17.4</b>  | <b>Data Structure Documentation .....</b>               | <b>179</b>      |
| 17.4.1   | struct_iocon_group .....                                | 179             |
| <b>17.5</b>  | <b>Macro Definition Documentation .....</b>             | <b>179</b>      |

| <b>Section No.</b>  | <b>Title</b>  | <b>Page No.</b> |
|---|---|-----------------|
| 17.5.1  | LPC_IOCON_DRIVER_VERSION .....                      | 179             |
| <b>17.6</b>   | <b>Function Documentation</b> .....                 | <b>179</b>      |
| 17.6.1  | IOCON_PinMuxSet .....                               | 179             |
| 17.6.2  | IOCON_SetPinMuxing .....                            | 179             |
| <br><b>Chapter 18 SPI: Serial Peripheral Interface Driver</b>                   |   |                 |
| <b>18.1</b>   | <b>Overview</b> .....                               | <b>181</b>      |
| <b>18.2</b>   | <b>Typical use case</b> .....                       | <b>181</b>      |
| 18.2.1  | SPI master transfer using an interrupt method ..... | 181             |
| <b>18.3</b>   | <b>SPI Driver</b> .....                             | <b>182</b>      |
| 18.3.1  | Overview .....                                      | 182             |
| 18.3.2  | Data Structure Documentation .....                  | 186             |
| 18.3.3  | Macro Definition Documentation .....                | 190             |
| 18.3.4  | Typedef Documentation .....                         | 190             |
| 18.3.5  | Enumeration Type Documentation .....                | 190             |
| 18.3.6  | Function Documentation .....                        | 192             |
| <br><b>Chapter 19 USART: Universal Asynchronous Receiver/Transmitter Driver</b> |   |                 |
| <b>19.1</b>   | <b>Overview</b> .....                               | <b>204</b>      |
| <b>19.2</b>   | <b>Typical use case</b> .....                       | <b>205</b>      |
| 19.2.1  | USART Send/receive using a polling method .....     | 205             |
| 19.2.2  | USART Send/receive using an interrupt method .....  | 205             |
| 19.2.3  | USART Receive using the ringbuffer feature .....    | 205             |
| 19.2.4  | USART Send/Receive using the DMA method .....       | 205             |
| <b>19.3</b>   | <b>USART Driver</b> .....                           | <b>206</b>      |
| 19.3.1  | Overview .....                                      | 206             |
| 19.3.2  | Data Structure Documentation .....                  | 210             |
| 19.3.3  | Macro Definition Documentation .....                | 213             |
| 19.3.4  | Typedef Documentation .....                         | 214             |
| 19.3.5  | Enumeration Type Documentation .....                | 214             |
| 19.3.6  | Function Documentation .....                        | 216             |
| <br><b>Chapter 20 MRT: Multi-Rate Timer</b>                                     |   |                 |
| <b>20.1</b>   | <b>Overview</b> .....                               | <b>230</b>      |
| <b>20.2</b>   | <b>Function groups</b> .....                        | <b>230</b>      |
| 20.2.1  | Initialization and deinitialization .....           | 230             |

| <b>Section No.</b>   | <b>Title</b>  | <b>Page No.</b> |
|--|---|-----------------|
| 20.2.2   | Timer period Operations .....                                 | 230             |
| 20.2.3   | Start and Stop timer operations .....                         | 230             |
| 20.2.4   | Get and release channel .....                                 | 231             |
| 20.2.5   | Status .....  | 231             |
| 20.2.6   | Interrupt .....   | 231             |
| <b>20.3</b>  | <b>Typical use case .....</b>                                 | <b>231</b>      |
| 20.3.1   | MRT tick example .....  | 231             |
| <b>20.4</b>  | <b>Data Structure Documentation .....</b>                     | <b>233</b>      |
| 20.4.1   | struct_mrt_config .....                                       | 233             |
| <b>20.5</b>  | <b>Typedef Documentation .....</b>                            | <b>233</b>      |
| 20.5.1   | mrt_config_t .....  | 233             |
| <b>20.6</b>  | <b>Enumeration Type Documentation .....</b>                   | <b>234</b>      |
| 20.6.1   | _mrt_chnl .....   | 234             |
| 20.6.2   | _mrt_timer_mode .....   | 234             |
| 20.6.3   | _mrt_interrupt_enable .....                                   | 234             |
| 20.6.4   | _mrt_status_flags .....                                       | 234             |
| <b>20.7</b>  | <b>Function Documentation .....</b>                           | <b>234</b>      |
| 20.7.1   | MRT_Init .....  | 234             |
| 20.7.2   | MRT_Deinit .....  | 235             |
| 20.7.3   | MRT_GetDefaultConfig .....                                    | 235             |
| 20.7.4   | MRT_SetupChannelMode .....                                    | 235             |
| 20.7.5   | MRT_EnableInterrupts .....                                    | 235             |
| 20.7.6   | MRT_DisableInterrupts .....                                   | 236             |
| 20.7.7   | MRT_GetEnabledInterrupts .....                                | 236             |
| 20.7.8   | MRT_GetStatusFlags .....                                      | 236             |
| 20.7.9   | MRT_ClearStatusFlags .....                                    | 237             |
| 20.7.10  | MRT_UpdateTimerPeriod .....                                   | 237             |
| 20.7.11  | MRT_GetCurrentTimerCount .....                                | 238             |
| 20.7.12  | MRT_StartTimer .....  | 238             |
| 20.7.13  | MRT_StopTimer .....   | 239             |
| 20.7.14  | MRT_GetIdleChannel .....                                      | 239             |
| <br><b>Chapter 21 PINT: Pin Interrupt and Pattern Match Driver</b> |   |                 |
| <b>21.1</b>  | <b>Overview .....</b>   | <b>240</b>      |
| <b>21.2</b>  | <b>Pin Interrupt and Pattern match Driver operation .....</b> | <b>240</b>      |
| 21.2.1   | Pin Interrupt use case .....                                  | 240             |
| 21.2.2   | Pattern match use case .....                                  | 240             |
| <b>21.3</b>  | <b>Typedef Documentation .....</b>                            | <b>243</b>      |

| Section No.       | Title                                       | Page No.   |
|-------------------|---|------------|
| 21.3.1            | pint_cb_t .....                             | 243        |
| <b>21.4</b>       | <b>Enumeration Type Documentation .....</b> | <b>243</b> |
| 21.4.1            | _pint_pin_enable .....                      | 243        |
| 21.4.2            | _pint_int .....                             | 244        |
| 21.4.3            | _pint_pmatch_input_src .....                | 244        |
| 21.4.4            | _pint_pmatch_bslice .....                   | 244        |
| 21.4.5            | _pint_pmatch_bslice_cfg .....               | 245        |
| <b>21.5</b>       | <b>Function Documentation .....</b>         | <b>245</b> |
| 21.5.1            | PINT_Init .....                             | 245        |
| 21.5.2            | PINT_PinInterruptConfig .....               | 245        |
| 21.5.3            | PINT_PinInterruptGetConfig .....            | 246        |
| 21.5.4            | PINT_PinInterruptClrStatus .....            | 246        |
| 21.5.5            | PINT_PinInterruptGetStatus .....            | 246        |
| 21.5.6            | PINT_PinInterruptClrStatusAll .....         | 247        |
| 21.5.7            | PINT_PinInterruptGetStatusAll .....         | 247        |
| 21.5.8            | PINT_PinInterruptClrFallFlag .....          | 247        |
| 21.5.9            | PINT_PinInterruptGetFallFlag .....          | 248        |
| 21.5.10           | PINT_PinInterruptClrFallFlagAll .....       | 248        |
| 21.5.11           | PINT_PinInterruptGetFallFlagAll .....       | 248        |
| 21.5.12           | PINT_PinInterruptClrRiseFlag .....          | 249        |
| 21.5.13           | PINT_PinInterruptGetRiseFlag .....          | 249        |
| 21.5.14           | PINT_PinInterruptClrRiseFlagAll .....       | 250        |
| 21.5.15           | PINT_PinInterruptGetRiseFlagAll .....       | 250        |
| 21.5.16           | PINT_PatternMatchConfig .....               | 250        |
| 21.5.17           | PINT_PatternMatchGetConfig .....            | 251        |
| 21.5.18           | PINT_PatternMatchGetStatus .....            | 251        |
| 21.5.19           | PINT_PatternMatchGetStatusAll .....         | 252        |
| 21.5.20           | PINT_PatternMatchResetDetectLogic .....     | 252        |
| 21.5.21           | PINT_PatternMatchEnable .....               | 252        |
| 21.5.22           | PINT_PatternMatchDisable .....              | 253        |
| 21.5.23           | PINT_PatternMatchEnableRXEV .....           | 253        |
| 21.5.24           | PINT_PatternMatchDisableRXEV .....          | 253        |
| 21.5.25           | PINT_EnableCallback .....                   | 254        |
| 21.5.26           | PINT_DisableCallback .....                  | 254        |
| 21.5.27           | PINT_Deinit .....                           | 254        |
| 21.5.28           | PINT_EnableCallbackByIndex .....            | 255        |
| 21.5.29           | PINT_DisableCallbackByIndex .....           | 255        |
| <br>              |   |            |
| <b>Chapter 22</b> | <b>PLU: Programmable Logic Unit</b>         |            |
| <b>22.1</b>       | <b>Overview .....</b>                       | <b>256</b> |
| <b>22.2</b>       | <b>Function groups .....</b>                | <b>256</b> |

| <b>Section No.</b>                              | <b>Title</b>                                  | <b>Page No.</b> |
|---|---|-----------------|
| 22.2.1  | Initialization and de-initialization .....    | 256             |
| 22.2.2  | Set input/output source and Truth Table ..... | 256             |
| 22.2.3  | Read current Output State .....               | 256             |
| 22.2.4  | Wake-up/Interrupt Control .....               | 256             |
| <b>22.3</b>                                     | <b>Typical use case .....</b>                 | <b>257</b>      |
| 22.3.1  | PLU combination example .....                 | 257             |
| <b>22.4</b>                                     | <b>Typedef Documentation .....</b>            | <b>261</b>      |
| 22.4.1  | plu_lut_in_index_t .....                      | 261             |
| <b>22.5</b>                                     | <b>Enumeration Type Documentation .....</b>   | <b>261</b>      |
| 22.5.1  | _plu_lut_index .....                          | 261             |
| 22.5.2  | _plu_lut_in_index .....                       | 262             |
| 22.5.3  | _plu_lut_input_source .....                   | 262             |
| 22.5.4  | _plu_output_index .....                       | 263             |
| 22.5.5  | _plu_output_source .....                      | 263             |
| <b>22.6</b>                                     | <b>Function Documentation .....</b>           | <b>264</b>      |
| 22.6.1  | PLU_Init .....                                | 264             |
| 22.6.2  | PLU_Deinit .....                              | 264             |
| 22.6.3  | PLU_SetLutInputSource .....                   | 265             |
| 22.6.4  | PLU_SetOutputSource .....                     | 265             |
| 22.6.5  | PLU_SetLutTruthTable .....                    | 265             |
| 22.6.6  | PLU_ReadOutputState .....                     | 266             |
| <br><b>Chapter 23 SWM: Switch Matrix Module</b> |   |                 |
| <b>23.1</b>                                     | <b>Overview .....</b>                         | <b>267</b>      |
| <b>23.2</b>                                     | <b>SWM: Switch Matrix Module .....</b>        | <b>267</b>      |
| 23.2.1  | SWM Operations .....                          | 267             |
| <b>23.3</b>                                     | <b>Macro Definition Documentation .....</b>   | <b>273</b>      |
| 23.3.1  | FSL_SWM_DRIVER_VERSION .....                  | 273             |
| <b>23.4</b>                                     | <b>Enumeration Type Documentation .....</b>   | <b>273</b>      |
| 23.4.1  | _swm_pinassignfixed_port_pin_type_t .....     | 273             |
| 23.4.2  | _swm_port_pin_type_t .....                    | 274             |
| 23.4.3  | _swm_pinassignfixed_select_movable_t .....    | 275             |
| 23.4.4  | _swm_select_movable_t .....                   | 275             |
| 23.4.5  | _swm_select_fixed_pin_t .....                 | 276             |
| <b>23.5</b>                                     | <b>Function Documentation .....</b>           | <b>277</b>      |
| 23.5.1  | SWM_SetMovablePinSelect .....                 | 277             |
| 23.5.2  | SWM_SetFixedMovablePinSelect .....            | 277             |

| <b>Section No.</b>                             | <b>Title</b>                                | <b>Page No.</b> |
|--|---|-----------------|
| 23.5.3   | SWM_SetFixedPinSelect .....                 | 277             |
| <b>Chapter 24 SYSCON: System Configuration</b> |   |                 |
| <b>24.1</b>                                    | <b>Overview .....</b>                       | <b>279</b>      |
| <b>24.2</b>                                    | <b>Macro Definition Documentation .....</b> | <b>279</b>      |
| 24.2.1   | FSL_SYSON_DRIVER_VERSION .....              | 279             |
| <b>24.3</b>                                    | <b>Enumeration Type Documentation .....</b> | <b>280</b>      |
| 24.3.1   | _syscon_connection_t .....                  | 280             |
| <b>24.4</b>                                    | <b>Function Documentation .....</b>         | <b>280</b>      |
| 24.4.1   | SYSCON_AttachSignal .....                   | 280             |
| <b>Chapter 25 WKT: Self-wake-up Timer</b>      |   |                 |
| <b>25.1</b>                                    | <b>Overview .....</b>                       | <b>281</b>      |
| <b>25.2</b>                                    | <b>Function groups .....</b>                | <b>281</b>      |
| 25.2.1   | Initialization and deinitialization .....   | 281             |
| 25.2.2   | Read actual WKT counter value .....         | 281             |
| 25.2.3   | Start and Stop timer operations .....       | 281             |
| 25.2.4   | Status .....                                | 281             |
| <b>25.3</b>                                    | <b>Typical use case .....</b>               | <b>282</b>      |
| 25.3.1   | WKT tick example .....                      | 282             |
| <b>25.4</b>                                    | <b>Data Structure Documentation .....</b>   | <b>283</b>      |
| 25.4.1   | struct_wkt_config .....                     | 283             |
| <b>25.5</b>                                    | <b>Typedef Documentation .....</b>          | <b>283</b>      |
| 25.5.1   | wkt_clock_source_t .....                    | 283             |
| 25.5.2   | wkt_config_t .....                          | 283             |
| <b>25.6</b>                                    | <b>Enumeration Type Documentation .....</b> | <b>283</b>      |
| 25.6.1   | _wkt_clock_source .....                     | 283             |
| 25.6.2   | _wkt_status_flags .....                     | 284             |
| <b>25.7</b>                                    | <b>Function Documentation .....</b>         | <b>284</b>      |
| 25.7.1   | WKT_Init .....                              | 284             |
| 25.7.2   | WKT_Deinit .....                            | 284             |
| 25.7.3   | WKT_GetDefaultConfig .....                  | 284             |
| 25.7.4   | WKT_GetCounterValue .....                   | 285             |
| 25.7.5   | WKT_GetStatusFlags .....                    | 285             |
| 25.7.6   | WKT_ClearStatusFlags .....                  | 285             |

| <b>Section No.</b> | <b>Title</b>         | <b>Page No.</b> |
|--------------------|----------------------|-----------------|
| 25.7.7             | WKT_StartTimer ..... | 285             |
| 25.7.8             | WKT_StopTimer .....  | 286             |

## Chapter 26 WWDT: Windowed Watchdog Timer Driver

|             |   |            |
|-------------|---|------------|
| <b>26.1</b> | <b>Overview .....</b>                       | <b>287</b> |
| <b>26.2</b> | <b>Function groups .....</b>                | <b>287</b> |
| 26.2.1      | Initialization and deinitialization .....   | 287        |
| 26.2.2      | Status .....                                | 287        |
| 26.2.3      | Interrupt .....                             | 287        |
| 26.2.4      | Watch dog Refresh .....                     | 287        |
| <b>26.3</b> | <b>Typical use case .....</b>               | <b>287</b> |
| <b>26.4</b> | <b>Data Structure Documentation .....</b>   | <b>289</b> |
| 26.4.1      | struct_wwdt_config .....                    | 289        |
| <b>26.5</b> | <b>Macro Definition Documentation .....</b> | <b>289</b> |
| 26.5.1      | FSL_WWDT_DRIVER_VERSION .....               | 289        |
| <b>26.6</b> | <b>Typedef Documentation .....</b>          | <b>289</b> |
| 26.6.1      | wwdt_config_t .....                         | 289        |
| <b>26.7</b> | <b>Enumeration Type Documentation .....</b> | <b>289</b> |
| 26.7.1      | _wwdt_status_flags_t .....                  | 289        |
| <b>26.8</b> | <b>Function Documentation .....</b>         | <b>290</b> |
| 26.8.1      | WWDT_GetDefaultConfig .....                 | 290        |
| 26.8.2      | WWDT_Init .....                             | 290        |
| 26.8.3      | WWDT_Deinit .....                           | 290        |
| 26.8.4      | WWDT_Enable .....                           | 291        |
| 26.8.5      | WWDT_Disable .....                          | 291        |
| 26.8.6      | WWDT_GetStatusFlags .....                   | 291        |
| 26.8.7      | WWDT_ClearStatusFlags .....                 | 292        |
| 26.8.8      | WWDT_SetWarningValue .....                  | 292        |
| 26.8.9      | WWDT_SetTimeoutValue .....                  | 292        |
| 26.8.10     | WWDT_SetWindowValue .....                   | 294        |
| 26.8.11     | WWDT_Refresh .....                          | 294        |

## Chapter 27 Debug Console Lite

|             |                              |            |
|-------------|------------------------------|------------|
| <b>27.1</b> | <b>Overview .....</b>        | <b>295</b> |
| <b>27.2</b> | <b>Function groups .....</b> | <b>295</b> |
| 27.2.1      | Initialization .....         | 295        |

| Section No. | Title                                      | Page No.   |
|-------------|--|------------|
| 27.2.2      | Advanced Feature                           | 296        |
| 27.2.3      | SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART | 300        |
| <b>27.3</b> | <b>Typical use case</b>                    | <b>301</b> |
| <b>27.4</b> | <b>Macro Definition Documentation</b>      | <b>303</b> |
| 27.4.1      | DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN         | 303        |
| 27.4.2      | DEBUGCONSOLE_REDIRECT_TO_SDK               | 303        |
| 27.4.3      | DEBUGCONSOLE_DISABLE                       | 303        |
| 27.4.4      | SDK_DEBUGCONSOLE                           | 303        |
| 27.4.5      | PRINTF_FLOAT_ENABLE                        | 303        |
| 27.4.6      | SCANF_FLOAT_ENABLE                         | 303        |
| 27.4.7      | PRINTF_ADVANCED_ENABLE                     | 303        |
| 27.4.8      | SCANF_ADVANCED_ENABLE                      | 303        |
| 27.4.9      | PRINTF                                     | 303        |
| <b>27.5</b> | <b>Function Documentation</b>              | <b>303</b> |
| 27.5.1      | DbgConsole_Init                            | 303        |
| 27.5.2      | DbgConsole_Deinit                          | 304        |
| 27.5.3      | DbgConsole_EnterLowpower                   | 304        |
| 27.5.4      | DbgConsole_ExitLowpower                    | 305        |
| 27.5.5      | DbgConsole_Printf                          | 305        |
| 27.5.6      | DbgConsole_Vprintf                         | 305        |
| 27.5.7      | DbgConsole_Putchar                         | 305        |
| 27.5.8      | DbgConsole_Scanf                           | 306        |
| 27.5.9      | DbgConsole_Getchar                         | 306        |
| <b>27.6</b> | <b>Semihosting</b>                         | <b>307</b> |
| 27.6.1      | Guide Semihosting for IAR                  | 307        |
| 27.6.2      | Guide Semihosting for Keil $\mu$ Vision    | 307        |
| 27.6.3      | Guide Semihosting for MCUXpresso IDE       | 308        |
| 27.6.4      | Guide Semihosting for ARMGCC               | 308        |

# Chapter 1

## Introduction

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

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

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

The peripheral drivers and RTOS driver wrappers can be used across multiple devices within the product family without modification. The configuration items for each driver are encapsulated into C language data structures. Device-specific configuration information is provided as part of the MCUXpresso SDK and need not be modified by the user. If necessary, the user is able to modify the peripheral driver and RTOS wrapper driver configuration during runtime. The driver examples demonstrate how to configure the drivers by passing the proper configuration data to the APIs. The folder structure is organized to reduce the total number of includes required to compile a project.

The rest of this document describes the API references in detail for the peripheral drivers and RTOS wrapper drivers. For the latest version of this and other MCUXpresso SDK documents, see the [mcuxpresso.nxp.com/apidoc/](http://mcuxpresso.nxp.com/apidoc/).

| <b>Deliverable</b>  | <b>Location</b>                                   |
|---|---|
| Demo Applications   | <install_dir>/boards/<board_name>/demo_apps       |
| Driver Examples   | <install_dir>/boards/<board_name>/driver_examples |
| Documentation   | <install_dir>/docs                                |
| Middleware  | <install_dir>/middleware                          |
| Drivers   | <install_dir>/<device_name>/drivers/              |
| CMSIS Standard Arm Cortex-M Headers, math and DSP Libraries | <install_dir>/CMSIS                               |
| Device Startup and Linker                                   | <install_dir>/<device_name>/<toolchain>/          |
| MCUXpresso SDK Utilities                                    | <install_dir>/devices/<device_name>/utilities     |
| RTOS Kernel Code  | <install_dir>/rtos                                |

### **MCUXpresso SDK Folder Structure**

## Chapter 2

# Trademarks

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

How to Reach Us:

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

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

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions).

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TD-MI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2021 NXP B.V.

# Chapter 3

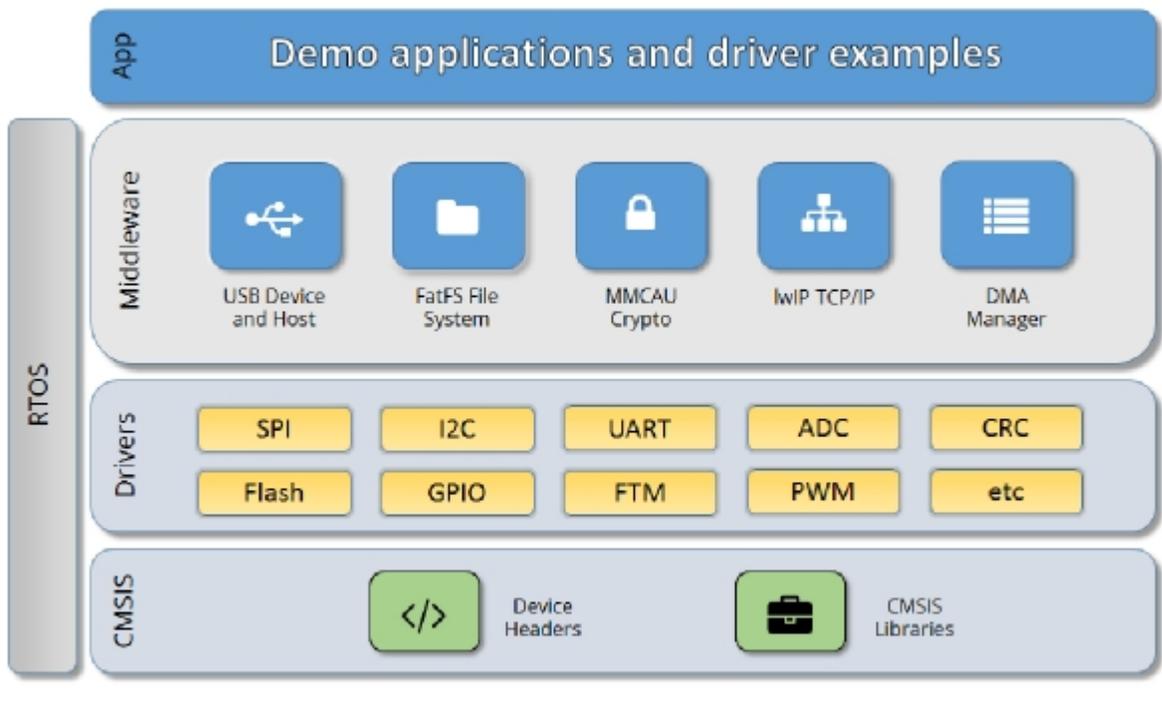
## Architectural Overview

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

### Overview

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

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



**MCUXpresso SDK Block Diagram**

### MCU header files

Each supported MCU device in the MCUXpresso SDK has an overall System-on Chip (SoC) memory-

mapped header file. This header file contains the memory map and register base address for each peripheral and the IRQ vector table with associated vector numbers. The overall SoC header file provides access to the peripheral registers through pointers and predefined bit masks. In addition to the overall SoC memory-mapped header file, the MCUXpresso SDK includes a feature header file for each device. The feature header file allows NXP to deliver a single software driver for a given peripheral. The feature file ensures that the driver is properly compiled for the target SOC.

## CMSIS Support

Along with the SoC header files and peripheral extension header files, the MCUXpresso SDK also includes common CMSIS header files for the Arm Cortex-M core and the math and DSP libraries from the latest CMSIS release. The CMSIS DSP library source code is also included for reference.

## MCUXpresso SDK Peripheral Drivers

The MCUXpresso SDK peripheral drivers mainly consist of low-level functional APIs for the MCU product family on-chip peripherals and also of high-level transactional APIs for some bus drivers/DM-A driver/eDMA driver to quickly enable the peripherals and perform transfers.

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

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

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

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

## Interrupt handling for transactional APIs

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

```
PUBWEAK SPI0_IRQHandler
PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

The first level of the weak implementation are the functions defined in the vector table. In the devices/⟨DEVICE\_NAME⟩/⟨TOOLCHAIN⟩/startup\_⟨DEVICE\_NAME⟩.s/.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0\_DriverIRQHandler) jumps to itself (B). The MCUXpresso SDK drivers with transactional APIs provide the reimplement of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0\_DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

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

The limitation of the double weak mechanism is that it cannot be used for peripherals that share the same vector entry. For this use case, redefine the first layer weak function to enable the desired peripheral interrupt functionality. For example, if the MCU's UART0 and UART1 share the same vector entry, redefine the UART0\_UART1\_IRQHandler according to the use case requirements.

### **Feature Header Files**

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

### **Application**

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

# Chapter 4

## Clock Driver

### 4.1 Overview

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

The clock driver supports:

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

The MCUXpresso SDK provides a peripheral clock driver for the SYSCON module of MCUXpresso SDK devices.

### 4.2 Function description

Clock driver provides these functions:

- Functions to initialize the Core clock to given frequency
- Functions to configure the clock selection muxes.
- Functions to setup peripheral clock dividers
- Functions to set the flash wait states for the input frequency
- Functions to get the frequency of the selected clock
- Functions to set PLL frequency

#### 4.2.1 SYSCON Clock frequency functions

SYSCON clock module provides clocks, such as MCLKCLK, ADCCLK, DMICCLK, MCGFLLCLK, FXCOMCLK, WDTOSC, RTCOSC, USBCLK, and SYSPLL. The functions `CLOCK_EnableClock()` and `CLOCK_DisableClock()` enables and disables the various clocks. `CLOCK_SetupFROClocking()` initializes the FRO to 12 MHz, 48 MHz, or 96 MHz frequency. `CLOCK_SetupPLLData()`, `CLOCK_SetupSystemPLLPrec()`, and `CLOCK_SetPLLFreq()` functions are used to setup the PLL. The SYSCON clock driver provides functions to get the frequency of these clocks, such as `CLOCK_GetFreq()`, `CLOCK_GetFro12MFreq()`, `CLOCK_GetExtClkFreq()`, `CLOCK_GetWdtOscFreq()`, `CLOCK_GetFroHfFreq()`, `CLOCK_GetPllOutFreq()`, `CLOCK_GetOsc32KFreq()`, `CLOCK_GetCoreSysClkFreq()`, `CLOCK_GetI2SMClkFreq()`, `CLOCK_GetFlexCommClkFreq()`, and `CLOCK_GetAsyncApbClkFreq()`.

#### 4.2.2 SYSCON clock Selection Muxes

The SYSCON clock driver provides the function to configure the clock selected. The function `CLOCK_AttachClk()` is implemented for this. The function selects the clock source for a particular peripheral like

MAINCLK, DMIC, FLEXCOMM, USB, ADC, and PLL.

### 4.2.3 SYSCON clock dividers

The SYSCON clock module provides the function to setup the peripheral clock dividers. The function `CLOCK_SetClkDiv()` configures the CLKDIV registers for various peripherals like USB, DMIC, I2S, SYSTICK, AHB, ADC, and also CLKOUT and TRACE functions.

### 4.2.4 SYSCON flash wait states

The SYSCON clock driver provides the function `CLOCK_SetFLASHAccessCyclesForFreq()` that configures FLASHCFG register with a selected FLASHTIM value.

## 4.3 Typical use case

```
POWER_DisablePD(kPDRUNCFG_PD_FRO_EN); /*!< Ensure FRO is on so that we can switch to its 12MHz
```

### Files

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

### Macros

- #define [CLOCK\\_FRO\\_SETTING\\_API\\_ROM\\_ADDRESS](#) (0x0F001CD3U)  
*FRO clock setting API address in ROM.*
- #define [CLOCK\\_FAIM\\_BASE](#) (0x50010000U)  
*FAIM base address.*
- #define [ADC\\_CLOCKS](#)  
*Clock ip name array for ADC.*
- #define [ACMP\\_CLOCKS](#)  
*Clock ip name array for ACMP.*
- #define [DAC\\_CLOCKS](#)  
*Clock ip name array for DAC.*
- #define [SWM\\_CLOCKS](#)  
*Clock ip name array for SWM.*
- #define [ROM\\_CLOCKS](#)  
*Clock ip name array for ROM.*
- #define [SRAM\\_CLOCKS](#)  
*Clock ip name array for SRAM.*
- #define [IOCON\\_CLOCKS](#)  
*Clock ip name array for IOCON.*
- #define [GPIO\\_CLOCKS](#)  
*Clock ip name array for GPIO.*
- #define [GPIO\\_INT\\_CLOCKS](#)  
*Clock ip name array for GPIO\_INT.*
- #define [CRC\\_CLOCKS](#)  
*Clock ip name array for CRC.*

- #define **WWDT\_CLOCKS**  
*Clock ip name array for WWDT.*
- #define **SCT\_CLOCKS**  
*Clock ip name array for SCT0.*
- #define **I2C\_CLOCKS**  
*Clock ip name array for I2C.*
- #define **USART\_CLOCKS**  
*Clock ip name array for I2C.*
- #define **SPI\_CLOCKS**  
*Clock ip name array for SPI.*
- #define **CAPT\_CLOCKS**  
*Clock ip name array for CAPT.*
- #define **CTIMER\_CLOCKS**  
*Clock ip name array for CTIMER.*
- #define **MRT\_CLOCKS**  
*Clock ip name array for MRT.*
- #define **WKT\_CLOCKS**  
*Clock ip name array for WKT.*
- #define **PLU\_CLOCKS**  
*Clock ip name array for PLU.*
- #define **CLK\_GATE\_DEFINE**(reg, bit) (((reg)&0xFFU) << 8U) | ((bit)&0xFFU)  
*Internal used Clock definition only.*

## Typedefs

- typedef enum **\_clock\_ip\_name** **clock\_ip\_name\_t**  
*Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.*
- typedef enum **\_clock\_name** **clock\_name\_t**  
*Clock name used to get clock frequency.*
- typedef enum **\_clock\_select** **clock\_select\_t**  
*Clock Mux Switches CLK\_MUX\_DEFINE(reg, mux) reg is used to define the mux register mux is used to define the mux value.*
- typedef enum **\_clock\_divider** **clock\_divider\_t**  
*Clock divider.*
- typedef enum **\_clock\_fro\_osc\_freq** **clock\_fro\_osc\_freq\_t**  
*fro output frequency source definition*
- typedef enum **\_clock\_main\_clk\_src** **clock\_main\_clk\_src\_t**  
*PLL clock definition.*

## Enumerations

- enum `_clock_ip_name` {
  - `kCLOCK_IpInvalid` = 0U,
  - `kCLOCK_Sys` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 0U),
  - `kCLOCK_Rom` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 1U),
  - `kCLOCK_Ram0` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 2U),
  - `kCLOCK_Flash` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 4U),
  - `kCLOCK_I2c0` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 5U),
  - `kCLOCK_Gpio0` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 6U),
  - `kCLOCK_Swm` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 7U),
  - `kCLOCK_Wkt` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 9U),
  - `kCLOCK_Mrt` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 10U),
  - `kCLOCK_Spi0` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 11U),
  - `kCLOCK_Crc` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 13U),
  - `kCLOCK_Uart0` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 14U),
  - `kCLOCK_Uart1` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 15U),
  - `kCLOCK_Wwdt` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 17U),
  - `kCLOCK_Iocon` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 18U),
  - `kCLOCK_Acmp` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 19U),
  - `kCLOCK_I2c1` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 21U),
  - `kCLOCK_Adc` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 24U),
  - `kCLOCK_Ctimer0` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 25U),
  - `kCLOCK_Dac` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 27U),
  - `kCLOCK_GpioInt` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL0, 28U),
  - `kCLOCK_Capt` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL1, 0U),
  - `kCLOCK_PLU` = CLK\_GATE\_DEFINE(SYS\_AHB\_CLK\_CTRL1, 5U) }

*Clock gate name used for CLOCK\_EnableClock/CLOCK\_DisableClock.*
- enum `_clock_name` {
  - `kCLOCK_CoreSysClk`,
  - `kCLOCK_MainClk`,
  - `kCLOCK_Fro`,
  - `kCLOCK_FroDiv`,
  - `kCLOCK_ExtClk`,
  - `kCLOCK_LPOsc`,
  - `kCLOCK_Frg0` }

*Clock name used to get clock frequency.*
- enum `_clock_select` {

```

kCAPT_Clk_From_Fro = CLK_MUX_DEFINE(CAPTCLKSEL, 0U),
kCAPT_Clk_From_MainClk = CLK_MUX_DEFINE(CAPTCLKSEL, 1U),
kCAPT_Clk_From_Fro_Div = CLK_MUX_DEFINE(CAPTCLKSEL, 3U),
kCAPT_Clk_From_LPOsc = CLK_MUX_DEFINE(CAPTCLKSEL, 4U),
kADC_Clk_From_Fro = CLK_MUX_DEFINE(ADCCLKSEL, 0U),
kADC_Clk_From_Extclk = CLK_MUX_DEFINE(ADCCLKSEL, 1U),
kUART0_Clk_From_Fro = CLK_MUX_DEFINE(UART0CLKSEL, 0U),
kUART0_Clk_From_MainClk = CLK_MUX_DEFINE(UART0CLKSEL, 1U),
kUART0_Clk_From_Frg0Clk = CLK_MUX_DEFINE(UART0CLKSEL, 2U),
kUART0_Clk_From_Fro_Div = CLK_MUX_DEFINE(UART0CLKSEL, 4U),
kUART1_Clk_From_Fro = CLK_MUX_DEFINE(UART1CLKSEL, 0U),
kUART1_Clk_From_MainClk = CLK_MUX_DEFINE(UART1CLKSEL, 1U),
kUART1_Clk_From_Frg0Clk = CLK_MUX_DEFINE(UART1CLKSEL, 2U),
kUART1_Clk_From_Fro_Div = CLK_MUX_DEFINE(UART1CLKSEL, 4U),
kI2C0_Clk_From_Fro = CLK_MUX_DEFINE(I2C0CLKSEL, 0U),
kI2C0_Clk_From_MainClk = CLK_MUX_DEFINE(I2C0CLKSEL, 1U),
kI2C0_Clk_From_Frg0Clk = CLK_MUX_DEFINE(I2C0CLKSEL, 2U),
kI2C0_Clk_From_Fro_Div = CLK_MUX_DEFINE(I2C0CLKSEL, 4U),
kI2C1_Clk_From_Fro = CLK_MUX_DEFINE(I2C1CLKSEL, 0U),
kI2C1_Clk_From_MainClk = CLK_MUX_DEFINE(I2C1CLKSEL, 1U),
kI2C1_Clk_From_Frg0Clk = CLK_MUX_DEFINE(I2C1CLKSEL, 2U),
kI2C1_Clk_From_Fro_Div = CLK_MUX_DEFINE(I2C1CLKSEL, 4U),
kSPI0_Clk_From_Fro = CLK_MUX_DEFINE(SPI0CLKSEL, 0U),
kSPI0_Clk_From_MainClk = CLK_MUX_DEFINE(SPI0CLKSEL, 1U),
kSPI0_Clk_From_Frg0Clk = CLK_MUX_DEFINE(SPI0CLKSEL, 2U),
kSPI0_Clk_From_Fro_Div = CLK_MUX_DEFINE(SPI0CLKSEL, 4U),
kFRG0_Clk_From_Fro = CLK_MUX_DEFINE(FRG[0].FRGCLKSEL, 0U),
kFRG0_Clk_From_MainClk = CLK_MUX_DEFINE(FRG[0].FRGCLKSEL, 1U),
kCLKOUT_From_Fro = CLK_MUX_DEFINE(CLKOUTSEL, 0U),
kCLKOUT_From_MainClk = CLK_MUX_DEFINE(CLKOUTSEL, 1U),
kCLKOUT_From_ExtClk = CLK_MUX_DEFINE(CLKOUTSEL, 3U),
kCLKOUT_From_Lposc = CLK_MUX_DEFINE(CLKOUTSEL, 4U) }

```

*Clock Mux Switches CLK\_MUX\_DEFINE(reg, mux) reg is used to define the mux register mux is used to define the mux value.*

- enum `_clock_divider` {
 

```

kCLOCK_DivAhbClk = CLK_DIV_DEFINE(SYSAHBCLKDIV),
kCLOCK_DivAdcClk = CLK_DIV_DEFINE(ADCCLKDIV),
kCLOCK_DivClkOut = CLK_DIV_DEFINE(CLKOUTDIV) }

```

*Clock divider.*

- enum `_clock_fro_osc_freq` {
 

```

kCLOCK_FroOscOut18M = 18000U,
kCLOCK_FroOscOut24M = 24000U,
kCLOCK_FroOscOut30M = 30000U }

```

*fro output frequency source definition*

- enum `_clock_main_clk_src` {

```
kCLOCK_MainClkSrcFro = CLK_MAIN_CLK_MUX_DEFINE(0U, 0U),
kCLOCK_MainClkSrcExtClk = CLK_MAIN_CLK_MUX_DEFINE(1U, 0U),
kCLOCK_MainClkSrcLPOsc = CLK_MAIN_CLK_MUX_DEFINE(2U, 0U),
kCLOCK_MainClkSrcFroDiv = CLK_MAIN_CLK_MUX_DEFINE(3U, 0U) }
    PLL clock definition.
```

## Variables

- volatile uint32\_t `g_LP_Osc_Freq`  
*lower power oscillator clock frequency.*
- volatile uint32\_t `g_Ext_Clk_Freq`  
*external clock frequency.*
- volatile uint32\_t `g_Fro_Osc_Freq`  
*external clock frequency.*

## Driver version

- #define `FSL_CLOCK_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 3)`)  
*CLOCK driver version 2.3.3.*

## Clock gate, mux, and divider.

PLL configuration structure

- static void `CLOCK_EnableClock` (`clock_ip_name_t` clk)
- static void `CLOCK_DisableClock` (`clock_ip_name_t` clk)
- static void `CLOCK_Select` (`clock_select_t` sel)
- static void `CLOCK_SetClkDivider` (`clock_divider_t` name, uint32\_t value)
- static uint32\_t `CLOCK_GetClkDivider` (`clock_divider_t` name)
- static void `CLOCK_SetCoreSysClkDiv` (uint32\_t value)
- void `CLOCK_SetMainClkSrc` (`clock_main_clk_src_t` src)  
*Set main clock reference source.*
- static void `CLOCK_SetFRGClkMul` (uint32\_t \*base, uint32\_t mul)

## Get frequency

Set FRO clock source

Parameters

|                    |   |
|--------------------|---|
| <i>src, please</i> | reference <code>_clock_fro_src</code> definition. |
|--------------------|---|

- uint32\_t `CLOCK_GetFRG0ClkFreq` (void)  
*Return Frequency of FRG0 Clock.*
- uint32\_t `CLOCK_GetMainClkFreq` (void)  
*Return Frequency of Main Clock.*
- uint32\_t `CLOCK_GetFroFreq` (void)  
*Return Frequency of FRO.*
- static uint32\_t `CLOCK_GetCoreSysClkFreq` (void)  
*Return Frequency of core.*
- uint32\_t `CLOCK_GetClockOutClkFreq` (void)

- `uint32_t` `CLOCK_GetUart0ClkFreq` (void)  
*Get UART0 frequency.*
- `uint32_t` `CLOCK_GetUart1ClkFreq` (void)  
*Get UART1 frequency.*
- `uint32_t` `CLOCK_GetFreq` (`clock_name_t` clockName)  
*Return Frequency of selected clock.*
- `static uint32_t` `CLOCK_GetLPOscFreq` (void)  
*Get watch dog OSC frequency.*
- `static uint32_t` `CLOCK_GetExtClkFreq` (void)  
*Get external clock frequency.*

## Fractional clock operations

System PLL initialize.

Parameters

|               |                            |
|---------------|----------------------------|
| <i>config</i> | System PLL configurations. |
|---------------|----------------------------|

- `bool` `CLOCK_SetFRG0ClkFreq` (`uint32_t` freq)  
*Set FRG0 output frequency.*

## External/internal oscillator clock operations

- `void` `CLOCK_InitExtClkin` (`uint32_t` clkInFreq)  
*Init external CLK IN, select the CLKIN as the external clock source.*
- `static void` `CLOCK_DeinitLpOsc` (void)  
*Deinit watch dog OSC.*
- `void` `CLOCK_SetFroOscFreq` (`clock_fro_osc_freq_t` freq)  
*Set FRO oscillator output frequency.*

## 4.4 Macro Definition Documentation

4.4.1 `#define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 3, 3))`

4.4.2 `#define CLOCK_FRO_SETTING_API_ROM_ADDRESS (0x0F001CD3U)`

4.4.3 `#define ADC_CLOCKS`

Value:

```
{
    \
    kCLOCK_Adc, \
}
```

#### 4.4.4 #define ACMP\_CLOCKS

Value:

```
{  
    kCLOCK_Acmp, \  
}
```

#### 4.4.5 #define DAC\_CLOCKS

Value:

```
{  
    kCLOCK_Dac, \  
}
```

#### 4.4.6 #define SWM\_CLOCKS

Value:

```
{  
    kCLOCK_Swm, \  
}
```

#### 4.4.7 #define ROM\_CLOCKS

Value:

```
{  
    kCLOCK_Rom, \  
}
```

#### 4.4.8 #define SRAM\_CLOCKS

Value:

```
{  
    kCLOCK_Ram0, \  
}
```

#### 4.4.9 #define IOCON\_CLOCKS

Value:

```
{  
    \kLOCK_Iocon, \  
}
```

#### 4.4.10 #define GPIO\_CLOCKS

Value:

```
{  
    \kLOCK_Gpio0, \  
}
```

#### 4.4.11 #define GPIO\_INT\_CLOCKS

Value:

```
{  
    \kLOCK_GpioInt, \  
}
```

#### 4.4.12 #define CRC\_CLOCKS

Value:

```
{  
    \kLOCK_Crc, \  
}
```

#### 4.4.13 #define WWDT\_CLOCKS

Value:

```
{  
    \kLOCK_Wwdt, \  
}
```

#### 4.4.14 #define SCT\_CLOCKS

Value:

```
{
    kCLOCK_Sct, \
}
```

#### 4.4.15 #define I2C\_CLOCKS

Value:

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

#### 4.4.16 #define USART\_CLOCKS

Value:

```
{
    kCLOCK_Uart0, kCLOCK_Uart1, \
}
```

#### 4.4.17 #define SPI\_CLOCKS

Value:

```
{
    kCLOCK_Spi0, \
}
```

#### 4.4.18 #define CAPT\_CLOCKS

Value:

```
{
    kCLOCK_Capt, \
}
```

#### 4.4.19 #define CTIMER\_CLOCKS

Value:

```
{  
    kCLOCK_Ctimer0, \  
}
```

#### 4.4.20 #define MRT\_CLOCKS

Value:

```
{  
    kCLOCK_Mrt, \  
}
```

#### 4.4.21 #define WKT\_CLOCKS

Value:

```
{  
    kCLOCK_Wkt, \  
}
```

#### 4.4.22 #define PLU\_CLOCKS

Value:

```
{  
    kCLOCK_PLU, \  
}
```

#### 4.4.23 #define CLK\_GATE\_DEFINE( *reg*, *bit* ) (((*reg*)&0xFFU) << 8U) | ((*bit*)&0xFFU)

## 4.5 Typedef Documentation

### 4.5.1 typedef enum \_clock\_ip\_name clock\_ip\_name\_t

### 4.5.2 typedef enum \_clock\_name clock\_name\_t

### 4.5.3 typedef enum \_clock\_select clock\_select\_t

### 4.5.4 typedef enum \_clock\_fro\_osc\_freq clock\_fro\_osc\_freq\_t

fro oscillator output frequency value definition

### 4.5.5 typedef enum \_clock\_main\_clk\_src clock\_main\_clk\_src\_t

< Main clock source definition

## 4.6 Enumeration Type Documentation

### 4.6.1 enum \_clock\_ip\_name

Enumerator

*kCLOCK\_IpInvalid* Invalid Ip Name.  
*kCLOCK\_Sys* Clock gate name: Sys.  
*kCLOCK\_Rom* Clock gate name: Rom.  
*kCLOCK\_Ram0* Clock gate name: Ram0.  
*kCLOCK\_Flash* Clock gate name: Flash.  
*kCLOCK\_I2c0* Clock gate name: I2c0.  
*kCLOCK\_Gpio0* Clock gate name: Gpio0.  
*kCLOCK\_Swm* Clock gate name: Swm.  
*kCLOCK\_Wkt* Clock gate name: Wkt.  
*kCLOCK\_Mrt* Clock gate name: Mrt.  
*kCLOCK\_Spi0* Clock gate name: Spi0.  
*kCLOCK\_Crc* Clock gate name: Crc.  
*kCLOCK\_Uart0* Clock gate name: Uart0.  
*kCLOCK\_Uart1* Clock gate name: Uart1.  
*kCLOCK\_Wwdt* Clock gate name: Wwdt.  
*kCLOCK\_Iocon* Clock gate name: Iocon.  
*kCLOCK\_Acmp* Clock gate name: Acmp.  
*kCLOCK\_I2c1* Clock gate name: I2c1.  
*kCLOCK\_Adc* Clock gate name: Adc.  
*kCLOCK\_Ctimer0* Clock gate name: Ctimer0.  
*kCLOCK\_Dac* Clock gate name: Dac.  
*kCLOCK\_GpioInt* Clock gate name: GpioInt.

*kCLOCK\_Capt* Clock gate name: Capt.

*kCLOCK\_PLU* Clock gate name: PLU.

#### 4.6.2 enum\_clock\_name

Enumerator

*kCLOCK\_CoreSysClk* Cpu/AHB/AHB matrix/Memories,etc.

*kCLOCK\_MainClk* Main clock.

*kCLOCK\_Fro* FRO18/24/30.

*kCLOCK\_FroDiv* FRO div clock.

*kCLOCK\_ExtClk* External Clock.

*kCLOCK\_LPOsc* Watchdog Oscillator.

*kCLOCK\_Frg0* fractional rate0

#### 4.6.3 enum\_clock\_select

Enumerator

*kCAPT\_Clk\_From\_Fro* Mux CAPT\_Clk from Fro.

*kCAPT\_Clk\_From\_MainClk* Mux CAPT\_Clk from MainClk.

*kCAPT\_Clk\_From\_Fro\_Div* Mux CAPT\_Clk from Fro\_Div.

*kCAPT\_Clk\_From\_LPOsc* Mux CAPT\_Clk from LPOsc.

*kADC\_Clk\_From\_Fro* Mux ADC\_Clk from Fro.

*kADC\_Clk\_From\_Extclk* Mux ADC\_Clk from Extclk.

*kUART0\_Clk\_From\_Fro* Mux UART0\_Clk from Fro.

*kUART0\_Clk\_From\_MainClk* Mux UART0\_Clk from MainClk.

*kUART0\_Clk\_From\_Frg0Clk* Mux UART0\_Clk from Frg0Clk.

*kUART0\_Clk\_From\_Fro\_Div* Mux UART0\_Clk from Fro\_Div.

*kUART1\_Clk\_From\_Fro* Mux UART1\_Clk from Fro.

*kUART1\_Clk\_From\_MainClk* Mux UART1\_Clk from MainClk.

*kUART1\_Clk\_From\_Frg0Clk* Mux UART1\_Clk from Frg0Clk.

*kUART1\_Clk\_From\_Fro\_Div* Mux UART1\_Clk from Fro\_Div.

*kI2C0\_Clk\_From\_Fro* Mux I2C0\_Clk from Fro.

*kI2C0\_Clk\_From\_MainClk* Mux I2C0\_Clk from MainClk.

*kI2C0\_Clk\_From\_Frg0Clk* Mux I2C0\_Clk from Frg0Clk.

*kI2C0\_Clk\_From\_Fro\_Div* Mux I2C0\_Clk from Fro\_Div.

*kI2C1\_Clk\_From\_Fro* Mux I2C1\_Clk from Fro.

*kI2C1\_Clk\_From\_MainClk* Mux I2C1\_Clk from MainClk.

*kI2C1\_Clk\_From\_Frg0Clk* Mux I2C1\_Clk from Frg0Clk.

*kI2C1\_Clk\_From\_Fro\_Div* Mux I2C1\_Clk from Fro\_Div.

*kSPI0\_Clk\_From\_Fro* Mux SPI0\_Clk from Fro.

*kSPI0\_Clk\_From\_MainClk* Mux SPI0\_Clk from MainClk.

*kSPI0\_Clk\_From\_Frg0Clk* Mux SPI0\_Clk from Frg0Clk.  
*kSPI0\_Clk\_From\_Fro\_Div* Mux SPI0\_Clk from Fro\_Div.  
*kFRG0\_Clk\_From\_Fro* Mux FRG0\_Clk from Fro.  
*kFRG0\_Clk\_From\_MainClk* Mux FRG0\_Clk from MainClk.  
*kCLKOUT\_From\_Fro* Mux CLKOUT from Fro.  
*kCLKOUT\_From\_MainClk* Mux CLKOUT from MainClk.  
*kCLKOUT\_From\_ExtClk* Mux CLKOUT from ExtClk.  
*kCLKOUT\_From\_Lposc* Mux CLKOUT from Lposc.

#### 4.6.4 enum \_clock\_divider

Enumerator

*kCLOCK\_DivAhbClk* Ahb Clock Divider.  
*kCLOCK\_DivAdcClk* Adc Clock Divider.  
*kCLOCK\_DivClkOut* Clk Out Divider.

#### 4.6.5 enum \_clock\_fro\_osc\_freq

fro oscillator output frequency value definition

Enumerator

*kCLOCK\_FroOscOut18M* FRO oscillator output 18M.  
*kCLOCK\_FroOscOut24M* FRO oscillator output 24M.  
*kCLOCK\_FroOscOut30M* FRO oscillator output 30M.

#### 4.6.6 enum \_clock\_main\_clk\_src

< Main clock source definition

Enumerator

*kCLOCK\_MainClkSrcFro* main clock source from FRO  
*kCLOCK\_MainClkSrcExtClk* main clock source from Ext clock  
*kCLOCK\_MainClkSrcLPOsc* main clock source from lower power oscillator  
*kCLOCK\_MainClkSrcFroDiv* main clock source from FRO Div

### 4.7 Function Documentation

#### 4.7.1 void CLOCK\_SetMainClkSrc ( clock\_main\_clk\_src\_t src )

## Parameters

|            |   |
|------------|---|
| <i>src</i> | Reference <code>clock_main_clk_src_t</code> to set the main clock source. |
|------------|---|

**4.7.2 uint32\_t CLOCK\_GetFRG0ClkFreq ( void )**

## Returns

Frequency of FRG0 Clock.

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

## Returns

Frequency of Main Clock.

**4.7.4 uint32\_t CLOCK\_GetFroFreq ( void )**

## Returns

Frequency of FRO.

**4.7.5 static uint32\_t CLOCK\_GetCoreSysClkFreq ( void ) [inline], [static]**

## Returns

Frequency of core.

**4.7.6 uint32\_t CLOCK\_GetClockOutClkFreq ( void )**

## Returns

Frequency of ClockOut

**4.7.7 uint32\_t CLOCK\_GetUart0ClkFreq ( void )**

Return values

|              |                  |
|--------------|------------------|
| <i>UART0</i> | frequency value. |
|--------------|------------------|

#### 4.7.8 `uint32_t CLOCK_GetUart1ClkFreq ( void )`

Return values

|              |                  |
|--------------|------------------|
| <i>UART1</i> | frequency value. |
|--------------|------------------|

#### 4.7.9 `uint32_t CLOCK_GetFreq ( clock_name_t clockName )`

Returns

Frequency of selected clock

#### 4.7.10 `static uint32_t CLOCK_GetLPOscFreq ( void ) [inline], [static]`

Return values

|              |                          |
|--------------|--------------------------|
| <i>watch</i> | dog OSC frequency value. |
|--------------|--------------------------|

#### 4.7.11 `static uint32_t CLOCK_GetExtClkFreq ( void ) [inline], [static]`

Return values

|                 |                        |
|-----------------|------------------------|
| <i>external</i> | clock frequency value. |
|-----------------|------------------------|

#### 4.7.12 `bool CLOCK_SetFRG0ClkFreq ( uint32_t freq )`

Parameters

---

|             |   |
|-------------|---|
| <i>freq</i> | target output frequency, $freq < input$ and $(input / freq) < 2$ should be satisfy. |
|-------------|---|

Return values

|             |   |
|-------------|---|
| <i>true</i> | - successfully, <i>false</i> - input argument is invalid. |
|-------------|---|

#### 4.7.13 void CLOCK\_InitExtClkin ( uint32\_t *clkInFreq* )

Parameters

|                  |                              |
|------------------|------------------------------|
| <i>clkInFreq</i> | external clock in frequency. |
|------------------|------------------------------|

#### 4.7.14 void CLOCK\_SetFroOscFreq ( clock\_fro\_osc\_freq\_t *freq* )

Initialize the FRO clock to given frequency (18, 24 or 30 MHz).

Parameters

|             |   |
|-------------|---|
| <i>freq</i> | Please refer to definition of <code>clock_fro_osc_freq_t</code> , frequency must be one of 18000, 24000 or 30000 KHz. |
|-------------|---|

## 4.8 Variable Documentation

### 4.8.1 volatile uint32\_t g\_LP\_Osc\_Freq

This variable is used to store the lower power oscillator frequency which is set by `CLOCK_InitLPOsc`, and it is returned by `CLOCK_GetLPOscFreq`.

### 4.8.2 volatile uint32\_t g\_Ext\_Clk\_Freq

This variable is used to store the external clock frequency which is include external oscillator clock and external clk in clock frequency value, it is set by `CLOCK_InitExtClkin` when CLK IN is used as external clock or by `CLOCK_InitSysOsc` when external oscillator is used as external clock ,and it is returned by `CLOCK_GetExtClkFreq`.

### 4.8.3 volatile uint32\_t g\_Fro\_Osc\_Freq

This variable is used to store the FRO osc clock frequency.

# Chapter 5

## Power Driver

### 5.1 Overview

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

#### Macros

- #define `PMUC_PCON_RESERVED_MASK`  $((0xf \ll 4) | (0x6 \ll 8) | 0xffff000u)$   
*PMU PCON reserved mask, used to clear reserved field which should not write 1.*

#### Typedefs

- typedef enum `_power_gen_reg` `power_gen_reg_t`  
*pmu general purpose register index*
- typedef enum `_power_bod_reset_level` `power_bod_reset_level_t`  
*BOD reset level, if VDD below reset level value, the reset will be asserted.*
- typedef enum `_power_bod_interrupt_level` `power_bod_interrupt_level_t`  
*BOD interrupt level, if VDD below interrupt level value, the BOD interrupt will be asserted.*

#### Enumerations

- enum `_power_wakeup`  
*Deep sleep and power down mode wake up configurations.*
- enum `_power_dpd_wakeup_pin`  
*Deep power down mode wake up pins.*
- enum `_power_deep_sleep_active`  
*Deep sleep/power down mode active part.*
- enum `_power_gen_reg` {  
    `kPmu_GenReg0` = 0U,  
    `kPmu_GenReg1` = 1U,  
    `kPmu_GenReg2` = 2U,  
    `kPmu_GenReg3` = 3U,  
    `kPmu_GenReg4` = 4U }  
*pmu general purpose register index*
- enum `_power_bod_reset_level` { `kBod_ResetLevel0` = 0U }  
*BOD reset level, if VDD below reset level value, the reset will be asserted.*
- enum `_power_bod_interrupt_level` {  
    `kBod_InterruptLevelReserved` = 0U,  
    `kBod_InterruptLevel1`,  
    `kBod_InterruptLevel2`,  
    `kBod_InterruptLevel3` }  
*BOD interrupt level, if VDD below interrupt level value, the BOD interrupt will be asserted.*

## Driver version

- #define `FSL_POWER_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 0)`)  
*power driver version 2.1.0.*

## SYSCON Power Configuration

- static void `POWER_EnablePD` (`pd_bit_t en`)  
*API to enable PDRUNCFG bit in the Syscon.*
- static void `POWER_DisablePD` (`pd_bit_t en`)  
*API to disable PDRUNCFG bit in the Syscon.*

## ARM core Power Configuration

- static void `POWER_EnableDeepSleep` (`void`)  
*API to enable deep sleep bit in the ARM Core.*
- static void `POWER_DisableDeepSleep` (`void`)  
*API to disable deep sleep bit in the ARM Core.*

## PMU functionality

- void `POWER_EnterSleep` (`void`)  
*API to enter sleep power mode.*
- void `POWER_EnterDeepSleep` (`uint32_t activePart`)  
*API to enter deep sleep power mode.*
- void `POWER_EnterPowerDown` (`uint32_t activePart`)  
*API to enter power down mode.*
- void `POWER_EnterDeepPowerDownMode` (`void`)  
*API to enter deep power down mode.*
- static `uint32_t POWER_GetSleepModeFlag` (`void`)  
*API to get sleep mode flag.*
- static void `POWER_ClrSleepModeFlag` (`void`)  
*API to clear sleep mode flag.*
- static `uint32_t POWER_GetDeepPowerDownModeFlag` (`void`)  
*API to get deep power down mode flag.*
- static void `POWER_ClrDeepPowerDownModeFlag` (`void`)  
*API to clear deep power down mode flag.*
- static void `POWER_ClrWakeupPinFlag` (`void`)  
*API to clear wake up pin status flag.*
- static void `POWER_EnableNonDpd` (`bool enable`)  
*API to enable non deep power down mode.*
- static void `POWER_EnableLPO` (`bool enable`)  
*API to enable LPO.*
- static void `POWER_WakeUpConfig` (`uint32_t mask`, `bool powerDown`)  
*API to config wakeup configurations for deep sleep mode and power down mode.*
- static void `POWER_DeepSleepConfig` (`uint32_t mask`, `bool powerDown`)  
*API to config active part for deep sleep mode and power down mode.*

## API to enable wake up pin for deep power down mode

### Parameters

|                   |  |
|-------------------|--|
| <i>wakeup_pin</i> | wake up pin for which to enable.reference <code>_power_dpd_wakeup_pin</code> . |
|-------------------|--|

### Returns

none

- static void **POWER\_DeepPowerDownWakeupSourceSelect** (uint32\_t wakeup\_pin)
- static void **POWER\_SetRetainData** (power\_gen\_reg\_t index, uint32\_t data)  
*API to restore data to general purpose register which can be retained during deep power down mode.*
- static uint32\_t **POWER\_GetRetainData** (power\_gen\_reg\_t index)  
*API to get data from general purpose register which is retained during deep power down mode.*
- static void **POWER\_SetBodLevel** (power\_bod\_reset\_level\_t resetLevel, power\_bod\_interrupt\_level\_t interruptLevel, bool enable)  
*Set Bod interrupt level and reset level.*

## 5.2 Macro Definition Documentation

### 5.2.1 #define FSL\_POWER\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

## 5.3 Enumeration Type Documentation

### 5.3.1 enum \_power\_gen\_reg

#### Enumerator

- kPmu\_GenReg0* general purpose register0
- kPmu\_GenReg1* general purpose register1
- kPmu\_GenReg2* general purpose register2
- kPmu\_GenReg3* general purpose register3
- kPmu\_GenReg4* general purpose register4

### 5.3.2 enum \_power\_bod\_reset\_level

#### Enumerator

- kBod\_ResetLevel0* BOD Reset Level0: 1.51V.

### 5.3.3 enum \_power\_bod\_interrupt\_level

#### Enumerator

- kBod\_InterruptLevelReserved* BOD interrupt level reserved.
- kBod\_InterruptLevel1* BOD interrupt level1: 2.24V.
- kBod\_InterruptLevel2* BOD interrupt level2: 2.52V.
- kBod\_InterruptLevel3* BOD interrupt level3: 2.81V.

## 5.4 Function Documentation

### 5.4.1 `static void POWER_EnablePD ( pd_bit_t en ) [inline], [static]`

Note that enabling the bit powers down the peripheral

Parameters

|           |   |
|-----------|---|
| <i>en</i> | peripheral for which to enable the PDRUNCFG bit |
|-----------|---|

Returns

none

### 5.4.2 `static void POWER_DisablePD ( pd_bit_t en ) [inline], [static]`

Note that disabling the bit powers up the peripheral

Parameters

|           |  |
|-----------|--|
| <i>en</i> | peripheral for which to disable the PDRUNCFG bit |
|-----------|--|

Returns

none

### 5.4.3 `static void POWER_EnableDeepSleep ( void ) [inline], [static]`

Returns

none

### 5.4.4 `static void POWER_DisableDeepSleep ( void ) [inline], [static]`

Returns

none

#### 5.4.5 void POWER\_EnterSleep ( void )

Returns

none

#### 5.4.6 void POWER\_EnterDeepSleep ( uint32\_t *activePart* )

Parameters

|                     |  |
|---------------------|--|
| <i>activePart,:</i> | should be a single or combine value of <code>_power_deep_sleep_active</code> . |
|---------------------|--|

Returns

none

#### 5.4.7 void POWER\_EnterPowerDown ( uint32\_t *activePart* )

Parameters

|                     |  |
|---------------------|--|
| <i>activePart,:</i> | should be a single or combine value of <code>_power_deep_sleep_active</code> . |
|---------------------|--|

Returns

none

#### 5.4.8 void POWER\_EnterDeepPowerDownMode ( void )

Returns

none

#### 5.4.9 static uint32\_t POWER\_GetSleepModeFlag ( void ) [inline], [static]

Returns

sleep mode flag: 0 is active mode, 1 is sleep mode entered.

#### 5.4.10 `static uint32_t POWER_GetDeepPowerDownModeFlag ( void ) [inline], [static]`

Returns

sleep mode flag: 0 not deep power down, 1 is deep power down mode entered.

#### 5.4.11 `static void POWER_EnableNonDpd ( bool enable ) [inline], [static]`

Parameters

|                  |  |
|------------------|--|
| <i>enable</i> ,: | true is enable non deep power down, otherwise disable. |
|------------------|--|

#### 5.4.12 `static void POWER_EnableLPO ( bool enable ) [inline], [static]`

Parameters

|                  |   |
|------------------|---|
| <i>enable</i> ,: | true to enable LPO, false to disable LPO. |
|------------------|---|

#### 5.4.13 `static void POWER_WakeUpConfig ( uint32_t mask, bool powerDown ) [inline], [static]`

Parameters

|                     |  |
|---------------------|--|
| <i>mask</i> ,:      | wake up configurations for deep sleep mode and power down mode, reference <code>_power_wakeup</code> . |
| <i>powerDown</i> ,: | true is power down the mask part, false is powered part.   |

#### 5.4.14 `static void POWER_DeepSleepConfig ( uint32_t mask, bool powerDown ) [inline], [static]`

Parameters

---

|                    |   |
|--------------------|---|
| <i>mask,:</i>      | active part configurations for deep sleep mode and power down mode, reference <code>_power_deep_sleep_active</code> . |
| <i>powerDown,:</i> | true is power down the mask part, false is powered part.  |

#### 5.4.15 `static void POWER_SetRetainData ( power_gen_reg_t index, uint32_t data )` **[inline], [static]**

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>index,:</i> | general purpose data register index. |
| <i>data,:</i>  | data to restore.                     |

#### 5.4.16 `static uint32_t POWER_GetRetainData ( power_gen_reg_t index )` **[inline], [static]**

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>index,:</i> | general purpose data register index. |
|----------------|--------------------------------------|

Returns

data stored in the general purpose register.

#### 5.4.17 `static void POWER_SetBodLevel ( power_bod_reset_level_t resetLevel, power_bod_interrupt_level_t interruptLevel, bool enable )` **[inline], [static]**

Parameters

|                       |  |
|-----------------------|--|
| <i>resetLevel</i>     | BOD reset threshold level, please refer to <a href="#">power_bod_reset_level_t</a> .         |
| <i>interruptLevel</i> | BOD interrupt threshold level, please refer to <a href="#">power_bod_interrupt_level_t</a> . |

|               |   |
|---------------|---|
| <i>enable</i> | Used to enable/disable the BOD interrupt and BOD reset. |
|---------------|---|

# Chapter 6

## Reset Driver

### 6.1 Overview

Reset driver supports peripheral reset and system reset.

#### Macros

- #define `FLASH_RSTS_N`

#### Typedefs

- typedef enum `_SYSCON_RSTn SYSCON_RSTn_t`  
*Enumeration for peripheral reset control bits.*

#### Enumerations

- enum `_SYSCON_RSTn` {  
    `kFLASH_RST_N_SHIFT_RSTn = 0 | 4U,`  
    `kI2C0_RST_N_SHIFT_RSTn = 0 | 5U,`  
    `kGPIO0_RST_N_SHIFT_RSTn = 0 | 6U,`  
    `kSWM_RST_N_SHIFT_RSTn = 0 | 7U,`  
    `kWKT_RST_N_SHIFT_RSTn = 0 | 9U,`  
    `kMRT_RST_N_SHIFT_RSTn = 0 | 10U,`  
    `kSPI0_RST_N_SHIFT_RSTn = 0 | 11U,`  
    `kCRC_RST_SHIFT_RSTn = 0 | 13U,`  
    `kUART0_RST_N_SHIFT_RSTn = 0 | 14U,`  
    `kUART1_RST_N_SHIFT_RSTn = 0 | 15U,`  
    `kIOCON_RST_N_SHIFT_RSTn = 0 | 18U,`  
    `kACMP_RST_N_SHIFT_RSTn = 0 | 19U,`  
    `kI2C1_RST_N_SHIFT_RSTn = 0 | 21U,`  
    `kADC_RST_N_SHIFT_RSTn = 0 | 24U,`  
    `kTIMER0_RST_N_SHIFT_RSTn = 0 | 25U,`  
    `kDAC0_RST_N_SHIFT_RSTn = 0 | 27U,`  
    `kGPIOINT_RST_N_SHIFT_RSTn = 0 | 28U,`  
    `kCAPT_RST_N_SHIFT_RSTn = 65536 | 0U,`  
    `kFRG0_RST_N_SHIFT_RSTn = 65536 | 3U,`  
    `kPLU_RST_N_SHIFT_RSTn = 65536 | 5U }`  
*Enumeration for peripheral reset control bits.*

#### Functions

- void `RESET_SetPeripheralReset` (reset\_ip\_name\_t peripheral)

- void `RESET_ClearPeripheralReset` (reset\_ip\_name\_t peripheral)  
*Assert reset to peripheral.*
- void `RESET_PeripheralReset` (reset\_ip\_name\_t peripheral)  
*Clear reset to peripheral.*
- static void `RESET_ReleasePeripheralReset` (reset\_ip\_name\_t peripheral)  
*Reset peripheral module.*
- static void `RESET_ReleasePeripheralReset` (reset\_ip\_name\_t peripheral)  
*Release peripheral module.*

### Driver version

- #define `FSL_RESET_DRIVER_VERSION` (`MAKE_VERSION`(2, 4, 0))  
*reset driver version 2.4.0*

## 6.2 Macro Definition Documentation

### 6.2.1 #define FLASH\_RSTS\_N

Value:

```
{
    kFLASH_RST_N_SHIFT_RSTn \
} /* Reset bits for Flash peripheral */
```

Array initializers with peripheral reset bits

## 6.3 Typedef Documentation

### 6.3.1 typedef enum \_SYSCON\_RSTn SYSCON\_RSTn\_t

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

## 6.4 Enumeration Type Documentation

### 6.4.1 enum \_SYSCON\_RSTn

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

Enumerator

*kFLASH\_RST\_N\_SHIFT\_RSTn* Flash controller reset control  
*kI2C0\_RST\_N\_SHIFT\_RSTn* I2C0 reset control  
*kGPIO0\_RST\_N\_SHIFT\_RSTn* GPIO0 reset control  
*kSWM\_RST\_N\_SHIFT\_RSTn* SWM reset control  
*kWKT\_RST\_N\_SHIFT\_RSTn* Self-wake-up timer(WKT) reset control  
*kMRT\_RST\_N\_SHIFT\_RSTn* Multi-rate timer(MRT) reset control

***kSPI0\_RST\_N\_SHIFT\_RSTn*** SPI0 reset control.  
***kCRC\_RST\_SHIFT\_RSTn*** CRC reset control  
***kUART0\_RST\_N\_SHIFT\_RSTn*** UART0 reset control  
***kUART1\_RST\_N\_SHIFT\_RSTn*** UART1 reset control  
***kIOCON\_RST\_N\_SHIFT\_RSTn*** IOCON reset control  
***kACMP\_RST\_N\_SHIFT\_RSTn*** Analog comparator reset control  
***kI2C1\_RST\_N\_SHIFT\_RSTn*** I2C1 reset control  
***kADC\_RST\_N\_SHIFT\_RSTn*** ADC reset control  
***kCTIMER0\_RST\_N\_SHIFT\_RSTn*** CTIMER0 reset control  
***kDAC0\_RST\_N\_SHIFT\_RSTn*** DAC0 reset control  
***kGPIOINT\_RST\_N\_SHIFT\_RSTn*** GPIOINT reset control  
***kCAPT\_RST\_N\_SHIFT\_RSTn*** Capacitive Touch reset control  
***kFRG0\_RST\_N\_SHIFT\_RSTn*** Fractional baud rate generator 0 reset control  
***kPLU\_RST\_N\_SHIFT\_RSTn*** PLU reset control

## 6.5 Function Documentation

### 6.5.1 void RESET\_SetPeripheralReset ( reset\_ip\_name\_t *peripheral* )

Asserts reset signal to specified peripheral module.

Parameters

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

### 6.5.2 void RESET\_ClearPeripheralReset ( reset\_ip\_name\_t *peripheral* )

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

Parameters

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

### 6.5.3 void RESET\_PeripheralReset ( reset\_ip\_name\_t *peripheral* )

Reset peripheral module.

Parameters

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

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

Release peripheral module.

Parameters

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

# Chapter 7

## CAPT: Capacitive Touch

### 7.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Capacitive Touch (CAPT) module of MCU-Xpresso SDK devices.

The Capacitive Touch module measures the change in capacitance of an electrode plate when an earth-ground connected object (for example, the finger or stylus) is brought within close proximity. Simply stated, the module delivers a small charge to an X capacitor (a mutual capacitance touch sensor), then transfers that charge to a larger Y capacitor (the measurement capacitor), and counts the number of iterations necessary for the voltage across the Y capacitor to cross a predetermined threshold.

### 7.2 Typical use case

#### 7.2.1 Normal Configuration

See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/capt/capt_basic`.

#### Files

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

#### Data Structures

- struct [\\_capt\\_config](#)  
*The structure for CAPT basic configuration. [More...](#)*
- struct [\\_capt\\_touch\\_data](#)  
*The structure for storing touch data. [More...](#)*

#### Typedefs

- typedef enum [\\_capt\\_trigger\\_mode](#) [capt\\_trigger\\_mode\\_t](#)  
*The enumeration for CAPT trigger mode.*
- typedef enum [\\_capt\\_inactive\\_xpins\\_mode](#) [capt\\_inactive\\_xpins\\_mode\\_t](#)  
*The enumeration for the inactive X pins mode.*
- typedef enum [\\_capt\\_measurement\\_delay](#) [capt\\_measurement\\_delay\\_t](#)  
*The enumeration for the delay of measuring voltage state.*
- typedef enum [\\_capt\\_reset\\_delay](#) [capt\\_reset\\_delay\\_t](#)  
*The enumeration for the delay of resetting or draining Cap.*
- typedef enum [\\_capt\\_polling\\_mode](#) [capt\\_polling\\_mode\\_t](#)

- *The enumeration of CAPT polling mode.*
- typedef enum `_capt_dma_mode` `capt_dma_mode_t`  
*The enumeration of CAPT DMA trigger mode.*
- typedef struct `_capt_config` `capt_config_t`  
*The structure for CAPT basic configuration.*
- typedef struct `_capt_touch_data` `capt_touch_data_t`  
*The structure for storing touch data.*

## Enumerations

- enum `_capt_xpins` {  
`kCAPT_X0Pin = 1U << 0U,`  
`kCAPT_X1Pin = 1U << 1U,`  
`kCAPT_X2Pin = 1U << 2U,`  
`kCAPT_X3Pin = 1U << 3U,`  
`kCAPT_X4Pin = 1U << 4U,`  
`kCAPT_X5Pin = 1U << 5U,`  
`kCAPT_X6Pin = 1U << 6U,`  
`kCAPT_X7Pin = 1U << 7U,`  
`kCAPT_X8Pin = 1U << 8U,`  
`kCAPT_X9Pin = 1U << 9U,`  
`kCAPT_X10Pin = 1U << 10U,`  
`kCAPT_X11Pin = 1U << 11U,`  
`kCAPT_X12Pin = 1U << 12U,`  
`kCAPT_X13Pin = 1U << 13U,`  
`kCAPT_X14Pin = 1U << 14U,`  
`kCAPT_X15Pin = 1U << 15U }`  
*The enumeration for X pins.*
- enum `_capt_interrupt_enable` {  
`kCAPT_InterruptOfYesTouchEnable,`  
`kCAPT_InterruptOfNoTouchEnable,`  
`kCAPT_InterruptOfPollDoneEnable = CAPT_INTENSET_POLLDONE_MASK,`  
`kCAPT_InterruptOfTimeOutEnable = CAPT_INTENSET_TIMEOUT_MASK,`  
`kCAPT_InterruptOfOverRunEnable = CAPT_INTENSET_OVERUN_MASK }`  
*The enumeration for enabling/disabling interrupts.*
- enum `_capt_interrupt_status_flags` {  
`kCAPT_InterruptOfYesTouchStatusFlag = CAPT_INTSTAT_YESTOUCH_MASK,`  
`kCAPT_InterruptOfNoTouchStatusFlag = CAPT_INTSTAT_NOTOUCH_MASK,`  
`kCAPT_InterruptOfPollDoneStatusFlag = CAPT_INTSTAT_POLLDONE_MASK,`  
`kCAPT_InterruptOfTimeOutStatusFlag = CAPT_INTSTAT_TIMEOUT_MASK,`  
`kCAPT_InterruptOfOverRunStatusFlag = CAPT_INTSTAT_OVERUN_MASK }`  
*The enumeration for interrupt status flags.*
- enum `_capt_status_flags` {  
`kCAPT_BusyStatusFlag = CAPT_STATUS_BUSY_MASK,`  
`kCAPT_XMAXStatusFlag = CAPT_STATUS_XMAX_MASK }`  
*The enumeration for CAPT status flags.*
- enum `_capt_trigger_mode` {

- ```

kCAPT_YHPortTriggerMode = 0U,
kCAPT_ComparatorTriggerMode = 1U }
    The enumeration for CAPT trigger mode.
• enum _capt_inactive_xpins_mode {
    kCAPT_InactiveXpinsHighZMode,
    kCAPT_InactiveXpinsDrivenLowMode }
    The enumeration for the inactive X pins mode.
• enum _capt_measurement_delay {
    kCAPT_MeasureDelayNoWait = 0U,
    kCAPT_MeasureDelayWait3FCLKs = 1U,
    kCAPT_MeasureDelayWait5FCLKs = 2U,
    kCAPT_MeasureDelayWait9FCLKs = 3U }
    The enumeration for the delay of measuring voltage state.
• enum _capt_reset_delay {
    kCAPT_ResetDelayNoWait = 0U,
    kCAPT_ResetDelayWait3FCLKs = 1U,
    kCAPT_ResetDelayWait5FCLKs = 2U,
    kCAPT_ResetDelayWait9FCLKs = 3U }
    The enumeration for the delay of resetting or draining Cap.
• enum _capt_polling_mode {
    kCAPT_PollInactiveMode,
    kCAPT_PollNowMode = 1U,
    kCAPT_PollContinuousMode }
    The enumeration of CAPT polling mode.
• enum _capt_dma_mode {
    kCAPT_DMATriggerOnTouchMode = 1U,
    kCAPT_DMATriggerOnBothMode = 2U,
    kCAPT_DMATriggerOnAllMode = 3U }
    The enumeration of CAPT DMA trigger mode.

```

## Driver version

- #define `FSL_CAPT_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 0)`)  
*CAPT driver version.*

## Initialization

- void `CAPT_Init` (`CAPT_Type *base`, const `capt_config_t *config`)  
*Initialize the CAPT module.*
- void `CAPT_Deinit` (`CAPT_Type *base`)  
*De-initialize the CAPT module.*
- void `CAPT_GetDefaultConfig` (`capt_config_t *config`)  
*Gets an available pre-defined settings for the CAPT's configuration.*
- static void `CAPT_SetThreshold` (`CAPT_Type *base`, `uint32_t count`)  
*Set Sets the count threshold in divided FCLKs between touch and no-touch.*
- void `CAPT_SetPollMode` (`CAPT_Type *base`, `capt_polling_mode_t mode`)  
*Set the CAPT polling mode.*
- static void `CAPT_EnableInterrupts` (`CAPT_Type *base`, `uint32_t mask`)

- *Enable interrupt features.*
- static void [CAPT\\_DisableInterrupts](#) (CAPT\_Type \*base, uint32\_t mask)  
*Disable interrupt features.*
- static uint32\_t [CAPT\\_GetInterruptStatusFlags](#) (CAPT\_Type \*base)  
*Get CAPT interrupts' status flags.*
- static void [CAPT\\_ClearInterruptStatusFlags](#) (CAPT\_Type \*base, uint32\_t mask)  
*Clear the interrupts' status flags.*
- static uint32\_t [CAPT\\_GetStatusFlags](#) (CAPT\_Type \*base)  
*Get CAPT status flags.*
- bool [CAPT\\_GetTouchData](#) (CAPT\_Type \*base, [capt\\_touch\\_data\\_t](#) \*data)  
*Get CAPT touch data.*
- void [CAPT\\_PollNow](#) (CAPT\_Type \*base, uint16\_t enableXpins)  
*Start touch data polling using poll-now method.*

## 7.3 Data Structure Documentation

### 7.3.1 struct `_capt_config`

#### Data Fields

- bool [enableWaitMode](#)  
*If enable the wait mode, when the touch event occurs, the module will wait until the TOUCH register is read before starting the next measurement.*
- bool [enableTouchLower](#)  
*enableTouchLower = true: Trigger at count < TCNT is a touch.*
- uint8\_t [clockDivider](#)  
*Function clock divider.*
- uint8\_t [timeOutCount](#)  
*Sets the count value at which a time-out event occurs if a measurement has not triggered.*
- uint8\_t [pollCount](#)  
*Sets the time delay between polling rounds (successive sets of X measurements).*
- uint16\_t [enableXpins](#)  
*Selects which of the available X pins are enabled.*
- [capt\\_trigger\\_mode\\_t](#) [triggerMode](#)  
*Select the methods of measuring the voltage across the measurement capacitor.*
- [capt\\_inactive\\_xpins\\_mode\\_t](#) [XpinsMode](#)  
*Determines how X pins enabled in the XPINSEL field are controlled when not active.*
- [capt\\_measurement\\_delay\\_t](#) [mDelay](#)  
*Set the time delay after entering step 3 (measure voltage state), before sampling the YH port pin or analog comparator output.*
- [capt\\_reset\\_delay\\_t](#) [rDelay](#)  
*Set the number of divided FCLKs the module will remain in Reset or Draining Cap.*

#### Field Documentation

##### (1) bool `_capt_config::enableWaitMode`

Other-wise, measurements continue.

**(2) bool \_capt\_config::enableTouchLower**

Trigger at count > TCNT is a no-touch. enableTouchLower = false: Trigger at count > TCNT is a touch. Trigger at count < TCNT is a no-touch. Notice: TCNT will be set by "CAPT\_DoCalibration" API.

**(3) uint8\_t \_capt\_config::clockDivider**

The function clock is divided by clockDivider+1 to produce the divided FCLK for the module. The available range is 0-15.

**(4) uint8\_t \_capt\_config::timeOutCount**

The time-out count value is calculated as  $2^{\text{timeOutCount}}$ . The available range is 0-12.

**(5) uint8\_t \_capt\_config::pollCount**

After each polling round completes, the module will wait  $4096 \times \text{PollCount}$  divided FCLKs before starting the next polling round. The available range is 0-255.

**(6) uint16\_t \_capt\_config::enableXpins**

Please refer to '\_capt\_xpins'. For example, if want to enable X0, X2 and X3 pins, you can set "enable-Xpins = kCAPT\_X0Pin | kCAPT\_X2Pin | kCAPT\_X3Pin".

**(7) capt\_trigger\_mode\_t \_capt\_config::triggerMode****(8) capt\_inactive\_xpins\_mode\_t \_capt\_config::XpinsMode****(9) capt\_measurement\_delay\_t \_capt\_config::mDelay****(10) capt\_reset\_delay\_t \_capt\_config::rDelay****7.3.2 struct \_capt\_touch\_data****Data Fields**

- bool [yesTimeOut](#)  
*'true': if the measurement resulted in a time-out event, 'false': otherwise.*
- bool [yesTouch](#)  
*'true': if the trigger is due to a touch even, 'false': if the trigger is due to a no-touch event.*
- uint8\_t [XpinsIndex](#)  
*Contains the index of the X pin for the current measurement, or lowest X for a multiple-pin poll now measurement.*
- uint8\_t [sequenceNumber](#)  
*Contains the 4-bit(0-7) sequence number, which increments at the end of each polling round.*
- uint16\_t [count](#)  
*Contains the count value reached at trigger or time-out.*

Field Documentation

- (1) `bool _capt_touch_data::yesTimeOut`
- (2) `bool _capt_touch_data::yesTouch`
- (3) `uint8_t _capt_touch_data::XpinsIndex`
- (4) `uint8_t _capt_touch_data::sequenceNumber`
- (5) `uint16_t _capt_touch_data::count`

7.4 Macro Definition Documentation

7.4.1 `#define FSL_CAPT_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

7.5 Enumeration Type Documentation

7.5.1 `enum _capt_xpins`

Enumerator

- kCAPT\_X0Pin* CAPT\_X0 pin.
- kCAPT\_X1Pin* CAPT\_X1 pin.
- kCAPT\_X2Pin* CAPT\_X2 pin.
- kCAPT\_X3Pin* CAPT\_X3 pin.
- kCAPT\_X4Pin* CAPT\_X4 pin.
- kCAPT\_X5Pin* CAPT\_X5 pin.
- kCAPT\_X6Pin* CAPT\_X6 pin.
- kCAPT\_X7Pin* CAPT\_X7 pin.
- kCAPT\_X8Pin* CAPT\_X8 pin.
- kCAPT\_X9Pin* CAPT\_X9 pin.
- kCAPT\_X10Pin* CAPT\_X10 pin.
- kCAPT\_X11Pin* CAPT\_X11 pin.
- kCAPT\_X12Pin* CAPT\_X12 pin.
- kCAPT\_X13Pin* CAPT\_X13 pin.
- kCAPT\_X14Pin* CAPT\_X14 pin.
- kCAPT\_X15Pin* CAPT\_X15 pin.

7.5.2 `enum _capt_interrupt_enable`

Enumerator

- kCAPT\_InterruptOfYesTouchEnable* Generate interrupt when a touch has been detected.
- kCAPT\_InterruptOfNoTouchEnable* Generate interrupt when a no-touch has been detected.
- kCAPT\_InterruptOfPollDoneEnable* Generate interrupt at the end of a polling round, or when a POLLNOW completes.

***kCAPT\_InterruptOfTimeOutEnable*** Generate interrupt when the count reaches the time-out count value before a trigger occurs.

***kCAPT\_InterruptOfOverRunEnable*** Generate interrupt when the Touch Data register has been updated before software has read the previous data, and the touch has been detected.

### 7.5.3 enum \_capt\_interrupt\_status\_flags

Enumerator

***kCAPT\_InterruptOfYesTouchStatusFlag*** YESTOUCH interrupt status flag.

***kCAPT\_InterruptOfNoTouchStatusFlag*** NOTOUCH interrupt status flag.

***kCAPT\_InterruptOfPollDoneStatusFlag*** POLLDONE interrupt status flag.

***kCAPT\_InterruptOfTimeOutStatusFlag*** TIMEOUT interrupt status flag.

***kCAPT\_InterruptOfOverRunStatusFlag*** OVERRUN interrupt status flag.

### 7.5.4 enum \_capt\_status\_flags

Enumerator

***kCAPT\_BusyStatusFlag*** Set while a poll is currently in progress, otherwise cleared.

***kCAPT\_XMAXStatusFlag*** The maximum number of X pins available for a given device is equal to XMAX+1.

### 7.5.5 enum \_capt\_trigger\_mode

Enumerator

***kCAPT\_YHPortTriggerMode*** YH port pin trigger mode.

***kCAPT\_ComparatorTriggerMode*** Analog comparator trigger mode.

### 7.5.6 enum \_capt\_inactive\_xpins\_mode

Enumerator

***kCAPT\_InactiveXpinsHighZMode*** Xpins enabled in the XPINSEL field are controlled to HIGH-Z mode when not active.

***kCAPT\_InactiveXpinsDrivenLowMode*** Xpins enabled in the XPINSEL field are controlled to be driven low mode when not active.

### 7.5.7 enum \_capt\_measurement\_delay

Enumerator

*kCAPT\_MeasureDelayNoWait* Don't wait.  
*kCAPT\_MeasureDelayWait3FCLKs* Wait 3 divided FCLKs.  
*kCAPT\_MeasureDelayWait5FCLKs* Wait 5 divided FCLKs.  
*kCAPT\_MeasureDelayWait9FCLKs* Wait 9 divided FCLKs.

### 7.5.8 enum \_capt\_reset\_delay

Enumerator

*kCAPT\_ResetDelayNoWait* Don't wait.  
*kCAPT\_ResetDelayWait3FCLKs* Wait 3 divided FCLKs.  
*kCAPT\_ResetDelayWait5FCLKs* Wait 5 divided FCLKs.  
*kCAPT\_ResetDelayWait9FCLKs* Wait 9 divided FCLKs.

### 7.5.9 enum \_capt\_polling\_mode

Enumerator

*kCAPT\_PollInactiveMode* No measurements are taken, no polls are performed. The module remains in the Reset/ Draining Cap.  
*kCAPT\_PollNowMode* Immediately launches (ignoring Poll Delay) a one-time-only, simultaneous poll of all X pins that are enabled in the XPINSEL field of the Control register, then stops, returning to Reset/Draining Cap.  
*kCAPT\_PollContinuousMode* Polling rounds are continuously performed, by walking through the enabled X pins.

### 7.5.10 enum \_capt\_dma\_mode

Enumerator

*kCAPT\_DMATriggerOnTouchMode* Trigger on touch.  
*kCAPT\_DMATriggerOnBothMode* Trigger on both touch and no-touch.  
*kCAPT\_DMATriggerOnAllMode* Trigger on all touch, no-touch and time-out.

## 7.6 Function Documentation

### 7.6.1 void CAPT\_Init ( CAPT\_Type \* base, const capt\_config\_t \* config )

## Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>base</i>   | CAPT peripheral base address.         |
| <i>config</i> | Pointer to "capt_config_t" structure. |

**7.6.2 void CAPT\_Deinit ( CAPT\_Type \* *base* )**

## Parameters

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

**7.6.3 void CAPT\_GetDefaultConfig ( capt\_config\_t \* *config* )**

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

```
* config->enableWaitMode = false;
* config->enableTouchLower = true;
* config->clockDivider = 15U;
* config->timeOutCount = 12U;
* config->pollCount = 0U;
* config->enableXpins = 0U;
* config->triggerMode = kCAPT_YHPortTriggerMode;
* config->XpinsMode = kCAPT_InactiveXpinsDrivenLowMode;
* config->mDelay = kCAPT_MeasureDelayNoWait;
* config->rDelay = kCAPT_ResetDelayWait9FCLKs;
*
```

## Parameters

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

**7.6.4 static void CAPT\_SetThreshold ( CAPT\_Type \* *base*, uint32\_t *count* )  
[inline], [static]**

## Parameters

---

|              |                               |
|--------------|-------------------------------|
| <i>base</i>  | CAPT peripheral base address. |
| <i>count</i> | The count threshold.          |

### 7.6.5 void CAPT\_SetPollMode ( CAPT\_Type \* *base*, capt\_polling\_mode\_t *mode* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | CAPT peripheral base address.  |
| <i>mode</i> | The selection of polling mode. |

### 7.6.6 static void CAPT\_EnableInterrupts ( CAPT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                    |
|-------------|------------------------------------------------------------------------------------|
| <i>base</i> | CAPT peripheral base address.                                                      |
| <i>mask</i> | The mask of enabling interrupt features. Please refer to "_capt_interrupt_enable". |

### 7.6.7 static void CAPT\_DisableInterrupts ( CAPT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                     |
|-------------|-------------------------------------------------------------------------------------|
| <i>base</i> | CAPT peripheral base address.                                                       |
| <i>mask</i> | The mask of disabling interrupt features. Please refer to "_capt_interrupt_enable". |

### 7.6.8 static uint32\_t CAPT\_GetInterruptStatusFlags ( CAPT\_Type \* *base* ) [inline], [static]

Parameters

---

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

Returns

The mask of interrupts' status flags. please refer to "\_capt\_interrupt\_status\_flags".

### 7.6.9 static void CAPT\_ClearInterruptStatusFlags ( CAPT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------|
| <i>base</i> | CAPT peripheral base address.                                                                      |
| <i>mask</i> | The mask of clearing the interrupts' status flags, please refer to "_capt_interrupt_status_flags". |

### 7.6.10 static uint32\_t CAPT\_GetStatusFlags ( CAPT\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

The mask of CAPT status flags. Please refer to "\_capt\_status\_flags" Or use CAPT\_GET\_XMAX\_NUMBER(mask) to get XMAX number.

### 7.6.11 bool CAPT\_GetTouchData ( CAPT\_Type \* *base*, capt\_touch\_data\_t \* *data* )

Parameters

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

|             |                                    |
|-------------|------------------------------------|
| <i>data</i> | The structure to store touch data. |
|-------------|------------------------------------|

Returns

If return 'true', which means get valid data. if return 'false', which means get invalid data.

**7.6.12 void CAPT\_PollNow ( CAPT\_Type \* *base*, uint16\_t *enableXpins* )**

This function starts new data polling using polling-now method, CAPT stops when the polling is finished, application could check the status or monitor interrupt to know when the progress is finished.

Note that this is simultaneous poll of all X pins, all enabled X pins are activated concurrently, rather than walked one-at-a-time

Parameters

|                    |                                     |
|--------------------|-------------------------------------|
| <i>base</i>        | CAPT peripheral base address.       |
| <i>enableXpins</i> | The X pins enabled in this polling. |

# Chapter 8

## Common Driver

### 8.1 Overview

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

#### Macros

- #define `FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ` 1  
*Macro to use the default weak IRQ handler in drivers.*
- #define `MAKE_STATUS`(group, code) (((group)\*100L) + (code))  
*Construct a status code value from a group and code number.*
- #define `MAKE_VERSION`(major, minor, bugfix) (((major)\*65536L) + ((minor)\*256L) + (bugfix))  
*Construct the version number for drivers.*
- #define `ARRAY_SIZE`(x) (sizeof(x) / sizeof((x)[0]))  
*Computes the number of elements in an array.*
- #define `SUPPRESS_FALL_THROUGH_WARNING`()  
*For switch case code block, if case section ends without "break;" statement, there wil be fallthrough warning with compiler flag -Wextra or -Wimplicit-fallthrough=n when using armgcc.*

#### Typedefs

- typedef int32\_t `status_t`  
*Type used for all status and error return values.*

## Enumerations

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

```
kStatusGroup_ELE = 167 }
```

*Status group numbers.*

- enum {
  - kStatus\_Success = MAKE\_STATUS(kStatusGroup\_Generic, 0),
  - kStatus\_Fail = MAKE\_STATUS(kStatusGroup\_Generic, 1),
  - kStatus\_ReadOnly = MAKE\_STATUS(kStatusGroup\_Generic, 2),
  - kStatus\_OutOfRange = MAKE\_STATUS(kStatusGroup\_Generic, 3),
  - kStatus\_InvalidArgument = MAKE\_STATUS(kStatusGroup\_Generic, 4),
  - kStatus\_Timeout = MAKE\_STATUS(kStatusGroup\_Generic, 5),
  - kStatus\_NoTransferInProgress,
  - kStatus\_Busy = MAKE\_STATUS(kStatusGroup\_Generic, 7),
  - kStatus\_NoData }

*Generic status return codes.*

## Functions

- void \* **SDK\_Malloc** (size\_t size, size\_t alignbytes)
  - Allocate memory with given alignment and aligned size.*
- void **SDK\_Free** (void \*ptr)
  - Free memory.*
- void **SDK\_DelayAtLeastUs** (uint32\_t delayTime\_us, uint32\_t coreClock\_Hz)
  - Delay at least for some time.*
- static **status\_t EnableIRQ** (IRQn\_Type interrupt)
  - Enable specific interrupt.*
- static **status\_t DisableIRQ** (IRQn\_Type interrupt)
  - Disable specific interrupt.*
- static **status\_t EnableIRQWithPriority** (IRQn\_Type interrupt, uint8\_t priNum)
  - Enable the IRQ, and also set the interrupt priority.*
- static **status\_t IRQ\_SetPriority** (IRQn\_Type interrupt, uint8\_t priNum)
  - Set the IRQ priority.*
- static **status\_t IRQ\_ClearPendingIRQ** (IRQn\_Type interrupt)
  - Clear the pending IRQ flag.*
- static uint32\_t **DisableGlobalIRQ** (void)
  - Disable the global IRQ.*
- static void **EnableGlobalIRQ** (uint32\_t primask)
  - Enable the global IRQ.*
- void **EnableDeepSleepIRQ** (IRQn\_Type interrupt)
  - Enable specific interrupt for wake-up from deep-sleep mode.*
- void **DisableDeepSleepIRQ** (IRQn\_Type interrupt)
  - Disable specific interrupt for wake-up from deep-sleep mode.*

## Driver version

- #define **FSL\_COMMON\_DRIVER\_VERSION** (MAKE\_VERSION(2, 4, 0))
  - common driver version.*

## Debug console type definition.

- #define **DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE** 0U

- *No debug console.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_UART` 1U  
*Debug console based on UART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_LPUART` 2U  
*Debug console based on LPUART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_LPSCI` 3U  
*Debug console based on LPSCI.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_USBCDC` 4U  
*Debug console based on USBCDC.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM` 5U  
*Debug console based on FLEXCOMM.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_IUART` 6U  
*Debug console based on i.MX UART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_VUSART` 7U  
*Debug console based on LPC\_VUSART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART` 8U  
*Debug console based on LPC\_USART.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_SWO` 9U  
*Debug console based on SWO.*
- #define `DEBUG_CONSOLE_DEVICE_TYPE_QSCI` 10U  
*Debug console based on QSCI.*

## Min/max macros

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

## UINT16\_MAX/UINT32\_MAX value

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

## Atomic modification

These macros are used for atomic access, such as read-modify-write to the peripheral registers.

Take `SDK_ATOMIC_LOCAL_CLEAR_AND_SET` as an example: the parameter `addr` means the address of the peripheral register or variable you want to modify atomically, the parameter `clearBits` is the bits to clear, the parameter `setBits` it the bits to set. For example, to set a 32-bit register bit1:bit0 to 0b10, use like this:

```
volatile uint32_t * reg = (volatile uint32_t *)REG_ADDR;
SDK_ATOMIC_LOCAL_CLEAR_AND_SET(reg, 0x03, 0x02);
```

In this example, the register bit1:bit0 are cleared and bit1 is set, as a result, register bit1:bit0 = 0b10.

## Note

For the platforms don't support exclusive load and store, these macros disable the global interrupt to protect the modification.

These macros only guarantee the local processor atomic operations. For the multi-processor devices, use hardware semaphore such as SEMA42 to guarantee exclusive access if necessary.

- #define `SDK_ATOMIC_LOCAL_ADD(addr, val)`  
*Add value val from the variable at address address.*
- #define `SDK_ATOMIC_LOCAL_SUB(addr, val)`  
*Subtract value val to the variable at address address.*
- #define `SDK_ATOMIC_LOCAL_SET(addr, bits)`  
*Set the bits specified by bits to the variable at address address.*
- #define `SDK_ATOMIC_LOCAL_CLEAR(addr, bits)`  
*Clear the bits specified by bits to the variable at address address.*
- #define `SDK_ATOMIC_LOCAL_TOGGLE(addr, bits)`  
*Toggle the bits specified by bits to the variable at address address.*
- #define `SDK_ATOMIC_LOCAL_CLEAR_AND_SET(addr, clearBits, setBits)`  
*For the variable at address address, clear the bits specified by clearBits and set the bits specified by setBits.*

## Timer utilities

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

## Alignment variable definition macros

- #define `SDK_ALIGN(var, alignbytes)` `var __attribute__((aligned(alignbytes)))`  
*Macro to define a variable with alignbytes alignment.*
- #define `SDK_SIZEALIGN(var, alignbytes)` `((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(~(unsigned int)((alignbytes)-1U)))`  
*Macro to define a variable with L1 d-cache line size alignment.*

## Non-cacheable region definition macros

For initialized non-zero non-cacheable variables, please use "AT\_NONCACHEABLE\_SECTION\_INIT(var)={xx};" or "AT\_NONCACHEABLE\_SECTION\_ALIGN\_INIT(var)={xx};" in your projects to define them.

For zero-initiated non-cacheable variables, please use "AT\_NONCACHEABLE\_SECTION(var);" or "AT\_NONCACHEABLE\_SECTION\_ALIGN(var);" to define them, these zero-initiated variables will be initialized to zero in system startup.

#### Note

For GCC, when the non-cacheable section is required, please define "\_\_STARTUP\_INITIALIZE\_NONCACHEDATA" in your projects to make sure the non-cacheable section variables will be initialized in system startup.

- #define `AT_NONCACHEABLE_SECTION`(var) var  
*Define a variable var, and place it in non-cacheable section.*
- #define `AT_NONCACHEABLE_SECTION_ALIGN`(var, alignbytes) `SDK_ALIGN`(var, alignbytes)  
*Define a variable var, and place it in non-cacheable section, the start address of the variable is aligned to alignbytes.*
- #define `AT_NONCACHEABLE_SECTION_INIT`(var) var  
*Define a variable var with initial value, and place it in non-cacheable section.*
- #define `AT_NONCACHEABLE_SECTION_ALIGN_INIT`(var, alignbytes) `SDK_ALIGN`(var, alignbytes)  
*Define a variable var with initial value, and place it in non-cacheable section, the start address of the variable is aligned to alignbytes.*

### Time sensitive region

- #define `AT_QUICKACCESS_SECTION_CODE`(func) `__attribute__((section("CodeQuickAccess"), __noinline__))` func  
*Place function in a section which can be accessed quickly by core.*
- #define `AT_QUICKACCESS_SECTION_DATA`(var) `__attribute__((section("DataQuickAccess")))` var  
*Place data in a section which can be accessed quickly by core.*
- #define `AT_QUICKACCESS_SECTION_DATA_ALIGN`(var, alignbytes) `__attribute__((section("DataQuickAccess"))) var __attribute__((aligned(alignbytes)))`  
*Place data in a section which can be accessed quickly by core, and the variable address is set to align with alignbytes.*

### Ram Function

- #define `RAMFUNCTION_SECTION_CODE`(func) `__attribute__((section("RamFunction")))` func  
*Place function in ram.*

## 8.2 Macro Definition Documentation

8.2.1 `#define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1`

8.2.2 `#define MAKE_STATUS( group, code ) (((group)*100L) + (code))`

8.2.3 `#define MAKE_VERSION( major, minor, bugfix ) (((major)*65536L) + ((minor)*256L) + (bugfix))`

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

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

8.2.4 **#define FSL\_COMMON\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 0))**

8.2.5 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_NONE 0U**

8.2.6 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_UART 1U**

8.2.7 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPUART 2U**

8.2.8 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_LPSCI 3U**

8.2.9 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_USBCDC 4U**

8.2.10 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_FLEXCOMM 5U**

8.2.11 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_IUART 6U**

8.2.12 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_VUSART 7U**

8.2.13 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_MINI\_USART 8U**

8.2.14 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_SWO 9U**

8.2.15 **#define DEBUG\_CONSOLE\_DEVICE\_TYPE\_QSCI 10U**

8.2.16 **#define MIN( a, b ) (((a) < (b)) ? (a) : (b))**

8.2.17 **#define MAX( a, b ) (((a) > (b)) ? (a) : (b))**

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

8.2.19 **#define UINT16\_MAX ((uint16\_t)-1)**

8.2.20 **#define UINT32\_MAX ((uint32\_t)-1)**

8.2.21 **#define SUPPRESS\_FALL\_THROUGH\_WARNING( )**

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

## 8.2.22 #define SDK\_SIZEALIGN( var, alignbytes ) ((unsigned int)((var) + ((alignbytes)-1U)) & (unsigned int)(~(unsigned int)((alignbytes)-1U)))

Macro to define a variable with L2 cache line size alignment

Macro to change a value to a given size aligned value

## 8.3 Typedef Documentation

### 8.3.1 typedef int32\_t status\_t

## 8.4 Enumeration Type Documentation

### 8.4.1 enum \_status\_groups

Enumerator

*kStatusGroup\_Generic* Group number for generic status codes.  
*kStatusGroup\_FLASH* Group number for FLASH status codes.  
*kStatusGroup\_LPSPI* Group number for LPSPI status codes.  
*kStatusGroup\_FLEXIO\_SPI* Group number for FLEXIO SPI status codes.  
*kStatusGroup\_DSPI* Group number for DSPI status codes.  
*kStatusGroup\_FLEXIO\_UART* Group number for FLEXIO UART status codes.  
*kStatusGroup\_FLEXIO\_I2C* Group number for FLEXIO I2C status codes.  
*kStatusGroup\_LPI2C* Group number for LPI2C status codes.  
*kStatusGroup\_UART* Group number for UART status codes.  
*kStatusGroup\_I2C* Group number for I2C status codes.  
*kStatusGroup\_LPSCI* Group number for LPSCI status codes.  
*kStatusGroup\_LPUART* Group number for LPUART status codes.  
*kStatusGroup\_SPI* Group number for SPI status code.  
*kStatusGroup\_XRDC* Group number for XRDC status code.  
*kStatusGroup\_SEMA42* Group number for SEMA42 status code.  
*kStatusGroup\_SDHC* Group number for SDHC status code.  
*kStatusGroup\_SDMMC* Group number for SDMMC status code.  
*kStatusGroup\_SAI* Group number for SAI status code.  
*kStatusGroup\_MCG* Group number for MCG status codes.  
*kStatusGroup\_SCG* Group number for SCG status codes.  
*kStatusGroup\_SDSPI* Group number for SDSPI status codes.  
*kStatusGroup\_FLEXIO\_I2S* Group number for FLEXIO I2S status codes.  
*kStatusGroup\_FLEXIO\_MCULCD* Group number for FLEXIO LCD status codes.  
*kStatusGroup\_FLASHIAP* Group number for FLASHIAP status codes.  
*kStatusGroup\_FLEXCOMM\_I2C* Group number for FLEXCOMM I2C status codes.  
*kStatusGroup\_I2S* Group number for I2S status codes.  
*kStatusGroup\_IUART* Group number for IUART status codes.  
*kStatusGroup\_CSI* Group number for CSI status codes.  
*kStatusGroup\_MIPI\_DSI* Group number for MIPI DSI status codes.

*kStatusGroup\_SDRAMC* Group number for SDRAMC status codes.  
*kStatusGroup\_POWER* Group number for POWER status codes.  
*kStatusGroup\_ENET* Group number for ENET status codes.  
*kStatusGroup\_PHY* Group number for PHY status codes.  
*kStatusGroup\_TRGMUX* Group number for TRGMUX status codes.  
*kStatusGroup\_SMARTCARD* Group number for SMARTCARD status codes.  
*kStatusGroup\_LMEM* Group number for LMEM status codes.  
*kStatusGroup\_QSPI* Group number for QSPI status codes.  
*kStatusGroup\_DMA* Group number for DMA status codes.  
*kStatusGroup\_EDMA* Group number for EDMA status codes.  
*kStatusGroup\_DMAMGR* Group number for DMAMGR status codes.  
*kStatusGroup\_FLEXCAN* Group number for FlexCAN status codes.  
*kStatusGroup\_LTC* Group number for LTC status codes.  
*kStatusGroup\_FLEXIO\_CAMERA* Group number for FLEXIO CAMERA status codes.  
*kStatusGroup\_LPC\_SPI* Group number for LPC\_SPI status codes.  
*kStatusGroup\_LPC\_USART* Group number for LPC\_USART status codes.  
*kStatusGroup\_DMIC* Group number for DMIC status codes.  
*kStatusGroup\_SDIF* Group number for SDIF status codes.  
*kStatusGroup\_SPIFI* Group number for SPIFI status codes.  
*kStatusGroup\_OTP* Group number for OTP status codes.  
*kStatusGroup\_MCAN* Group number for MCAN status codes.  
*kStatusGroup\_CAAM* Group number for CAAM status codes.  
*kStatusGroup\_ECSPi* Group number for ECSPi status codes.  
*kStatusGroup\_USDHC* Group number for USDHC status codes.  
*kStatusGroup\_LPC\_I2C* Group number for LPC\_I2C status codes.  
*kStatusGroup\_DCP* Group number for DCP status codes.  
*kStatusGroup\_MSCAN* Group number for MSCAN status codes.  
*kStatusGroup\_ESAI* Group number for ESAI status codes.  
*kStatusGroup\_FLEXSPI* Group number for FLEXSPI status codes.  
*kStatusGroup\_MMDC* Group number for MMDC status codes.  
*kStatusGroup\_PDM* Group number for MIC status codes.  
*kStatusGroup\_SDMA* Group number for SDMA status codes.  
*kStatusGroup\_ICS* Group number for ICS status codes.  
*kStatusGroup\_SPDIF* Group number for SPDIF status codes.  
*kStatusGroup\_LPC\_MINISPI* Group number for LPC\_MINISPI status codes.  
*kStatusGroup\_HASHCRYPT* Group number for Hashcrypt status codes.  
*kStatusGroup\_LPC\_SPI\_SSP* Group number for LPC\_SPI\_SSP status codes.  
*kStatusGroup\_I3C* Group number for I3C status codes.  
*kStatusGroup\_LPC\_I2C\_1* Group number for LPC\_I2C\_1 status codes.  
*kStatusGroup\_NOTIFIER* Group number for NOTIFIER status codes.  
*kStatusGroup\_DebugConsole* Group number for debug console status codes.  
*kStatusGroup\_SEMC* Group number for SEMC status codes.  
*kStatusGroup\_ApplicationRangeStart* Starting number for application groups.  
*kStatusGroup\_IAP* Group number for IAP status codes.  
*kStatusGroup\_SFA* Group number for SFA status codes.

*kStatusGroup\_SPC* Group number for SPC status codes.  
*kStatusGroup\_PUF* Group number for PUF status codes.  
*kStatusGroup\_TOUCH\_PANEL* Group number for touch panel status codes.  
*kStatusGroup\_VBAT* Group number for VBAT status codes.  
*kStatusGroup\_HAL\_GPIO* Group number for HAL GPIO status codes.  
*kStatusGroup\_HAL\_UART* Group number for HAL UART status codes.  
*kStatusGroup\_HAL\_TIMER* Group number for HAL TIMER status codes.  
*kStatusGroup\_HAL\_SPI* Group number for HAL SPI status codes.  
*kStatusGroup\_HAL\_I2C* Group number for HAL I2C status codes.  
*kStatusGroup\_HAL\_FLASH* Group number for HAL FLASH status codes.  
*kStatusGroup\_HAL\_PWM* Group number for HAL PWM status codes.  
*kStatusGroup\_HAL\_RNG* Group number for HAL RNG status codes.  
*kStatusGroup\_HAL\_I2S* Group number for HAL I2S status codes.  
*kStatusGroup\_HAL\_ADC\_SENSOR* Group number for HAL ADC SENSOR status codes.  
*kStatusGroup\_TIMERMANAGER* Group number for TiMER MANAGER status codes.  
*kStatusGroup\_SERIALMANAGER* Group number for SERIAL MANAGER status codes.  
*kStatusGroup\_LED* Group number for LED status codes.  
*kStatusGroup\_BUTTON* Group number for BUTTON status codes.  
*kStatusGroup\_EXTERN\_EEPROM* Group number for EXTERN EEPROM status codes.  
*kStatusGroup\_SHELL* Group number for SHELL status codes.  
*kStatusGroup\_MEM\_MANAGER* Group number for MEM MANAGER status codes.  
*kStatusGroup\_LIST* Group number for List status codes.  
*kStatusGroup\_OSA* Group number for OSA status codes.  
*kStatusGroup\_COMMON\_TASK* Group number for Common task status codes.  
*kStatusGroup\_MSG* Group number for messaging status codes.  
*kStatusGroup\_SDK\_OCOTP* Group number for OCOTP status codes.  
*kStatusGroup\_SDK\_FLEXSPINOR* Group number for FLEXSPINOR status codes.  
*kStatusGroup\_CODEC* Group number for codec status codes.  
*kStatusGroup\_ASRC* Group number for codec status ASRC.  
*kStatusGroup\_OTFAD* Group number for codec status codes.  
*kStatusGroup\_SDIO SLV* Group number for SDIO SLV status codes.  
*kStatusGroup\_MECC* Group number for MECC status codes.  
*kStatusGroup\_ENET\_QOS* Group number for ENET\_QOS status codes.  
*kStatusGroup\_LOG* Group number for LOG status codes.  
*kStatusGroup\_I3CBUS* Group number for I3CBUS status codes.  
*kStatusGroup\_QSCI* Group number for QSCI status codes.  
*kStatusGroup\_ELEMU* Group number for ELEMU status codes.  
*kStatusGroup\_QUEUEDSPI* Group number for QSPI status codes.  
*kStatusGroup\_POWER\_MANAGER* Group number for POWER\_MANAGER status codes.  
*kStatusGroup\_IPED* Group number for IPED status codes.  
*kStatusGroup\_ELS\_PKC* Group number for ELS PKC status codes.  
*kStatusGroup\_CSS\_PKC* Group number for CSS PKC status codes.  
*kStatusGroup\_HOSTIF* Group number for HOSTIF status codes.  
*kStatusGroup\_CLIF* Group number for CLIF status codes.  
*kStatusGroup\_BMA* Group number for BMA status codes.

*kStatusGroup\_NETC* Group number for NETC status codes.

*kStatusGroup\_ELE* Group number for ELE status codes.

## 8.4.2 anonymous enum

Enumerator

*kStatus\_Success* Generic status for Success.

*kStatus\_Fail* Generic status for Fail.

*kStatus\_ReadOnly* Generic status for read only failure.

*kStatus\_OutOfRange* Generic status for out of range access.

*kStatus\_InvalidArgument* Generic status for invalid argument check.

*kStatus\_Timeout* Generic status for timeout.

*kStatus\_NoTransferInProgress* Generic status for no transfer in progress.

*kStatus\_Busy* Generic status for module is busy.

*kStatus\_NoData* Generic status for no data is found for the operation.

## 8.5 Function Documentation

### 8.5.1 void\* SDK\_Malloc ( size\_t size, size\_t alignbytes )

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

Parameters

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

Return values

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

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

Parameters

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

### 8.5.3 void SDK\_DelayAtLeastUs ( uint32\_t delayTime\_us, uint32\_t coreClock\_Hz )

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

## Parameters

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

**8.5.4 static status\_t EnableIRQ ( IRQn\_Type *interrupt* ) [inline], [static]**

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

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

## Parameters

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

## Return values

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

**8.5.5 static status\_t DisableIRQ ( IRQn\_Type *interrupt* ) [inline], [static]**

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

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

## Parameters

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

Return values

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

### 8.5.6 static status\_t EnableIRQWithPriority ( IRQn\_Type *interrupt*, uint8\_t *priNum* ) [inline], [static]

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

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>interrupt</i> | The IRQ to Enable.                                    |
| <i>priNum</i>    | Priority number set to interrupt controller register. |

Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Interrupt priority set successfully   |
| <i>kStatus_Fail</i>    | Failed to set the interrupt priority. |

### 8.5.7 static status\_t IRQ\_SetPriority ( IRQn\_Type *interrupt*, uint8\_t *priNum* ) [inline], [static]

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

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

## Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>interrupt</i> | The IRQ to set.                                       |
| <i>priNum</i>    | Priority number set to interrupt controller register. |

## Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Interrupt priority set successfully   |
| <i>kStatus_Fail</i>    | Failed to set the interrupt priority. |

### 8.5.8 static status\_t IRQ\_ClearPendingIRQ ( IRQn\_Type *interrupt* ) [inline], [static]

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

This function only handles the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro FSL\_FEATURE\_NUMBER\_OF\_LEVEL1\_INT\_VECTORS.

## Parameters

|                  |                              |
|------------------|------------------------------|
| <i>interrupt</i> | The flag which IRQ to clear. |
|------------------|------------------------------|

## Return values

|                        |                                       |
|------------------------|---------------------------------------|
| <i>kStatus_Success</i> | Interrupt priority set successfully   |
| <i>kStatus_Fail</i>    | Failed to set the interrupt priority. |

### 8.5.9 static uint32\_t DisableGlobalIRQ ( void ) [inline], [static]

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

Returns

Current primask value.

### 8.5.10 `static void EnableGlobalIRQ ( uint32_t primask ) [inline], [static]`

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

Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>primask</i> | value of primask register to be restored. The primask value is supposed to be provided by the <a href="#">DisableGlobalIRQ()</a> . |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|

### 8.5.11 `void EnableDeepSleepIRQ ( IRQn_Type interrupt )`

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

Note

This function also enables the interrupt in the NVIC ([EnableIRQ\(\)](#) is called internally).

Parameters

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

### 8.5.12 `void DisableDeepSleepIRQ ( IRQn_Type interrupt )`

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

Note

This function also disables the interrupt in the NVIC ([DisableIRQ\(\)](#) is called internally).

Parameters

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

## Chapter 9

# CTIMER: Standard counter/timers

### 9.1 Overview

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

### 9.2 Function groups

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

#### 9.2.1 Initialization and deinitialization

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

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

#### 9.2.2 PWM Operations

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

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

#### 9.2.3 Match Operation

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

#### 9.2.4 Input capture operations

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

## 9.3 Typical use case

### 9.3.1 Match example

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

### 9.3.2 PWM output example

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

## Files

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

## Data Structures

- struct [\\_ctimer\\_match\\_config](#)  
*Match configuration. [More...](#)*
- struct [\\_ctimer\\_config](#)  
*Timer configuration structure. [More...](#)*

## Typedefs

- typedef enum  
[\\_ctimer\\_capture\\_channel](#) [ctimer\\_capture\\_channel\\_t](#)  
*List of Timer capture channels.*
- typedef enum [\\_ctimer\\_capture\\_edge](#) [ctimer\\_capture\\_edge\\_t](#)  
*List of capture edge options.*
- typedef enum [\\_ctimer\\_match](#) [ctimer\\_match\\_t](#)  
*List of Timer match registers.*
- typedef enum [\\_ctimer\\_external\\_match](#) [ctimer\\_external\\_match\\_t](#)  
*List of external match.*
- typedef enum  
[\\_ctimer\\_match\\_output\\_control](#) [ctimer\\_match\\_output\\_control\\_t](#)  
*List of output control options.*
- typedef enum [\\_ctimer\\_timer\\_mode](#) [ctimer\\_timer\\_mode\\_t](#)  
*List of Timer modes.*
- typedef enum  
[\\_ctimer\\_interrupt\\_enable](#) [ctimer\\_interrupt\\_enable\\_t](#)  
*List of Timer interrupts.*
- typedef enum [\\_ctimer\\_status\\_flags](#) [ctimer\\_status\\_flags\\_t](#)  
*List of Timer flags.*
- typedef struct [\\_ctimer\\_match\\_config](#) [ctimer\\_match\\_config\\_t](#)  
*Match configuration.*
- typedef struct [\\_ctimer\\_config](#) [ctimer\\_config\\_t](#)  
*Timer configuration structure.*

## Enumerations

- enum `_ctimer_capture_channel` {  
`kCTIMER_Capture_0` = 0U,  
`kCTIMER_Capture_1`,  
`kCTIMER_Capture_2` }
- List of Timer capture channels.*
- enum `_ctimer_capture_edge` {  
`kCTIMER_Capture_RiseEdge` = 1U,  
`kCTIMER_Capture_FallEdge` = 2U,  
`kCTIMER_Capture_BothEdge` = 3U }
- List of capture edge options.*
- enum `_ctimer_match` {  
`kCTIMER_Match_0` = 0U,  
`kCTIMER_Match_1`,  
`kCTIMER_Match_2`,  
`kCTIMER_Match_3` }
- List of Timer match registers.*
- enum `_ctimer_external_match` {  
`kCTIMER_External_Match_0` = (1UL << 0),  
`kCTIMER_External_Match_1` = (1UL << 1),  
`kCTIMER_External_Match_2` = (1UL << 2),  
`kCTIMER_External_Match_3` = (1UL << 3) }
- List of external match.*
- enum `_ctimer_match_output_control` {  
`kCTIMER_Output_NoAction` = 0U,  
`kCTIMER_Output_Clear`,  
`kCTIMER_Output_Set`,  
`kCTIMER_Output_Toggle` }
- List of output control options.*
- enum `_ctimer_timer_mode`
- List of Timer modes.*
- enum `_ctimer_interrupt_enable` {  
`kCTIMER_Match0InterruptEnable` = CTIMER\_MCR\_MR0I\_MASK,  
`kCTIMER_Match1InterruptEnable` = CTIMER\_MCR\_MR1I\_MASK,  
`kCTIMER_Match2InterruptEnable` = CTIMER\_MCR\_MR2I\_MASK,  
`kCTIMER_Match3InterruptEnable` = CTIMER\_MCR\_MR3I\_MASK,  
`kCTIMER_Capture0InterruptEnable` = CTIMER\_CCR\_CAP0I\_MASK,  
`kCTIMER_Capture1InterruptEnable` = CTIMER\_CCR\_CAP1I\_MASK,  
`kCTIMER_Capture2InterruptEnable` = CTIMER\_CCR\_CAP2I\_MASK }
- List of Timer interrupts.*
- enum `_ctimer_status_flags` {

```

kCTIMER_Match0Flag = CTIMER_IR_MR0INT_MASK,
kCTIMER_Match1Flag = CTIMER_IR_MR1INT_MASK,
kCTIMER_Match2Flag = CTIMER_IR_MR2INT_MASK,
kCTIMER_Match3Flag = CTIMER_IR_MR3INT_MASK,
kCTIMER_Capture0Flag = CTIMER_IR_CR0INT_MASK,
kCTIMER_Capture1Flag = CTIMER_IR_CR1INT_MASK,
kCTIMER_Capture2Flag = CTIMER_IR_CR2INT_MASK }

```

*List of Timer flags.*

- enum `ctimer_callback_type_t` {  
`kCTIMER_SingleCallback`,  
`kCTIMER_MultipleCallback` }

*Callback type when registering for a callback.*

## Functions

- void `CTIMER_SetupMatch` (`CTIMER_Type *base`, `ctimer_match_t matchChannel`, const `ctimer_match_config_t *config`)  
*Setup the match register.*
- uint32\_t `CTIMER_GetOutputMatchStatus` (`CTIMER_Type *base`, uint32\_t `matchChannel`)  
*Get the status of output match.*
- void `CTIMER_SetupCapture` (`CTIMER_Type *base`, `ctimer_capture_channel_t capture`, `ctimer_capture_edge_t edge`, bool `enableInt`)  
*Setup the capture.*
- static uint32\_t `CTIMER_GetTimerCountValue` (`CTIMER_Type *base`)  
*Get the timer count value from TC register.*
- void `CTIMER_RegisterCallback` (`CTIMER_Type *base`, `ctimer_callback_t *cb_func`, `ctimer_callback_type_t cb_type`)  
*Register callback.*
- static void `CTIMER_Reset` (`CTIMER_Type *base`)  
*Reset the counter.*
- static void `CTIMER_SetPrescale` (`CTIMER_Type *base`, uint32\_t `prescale`)  
*Setup the timer prescale value.*
- static uint32\_t `CTIMER_GetCaptureValue` (`CTIMER_Type *base`, `ctimer_capture_channel_t capture`)  
*Get capture channel value.*
- static void `CTIMER_EnableResetMatchChannel` (`CTIMER_Type *base`, `ctimer_match_t match`, bool `enable`)  
*Enable reset match channel.*
- static void `CTIMER_EnableStopMatchChannel` (`CTIMER_Type *base`, `ctimer_match_t match`, bool `enable`)  
*Enable stop match channel.*
- static void `CTIMER_EnableMatchChannelReload` (`CTIMER_Type *base`, `ctimer_match_t match`, bool `enable`)  
*Enable reload channel falling edge.*
- static void `CTIMER_EnableRisingEdgeCapture` (`CTIMER_Type *base`, `ctimer_capture_channel_t capture`, bool `enable`)  
*Enable capture channel rising edge.*
- static void `CTIMER_EnableFallingEdgeCapture` (`CTIMER_Type *base`, `ctimer_capture_channel_t capture`, bool `enable`)

- Enable capture channel falling edge.*
- static void `CTIMER_SetShadowValue` (CTIMER\_Type \*base, `ctimer_match_t` match, uint32\_t matchvalue)

*Set the specified match shadow channel.*

## Driver version

- #define `FSL_CTIMER_DRIVER_VERSION` (MAKE\_VERSION(2, 3, 1))

*Version 2.3.1.*

## Initialization and deinitialization

- void `CTIMER_Init` (CTIMER\_Type \*base, const `ctimer_config_t` \*config)

*Ungates the clock and configures the peripheral for basic operation.*
- void `CTIMER_Deinit` (CTIMER\_Type \*base)

*Gates the timer clock.*
- void `CTIMER_GetDefaultConfig` (`ctimer_config_t` \*config)

*Fills in the timers configuration structure with the default settings.*

## PWM setup operations

- `status_t` `CTIMER_SetupPwmPeriod` (CTIMER\_Type \*base, const `ctimer_match_t` pwmPeriodChannel, `ctimer_match_t` matchChannel, uint32\_t pwmPeriod, uint32\_t pulsePeriod, bool enableInt)

*Configures the PWM signal parameters.*
- `status_t` `CTIMER_SetupPwm` (CTIMER\_Type \*base, const `ctimer_match_t` pwmPeriodChannel, `ctimer_match_t` matchChannel, uint8\_t dutyCyclePercent, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz, bool enableInt)

*Configures the PWM signal parameters.*
- static void `CTIMER_UpdatePwmPulsePeriod` (CTIMER\_Type \*base, `ctimer_match_t` matchChannel, uint32\_t pulsePeriod)

*Updates the pulse period of an active PWM signal.*
- void `CTIMER_UpdatePwmDutycycle` (CTIMER\_Type \*base, const `ctimer_match_t` pwmPeriodChannel, `ctimer_match_t` matchChannel, uint8\_t dutyCyclePercent)

*Updates the duty cycle of an active PWM signal.*

## Interrupt Interface

- static void `CTIMER_EnableInterrupts` (CTIMER\_Type \*base, uint32\_t mask)

*Enables the selected Timer interrupts.*
- static void `CTIMER_DisableInterrupts` (CTIMER\_Type \*base, uint32\_t mask)

*Disables the selected Timer interrupts.*
- static uint32\_t `CTIMER_GetEnabledInterrupts` (CTIMER\_Type \*base)

*Gets the enabled Timer interrupts.*

## Status Interface

- static uint32\_t `CTIMER_GetStatusFlags` (CTIMER\_Type \*base)

*Gets the Timer status flags.*

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

## Counter Start and Stop

- static void [CTIMER\\_StartTimer](#) (CTIMER\_Type \*base)  
*Starts the Timer counter.*
- static void [CTIMER\\_StopTimer](#) (CTIMER\_Type \*base)  
*Stops the Timer counter.*

## 9.4 Data Structure Documentation

### 9.4.1 struct \_ctimer\_match\_config

This structure holds the configuration settings for each match register.

#### Data Fields

- uint32\_t [matchValue](#)  
*This is stored in the match register.*
- bool [enableCounterReset](#)  
*true: Match will reset the counter false: Match will not reset the counter*
- bool [enableCounterStop](#)  
*true: Match will stop the counter false: Match will not stop the counter*
- [ctimer\\_match\\_output\\_control\\_t](#) [outControl](#)  
*Action to be taken on a match on the EM bit/output.*
- bool [outPinInitState](#)  
*Initial value of the EM bit/output.*
- bool [enableInterrupt](#)  
*true: Generate interrupt upon match false: Do not generate interrupt on match*

### 9.4.2 struct \_ctimer\_config

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the [CTIMER\\_GetDefaultConfig\(\)](#) function and pass a pointer to the configuration structure instance.

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

#### Data Fields

- [ctimer\\_timer\\_mode\\_t](#) [mode](#)  
*Timer mode.*
- [ctimer\\_capture\\_channel\\_t](#) [input](#)  
*Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC.*

- uint32\_t [prescale](#)  
Prescale value.

## 9.5 Typedef Documentation

### 9.5.1 typedef struct \_ctimer\_match\_config ctimer\_match\_config\_t

This structure holds the configuration settings for each match register.

### 9.5.2 typedef struct \_ctimer\_config ctimer\_config\_t

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the [CTIMER\\_GetDefaultConfig\(\)](#) function and pass a pointer to the configuration structure instance.

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

## 9.6 Enumeration Type Documentation

### 9.6.1 enum \_ctimer\_capture\_channel

Enumerator

- kCTIMER\_Capture\_0* Timer capture channel 0.
- kCTIMER\_Capture\_1* Timer capture channel 1.
- kCTIMER\_Capture\_2* Timer capture channel 2.

### 9.6.2 enum \_ctimer\_capture\_edge

Enumerator

- kCTIMER\_Capture\_RiseEdge* Capture on rising edge.
- kCTIMER\_Capture\_FallEdge* Capture on falling edge.
- kCTIMER\_Capture\_BothEdge* Capture on rising and falling edge.

### 9.6.3 enum \_ctimer\_match

Enumerator

- kCTIMER\_Match\_0* Timer match register 0.
- kCTIMER\_Match\_1* Timer match register 1.
- kCTIMER\_Match\_2* Timer match register 2.
- kCTIMER\_Match\_3* Timer match register 3.

### 9.6.4 enum\_ctimer\_external\_match

Enumerator

*kCTIMER\_External\_Match\_0* External match 0.  
*kCTIMER\_External\_Match\_1* External match 1.  
*kCTIMER\_External\_Match\_2* External match 2.  
*kCTIMER\_External\_Match\_3* External match 3.

### 9.6.5 enum\_ctimer\_match\_output\_control

Enumerator

*kCTIMER\_Output\_NoAction* No action is taken.  
*kCTIMER\_Output\_Clear* Clear the EM bit/output to 0.  
*kCTIMER\_Output\_Set* Set the EM bit/output to 1.  
*kCTIMER\_Output\_Toggle* Toggle the EM bit/output.

### 9.6.6 enum\_ctimer\_interrupt\_enable

Enumerator

*kCTIMER\_Match0InterruptEnable* Match 0 interrupt.  
*kCTIMER\_Match1InterruptEnable* Match 1 interrupt.  
*kCTIMER\_Match2InterruptEnable* Match 2 interrupt.  
*kCTIMER\_Match3InterruptEnable* Match 3 interrupt.  
*kCTIMER\_Capture0InterruptEnable* Capture 0 interrupt.  
*kCTIMER\_Capture1InterruptEnable* Capture 1 interrupt.  
*kCTIMER\_Capture2InterruptEnable* Capture 2 interrupt.

### 9.6.7 enum\_ctimer\_status\_flags

Enumerator

*kCTIMER\_Match0Flag* Match 0 interrupt flag.  
*kCTIMER\_Match1Flag* Match 1 interrupt flag.  
*kCTIMER\_Match2Flag* Match 2 interrupt flag.  
*kCTIMER\_Match3Flag* Match 3 interrupt flag.  
*kCTIMER\_Capture0Flag* Capture 0 interrupt flag.  
*kCTIMER\_Capture1Flag* Capture 1 interrupt flag.  
*kCTIMER\_Capture2Flag* Capture 2 interrupt flag.

## 9.6.8 enum `ctimer_callback_type_t`

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

Enumerator

***kCTIMER\_SingleCallback*** Single Callback type where there is only one callback for the timer. based on the status flags different channels needs to be handled differently

***kCTIMER\_MultipleCallback*** Multiple Callback type where there can be 8 valid callbacks, one per channel. for both match/capture

## 9.7 Function Documentation

### 9.7.1 void `CTIMER_Init ( CTIMER_Type * base, const ctimer_config_t * config )`

Note

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

Parameters

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

### 9.7.2 void `CTIMER_Deinit ( CTIMER_Type * base )`

Parameters

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

### 9.7.3 void `CTIMER_GetDefaultConfig ( ctimer_config_t * config )`

The default values are:

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

## Parameters

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

#### 9.7.4 **status\_t CTIMER\_SetupPwmPeriod ( CTIMER\_Type \* *base*, const ctimer\_match\_t *pwmPeriodChannel*, ctimer\_match\_t *matchChannel*, uint32\_t *pwmPeriod*, uint32\_t *pulsePeriod*, bool *enableInt* )**

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

## Note

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

## Parameters

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

#### 9.7.5 **status\_t CTIMER\_SetupPwm ( CTIMER\_Type \* *base*, const ctimer\_match\_t *pwmPeriodChannel*, ctimer\_match\_t *matchChannel*, uint8\_t *dutyCyclePercent*, uint32\_t *pwmFreq\_Hz*, uint32\_t *srcClock\_Hz*, bool *enableInt* )**

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

## Note

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

## Parameters

|                          |                                                                                                                                 |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | Ctimer peripheral base address                                                                                                  |
| <i>pwmPeriod-Channel</i> | Specify the channel to control the PWM period                                                                                   |
| <i>matchChannel</i>      | Match pin to be used to output the PWM signal                                                                                   |
| <i>dutyCycle-Percent</i> | PWM pulse width; the value should be between 0 to 100                                                                           |
| <i>pwmFreq_Hz</i>        | PWM signal frequency in Hz                                                                                                      |
| <i>srcClock_Hz</i>       | Timer counter clock in Hz                                                                                                       |
| <i>enableInt</i>         | Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt will be generated. |

**9.7.6 static void CTIMER\_UpdatePwmPulsePeriod ( CTIMER\_Type \* *base*, ctimer\_match\_t *matchChannel*, uint32\_t *pulsePeriod* ) [inline], [static]**

## Parameters

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>base</i>         | Ctimer peripheral base address                |
| <i>matchChannel</i> | Match pin to be used to output the PWM signal |
| <i>pulsePeriod</i>  | New PWM pulse width match value               |

**9.7.7 void CTIMER\_UpdatePwmDutycycle ( CTIMER\_Type \* *base*, const ctimer\_match\_t *pwmPeriodChannel*, ctimer\_match\_t *matchChannel*, uint8\_t *dutyCyclePercent* )**

## Note

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

## Parameters

|                          |                                                           |
|--------------------------|-----------------------------------------------------------|
| <i>base</i>              | Ctimer peripheral base address                            |
| <i>pwmPeriod-Channel</i> | Specify the channel to control the PWM period             |
| <i>matchChannel</i>      | Match pin to be used to output the PWM signal             |
| <i>dutyCycle-Percent</i> | New PWM pulse width; the value should be between 0 to 100 |

### 9.7.8 void CTIMER\_SetupMatch ( CTIMER\_Type \* *base*, ctimer\_match\_t *matchChannel*, const ctimer\_match\_config\_t \* *config* )

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

Parameters

|                     |                                              |
|---------------------|----------------------------------------------|
| <i>base</i>         | Ctimer peripheral base address               |
| <i>matchChannel</i> | Match register to configure                  |
| <i>config</i>       | Pointer to the match configuration structure |

### 9.7.9 uint32\_t CTIMER\_GetOutputMatchStatus ( CTIMER\_Type \* *base*, uint32\_t *matchChannel* )

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

Parameters

|                     |                                                                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | Ctimer peripheral base address                                                                                                                                               |
| <i>matchChannel</i> | External match channel, user can obtain the status of multiple match channels at the same time by using the logic of " " enumeration <a href="#">ctimer_external_match_t</a> |

Returns

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

### 9.7.10 void CTIMER\_SetupCapture ( CTIMER\_Type \* *base*, ctimer\_capture\_channel\_t *capture*, ctimer\_capture\_edge\_t *edge*, bool *enableInt* )

## Parameters

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

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

## Parameters

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

## Returns

return the timer count value.

### 9.7.12 void CTIMER\_RegisterCallback ( CTIMER\_Type \* *base*, ctimer\_callback\_t \* *cb\_func*, ctimer\_callback\_type\_t *cb\_type* )

## Parameters

|                |                                              |
|----------------|----------------------------------------------|
| <i>base</i>    | Ctimer peripheral base address               |
| <i>cb_func</i> | callback function                            |
| <i>cb_type</i> | callback function type, singular or multiple |

### 9.7.13 static void CTIMER\_EnableInterrupts ( CTIMER\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                        |
|-------------|------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Ctimer peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">ctimer-interrupt_enable_t</a> |

#### 9.7.14 **static void CTIMER\_DisableInterrupts ( CTIMER\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                                                        |
|-------------|------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Ctimer peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">ctimer-interrupt_enable_t</a> |

#### 9.7.15 **static uint32\_t CTIMER\_GetEnabledInterrupts ( CTIMER\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [ctimer\\_interrupt\\_enable\\_t](#)

#### 9.7.16 **static uint32\_t CTIMER\_GetStatusFlags ( CTIMER\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [ctimer\\_status\\_flags\\_t](#)

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

Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Ctimer peripheral base address                                                                                       |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">ctimer-_status_flags_t</a> |

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

Parameters

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

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

Parameters

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

### 9.7.20 static void CTIMER\_Reset ( CTIMER\_Type \* *base* ) [inline], [static]

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

Parameters

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

### 9.7.21 static void CTIMER\_SetPrescale ( CTIMER\_Type \* *base*, uint32\_t *prescale* ) [inline], [static]

Specifies the maximum value for the Prescale Counter.

Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>base</i>     | Ctimer peripheral base address |
| <i>prescale</i> | Prescale value                 |

### 9.7.22 **static uint32\_t CTIMER\_GetCaptureValue ( CTIMER\_Type \* *base*, ctimer\_capture\_channel\_t *capture* ) [inline], [static]**

Get the counter/timer value on the corresponding capture channel.

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | Ctimer peripheral base address |
| <i>capture</i> | Select capture channel         |

Returns

The timer count capture value.

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

Set the specified match channel reset operation.

Parameters

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

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

Set the specified match channel stop operation.

Parameters

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

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

Enable the specified match channel reload match shadow value.

Parameters

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

**9.7.26 static void CTIMER\_EnableRisingEdgeCapture ( CTIMER\_Type \* *base*,  
ctimer\_capture\_channel\_t *capture*, bool *enable* ) [inline], [static]**

Sets the specified capture channel for rising edge capture.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | Ctimer peripheral base address. |
| <i>capture</i> | capture channel used.           |
| <i>enable</i>  | Enable rising edge capture.     |

**9.7.27 static void CTIMER\_EnableFallingEdgeCapture ( CTIMER\_Type \* *base*,  
ctimer\_capture\_channel\_t *capture*, bool *enable* ) [inline], [static]**

Sets the specified capture channel for falling edge capture.

Parameters

---

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | Ctimer peripheral base address. |
| <i>capture</i> | capture channel used.           |
| <i>enable</i>  | Enable falling edge capture.    |

**9.7.28** `static void CTIMER_SetShadowValue ( CTIMER_Type * base,  
ctimer_match_t match, uint32_t matchvalue ) [inline], [static]`

Parameters

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

# Chapter 10

## IAP: In Application Programming Driver

### 10.1 Overview

The MCUXpresso SDK provides a driver for the In Application Programming (IAP) module of MCU-Xpresso SDK devices.

### 10.2 Function groups

The driver provides a set of functions to call the on-chip in application programming interface. User code executing from on-chip RAM can call these functions to read information like part id; read and write flash, EEPROM and FAIM.

#### 10.2.1 Basic operations

The function [IAP\\_ReadPartID\(\)](#) reads the part id of the board.

The function [IAP\\_ReadBootCodeVersion\(\)](#) reads the boot code Version.

The function [IAP\\_ReadUniqueID\(\)](#) reads the unique id of the boards.

The function [IAP\\_ReinvokeISP\(\)](#) reinvokes the ISP mode.

The function [IAP\\_ReadFactorySettings\(\)](#) reads the factory settings.

#### 10.2.2 Flash operations

The function [IAP\\_PrepareSectorForWrite\(\)](#) prepares a sector for write or erase operation. Then, the function [IAP\\_CopyRamToFlash\(\)](#) programs the flash memory.

The function [IAP\\_EraseSector\(\)](#) erases a flash sector while the function [IAP\\_ErasePage\(\)](#) erases a flash page.

The function [IAP\\_BlankCheckSector\(\)](#) is used to blank check a sector or multiple sectors of on-chip flash memory.

The function [IAP\\_Compare\(\)](#) is used to compare the memory contents at two locations. The user can compare several bytes (must be a multiple of 4) content in two different flash locations.

The function [IAP\\_ReadFlashSignature\(\)](#) can get the 32-bits signature of the entire flash and the function [IAP\\_ExtendedFlashSignatureRead\(\)](#) can calculate the signature of one or more flash pages.

### 10.2.3 EEPROM operations

The function `IAP_ReadEEPROMPage()` reads the 128 bytes content of an EEPROM page and `IAP_WriteEEPROMPage()` writes 128 bytes content in an EEPROM page

### 10.2.4 FAIM operations

The function `IAP_ReadEEPROMPage()` reads the 32 bits content of an FAIM page and `IAP_WriteEEPROMPage()` writes 32 bits content in an FAIM page

## 10.3 Typical use case

### 10.3.1 IAP Basic Operations

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

### 10.3.2 IAP Flash Operations

Refer to the driver example codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/iap/iap_flash/`

### 10.3.3 IAP EEPROM Operations

Refer to the driver example codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/iap/iap_eeprom/`

### 10.3.4 IAP FAIM Operations

Refer to the driver example codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/iap/iap_faim/`

## Files

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

## Enumerations

- enum {
  - kStatus\_IAP\_Success = kStatus\_Success,
  - kStatus\_IAP\_InvalidCommand = MAKE\_STATUS(kStatusGroup\_IAP, 1U),
  - kStatus\_IAP\_SrcAddrError = MAKE\_STATUS(kStatusGroup\_IAP, 2U),
  - kStatus\_IAP\_DstAddrError,
  - kStatus\_IAP\_SrcAddrNotMapped,
  - kStatus\_IAP\_DstAddrNotMapped,
  - kStatus\_IAP\_CountError,
  - kStatus\_IAP\_InvalidSector,
  - kStatus\_IAP\_SectorNotblank = MAKE\_STATUS(kStatusGroup\_IAP, 8U),
  - kStatus\_IAP\_NotPrepared,
  - kStatus\_IAP\_CompareError,
  - kStatus\_IAP\_Busy = MAKE\_STATUS(kStatusGroup\_IAP, 11U),
  - kStatus\_IAP\_ParamError,
  - kStatus\_IAP\_AddrError = MAKE\_STATUS(kStatusGroup\_IAP, 13U),
  - kStatus\_IAP\_AddrNotMapped = MAKE\_STATUS(kStatusGroup\_IAP, 14U),
  - kStatus\_IAP\_NoPower = MAKE\_STATUS(kStatusGroup\_IAP, 24U),
  - kStatus\_IAP\_NoClock = MAKE\_STATUS(kStatusGroup\_IAP, 27U),
  - kStatus\_IAP\_ReinvokeISPConfig = MAKE\_STATUS(kStatusGroup\_IAP, 0x1CU) }

*iap status codes.*
- enum \_iap\_commands {
  - kIapCmd\_IAP\_ReadFactorySettings = 40U,
  - kIapCmd\_IAP\_PrepareSectorforWrite = 50U,
  - kIapCmd\_IAP\_CopyRamToFlash = 51U,
  - kIapCmd\_IAP\_EraseSector = 52U,
  - kIapCmd\_IAP\_BlankCheckSector = 53U,
  - kIapCmd\_IAP\_ReadPartId = 54U,
  - kIapCmd\_IAP\_Read\_BootromVersion = 55U,
  - kIapCmd\_IAP\_Compare = 56U,
  - kIapCmd\_IAP\_ReinvokeISP = 57U,
  - kIapCmd\_IAP\_ReadUid = 58U,
  - kIapCmd\_IAP\_ErasePage = 59U,
  - kIapCmd\_IAP\_ReadSignature = 70U,
  - kIapCmd\_IAP\_ExtendedReadSignature = 73U,
  - kIapCmd\_IAP\_ReadEEPROMPage = 80U,
  - kIapCmd\_IAP\_WriteEEPROMPage = 81U }

*iap command codes.*
- enum \_flash\_access\_time { ,
  - kFlash\_IAP\_TwoSystemClockTime = 1U,
  - kFlash\_IAP\_ThreeSystemClockTime = 2U }

*Flash memory access time.*

## Driver version

- #define FSL\_IAP\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 7))

## Basic operations

- [status\\_t IAP\\_ReadPartID](#) (uint32\_t \*partID)  
*Read part identification number.*
- [status\\_t IAP\\_ReadBootCodeVersion](#) (uint32\_t \*bootCodeVersion)  
*Read boot code version number.*
- void [IAP\\_ReinvokeISP](#) (uint8\_t ispType, uint32\_t \*status)  
*Reinvoke ISP.*
- [status\\_t IAP\\_ReadUniqueID](#) (uint32\_t \*uniqueID)  
*Read unique identification.*

## Flash operations

- [status\\_t IAP\\_PrepareSectorForWrite](#) (uint32\_t startSector, uint32\_t endSector)  
*Prepare sector for write operation.*
- [status\\_t IAP\\_CopyRamToFlash](#) (uint32\_t dstAddr, uint32\_t \*srcAddr, uint32\_t numOfBytes, uint32\_t systemCoreClock)  
*Copy RAM to flash.*
- [status\\_t IAP\\_EraseSector](#) (uint32\_t startSector, uint32\_t endSector, uint32\_t systemCoreClock)  
*Erase sector.*
- [status\\_t IAP\\_ErasePage](#) (uint32\_t startPage, uint32\_t endPage, uint32\_t systemCoreClock)  
*Erase page.*
- [status\\_t IAP\\_BlankCheckSector](#) (uint32\_t startSector, uint32\_t endSector)  
*Blank check sector(s)*
- [status\\_t IAP\\_Compare](#) (uint32\_t dstAddr, uint32\_t \*srcAddr, uint32\_t numOfBytes)  
*Compare memory contents of flash with ram.*

## 10.4 Enumeration Type Documentation

### 10.4.1 anonymous enum

Enumerator

***kStatus\_IAP\_Success*** Api is executed successfully.

***kStatus\_IAP\_InvalidCommand*** Invalid command.

***kStatus\_IAP\_SrcAddrError*** Source address is not on word boundary.

***kStatus\_IAP\_DstAddrError*** Destination address is not on a correct boundary.

***kStatus\_IAP\_SrcAddrNotMapped*** Source address is not mapped in the memory map.

***kStatus\_IAP\_DstAddrNotMapped*** Destination address is not mapped in the memory map.

***kStatus\_IAP\_CountError*** Byte count is not multiple of 4 or is not a permitted value.

***kStatus\_IAP\_InvalidSector*** Sector/page number is invalid or end sector/page number is greater than start sector/page number.

***kStatus\_IAP\_SectorNotblank*** One or more sectors are not blank.

***kStatus\_IAP\_NotPrepared*** Command to prepare sector for write operation has not been executed.

***kStatus\_IAP\_CompareError*** Destination and source memory contents do not match.

***kStatus\_IAP\_Busy*** Flash programming hardware interface is busy.

***kStatus\_IAP\_ParamError*** Insufficient number of parameters or invalid parameter.

***kStatus\_IAP\_AddrError*** Address is not on word boundary.

***kStatus\_IAP\_AddrNotMapped*** Address is not mapped in the memory map.

*kStatus\_IAP\_NoPower* Flash memory block is powered down.  
*kStatus\_IAP\_NoClock* Flash memory block or controller is not clocked.  
*kStatus\_IAP\_ReinvokeISPConfig* Reinvoke configuration error.

## 10.4.2 enum\_iap\_commands

Enumerator

*kIapCmd\_IAP\_ReadFactorySettings* Read the factory settings.  
*kIapCmd\_IAP\_PrepareSectorforWrite* Prepare Sector for write.  
*kIapCmd\_IAP\_CopyRamToFlash* Copy RAM to flash.  
*kIapCmd\_IAP\_EraseSector* Erase Sector.  
*kIapCmd\_IAP\_BlankCheckSector* Blank check sector.  
*kIapCmd\_IAP\_ReadPartId* Read part id.  
*kIapCmd\_IAP\_Read\_BootromVersion* Read bootrom version.  
*kIapCmd\_IAP\_Compare* Compare.  
*kIapCmd\_IAP\_ReinvokeISP* Reinvoke ISP.  
*kIapCmd\_IAP\_ReadUid* Read Uid.  
*kIapCmd\_IAP\_ErasePage* Erase Page.  
*kIapCmd\_IAP\_ReadSignature* Read Signature.  
*kIapCmd\_IAP\_ExtendedReadSignature* Extended Read Signature.  
*kIapCmd\_IAP\_ReadEEPROMPage* Read EEPROM page.  
*kIapCmd\_IAP\_WriteEEPROMPage* Write EEPROM page.

## 10.4.3 enum\_flash\_access\_time

Enumerator

*kFlash\_IAP\_TwoSystemClockTime* 1 system clock flash access time  
*kFlash\_IAP\_ThreeSystemClockTime* 2 system clock flash access time

## 10.5 Function Documentation

### 10.5.1 status\_t IAP\_ReadPartID ( uint32\_t \* partID )

This function is used to read the part identification number.

Parameters

---

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>partID</i> | Address to store the part identification number. |
|---------------|--------------------------------------------------|

Return values

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>kStatus_IAP_Success</i> | Api has been executed successfully. |
|----------------------------|-------------------------------------|

### 10.5.2 status\_t IAP\_ReadBootCodeVersion ( uint32\_t \* bootCodeVersion )

This function is used to read the boot code version number.

Parameters

|                              |                                         |
|------------------------------|-----------------------------------------|
| <i>bootCode-<br/>Version</i> | Address to store the boot code version. |
|------------------------------|-----------------------------------------|

Return values

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>kStatus_IAP_Success</i> | Api has been executed successfully. |
|----------------------------|-------------------------------------|

note Boot code version is two 32-bit words. Word 0 is the major version, word 1 is the minor version.

### 10.5.3 void IAP\_ReinvokeISP ( uint8\_t ispType, uint32\_t \* status )

This function is used to invoke the boot loader in ISP mode. It maps boot vectors and configures the peripherals for ISP.

Parameters

|                |                            |
|----------------|----------------------------|
| <i>ispType</i> | ISP type selection.        |
| <i>status</i>  | store the possible status. |

Return values

|                                            |                               |
|--------------------------------------------|-------------------------------|
| <i>kStatus_IAP_ReinvokeIS-<br/>PConfig</i> | reinvoke configuration error. |
|--------------------------------------------|-------------------------------|

note The error response will be returned when IAP is disabled or an invalid ISP type selection appears. The call won't return unless an error occurs, so there can be no status code.

### 10.5.4 status\_t IAP\_ReadUniqueID ( uint32\_t \* uniqueID )

This function is used to read the unique id.

## Parameters

|                 |                     |
|-----------------|---------------------|
| <i>uniqueID</i> | store the uniqueID. |
|-----------------|---------------------|

## Return values

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>kStatus_IAP_Success</i> | Api has been executed successfully. |
|----------------------------|-------------------------------------|

### 10.5.5 **status\_t IAP\_PrepareSectorForWrite ( uint32\_t startSector, uint32\_t endSector )**

This function prepares sector(s) for write/erase operation. This function must be called before calling the [IAP\\_CopyRamToFlash\(\)](#) or [IAP\\_EraseSector\(\)](#) or [IAP\\_ErasePage\(\)](#) function. The end sector number must be greater than or equal to the start sector number.

## Parameters

|                    |                      |
|--------------------|----------------------|
| <i>startSector</i> | Start sector number. |
| <i>endSector</i>   | End sector number.   |

## Return values

|                                  |                                                                                    |
|----------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_IAP_Success</i>       | Api has been executed successfully.                                                |
| <i>kStatus_IAP_NoPower</i>       | Flash memory block is powered down.                                                |
| <i>kStatus_IAP_NoClock</i>       | Flash memory block or controller is not clocked.                                   |
| <i>kStatus_IAP_InvalidSector</i> | Sector number is invalid or end sector number is greater than start sector number. |
| <i>kStatus_IAP_Busy</i>          | Flash programming hardware interface is busy.                                      |

### 10.5.6 **status\_t IAP\_CopyRamToFlash ( uint32\_t dstAddr, uint32\_t \* srcAddr, uint32\_t numOfBytes, uint32\_t systemCoreClock )**

This function programs the flash memory. Corresponding sectors must be prepared via [IAP\\_PrepareSectorForWrite](#) before calling this function.

## Parameters

---

|                        |                                                                                                                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dstAddr</i>         | Destination flash address where data bytes are to be written, the address should be multiples of FSL_FEATURE_SYSCON_FLASH_PAGE_SIZE_BYTES boundary.                                                        |
| <i>srcAddr</i>         | Source ram address from where data bytes are to be read.                                                                                                                                                   |
| <i>numOfBytes</i>      | Number of bytes to be written, it should be multiples of FSL_FEATURE_SYSCON_FLASH_PAGE_SIZE_BYTES, and ranges from FSL_FEATURE_SYSCON_FLASH_PAGE_SIZE_BYTES to FSL_FEATURE_SYSCON_FLASH_SECTOR_SIZE_BYTES. |
| <i>systemCoreClock</i> | SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function. When the flash controller has a fixed reference clock, this parameter is bypassed.                                      |

## Return values

|                                      |                                                                      |
|--------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_IAP_Success</i>           | Api has been executed successfully.                                  |
| <i>kStatus_IAP_NoPower</i>           | Flash memory block is powered down.                                  |
| <i>kStatus_IAP_NoClock</i>           | Flash memory block or controller is not clocked.                     |
| <i>kStatus_IAP_SrcAddr-Error</i>     | Source address is not on word boundary.                              |
| <i>kStatus_IAP_DstAddr-Error</i>     | Destination address is not on a correct boundary.                    |
| <i>kStatus_IAP_SrcAddrNotMapped</i>  | Source address is not mapped in the memory map.                      |
| <i>kStatus_IAP_DstAddr-NotMapped</i> | Destination address is not mapped in the memory map.                 |
| <i>kStatus_IAP_CountError</i>        | Byte count is not multiple of 4 or is not a permitted value.         |
| <i>kStatus_IAP_NotPrepared</i>       | Command to prepare sector for write operation has not been executed. |
| <i>kStatus_IAP_Busy</i>              | Flash programming hardware interface is busy.                        |

### 10.5.7 **status\_t IAP\_EraseSector ( uint32\_t startSector, uint32\_t endSector, uint32\_t systemCoreClock )**

This function erases sector(s). The end sector number must be greater than or equal to the start sector number.

## Parameters

|                         |                                                                                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>startSector</i>      | Start sector number.                                                                                                                                                  |
| <i>endSector</i>        | End sector number.                                                                                                                                                    |
| <i>systemCore-Clock</i> | SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function. When the flash controller has a fixed reference clock, this parameter is bypassed. |

## Return values

|                                   |                                                                                    |
|-----------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_IAP_Success</i>        | Api has been executed successfully.                                                |
| <i>kStatus_IAP_NoPower</i>        | Flash memory block is powered down.                                                |
| <i>kStatus_IAP_NoClock</i>        | Flash memory block or controller is not clocked.                                   |
| <i>kStatus_IAP_Invalid-Sector</i> | Sector number is invalid or end sector number is greater than start sector number. |
| <i>kStatus_IAP_Not-Prepared</i>   | Command to prepare sector for write operation has not been executed.               |
| <i>kStatus_IAP_Busy</i>           | Flash programming hardware interface is busy.                                      |

### 10.5.8 **status\_t IAP\_ErasePage ( uint32\_t startPage, uint32\_t endPage, uint32\_t systemCoreClock )**

This function erases page(s). The end page number must be greater than or equal to the start page number.

## Parameters

|                         |                                                                                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>startPage</i>        | Start page number.                                                                                                                                                    |
| <i>endPage</i>          | End page number.                                                                                                                                                      |
| <i>systemCore-Clock</i> | SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function. When the flash controller has a fixed reference clock, this parameter is bypassed. |

## Return values

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>kStatus_IAP_Success</i> | Api has been executed successfully. |
| <i>kStatus_IAP_NoPower</i> | Flash memory block is powered down. |

|                                  |                                                                              |
|----------------------------------|------------------------------------------------------------------------------|
| <i>kStatus_IAP_NoClock</i>       | Flash memory block or controller is not clocked.                             |
| <i>kStatus_IAP_InvalidSector</i> | Page number is invalid or end page number is greater than start page number. |
| <i>kStatus_IAP_NotPrepared</i>   | Command to prepare sector for write operation has not been executed.         |
| <i>kStatus_IAP_Busy</i>          | Flash programming hardware interface is busy.                                |

### 10.5.9 status\_t IAP\_BlankCheckSector ( uint32\_t startSector, uint32\_t endSector )

Blank check single or multiples sectors of flash memory. The end sector number must be greater than or equal to the start sector number. It can be used to verify the sector erasure after IAP\_EraseSector call.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>startSector</i> | Start sector number. |
| <i>endSector</i>   | End sector number.   |

Return values

|                                   |                                                  |
|-----------------------------------|--------------------------------------------------|
| <i>kStatus_IAP_Success</i>        | One or more sectors are in erased state.         |
| <i>kStatus_IAP_NoPower</i>        | Flash memory block is powered down.              |
| <i>kStatus_IAP_NoClock</i>        | Flash memory block or controller is not clocked. |
| <i>kStatus_IAP_SectorNotblank</i> | One or more sectors are not blank.               |

### 10.5.10 status\_t IAP\_Compare ( uint32\_t dstAddr, uint32\_t \* srcAddr, uint32\_t numOfBytes )

This function compares the contents of flash and ram. It can be used to verify the flash memory contents after IAP\_CopyRamToFlash call.

Parameters

|                |                            |
|----------------|----------------------------|
| <i>dstAddr</i> | Destination flash address. |
|----------------|----------------------------|

|                   |                                 |
|-------------------|---------------------------------|
| <i>srcAddr</i>    | Source ram address.             |
| <i>numOfBytes</i> | Number of bytes to be compared. |

Return values

|                                   |                                                              |
|-----------------------------------|--------------------------------------------------------------|
| <i>kStatus_IAP_Success</i>        | Contents of flash and ram match.                             |
| <i>kStatus_IAP_NoPower</i>        | Flash memory block is powered down.                          |
| <i>kStatus_IAP_NoClock</i>        | Flash memory block or controller is not clocked.             |
| <i>kStatus_IAP_AddrError</i>      | Address is not on word boundary.                             |
| <i>kStatus_IAP_AddrNot-Mapped</i> | Address is not mapped in the memory map.                     |
| <i>kStatus_IAP_CountError</i>     | Byte count is not multiple of 4 or is not a permitted value. |
| <i>kStatus_IAP_Compare-Error</i>  | Destination and source memory contents do not match.         |

# Chapter 11

## LPC\_ACOMP: Analog comparator Driver

### 11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Analog comparator (LPC\_ACOMP) module of MCUXpresso SDK devices.

### 11.2 Typical use case

#### 11.2.1 Polling Configuration

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

#### 11.2.2 Interrupt Configuration

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

### Files

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

### Data Structures

- struct [\\_acomp\\_config](#)  
*The structure for ACOMP basic configuration. [More...](#)*
- struct [\\_acomp\\_ladder\\_config](#)  
*The structure for ACOMP voltage ladder. [More...](#)*

### Typedefs

- typedef enum  
[\\_acomp\\_ladder\\_reference\\_voltage](#) [acomp\\_ladder\\_reference\\_voltage\\_t](#)  
*The ACOMP ladder reference voltage.*
- typedef enum  
[\\_acomp\\_interrupt\\_enable](#) [acomp\\_interrupt\\_enable\\_t](#)  
*The ACOMP interrupts enable.*
- typedef enum  
[\\_acomp\\_hysteresis\\_selection](#) [acomp\\_hysteresis\\_selection\\_t](#)  
*The ACOMP hysteresis selection.*
- typedef struct [\\_acomp\\_config](#) [acomp\\_config\\_t](#)

*The structure for ACOMP basic configuration.*

- typedef struct `_acomp_ladder_config` `acomp_ladder_config_t`  
*The structure for ACOMP voltage ladder.*

## Enumerations

- enum `_acomp_ladder_reference_voltage` {  
  `kACOMP_LadderRefVoltagePinVDD` = 0U,  
  `kACOMP_LadderRefVoltagePinVDDCMP` = 1U }  
*The ACOMP ladder reference voltage.*
- enum `_acomp_interrupt_enable` {  
  `kACOMP_InterruptsFallingEdgeEnable` = 0U,  
  `kACOMP_InterruptsRisingEdgeEnable` = 1U,  
  `kACOMP_InterruptsBothEdgesEnable` = 2U,  
  `kACOMP_InterruptsDisable` = 3U }  
*The ACOMP interrupts enable.*
- enum `_acomp_hysteresis_selection` {  
  `kACOMP_HysteresisNoneSelection` = 0U,  
  `kACOMP_Hysteresis5MVSelection` = 1U,  
  `kACOMP_Hysteresis10MVSelection` = 2U,  
  `kACOMP_Hysteresis20MVSelection` = 3U }  
*The ACOMP hysteresis selection.*

## Driver version

- #define `FSL_ACOMP_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 0)`)  
*ACOMP driver version 2.1.0.*

## Initialization

- void `ACOMP_Init` (`ACOMP_Type *base`, const `acomp_config_t *config`)  
*Initialize the ACOMP module.*
- void `ACOMP_Deinit` (`ACOMP_Type *base`)  
*De-initialize the ACOMP module.*
- void `ACOMP_GetDefaultConfig` (`acomp_config_t *config`)  
*Gets an available pre-defined settings for the ACOMP's configuration.*
- void `ACOMP_EnableInterrupts` (`ACOMP_Type *base`, `acomp_interrupt_enable_t enable`)  
*Enable ACOMP interrupts.*
- static bool `ACOMP_GetInterruptsStatusFlags` (`ACOMP_Type *base`)  
*Get interrupts status flags.*
- static void `ACOMP_ClearInterruptsStatusFlags` (`ACOMP_Type *base`)  
*Clear the ACOMP interrupts status flags.*
- static bool `ACOMP_GetOutputStatusFlags` (`ACOMP_Type *base`)  
*Get ACOMP output status flags.*
- static void `ACOMP_SetInputChannel` (`ACOMP_Type *base`, `uint32_t positiveInputChannel`, `uint32_t negativeInputChannel`)  
*Set the ACOMP positive and negative input channel.*
- void `ACOMP_SetLadderConfig` (`ACOMP_Type *base`, const `acomp_ladder_config_t *config`)  
*Set the voltage ladder configuration.*

## 11.3 Data Structure Documentation

### 11.3.1 struct \_acomp\_config

#### Data Fields

- bool [enableSyncToBusClk](#)  
*If true, Comparator output is synchronized to the bus clock for output to other modules.*
- [acomp\\_hysteresis\\_selection\\_t hysteresisSelection](#)  
*Controls the hysteresis of the comparator.*

#### Field Documentation

##### (1) bool \_acomp\_config::enableSyncToBusClk

If false, Comparator output is used directly.

##### (2) [acomp\\_hysteresis\\_selection\\_t \\_acomp\\_config::hysteresisSelection](#)

### 11.3.2 struct \_acomp\_ladder\_config

#### Data Fields

- uint8\_t [ladderValue](#)  
*Voltage ladder value.*
- [acomp\\_ladder\\_reference\\_voltage\\_t referenceVoltage](#)  
*Selects the reference voltage( $V_{ref}$ ) for the voltage ladder.*

#### Field Documentation

##### (1) uint8\_t \_acomp\_ladder\_config::ladderValue

00000 =  $V_{ss}$ , 00001 =  $1 * V_{ref} / 31$ , ..., 11111 =  $V_{ref}$ .

##### (2) [acomp\\_ladder\\_reference\\_voltage\\_t \\_acomp\\_ladder\\_config::referenceVoltage](#)

## 11.4 Macro Definition Documentation

### 11.4.1 #define FSL\_ACOMP\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

## 11.5 Enumeration Type Documentation

### 11.5.1 enum \_acomp\_ladder\_reference\_voltage

Enumerator

*kACOMP\_LadderRefVoltagePinVDD* Supply from pin VDD.

*kACOMP\_LadderRefVoltagePinVDDCMP* Supply from pin VDDCMP.

## 11.5.2 enum \_acomp\_interrupt\_enable

Enumerator

***kACOMP\_InterruptsFallingEdgeEnable*** Enable the falling edge interrupts.

***kACOMP\_InterruptsRisingEdgeEnable*** Enable the rising edge interrupts.

***kACOMP\_InterruptsBothEdgesEnable*** Enable the both edges interrupts.

***kACOMP\_InterruptsDisable*** Disable the interrupts.

## 11.5.3 enum \_acomp\_hysteresis\_selection

Enumerator

***kACOMP\_HysteresisNoneSelection*** None (the output will switch as the voltages cross).

***kACOMP\_Hysteresis5MVSelection*** 5mV.

***kACOMP\_Hysteresis10MVSelection*** 10mV.

***kACOMP\_Hysteresis20MVSelection*** 20mV.

## 11.6 Function Documentation

### 11.6.1 void ACOMP\_Init ( ACOMP\_Type \* *base*, const acomp\_config\_t \* *config* )

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | ACOMP peripheral base address.         |
| <i>config</i> | Pointer to "acomp_config_t" structure. |

### 11.6.2 void ACOMP\_Deinit ( ACOMP\_Type \* *base* )

Parameters

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

### 11.6.3 void ACOMP\_GetDefaultConfig ( acomp\_config\_t \* *config* )

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

```
* config->enableSyncToBusClk = false;
* config->hysteresisSelection = kACOMP_hysteresisNoneSelection;
*
```

In default configuration, the ACOMP's output would be used directly and switch as the voltages cross.

Parameters

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

**11.6.4 void ACOMP\_EnableInterrupts ( ACOMP\_Type \* *base*,  
acomp\_interrupt\_enable\_t *enable* )**

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | ACOMP peripheral base address.    |
| <i>enable</i> | Enable/Disable interrupt feature. |

**11.6.5 static bool ACOMP\_GetInterruptsStatusFlags ( ACOMP\_Type \* *base* )  
[inline], [static]**

Parameters

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

Returns

Reflect the state ACOMP edge-detect status, true or false.

**11.6.6 static void ACOMP\_ClearInterruptsStatusFlags ( ACOMP\_Type \* *base* )  
[inline], [static]**

Parameters

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

**11.6.7 static bool ACOMP\_GetOutputStatusFlags ( ACOMP\_Type \* *base* )  
[inline], [static]**

## Parameters

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

## Returns

Reflect the state of the comparator output, true or false.

### 11.6.8 static void ACOMP\_SetInputChannel ( ACOMP\_Type \* *base*, uint32\_t *positiveInputChannel*, uint32\_t *negativeInputChannel* ) [inline], [static]

## Parameters

|                             |                                      |
|-----------------------------|--------------------------------------|
| <i>base</i>                 | ACOMP peripheral base address.       |
| <i>positiveInputChannel</i> | The index of positive input channel. |
| <i>negativeInputChannel</i> | The index of negative input channel. |

### 11.6.9 void ACOMP\_SetLadderConfig ( ACOMP\_Type \* *base*, const *acomp\_ladder\_config\_t* \* *config* )

## Parameters

|               |                                                                                                                                                          |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | ACOMP peripheral base address.                                                                                                                           |
| <i>config</i> | The structure for voltage ladder. If the config is NULL, voltage ladder would be disabled, otherwise the voltage ladder would be configured and enabled. |

# Chapter 12

## ADC: 12-bit SAR Analog-to-Digital Converter Driver

### 12.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit Successive Approximation (SAR) Analog-to-Digital Converter (ADC) module of MCUXpresso SDK devices.

### 12.2 Typical use case

#### 12.2.1 Polling Configuration

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

#### 12.2.2 Interrupt Configuration

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

### Files

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

### Data Structures

- struct [\\_adc\\_config](#)  
*Define structure for configuring the block. [More...](#)*
- struct [\\_adc\\_conv\\_seq\\_config](#)  
*Define structure for configuring conversion sequence. [More...](#)*
- struct [\\_adc\\_result\\_info](#)  
*Define structure of keeping conversion result information. [More...](#)*

### Typedefs

- typedef enum [\\_adc\\_trigger\\_polarity](#) [adc\\_trigger\\_polarity\\_t](#)  
*Define selection of polarity of selected input trigger for conversion sequence.*
- typedef enum [\\_adc\\_priority](#) [adc\\_priority\\_t](#)  
*Define selection of conversion sequence's priority.*
- typedef enum [\\_adc\\_seq\\_interrupt\\_mode](#) [adc\\_seq\\_interrupt\\_mode\\_t](#)  
*Define selection of conversion sequence's interrupt.*
- typedef enum [\\_adc\\_threshold\\_compare\\_status](#) [adc\\_threshold\\_compare\\_status\\_t](#)

- *Define status of threshold compare result.*  
typedef enum  
[\\_adc\\_threshold\\_crossing\\_status](#) [adc\\_threshold\\_crossing\\_status\\_t](#)  
*Define status of threshold crossing detection result.*
- typedef enum  
[\\_adc\\_threshold\\_interrupt\\_mode](#) [adc\\_threshold\\_interrupt\\_mode\\_t](#)  
*Define interrupt mode for threshold compare event.*
- typedef enum [\\_adc\\_inforeresultshift](#) [adc\\_inforeresult\\_t](#)  
*Define the info result mode of different resolution.*
- typedef enum  
[\\_adc\\_tempsensor\\_common\\_mode](#) [adc\\_tempsensor\\_common\\_mode\\_t](#)  
*Define common modes for Temperature sensor.*
- typedef enum [\\_adc\\_second\\_control](#) [adc\\_second\\_control\\_t](#)  
*Define source impedance modes for GPADC control.*
- typedef struct [\\_adc\\_config](#) [adc\\_config\\_t](#)  
*Define structure for configuring the block.*
- typedef struct [\\_adc\\_conv\\_seq\\_config](#) [adc\\_conv\\_seq\\_config\\_t](#)  
*Define structure for configuring conversion sequence.*
- typedef struct [\\_adc\\_result\\_info](#) [adc\\_result\\_info\\_t](#)  
*Define structure of keeping conversion result information.*

## Enumerations

- enum `_adc_status_flags` {
  - `kADC_ThresholdCompareFlagOnChn0` = 1U << 0U,
  - `kADC_ThresholdCompareFlagOnChn1` = 1U << 1U,
  - `kADC_ThresholdCompareFlagOnChn2` = 1U << 2U,
  - `kADC_ThresholdCompareFlagOnChn3` = 1U << 3U,
  - `kADC_ThresholdCompareFlagOnChn4` = 1U << 4U,
  - `kADC_ThresholdCompareFlagOnChn5` = 1U << 5U,
  - `kADC_ThresholdCompareFlagOnChn6` = 1U << 6U,
  - `kADC_ThresholdCompareFlagOnChn7` = 1U << 7U,
  - `kADC_ThresholdCompareFlagOnChn8` = 1U << 8U,
  - `kADC_ThresholdCompareFlagOnChn9` = 1U << 9U,
  - `kADC_ThresholdCompareFlagOnChn10` = 1U << 10U,
  - `kADC_ThresholdCompareFlagOnChn11` = 1U << 11U,
  - `kADC_OverrunFlagForChn0`,
  - `kADC_OverrunFlagForChn1`,
  - `kADC_OverrunFlagForChn2`,
  - `kADC_OverrunFlagForChn3`,
  - `kADC_OverrunFlagForChn4`,
  - `kADC_OverrunFlagForChn5`,
  - `kADC_OverrunFlagForChn6`,
  - `kADC_OverrunFlagForChn7`,
  - `kADC_OverrunFlagForChn8`,
  - `kADC_OverrunFlagForChn9`,
  - `kADC_OverrunFlagForChn10`,
  - `kADC_OverrunFlagForChn11`,
  - `kADC_GlobalOverrunFlagForSeqA` = 1U << 24U,
  - `kADC_GlobalOverrunFlagForSeqB` = 1U << 25U,
  - `kADC_ConvSeqAInterruptFlag` = 1U << 28U,
  - `kADC_ConvSeqBInterruptFlag` = 1U << 29U,
  - `kADC_ThresholdCompareInterruptFlag` = 1U << 30U,
  - `kADC_OverrunInterruptFlag` = (int)(1U << 31U) }

*Flags.*
- enum `_adc_interrupt_enable` {
  - `kADC_ConvSeqAInterruptEnable` = ADC\_INTEN\_SEQA\_INTEN\_MASK,
  - `kADC_ConvSeqBInterruptEnable` = ADC\_INTEN\_SEQB\_INTEN\_MASK,
  - `kADC_OverrunInterruptEnable` = ADC\_INTEN\_OVR\_INTEN\_MASK }

*Interrupts.*
- enum `_adc_trigger_polarity` {
  - `kADC_TriggerPolarityNegativeEdge` = 0U,
  - `kADC_TriggerPolarityPositiveEdge` = 1U }

*Define selection of polarity of selected input trigger for conversion sequence.*
- enum `_adc_priority` {
  - `kADC_PriorityLow` = 0U,
  - `kADC_PriorityHigh` = 1U }

- Define selection of conversion sequence's priority.*

  - enum `_adc_seq_interrupt_mode` {  
`kADC_InterruptForEachConversion` = 0U,  
`kADC_InterruptForEachSequence` = 1U }

*Define selection of conversion sequence's interrupt.*

  - enum `_adc_threshold_compare_status` {  
`kADC_ThresholdCompareInRange` = 0U,  
`kADC_ThresholdCompareBelowRange` = 1U,  
`kADC_ThresholdCompareAboveRange` = 2U }

*Define status of threshold compare result.*

  - enum `_adc_threshold_crossing_status` {  
`kADC_ThresholdCrossingNoDetected` = 0U,  
`kADC_ThresholdCrossingDownward` = 2U,  
`kADC_ThresholdCrossingUpward` = 3U }

*Define status of threshold crossing detection result.*

  - enum `_adc_threshold_interrupt_mode` {  
`kADC_ThresholdInterruptDisabled` = 0U,  
`kADC_ThresholdInterruptOnOutside` = 1U,  
`kADC_ThresholdInterruptOnCrossing` = 2U }

*Define interrupt mode for threshold compare event.*

  - enum `_adc_inforesultshift` {  
`kADC_Resolution12bitInfoResultShift` = 0U,  
`kADC_Resolution10bitInfoResultShift` = 2U,  
`kADC_Resolution8bitInfoResultShift` = 4U,  
`kADC_Resolution6bitInfoResultShift` = 6U }

*Define the info result mode of different resolution.*

  - enum `_adc_tempsensor_common_mode` {  
`kADC_HighNegativeOffsetAdded` = 0x0U,  
`kADC_IntermediateNegativeOffsetAdded`,  
`kADC_NoOffsetAdded` = 0x8U,  
`kADC_LowPositiveOffsetAdded` = 0xcU }

*Define common modes for Temperature sensor.*

  - enum `_adc_second_control` {  
`kADC_Impedance621Ohm` = 0x1U << 9U,  
`kADC_Impedance55kOhm`,  
`kADC_Impedance87kOhm` = 0x1fU << 9U,  
`kADC_NormalFunctionalMode` = 0x0U << 14U,  
`kADC_MultiplexeTestMode` = 0x1U << 14U,  
`kADC_ADCInUnityGainMode` = 0x2U << 14U }

*Define source impedance modes for GPADC control.*

## Driver version

- #define `FSL_ADC_DRIVER_VERSION` (`MAKE_VERSION(2, 5,3)`)  
ADC driver version 2.5.3.

## Initialization and Deinitialization

- void `ADC_Init` (ADC\_Type \*base, const `adc_config_t` \*config)  
*Initialize the ADC module.*
- void `ADC_Deinit` (ADC\_Type \*base)  
*Deinitialize the ADC module.*
- void `ADC_GetDefaultConfig` (`adc_config_t` \*config)  
*Gets an available pre-defined settings for initial configuration.*

## Control conversion sequence A.

- static void `ADC_EnableConvSeqA` (ADC\_Type \*base, bool enable)  
*Enable the conversion sequence A.*
- void `ADC_SetConvSeqAConfig` (ADC\_Type \*base, const `adc_conv_seq_config_t` \*config)  
*Configure the conversion sequence A.*
- static void `ADC_DoSoftwareTriggerConvSeqA` (ADC\_Type \*base)  
*Do trigger the sequence's conversion by software.*
- static void `ADC_EnableConvSeqABurstMode` (ADC\_Type \*base, bool enable)  
*Enable the burst conversion of sequence A.*
- static void `ADC_SetConvSeqAHighPriority` (ADC\_Type \*base)  
*Set the high priority for conversion sequence A.*

## Control conversion sequence B.

- static void `ADC_EnableConvSeqB` (ADC\_Type \*base, bool enable)  
*Enable the conversion sequence B.*
- void `ADC_SetConvSeqBConfig` (ADC\_Type \*base, const `adc_conv_seq_config_t` \*config)  
*Configure the conversion sequence B.*
- static void `ADC_DoSoftwareTriggerConvSeqB` (ADC\_Type \*base)  
*Do trigger the sequence's conversion by software.*
- static void `ADC_EnableConvSeqBBurstMode` (ADC\_Type \*base, bool enable)  
*Enable the burst conversion of sequence B.*
- static void `ADC_SetConvSeqBHighPriority` (ADC\_Type \*base)  
*Set the high priority for conversion sequence B.*

## Data result.

- bool `ADC_GetConvSeqAGlobalConversionResult` (ADC\_Type \*base, `adc_result_info_t` \*info)  
*Get the global ADC conversion information of sequence A.*
- bool `ADC_GetConvSeqBGlobalConversionResult` (ADC\_Type \*base, `adc_result_info_t` \*info)  
*Get the global ADC conversion information of sequence B.*
- bool `ADC_GetChannelConversionResult` (ADC\_Type \*base, uint32\_t channel, `adc_result_info_t` \*info)  
*Get the channel's ADC conversion completed under each conversion sequence.*

## Threshold function.

- static void `ADC_SetThresholdPair0` (ADC\_Type \*base, uint32\_t lowValue, uint32\_t highValue)  
*Set the threshold pair 0 with low and high value.*
- static void `ADC_SetThresholdPair1` (ADC\_Type \*base, uint32\_t lowValue, uint32\_t highValue)  
*Set the threshold pair 1 with low and high value.*

- static void [ADC\\_SetChannelWithThresholdPair0](#) (ADC\_Type \*base, uint32\_t channelMask)  
*Set given channels to apply the threshold pare 0.*
- static void [ADC\\_SetChannelWithThresholdPair1](#) (ADC\_Type \*base, uint32\_t channelMask)  
*Set given channels to apply the threshold pare 1.*

## Interrupts.

- static void [ADC\\_EnableInterrupts](#) (ADC\_Type \*base, uint32\_t mask)  
*Enable interrupts for conversion sequences.*
- static void [ADC\\_DisableInterrupts](#) (ADC\_Type \*base, uint32\_t mask)  
*Disable interrupts for conversion sequence.*
- static void [ADC\\_EnableThresholdCompareInterrupt](#) (ADC\_Type \*base, uint32\_t channel, [adc\\_threshold\\_interrupt\\_mode\\_t](#) mode)  
*Enable the interrupt of threshold compare event for each channel.*

## Status.

- static uint32\_t [ADC\\_GetStatusFlags](#) (ADC\_Type \*base)  
*Get status flags of ADC module.*
- static void [ADC\\_ClearStatusFlags](#) (ADC\_Type \*base, uint32\_t mask)  
*Clear status flags of ADC module.*

## 12.3 Data Structure Documentation

### 12.3.1 struct \_adc\_config

#### Data Fields

- uint32\_t [clockDividerNumber](#)  
*This field is only available when using kADC\_ClockSynchronousMode for "clockMode" field.*
- bool [enableLowPowerMode](#)  
*If disable low-power mode, ADC remains activated even when no conversions are requested.*

#### Field Documentation

##### (1) uint32\_t \_adc\_config::clockDividerNumber

The divider would be plused by 1 based on the value in this field. The available range is in 8 bits.

##### (2) bool \_adc\_config::enableLowPowerMode

If enable low-power mode, The ADC is automatically powered-down when no conversions are taking place.

### 12.3.2 struct \_adc\_conv\_seq\_config

#### Data Fields

- uint32\_t [channelMask](#)  
Selects which one or more of the ADC channels will be sampled and converted when the conversion sequence is launched.
- uint32\_t [triggerMask](#)  
Selects which one or more of the available hardware trigger sources will be used to initiate the conversion sequence to be initiated.
- [adc\\_trigger\\_polarity\\_t](#) [triggerPolarity](#)  
Select the trigger to launch conversion sequence.
- bool [enableSyncBypass](#)  
To enable this feature allows the hardware trigger input to bypass synchronous flip-flop stages and therefore shorten the time between the trigger input signal and the start of a conversion.
- bool [enableSingleStep](#)  
When enabling this feature, a trigger will launch a single conversion on the selected channel in the sequence instead of the default response of launching an entire sequence of conversions.
- [adc\\_seq\\_interrupt\\_mode\\_t](#) [interruptMode](#)  
Select the interrupt/DMA trigger mode.

#### Field Documentation

##### (1) uint32\_t \_adc\_conv\_seq\_config::channelMask

The masked channels would be involved in current conversion sequence, beginning with the lowest-order. The available range is in 12-bit.

##### (2) uint32\_t \_adc\_conv\_seq\_config::triggerMask

The available range is 6-bit.

##### (3) [adc\\_trigger\\_polarity\\_t](#) \_adc\_conv\_seq\_config::triggerPolarity

##### (4) bool \_adc\_conv\_seq\_config::enableSyncBypass

##### (5) bool \_adc\_conv\_seq\_config::enableSingleStep

##### (6) [adc\\_seq\\_interrupt\\_mode\\_t](#) \_adc\_conv\_seq\_config::interruptMode

### 12.3.3 struct \_adc\_result\_info

#### Data Fields

- uint32\_t [result](#)  
Keep the conversion data value.
- [adc\\_threshold\\_compare\\_status\\_t](#) [thresholdCompareStatus](#)  
Keep the threshold compare status.
- [adc\\_threshold\\_crossing\\_status\\_t](#) [thresholdCorssingStatus](#)

- *Keep the threshold crossing status.*  
uint32\_t `channelNumber`  
*Keep the channel number for this conversion.*
- bool `overrunFlag`  
*Keep the status whether the conversion is overrun or not.*

### Field Documentation

- (1) `uint32_t _adc_result_info::result`
- (2) `adc_threshold_compare_status_t _adc_result_info::thresholdCompareStatus`
- (3) `adc_threshold_crossing_status_t _adc_result_info::thresholdCorssingStatus`
- (4) `uint32_t _adc_result_info::channelNumber`
- (5) `bool _adc_result_info::overrunFlag`

## 12.4 Macro Definition Documentation

### 12.4.1 #define FSL\_ADC\_DRIVER\_VERSION (MAKE\_VERSION(2, 5,3))

## 12.5 Enumeration Type Documentation

### 12.5.1 enum \_adc\_status\_flags

#### Enumerator

- |                                         |                                                                            |
|-----------------------------------------|----------------------------------------------------------------------------|
| <i>kADC_ThresholdCompareFlagOnChn0</i>  | Threshold comparison event on Channel 0.                                   |
| <i>kADC_ThresholdCompareFlagOnChn1</i>  | Threshold comparison event on Channel 1.                                   |
| <i>kADC_ThresholdCompareFlagOnChn2</i>  | Threshold comparison event on Channel 2.                                   |
| <i>kADC_ThresholdCompareFlagOnChn3</i>  | Threshold comparison event on Channel 3.                                   |
| <i>kADC_ThresholdCompareFlagOnChn4</i>  | Threshold comparison event on Channel 4.                                   |
| <i>kADC_ThresholdCompareFlagOnChn5</i>  | Threshold comparison event on Channel 5.                                   |
| <i>kADC_ThresholdCompareFlagOnChn6</i>  | Threshold comparison event on Channel 6.                                   |
| <i>kADC_ThresholdCompareFlagOnChn7</i>  | Threshold comparison event on Channel 7.                                   |
| <i>kADC_ThresholdCompareFlagOnChn8</i>  | Threshold comparison event on Channel 8.                                   |
| <i>kADC_ThresholdCompareFlagOnChn9</i>  | Threshold comparison event on Channel 9.                                   |
| <i>kADC_ThresholdCompareFlagOnChn10</i> | Threshold comparison event on Channel 10.                                  |
| <i>kADC_ThresholdCompareFlagOnChn11</i> | Threshold comparison event on Channel 11.                                  |
| <i>kADC_OverrunFlagForChn0</i>          | Mirror the OVERRUN status flag from the result register for ADC channel 0. |
| <i>kADC_OverrunFlagForChn1</i>          | Mirror the OVERRUN status flag from the result register for ADC channel 1. |
| <i>kADC_OverrunFlagForChn2</i>          | Mirror the OVERRUN status flag from the result register for ADC channel 2. |
| <i>kADC_OverrunFlagForChn3</i>          | Mirror the OVERRUN status flag from the result register for ADC channel 3. |

- kADC\_OvrrunFlagForChn4*** Mirror the OVERRUN status flag from the result register for ADC channel 4.
- kADC\_OvrrunFlagForChn5*** Mirror the OVERRUN status flag from the result register for ADC channel 5.
- kADC\_OvrrunFlagForChn6*** Mirror the OVERRUN status flag from the result register for ADC channel 6.
- kADC\_OvrrunFlagForChn7*** Mirror the OVERRUN status flag from the result register for ADC channel 7.
- kADC\_OvrrunFlagForChn8*** Mirror the OVERRUN status flag from the result register for ADC channel 8.
- kADC\_OvrrunFlagForChn9*** Mirror the OVERRUN status flag from the result register for ADC channel 9.
- kADC\_OvrrunFlagForChn10*** Mirror the OVERRUN status flag from the result register for ADC channel 10.
- kADC\_OvrrunFlagForChn11*** Mirror the OVERRUN status flag from the result register for ADC channel 11.
- kADC\_GlobalOvrrunFlagForSeqA*** Mirror the global OVERRUN status flag for conversion sequence A.
- kADC\_GlobalOvrrunFlagForSeqB*** Mirror the global OVERRUN status flag for conversion sequence B.
- kADC\_ConvSeqAInterruptFlag*** Sequence A interrupt/DMA trigger.
- kADC\_ConvSeqBInterruptFlag*** Sequence B interrupt/DMA trigger.
- kADC\_ThresholdCompareInterruptFlag*** Threshold comparison interrupt flag.
- kADC\_OvrrunInterruptFlag*** Overrun interrupt flag.

### 12.5.2 enum \_adc\_interrupt\_enable

Note

Not all the interrupt options are listed here

Enumerator

- kADC\_ConvSeqAInterruptEnable*** Enable interrupt upon completion of each individual conversion in sequence A, or entire sequence.
- kADC\_ConvSeqBInterruptEnable*** Enable interrupt upon completion of each individual conversion in sequence B, or entire sequence.
- kADC\_OvrrunInterruptEnable*** Enable the detection of an overrun condition on any of the channel data registers will cause an overrun interrupt/DMA trigger.

### 12.5.3 enum \_adc\_trigger\_polarity

Enumerator

*kADC\_TriggerPolarityNegativeEdge* A negative edge launches the conversion sequence on the trigger(s).

*kADC\_TriggerPolarityPositiveEdge* A positive edge launches the conversion sequence on the trigger(s).

### 12.5.4 enum \_adc\_priority

Enumerator

*kADC\_PriorityLow* This sequence would be preempted when another sequence is started.

*kADC\_PriorityHigh* This sequence would preempt other sequence even when it is started.

### 12.5.5 enum \_adc\_seq\_interrupt\_mode

Enumerator

*kADC\_InterruptForEachConversion* The sequence interrupt/DMA trigger will be set at the end of each individual ADC conversion inside this conversion sequence.

*kADC\_InterruptForEachSequence* The sequence interrupt/DMA trigger will be set when the entire set of this sequence conversions completes.

### 12.5.6 enum \_adc\_threshold\_compare\_status

Enumerator

*kADC\_ThresholdCompareInRange* LOW threshold  $\leq$  conversion value  $\leq$  HIGH threshold.

*kADC\_ThresholdCompareBelowRange* conversion value  $<$  LOW threshold.

*kADC\_ThresholdCompareAboveRange* conversion value  $>$  HIGH threshold.

### 12.5.7 enum \_adc\_threshold\_crossing\_status

Enumerator

*kADC\_ThresholdCrossingNoDetected* No threshold Crossing detected.

*kADC\_ThresholdCrossingDownward* Downward Threshold Crossing detected.

*kADC\_ThresholdCrossingUpward* Upward Threshold Crossing Detected.

### 12.5.8 enum \_adc\_threshold\_interrupt\_mode

Enumerator

- kADC\_ThresholdInterruptDisabled* Threshold comparison interrupt is disabled.
- kADC\_ThresholdInterruptOnOutside* Threshold comparison interrupt is enabled on outside threshold.
- kADC\_ThresholdInterruptOnCrossing* Threshold comparison interrupt is enabled on crossing threshold.

### 12.5.9 enum \_adc\_inforesultshift

Enumerator

- kADC\_Resolution12bitInfoResultShift* Info result shift of Resolution12bit.
- kADC\_Resolution10bitInfoResultShift* Info result shift of Resolution10bit.
- kADC\_Resolution8bitInfoResultShift* Info result shift of Resolution8bit.
- kADC\_Resolution6bitInfoResultShift* Info result shift of Resolution6bit.

### 12.5.10 enum \_adc\_tempsensor\_common\_mode

Enumerator

- kADC\_HighNegativeOffsetAdded* Temperature sensor common mode: high negative offset added.
- kADC\_IntermediateNegativeOffsetAdded* Temperature sensor common mode: intermediate negative offset added.
- kADC\_NoOffsetAdded* Temperature sensor common mode: no offset added.
- kADC\_LowPositiveOffsetAdded* Temperature sensor common mode: low positive offset added.

### 12.5.11 enum \_adc\_second\_control

Enumerator

- kADC\_Impedance6210hm* Extand ADC sampling time according to source impedance 1: 0.621 kOhm.
- kADC\_Impedance55kOhm* Extand ADC sampling time according to source impedance 20 (default): 55 kOhm.
- kADC\_Impedance87kOhm* Extand ADC sampling time according to source impedance 31: 87 k-Ohm.
- kADC\_NormalFunctionalMode* TEST mode: Normal functional mode.
- kADC\_MultiplexeTestMode* TEST mode: Multiplexer test mode.
- kADC\_ADCInUnityGainMode* TEST mode: ADC in unity gain mode.

## 12.6 Function Documentation

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

## Parameters

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                              |
| <i>config</i> | Pointer to configuration structure, see to <a href="#">adc_config_t</a> . |

**12.6.2 void ADC\_Deinit ( ADC\_Type \* *base* )**

## Parameters

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

**12.6.3 void ADC\_GetDefaultConfig ( adc\_config\_t \* *config* )**

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

```
* config->clockMode = kADC_ClockSynchronousMode;
* config->clockDividerNumber = 0U;
* config->resolution = kADC_Resolution12bit;
* config->enableBypassCalibration = false;
* config->sampleTimeNumber = 0U;
*
```

## Parameters

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

**12.6.4 static void ADC\_EnableConvSeqA ( ADC\_Type \* *base*, bool *enable* )  
[inline], [static]**

In order to avoid spuriously triggering the sequence, the trigger to conversion sequence should be ready before the sequence is ready. when the sequence is disabled, the trigger would be ignored. Also, it is suggested to disable the sequence during changing the sequence's setting.

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | ADC peripheral base address.           |
| <i>enable</i> | Switcher to enable the feature or not. |

### 12.6.5 void ADC\_SetConvSeqAConfig ( ADC\_Type \* *base*, const adc\_conv\_seq\_config\_t \* *config* )

Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                                       |
| <i>config</i> | Pointer to configuration structure, see to <a href="#">adc_conv_seq_config_t</a> . |

### 12.6.6 static void ADC\_DoSoftwareTriggerConvSeqA ( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

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

Enable the burst mode would cause the conversion sequence to be continuously cycled through. Other triggers would be ignored while this mode is enabled. Repeated conversions could be halted by disabling this mode. And the sequence currently in process will be completed before conversions are terminated. Note that a new sequence could begin just before the burst mode is disabled.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | ADC peripheral base address.     |
| <i>enable</i> | Switcher to enable this feature. |

### 12.6.8 static void ADC\_SetConvSeqAHighPriority ( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

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

In order to avoid spuriously triggering the sequence, the trigger to conversion sequence should be ready before the sequence is ready. when the sequence is disabled, the trigger would be ignored. Also, it is suggested to disable the sequence during changing the sequence's setting.

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | ADC peripheral base address.           |
| <i>enable</i> | Switcher to enable the feature or not. |

### 12.6.10 void ADC\_SetConvSeqBConfig ( ADC\_Type \* *base*, const adc\_conv\_seq\_config\_t \* *config* )

Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                                       |
| <i>config</i> | Pointer to configuration structure, see to <a href="#">adc_conv_seq_config_t</a> . |

### 12.6.11 static void ADC\_DoSoftwareTriggerConvSeqB ( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

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

Enable the burst mode would cause the conversion sequence to be continuously cycled through. Other triggers would be ignored while this mode is enabled. Repeated conversions could be halted by disabling

this mode. And the sequence currently in process will be completed before conversions are terminated. Note that a new sequence could begin just before the burst mode is disabled.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | ADC peripheral base address.     |
| <i>enable</i> | Switcher to enable this feature. |

### 12.6.13 static void ADC\_SetConvSeqBHighPriority ( ADC\_Type \* *base* ) [inline], [static]

Parameters

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

### 12.6.14 bool ADC\_GetConvSeqAGlobalConversionResult ( ADC\_Type \* *base*, adc\_result\_info\_t \* *info* )

Parameters

|             |                                                                              |
|-------------|------------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                                 |
| <i>info</i> | Pointer to information structure, see to <a href="#">adc_result_info_t</a> ; |

Return values

|              |                                         |
|--------------|-----------------------------------------|
| <i>true</i>  | The conversion result is ready.         |
| <i>false</i> | The conversion result is not ready yet. |

### 12.6.15 bool ADC\_GetConvSeqBGlobalConversionResult ( ADC\_Type \* *base*, adc\_result\_info\_t \* *info* )

Parameters

|             |                                                                              |
|-------------|------------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                                 |
| <i>info</i> | Pointer to information structure, see to <a href="#">adc_result_info_t</a> ; |

Return values

|              |                                         |
|--------------|-----------------------------------------|
| <i>true</i>  | The conversion result is ready.         |
| <i>false</i> | The conversion result is not ready yet. |

### 12.6.16 **bool** ADC\_GetChannelConversionResult ( ADC\_Type \* *base*, uint32\_t *channel*, adc\_result\_info\_t \* *info* )

Parameters

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| <i>base</i>    | ADC peripheral base address.                                                 |
| <i>channel</i> | The indicated channel number.                                                |
| <i>info</i>    | Pointer to information structure, see to <a href="#">adc_result_info_t</a> ; |

Return values

|              |                                         |
|--------------|-----------------------------------------|
| <i>true</i>  | The conversion result is ready.         |
| <i>false</i> | The conversion result is not ready yet. |

### 12.6.17 **static void** ADC\_SetThresholdPair0 ( ADC\_Type \* *base*, uint32\_t *lowValue*, uint32\_t *highValue* ) [inline], [static]

Parameters

|                  |                              |
|------------------|------------------------------|
| <i>base</i>      | ADC peripheral base address. |
| <i>lowValue</i>  | LOW threshold value.         |
| <i>highValue</i> | HIGH threshold value.        |

### 12.6.18 **static void** ADC\_SetThresholdPair1 ( ADC\_Type \* *base*, uint32\_t *lowValue*, uint32\_t *highValue* ) [inline], [static]

Parameters

\_\_\_\_\_

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | ADC peripheral base address.                              |
| <i>lowValue</i>  | LOW threshold value. The available value is with 12-bit.  |
| <i>highValue</i> | HIGH threshold value. The available value is with 12-bit. |

**12.6.19** `static void ADC_SetChannelWithThresholdPair0 ( ADC_Type * base,  
uint32_t channelMask ) [inline], [static]`

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | ADC peripheral base address. |
| <i>channelMask</i> | Indicated channels' mask.    |

**12.6.20** `static void ADC_SetChannelWithThresholdPair1 ( ADC_Type * base,  
uint32_t channelMask ) [inline], [static]`

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | ADC peripheral base address. |
| <i>channelMask</i> | Indicated channels' mask.    |

**12.6.21** `static void ADC_EnableInterrupts ( ADC_Type * base, uint32_t mask )  
[inline], [static]`

Parameters

|             |                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                                                                            |
| <i>mask</i> | Mask of interrupt mask value for global block except each channel, see to <a href="#">_adc_-<br/>interrupt_enable</a> . |

**12.6.22** `static void ADC_DisableInterrupts ( ADC_Type * base, uint32_t mask )  
[inline], [static]`

## Parameters

|             |                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                                                                      |
| <i>mask</i> | Mask of interrupt mask value for global block except each channel, see to <a href="#">_adc_interrupt_enable</a> . |

**12.6.23** `static void ADC_EnableThresholdCompareInterrupt ( ADC_Type * base, uint32_t channel, adc_threshold_interrupt_mode_t mode ) [inline], [static]`

## Parameters

|                |                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i>    | ADC peripheral base address.                                                                        |
| <i>channel</i> | Channel number.                                                                                     |
| <i>mode</i>    | Interrupt mode for threshold compare event, see to <a href="#">adc_threshold_interrupt_mode_t</a> . |

**12.6.24** `static uint32_t ADC_GetStatusFlags ( ADC_Type * base ) [inline], [static]`

## Parameters

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

## Returns

Mask of status flags of module, see to [\\_adc\\_status\\_flags](#).

**12.6.25** `static void ADC_ClearStatusFlags ( ADC_Type * base, uint32_t mask ) [inline], [static]`

## Parameters

---

|             |                                                                            |
|-------------|----------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                               |
| <i>mask</i> | Mask of status flags of module, see to <a href="#">_adc_status_flags</a> . |

## Chapter 13

# CRC: Cyclic Redundancy Check Driver

### 13.1 Overview

MCUXpresso SDK provides a peripheral driver for the Cyclic Redundancy Check (CRC) module of MCUXpresso SDK devices.

The cyclic redundancy check (CRC) module generates 16/32-bit CRC code for error detection. The CRC module provides three variants of polynomials, a programmable seed, and other parameters required to implement a 16-bit or 32-bit CRC standard.

### 13.2 CRC Driver Initialization and Configuration

[CRC\\_Init\(\)](#) function enables the clock for the CRC module in the LPC SYSCON block and fully (re-)configures the CRC module according to configuration structure. It also starts checksum computation by writing the seed.

The seed member of the configuration structure is the initial checksum for which new data can be added to. When starting new checksum computation, the seed should be set to the initial checksum per the CRC protocol specification. For continued checksum operation, the seed should be set to the intermediate checksum value as obtained from previous calls to [CRC\\_GetConfig\(\)](#) function. After [CRC\\_Init\(\)](#), one or multiple [CRC\\_WriteData\(\)](#) calls follow to update checksum with data, then [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) follows to read the result. [CRC\\_Init\(\)](#) can be called as many times as required, which allows for runtime changes of the CRC protocol.

[CRC\\_GetDefaultConfig\(\)](#) function can be used to set the module configuration structure with parameters for CRC-16/CCITT-FALSE protocol.

[CRC\\_Deinit\(\)](#) function disables clock to the CRC module.

[CRC\\_Reset\(\)](#) performs hardware reset of the CRC module.

### 13.3 CRC Write Data

The [CRC\\_WriteData\(\)](#) function is used to add data to actual CRC. Internally it tries to use 32-bit reads and writes for all aligned data in the user buffer and it uses 8-bit reads and writes for all unaligned data in the user buffer. This function can update CRC with user supplied data chunks of arbitrary size, so one can update CRC byte by byte or with all bytes at once. Prior call of CRC configuration function [CRC\\_Init\(\)](#) fully specifies the CRC module configuration for [CRC\\_WriteData\(\)](#) call.

[CRC\\_WriteSeed\(\)](#) Write seed (initial checksum) to CRC module.

### 13.4 CRC Get Checksum

The [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) function is used to read the CRC module checksum register. The bit reverse and 1's complement operations are already applied to the result if previously

configured. Use [CRC\\_GetConfig\(\)](#) function to get the actual checksum without bit reverse and 1's complement applied so it can be used as seed when resuming calculation later.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / [CRC\\_Get16bitResult\(\)](#) to get final checksum.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / ... / [CRC\\_WriteData\(\)](#) / [CRC\\_Get16bitResult\(\)](#) to get final checksum.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / [CRC\\_GetConfig\(\)](#) to get intermediate checksum to be used as seed value in future.

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / ... / [CRC\\_WriteData\(\)](#) / [CRC\\_GetConfig\(\)](#) to get intermediate checksum.

### 13.5 Comments about API usage in RTOS

If multiple RTOS tasks share the CRC module to compute checksums with different data and/or protocols, the following needs to be implemented by the user:

The triplets

[CRC\\_Init\(\)](#) / [CRC\\_WriteData\(\)](#) / [CRC\\_Get16bitResult\(\)](#) or [CRC\\_Get32bitResult\(\)](#) or [CRC\\_GetConfig\(\)](#)

Should be protected by RTOS mutex to protect CRC module against concurrent accesses from different tasks. For example: Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/crc` Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/crc` Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/crc` Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/crc` Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/crc` Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/crc`

#### Files

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

#### Data Structures

- struct [\\_crc\\_config](#)  
*CRC protocol configuration. [More...](#)*

#### Macros

- #define [CRC\\_DRIVER\\_USE\\_CRC16\\_CCITT\\_FALSE\\_AS\\_DEFAULT](#) 1  
*Default configuration structure filled by [CRC\\_GetDefaultConfig\(\)](#).*

#### Typedefs

- typedef enum [\\_crc\\_polynomial](#) [crc\\_polynomial\\_t](#)  
*CRC polynomials to use.*
- typedef struct [\\_crc\\_config](#) [crc\\_config\\_t](#)  
*CRC protocol configuration.*

## Enumerations

- enum `_crc_polynomial` {  
`kCRC_Polynomial_CRC_CCITT` = 0U,  
`kCRC_Polynomial_CRC_16` = 1U,  
`kCRC_Polynomial_CRC_32` = 2U }  
*CRC polynomials to use.*

## Functions

- void `CRC_Init` (`CRC_Type *base`, const `crc_config_t *config`)  
*Enables and configures the CRC peripheral module.*
- static void `CRC_Deinit` (`CRC_Type *base`)  
*Disables the CRC peripheral module.*
- void `CRC_Reset` (`CRC_Type *base`)  
*resets CRC peripheral module.*
- void `CRC_WriteSeed` (`CRC_Type *base`, `uint32_t seed`)  
*Write seed to CRC peripheral module.*
- void `CRC_GetDefaultConfig` (`crc_config_t *config`)  
*Loads default values to CRC protocol configuration structure.*
- void `CRC_GetConfig` (`CRC_Type *base`, `crc_config_t *config`)  
*Loads actual values configured in CRC peripheral to CRC protocol configuration structure.*
- void `CRC_WriteData` (`CRC_Type *base`, const `uint8_t *data`, `size_t dataSize`)  
*Writes data to the CRC module.*
- static `uint32_t CRC_Get32bitResult` (`CRC_Type *base`)  
*Reads 32-bit checksum from the CRC module.*
- static `uint16_t CRC_Get16bitResult` (`CRC_Type *base`)  
*Reads 16-bit checksum from the CRC module.*

## Driver version

- `#define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`  
*CRC driver version.*

## 13.6 Data Structure Documentation

### 13.6.1 struct `_crc_config`

This structure holds the configuration for the CRC protocol.

#### Data Fields

- `crc_polynomial_t polynomial`  
*CRC polynomial.*
- bool `reverseIn`  
*Reverse bits on input.*
- bool `complementIn`  
*Perform 1's complement on input.*
- bool `reverseOut`

- *Reverse bits on output.*
- bool `complementOut`  
*Perform 1's complement on output.*
- uint32\_t `seed`  
*Starting checksum value.*

### Field Documentation

- (1) `crc_polynomial_t _crc_config::polynomial`
- (2) `bool _crc_config::reverseIn`
- (3) `bool _crc_config::complementIn`
- (4) `bool _crc_config::reverseOut`
- (5) `bool _crc_config::complementOut`
- (6) `uint32_t _crc_config::seed`

## 13.7 Macro Definition Documentation

### 13.7.1 `#define FSL_CRC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.0.0
  - initial version
- Version 2.0.1
  - add explicit type cast when writing to WR\_DATA
- Version 2.0.2
  - Fix MISRA issue
- Version 2.1.0
  - Add CRC\_WriteSeed function
- Version 2.1.1
  - Fix MISRA issue

### 13.7.2 `#define CRC_DRIVER_USE_CRC16_CCITT_FALSE_AS_DEFAULT 1`

Uses CRC-16/CCITT-FALSE as default.

## 13.8 Typedef Documentation

### 13.8.1 typedef enum \_crc\_polynomial crc\_polynomial\_t

### 13.8.2 typedef struct \_crc\_config crc\_config\_t

This structure holds the configuration for the CRC protocol.

## 13.9 Enumeration Type Documentation

### 13.9.1 enum \_crc\_polynomial

Enumerator

*kCRC\_Polynomial\_CRC\_CCITT*  $x^{16}+x^{12}+x^5+1$

*kCRC\_Polynomial\_CRC\_16*  $x^{16}+x^{15}+x^2+1$

*kCRC\_Polynomial\_CRC\_32*  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

## 13.10 Function Documentation

### 13.10.1 void CRC\_Init ( CRC\_Type \* *base*, const crc\_config\_t \* *config* )

This function enables the CRC peripheral clock in the LPC SYSCON block. It also configures the CRC engine and starts checksum computation by writing the seed.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | CRC peripheral address.             |
| <i>config</i> | CRC module configuration structure. |

### 13.10.2 static void CRC\_Deinit ( CRC\_Type \* *base* ) [inline], [static]

This function disables the CRC peripheral clock in the LPC SYSCON block.

Parameters

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

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

Parameters

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

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

Parameters

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

### 13.10.5 void CRC\_GetDefaultConfig ( crc\_config\_t \* *config* )

Loads default values to CRC protocol configuration structure. The default values are:

```
* config->polynomial = kCRC_Polynomial_CRC_CCITT;
* config->reverseIn = false;
* config->complementIn = false;
* config->reverseOut = false;
* config->complementOut = false;
* config->seed = 0xFFFFU;
*
```

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>config</i> | CRC protocol configuration structure |
|---------------|--------------------------------------|

### 13.10.6 void CRC\_GetConfig ( CRC\_Type \* *base*, crc\_config\_t \* *config* )

The values, including seed, can be used to resume CRC calculation later.

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | CRC peripheral address.              |
| <i>config</i> | CRC protocol configuration structure |

### 13.10.7 void CRC\_WriteData ( CRC\_Type \* *base*, const uint8\_t \* *data*, size\_t *dataSize* )

Writes input data buffer bytes to CRC data register.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | CRC peripheral address.                 |
| <i>data</i>     | Input data stream, MSByte in data[0].   |
| <i>dataSize</i> | Size of the input data buffer in bytes. |

**13.10.8** `static uint32_t CRC_Get32bitResult ( CRC_Type * base ) [inline], [static]`

Reads CRC data register.

Parameters

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

Returns

final 32-bit checksum, after configured bit reverse and complement operations.

**13.10.9** `static uint16_t CRC_Get16bitResult ( CRC_Type * base ) [inline], [static]`

Reads CRC data register.

Parameters

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

Returns

final 16-bit checksum, after configured bit reverse and complement operations.

## Chapter 14

# DAC: 10-bit Digital To Analog Converter Driver

### 14.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 10-bit digital to analog converter (DAC) module of MCUXpresso SDK devices.

### 14.2 Typical use case

#### 14.2.1 Polling Configuration

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

#### 14.2.2 Interrupt Configuration

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

### Files

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

### Data Structures

- struct [\\_dac\\_config](#)  
*The configuration of DAC. [More...](#)*

### Typedefs

- typedef enum [\\_dac\\_settling\\_time](#) [dac\\_settling\\_time\\_t](#)  
*The DAC settling time.*
- typedef struct [\\_dac\\_config](#) [dac\\_config\\_t](#)  
*The configuration of DAC.*

### Enumerations

- enum [\\_dac\\_settling\\_time](#) {  
    [kDAC\\_SettlingTimeIs1us](#) = 0U,  
    [kDAC\\_SettlingTimeIs25us](#) = 1U }  
*The DAC settling time.*

## Functions

- void `DAC_Init` (DAC\_Type \*base, const `dac_config_t` \*config)  
*Initialize the DAC module.*
- void `DAC_Deinit` (DAC\_Type \*base)  
*De-Initialize the DAC module.*
- void `DAC_GetDefaultConfig` (`dac_config_t` \*config)  
*Initializes the DAC user configuration structure.*
- void `DAC_EnableDoubleBuffering` (DAC\_Type \*base, bool enable)  
*Enable/Diable double-buffering feature.*
- void `DAC_SetBufferValue` (DAC\_Type \*base, uint32\_t value)  
*Write DAC output value into CR register or pre-buffer.*
- void `DAC_SetCounterValue` (DAC\_Type \*base, uint32\_t value)  
*Write DAC counter value into CNTVAL register.*
- static void `DAC_EnableCounter` (DAC\_Type \*base, bool enable)  
*Enable/Disable the counter operation.*
- static bool `DAC_GetDMAInterruptRequestFlag` (DAC\_Type \*base)  
*Get the status flag of DMA or interrupt request.*

## Driver version

- #define `LPC_DAC_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 2)`)  
*DAC driver version 2.0.2.*

## 14.3 Data Structure Documentation

### 14.3.1 struct `_dac_config`

#### Data Fields

- `dac_settling_time_t` `settlingTime`  
*The settling times are valid for a capacitance load on the DAC\_OUT pin not exceeding 100 pF.*

#### Field Documentation

##### (1) `dac_settling_time_t _dac_config::settlingTime`

A load impedance value greater than that value will cause settling time longer than the specified time. One or more graphs of load impedance vs. settling time will be included in the final data sheet.

## 14.4 Macro Definition Documentation

### 14.4.1 #define `LPC_DAC_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 2)`)

## 14.5 Enumeration Type Documentation

### 14.5.1 enum \_dac\_settling\_time

Enumerator

***kDAC\_SettlingTimeIs1us*** The settling time of the DAC is 1us max, and the maximum current is 700 mA. This allows a maximum update rate of 1 MHz.

***kDAC\_SettlingTimeIs25us*** The settling time of the DAC is 2.5us and the maximum current is 350uA. This allows a maximum update rate of 400 kHz.

## 14.6 Function Documentation

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

Parameters

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <i>base</i>   | DAC peripheral base address.                                                      |
| <i>config</i> | The pointer to configuration structure. Please refer to "dac_config_t" structure. |

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

Parameters

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

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

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

```
* config->settlingTime = kDAC_SettlingTimeIs1us;
*
```

Parameters

---

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

#### 14.6.4 void DAC\_EnableDoubleBuffering ( DAC\_Type \* *base*, bool *enable* )

Notice: Disabling the double-buffering feature will disable counter operation. If double-buffering feature is disabled, any writes to the CR address will go directly to the CR register. If double-buffering feature is enabled, any write to the CR register will only load the pre-buffer, which shares its register address with the CR register. The CR itself will be loaded from the pre-buffer whenever the counter reaches zero and the DMA request is set.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | DAC peripheral base address.   |
| <i>enable</i> | Enable or disable the feature. |

#### 14.6.5 void DAC\_SetBufferValue ( DAC\_Type \* *base*, uint32\_t *value* )

The DAC output voltage is  $VALUE * ((VREFP) / 1024)$ .

Parameters

|              |                                                                   |
|--------------|-------------------------------------------------------------------|
| <i>base</i>  | DAC peripheral base address.                                      |
| <i>value</i> | Setting the value for items in the buffer. 10-bits are available. |

#### 14.6.6 void DAC\_SetCounterValue ( DAC\_Type \* *base*, uint32\_t *value* )

When the counter is enabled bit, the 16-bit counter will begin counting down, at the rate  $f_{clk} / 2^N$  from the value programmed into the DACCNTVAL register. The counter is decremented Each time

reaches zero, the counter will be reloaded by the value of DACCNTVAL and the DMA request bit INT\_DMA\_REQ will be set in hardware.

Parameters

---

|              |                                                                    |
|--------------|--------------------------------------------------------------------|
| <i>base</i>  | DAC peripheral basic address.                                      |
| <i>value</i> | Setting the value for items in the counter. 16-bits are available. |

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

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | DAC peripheral base address.   |
| <i>enable</i> | Enable or disable the feature. |

#### 14.6.8 static bool DAC\_GetDMAInterruptRequestFlag ( DAC\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

If return 'true', it means DMA request or interrupt occurs. If return 'false', it means DMA request or interrupt doesn't occur.

# Chapter 15

## GPIO: General Purpose I/O

### 15.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Purpose I/O (GPIO) module of MCUXpresso SDK devices.

### 15.2 Function groups

#### 15.2.1 Initialization and deinitialization

The function [GPIO\\_PinInit\(\)](#) initializes the GPIO with specified configuration.

#### 15.2.2 Pin manipulation

The function [GPIO\\_PinWrite\(\)](#) set output state of selected GPIO pin. The function [GPIO\\_PinRead\(\)](#) read input value of selected GPIO pin.

#### 15.2.3 Port manipulation

The function [GPIO\\_PortSet\(\)](#) sets the output level of selected GPIO pins to the logic 1. The function [GPIO\\_PortClear\(\)](#) sets the output level of selected GPIO pins to the logic 0. The function [GPIO\\_PortToggle\(\)](#) reverse the output level of selected GPIO pins. The function [GPIO\\_PortRead\(\)](#) read input value of selected port.

#### 15.2.4 Port masking

The function [GPIO\\_PortMaskedSet\(\)](#) set port mask, only pins masked by 0 will be enabled in following functions. The function [GPIO\\_PortMaskedWrite\(\)](#) sets the state of selected GPIO port, only pins masked by 0 will be affected. The function [GPIO\\_PortMaskedRead\(\)](#) reads the state of selected GPIO port, only pins masked by 0 are enabled for read, pins masked by 1 are read as 0.

### 15.3 Typical use case

Example use of GPIO API. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/gpio`

## Files

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

## Data Structures

- struct [\\_gpio\\_pin\\_config](#)  
*The GPIO pin configuration structure. [More...](#)*

## Typedefs

- typedef enum [\\_gpio\\_pin\\_direction](#) [gpio\\_pin\\_direction\\_t](#)  
*LPC GPIO direction definition.*
- typedef struct [\\_gpio\\_pin\\_config](#) [gpio\\_pin\\_config\\_t](#)  
*The GPIO pin configuration structure.*

## Enumerations

- enum [\\_gpio\\_pin\\_direction](#) {  
[kGPIO\\_DigitalInput](#) = 0U,  
[kGPIO\\_DigitalOutput](#) = 1U }  
*LPC GPIO direction definition.*

## Functions

- static void [GPIO\\_PortSet](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_PortClear](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_PortToggle](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Reverses current output logic of the multiple GPIO pins.*

## Driver version

- #define [FSL\\_GPIO\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 7))  
*LPC GPIO driver version.*

## GPIO Configuration

- void [GPIO\\_PortInit](#) (GPIO\_Type \*base, uint32\_t port)  
*Initializes the GPIO peripheral.*
- void [GPIO\\_PinInit](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin, const [gpio\\_pin\\_config\\_t](#) \*config)  
*Initializes a GPIO pin used by the board.*

## GPIO Output Operations

- static void [GPIO\\_PinWrite](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin, uint8\_t output)  
*Sets the output level of the one GPIO pin to the logic 1 or 0.*

## GPIO Input Operations

- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin)  
*Reads the current input value of the GPIO PIN.*

## 15.4 Data Structure Documentation

### 15.4.1 struct \_gpio\_pin\_config

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

#### Data Fields

- [gpio\\_pin\\_direction\\_t pinDirection](#)  
*GPIO direction, input or output.*
- uint8\_t [outputLogic](#)  
*Set default output logic, no use in input.*

## 15.5 Macro Definition Documentation

### 15.5.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 7))

## 15.6 Typedef Documentation

### 15.6.1 typedef struct \_gpio\_pin\_config 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.

## 15.7 Enumeration Type Documentation

### 15.7.1 enum \_gpio\_pin\_direction

Enumerator

- kGPIO\_DigitalInput* Set current pin as digital input.
- kGPIO\_DigitalOutput* Set current pin as digital output.

## 15.8 Function Documentation

### 15.8.1 void GPIO\_PortInit ( GPIO\_Type \* base, uint32\_t port )

This function ungates the GPIO clock.

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | GPIO peripheral base pointer. |
| <i>port</i> | GPIO port number.             |

### 15.8.2 void GPIO\_PinInit ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *pin*, const gpio\_pin\_config\_t \* *config* )

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the [GPIO\\_PinInit\(\)](#) function.

This is an example to define an input pin or output pin configuration:

```
* Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalOutput,
*   0,
* }
*
```

## Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i>   | GPIO port number                             |
| <i>pin</i>    | GPIO pin number                              |
| <i>config</i> | GPIO pin configuration pointer               |

### 15.8.3 static void GPIO\_PinWrite ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]

## Parameters

|               |                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer(Typically GPIO)                                                                                                                                           |
| <i>port</i>   | GPIO port number                                                                                                                                                                       |
| <i>pin</i>    | GPIO pin number                                                                                                                                                                        |
| <i>output</i> | GPIO pin output logic level. <ul style="list-style-type: none"> <li>• 0: corresponding pin output low-logic level.</li> <li>• 1: corresponding pin output high-logic level.</li> </ul> |

#### 15.8.4 static uint32\_t GPIO\_PinRead ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *pin* ) [inline], [static]

##### Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i> | GPIO port number                             |
| <i>pin</i>  | GPIO pin number                              |

##### Return values

|             |                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>GPIO</i> | port input value <ul style="list-style-type: none"> <li>• 0: corresponding pin input low-logic level.</li> <li>• 1: corresponding pin input high-logic level.</li> </ul> |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 15.8.5 static void GPIO\_PortSet ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *mask* ) [inline], [static]

##### Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i> | GPIO port number                             |
| <i>mask</i> | GPIO pin number macro                        |

#### 15.8.6 static void GPIO\_PortClear ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i> | GPIO port number                             |
| <i>mask</i> | GPIO pin number macro                        |

**15.8.7 static void GPIO\_PortToggle ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i> | GPIO port number                             |
| <i>mask</i> | GPIO pin number macro                        |

# Chapter 16

## I2C: Inter-Integrated Circuit Driver

### 16.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of MCUXpresso SDK devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions [I2C\\_MasterTransferNonBlocking\(\)](#) set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

### 16.2 Typical use case

#### 16.2.1 Master Operation in functional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c-`  
Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c-`

#### 16.2.2 Master Operation in DMA transactional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

#### 16.2.3 Slave Operation in functional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

## 16.2.4 Slave Operation in interrupt transactional method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/i2c`

### Modules

- [I2C Driver](#)
- [I2C Master Driver](#)
- [I2C Slave Driver](#)

## 16.3 I2C Driver

### 16.3.1 Overview

#### Files

- file [fsl\\_i2c.h](#)

#### Macros

- #define [I2C\\_RETRY\\_TIMES](#) 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*
- #define [I2C\\_STAT\\_MSTCODE\\_IDLE](#) (0)  
*Master Idle State Code.*
- #define [I2C\\_STAT\\_MSTCODE\\_RXREADY](#) (1UL)  
*Master Receive Ready State Code.*
- #define [I2C\\_STAT\\_MSTCODE\\_TXREADY](#) (2UL)  
*Master Transmit Ready State Code.*
- #define [I2C\\_STAT\\_MSTCODE\\_NACKADR](#) (3UL)  
*Master NACK by slave on address State Code.*
- #define [I2C\\_STAT\\_MSTCODE\\_NACKDAT](#) (4UL)  
*Master NACK by slave on data State Code.*

#### Enumerations

- enum {  
[kStatus\\_I2C\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 0),  
[kStatus\\_I2C\\_Idle](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 1),  
[kStatus\\_I2C\\_Nak](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 2),  
[kStatus\\_I2C\\_InvalidParameter](#),  
[kStatus\\_I2C\\_BitError](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 4),  
[kStatus\\_I2C\\_ArbitrationLost](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 5),  
[kStatus\\_I2C\\_NoTransferInProgress](#),  
[kStatus\\_I2C\\_DmaRequestFail](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 7),  
[kStatus\\_I2C\\_StartStopError](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 8),  
[kStatus\\_I2C\\_UnexpectedState](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 9),  
[kStatus\\_I2C\\_Addr\\_Nak](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 10),  
[kStatus\\_I2C\\_Timeout](#) = MAKE\_STATUS(kStatusGroup\_LPC\_I2C, 11) }  
*I2C status return codes.*

#### Driver version

- #define [FSL\\_I2C\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 1, 0))  
*I2C driver version.*

## 16.3.2 Macro Definition Documentation

16.3.2.1 `#define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

16.3.2.2 `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

## 16.3.3 Enumeration Type Documentation

### 16.3.3.1 anonymous enum

Enumerator

*kStatus\_I2C\_Busy* The master is already performing a transfer.

*kStatus\_I2C\_Idle* The slave driver is idle.

*kStatus\_I2C\_Nak* The slave device sent a NAK in response to a byte.

*kStatus\_I2C\_InvalidParameter* Unable to proceed due to invalid parameter.

*kStatus\_I2C\_BitError* Transferred bit was not seen on the bus.

*kStatus\_I2C\_ArbitrationLost* Arbitration lost error.

*kStatus\_I2C\_NoTransferInProgress* Attempt to abort a transfer when one is not in progress.

*kStatus\_I2C\_DmaRequestFail* DMA request failed.

*kStatus\_I2C\_StartStopError* Start and stop error.

*kStatus\_I2C\_UnexpectedState* Unexpected state.

*kStatus\_I2C\_Addr\_Nak* NAK received during the address probe.

*kStatus\_I2C\_Timeout* Timeout polling status flags.

## 16.4 I2C Master Driver

### 16.4.1 Overview

#### Data Structures

- struct `_i2c_master_config`  
*Structure with settings to initialize the I2C master module. [More...](#)*
- struct `_i2c_master_transfer`  
*Non-blocking transfer descriptor structure. [More...](#)*
- struct `_i2c_master_handle`  
*Driver handle for master non-blocking APIs. [More...](#)*

#### Typedefs

- typedef enum `_i2c_direction` `i2c_direction_t`  
*Direction of master and slave transfers.*
- typedef struct `_i2c_master_config` `i2c_master_config_t`  
*Structure with settings to initialize the I2C master module.*
- typedef struct `_i2c_master_transfer` `i2c_master_transfer_t`  
*I2C master transfer typedef.*
- typedef struct `_i2c_master_handle` `i2c_master_handle_t`  
*I2C master handle typedef.*
- typedef void(\* `i2c_master_transfer_callback_t` )(I2C\_Type \*base, `i2c_master_handle_t` \*handle, `status_t` completionStatus, void \*userData)  
*Master completion callback function pointer type.*

#### Enumerations

- enum `_i2c_master_flags` {  
`kI2C_MasterPendingFlag` = `I2C_STAT_MSTPENDING_MASK`,  
`kI2C_MasterArbitrationLostFlag`,  
`kI2C_MasterStartStopErrorFlag` }
- enum `_i2c_direction` {  
`kI2C_Write` = 0U,  
`kI2C_Read` = 1U }
- enum `_i2c_master_transfer_flags` {  
`kI2C_TransferDefaultFlag` = 0x00U,  
`kI2C_TransferNoStartFlag` = 0x01U,  
`kI2C_TransferRepeatedStartFlag` = 0x02U,  
`kI2C_TransferNoStopFlag` = 0x04U }
- enum `_i2c_transfer_states`  
*States for the state machine used by transactional APIs.*

## Initialization and deinitialization

- void **I2C\_MasterGetDefaultConfig** (i2c\_master\_config\_t \*masterConfig)  
*Provides a default configuration for the I2C master peripheral.*
- void **I2C\_MasterInit** (I2C\_Type \*base, const i2c\_master\_config\_t \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes the I2C master peripheral.*
- void **I2C\_MasterDeinit** (I2C\_Type \*base)  
*Deinitializes the I2C master peripheral.*
- uint32\_t **I2C\_GetInstance** (I2C\_Type \*base)  
*Returns an instance number given a base address.*
- static void **I2C\_MasterReset** (I2C\_Type \*base)  
*Performs a software reset.*
- static void **I2C\_MasterEnable** (I2C\_Type \*base, bool enable)  
*Enables or disables the I2C module as master.*

## Status

- static uint32\_t **I2C\_GetStatusFlags** (I2C\_Type \*base)  
*Gets the I2C status flags.*
- static void **I2C\_MasterClearStatusFlags** (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C master status flag state.*

## Interrupts

- static void **I2C\_EnableInterrupts** (I2C\_Type \*base, uint32\_t interruptMask)  
*Enables the I2C master interrupt requests.*
- static void **I2C\_DisableInterrupts** (I2C\_Type \*base, uint32\_t interruptMask)  
*Disables the I2C master interrupt requests.*
- static uint32\_t **I2C\_GetEnabledInterrupts** (I2C\_Type \*base)  
*Returns the set of currently enabled I2C master interrupt requests.*

## Bus operations

- void **I2C\_MasterSetBaudRate** (I2C\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the I2C bus frequency for master transactions.*
- static bool **I2C\_MasterGetBusIdleState** (I2C\_Type \*base)  
*Returns whether the bus is idle.*
- **status\_t I2C\_MasterStart** (I2C\_Type \*base, uint8\_t address, i2c\_direction\_t direction)  
*Sends a START on the I2C bus.*
- **status\_t I2C\_MasterStop** (I2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- static **status\_t I2C\_MasterRepeatedStart** (I2C\_Type \*base, uint8\_t address, i2c\_direction\_t direction)  
*Sends a REPEATED START on the I2C bus.*
- **status\_t I2C\_MasterWriteBlocking** (I2C\_Type \*base, const void \*txBuff, size\_t txSize, uint32\_t flags)

- Performs a polling send transfer on the I2C bus.  
 • [status\\_t I2C\\_MasterReadBlocking](#) (I2C\_Type \*base, void \*rxBuff, size\_t rxSize, uint32\_t flags)
- Performs a polling receive transfer on the I2C bus.  
 • [status\\_t I2C\\_MasterTransferBlocking](#) (I2C\_Type \*base, [i2c\\_master\\_transfer\\_t](#) \*xfer)
- Performs a master polling transfer on the I2C bus.

## Non-blocking

- void [I2C\\_MasterTransferCreateHandle](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
 Creates a new handle for the I2C master non-blocking APIs.
- [status\\_t I2C\\_MasterTransferNonBlocking](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
 Performs a non-blocking transaction on the I2C bus.
- [status\\_t I2C\\_MasterTransferGetCount](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle, size\_t \*count)  
 Returns number of bytes transferred so far.
- [status\\_t I2C\\_MasterTransferAbort](#) (I2C\_Type \*base, [i2c\\_master\\_handle\\_t](#) \*handle)  
 Terminates a non-blocking I2C master transmission early.

## IRQ handler

- void [I2C\\_MasterTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
 Reusable routine to handle master interrupts.

## 16.4.2 Data Structure Documentation

### 16.4.2.1 struct [i2c\\_master\\_config](#)

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the [I2C\\_MasterGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

### Data Fields

- bool [enableMaster](#)  
 Whether to enable master mode.
- uint32\_t [baudRate\\_Bps](#)  
 Desired baud rate in bits per second.
- bool [enableTimeout](#)  
 Enable internal timeout function.

### Field Documentation

- (1) `bool _i2c_master_config::enableMaster`
- (2) `uint32_t _i2c_master_config::baudRate_Bps`
- (3) `bool _i2c_master_config::enableTimeout`

#### 16.4.2.2 struct `_i2c_master_transfer`

This structure is used to pass transaction parameters to the `I2C_MasterTransferNonBlocking()` API.

### Data Fields

- `uint32_t flags`  
*Bit mask of options for the transfer.*
- `uint16_t slaveAddress`  
*The 7-bit slave address.*
- `i2c_direction_t direction`  
*Either `kI2C_Read` or `kI2C_Write`.*
- `uint32_t subaddress`  
*Sub address.*
- `size_t subaddressSize`  
*Length of sub address to send in bytes.*
- `void * data`  
*Pointer to data to transfer.*
- `size_t dataSize`  
*Number of bytes to transfer.*

### Field Documentation

#### (1) `uint32_t _i2c_master_transfer::flags`

See enumeration `_i2c_master_transfer_flags` for available options. Set to 0 or `kI2C_TransferDefaultFlag` for normal transfers.

#### (2) `uint16_t _i2c_master_transfer::slaveAddress`

#### (3) `i2c_direction_t _i2c_master_transfer::direction`

#### (4) `uint32_t _i2c_master_transfer::subaddress`

Transferred MSB first.

#### (5) `size_t _i2c_master_transfer::subaddressSize`

Maximum size is 4 bytes.

(6) `void* _i2c_master_transfer::data`

(7) `size_t _i2c_master_transfer::dataSize`

### 16.4.2.3 struct `_i2c_master_handle`

Note

The contents of this structure are private and subject to change.

#### Data Fields

- `uint8_t state`  
*Transfer state machine current state.*
- `uint32_t transferCount`  
*Indicates progress of the transfer.*
- `uint32_t remainingBytes`  
*Remaining byte count in current state.*
- `uint8_t * buf`  
*Buffer pointer for current state.*
- `i2c_master_transfer_t transfer`  
*Copy of the current transfer info.*
- `i2c_master_transfer_callback_t completionCallback`  
*Callback function pointer.*
- `void * userData`  
*Application data passed to callback.*

#### Field Documentation

(1) `uint8_t _i2c_master_handle::state`

(2) `uint32_t _i2c_master_handle::remainingBytes`

(3) `uint8_t* _i2c_master_handle::buf`

(4) `i2c_master_transfer_t _i2c_master_handle::transfer`

(5) `i2c_master_transfer_callback_t _i2c_master_handle::completionCallback`

(6) `void* _i2c_master_handle::userData`

### 16.4.3 Typedef Documentation

16.4.3.1 `typedef enum _i2c_direction i2c_direction_t`

16.4.3.2 `typedef struct _i2c_master_config i2c_master_config_t`

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the `I2C_MasterGetDefaultConfig()` function and pass a pointer to your configuration

structure instance.

The configuration structure can be made constant so it resides in flash.

### 16.4.3.3 typedef void(\* i2c\_master\_transfer\_callback\_t)(I2C\_Type \*base, i2c\_master\_handle\_t \*handle, status\_t completionStatus, void \*userData)

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to [I2C\\_MasterTransferCreateHandle\(\)](#).

Parameters

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| <i>base</i>              | The I2C peripheral base address.                                               |
| <i>completion-Status</i> | Either kStatus_Success or an error code describing how the transfer completed. |
| <i>userData</i>          | Arbitrary pointer-sized value passed from the application.                     |

## 16.4.4 Enumeration Type Documentation

### 16.4.4.1 enum \_i2c\_master\_flags

Note

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

Enumerator

***kI2C\_MasterPendingFlag*** The I2C module is waiting for software interaction.

***kI2C\_MasterArbitrationLostFlag*** The arbitration of the bus was lost. There was collision on the bus

***kI2C\_MasterStartStopErrorFlag*** There was an error during start or stop phase of the transaction.

### 16.4.4.2 enum \_i2c\_direction

Enumerator

***kI2C\_Write*** Master transmit.

***kI2C\_Read*** Master receive.

### 16.4.4.3 enum \_i2c\_master\_transfer\_flags

## Note

These enumerations are intended to be OR'd together to form a bit mask of options for the `_i2c_-master_transfer::flags` field.

## Enumerator

- kI2C\_TransferDefaultFlag*** Transfer starts with a start signal, stops with a stop signal.
- kI2C\_TransferNoStartFlag*** Don't send a start condition, address, and sub address.
- kI2C\_TransferRepeatedStartFlag*** Send a repeated start condition.
- kI2C\_TransferNoStopFlag*** Don't send a stop condition.

16.4.4.4 enum `_i2c_transfer_states`

## 16.4.5 Function Documentation

16.4.5.1 void `I2C_MasterGetDefaultConfig ( i2c_master_config_t * masterConfig )`

This function provides the following default configuration for the I2C master peripheral:

```
* masterConfig->enableMaster      = true;
* masterConfig->baudRate_Bps      = 100000U;
* masterConfig->enableTimeout     = false;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with `I2C_MasterInit()`.

## Parameters

|     |                     |                                                                                                        |
|-----|---------------------|--------------------------------------------------------------------------------------------------------|
| out | <i>masterConfig</i> | User provided configuration structure for default values. Refer to <code>i2c_-master_config_t</code> . |
|-----|---------------------|--------------------------------------------------------------------------------------------------------|

16.4.5.2 void `I2C_MasterInit ( I2C_Type * base, const i2c_master_config_t * masterConfig, uint32_t srcClock_Hz )`

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

## Parameters

---

|                     |                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | The I2C peripheral base address.                                                                                                         |
| <i>masterConfig</i> | User provided peripheral configuration. Use <a href="#">I2C_MasterGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>srcClock_Hz</i>  | Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.            |

#### 16.4.5.3 void I2C\_MasterDeinit ( I2C\_Type \* *base* )

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

#### 16.4.5.4 uint32\_t I2C\_GetInstance ( I2C\_Type \* *base* )

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

Returns

I2C instance number starting from 0.

#### 16.4.5.5 static void I2C\_MasterReset ( I2C\_Type \* *base* ) [inline], [static]

Restores the I2C master peripheral to reset conditions.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

#### 16.4.5.6 static void I2C\_MasterEnable ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                     |
| <i>enable</i> | Pass true to enable or false to disable the specified I2C as master. |

**16.4.5.7 static uint32\_t I2C\_GetStatusFlags ( I2C\_Type \* *base* ) [inline], [static]**

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

## Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

## See Also

[\\_i2c\\_master\\_flags](#)

**16.4.5.8 static void I2C\_MasterClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]**

The following status register flags can be cleared:

- [kI2C\\_MasterArbitrationLostFlag](#)
- [kI2C\\_MasterStartStopErrorFlag](#)

Attempts to clear other flags has no effect.

## Parameters

|                   |                                                                                                                                                                                                                             |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I2C peripheral base address.                                                                                                                                                                                            |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_i2c_master_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I2C_GetStatusFlags()</a> . |

## See Also

[\\_i2c\\_master\\_flags](#).

16.4.5.9 `static void I2C_EnableInterrupts ( I2C_Type * base, uint32_t interruptMask )`  
`[inline], [static]`

## Parameters

|                      |                                                                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_i2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 16.4.5.10 static void I2C\_DisableInterrupts ( I2C\_Type \* *base*, uint32\_t *interruptMask* ) [inline], [static]

## Parameters

|                      |                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I2C peripheral base address.                                                                                                                     |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_i2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

#### 16.4.5.11 static uint32\_t I2C\_GetEnabledInterrupts ( I2C\_Type \* *base* ) [inline], [static]

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

## Returns

A bitmask composed of [\\_i2c\\_master\\_flags](#) enumerators OR'd together to indicate the set of enabled interrupts.

#### 16.4.5.12 void I2C\_MasterSetBaudRate ( I2C\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

The I2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

## Parameters

|                     |                                             |
|---------------------|---------------------------------------------|
| <i>base</i>         | The I2C peripheral base address.            |
| <i>srcClock_Hz</i>  | I2C functional clock frequency in Hertz.    |
| <i>baudRate_Bps</i> | Requested bus frequency in bits per second. |

#### 16.4.5.13 `static bool I2C_MasterGetBusIdleState ( I2C_Type * base ) [inline], [static]`

Requires the master mode to be enabled.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

#### 16.4.5.14 `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.                |

#### 16.4.5.15 `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.  |

**16.4.5.16** `static status_t I2C_MasterRepeatedStart ( I2C_Type * base, uint8_t address, i2c_direction_t direction ) [inline], [static]`

Parameters

|                  |                                               |
|------------------|-----------------------------------------------|
| <i>base</i>      | I2C peripheral base pointer                   |
| <i>address</i>   | 7-bit slave device address.                   |
| <i>direction</i> | Master transfer directions(transmit/receive). |

Return values

|                         |                                                             |
|-------------------------|-------------------------------------------------------------|
| <i>kStatus_Success</i>  | Successfully send the start signal.                         |
| <i>kStatus_I2C_Busy</i> | Current bus is busy but not occupied by current I2C master. |

**16.4.5.17** `status_t I2C_MasterWriteBlocking ( I2C_Type * base, const void * txBuff, size_t txSize, uint32_t flags )`

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns [kStatus\\_I2C\\_Nak](#).

Parameters

|               |                                                                                                                                     |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                                                                                    |
| <i>txBuff</i> | The pointer to the data to be transferred.                                                                                          |
| <i>txSize</i> | The length in bytes of the data to be transferred.                                                                                  |
| <i>flags</i>  | Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag |

Return values

|                                     |                                                    |
|-------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>              | Data was sent successfully.                        |
| <i>kStatus_I2C_Busy</i>             | Another master is currently utilizing the bus.     |
| <i>kStatus_I2C_Nak</i>              | The slave device sent a NAK in response to a byte. |
| <i>kStatus_I2C_Arbitration-Lost</i> | Arbitration lost error.                            |

#### 16.4.5.18 `status_t I2C_MasterReadBlocking ( I2C_Type * base, void * rxBuff, size_t rxSize, uint32_t flags )`

Parameters

|               |                                                                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                                                                                                 |
| <i>rxBuff</i> | The pointer to the data to be transferred.                                                                                                       |
| <i>rxSize</i> | The length in bytes of the data to be transferred.                                                                                               |
| <i>flags</i>  | Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use <code>kI2C_TransferDefaultFlag</code> |

Return values

|                                     |                                                    |
|-------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>              | Data was received successfully.                    |
| <i>kStatus_I2C_Busy</i>             | Another master is currently utilizing the bus.     |
| <i>kStatus_I2C_Nak</i>              | The slave device sent a NAK in response to a byte. |
| <i>kStatus_I2C_Arbitration-Lost</i> | Arbitration lost error.                            |

#### 16.4.5.19 `status_t I2C_MasterTransferBlocking ( I2C_Type * base, i2c_master_transfer_t * xfer )`

Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | I2C peripheral base address.       |
| <i>xfer</i> | Pointer to the transfer structure. |

Return values

|                                     |                                              |
|-------------------------------------|----------------------------------------------|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Busy</i>             | Previous transmission still not finished.    |
| <i>kStatus_I2C_Timeout</i>          | Transfer error, wait signal timeout.         |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStataus_I2C_Nak</i>             | Transfer error, receive NAK during transfer. |

**16.4.5.20 void I2C\_MasterTransferCreateHandle ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [I2C\\_MasterTransferAbort\(\)](#) API shall be called.

Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The I2C peripheral base address.                             |
| out | <i>handle</i>   | Pointer to the I2C master driver handle.                     |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

**16.4.5.21 status\_t I2C\_MasterTransferNonBlocking ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )**

Parameters

|  |               |                                          |
|--|---------------|------------------------------------------|
|  | <i>base</i>   | The I2C peripheral base address.         |
|  | <i>handle</i> | Pointer to the I2C master driver handle. |

|             |                                         |
|-------------|-----------------------------------------|
| <i>xfer</i> | The pointer to the transfer descriptor. |
|-------------|-----------------------------------------|

## Return values

|                         |                                                                                                             |
|-------------------------|-------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>  | The transaction was started successfully.                                                                   |
| <i>kStatus_I2C_Busy</i> | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |

#### 16.4.5.22 `status_t I2C_MasterTransferGetCount ( I2C_Type * base, i2c_master_handle_t * handle, size_t * count )`

## Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The I2C peripheral base address.                                    |
|     | <i>handle</i> | Pointer to the I2C master driver handle.                            |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

## Return values

|                         |  |
|-------------------------|--|
| <i>kStatus_Success</i>  |  |
| <i>kStatus_I2C_Busy</i> |  |

#### 16.4.5.23 `status_t I2C_MasterTransferAbort ( I2C_Type * base, i2c_master_handle_t * handle )`

## Note

It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

## Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.         |
| <i>handle</i> | Pointer to the I2C master driver handle. |

Return values

|                            |                                             |
|----------------------------|---------------------------------------------|
| <i>kStatus_Success</i>     | A transaction was successfully aborted.     |
| <i>kStatus_I2C_Timeout</i> | Abort failure due to flags polling timeout. |

#### 16.4.5.24 void I2C\_MasterTransferHandleIRQ ( I2C\_Type \* *base*, void \* *i2cHandle* )

Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

Parameters

|                  |                                                                            |
|------------------|----------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base address.                                           |
| <i>i2cHandle</i> | Pointer to the I2C master driver handle <code>i2c_master_handle_t</code> . |

## 16.5 I2C Slave Driver

### 16.5.1 Overview

#### Data Structures

- struct [\\_i2c\\_slave\\_address](#)  
*Data structure with 7-bit Slave address and Slave address disable. [More...](#)*
- struct [\\_i2c\\_slave\\_config](#)  
*Structure with settings to initialize the I2C slave module. [More...](#)*
- struct [\\_i2c\\_slave\\_transfer](#)  
*I2C slave transfer structure. [More...](#)*
- struct [\\_i2c\\_slave\\_handle](#)  
*I2C slave handle structure. [More...](#)*

#### Typedefs

- typedef enum  
[\\_i2c\\_slave\\_address\\_register](#) [i2c\\_slave\\_address\\_register\\_t](#)  
*I2C slave address register.*
- typedef struct [\\_i2c\\_slave\\_address](#) [i2c\\_slave\\_address\\_t](#)  
*Data structure with 7-bit Slave address and Slave address disable.*
- typedef enum  
[\\_i2c\\_slave\\_address\\_qual\\_mode](#) [i2c\\_slave\\_address\\_qual\\_mode\\_t](#)  
*I2C slave address match options.*
- typedef enum [\\_i2c\\_slave\\_bus\\_speed](#) [i2c\\_slave\\_bus\\_speed\\_t](#)  
*I2C slave bus speed options.*
- typedef struct [\\_i2c\\_slave\\_config](#) [i2c\\_slave\\_config\\_t](#)  
*Structure with settings to initialize the I2C slave module.*
- typedef enum  
[\\_i2c\\_slave\\_transfer\\_event](#) [i2c\\_slave\\_transfer\\_event\\_t](#)  
*Set of events sent to the callback for non blocking slave transfers.*
- typedef struct [\\_i2c\\_slave\\_handle](#) [i2c\\_slave\\_handle\\_t](#)  
*I2C slave handle typedef.*
- typedef struct [\\_i2c\\_slave\\_transfer](#) [i2c\\_slave\\_transfer\\_t](#)  
*I2C slave transfer structure.*
- typedef void(\* [i2c\\_slave\\_transfer\\_callback\\_t](#) )(I2C\_Type \*base, volatile [i2c\\_slave\\_transfer\\_t](#) \*transfer, void \*userData)  
*Slave event callback function pointer type.*
- typedef enum [\\_i2c\\_slave\\_fsm](#) [i2c\\_slave\\_fsm\\_t](#)  
*I2C slave software finite state machine states.*
- typedef void(\* [i2c\\_isr\\_t](#) )(I2C\_Type \*base, void \*i2cHandle)  
*Typedef for interrupt handler.*

## Enumerations

- enum `_i2c_slave_flags` {  
`kI2C_SlavePendingFlag` = `I2C_STAT_SLVPENDING_MASK`,  
`kI2C_SlaveNotStretching`,  
`kI2C_SlaveSelected` = `I2C_STAT_SLVSEL_MASK`,  
`kI2C_SaveDeselected` }  
*I2C slave peripheral flags.*
- enum `_i2c_slave_address_register` {  
`kI2C_SlaveAddressRegister0` = `0U`,  
`kI2C_SlaveAddressRegister1` = `1U`,  
`kI2C_SlaveAddressRegister2` = `2U`,  
`kI2C_SlaveAddressRegister3` = `3U` }  
*I2C slave address register.*
- enum `_i2c_slave_address_qual_mode` {  
`kI2C_QualModeMask` = `0U`,  
`kI2C_QualModeExtend` }  
*I2C slave address match options.*
- enum `_i2c_slave_bus_speed`  
*I2C slave bus speed options.*
- enum `_i2c_slave_transfer_event` {  
`kI2C_SlaveAddressMatchEvent` = `0x01U`,  
`kI2C_SlaveTransmitEvent` = `0x02U`,  
`kI2C_SlaveReceiveEvent` = `0x04U`,  
`kI2C_SlaveCompletionEvent` = `0x20U`,  
`kI2C_SlaveDeselectedEvent`,  
`kI2C_SlaveAllEvents` }  
*Set of events sent to the callback for non blocking slave transfers.*
- enum `_i2c_slave_fsm`  
*I2C slave software finite state machine states.*

## Slave initialization and deinitialization

- void `I2C_SlaveGetDefaultConfig` (`i2c_slave_config_t` \*slaveConfig)  
*Provides a default configuration for the I2C slave peripheral.*
- `status_t I2C_SlaveInit` (`I2C_Type` \*base, const `i2c_slave_config_t` \*slaveConfig, `uint32_t` srcClock\_Hz)  
*Initializes the I2C slave peripheral.*
- void `I2C_SlaveSetAddress` (`I2C_Type` \*base, `i2c_slave_address_register_t` addressRegister, `uint8_t` address, `bool` addressDisable)  
*Configures Slave Address n register.*
- void `I2C_SlaveDeinit` (`I2C_Type` \*base)  
*Deinitializes the I2C slave peripheral.*
- static void `I2C_SlaveEnable` (`I2C_Type` \*base, `bool` enable)  
*Enables or disables the I2C module as slave.*

## Slave status

- static void [I2C\\_SlaveClearStatusFlags](#) (I2C\_Type \*base, uint32\_t statusMask)  
*Clears the I2C status flag state.*

## Slave bus operations

- [status\\_t I2C\\_SlaveWriteBlocking](#) (I2C\_Type \*base, const uint8\_t \*txBuff, size\_t txSize)  
*Performs a polling send transfer on the I2C bus.*
- [status\\_t I2C\\_SlaveReadBlocking](#) (I2C\_Type \*base, uint8\_t \*rxBuff, size\_t rxSize)  
*Performs a polling receive transfer on the I2C bus.*

## Slave non-blocking

- void [I2C\\_SlaveTransferCreateHandle](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle, [i2c\\_slave\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the I2C slave non-blocking APIs.*
- [status\\_t I2C\\_SlaveTransferNonBlocking](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle, uint32\_t eventMask)  
*Starts accepting slave transfers.*
- [status\\_t I2C\\_SlaveSetSendBuffer](#) (I2C\_Type \*base, volatile [i2c\\_slave\\_transfer\\_t](#) \*transfer, const void \*txData, size\_t txSize, uint32\_t eventMask)  
*Starts accepting master read from slave requests.*
- [status\\_t I2C\\_SlaveSetReceiveBuffer](#) (I2C\_Type \*base, volatile [i2c\\_slave\\_transfer\\_t](#) \*transfer, void \*rxData, size\_t rxSize, uint32\_t eventMask)  
*Starts accepting master write to slave requests.*
- static uint32\_t [I2C\\_SlaveGetReceivedAddress](#) (I2C\_Type \*base, volatile [i2c\\_slave\\_transfer\\_t](#) \*transfer)  
*Returns the slave address sent by the I2C master.*
- void [I2C\\_SlaveTransferAbort](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle)  
*Aborts the slave non-blocking transfers.*
- [status\\_t I2C\\_SlaveTransferGetCount](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.*

## Slave IRQ handler

- void [I2C\\_SlaveTransferHandleIRQ](#) (I2C\_Type \*base, void \*i2cHandle)  
*Reusable routine to handle slave interrupts.*

## 16.5.2 Data Structure Documentation

### 16.5.2.1 struct `_i2c_slave_address`

#### Data Fields

- `uint8_t address`  
*7-bit Slave address SLVADR.*
- `bool addressDisable`  
*Slave address disable SADISABLE.*

#### Field Documentation

(1) `uint8_t _i2c_slave_address::address`

(2) `bool _i2c_slave_address::addressDisable`

### 16.5.2.2 struct `_i2c_slave_config`

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the `I2C_SlaveGetDefaultConfig()` function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

#### Data Fields

- `i2c_slave_address_t address0`  
*Slave's 7-bit address and disable.*
- `i2c_slave_address_t address1`  
*Alternate slave 7-bit address and disable.*
- `i2c_slave_address_t address2`  
*Alternate slave 7-bit address and disable.*
- `i2c_slave_address_t address3`  
*Alternate slave 7-bit address and disable.*
- `i2c_slave_address_qual_mode_t qualMode`  
*Qualify mode for slave address 0.*
- `uint8_t qualAddress`  
*Slave address qualifier for address 0.*
- `i2c_slave_bus_speed_t busSpeed`  
*Slave bus speed mode.*
- `bool enableSlave`  
*Enable slave mode.*

## Field Documentation

- (1) `i2c_slave_address_t_i2c_slave_config::address0`
- (2) `i2c_slave_address_t_i2c_slave_config::address1`
- (3) `i2c_slave_address_t_i2c_slave_config::address2`
- (4) `i2c_slave_address_t_i2c_slave_config::address3`
- (5) `i2c_slave_address_qual_mode_t_i2c_slave_config::qualMode`
- (6) `uint8_t_i2c_slave_config::qualAddress`
- (7) `i2c_slave_bus_speed_t_i2c_slave_config::busSpeed`

If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The `busSpeed` value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the `busSpeed` mode is unknown at compile time, use the longest data setup time `kI2C_SlaveStandardMode` (250 ns)

- (8) `bool_i2c_slave_config::enableSlave`

### 16.5.2.3 struct `_i2c_slave_transfer`

#### Data Fields

- `i2c_slave_handle_t * handle`  
*Pointer to handle that contains this transfer.*
- `i2c_slave_transfer_event_t event`  
*Reason the callback is being invoked.*
- `uint8_t receivedAddress`  
*Matching address send by master.*
- `uint32_t eventMask`  
*Mask of enabled events.*
- `uint8_t * rxData`  
*Transfer buffer for receive data.*
- `const uint8_t * txData`  
*Transfer buffer for transmit data.*
- `size_t txSize`  
*Transfer size.*
- `size_t rxSize`  
*Transfer size.*
- `size_t transferredCount`  
*Number of bytes transferred during this transfer.*
- `status_t completionStatus`  
*Success or error code describing how the transfer completed.*

**Field Documentation**

- (1) `i2c_slave_handle_t* _i2c_slave_transfer::handle`
- (2) `i2c_slave_transfer_event_t _i2c_slave_transfer::event`
- (3) `uint8_t _i2c_slave_transfer::receivedAddress`  
7-bits plus R/nW bit0
- (4) `uint32_t _i2c_slave_transfer::eventMask`
- (5) `size_t _i2c_slave_transfer::transferredCount`
- (6) `status_t _i2c_slave_transfer::completionStatus`

Only applies for `kI2C_SlaveCompletionEvent`.

**16.5.2.4 struct \_i2c\_slave\_handle**

Note

The contents of this structure are private and subject to change.

**Data Fields**

- volatile `i2c_slave_transfer_t transfer`  
*I2C slave transfer.*
- volatile bool `isBusy`  
*Whether transfer is busy.*
- volatile `i2c_slave_fsm_t slaveFsm`  
*slave transfer state machine.*
- `i2c_slave_transfer_callback_t callback`  
*Callback function called at transfer event.*
- void \* `userData`  
*Callback parameter passed to callback.*

## Field Documentation

- (1) `volatile i2c_slave_transfer_t i2c_slave_handle::transfer`
- (2) `volatile bool i2c_slave_handle::isBusy`
- (3) `volatile i2c_slave_fsm_t i2c_slave_handle::slaveFsm`
- (4) `i2c_slave_transfer_callback_t i2c_slave_handle::callback`
- (5) `void* i2c_slave_handle::userData`

## 16.5.3 Typedef Documentation

**16.5.3.1** `typedef enum i2c_slave_address_register i2c_slave_address_register_t`

**16.5.3.2** `typedef struct i2c_slave_address i2c_slave_address_t`

**16.5.3.3** `typedef enum i2c_slave_address_qual_mode i2c_slave_address_qual_mode_t`

**16.5.3.4** `typedef enum i2c_slave_bus_speed i2c_slave_bus_speed_t`

**16.5.3.5** `typedef struct i2c_slave_config i2c_slave_config_t`

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the [I2C\\_SlaveGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

**16.5.3.6** `typedef enum i2c_slave_transfer_event 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.

**16.5.3.7 typedef struct \_i2c\_slave\_handle i2c\_slave\_handle\_t**

**16.5.3.8 typedef void(\* i2c\_slave\_transfer\_callback\_t)(I2C\_Type \*base, volatile i2c\_slave\_transfer\_t \*transfer, void \*userData)**

This callback is used only for the slave non-blocking transfer API. To install a callback, use the I2C\_SlaveSetCallback() function after you have created a handle.

## Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address for the I2C instance on which the event occurred.                       |
| <i>transfer</i> | Pointer to transfer descriptor containing values passed to and/or from the callback. |
| <i>userData</i> | Arbitrary pointer-sized value passed from the application.                           |

**16.5.3.9 typedef void(\* i2c\_isr\_t)(I2C\_Type \*base, void \*i2cHandle)**

## 16.5.4 Enumeration Type Documentation

### 16.5.4.1 enum \_i2c\_slave\_flags

#### Note

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

#### Enumerator

***kI2C\_SlavePendingFlag*** The I2C module is waiting for software interaction.

***kI2C\_SlaveNotStretching*** Indicates whether the slave is currently stretching clock (0 = yes, 1 = no).

***kI2C\_SlaveSelected*** Indicates whether the slave is selected by an address match.

***kI2C\_SaveDeselected*** Indicates that slave was previously deselected (deselect event took place, w1c).

### 16.5.4.2 enum \_i2c\_slave\_address\_register

#### Enumerator

***kI2C\_SlaveAddressRegister0*** Slave Address 0 register.

***kI2C\_SlaveAddressRegister1*** Slave Address 1 register.

***kI2C\_SlaveAddressRegister2*** Slave Address 2 register.

***kI2C\_SlaveAddressRegister3*** Slave Address 3 register.

### 16.5.4.3 enum \_i2c\_slave\_address\_qual\_mode

#### Enumerator

***kI2C\_QualModeMask*** The SLVQUAL0 field (qualAddress) is used as a logical mask for matching address0.

***kI2C\_QualModeExtend*** The SLVQUAL0 (qualAddress) field is used to extend address 0 matching in a range of addresses.

#### 16.5.4.4 enum `_i2c_slave_bus_speed`

#### 16.5.4.5 enum `_i2c_slave_transfer_event`

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to `I2C_SlaveTransferNonBlocking()` in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its `transfer` parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

**`kI2C_SlaveAddressMatchEvent`** Received the slave address after a start or repeated start.

**`kI2C_SlaveTransmitEvent`** Callback is requested to provide data to transmit (slave-transmitter role).

**`kI2C_SlaveReceiveEvent`** Callback is requested to provide a buffer in which to place received data (slave-receiver role).

**`kI2C_SlaveCompletionEvent`** All data in the active transfer have been consumed.

**`kI2C_SlaveDeselectedEvent`** The slave function has become deselected (SLVSEL flag changing from 1 to 0).

**`kI2C_SlaveAllEvents`** Bit mask of all available events.

### 16.5.5 Function Documentation

#### 16.5.5.1 void `I2C_SlaveGetDefaultConfig ( i2c_slave_config_t * slaveConfig )`

This function provides the following default configuration for the I2C slave peripheral:

```
* slaveConfig->enableSlave = true;
* slaveConfig->address0.disable = false;
* slaveConfig->address0.address = 0u;
* slaveConfig->address1.disable = true;
* slaveConfig->address2.disable = true;
* slaveConfig->address3.disable = true;
* slaveConfig->busSpeed = kI2C_SlaveStandardMode;
*
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with `I2C_SlaveInit()`. Be sure to override at least the `address0.address` member of the configuration structure with the desired slave address.

## Parameters

|     |                    |                                                                                                                    |
|-----|--------------------|--------------------------------------------------------------------------------------------------------------------|
| out | <i>slaveConfig</i> | User provided configuration structure that is set to default values. Refer to <a href="#">i2c_slave_config_t</a> . |
|-----|--------------------|--------------------------------------------------------------------------------------------------------------------|

### 16.5.5.2 **status\_t I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *srcClock\_Hz* )**

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

## Parameters

|                    |                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | The I2C peripheral base address.                                                                                                                            |
| <i>slaveConfig</i> | User provided peripheral configuration. Use <a href="#">I2C_SlaveGetDefaultConfig()</a> to get a set of defaults that you can override.                     |
| <i>srcClock_Hz</i> | Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock. |

### 16.5.5.3 **void I2C\_SlaveSetAddress ( I2C\_Type \* *base*, i2c\_slave\_address\_register\_t *addressRegister*, uint8\_t *address*, bool *addressDisable* )**

This function writes new value to Slave Address register.

## Parameters

|                         |                                                                                                      |
|-------------------------|------------------------------------------------------------------------------------------------------|
| <i>base</i>             | The I2C peripheral base address.                                                                     |
| <i>address-Register</i> | The module supports multiple address registers. The parameter determines which one shall be changed. |
| <i>address</i>          | The slave address to be stored to the address register for matching.                                 |
| <i>addressDisable</i>   | Disable matching of the specified address register.                                                  |

### 16.5.5.4 **void I2C\_SlaveDeinit ( I2C\_Type \* *base* )**

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

**16.5.5.5** `static void I2C_SlaveEnable ( I2C_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | The I2C peripheral base address.    |
| <i>enable</i> | True to enable or false to disable. |

**16.5.5.6** `static void I2C_SlaveClearStatusFlags ( I2C_Type * base, uint32_t statusMask ) [inline], [static]`

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

Parameters

|                   |                                                                                                                                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I2C peripheral base address.                                                                                                                                                                                             |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_i2c_slave_flags</a> enumerators OR'd together. You may pass the result of a previous call to <code>I2C_SlaveGetStatusFlags()</code> . |

See Also

[\\_i2c\\_slave\\_flags](#).

**16.5.5.7** `status_t I2C_SlaveWriteBlocking ( I2C_Type * base, const uint8_t * txBuff, size_t txSize )`

The function executes blocking address phase and blocking data phase.

## Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                   |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

## Returns

kStatus\_Success Data has been sent.

kStatus\_Fail Unexpected slave state (master data write while master read from slave is expected).

#### 16.5.5.8 **status\_t I2C\_SlaveReadBlocking ( I2C\_Type \* *base*, uint8\_t \* *rxBuff*, size\_t *rxSize* )**

The function executes blocking address phase and blocking data phase.

## Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                   |
| <i>rxBuff</i> | The pointer to the data to be transferred.         |
| <i>rxSize</i> | The length in bytes of the data to be transferred. |

## Returns

kStatus\_Success Data has been received.

kStatus\_Fail Unexpected slave state (master data read while master write to slave is expected).

#### 16.5.5.9 **void I2C\_SlaveTransferCreateHandle ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, i2c\_slave\_transfer\_callback\_t *callback*, void \* *userData* )**

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [I2C\\_SlaveTransferAbort\(\)](#) API shall be called.

## Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The I2C peripheral base address.                             |
| out | <i>handle</i>   | Pointer to the I2C slave driver handle.                      |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

#### 16.5.5.10 `status_t I2C_SlaveTransferNonBlocking ( I2C_Type * base, i2c_slave_handle_t * handle, uint32_t eventMask )`

Call this API after calling [I2C\\_SlaveInit\(\)](#) and [I2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to [I2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

If no slave Tx transfer is busy, a master read from slave request invokes [kI2C\\_SlaveTransmitEvent](#) callback. If no slave Rx transfer is busy, a master write to slave request invokes [kI2C\\_SlaveReceiveEvent](#) callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [kI2C\\_SlaveTransmitEvent](#) and [kI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

##### Parameters

|                  |                                                                                                                                                                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base address.                                                                                                                                                                                                                                                                   |
| <i>handle</i>    | Pointer to <a href="#">i2c_slave_handle_t</a> structure which stores the transfer state.                                                                                                                                                                                                           |
| <i>eventMask</i> | Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events. |

##### Return values

|                                  |                                                           |
|----------------------------------|-----------------------------------------------------------|
| <a href="#">kStatus_Success</a>  | Slave transfers were successfully started.                |
| <a href="#">kStatus_I2C_Busy</a> | Slave transfers have already been started on this handle. |

### 16.5.5.11 `status_t I2C_SlaveSetSendBuffer ( I2C_Type * base, volatile i2c_slave_transfer_t * transfer, const void * txData, size_t txSize, uint32_t eventMask )`

The function can be called in response to [kI2C\\_SlaveTransmitEvent](#) callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [kI2C\\_SlaveTransmitEvent](#) and [kI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

#### Parameters

|                  |                                                                                                                                                                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base address.                                                                                                                                                                                                                                                                   |
| <i>transfer</i>  | Pointer to <a href="#">i2c_slave_transfer_t</a> structure.                                                                                                                                                                                                                                         |
| <i>txData</i>    | Pointer to data to send to master.                                                                                                                                                                                                                                                                 |
| <i>txSize</i>    | Size of <i>txData</i> in bytes.                                                                                                                                                                                                                                                                    |
| <i>eventMask</i> | Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events. |

#### Return values

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

### 16.5.5.12 `status_t I2C_SlaveSetReceiveBuffer ( I2C_Type * base, volatile i2c_slave_transfer_t * transfer, void * rxData, size_t rxSize, uint32_t eventMask )`

The function can be called in response to [kI2C\\_SlaveReceiveEvent](#) callback to start a new slave Rx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [kI2C\\_SlaveTransmitEvent](#) and [kI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

## Parameters

|                  |                                                                                                                                                                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base address.                                                                                                                                                                                                                                                                   |
| <i>transfer</i>  | Pointer to <a href="#">i2c_slave_transfer_t</a> structure.                                                                                                                                                                                                                                         |
| <i>rxData</i>    | Pointer to data to store data from master.                                                                                                                                                                                                                                                         |
| <i>rxSize</i>    | Size of rxData in bytes.                                                                                                                                                                                                                                                                           |
| <i>eventMask</i> | Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events. |

## Return values

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

#### 16.5.5.13 `static uint32_t I2C_SlaveGetReceivedAddress ( I2C_Type * base, volatile i2c_slave_transfer_t * transfer ) [inline], [static]`

This function should only be called from the address match event callback [kI2C\\_SlaveAddressMatch-Event](#).

## Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>base</i>     | The I2C peripheral base address. |
| <i>transfer</i> | The I2C slave transfer.          |

## Returns

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

#### 16.5.5.14 `void I2C_SlaveTransferAbort ( I2C_Type * base, i2c_slave_handle_t * handle )`

## Note

This API could be called at any time to stop slave for handling the bus events.

## Parameters

|               |                                                                                       |
|---------------|---------------------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                                      |
| <i>handle</i> | Pointer to <code>i2c_slave_handle_t</code> structure which stores the transfer state. |

## Return values

|                         |  |
|-------------------------|--|
| <i>kStatus_Success</i>  |  |
| <i>kStatus_I2C_Idle</i> |  |

#### 16.5.5.15 `status_t I2C_SlaveTransferGetCount ( I2C_Type * base, i2c_slave_handle_t * handle, size_t * count )`

## Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | I2C base pointer.                                                   |
| <i>handle</i> | pointer to <code>i2c_slave_handle_t</code> 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. |

#### 16.5.5.16 `void I2C_SlaveTransferHandleIRQ ( I2C_Type * base, void * i2cHandle )`

## Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

## Parameters

|                  |                                                                                       |
|------------------|---------------------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base address.                                                      |
| <i>i2cHandle</i> | Pointer to <code>i2c_slave_handle_t</code> structure which stores the transfer state. |

# Chapter 17

## IOCON: I/O pin configuration

### 17.1 Overview

The MCUXpresso SDK provides Peripheral driver for the I/O pin configuration (IOCON) module of MCUXpresso SDK devices.

### 17.2 Function groups

#### 17.2.1 Pin mux set

The function `IOCONPinMuxSet()` set pinmux for single pin according to selected configuration.

#### 17.2.2 Pin mux set

The function `IOCON_SetPinMuxing()` set pinmux for group of pins according to selected configuration.

### 17.3 Typical use case

Example use of IOCON API to selection of GPIO mode. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/iocon`

### Files

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

### Data Structures

- struct [\\_iocon\\_group](#)  
*Array of IOCON pin definitions passed to `IOCON_SetPinMuxing()` must be in this format. [More...](#)*

### Typedefs

- typedef struct [\\_iocon\\_group](#) [iocon\\_group\\_t](#)  
*Array of IOCON pin definitions passed to `IOCON_SetPinMuxing()` must be in this format.*

### Functions

- `__STATIC_INLINE void IOCON\_PinMuxSet (IOCON_Type *base, uint8_t ionumber, uint32_t modefunc)`  
*IOCON function and mode selection definitions.*
- `__STATIC_INLINE void IOCON\_SetPinMuxing (IOCON_Type *base, const iocon\_group\_t *pinArray, uint32_t arrayLength)`  
*Set all I/O Control pin muxing.*

## Driver version

- #define `LPC_IOCON_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 2)`)  
*IOCON driver version 2.0.2.*

## 17.4 Data Structure Documentation

### 17.4.1 struct `_iocon_group`

## 17.5 Macro Definition Documentation

### 17.5.1 #define `LPC_IOCON_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 2)`)

## 17.6 Function Documentation

### 17.6.1 `__STATIC_INLINE` void `IOCON_PinMuxSet` ( `IOCON_Type` \* *base*, `uint8_t` *ionumber*, `uint32_t` *modefunc* )

Note

See the User Manual for specific modes and functions supported by the various pins. Sets I/O Control pin mux

Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>base</i>     | : The base of IOCON peripheral on the chip  |
| <i>ionumber</i> | : GPIO number to mux                        |
| <i>modefunc</i> | : OR'ed values of type <code>IOCON_*</code> |

Returns

Nothing

### 17.6.2 `__STATIC_INLINE` void `IOCON_SetPinMuxing` ( `IOCON_Type` \* *base*, `const iocon_group_t` \* *pinArray*, `uint32_t` *arrayLength* )

Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>base</i>        | : The base of IOCON peripheral on the chip |
| <i>pinArray</i>    | : Pointer to array of pin mux selections   |
| <i>arrayLength</i> | : Number of entries in pinArray            |

Returns

Nothing

## Chapter 18

# SPI: Serial Peripheral Interface Driver

### 18.1 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 the purpose of optimization/customization. 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.

### 18.2 Typical use case

#### 18.2.1 SPI master transfer using an interrupt method

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

### Modules

- [SPI Driver](#)

## 18.3 SPI Driver

### 18.3.1 Overview

This section describes the programming interface of the SPI driver.

#### Files

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

#### Data Structures

- struct [\\_spi\\_delay\\_config](#)  
*SPI delay time configure structure. [More...](#)*
- struct [\\_spi\\_master\\_config](#)  
*SPI master user configure structure. [More...](#)*
- struct [\\_spi\\_slave\\_config](#)  
*SPI slave user configure structure. [More...](#)*
- struct [\\_spi\\_transfer](#)  
*SPI transfer structure. [More...](#)*
- struct [\\_spi\\_master\\_handle](#)  
*SPI transfer handle structure. [More...](#)*

#### Macros

- #define [SPI\\_DUMMYDATA](#) (0x00U)  
*SPI dummy transfer data, the data is sent while txBuff is NULL.*
- #define [SPI\\_RETRY\\_TIMES](#) 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*

#### Typedefs

- typedef enum [\\_spi\\_shift\\_direction](#) [spi\\_shift\\_direction\\_t](#)  
*SPI data shifter direction options.*
- typedef enum [\\_spi\\_clock\\_polarity](#) [spi\\_clock\\_polarity\\_t](#)  
*SPI clock polarity configuration.*
- typedef enum [\\_spi\\_clock\\_phase](#) [spi\\_clock\\_phase\\_t](#)  
*SPI clock phase configuration.*
- typedef enum [\\_spi\\_ssel](#) [spi\\_ssel\\_t](#)  
*Slave select.*
- typedef enum [\\_spi\\_spol](#) [spi\\_spol\\_t](#)  
*ssel polarity*
- typedef enum [\\_spi\\_data\\_width](#) [spi\\_data\\_width\\_t](#)  
*Transfer data width.*

- typedef struct `_spi_delay_config spi_delay_config_t`  
*SPI delay time configure structure.*
- typedef struct `_spi_master_config spi_master_config_t`  
*SPI master user configure structure.*
- typedef struct `_spi_slave_config spi_slave_config_t`  
*SPI slave user configure structure.*
- typedef struct `_spi_transfer spi_transfer_t`  
*SPI transfer structure.*
- typedef struct `_spi_master_handle spi_master_handle_t`  
*Master handle type.*
- typedef struct `spi_master_handle_t spi_slave_handle_t`  
*Slave handle type.*
- typedef void(\* `spi_master_callback_t`)(SPI\_Type \*base, `spi_master_handle_t` \*handle, `status_t` status, void \*userData)  
*SPI master callback for finished transmit.*
- typedef void(\* `spi_slave_callback_t`)(SPI\_Type \*base, `spi_slave_handle_t` \*handle, `status_t` status, void \*userData)  
*SPI slave callback for finished transmit.*

## Enumerations

- enum `_spi_xfer_option` {  
`kSPI_EndOfFrame` = (SPI\_TXDATCTL\_EOF\_MASK),  
`kSPI_EndOfTransfer`,  
`kSPI_ReceiveIgnore` = (SPI\_TXDATCTL\_RXIGNORE\_MASK) }  
*SPI transfer option.*
- enum `_spi_shift_direction` {  
`kSPI_MsbFirst` = 0U,  
`kSPI_LsbFirst` = 1U }  
*SPI data shifter direction options.*
- enum `_spi_clock_polarity` {  
`kSPI_ClockPolarityActiveHigh` = 0x0U,  
`kSPI_ClockPolarityActiveLow` = 0x1U }  
*SPI clock polarity configuration.*
- enum `_spi_clock_phase` {  
`kSPI_ClockPhaseFirstEdge` = 0x0U,  
`kSPI_ClockPhaseSecondEdge` = 0x1U }  
*SPI clock phase configuration.*
- enum `_spi_ssel` { `kSPI_Ssel0Assert` = (int)(~SPI\_TXDATCTL\_TXSSEL0\_N\_MASK) }  
*Slave select.*
- enum `_spi_spol`  
*ssel polarity*
- enum `_spi_data_width` {

- ```

kSPI_Data4Bits = 3,
kSPI_Data5Bits = 4,
kSPI_Data6Bits = 5,
kSPI_Data7Bits = 6,
kSPI_Data8Bits = 7,
kSPI_Data9Bits = 8,
kSPI_Data10Bits = 9,
kSPI_Data11Bits = 10,
kSPI_Data12Bits = 11,
kSPI_Data13Bits = 12,
kSPI_Data14Bits = 13,
kSPI_Data15Bits = 14,
kSPI_Data16Bits = 15 }

```
- Transfer data width.*
- enum {

```

kStatus_SPI_Busy = MAKE_STATUS(kStatusGroup_LPC_MINISPI, 0),
kStatus_SPI_Idle = MAKE_STATUS(kStatusGroup_LPC_MINISPI, 1),
kStatus_SPI_Error = MAKE_STATUS(kStatusGroup_LPC_MINISPI, 2),
kStatus_SPI_BaudrateNotSupport,
kStatus_SPI_Timeout = MAKE_STATUS(kStatusGroup_LPC_MINISPI, 4) }

```
- SPI transfer status.*
- enum `_spi_interrupt_enable` {

```

kSPI_RxReadyInterruptEnable = SPI_INTENSET_RXRDYEN_MASK,
kSPI_TxReadyInterruptEnable = SPI_INTENSET_TXRDYEN_MASK,
kSPI_RxOverrunInterruptEnable = SPI_INTENSET_RXOVEN_MASK,
kSPI_TxUnderrunInterruptEnable = SPI_INTENSET_TXUREN_MASK,
kSPI_SlaveSelectAssertInterruptEnable = SPI_INTENSET_SSAEN_MASK,
kSPI_SlaveSelectDeassertInterruptEnable = SPI_INTENSET_SSDEN_MASK }

```
- SPI interrupt sources.*
- enum `_spi_status_flags` {

```

kSPI_RxReadyFlag = SPI_STAT_RXRDY_MASK,
kSPI_TxReadyFlag = SPI_STAT_TXRDY_MASK,
kSPI_RxOverrunFlag = SPI_STAT_RXOV_MASK,
kSPI_TxUnderrunFlag = SPI_STAT_TXUR_MASK,
kSPI_SlaveSelectAssertFlag = SPI_STAT_SSA_MASK,
kSPI_SlaveSelectDeassertFlag = SPI_STAT_SSD_MASK,
kSPI_StallFlag = SPI_STAT_STALLED_MASK,
kSPI_EndTransferFlag = SPI_STAT_ENDTRANSFER_MASK,
kSPI_MasterIdleFlag = SPI_STAT_MSTIDLE_MASK }

```
- SPI status flags.*

## Functions

- uint32\_t `SPI_GetInstance` (SPI\_Type \*base)  
*Returns instance number for SPI peripheral base address.*

## Driver version

- #define `FSL_SPI_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 6)`)  
*SPI driver version.*

## Initialization and deinitialization

- void `SPI_MasterGetDefaultConfig` (`spi_master_config_t *config`)  
*Sets the SPI master configuration structure to default values.*
- `status_t 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.*
- `status_t 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)  
*Enable or disable the SPI Master or Slave.*

## Status

- static `uint32_t SPI_GetStatusFlags` (`SPI_Type *base`)  
*Gets the status flag.*
- static void `SPI_ClearStatusFlags` (`SPI_Type *base`, `uint32_t mask`)  
*Clear the status flag.*

## Interrupts

- static void `SPI_EnableInterrupts` (`SPI_Type *base`, `uint32_t irq`)  
*Enables the interrupt for the SPI.*
- static void `SPI_DisableInterrupts` (`SPI_Type *base`, `uint32_t irq`)  
*Disables the interrupt for the SPI.*

## Bus Operations

- static bool `SPI_IsMaster` (`SPI_Type *base`)  
*Returns whether the SPI module is in master mode.*
- `status_t SPI_MasterSetBaudRate` (`SPI_Type *base`, `uint32_t baudrate_Bps`, `uint32_t srcClock_Hz`)  
*Sets the baud rate for SPI transfer.*
- static void `SPI_WriteData` (`SPI_Type *base`, `uint16_t data`)  
*Writes a data into the SPI data register directly.*
- static void `SPI_WriteConfigFlags` (`SPI_Type *base`, `uint32_t configFlags`)  
*Writes a data into the SPI TXCTL register directly.*

- void `SPI_WriteDataWithConfigFlags` (SPI\_Type \*base, uint16\_t data, uint32\_t configFlags)  
*Writes a data control info and data into the SPI TX register directly.*
- static uint32\_t `SPI_ReadData` (SPI\_Type \*base)  
*Gets a data from the SPI data register.*
- void `SPI_SetTransferDelay` (SPI\_Type \*base, const spi\_delay\_config\_t \*config)  
*Set delay time for transfer.*
- void `SPI_SetDummyData` (SPI\_Type \*base, uint16\_t dummyData)  
*Set up the dummy data.*
- status\_t `SPI_MasterTransferBlocking` (SPI\_Type \*base, spi\_transfer\_t \*xfer)  
*Transfers a block of data using a polling method.*

## Transactional

- status\_t `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_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 master transfer count.*
- void `SPI_MasterTransferAbort` (SPI\_Type \*base, spi\_master\_handle\_t \*handle)  
*SPI master aborts a transfer using an interrupt.*
- void `SPI_MasterTransferHandleIRQ` (SPI\_Type \*base, spi\_master\_handle\_t \*handle)  
*Interrupts the handler for the SPI.*
- status\_t `SPI_SlaveTransferCreateHandle` (SPI\_Type \*base, spi\_slave\_handle\_t \*handle, spi\_slave\_callback\_t callback, void \*userData)  
*Initializes the SPI slave handle.*
- 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 slave transfer count.*
- static void `SPI_SlaveTransferAbort` (SPI\_Type \*base, spi\_slave\_handle\_t \*handle)  
*SPI slave aborts a transfer using an interrupt.*
- void `SPI_SlaveTransferHandleIRQ` (SPI\_Type \*base, spi\_slave\_handle\_t \*handle)  
*Interrupts a handler for the SPI slave.*

## 18.3.2 Data Structure Documentation

### 18.3.2.1 struct spi\_delay\_config

#### Data Fields

- uint8\_t `preDelay`

- `uint8_t postDelay`  
*Delay between SSEL assertion and the beginning of transfer.*
- `uint8_t frameDelay`  
*Delay between the end of transfer and SSEL deassertion.*
- `uint8_t transferDelay`  
*Delay between frame to frame.*
- `uint8_t transferDelay`  
*Delay between transfer to transfer.*

### Field Documentation

- (1) `uint8_t _spi_delay_config::preDelay`
- (2) `uint8_t _spi_delay_config::postDelay`
- (3) `uint8_t _spi_delay_config::frameDelay`
- (4) `uint8_t _spi_delay_config::transferDelay`

### 18.3.2.2 struct \_spi\_master\_config

#### Data Fields

- `bool enableLoopback`  
*Enable loopback for test purpose.*
- `bool enableMaster`  
*Enable SPI at initialization time.*
- `uint32_t baudRate_Bps`  
*Baud Rate for SPI in Hz.*
- `spi_clock_polarity_t clockPolarity`  
*Clock polarity.*
- `spi_clock_phase_t clockPhase`  
*Clock phase.*
- `spi_shift_direction_t direction`  
*MSB or LSB.*
- `uint8_t dataWidth`  
*Width of the data.*
- `spi_ssel_t sselNumber`  
*Slave select number.*
- `spi_spol_t sselPolarity`  
*Configure active CS polarity.*
- `spi_delay_config_t delayConfig`  
*Configure for delay time.*

## Field Documentation

### (1) spi\_delay\_config\_t spi\_master\_config::delayConfig

#### 18.3.2.3 struct spi\_slave\_config

##### Data Fields

- bool `enableSlave`  
*Enable SPI at initialization time.*
- `spi_clock_polarity_t` `clockPolarity`  
*Clock polarity.*
- `spi_clock_phase_t` `clockPhase`  
*Clock phase.*
- `spi_shift_direction_t` `direction`  
*MSB or LSB.*
- `uint8_t` `dataWidth`  
*Width of the data.*
- `spi_spol_t` `sselPolarity`  
*Configure active CS polarity.*

#### 18.3.2.4 struct spi\_transfer

##### Data Fields

- `uint8_t *` `txData`  
*Send buffer.*
- `uint8_t *` `rxData`  
*Receive buffer.*
- `size_t` `dataSize`  
*Transfer bytes.*
- `uint32_t` `configFlags`  
*Additional option to control transfer `_spi_xfer_option`.*

## Field Documentation

### (1) uint32\_t spi\_transfer::configFlags

#### 18.3.2.5 struct spi\_master\_handle

##### Data Fields

- `uint8_t *` `volatile txData`  
*Transfer buffer.*
- `uint8_t *` `volatile rxData`  
*Receive buffer.*
- `volatile size_t` `txRemainingBytes`  
*Number of data to be transmitted [in bytes].*
- `volatile size_t` `rxRemainingBytes`  
*Number of data to be received [in bytes].*

- `size_t totalByteCount`  
*A number of transfer bytes.*
- `volatile uint32_t state`  
*SPI internal state.*
- `spi_master_callback_t callback`  
*SPI callback.*
- `void * userData`  
*Callback parameter.*
- `uint8_t dataWidth`  
*Width of the data [Valid values: 1 to 16].*
- `uint32_t lastCommand`  
*Last command for transfer.*

## Field Documentation

(1) `uint32_t _spi_master_handle::lastCommand`

### 18.3.3 Macro Definition Documentation

18.3.3.1 `#define FSL_SPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))`

18.3.3.2 `#define SPI_DUMMYDATA (0x00U)`

18.3.3.3 `#define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

### 18.3.4 Typedef Documentation

18.3.4.1 `typedef enum _spi_shift_direction spi_shift_direction_t`

18.3.4.2 `typedef enum _spi_clock_polarity spi_clock_polarity_t`

18.3.4.3 `typedef enum _spi_clock_phase spi_clock_phase_t`

18.3.4.4 `typedef struct _spi_delay_config spi_delay_config_t`

18.3.4.5 `typedef struct _spi_master_config spi_master_config_t`

18.3.4.6 `typedef struct _spi_slave_config spi_slave_config_t`

### 18.3.5 Enumeration Type Documentation

#### 18.3.5.1 `enum _spi_xfer_option`

Enumerator

*kSPI\_EndOfFrame* Add delay at the end of each frame(the last clk edge).

*kSPI\_EndOfTransfer* Re-assert the CS signal after transfer finishes to deselect slave.

*kSPI\_ReceiveIgnore* Ignore the receive data.

#### 18.3.5.2 `enum _spi_shift_direction`

Enumerator

*kSPI\_MsbFirst* Data transfers start with most significant bit.

*kSPI\_LsbFirst* Data transfers start with least significant bit.

### 18.3.5.3 enum \_spi\_clock\_polarity

Enumerator

*kSPI\_ClockPolarityActiveHigh* Active-high SPI clock (idles low).

*kSPI\_ClockPolarityActiveLow* Active-low SPI clock (idles high).

### 18.3.5.4 enum \_spi\_clock\_phase

Enumerator

*kSPI\_ClockPhaseFirstEdge* First edge on SCK occurs at the middle of the first cycle of a data transfer.

*kSPI\_ClockPhaseSecondEdge* First edge on SCK occurs at the start of the first cycle of a data transfer.

### 18.3.5.5 enum \_spi\_ssel

Enumerator

*kSPI\_Ssel0Assert* Slave select 0.

### 18.3.5.6 enum \_spi\_data\_width

Enumerator

*kSPI\_Data4Bits* 4 bits data width

*kSPI\_Data5Bits* 5 bits data width

*kSPI\_Data6Bits* 6 bits data width

*kSPI\_Data7Bits* 7 bits data width

*kSPI\_Data8Bits* 8 bits data width

*kSPI\_Data9Bits* 9 bits data width

*kSPI\_Data10Bits* 10 bits data width

*kSPI\_Data11Bits* 11 bits data width

*kSPI\_Data12Bits* 12 bits data width

*kSPI\_Data13Bits* 13 bits data width

*kSPI\_Data14Bits* 14 bits data width

*kSPI\_Data15Bits* 15 bits data width

*kSPI\_Data16Bits* 16 bits data width

### 18.3.5.7 anonymous enum

Enumerator

*kStatus\_SPI\_Busy* SPI bus is busy.

*kStatus\_SPI\_Idle* SPI is idle.  
*kStatus\_SPI\_Error* SPI error.  
*kStatus\_SPI\_BaudrateNotSupport* Baudrate is not support in current clock source.  
*kStatus\_SPI\_Timeout* SPI Timeout polling status flags.

### 18.3.5.8 enum \_spi\_interrupt\_enable

Enumerator

*kSPI\_RxReadyInterruptEnable* Rx ready interrupt.  
*kSPI\_TxReadyInterruptEnable* Tx ready interrupt.  
*kSPI\_RxOverrunInterruptEnable* Rx overrun interrupt.  
*kSPI\_TxUnderrunInterruptEnable* Tx underrun interrupt.  
*kSPI\_SlaveSelectAssertInterruptEnable* Slave select assert interrupt.  
*kSPI\_SlaveSelectDeassertInterruptEnable* Slave select deassert interrupt.

### 18.3.5.9 enum \_spi\_status\_flags

Enumerator

*kSPI\_RxReadyFlag* Receive ready flag.  
*kSPI\_TxReadyFlag* Transmit ready flag.  
*kSPI\_RxOverrunFlag* Receive overrun flag.  
*kSPI\_TxUnderrunFlag* Transmit underrun flag.  
*kSPI\_SlaveSelectAssertFlag* Slave select assert flag.  
*kSPI\_SlaveSelectDeassertFlag* slave select deassert flag.  
*kSPI\_StallFlag* Stall flag.  
*kSPI\_EndTransferFlag* End transfer bit.  
*kSPI\_MasterIdleFlag* Master in idle status flag.

## 18.3.6 Function Documentation

### 18.3.6.1 uint32\_t SPI\_GetInstance ( SPI\_Type \* *base* )

### 18.3.6.2 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 |
|---------------|------------------------------------|

### 18.3.6.3 `status_t 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 = 500000,
    ...
};
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.                   |

### 18.3.6.4 `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);
```

## Parameters

|               |  |
|---------------|--|
| <i>config</i> | pointer to slave configuration structure |
|---------------|--|

### 18.3.6.5 `status_t 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\\_SlaveGetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
spi_slave_config_t config = {
    .polarity = kSPI_ClockPolarityActiveHigh;
    .phase = kSPI_ClockPhaseFirstEdge;
    .direction = kSPI_MsbFirst;
    ...
};
SPI_SlaveInit(SPI0, &config);
```

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI base pointer                         |
| <i>config</i> | pointer to slave configuration structure |

### 18.3.6.6 void SPI\_Deinit ( SPI\_Type \* *base* )

Calling this API resets the SPI module, gates the SPI clock. Disable the fifo if enabled. The SPI module can't work unless calling the SPI\_MasterInit/SPI\_SlaveInit to initialize module.

Parameters

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

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

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI base pointer                             |
| <i>enable</i> | or disable ( true = enable, false = disable) |

### 18.3.6.8 static uint32\_t SPI\_GetStatusFlags ( SPI\_Type \* *base* ) [inline], [static]

Parameters

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

Returns

SPI Status, use status flag to AND [\\_spi\\_status\\_flags](#) could get the related status.

### 18.3.6.9 static void SPI\_ClearStatusFlags ( SPI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |   |
|-------------|---|
| <i>base</i> | SPI base pointer  |
| <i>mask</i> | SPI Status, use status flag to AND <code>_spi_status_flags</code> could get the related status. |

### 18.3.6.10 `static void SPI_EnableInterrupts ( SPI_Type * base, uint32_t irqs ) [inline], [static]`

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | SPI base pointer   |
| <i>irqs</i> | SPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• <code>kSPI_RxReadyInterruptEnable</code></li> <li>• <code>kSPI_TxReadyInterruptEnable</code></li> </ul> |

### 18.3.6.11 `static void SPI_DisableInterrupts ( SPI_Type * base, uint32_t irqs ) [inline], [static]`

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | SPI base pointer   |
| <i>irqs</i> | SPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• <code>kSPI_RxReadyInterruptEnable</code></li> <li>• <code>kSPI_TxReadyInterruptEnable</code></li> </ul> |

### 18.3.6.12 `static bool SPI_IsMaster ( SPI_Type * base ) [inline], [static]`

## Parameters

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

## Returns

Returns true if the module is in master mode or false if the module is in slave mode.

**18.3.6.13** `status_t 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. |

**18.3.6.14** `static void SPI_WriteData ( SPI_Type * base, uint16_t data ) [inline], [static]`

## Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | SPI base pointer   |
| <i>data</i> | needs to be write. |

**18.3.6.15** `static void SPI_WriteConfigFlags ( SPI_Type * base, uint32_t configFlags ) [inline], [static]`

## Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | SPI base pointer                     |
| <i>configFlags</i> | control command needs to be written. |

**18.3.6.16** `void SPI_WriteDataWithConfigFlags ( SPI_Type * base, uint16_t data, uint32_t configFlags )`

## Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | SPI base pointer                     |
| <i>data</i>        | value needs to be written.           |
| <i>configFlags</i> | control command needs to be written. |

**18.3.6.17** `static uint32_t SPI_ReadData ( SPI_Type * base ) [inline], [static]`

## Parameters

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

## Returns

Data in the register.

### 18.3.6.18 void SPI\_SetTransferDelay ( SPI\_Type \* *base*, const spi\_delay\_config\_t \* *config* )

the delay uint is SPI clock time, maximum value is 0xF.

## Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI base pointer  |
| <i>config</i> | configuration for delay option <a href="#">spi_delay_config_t</a> . |

### 18.3.6.19 void SPI\_SetDummyData ( SPI\_Type \* *base*, uint16\_t *dummyData* )

This API can change the default data to be transferred when users set the tx buffer to NULL.

## Parameters

|                  |  |
|------------------|--|
| <i>base</i>      | SPI peripheral address.                        |
| <i>dummyData</i> | Data to be transferred when tx buffer is NULL. |

### 18.3.6.20 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.              |
| <i>kStatus_SPI_Timeout</i>     | The transfer timed out and was aborted. |

**18.3.6.21** `status_t 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.           |
| <i>userData</i> | User data.                   |

**18.3.6.22** `status_t SPI_MasterTransferNonBlocking ( SPI_Type * base, spi_master_handle_t * handle, spi_transfer_t * xfer )`

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. |

**18.3.6.23** `status_t SPI_MasterTransferGetCount ( SPI_Type * base, spi_master_handle_t * handle, size_t * count )`

This function gets the master transfer count.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI peripheral base address.   |
| <i>handle</i> | Pointer to the <code>spi_master_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | The number of bytes transferred by using the non-blocking transaction.                     |

Returns

status of `status_t`.

**18.3.6.24 void SPI\_MasterTransferAbort ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle* )**

This function aborts a transfer using an interrupt.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI peripheral base address.   |
| <i>handle</i> | Pointer to the <code>spi_master_handle_t</code> structure which stores the transfer state. |

**18.3.6.25 void SPI\_MasterTransferHandleIRQ ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle* )**

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI peripheral base address.   |
| <i>handle</i> | pointer to <code>spi_master_handle_t</code> structure which stores the transfer state. |

**18.3.6.26 status\_t 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.                   |

**18.3.6.27** `status_t SPI_SlaveTransferNonBlocking ( SPI_Type * base, spi_slave_handle_t * handle, spi_transfer_t * xfer )`

Note

The API returns immediately after the transfer initialization is finished.

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | SPI peripheral base address.  |
| <i>handle</i> | pointer to <code>spi_master_handle_t</code> structure which stores the transfer state |
| <i>xfer</i>   | pointer to <code>spi_xfer_config_t</code> 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. |

**18.3.6.28** `static status_t SPI_SlaveTransferGetCount ( SPI_Type * base, spi_slave_handle_t * handle, size_t * count ) [inline], [static]`

This function gets the slave transfer count.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI peripheral base address.   |
| <i>handle</i> | Pointer to the <code>spi_master_handle_t</code> structure which stores the transfer state. |
| <i>count</i>  | The number of bytes transferred by using the non-blocking transaction.                     |

Returns

status of `status_t`.

**18.3.6.29** `static void SPI_SlaveTransferAbort ( SPI_Type * base, spi_slave_handle_t * handle ) [inline], [static]`

This function aborts a transfer using an interrupt.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | SPI peripheral base address.   |
| <i>handle</i> | Pointer to the spi_slave_handle_t structure which stores the transfer state. |

**18.3.6.30 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 |

## Chapter 19

# USART: Universal Asynchronous Receiver/Transmitter Driver

### 19.1 Overview

The MCUXpresso SDK provides a peripheral USART driver for the Universal Synchronous Receiver/Transmitter (USART) module of MCUXpresso SDK devices. The driver does not support synchronous mode.

The USART driver includes two parts: functional APIs and transactional APIs.

Functional APIs are used for USART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the USART peripheral and know how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. USART functional operation groups provide the functional APIs set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `usart_handle_t` as the second parameter. Initialize the handle by calling the [USART\\_TransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer, which means that the functions [USART\\_TransferSendNonBlocking\(\)](#) and [USART\\_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_USART_TxIdle` and `kStatus_USART_RxIdle`.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the [USART\\_TransferCreateHandle\(\)](#). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The [USART\\_TransferReceiveNonBlocking\(\)](#) function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the `kStatus_USART_RxIdle`.

If the receive ring buffer is full, the upper layer is informed through a callback with the `kStatus_USART_RxRingBufferOverrun`. In the callback function, the upper layer reads data out from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code:

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/usart`. In this example, the buffer size is 32, but only 31 bytes are used for saving data.

## 19.2 Typical use case

### 19.2.1 USART Send/receive using a polling method

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

### 19.2.2 USART Send/receive using an interrupt method

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

### 19.2.3 USART Receive using the ringbuffer feature

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

### 19.2.4 USART Send/Receive using the DMA method

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

## Modules

- [USART Driver](#)

## 19.3 USART Driver

### 19.3.1 Overview

#### Data Structures

- struct `_usart_config`  
*USART configuration structure. [More...](#)*
- struct `_usart_transfer`  
*USART transfer structure. [More...](#)*
- struct `_usart_handle`  
*USART handle structure. [More...](#)*

#### Macros

- #define `FSL_SDK_ENABLE_USART_DRIVER_TRANSACTIONAL_APIS` 1  
*Macro gate for enable transaction API.*
- #define `FSL_SDK_USART_DRIVER_ENABLE_BAUDRATE_AUTO_GENERATE` 1  
*USART baud rate auto generate switch gate.*
- #define `UART_RETRY_TIMES` 0U  
*Retry times for waiting flag.*

#### Typedefs

- typedef enum `_usart_parity_mode` `usart_parity_mode_t`  
*USART parity mode.*
- typedef enum `_usart_sync_mode` `usart_sync_mode_t`  
*USART synchronous mode.*
- typedef enum `_usart_stop_bit_count` `usart_stop_bit_count_t`  
*USART stop bit count.*
- typedef enum `_usart_data_len` `usart_data_len_t`  
*USART data size.*
- typedef enum `_usart_clock_polarity` `usart_clock_polarity_t`  
*USART clock polarity configuration, used in sync mode.*
- typedef struct `_usart_config` `usart_config_t`  
*USART configuration structure.*
- typedef struct `_usart_transfer` `usart_transfer_t`  
*USART transfer structure.*
- typedef void(\* `usart_transfer_callback_t` )(USART\_Type \*base, `usart_handle_t` \*handle, `status_t` status, void \*userData)  
*USART transfer callback function.*

## Enumerations

- enum {
  - kStatus\_USART\_TxBusy = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 0),
  - kStatus\_USART\_RxBusy = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 1),
  - kStatus\_USART\_TxIdle = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 2),
  - kStatus\_USART\_RxIdle = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 3),
  - kStatus\_USART\_TxError = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 4),
  - kStatus\_USART\_RxError = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 5),
  - kStatus\_USART\_RxRingBufferOverrun = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 6),
  - kStatus\_USART\_NoiseError = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 7),
  - kStatus\_USART\_FramingError = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 8),
  - kStatus\_USART\_ParityError = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 9),
  - kStatus\_USART\_HardwareOverrun = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 10),
  - kStatus\_USART\_BaudrateNotSupport,
  - kStatus\_USART\_Timeout = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 12) }

*Error codes for the USART driver.*
- enum \_usart\_parity\_mode {
  - kUSART\_ParityDisabled = 0x0U,
  - kUSART\_ParityEven = 0x2U,
  - kUSART\_ParityOdd = 0x3U }

*USART parity mode.*
- enum \_usart\_sync\_mode {
  - kUSART\_SyncModeDisabled = 0x0U,
  - kUSART\_SyncModeSlave = 0x2U,
  - kUSART\_SyncModeMaster = 0x3U }

*USART synchronous mode.*
- enum \_usart\_stop\_bit\_count {
  - kUSART\_OneStopBit = 0U,
  - kUSART\_TwoStopBit = 1U }

*USART stop bit count.*
- enum \_usart\_data\_len {
  - kUSART\_7BitsPerChar = 0U,
  - kUSART\_8BitsPerChar = 1U }

*USART data size.*
- enum \_usart\_clock\_polarity {
  - kUSART\_RxSampleOnFallingEdge = 0x0U,
  - kUSART\_RxSampleOnRisingEdge = 0x1U }

*USART clock polarity configuration, used in sync mode.*
- enum \_usart\_interrupt\_enable {

```

kUSART_RxReadyInterruptEnable = (USART_INTENSET_RXRDYEN_MASK),
kUSART_TxReadyInterruptEnable = (USART_INTENSET_TXRDYEN_MASK),
kUSART_TxIdleInterruptEnable = (USART_INTENSET_TXIDLEEN_MASK),
kUSART_DeltaCtsInterruptEnable = (USART_INTENSET_DELTACTSEN_MASK),
kUSART_TxDisableInterruptEnable = (USART_INTENSET_TXDISEN_MASK),
kUSART_HardwareOverRunInterruptEnable = (USART_INTENSET_OVERRUNEN_MASK),
kUSART_RxBreakInterruptEnable = (USART_INTENSET_DELTARXBRKEN_MASK),
kUSART_RxStartInterruptEnable = (USART_INTENSET_STARTEN_MASK),
kUSART_FramErrorInterruptEnable = (USART_INTENSET_FRAMERREN_MASK),
kUSART_ParityErrorInterruptEnable = (USART_INTENSET_PARITYERREN_MASK),
kUSART_RxNoiseInterruptEnable = (USART_INTENSET_RXNOISEEN_MASK),
kUSART_AutoBaudErrorInterruptEnable = (USART_INTENSET_ABERREN_MASK),
kUSART_AllInterruptEnable }

```

*USART interrupt configuration structure, default settings all disabled.*

- enum `_usart_flags` {

```

kUSART_RxReady = (USART_STAT_RXRDY_MASK),
kUSART_RxIdleFlag = (USART_STAT_RXIDLE_MASK),
kUSART_TxReady = (USART_STAT_TXRDY_MASK),
kUSART_TxIdleFlag = (USART_STAT_TXIDLE_MASK),
kUSART_CtsState = (USART_STAT_CTS_MASK),
kUSART_DeltaCtsFlag = (USART_STAT_DELTACTS_MASK),
kUSART_TxDisableFlag = (USART_STAT_TXDISSTAT_MASK),
kUSART_HardwareOverrunFlag = (USART_STAT_OVERRUNINT_MASK),
kUSART_RxBreakFlag = (USART_STAT_DELTARXBRK_MASK),
kUSART_RxStartFlag = (USART_STAT_START_MASK),
kUSART_FramErrorFlag = (USART_STAT_FRAMERRINT_MASK),
kUSART_ParityErrorFlag = (USART_STAT_PARITYERRINT_MASK),
kUSART_RxNoiseFlag = (USART_STAT_RXNOISEINT_MASK),
kUSART_AutoBaudErrorFlag = (USART_STAT_ABERR_MASK) }

```

*USART status flags.*

## Driver version

- #define `FSL_USART_DRIVER_VERSION` (`MAKE_VERSION(2, 5, 1)`)  
*USART driver version.*

## Get the instance of USART

- uint32\_t `USART_GetInstance` (USART\_Type \*base)  
*Returns instance number for USART peripheral base address.*

## Initialization and deinitialization

- `status_t USART_Init` (USART\_Type \*base, const `usart_config_t` \*config, uint32\_t srcClock\_Hz)  
*Initializes a USART instance with user configuration structure and peripheral clock.*
- `void USART_Deinit` (USART\_Type \*base)  
*Deinitializes a USART instance.*
- `void USART_GetDefaultConfig` (`usart_config_t` \*config)  
*Gets the default configuration structure.*
- `status_t USART_SetBaudRate` (USART\_Type \*base, uint32\_t baudrate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the USART instance baud rate.*

## Status

- `static uint32_t USART_GetStatusFlags` (USART\_Type \*base)  
*Get USART status flags.*
- `static void USART_ClearStatusFlags` (USART\_Type \*base, uint32\_t mask)  
*Clear USART status flags.*

## Interrupts

- `static void USART_EnableInterrupts` (USART\_Type \*base, uint32\_t mask)  
*Enables USART interrupts according to the provided mask.*
- `static void USART_DisableInterrupts` (USART\_Type \*base, uint32\_t mask)  
*Disables USART interrupts according to a provided mask.*
- `static uint32_t USART_GetEnabledInterrupts` (USART\_Type \*base)  
*Returns enabled USART interrupts.*

## Bus Operations

- `static void USART_EnableContinuousSCLK` (USART\_Type \*base, bool enable)  
*Continuous Clock generation.*
- `static void USART_EnableAutoClearSCLK` (USART\_Type \*base, bool enable)  
*Enable Continuous Clock generation bit auto clear.*
- `static void USART_EnableCTS` (USART\_Type \*base, bool enable)  
*Enable CTS.*
- `static void USART_EnableTx` (USART\_Type \*base, bool enable)  
*Enable the USART transmit.*
- `static void USART_EnableRx` (USART\_Type \*base, bool enable)  
*Enable the USART receive.*
- `static void USART_WriteByte` (USART\_Type \*base, uint8\_t data)  
*Writes to the TXDAT register.*
- `static uint8_t USART_ReadByte` (USART\_Type \*base)  
*Reads the RXDAT directly.*
- `status_t USART_WriteBlocking` (USART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*
- `status_t USART_ReadBlocking` (USART\_Type \*base, uint8\_t \*data, size\_t length)

Read RX data register using a blocking method.

## Transactional

- [status\\_t USART\\_TransferCreateHandle](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle, [usart\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the USART handle.*
- [status\\_t USART\\_TransferSendNonBlocking](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle, [usart\\_transfer\\_t](#) \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- [void USART\\_TransferStartRingBuffer](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- [void USART\\_TransferStopRingBuffer](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- [size\\_t USART\\_TransferGetRxRingBufferLength](#) ([usart\\_handle\\_t](#) \*handle)  
*Get the length of received data in RX ring buffer.*
- [void USART\\_TransferAbortSend](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle)  
*Aborts the interrupt-driven data transmit.*
- [status\\_t USART\\_TransferGetSendCount](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle, uint32\_t \*count)  
*Get the number of bytes that have been written to USART TX register.*
- [status\\_t USART\\_TransferReceiveNonBlocking](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle, [usart\\_transfer\\_t](#) \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using an interrupt method.*
- [void USART\\_TransferAbortReceive](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle)  
*Aborts the interrupt-driven data receiving.*
- [status\\_t USART\\_TransferGetReceiveCount](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*
- [void USART\\_TransferHandleIRQ](#) (USART\_Type \*base, [usart\\_handle\\_t](#) \*handle)  
*USART IRQ handle function.*

## 19.3.2 Data Structure Documentation

### 19.3.2.1 struct \_usart\_config

#### Data Fields

- uint32\_t [baudRate\\_Bps](#)  
*USART baud rate.*
- bool [enableRx](#)  
*USART receive enable.*
- bool [enableTx](#)  
*USART transmit enable.*
- bool [loopback](#)  
*Enable peripheral loopback.*

- bool `enableContinuousSCLK`  
*USART continuous Clock generation enable in synchronous master mode.*
- bool `enableHardwareFlowControl`  
*Enable hardware control RTS/CTS.*
- `usart_parity_mode_t` `parityMode`  
*Parity mode, disabled (default), even, odd.*
- `usart_stop_bit_count_t` `stopBitCount`  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- `usart_data_len_t` `bitCountPerChar`  
*Data length - 7 bit, 8 bit.*
- `usart_sync_mode_t` `syncMode`  
*Transfer mode - asynchronous, synchronous master, synchronous slave.*
- `usart_clock_polarity_t` `clockPolarity`  
*Selects the clock polarity and sampling edge in sync mode.*

### Field Documentation

- (1) `bool _usart_config::enableRx`
- (2) `bool _usart_config::enableTx`
- (3) `bool _usart_config::enableContinuousSCLK`
- (4) `usart_sync_mode_t _usart_config::syncMode`
- (5) `usart_clock_polarity_t _usart_config::clockPolarity`

### 19.3.2.2 struct \_usart\_transfer

#### Data Fields

- `size_t` `dataSize`  
*The byte count to be transfer.*
- `uint8_t *` `data`  
*The buffer of data to be transfer.*
- `uint8_t *` `rxData`  
*The buffer to receive data.*
- `const uint8_t *` `txData`  
*The buffer of data to be sent.*

## Field Documentation

- (1) `uint8_t* _usart_transfer::data`
- (2) `uint8_t* _usart_transfer::rxData`
- (3) `const uint8_t* _usart_transfer::txData`
- (4) `size_t _usart_transfer::dataSize`

### 19.3.2.3 struct \_usart\_handle

#### Data Fields

- `const uint8_t *volatile txData`  
*Address of remaining data to send.*
- `volatile size_t txDataSize`  
*Size of the remaining data to send.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `uint8_t *volatile rxData`  
*Address of remaining data to receive.*
- `volatile size_t rxDataSize`  
*Size of the remaining data to receive.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `uint8_t * rxRingBuffer`  
*Start address of the receiver ring buffer.*
- `size_t rxRingBufferSize`  
*Size of the ring buffer.*
- `volatile uint16_t rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- `volatile uint16_t rxRingBufferTail`  
*Index for the user to get data from the ring buffer.*
- `usart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*USART callback function parameter.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*

**Field Documentation**

- (1) `const uint8_t* volatile _usart_handle::txData`
- (2) `volatile size_t _usart_handle::txDataSize`
- (3) `size_t _usart_handle::txDataSizeAll`
- (4) `uint8_t* volatile _usart_handle::rxData`
- (5) `volatile size_t _usart_handle::rxDataSize`
- (6) `size_t _usart_handle::rxDataSizeAll`
- (7) `uint8_t* _usart_handle::rxRingBuffer`
- (8) `size_t _usart_handle::rxRingBufferSize`
- (9) `volatile uint16_t _usart_handle::rxRingBufferHead`
- (10) `volatile uint16_t _usart_handle::rxRingBufferTail`
- (11) `usart_transfer_callback_t _usart_handle::callback`
- (12) `void* _usart_handle::userData`
- (13) `volatile uint8_t _usart_handle::txState`

**19.3.3 Macro Definition Documentation**

**19.3.3.1** `#define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))`

**19.3.3.2** `#define FSL_SDK_ENABLE_USART_DRIVER_TRANSACTIONAL_APIS 1`

1 for enable, 0 for disable.

**19.3.3.3** `#define FSL_SDK_USART_DRIVER_ENABLE_BAUDRATE_AUTO_GENERATE 1`

1 for enable, 0 for disable

**19.3.3.4** `#define UART_RETRY_TIMES 0U`

Defining to zero means to keep waiting for the flag until it is assert/deassert.

### 19.3.4 Typedef Documentation

19.3.4.1 typedef enum `_usart_parity_mode` `usart_parity_mode_t`

19.3.4.2 typedef enum `_usart_sync_mode` `usart_sync_mode_t`

19.3.4.3 typedef enum `_usart_stop_bit_count` `usart_stop_bit_count_t`

19.3.4.4 typedef enum `_usart_data_len` `usart_data_len_t`

19.3.4.5 typedef enum `_usart_clock_polarity` `usart_clock_polarity_t`

19.3.4.6 typedef struct `_usart_config` `usart_config_t`

19.3.4.7 typedef struct `_usart_transfer` `usart_transfer_t`

19.3.4.8 typedef void(\* `usart_transfer_callback_t`)(`USART_Type *base`, `usart_handle_t *handle`, `status_t status`, void \*`userData`)

### 19.3.5 Enumeration Type Documentation

#### 19.3.5.1 anonymous enum

Enumerator

*kStatus\_USART\_TxBusy* Transmitter is busy.  
*kStatus\_USART\_RxBusy* Receiver is busy.  
*kStatus\_USART\_TxIdle* USART transmitter is idle.  
*kStatus\_USART\_RxIdle* USART receiver is idle.  
*kStatus\_USART\_TxError* Error happens on tx.  
*kStatus\_USART\_RxError* Error happens on rx.  
*kStatus\_USART\_RxRingBufferOverrun* Error happens on rx ring buffer.  
*kStatus\_USART\_NoiseError* USART noise error.  
*kStatus\_USART\_FramingError* USART framing error.  
*kStatus\_USART\_ParityError* USART parity error.  
*kStatus\_USART\_HardwareOverrun* USART hardware over flow.  
*kStatus\_USART\_BaudrateNotSupport* Baudrate is not support in current clock source.  
*kStatus\_USART\_Timeout* USART times out.

#### 19.3.5.2 enum `_usart_parity_mode`

Enumerator

*kUSART\_ParityDisabled* Parity disabled.

*kUSART\_ParityEven* Parity enabled, type even, bit setting: PARITYSEL = 10.

*kUSART\_ParityOdd* Parity enabled, type odd, bit setting: PARITYSEL = 11.

### 19.3.5.3 enum \_usart\_sync\_mode

Enumerator

*kUSART\_SyncModeDisabled* Asynchronous mode.

*kUSART\_SyncModeSlave* Synchronous slave mode.

*kUSART\_SyncModeMaster* Synchronous master mode.

### 19.3.5.4 enum \_usart\_stop\_bit\_count

Enumerator

*kUSART\_OneStopBit* One stop bit.

*kUSART\_TwoStopBit* Two stop bits.

### 19.3.5.5 enum \_usart\_data\_len

Enumerator

*kUSART\_7BitsPerChar* Seven bit mode.

*kUSART\_8BitsPerChar* Eight bit mode.

### 19.3.5.6 enum \_usart\_clock\_polarity

Enumerator

*kUSART\_RxSampleOnFallingEdge* Un\_RXD is sampled on the falling edge of SCLK.

*kUSART\_RxSampleOnRisingEdge* Un\_RXD is sampled on the rising edge of SCLK.

### 19.3.5.7 enum \_usart\_interrupt\_enable

Enumerator

*kUSART\_RxReadyInterruptEnable* Receive ready interrupt.

*kUSART\_TxReadyInterruptEnable* Transmit ready interrupt.

*kUSART\_TxIdleInterruptEnable* Transmit idle interrupt.

*kUSART\_DeltaCtsInterruptEnable* Cts pin change interrupt.

*kUSART\_TxDisableInterruptEnable* Transmit disable interrupt.

*kUSART\_HardwareOverRunInterruptEnable* hardware over run interrupt.  
*kUSART\_RxBreakInterruptEnable* Receive break interrupt.  
*kUSART\_RxStartInterruptEnable* Receive ready interrupt.  
*kUSART\_FramErrorInterruptEnable* Receive start interrupt.  
*kUSART\_ParityErrorInterruptEnable* Receive frame error interrupt.  
*kUSART\_RxNoiseInterruptEnable* Receive noise error interrupt.  
*kUSART\_AutoBaudErrorInterruptEnable* Receive auto baud error interrupt.  
*kUSART\_AllInterruptEnable* All interrupt.

### 19.3.5.8 enum \_usart\_flags

This provides constants for the USART status flags for use in the USART functions.

Enumerator

*kUSART\_RxReady* Receive ready flag.  
*kUSART\_RxIdleFlag* Receive IDLE flag.  
*kUSART\_TxReady* Transmit ready flag.  
*kUSART\_TxIdleFlag* Transmit idle flag.  
*kUSART\_CtsState* Cts pin status.  
*kUSART\_DeltaCtsFlag* Cts pin change flag.  
*kUSART\_TxDisableFlag* Transmit disable flag.  
*kUSART\_HardwareOverrunFlag* Hardware over run flag.  
*kUSART\_RxBreakFlag* Receive break flag.  
*kUSART\_RxStartFlag* receive start flag.  
*kUSART\_FramErrorFlag* Frame error flag.  
*kUSART\_ParityErrorFlag* Parity error flag.  
*kUSART\_RxNoiseFlag* Receive noise flag.  
*kUSART\_AutoBaudErrorFlag* Auto baud error flag.

## 19.3.6 Function Documentation

19.3.6.1 `uint32_t USART_GetInstance ( USART_Type * base )`

19.3.6.2 `status_t USART_Init ( USART_Type * base, const usart_config_t * config, uint32_t srcClock_Hz )`

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the [USART\\_GetDefaultConfig\(\)](#) function. Example below shows how to use this API to configure USART.

```

* usart_config_t usartConfig;
* usartConfig.baudRate_Bps = 115200U;
* usartConfig.parityMode = kUSART_ParityDisabled;

```

```
* usartConfig.stopBitCount = kUSART_OneStopBit;
* USART_Init(USART1, &usartConfig, 20000000U);
*
```

## Parameters

|                    |  |
|--------------------|--|
| <i>base</i>        | USART peripheral base address.                   |
| <i>config</i>      | Pointer to user-defined configuration structure. |
| <i>srcClock_Hz</i> | USART clock source frequency in HZ.              |

## Return values

|  |  |
|--|--|
| <i>kStatus_USART_-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_InvalidArgument</i>           | USART base address is not valid                  |
| <i>kStatus_Success</i>                   | Status USART initialize succeed                  |

### 19.3.6.3 void USART\_Deinit ( USART\_Type \* *base* )

This function waits for TX complete, disables the USART clock.

## Parameters

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

### 19.3.6.4 void USART\_GetDefaultConfig ( usart\_config\_t \* *config* )

This function initializes the USART configuration structure to a default value. The default values are:  
 : usartConfig->baudRate\_Bps = 9600U; usartConfig->parityMode = kUSART\_ParityDisabled; usartConfig->stopBitCount = kUSART\_OneStopBit; usartConfig->bitCountPerChar = kUSART\_8BitsPerChar; usartConfig->loopback = false; usartConfig->enableTx = false; usartConfig->enableRx = false;  
 ...

## Parameters

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

### 19.3.6.5 status\_t USART\_SetBaudRate ( USART\_Type \* *base*, uint32\_t *baudrate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the USART\_Init.

```
* USART_SetBaudRate(USART1, 115200U, 20000000U);
*
```

Parameters

|                     |                                     |
|---------------------|-------------------------------------|
| <i>base</i>         | USART peripheral base address.      |
| <i>baudrate_Bps</i> | USART baudrate to be set.           |
| <i>srcClock_Hz</i>  | USART clock source frequency in HZ. |

Return values

|  |  |
|--|--|
| <i>kStatus_USART_-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                   | Set baudrate succeed.                            |
| <i>kStatus_InvalidArgument</i>           | One or more arguments are invalid.               |

**19.3.6.6 static uint32\_t USART\_GetStatusFlags ( USART\_Type \* *base* ) [inline], [static]**

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators `_usart_flags`. To check a specific status, compare the return value with enumerators in `_usart_flags`. For example, to check whether the RX is ready:

```
* if (kUSART_RxReady & USART_GetStatusFlags(USART1))
* {
*     ...
* }
*
```

Parameters

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

Returns

USART status flags which are ORed by the enumerators in the `_usart_flags`.

**19.3.6.7 static void USART\_ClearStatusFlags ( USART\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

This function clear supported USART status flags For example:

```
* USART_ClearStatusFlags(USART1,
* kUSART_HardwareOverrunFlag)
*
```

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
| <i>mask</i> | status flags to be cleared.    |

**19.3.6.8 static void USART\_EnableInterrupts ( USART\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_usart\\_interrupt\\_enable](#). For example, to enable TX ready interrupt and RX ready interrupt:

```
* USART_EnableInterrupts(USART1,
*   kUSART_RxReadyInterruptEnable |
*   kUSART_TxReadyInterruptEnable);
*
```

Parameters

|             |   |
|-------------|---|
| <i>base</i> | USART peripheral base address.  |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_usart_interrupt_enable</a> . |

**19.3.6.9 static void USART\_DisableInterrupts ( USART\_Type \* *base*, uint32\_t *mask* )  
[inline], [static]**

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [\\_usart\\_interrupt\\_enable](#). This example shows how to disable the TX ready interrupt and RX ready interrupt:

```
* USART_DisableInterrupts(USART1,
*   kUSART_TxReadyInterruptEnable |
*   kUSART_RxReadyInterruptEnable);
*
```

Parameters

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

|             |  |
|-------------|--|
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_usart_interrupt_enable</a> . |
|-------------|--|

**19.3.6.10 static uint32\_t USART\_GetEnabledInterrupts ( USART\_Type \* *base* )  
[inline], [static]**

This function returns the enabled USART interrupts.

Parameters

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

**19.3.6.11 static void USART\_EnableContinuousSCLK ( USART\_Type \* *base*, bool *enable* )  
[inline], [static]**

By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this function, SCLK will run continuously in synchronous mode, allowing characters to be received on Un\_RxD independently from transmission on Un\_TXD).

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | USART peripheral base address.   |
| <i>enable</i> | Enable Continuous Clock generation mode or not, true for enable and false for disable. |

**19.3.6.12 static void USART\_EnableAutoClearSCLK ( USART\_Type \* *base*, bool *enable* )  
[inline], [static]**

While enable this function, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | USART peripheral base address.                                   |
| <i>enable</i> | Enable auto clear or not, true for enable and false for disable. |

**19.3.6.13 static void USART\_EnableCTS ( USART\_Type \* *base*, bool *enable* )  
[inline], [static]**

This function will determine whether CTS is used for flow control.

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | USART peripheral base address.                            |
| <i>enable</i> | Enable CTS or not, true for enable and false for disable. |

**19.3.6.14 static void USART\_EnableTx ( USART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function will enable or disable the USART transmit.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | USART peripheral base address.         |
| <i>enable</i> | true for enable and false for disable. |

**19.3.6.15 static void USART\_EnableRx ( USART\_Type \* *base*, bool *enable* ) [inline], [static]**

This function will enable or disable the USART receive. Note: if the transmit is enabled, the receive will not be disabled.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | USART peripheral base address.         |
| <i>enable</i> | true for enable and false for disable. |

**19.3.6.16 static void USART\_WriteByte ( USART\_Type \* *base*, uint8\_t *data* ) [inline], [static]**

This function will writes data to the TXDAT automatly.The upper layer must ensure that TXDATA has space for data to write before calling this function.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
| <i>data</i> | The byte to write.             |

**19.3.6.17** `static uint8_t USART_ReadByte ( USART_Type * base ) [inline],`  
`[static]`

This function reads data from the RXDAT automatly. The upper layer must ensure that the RXDAT is not empty before calling this function.

Parameters

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

Returns

The byte read from USART data register.

**19.3.6.18 status\_t USART\_WriteBlocking ( USART\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )**

This function polls the TX register, waits for the TX register to be empty.

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | USART peripheral base address.      |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

Return values

|                              |   |
|------------------------------|---|
| <i>kStatus_USART_Timeout</i> | Transmission timed out and was aborted. |
| <i>kStatus_Success</i>       | Successfully wrote all data.            |

**19.3.6.19 status\_t USART\_ReadBlocking ( USART\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )**

This function polls the RX register, waits for the RX register to be full.

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | USART peripheral base address.                          |
| <i>data</i>   | Start address of the buffer to store the received data. |
| <i>length</i> | Size of the buffer.                                     |

Return values

|                                    |   |
|------------------------------------|---|
| <i>kStatus_USART_-FramingError</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_USART_Parity-Error</i>  | Noise error happened while receiving data.      |
| <i>kStatus_USART_Noise-Error</i>   | Framing error happened while receiving data.    |
| <i>kStatus_USART_RxError</i>       | Overflow or underflow happened.                 |
| <i>kStatus_USART_Timeout</i>       | Transmission timed out and was aborted.         |
| <i>kStatus_Success</i>             | Successfully received all data.                 |

**19.3.6.20 status\_t USART\_TransferCreateHandle ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the USART handle which can be used for other USART transactional APIs. Usually, for a specified USART instance, call this API once to get the initialized handle.

Parameters

|                 |   |
|-----------------|---|
| <i>base</i>     | USART peripheral base address.          |
| <i>handle</i>   | USART handle pointer.                   |
| <i>callback</i> | The callback function.                  |
| <i>userData</i> | The parameter of the callback function. |

**19.3.6.21 status\_t USART\_TransferSendNonBlocking ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer* )**

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the IRQ handler, the USART driver calls the callback function and passes the [kStatus\\_USART\\_TxIdle](#) as status parameter.

Note

The [kStatus\\_USART\\_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 [kUSART-TransmissionCompleteFlag](#) to ensure that the TX is finished.

Parameters

|               |  |
|---------------|--|
| <i>base</i>   | USART peripheral base address.                                   |
| <i>handle</i> | USART handle pointer.  |
| <i>xfer</i>   | USART transfer structure. See <a href="#">usart_transfer_t</a> . |

Return values

|                                |  |
|--------------------------------|--|
| <i>kStatus_Success</i>         | Successfully start the data transmission.  |
| <i>kStatus_USART_TxBusy</i>    | Previous transmission still not finished, data not all written to TX register yet. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.  |

**19.3.6.22 void USART\_TransferStartRingBuffer ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )**

This function sets up the RX ring buffer to a specific USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the [USART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if *ringBufferSize* is 32, then only 31 bytes are used for saving data.

Parameters

|                       |  |
|-----------------------|--|
| <i>base</i>           | USART peripheral base address.   |
| <i>handle</i>         | USART handle pointer.  |
| <i>ringBuffer</i>     | Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.   |

**19.3.6.23 void USART\_TransferStopRingBuffer ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )**

This function aborts the background transfer and uninstalls the ring buffer.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

#### 19.3.6.24 **size\_t USART\_TransferGetRxRingBufferLength ( usart\_handle\_t \* *handle* )**

Parameters

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

Returns

Length of received data in RX ring buffer.

#### 19.3.6.25 **void USART\_TransferAbortSend ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )**

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are still not sent out.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

#### 19.3.6.26 **status\_t USART\_TransferGetSendCount ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been written to USART TX register by interrupt method.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

|              |                   |
|--------------|-------------------|
| <i>count</i> | Send bytes count. |
|--------------|-------------------|

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

**19.3.6.27 status\_t USART\_TransferReceiveNonBlocking ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )**

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter [kStatus\\_USART\\_RxIdle](#). For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the *xfer->data* and this function returns with the parameter *receivedBytes* set to 5. For the left 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the USART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the *xfer->data*. When all data is received, the upper layer is notified.

Parameters

|                      |  |
|----------------------|--|
| <i>base</i>          | USART peripheral base address.                                   |
| <i>handle</i>        | USART handle pointer.  |
| <i>xfer</i>          | USART transfer structure, see <a href="#">usart_transfer_t</a> . |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.                    |

Return values

|                             |  |
|-----------------------------|--|
| <i>kStatus_Success</i>      | Successfully queue the transfer into transmit queue. |
| <i>kStatus_USART_RxBusy</i> | Previous receive request is not finished.            |

|                                |                   |
|--------------------------------|-------------------|
| <i>kStatus_InvalidArgument</i> | Invalid argument. |
|--------------------------------|-------------------|

**19.3.6.28 void USART\_TransferAbortReceive ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )**

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

**19.3.6.29 status\_t USART\_TransferGetReceiveCount ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been received.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |
| <i>count</i>  | Receive bytes count.           |

Return values

|                                     |   |
|-------------------------------------|---|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                               |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

**19.3.6.30 void USART\_TransferHandleIRQ ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )**

This function handles the USART transmit and receive IRQ request.

## Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

# Chapter 20

## MRT: Multi-Rate Timer

### 20.1 Overview

The MCUXpresso SDK provides a driver for the Multi-Rate Timer (MRT) of MCUXpresso SDK devices.

### 20.2 Function groups

The MRT driver supports operating the module as a time counter.

#### 20.2.1 Initialization and deinitialization

The function [MRT\\_Init\(\)](#) initializes the MRT with specified configurations. The function [MRT\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the MRT operating mode.

The function [MRT\\_Deinit\(\)](#) stops the MRT timers and disables the module clock.

#### 20.2.2 Timer period Operations

The function [MRT\\_UpdateTimerPeriod\(\)](#) is used to update the timer period in units of count. The new value is immediately loaded or will be loaded at the end of the current time interval.

The function [MRT\\_GetCurrentTimerCount\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

The timer period operation functions takes the count value in ticks. The user can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds

#### 20.2.3 Start and Stop timer operations

The function [MRT\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value, counts down to 0 and depending on the timer mode it either loads the respective start value again or stop. When the timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

The function [MRT\\_StopTimer\(\)](#) stops the timer counting.

## 20.2.4 Get and release channel

These functions can be used to reserve and release a channel. The function [MRT\\_GetIdleChannel\(\)](#) finds the available channel. This function returns the lowest available channel number. The function [MRT\\_ReleaseChannel\(\)](#) release the channel when the timer is using the multi-task mode. In multi-task mode, the INUSE flags allow more control over when MRT channels are released for further use.

## 20.2.5 Status

Provides functions to get and clear the PIT status.

## 20.2.6 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

## 20.3 Typical use case

### 20.3.1 MRT tick example

Updates the MRT period and toggles an LED periodically. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/mrt`

## Files

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

## Data Structures

- struct [\\_mrt\\_config](#)  
*MRT configuration structure. [More...](#)*

## Typedefs

- typedef enum [\\_mrt\\_chnl](#) [mrt\\_chnl\\_t](#)  
*List of MRT channels.*
- typedef enum [\\_mrt\\_timer\\_mode](#) [mrt\\_timer\\_mode\\_t](#)  
*List of MRT timer modes.*
- typedef enum [\\_mrt\\_interrupt\\_enable](#) [mrt\\_interrupt\\_enable\\_t](#)  
*List of MRT interrupts.*
- typedef enum [\\_mrt\\_status\\_flags](#) [mrt\\_status\\_flags\\_t](#)  
*List of MRT status flags.*
- typedef struct [\\_mrt\\_config](#) [mrt\\_config\\_t](#)  
*MRT configuration structure.*

## Enumerations

- enum `_mrt_chnl` {  
`kMRT_Channel_0` = 0U,  
`kMRT_Channel_1`,  
`kMRT_Channel_2`,  
`kMRT_Channel_3` }  
*List of MRT channels.*
- enum `_mrt_timer_mode` {  
`kMRT_RepeatMode` = (0 << MRT\_CHANNEL\_CTRL\_MODE\_SHIFT),  
`kMRT_OneShotMode` = (1 << MRT\_CHANNEL\_CTRL\_MODE\_SHIFT),  
`kMRT_OneShotStallMode` = (2 << MRT\_CHANNEL\_CTRL\_MODE\_SHIFT) }  
*List of MRT timer modes.*
- enum `_mrt_interrupt_enable` { `kMRT_TimerInterruptEnable` = MRT\_CHANNEL\_CTRL\_INTEN-  
`_MASK` }  
*List of MRT interrupts.*
- enum `_mrt_status_flags` {  
`kMRT_TimerInterruptFlag` = MRT\_CHANNEL\_STAT\_INTFLAG\_MASK,  
`kMRT_TimerRunFlag` = MRT\_CHANNEL\_STAT\_RUN\_MASK }  
*List of MRT status flags.*

## Driver version

- #define `FSL_MRT_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 3)`)  
*Version 2.0.3.*

## Initialization and deinitialization

- void `MRT_Init` (`MRT_Type *base`, const `mrt_config_t *config`)  
*Un-gates the MRT clock and configures the peripheral for basic operation.*
- void `MRT_Deinit` (`MRT_Type *base`)  
*Gate the MRT clock.*
- static void `MRT_GetDefaultConfig` (`mrt_config_t *config`)  
*Fill in the MRT config struct with the default settings.*
- static void `MRT_SetupChannelMode` (`MRT_Type *base`, `mrt_chnl_t channel`, const `mrt_timer_-  
mode_t mode`)  
*Sets up an MRT channel mode.*

## Interrupt Interface

- static void `MRT_EnableInterrupts` (`MRT_Type *base`, `mrt_chnl_t channel`, `uint32_t mask`)  
*Enables the MRT interrupt.*
- static void `MRT_DisableInterrupts` (`MRT_Type *base`, `mrt_chnl_t channel`, `uint32_t mask`)  
*Disables the selected MRT interrupt.*
- static `uint32_t MRT_GetEnabledInterrupts` (`MRT_Type *base`, `mrt_chnl_t channel`)  
*Gets the enabled MRT interrupts.*

## Status Interface

- static `uint32_t MRT_GetStatusFlags` (`MRT_Type *base`, `mrt_chnl_t channel`)

*Gets the MRT status flags.*

- static void [MRT\\_ClearStatusFlags](#) (MRT\_Type \*base, [mrt\\_chnl\\_t](#) channel, uint32\_t mask)  
*Clears the MRT status flags.*

## Read and Write the timer period

- void [MRT\\_UpdateTimerPeriod](#) (MRT\_Type \*base, [mrt\\_chnl\\_t](#) channel, uint32\_t count, bool immediateLoad)  
*Used to update the timer period in units of count.*
- static uint32\_t [MRT\\_GetCurrentTimerCount](#) (MRT\_Type \*base, [mrt\\_chnl\\_t](#) channel)  
*Reads the current timer counting value.*

## Timer Start and Stop

- static void [MRT\\_StartTimer](#) (MRT\_Type \*base, [mrt\\_chnl\\_t](#) channel, uint32\_t count)  
*Starts the timer counting.*
- static void [MRT\\_StopTimer](#) (MRT\_Type \*base, [mrt\\_chnl\\_t](#) channel)  
*Stops the timer counting.*

## Get & release channel

- static uint32\_t [MRT\\_GetIdleChannel](#) (MRT\_Type \*base)  
*Find the available channel.*

## 20.4 Data Structure Documentation

### 20.4.1 struct \_mrt\_config

This structure holds the configuration settings for the MRT peripheral. To initialize this structure to reasonable defaults, call the [MRT\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

### Data Fields

- bool [enableMultiTask](#)  
*true: Timers run in multi-task mode; false: Timers run in hardware status mode*

## 20.5 Typedef Documentation

### 20.5.1 typedef struct \_mrt\_config mrt\_config\_t

This structure holds the configuration settings for the MRT peripheral. To initialize this structure to reasonable defaults, call the [MRT\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

## 20.6 Enumeration Type Documentation

### 20.6.1 enum \_mrt\_chnl

Enumerator

- kMRT\_Channel\_0* MRT channel number 0.
- kMRT\_Channel\_1* MRT channel number 1.
- kMRT\_Channel\_2* MRT channel number 2.
- kMRT\_Channel\_3* MRT channel number 3.

### 20.6.2 enum \_mrt\_timer\_mode

Enumerator

- kMRT\_RepeatMode* Repeat Interrupt mode.
- kMRT\_OneShotMode* One-shot Interrupt mode.
- kMRT\_OneShotStallMode* One-shot stall mode.

### 20.6.3 enum \_mrt\_interrupt\_enable

Enumerator

- kMRT\_TimerInterruptEnable* Timer interrupt enable.

### 20.6.4 enum \_mrt\_status\_flags

Enumerator

- kMRT\_TimerInterruptFlag* Timer interrupt flag.
- kMRT\_TimerRunFlag* Indicates state of the timer.

## 20.7 Function Documentation

### 20.7.1 void MRT\_Init ( MRT\_Type \* *base*, const mrt\_config\_t \* *config* )

Note

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

## Parameters

|               |  |
|---------------|--|
| <i>base</i>   | Multi-Rate timer peripheral base address   |
| <i>config</i> | Pointer to user's MRT config structure. If MRT has MULTITASK bit field in MOD-CFG register, param config is useless. |

**20.7.2 void MRT\_Deinit ( MRT\_Type \* *base* )**

## Parameters

|             |  |
|-------------|--|
| <i>base</i> | Multi-Rate timer peripheral base address |
|-------------|--|

**20.7.3 static void MRT\_GetDefaultConfig ( mrt\_config\_t \* *config* ) [inline], [static]**

The default values are:

```
*   config->enableMultiTask = false;
*
```

## Parameters

|               |   |
|---------------|---|
| <i>config</i> | Pointer to user's MRT config structure. |
|---------------|---|

**20.7.4 static void MRT\_SetupChannelMode ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, const mrt\_timer\_mode\_t *mode* ) [inline], [static]**

## Parameters

|                |  |
|----------------|--|
| <i>base</i>    | Multi-Rate timer peripheral base address |
| <i>channel</i> | Channel that is being configured.        |
| <i>mode</i>    | Timer mode to use for the channel.       |

**20.7.5 static void MRT\_EnableInterrupts ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]**

## Parameters

|                |   |
|----------------|---|
| <i>base</i>    | Multi-Rate timer peripheral base address  |
| <i>channel</i> | Timer channel number  |
| <i>mask</i>    | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">mrt_interrupt_enable_t</a> |

**20.7.6** `static void MRT_DisableInterrupts ( MRT_Type * base, mrt_chnl_t channel, uint32_t mask ) [inline], [static]`

## Parameters

|                |  |
|----------------|--|
| <i>base</i>    | Multi-Rate timer peripheral base address   |
| <i>channel</i> | Timer channel number   |
| <i>mask</i>    | The interrupts to disable. This is a logical OR of members of the enumeration <a href="#">mrt_interrupt_enable_t</a> |

**20.7.7** `static uint32_t MRT_GetEnabledInterrupts ( MRT_Type * base, mrt_chnl_t channel ) [inline], [static]`

## Parameters

|                |  |
|----------------|--|
| <i>base</i>    | Multi-Rate timer peripheral base address |
| <i>channel</i> | Timer channel number                     |

## Returns

The enabled interrupts. This is the logical OR of members of the enumeration [mrt\\_interrupt\\_enable\\_t](#)

**20.7.8** `static uint32_t MRT_GetStatusFlags ( MRT_Type * base, mrt_chnl_t channel ) [inline], [static]`

## Parameters

|                |  |
|----------------|--|
| <i>base</i>    | Multi-Rate timer peripheral base address |
| <i>channel</i> | Timer channel number                     |

## Returns

The status flags. This is the logical OR of members of the enumeration [mrt\\_status\\_flags\\_t](#)

### 20.7.9 static void MRT\_ClearStatusFlags ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, uint32\_t *mask* ) [inline], [static]

## Parameters

|                |  |
|----------------|--|
| <i>base</i>    | Multi-Rate timer peripheral base address   |
| <i>channel</i> | Timer channel number   |
| <i>mask</i>    | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">mrt_status_flags_t</a> |

### 20.7.10 void MRT\_UpdateTimerPeriod ( MRT\_Type \* *base*, mrt\_chnl\_t *channel*, uint32\_t *count*, bool *immediateLoad* )

The new value will be immediately loaded or will be loaded at the end of the current time interval. For one-shot interrupt mode the new value will be immediately loaded.

## Note

User can call the utility macros provided in `fsl_common.h` to convert to ticks

## Parameters

|                |  |
|----------------|--|
| <i>base</i>    | Multi-Rate timer peripheral base address |
| <i>channel</i> | Timer channel number                     |

|                      |  |
|----------------------|--|
| <i>count</i>         | Timer period in units of ticks   |
| <i>immediateLoad</i> | true: Load the new value immediately into the TIMER register; false: Load the new value at the end of current timer interval |

### 20.7.11 `static uint32_t MRT_GetCurrentTimerCount ( MRT_Type * base, mrt_chnl_t channel ) [inline], [static]`

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

User can call the utility macros provided in `fsl_common.h` to convert ticks to usec or msec

Parameters

|                |  |
|----------------|--|
| <i>base</i>    | Multi-Rate timer peripheral base address |
| <i>channel</i> | Timer channel number                     |

Returns

Current timer counting value in ticks

### 20.7.12 `static void MRT_StartTimer ( MRT_Type * base, mrt_chnl_t channel, uint32_t count ) [inline], [static]`

After calling this function, timers load period value, counts down to 0 and depending on the timer mode it will either load the respective start value again or stop.

Note

User can call the utility macros provided in `fsl_common.h` to convert to ticks

Parameters

---

|                |   |
|----------------|---|
| <i>base</i>    | Multi-Rate timer peripheral base address  |
| <i>channel</i> | Timer channel number.   |
| <i>count</i>   | Timer period in units of ticks. Count can contain the LOAD bit, which control the force load feature. |

### 20.7.13 `static void MRT_StopTimer ( MRT_Type * base, mrt_chnl_t channel ) [inline], [static]`

This function stops the timer from counting.

Parameters

|                |  |
|----------------|--|
| <i>base</i>    | Multi-Rate timer peripheral base address |
| <i>channel</i> | Timer channel number.                    |

### 20.7.14 `static uint32_t MRT_GetIdleChannel ( MRT_Type * base ) [inline], [static]`

This function returns the lowest available channel number.

Parameters

|             |  |
|-------------|--|
| <i>base</i> | Multi-Rate timer peripheral base address |
|-------------|--|

## Chapter 21

# PINT: Pin Interrupt and Pattern Match Driver

### 21.1 Overview

The MCUXpresso SDK provides a driver for the Pin Interrupt and Pattern match (PINT).

It can configure one or more pins to generate a pin interrupt when the pin or pattern match conditions are met. The pins do not have to be configured as gpio pins however they must be connected to PINT via INPUTMUX. Only the pin interrupt or pattern match function can be active for interrupt generation. If the pin interrupt function is enabled then the pattern match function can be used for wakeup via RXEV.

### 21.2 Pin Interrupt and Pattern match Driver operation

[PINT\\_PinInterruptConfig\(\)](#) function configures the pins for pin interrupt.

[PINT\\_PatternMatchConfig\(\)](#) function configures the pins for pattern match.

#### 21.2.1 Pin Interrupt use case

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

#### 21.2.2 Pattern match use case

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

### Files

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

### Typedefs

- typedef enum [\\_pint\\_pin\\_enable](#) [pint\\_pin\\_enable\\_t](#)  
*PINT Pin Interrupt enable type.*
- typedef enum [\\_pint\\_int](#) [pint\\_pin\\_int\\_t](#)  
*PINT Pin Interrupt type.*
- typedef enum [\\_pint\\_pmatch\\_input\\_src](#) [pint\\_pmatch\\_input\\_src\\_t](#)  
*PINT Pattern Match bit slice input source type.*
- typedef enum [\\_pint\\_pmatch\\_bslice](#) [pint\\_pmatch\\_bslice\\_t](#)  
*PINT Pattern Match bit slice type.*
- typedef enum [\\_pint\\_pmatch\\_bslice\\_cfg](#) [pint\\_pmatch\\_bslice\\_cfg\\_t](#)  
*PINT Pattern Match configuration type.*

- typedef void(\* [pint\\_cb\\_t](#))([pint\\_pin\\_int\\_t](#) pintr, uint32\_t pmatch\_status)  
*PINT Callback function.*

## Enumerations

- enum [\\_pint\\_pin\\_enable](#) {  
[kPINT\\_PinIntEnableNone](#) = 0U,  
[kPINT\\_PinIntEnableRiseEdge](#) = PINT\_PIN\_RISE\_EDGE,  
[kPINT\\_PinIntEnableFallEdge](#) = PINT\_PIN\_FALL\_EDGE,  
[kPINT\\_PinIntEnableBothEdges](#) = PINT\_PIN\_BOTH\_EDGE,  
[kPINT\\_PinIntEnableLowLevel](#) = PINT\_PIN\_LOW\_LEVEL,  
[kPINT\\_PinIntEnableHighLevel](#) = PINT\_PIN\_HIGH\_LEVEL }

*PINT Pin Interrupt enable type.*

- enum [\\_pint\\_int](#) {  
[kPINT\\_PinInt0](#) = 0U,  
[kPINT\\_PinInt1](#) = 1U,  
[kPINT\\_PinInt2](#) = 2U,  
[kPINT\\_PinInt3](#) = 3U,  
[kPINT\\_PinInt4](#) = 4U,  
[kPINT\\_PinInt5](#) = 5U,  
[kPINT\\_PinInt6](#) = 6U,  
[kPINT\\_PinInt7](#) = 7U }

*PINT Pin Interrupt type.*

- enum [\\_pint\\_pmatch\\_input\\_src](#) {  
[kPINT\\_PatternMatchInp0Src](#) = 0U,  
[kPINT\\_PatternMatchInp1Src](#) = 1U,  
[kPINT\\_PatternMatchInp2Src](#) = 2U,  
[kPINT\\_PatternMatchInp3Src](#) = 3U,  
[kPINT\\_PatternMatchInp4Src](#) = 4U,  
[kPINT\\_PatternMatchInp5Src](#) = 5U,  
[kPINT\\_PatternMatchInp6Src](#) = 6U,  
[kPINT\\_PatternMatchInp7Src](#) = 7U,  
[kPINT\\_SecPatternMatchInp0Src](#) = 0U,  
[kPINT\\_SecPatternMatchInp1Src](#) = 1U }

*PINT Pattern Match bit slice input source type.*

- enum [\\_pint\\_pmatch\\_bslic](#) {  
[kPINT\\_PatternMatchBSlice0](#) = 0U,  
[kPINT\\_PatternMatchBSlice1](#) = 1U,  
[kPINT\\_PatternMatchBSlice2](#) = 2U,  
[kPINT\\_PatternMatchBSlice3](#) = 3U,  
[kPINT\\_PatternMatchBSlice4](#) = 4U,  
[kPINT\\_PatternMatchBSlice5](#) = 5U,  
[kPINT\\_PatternMatchBSlice6](#) = 6U,  
[kPINT\\_PatternMatchBSlice7](#) = 7U }

*PINT Pattern Match bit slice type.*

- enum [\\_pint\\_pmatch\\_bslic\\_cfg](#) {

```

kPINT_PatternMatchAlways = 0U,
kPINT_PatternMatchStickyRise = 1U,
kPINT_PatternMatchStickyFall = 2U,
kPINT_PatternMatchStickyBothEdges = 3U,
kPINT_PatternMatchHigh = 4U,
kPINT_PatternMatchLow = 5U,
kPINT_PatternMatchNever = 6U,
kPINT_PatternMatchBothEdges = 7U }
    PINT Pattern Match configuration type.

```

## Functions

- void **PINT\_Init** (PINT\_Type \*base)  
*Initialize PINT peripheral.*
- void **PINT\_PinInterruptConfig** (PINT\_Type \*base, pint\_pin\_int\_t intr, pint\_pin\_enable\_t enable, pint\_cb\_t callback)  
*Configure PINT peripheral pin interrupt.*
- void **PINT\_PinInterruptGetConfig** (PINT\_Type \*base, pint\_pin\_int\_t pintr, pint\_pin\_enable\_t \*enable, pint\_cb\_t \*callback)  
*Get PINT peripheral pin interrupt configuration.*
- void **PINT\_PinInterruptClrStatus** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.*
- static uint32\_t **PINT\_PinInterruptGetStatus** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Get Selected pin interrupt status.*
- void **PINT\_PinInterruptClrStatusAll** (PINT\_Type \*base)  
*Clear all pin interrupts status only when pins were triggered by edge-sensitive.*
- static uint32\_t **PINT\_PinInterruptGetStatusAll** (PINT\_Type \*base)  
*Get all pin interrupts status.*
- static void **PINT\_PinInterruptClrFallFlag** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Clear Selected pin interrupt fall flag.*
- static uint32\_t **PINT\_PinInterruptGetFallFlag** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Get selected pin interrupt fall flag.*
- static void **PINT\_PinInterruptClrFallFlagAll** (PINT\_Type \*base)  
*Clear all pin interrupt fall flags.*
- static uint32\_t **PINT\_PinInterruptGetFallFlagAll** (PINT\_Type \*base)  
*Get all pin interrupt fall flags.*
- static void **PINT\_PinInterruptClrRiseFlag** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Clear Selected pin interrupt rise flag.*
- static uint32\_t **PINT\_PinInterruptGetRiseFlag** (PINT\_Type \*base, pint\_pin\_int\_t pintr)  
*Get selected pin interrupt rise flag.*
- static void **PINT\_PinInterruptClrRiseFlagAll** (PINT\_Type \*base)  
*Clear all pin interrupt rise flags.*
- static uint32\_t **PINT\_PinInterruptGetRiseFlagAll** (PINT\_Type \*base)  
*Get all pin interrupt rise flags.*
- void **PINT\_PatternMatchConfig** (PINT\_Type \*base, pint\_pmatch\_bslice\_t bslice, pint\_pmatch\_cfg\_t \*cfg)  
*Configure PINT pattern match.*
- void **PINT\_PatternMatchGetConfig** (PINT\_Type \*base, pint\_pmatch\_bslice\_t bslice, pint\_pmatch\_cfg\_t \*cfg)  
*Get PINT pattern match configuration.*

- static uint32\_t [PINT\\_PatternMatchGetStatus](#) (PINT\_Type \*base, [pint\\_pmatch\\_bslice\\_t](#) bslice)  
*Get pattern match bit slice status.*
- static uint32\_t [PINT\\_PatternMatchGetStatusAll](#) (PINT\_Type \*base)  
*Get status of all pattern match bit slices.*
- uint32\_t [PINT\\_PatternMatchResetDetectLogic](#) (PINT\_Type \*base)  
*Reset pattern match detection logic.*
- static void [PINT\\_PatternMatchEnable](#) (PINT\_Type \*base)  
*Enable pattern match function.*
- static void [PINT\\_PatternMatchDisable](#) (PINT\_Type \*base)  
*Disable pattern match function.*
- static void [PINT\\_PatternMatchEnableRXEV](#) (PINT\_Type \*base)  
*Enable RXEV output.*
- static void [PINT\\_PatternMatchDisableRXEV](#) (PINT\_Type \*base)  
*Disable RXEV output.*
- void [PINT\\_EnableCallback](#) (PINT\_Type \*base)  
*Enable callback.*
- void [PINT\\_DisableCallback](#) (PINT\_Type \*base)  
*Disable callback.*
- void [PINT\\_Deinit](#) (PINT\_Type \*base)  
*Deinitialize PINT peripheral.*
- void [PINT\\_EnableCallbackByIndex](#) (PINT\_Type \*base, [pint\\_pin\\_int\\_t](#) pintIdx)  
*enable callback by pin index.*
- void [PINT\\_DisableCallbackByIndex](#) (PINT\_Type \*base, [pint\\_pin\\_int\\_t](#) pintIdx)  
*disable callback by pin index.*

### Driver version

- #define [FSL\\_PINT\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 12))

## 21.3 Typedef Documentation

### 21.3.1 typedef void(\* pint\_cb\_t)(pint\_pin\_int\_t pintr, uint32\_t pmatch\_status)

## 21.4 Enumeration Type Documentation

### 21.4.1 enum \_pint\_pin\_enable

Enumerator

- kPINT\_PinIntEnableNone*** Do not generate Pin Interrupt.
- kPINT\_PinIntEnableRiseEdge*** Generate Pin Interrupt on rising edge.
- kPINT\_PinIntEnableFalleEdge*** Generate Pin Interrupt on falling edge.
- kPINT\_PinIntEnableBothEdges*** Generate Pin Interrupt on both edges.
- kPINT\_PinIntEnableLowLevel*** Generate Pin Interrupt on low level.
- kPINT\_PinIntEnableHighLevel*** Generate Pin Interrupt on high level.

### 21.4.2 enum \_pint\_int

Enumerator

*kPINT\_PinInt0* Pin Interrupt 0.  
*kPINT\_PinInt1* Pin Interrupt 1.  
*kPINT\_PinInt2* Pin Interrupt 2.  
*kPINT\_PinInt3* Pin Interrupt 3.  
*kPINT\_PinInt4* Pin Interrupt 4.  
*kPINT\_PinInt5* Pin Interrupt 5.  
*kPINT\_PinInt6* Pin Interrupt 6.  
*kPINT\_PinInt7* Pin Interrupt 7.

### 21.4.3 enum \_pint\_pmatch\_input\_src

Enumerator

*kPINT\_PatternMatchInp0Src* Input source 0.  
*kPINT\_PatternMatchInp1Src* Input source 1.  
*kPINT\_PatternMatchInp2Src* Input source 2.  
*kPINT\_PatternMatchInp3Src* Input source 3.  
*kPINT\_PatternMatchInp4Src* Input source 4.  
*kPINT\_PatternMatchInp5Src* Input source 5.  
*kPINT\_PatternMatchInp6Src* Input source 6.  
*kPINT\_PatternMatchInp7Src* Input source 7.  
*kPINT\_SecPatternMatchInp0Src* Input source 0.  
*kPINT\_SecPatternMatchInp1Src* Input source 1.

### 21.4.4 enum \_pint\_pmatch\_bslice

Enumerator

*kPINT\_PatternMatchBSlice0* Bit slice 0.  
*kPINT\_PatternMatchBSlice1* Bit slice 1.  
*kPINT\_PatternMatchBSlice2* Bit slice 2.  
*kPINT\_PatternMatchBSlice3* Bit slice 3.  
*kPINT\_PatternMatchBSlice4* Bit slice 4.  
*kPINT\_PatternMatchBSlice5* Bit slice 5.  
*kPINT\_PatternMatchBSlice6* Bit slice 6.  
*kPINT\_PatternMatchBSlice7* Bit slice 7.

## 21.4.5 enum \_pint\_pmatch\_bslice\_cfg

Enumerator

*kPINT\_PatternMatchAlways* Always Contributes to product term match.

*kPINT\_PatternMatchStickyRise* Sticky Rising edge.

*kPINT\_PatternMatchStickyFall* Sticky Falling edge.

*kPINT\_PatternMatchStickyBothEdges* Sticky Rising or Falling edge.

*kPINT\_PatternMatchHigh* High level.

*kPINT\_PatternMatchLow* Low level.

*kPINT\_PatternMatchNever* Never contributes to product term match.

*kPINT\_PatternMatchBothEdges* Either rising or falling edge.

## 21.5 Function Documentation

### 21.5.1 void PINT\_Init ( PINT\_Type \* *base* )

This function initializes the PINT peripheral and enables the clock.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

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

### 21.5.2 void PINT\_PinInterruptConfig ( PINT\_Type \* *base*, pint\_pin\_int\_t *intr*, pint\_pin\_enable\_t *enable*, pint\_cb\_t *callback* )

This function configures a given pin interrupt.

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>base</i>     | Base address of the PINT peripheral. |
| <i>intr</i>     | Pin interrupt.                       |
| <i>enable</i>   | Selects detection logic.             |
| <i>callback</i> | Callback.                            |

Return values

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

### 21.5.3 void PINT\_PinInterruptGetConfig ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr*, pint\_pin\_enable\_t \* *enable*, pint\_cb\_t \* *callback* )

This function returns the configuration of a given pin interrupt.

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>base</i>     | Base address of the PINT peripheral.  |
| <i>pintr</i>    | Pin interrupt.                        |
| <i>enable</i>   | Pointer to store the detection logic. |
| <i>callback</i> | Callback.                             |

Return values

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

### 21.5.4 void PINT\_PinInterruptClrStatus ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* )

This function clears the selected pin interrupt status.

Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

Return values

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

### 21.5.5 static uint32\_t PINT\_PinInterruptGetStatus ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* ) [inline], [static]

This function returns the selected pin interrupt status.

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

## Return values

|               |  |
|---------------|--|
| <i>status</i> | = 0 No pin interrupt request. = 1 Selected Pin interrupt request active. |
|---------------|--|

**21.5.6 void PINT\_PinInterruptClrStatusAll ( PINT\_Type \* *base* )**

This function clears the status of all pin interrupts.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

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

**21.5.7 static uint32\_t PINT\_PinInterruptGetStatusAll ( PINT\_Type \* *base* )  
[inline], [static]**

This function returns the status of all pin interrupts.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

|               |  |
|---------------|--|
| <i>status</i> | Each bit position indicates the status of corresponding pin interrupt. = 0 No pin interrupt request. = 1 Pin interrupt request active. |
|---------------|--|

**21.5.8 static void PINT\_PinInterruptClrFallFlag ( PINT\_Type \* *base*, pint\_pin\_int\_t  
*pintr* ) [inline], [static]**

This function clears the selected pin interrupt fall flag.

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

## Return values

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

### 21.5.9 static uint32\_t PINT\_PinInterruptGetFallFlag ( PINT\_Type \* *base*, pintr\_pin\_int\_t *pintr* ) [inline], [static]

This function returns the selected pin interrupt fall flag.

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

## Return values

|             |   |
|-------------|---|
| <i>flag</i> | = 0 Falling edge has not been detected. = 1 Falling edge has been detected. |
|-------------|---|

### 21.5.10 static void PINT\_PinInterruptClrFallFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function clears the fall flag for all pin interrupts.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

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

### 21.5.11 static uint32\_t PINT\_PinInterruptGetFallFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the fall flag of all pin interrupts.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

|              |  |
|--------------|--|
| <i>flags</i> | Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected. |
|--------------|--|

### 21.5.12 static void PINT\_PinInterruptClrRiseFlag ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* ) [inline], [static]

This function clears the selected pin interrupt rise flag.

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

## Return values

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

### 21.5.13 static uint32\_t PINT\_PinInterruptGetRiseFlag ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* ) [inline], [static]

This function returns the selected pin interrupt rise flag.

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

## Return values

---

|             |   |
|-------------|---|
| <i>flag</i> | = 0 Rising edge has not been detected. = 1 Rising edge has been detected. |
|-------------|---|

#### 21.5.14 static void PINT\_PinInterruptClrRiseFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function clears the rise flag for all pin interrupts.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

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

#### 21.5.15 static uint32\_t PINT\_PinInterruptGetRiseFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the rise flag of all pin interrupts.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |   |
|--------------|---|
| <i>flags</i> | Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected. |
|--------------|---|

#### 21.5.16 void PINT\_PatternMatchConfig ( PINT\_Type \* *base*, pint\_pmatch\_bslice\_t *bslice*, pint\_pmatch\_cfg\_t \* *cfg* )

This function configures a given pattern match bit slice.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | Base address of the PINT peripheral. |
| <i>bslice</i> | Pattern match bit slice number.      |
| <i>cfg</i>    | Pointer to bit slice configuration.  |

## Return values

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

### 21.5.17 void PINT\_PatternMatchGetConfig ( PINT\_Type \* *base*, pint\_pmatch\_bslice\_t *bslice*, pint\_pmatch\_cfg\_t \* *cfg* )

This function returns the configuration of a given pattern match bit slice.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | Base address of the PINT peripheral. |
| <i>bslice</i> | Pattern match bit slice number.      |
| <i>cfg</i>    | Pointer to bit slice configuration.  |

## Return values

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

### 21.5.18 static uint32\_t PINT\_PatternMatchGetStatus ( PINT\_Type \* *base*, pint\_pmatch\_bslice\_t *bslice* ) [inline], [static]

This function returns the status of selected bit slice.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | Base address of the PINT peripheral. |
| <i>bslice</i> | Pattern match bit slice number.      |

## Return values

---

|               |   |
|---------------|---|
| <i>status</i> | = 0 Match has not been detected. = 1 Match has been detected. |
|---------------|---|

### 21.5.19 static uint32\_t PINT\_PatternMatchGetStatusAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the status of all bit slices.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|               |  |
|---------------|--|
| <i>status</i> | Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected. |
|---------------|--|

### 21.5.20 uint32\_t PINT\_PatternMatchResetDetectLogic ( PINT\_Type \* *base* )

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|                 |  |
|-----------------|--|
| <i>pmstatus</i> | Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected. |
|-----------------|--|

### 21.5.21 static void PINT\_PatternMatchEnable ( PINT\_Type \* *base* ) [inline], [static]

This function enables the pattern match function.

Parameters

---

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

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

### 21.5.22 `static void PINT_PatternMatchDisable ( PINT_Type * base ) [inline], [static]`

This function disables the pattern match function.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

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

### 21.5.23 `static void PINT_PatternMatchEnableRXEV ( PINT_Type * base ) [inline], [static]`

This function enables the pattern match RXEV output.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

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

### 21.5.24 `static void PINT_PatternMatchDisableRXEV ( PINT_Type * base ) [inline], [static]`

This function disables the pattern match RXEV output.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

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

**21.5.25 void PINT\_EnableCallback ( PINT\_Type \* *base* )**

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

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

**21.5.26 void PINT\_DisableCallback ( PINT\_Type \* *base* )**

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | Base address of the peripheral. |
|-------------|---------------------------------|

## Return values

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

**21.5.27 void PINT\_Deinit ( PINT\_Type \* *base* )**

This function disables the PINT clock.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

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

### 21.5.28 void PINT\_EnableCallbackByIndex ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintIdx* )

This function enables callback by pin index instead of enabling all pins.

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | Base address of the peripheral. |
| <i>pintIdx</i> | pin index.                      |

## Return values

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

### 21.5.29 void PINT\_DisableCallbackByIndex ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintIdx* )

This function disables callback by pin index instead of disabling all pins.

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | Base address of the peripheral. |
| <i>pintIdx</i> | pin index.                      |

## Return values

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

## Chapter 22

# PLU: Programmable Logic Unit

### 22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Programmable Logic Unit module of MCU-Xpresso SDK devices.

### 22.2 Function groups

The PLU driver supports the creation of small combinatorial and/or sequential logic networks including simple state machines.

#### 22.2.1 Initialization and de-initialization

The function `PLU_Init()` enables the PLU clock and reset the module.

The function `PIT_Deinit()` gates the PLU clock.

#### 22.2.2 Set input/output source and Truth Table

The function `PLU_SetLutInputSource()` sets the input source for the LUT element.

The function `PLU_SetOutputSource()` sets output source of the PLU module.

The function `PLU_SetLutTruthTable()` sets the truth table for the LUT element.

#### 22.2.3 Read current Output State

The function `PLU_ReadOutputState()` reads the current state of the 8 designated PLU Outputs.

#### 22.2.4 Wake-up/Interrupt Control

The function `PLU_EnableWakeIntRequest()` enables the wake-up/interrupt request on a PLU output pin with a optional configuration to eliminate the glitches. The function `PLU_GetDefaultWakeIntConfig()` gets the default configuration which can be used in a case with a given `PLU_CLKIN`.

The function `PLU_LatchInterrupt()` latches the interrupt and it can be cleared by function `PLU_ClearLatchedInterrupt()`.

## 22.3 Typical use case

### 22.3.1 PLU combination example

Create a simple combinatorial logic network to control the LED. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/plu/combination`

#### Typedefs

- typedef enum `_plu_lut_index` `plu_lut_index_t`  
*Index of LUT.*
- typedef enum `_plu_lut_in_index` `plu_lut_in_index_t`  
*Inputs of LUT.*
- typedef enum `_plu_lut_input_source` `plu_lut_input_source_t`  
*Available sources of LUT input.*
- typedef enum `_plu_output_index` `plu_output_index_t`  
*PLU output multiplexer registers.*
- typedef enum `_plu_output_source` `plu_output_source_t`  
*Available sources of PLU output.*

## Enumerations

- enum `_plu_lut_index` {
  - `kPLU_LUT_0` = 0U,
  - `kPLU_LUT_1` = 1U,
  - `kPLU_LUT_2` = 2U,
  - `kPLU_LUT_3` = 3U,
  - `kPLU_LUT_4` = 4U,
  - `kPLU_LUT_5` = 5U,
  - `kPLU_LUT_6` = 6U,
  - `kPLU_LUT_7` = 7U,
  - `kPLU_LUT_8` = 8U,
  - `kPLU_LUT_9` = 9U,
  - `kPLU_LUT_10` = 10U,
  - `kPLU_LUT_11` = 11U,
  - `kPLU_LUT_12` = 12U,
  - `kPLU_LUT_13` = 13U,
  - `kPLU_LUT_14` = 14U,
  - `kPLU_LUT_15` = 15U,
  - `kPLU_LUT_16` = 16U,
  - `kPLU_LUT_17` = 17U,
  - `kPLU_LUT_18` = 18U,
  - `kPLU_LUT_19` = 19U,
  - `kPLU_LUT_20` = 20U,
  - `kPLU_LUT_21` = 21U,
  - `kPLU_LUT_22` = 22U,
  - `kPLU_LUT_23` = 23U,
  - `kPLU_LUT_24` = 24U,
  - `kPLU_LUT_25` = 25U }

*Index of LUT.*
- enum `_plu_lut_in_index` {
  - `kPLU_LUT_IN_0` = 0U,
  - `kPLU_LUT_IN_1` = 1U,
  - `kPLU_LUT_IN_2` = 2U,
  - `kPLU_LUT_IN_3` = 3U,
  - `kPLU_LUT_IN_4` = 4U }

*Inputs of LUT.*
- enum `_plu_lut_input_source` {

```

kPLU_LUT_IN_SRC_PLU_IN_0 = 0U,
kPLU_LUT_IN_SRC_PLU_IN_1 = 1U,
kPLU_LUT_IN_SRC_PLU_IN_2 = 2U,
kPLU_LUT_IN_SRC_PLU_IN_3 = 3U,
kPLU_LUT_IN_SRC_PLU_IN_4 = 4U,
kPLU_LUT_IN_SRC_PLU_IN_5 = 5U,
kPLU_LUT_IN_SRC_LUT_OUT_0 = 6U,
kPLU_LUT_IN_SRC_LUT_OUT_1 = 7U,
kPLU_LUT_IN_SRC_LUT_OUT_2 = 8U,
kPLU_LUT_IN_SRC_LUT_OUT_3 = 9U,
kPLU_LUT_IN_SRC_LUT_OUT_4 = 10U,
kPLU_LUT_IN_SRC_LUT_OUT_5 = 11U,
kPLU_LUT_IN_SRC_LUT_OUT_6 = 12U,
kPLU_LUT_IN_SRC_LUT_OUT_7 = 13U,
kPLU_LUT_IN_SRC_LUT_OUT_8 = 14U,
kPLU_LUT_IN_SRC_LUT_OUT_9 = 15U,
kPLU_LUT_IN_SRC_LUT_OUT_10 = 16U,
kPLU_LUT_IN_SRC_LUT_OUT_11 = 17U,
kPLU_LUT_IN_SRC_LUT_OUT_12 = 18U,
kPLU_LUT_IN_SRC_LUT_OUT_13 = 19U,
kPLU_LUT_IN_SRC_LUT_OUT_14 = 20U,
kPLU_LUT_IN_SRC_LUT_OUT_15 = 21U,
kPLU_LUT_IN_SRC_LUT_OUT_16 = 22U,
kPLU_LUT_IN_SRC_LUT_OUT_17 = 23U,
kPLU_LUT_IN_SRC_LUT_OUT_18 = 24U,
kPLU_LUT_IN_SRC_LUT_OUT_19 = 25U,
kPLU_LUT_IN_SRC_LUT_OUT_20 = 26U,
kPLU_LUT_IN_SRC_LUT_OUT_21 = 27U,
kPLU_LUT_IN_SRC_LUT_OUT_22 = 28U,
kPLU_LUT_IN_SRC_LUT_OUT_23 = 29U,
kPLU_LUT_IN_SRC_LUT_OUT_24 = 30U,
kPLU_LUT_IN_SRC_LUT_OUT_25 = 31U,
kPLU_LUT_IN_SRC_FLIPFLOP_0 = 32U,
kPLU_LUT_IN_SRC_FLIPFLOP_1 = 33U,
kPLU_LUT_IN_SRC_FLIPFLOP_2 = 34U,
kPLU_LUT_IN_SRC_FLIPFLOP_3 = 35U }

```

*Available sources of LUT input.*

- enum `_plu_output_index` {
 

```

kPLU_OUTPUT_0 = 0U,
kPLU_OUTPUT_1 = 1U,
kPLU_OUTPUT_2 = 2U,
kPLU_OUTPUT_3 = 3U,
kPLU_OUTPUT_4 = 4U,
kPLU_OUTPUT_5 = 5U,
kPLU_OUTPUT_6 = 6U,

```

```
kPLU_OUTPUT_7 = 7U }
```

*PLU output multiplexer registers.*

- enum `_plu_output_source` {
  - `kPLU_OUT_SRC_LUT_0` = 0U,
  - `kPLU_OUT_SRC_LUT_1` = 1U,
  - `kPLU_OUT_SRC_LUT_2` = 2U,
  - `kPLU_OUT_SRC_LUT_3` = 3U,
  - `kPLU_OUT_SRC_LUT_4` = 4U,
  - `kPLU_OUT_SRC_LUT_5` = 5U,
  - `kPLU_OUT_SRC_LUT_6` = 6U,
  - `kPLU_OUT_SRC_LUT_7` = 7U,
  - `kPLU_OUT_SRC_LUT_8` = 8U,
  - `kPLU_OUT_SRC_LUT_9` = 9U,
  - `kPLU_OUT_SRC_LUT_10` = 10U,
  - `kPLU_OUT_SRC_LUT_11` = 11U,
  - `kPLU_OUT_SRC_LUT_12` = 12U,
  - `kPLU_OUT_SRC_LUT_13` = 13U,
  - `kPLU_OUT_SRC_LUT_14` = 14U,
  - `kPLU_OUT_SRC_LUT_15` = 15U,
  - `kPLU_OUT_SRC_LUT_16` = 16U,
  - `kPLU_OUT_SRC_LUT_17` = 17U,
  - `kPLU_OUT_SRC_LUT_18` = 18U,
  - `kPLU_OUT_SRC_LUT_19` = 19U,
  - `kPLU_OUT_SRC_LUT_20` = 20U,
  - `kPLU_OUT_SRC_LUT_21` = 21U,
  - `kPLU_OUT_SRC_LUT_22` = 22U,
  - `kPLU_OUT_SRC_LUT_23` = 23U,
  - `kPLU_OUT_SRC_LUT_24` = 24U,
  - `kPLU_OUT_SRC_LUT_25` = 25U,
  - `kPLU_OUT_SRC_FLIPFLOP_0` = 26U,
  - `kPLU_OUT_SRC_FLIPFLOP_1` = 27U,
  - `kPLU_OUT_SRC_FLIPFLOP_2` = 28U,
  - `kPLU_OUT_SRC_FLIPFLOP_3` = 29U }

*Available sources of PLU output.*

## Driver version

- #define `FSL_PLU_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 1)`)  
*Version 2.2.1.*

## Initialization and deinitialization

- void `PLU_Init` (`PLU_Type *base`)  
*Enable the PLU clock and reset the module.*
- void `PLU_Deinit` (`PLU_Type *base`)  
*Gate the PLU clock.*

## Set input/output source and Truth Table

- static void `PLU_SetLutInputSource` (PLU\_Type \*base, `plu_lut_index_t` lutIndex, `plu_lut_in_index_t` lutInIndex, `plu_lut_input_source_t` inputSrc)  
*Set Input source of LUT.*
- static void `PLU_SetOutputSource` (PLU\_Type \*base, `plu_output_index_t` outputIndex, `plu_output_source_t` outputSrc)  
*Set Output source of PLU.*
- static void `PLU_SetLutTruthTable` (PLU\_Type \*base, `plu_lut_index_t` lutIndex, `uint32_t` truthTable)  
*Set Truth Table of LUT.*

## Read current Output State

- static `uint32_t` `PLU_ReadOutputState` (PLU\_Type \*base)  
*Read the current state of the 8 designated PLU Outputs.*

## 22.4 Typedef Documentation

### 22.4.1 typedef enum `_plu_lut_in_index` `plu_lut_in_index_t`

5 input present for each LUT.

## 22.5 Enumeration Type Documentation

### 22.5.1 enum `_plu_lut_index`

Enumerator

`kPLU_LUT_0` 5-input Look-up Table 0  
`kPLU_LUT_1` 5-input Look-up Table 1  
`kPLU_LUT_2` 5-input Look-up Table 2  
`kPLU_LUT_3` 5-input Look-up Table 3  
`kPLU_LUT_4` 5-input Look-up Table 4  
`kPLU_LUT_5` 5-input Look-up Table 5  
`kPLU_LUT_6` 5-input Look-up Table 6  
`kPLU_LUT_7` 5-input Look-up Table 7  
`kPLU_LUT_8` 5-input Look-up Table 8  
`kPLU_LUT_9` 5-input Look-up Table 9  
`kPLU_LUT_10` 5-input Look-up Table 10  
`kPLU_LUT_11` 5-input Look-up Table 11  
`kPLU_LUT_12` 5-input Look-up Table 12  
`kPLU_LUT_13` 5-input Look-up Table 13  
`kPLU_LUT_14` 5-input Look-up Table 14  
`kPLU_LUT_15` 5-input Look-up Table 15  
`kPLU_LUT_16` 5-input Look-up Table 16  
`kPLU_LUT_17` 5-input Look-up Table 17

*kPLU\_LUT\_18* 5-input Look-up Table 18  
*kPLU\_LUT\_19* 5-input Look-up Table 19  
*kPLU\_LUT\_20* 5-input Look-up Table 20  
*kPLU\_LUT\_21* 5-input Look-up Table 21  
*kPLU\_LUT\_22* 5-input Look-up Table 22  
*kPLU\_LUT\_23* 5-input Look-up Table 23  
*kPLU\_LUT\_24* 5-input Look-up Table 24  
*kPLU\_LUT\_25* 5-input Look-up Table 25

### 22.5.2 enum \_plu\_lut\_in\_index

5 input present for each LUT.

Enumerator

*kPLU\_LUT\_IN\_0* LUT input 0.  
*kPLU\_LUT\_IN\_1* LUT input 1.  
*kPLU\_LUT\_IN\_2* LUT input 2.  
*kPLU\_LUT\_IN\_3* LUT input 3.  
*kPLU\_LUT\_IN\_4* LUT input 4.

### 22.5.3 enum \_plu\_lut\_input\_source

Enumerator

*kPLU\_LUT\_IN\_SRC\_PLU\_IN\_0* Select PLU input 0 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_PLU\_IN\_1* Select PLU input 1 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_PLU\_IN\_2* Select PLU input 2 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_PLU\_IN\_3* Select PLU input 3 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_PLU\_IN\_4* Select PLU input 4 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_PLU\_IN\_5* Select PLU input 5 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_0* Select LUT output 0 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_1* Select LUT output 1 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_2* Select LUT output 2 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_3* Select LUT output 3 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_4* Select LUT output 4 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_5* Select LUT output 5 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_6* Select LUT output 6 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_7* Select LUT output 7 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_8* Select LUT output 8 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_9* Select LUT output 9 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_10* Select LUT output 10 to be connected to LUTn Input x.  
*kPLU\_LUT\_IN\_SRC\_LUT\_OUT\_11* Select LUT output 11 to be connected to LUTn Input x.

|                                   |  |
|-----------------------------------|--|
| <i>kPLU_LUT_IN_SRC_LUT_OUT_12</i> | Select LUT output 12 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_13</i> | Select LUT output 13 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_14</i> | Select LUT output 14 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_15</i> | Select LUT output 15 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_16</i> | Select LUT output 16 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_17</i> | Select LUT output 17 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_18</i> | Select LUT output 18 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_19</i> | Select LUT output 19 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_20</i> | Select LUT output 20 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_21</i> | Select LUT output 21 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_22</i> | Select LUT output 22 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_23</i> | Select LUT output 23 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_24</i> | Select LUT output 24 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_LUT_OUT_25</i> | Select LUT output 25 to be connected to LUTn Input x.      |
| <i>kPLU_LUT_IN_SRC_FLIPFLOP_0</i> | Select Flip-Flops state 0 to be connected to LUTn Input x. |
| <i>kPLU_LUT_IN_SRC_FLIPFLOP_1</i> | Select Flip-Flops state 1 to be connected to LUTn Input x. |
| <i>kPLU_LUT_IN_SRC_FLIPFLOP_2</i> | Select Flip-Flops state 2 to be connected to LUTn Input x. |
| <i>kPLU_LUT_IN_SRC_FLIPFLOP_3</i> | Select Flip-Flops state 3 to be connected to LUTn Input x. |

#### 22.5.4 enum\_plu\_output\_index

Enumerator

|                      |               |
|----------------------|---------------|
| <i>kPLU_OUTPUT_0</i> | PLU OUTPUT 0. |
| <i>kPLU_OUTPUT_1</i> | PLU OUTPUT 1. |
| <i>kPLU_OUTPUT_2</i> | PLU OUTPUT 2. |
| <i>kPLU_OUTPUT_3</i> | PLU OUTPUT 3. |
| <i>kPLU_OUTPUT_4</i> | PLU OUTPUT 4. |
| <i>kPLU_OUTPUT_5</i> | PLU OUTPUT 5. |
| <i>kPLU_OUTPUT_6</i> | PLU OUTPUT 6. |
| <i>kPLU_OUTPUT_7</i> | PLU OUTPUT 7. |

#### 22.5.5 enum\_plu\_output\_source

Enumerator

|                           |   |
|---------------------------|---|
| <i>kPLU_OUT_SRC_LUT_0</i> | Select LUT0 output to be connected to PLU output. |
| <i>kPLU_OUT_SRC_LUT_1</i> | Select LUT1 output to be connected to PLU output. |
| <i>kPLU_OUT_SRC_LUT_2</i> | Select LUT2 output to be connected to PLU output. |
| <i>kPLU_OUT_SRC_LUT_3</i> | Select LUT3 output to be connected to PLU output. |
| <i>kPLU_OUT_SRC_LUT_4</i> | Select LUT4 output to be connected to PLU output. |
| <i>kPLU_OUT_SRC_LUT_5</i> | Select LUT5 output to be connected to PLU output. |

***kPLU\_OUT\_SRC\_LUT\_6*** Select LUT6 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_7*** Select LUT7 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_8*** Select LUT8 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_9*** Select LUT9 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_10*** Select LUT10 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_11*** Select LUT11 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_12*** Select LUT12 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_13*** Select LUT13 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_14*** Select LUT14 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_15*** Select LUT15 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_16*** Select LUT16 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_17*** Select LUT17 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_18*** Select LUT18 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_19*** Select LUT19 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_20*** Select LUT20 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_21*** Select LUT21 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_22*** Select LUT22 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_23*** Select LUT23 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_24*** Select LUT24 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_LUT\_25*** Select LUT25 output to be connected to PLU output.  
***kPLU\_OUT\_SRC\_FLIPFLOP\_0*** Select Flip-Flops state(0) to be connected to PLU output.  
***kPLU\_OUT\_SRC\_FLIPFLOP\_1*** Select Flip-Flops state(1) to be connected to PLU output.  
***kPLU\_OUT\_SRC\_FLIPFLOP\_2*** Select Flip-Flops state(2) to be connected to PLU output.  
***kPLU\_OUT\_SRC\_FLIPFLOP\_3*** Select Flip-Flops state(3) to be connected to PLU output.

## 22.6 Function Documentation

### 22.6.1 void PLU\_Init ( PLU\_Type \* *base* )

Note

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

Parameters

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

### 22.6.2 void PLU\_Deinit ( PLU\_Type \* *base* )

## Parameters

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

**22.6.3 static void PLU\_SetLutInputSource ( PLU\_Type \* *base*, plu\_lut\_index\_t *lutIndex*, plu\_lut\_in\_index\_t *lutInIndex*, plu\_lut\_input\_source\_t *inputSrc* ) [inline], [static]**

Note: An external clock must be applied to the PLU\_CLKIN input when using FFs. For each LUT, the slot associated with the output from LUTn itself is tied low.

## Parameters

|                   |  |
|-------------------|--|
| <i>base</i>       | PLU peripheral base address.   |
| <i>lutIndex</i>   | LUT index (see <a href="#">plu_lut_index_t</a> typedef enumeration).               |
| <i>lutInIndex</i> | LUT input index (see <a href="#">plu_lut_in_index_t</a> typedef enumeration).      |
| <i>inputSrc</i>   | LUT input source (see <a href="#">plu_lut_input_source_t</a> typedef enumeration). |

**22.6.4 static void PLU\_SetOutputSource ( PLU\_Type \* *base*, plu\_output\_index\_t *outputIndex*, plu\_output\_source\_t *outputSrc* ) [inline], [static]**

Note: An external clock must be applied to the PLU\_CLKIN input when using FFs.

## Parameters

|                    |  |
|--------------------|--|
| <i>base</i>        | PLU peripheral base address.   |
| <i>outputIndex</i> | PLU output index (see <a href="#">plu_output_index_t</a> typedef enumeration).   |
| <i>outputSrc</i>   | PLU output source (see <a href="#">plu_output_source_t</a> typedef enumeration). |

**22.6.5 static void PLU\_SetLutTruthTable ( PLU\_Type \* *base*, plu\_lut\_index\_t *lutIndex*, uint32\_t *truthTable* ) [inline], [static]**

## Parameters

|                   |  |
|-------------------|--|
| <i>base</i>       | PLU peripheral base address.   |
| <i>lutIndex</i>   | LUT index (see <a href="#">plu_lut_index_t</a> typedef enumeration). |
| <i>truthTable</i> | Truth Table value.   |

### 22.6.6 `static uint32_t PLU_ReadOutputState ( PLU_Type * base ) [inline], [static]`

Note: The PLU bus clock must be re-enabled prior to reading the Outpus Register if PLU bus clock is shut-off.

Parameters

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

Returns

Current PLU output state value.

## Chapter 23

# SWM: Switch Matrix Module

### 23.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Switch Matrix Module (SWM) module of MCUXpresso SDK devices.

### 23.2 SWM: Switch Matrix Module

#### 23.2.1 SWM Operations

The function [SWM\\_SetMovablePinSelect\(\)](#) will select a movable pin designated by its GPIO port and bit numbers to a function.

The function [SWM\\_SetFixedMovablePinSelect\(\)](#) will select a fixed movable pin designated by its GPIO port and bit numbers to a function.

The function [SWM\\_SetFixedPinSelect\(\)](#) will enable a fixed-pin function in PINENABLE0 or PINENABLE1.

#### Files

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

#### Functions

- void [SWM\\_SetMovablePinSelect](#) (SWM\_Type \*base, [swm\\_select\\_movable\\_t](#) func, [swm\\_port\\_pin\\_type\\_t](#) swm\_port\_pin)  
*Assignment of digital peripheral functions to pins.*
- void [SWM\\_SetFixedMovablePinSelect](#) (SWM\_Type \*base, [swm\\_select\\_fixed\\_movable\\_t](#) func, [swm\\_fixed\\_port\\_pin\\_type\\_t](#) swm\_port\_pin)  
*Assignment of digital peripheral functions to pins.*
- void [SWM\\_SetFixedPinSelect](#) (SWM\_Type \*base, [swm\\_select\\_fixed\\_pin\\_t](#) func, bool enable)  
*Enable the fixed-pin function.*

**swm connections**

- enum `_swm_pinassignfixed_port_pin_type_t` {
  - `kSWM_PLU_INPUT0_PortPin_P0_0 = 0x00U,`
  - `kSWM_PLU_INPUT0_PortPin_P0_8 = 0x01U,`
  - `kSWM_PLU_INPUT0_PortPin_P0_17 = 0x02U,`
  - `kSWM_PLU_INPUT1_PortPin_P0_1 = 0x00U,`
  - `kSWM_PLU_INPUT1_PortPin_P0_9 = 0x01U,`
  - `kSWM_PLU_INPUT1_PortPin_P0_18 = 0x02U,`
  - `kSWM_PLU_INPUT2_PortPin_P0_2 = 0x00U,`
  - `kSWM_PLU_INPUT2_PortPin_P0_10 = 0x01U,`
  - `kSWM_PLU_INPUT2_PortPin_P0_19 = 0x02U,`
  - `kSWM_PLU_INPUT3_PortPin_P0_3 = 0x00U,`
  - `kSWM_PLU_INPUT3_PortPin_P0_11 = 0x01U,`
  - `kSWM_PLU_INPUT3_PortPin_P0_20 = 0x02U,`
  - `kSWM_PLU_INPUT4_PortPin_P0_4 = 0x00U,`
  - `kSWM_PLU_INPUT4_PortPin_P0_12 = 0x01U,`
  - `kSWM_PLU_INPUT4_PortPin_P0_21 = 0x02U,`
  - `kSWM_PLU_INPUT5_PortPin_P0_5 = 0x00U,`
  - `kSWM_PLU_INPUT5_PortPin_P0_13 = 0x01U,`
  - `kSWM_PLU_INPUT5_PortPin_P0_22 = 0x02U,`
  - `kSWM_PLU_OUT0_PortPin_P0_7 = 0x00U,`
  - `kSWM_PLU_OUT0_PortPin_P0_14 = 0x01U,`
  - `kSWM_PLU_OUT0_PortPin_P0_23 = 0x02U,`
  - `kSWM_PLU_OUT1_PortPin_P0_8 = 0x00U,`
  - `kSWM_PLU_OUT1_PortPin_P0_15 = 0x01U,`
  - `kSWM_PLU_OUT1_PortPin_P0_24 = 0x02U,`
  - `kSWM_PLU_OUT2_PortPin_P0_9 = 0x00U,`
  - `kSWM_PLU_OUT2_PortPin_P0_16 = 0x01U,`
  - `kSWM_PLU_OUT2_PortPin_P0_25 = 0x02U,`
  - `kSWM_PLU_OUT3_PortPin_P0_10 = 0x00U,`
  - `kSWM_PLU_OUT3_PortPin_P0_17 = 0x01U,`
  - `kSWM_PLU_OUT3_PortPin_P0_26 = 0x02U,`
  - `kSWM_PLU_OUT4_PortPin_P0_11 = 0x00U,`
  - `kSWM_PLU_OUT4_PortPin_P0_18 = 0x01U,`
  - `kSWM_PLU_OUT4_PortPin_P0_27 = 0x02U,`
  - `kSWM_PLU_OUT5_PortPin_P0_12 = 0x00U,`
  - `kSWM_PLU_OUT5_PortPin_P0_19 = 0x01U,`
  - `kSWM_PLU_OUT5_PortPin_P0_28 = 0x02U,`
  - `kSWM_PLU_OUT6_PortPin_P0_13 = 0x00U,`
  - `kSWM_PLU_OUT6_PortPin_P0_20 = 0x01U,`
  - `kSWM_PLU_OUT6_PortPin_P0_29 = 0x02U,`
  - `kSWM_PLU_OUT7_PortPin_P0_14 = 0x00U,`
  - `kSWM_PLU_OUT7_PortPin_P0_21 = 0x01U,`
  - `kSWM_PLU_OUT7_PortPin_P0_30 = 0x02U }`

*SWM pinassignfixed\_port\_pin number.*

- enum `_swm_port_pin_type_t` {
  - `kSWM_PortPin_P0_0 = 0U,`
  - `kSWM_PortPin_P0_1 = 1U,`
  - `kSWM_PortPin_P0_2 = 2U,`
  - `kSWM_PortPin_P0_3 = 3U,`
  - `kSWM_PortPin_P0_4 = 4U,`
  - `kSWM_PortPin_P0_5 = 5U,`
  - `kSWM_PortPin_P0_6 = 6U,`
  - `kSWM_PortPin_P0_7 = 7U,`
  - `kSWM_PortPin_P0_8 = 8U,`
  - `kSWM_PortPin_P0_9 = 9U,`
  - `kSWM_PortPin_P0_10 = 10U,`
  - `kSWM_PortPin_P0_11 = 11U,`
  - `kSWM_PortPin_P0_12 = 12U,`
  - `kSWM_PortPin_P0_13 = 13U,`
  - `kSWM_PortPin_P0_14 = 14U,`
  - `kSWM_PortPin_P0_15 = 15U,`
  - `kSWM_PortPin_P0_16 = 16U,`
  - `kSWM_PortPin_P0_17 = 17U,`
  - `kSWM_PortPin_P0_18 = 18U,`
  - `kSWM_PortPin_P0_19 = 19U,`
  - `kSWM_PortPin_P0_20 = 20U,`
  - `kSWM_PortPin_P0_21 = 21U,`
  - `kSWM_PortPin_P0_22 = 22U,`
  - `kSWM_PortPin_P0_23 = 23U,`
  - `kSWM_PortPin_P0_24 = 24U,`
  - `kSWM_PortPin_P0_25 = 25U,`
  - `kSWM_PortPin_P0_26 = 26U,`
  - `kSWM_PortPin_P0_27 = 27U,`
  - `kSWM_PortPin_P0_28 = 28U,`
  - `kSWM_PortPin_P0_29 = 29U,`
  - `kSWM_PortPin_P0_30 = 30U,`
  - `kSWM_PortPin_P0_31 = 31U,`
  - `kSWM_PortPin_Reset = 0xffU }`
- enum `_swm_pinassignfixed_select_movable_t` {

*SWM port\_pin number.*

- kSWM\_PLU\_INPUT0 = 0U,
  - kSWM\_PLU\_INPUT1 = 1U,
  - kSWM\_PLU\_INPUT2 = 2U,
  - kSWM\_PLU\_INPUT3 = 3U,
  - kSWM\_PLU\_INPUT4 = 4U,
  - kSWM\_PLU\_INPUT5 = 5U,
  - kSWM\_PLU\_OUT0 = 6U,
  - kSWM\_PLU\_OUT1 = 7U,
  - kSWM\_PLU\_OUT2 = 8U,
  - kSWM\_PLU\_OUT3 = 9U,
  - kSWM\_PLU\_OUT4 = 10U,
  - kSWM\_PLU\_OUT5 = 11U,
  - kSWM\_PLU\_OUT6 = 12U,
  - kSWM\_PLU\_OUT7 = 13U,
  - kSWM\_PINASSIGNFIXED\_MOVBABLE\_NUM\_FUNCS = 14U }
- SWM pinassignfixed movable selection.*
- enum `_swm_select_movable_t` {

```

kSWM_USART0_TXD = 0U,
kSWM_USART0_RXD = 1U,
kSWM_USART0_RTS = 2U,
kSWM_USART0_CTS = 3U,
kSWM_USART0_SCLK = 4U,
kSWM_USART1_TXD = 5U,
kSWM_USART1_RXD = 6U,
kSWM_USART1_SCLK = 7U,
kSWM_SPI0_SCK = 8U,
kSWM_SPI0_MOSI = 9U,
kSWM_SPI0_MISO = 10U,
kSWM_SPI0_SSEL0 = 11U,
kSWM_SPI0_SSEL1 = 12U,
kSWM_T0_CAP_CHN0 = 13U,
kSWM_T0_CAP_CHN1 = 14U,
kSWM_T0_CAP_CHN2 = 15U,
kSWM_T0_MAT_CHN0 = 16U,
kSWM_T0_MAT_CHN1 = 17U,
kSWM_T0_MAT_CHN2 = 18U,
kSWM_T0_MAT_CHN3 = 19U,
kSWM_I2C0_SDA = 20U,
kSWM_I2C0_SCL = 21U,
kSWM_ACMP_OUT = 22U,
kSWM_CLKOUT = 23U,
kSWM_GPIO_INT_BMAT = 24U,
kSWM_LVLSHFT_IN0 = 25U,
kSWM_LVLSHFT_IN1 = 26U,
kSWM_LVLSHFT_OUT0 = 27U,
kSWM_LVLSHFT_OUT1 = 28U,
kSWM_I2C1_SDA = 29U,
kSWM_I2C1_SCL = 30U,
kSWM_PLU_CLKIN_IN = 31U,
kSWM_CAPT_X0 = 32U,
kSWM_CAPT_X1 = 33U,
kSWM_CAPT_X2 = 34U,
kSWM_CAPT_X3 = 35U,
kSWM_CAPT_X4 = 36U,
kSWM_CAPT_YL = 37U,
kSWM_CAPT_YH = 38U,
kSWM_MOVBLE_NUM_FUNCS = 39U }

```

*SWM movable selection.*

- enum `_swm_select_fixed_pin_t` {

```

kSWM_ACOMP_INPUT1 = SWM_PINENABLE0_ACOMP_I1_MASK,
kSWM_ACOMP_INPUT2 = SWM_PINENABLE0_ACOMP_I2_MASK,
kSWM_ACOMP_INPUT3 = SWM_PINENABLE0_ACOMP_I3_MASK,
kSWM_ACOMP_INPUT4 = SWM_PINENABLE0_ACOMP_I4_MASK,
kSWM_SWCLK = SWM_PINENABLE0_SWCLK_MASK,
kSWM_SWDIO = SWM_PINENABLE0_SWDIO_MASK,
kSWM_RESETN = SWM_PINENABLE0_RESETN_MASK,
kSWM_CLKIN = SWM_PINENABLE0_CLKIN_MASK,
kSWM_WKCLKIN = SWM_PINENABLE0_WKCLKIN_MASK,
kSWM_VDDCMP = SWM_PINENABLE0_VDDCMP_MASK,
kSWM_ADC_CHN0 = SWM_PINENABLE0_ADC_0_MASK,
kSWM_ADC_CHN1 = SWM_PINENABLE0_ADC_1_MASK,
kSWM_ADC_CHN2 = SWM_PINENABLE0_ADC_2_MASK,
kSWM_ADC_CHN3 = SWM_PINENABLE0_ADC_3_MASK,
kSWM_ADC_CHN4 = SWM_PINENABLE0_ADC_4_MASK,
kSWM_ADC_CHN5 = SWM_PINENABLE0_ADC_5_MASK,
kSWM_ADC_CHN6 = SWM_PINENABLE0_ADC_6_MASK,
kSWM_ADC_CHN7 = SWM_PINENABLE0_ADC_7_MASK,
kSWM_ADC_CHN8 = SWM_PINENABLE0_ADC_8_MASK,
kSWM_ADC_CHN9 = SWM_PINENABLE0_ADC_9_MASK,
kSWM_ADC_CHN10 = SWM_PINENABLE0_ADC_10_MASK,
kSWM_ADC_CHN11 = SWM_PINENABLE0_ADC_11_MASK,
kSWM_ACOMP_INPUT5 = SWM_PINENABLE0_ACOMP_I5_MASK,
kSWM_DAC_OUT0 = SWM_PINENABLE0_DACOUT0_MASK,
kSWM_FIXEDPIN_NUM_FUNCS = (int)0x80000001U }

```

*SWM fixed pin selection.*

- typedef enum  
[\\_swm\\_pinassignfixed\\_port\\_pin\\_type\\_t](#) [swm\\_fixed\\_port\\_pin\\_type\\_t](#)  
*SWM pinassignfixed\_port\_pin number.*
- typedef enum [\\_swm\\_port\\_pin\\_type\\_t](#) [swm\\_port\\_pin\\_type\\_t](#)  
*SWM port\_pin number.*
- typedef enum  
[\\_swm\\_pinassignfixed\\_select\\_movable\\_t](#) [swm\\_select\\_fixed\\_movable\\_t](#)  
*SWM pinassignfixed\_movable selection.*
- typedef enum [\\_swm\\_select\\_movable\\_t](#) [swm\\_select\\_movable\\_t](#)  
*SWM movable selection.*
- typedef enum  
[\\_swm\\_select\\_fixed\\_pin\\_t](#) [swm\\_select\\_fixed\\_pin\\_t](#)  
*SWM fixed pin selection.*

## Driver version

- #define [FSL\\_SWM\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 2))  
*LPC SWM driver version.*

## 23.3 Macro Definition Documentation

### 23.3.1 #define FSL\_SWM\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))

## 23.4 Enumeration Type Documentation

### 23.4.1 enum \_swm\_pinassignfixed\_port\_pin\_type\_t

Enumerator

*kSWM\_PLU\_INPUT0\_PortPin\_P0\_0* port\_pin number P0\_0.  
*kSWM\_PLU\_INPUT0\_PortPin\_P0\_8* port\_pin number P0\_8.  
*kSWM\_PLU\_INPUT0\_PortPin\_P0\_17* port\_pin number P0\_17.  
*kSWM\_PLU\_INPUT1\_PortPin\_P0\_1* port\_pin number P0\_1.  
*kSWM\_PLU\_INPUT1\_PortPin\_P0\_9* port\_pin number P0\_9.  
*kSWM\_PLU\_INPUT1\_PortPin\_P0\_18* port\_pin number P0\_18.  
*kSWM\_PLU\_INPUT2\_PortPin\_P0\_2* port\_pin number P0\_2.  
*kSWM\_PLU\_INPUT2\_PortPin\_P0\_10* port\_pin number P0\_10.  
*kSWM\_PLU\_INPUT2\_PortPin\_P0\_19* port\_pin number P0\_19.  
*kSWM\_PLU\_INPUT3\_PortPin\_P0\_3* port\_pin number P0\_3.  
*kSWM\_PLU\_INPUT3\_PortPin\_P0\_11* port\_pin number P0\_11.  
*kSWM\_PLU\_INPUT3\_PortPin\_P0\_20* port\_pin number P0\_20.  
*kSWM\_PLU\_INPUT4\_PortPin\_P0\_4* port\_pin number P0\_4.  
*kSWM\_PLU\_INPUT4\_PortPin\_P0\_12* port\_pin number P0\_12.  
*kSWM\_PLU\_INPUT4\_PortPin\_P0\_21* port\_pin number P0\_21.  
*kSWM\_PLU\_INPUT5\_PortPin\_P0\_5* port\_pin number P0\_5.  
*kSWM\_PLU\_INPUT5\_PortPin\_P0\_13* port\_pin number P0\_13.  
*kSWM\_PLU\_INPUT5\_PortPin\_P0\_22* port\_pin number P0\_22.  
*kSWM\_PLU\_OUT0\_PortPin\_P0\_7* port\_pin number P0\_7.  
*kSWM\_PLU\_OUT0\_PortPin\_P0\_14* port\_pin number P0\_14.  
*kSWM\_PLU\_OUT0\_PortPin\_P0\_23* port\_pin number P0\_23.  
*kSWM\_PLU\_OUT1\_PortPin\_P0\_8* port\_pin number P0\_8.  
*kSWM\_PLU\_OUT1\_PortPin\_P0\_15* port\_pin number P0\_15.  
*kSWM\_PLU\_OUT1\_PortPin\_P0\_24* port\_pin number P0\_24.  
*kSWM\_PLU\_OUT2\_PortPin\_P0\_9* port\_pin number P0\_9.  
*kSWM\_PLU\_OUT2\_PortPin\_P0\_16* port\_pin number P0\_16.  
*kSWM\_PLU\_OUT2\_PortPin\_P0\_25* port\_pin number P0\_25.  
*kSWM\_PLU\_OUT3\_PortPin\_P0\_10* port\_pin number P0\_10.  
*kSWM\_PLU\_OUT3\_PortPin\_P0\_17* port\_pin number P0\_17.  
*kSWM\_PLU\_OUT3\_PortPin\_P0\_26* port\_pin number P0\_26.  
*kSWM\_PLU\_OUT4\_PortPin\_P0\_11* port\_pin number P0\_11.  
*kSWM\_PLU\_OUT4\_PortPin\_P0\_18* port\_pin number P0\_18.  
*kSWM\_PLU\_OUT4\_PortPin\_P0\_27* port\_pin number P0\_27.  
*kSWM\_PLU\_OUT5\_PortPin\_P0\_12* port\_pin number P0\_12.  
*kSWM\_PLU\_OUT5\_PortPin\_P0\_19* port\_pin number P0\_19.  
*kSWM\_PLU\_OUT5\_PortPin\_P0\_28* port\_pin number P0\_28.

*kSWM\_PLU\_OUT6\_PortPin\_P0\_13* port\_pin number P0\_13.  
*kSWM\_PLU\_OUT6\_PortPin\_P0\_20* port\_pin number P0\_20.  
*kSWM\_PLU\_OUT6\_PortPin\_P0\_29* port\_pin number P0\_29.  
*kSWM\_PLU\_OUT7\_PortPin\_P0\_14* port\_pin number P0\_14.  
*kSWM\_PLU\_OUT7\_PortPin\_P0\_21* port\_pin number P0\_21.  
*kSWM\_PLU\_OUT7\_PortPin\_P0\_30* port\_pin number P0\_30.

### 23.4.2 enum\_swm\_port\_pin\_type\_t

Enumerator

*kSWM\_PortPin\_P0\_0* port\_pin number P0\_0.  
*kSWM\_PortPin\_P0\_1* port\_pin number P0\_1.  
*kSWM\_PortPin\_P0\_2* port\_pin number P0\_2.  
*kSWM\_PortPin\_P0\_3* port\_pin number P0\_3.  
*kSWM\_PortPin\_P0\_4* port\_pin number P0\_4.  
*kSWM\_PortPin\_P0\_5* port\_pin number P0\_5.  
*kSWM\_PortPin\_P0\_6* port\_pin number P0\_6.  
*kSWM\_PortPin\_P0\_7* port\_pin number P0\_7.  
*kSWM\_PortPin\_P0\_8* port\_pin number P0\_8.  
*kSWM\_PortPin\_P0\_9* port\_pin number P0\_9.  
*kSWM\_PortPin\_P0\_10* port\_pin number P0\_10.  
*kSWM\_PortPin\_P0\_11* port\_pin number P0\_11.  
*kSWM\_PortPin\_P0\_12* port\_pin number P0\_12.  
*kSWM\_PortPin\_P0\_13* port\_pin number P0\_13.  
*kSWM\_PortPin\_P0\_14* port\_pin number P0\_14.  
*kSWM\_PortPin\_P0\_15* port\_pin number P0\_15.  
*kSWM\_PortPin\_P0\_16* port\_pin number P0\_16.  
*kSWM\_PortPin\_P0\_17* port\_pin number P0\_17.  
*kSWM\_PortPin\_P0\_18* port\_pin number P0\_18.  
*kSWM\_PortPin\_P0\_19* port\_pin number P0\_19.  
*kSWM\_PortPin\_P0\_20* port\_pin number P0\_20.  
*kSWM\_PortPin\_P0\_21* port\_pin number P0\_21.  
*kSWM\_PortPin\_P0\_22* port\_pin number P0\_22.  
*kSWM\_PortPin\_P0\_23* port\_pin number P0\_23.  
*kSWM\_PortPin\_P0\_24* port\_pin number P0\_24.  
*kSWM\_PortPin\_P0\_25* port\_pin number P0\_25.  
*kSWM\_PortPin\_P0\_26* port\_pin number P0\_26.  
*kSWM\_PortPin\_P0\_27* port\_pin number P0\_27.  
*kSWM\_PortPin\_P0\_28* port\_pin number P0\_28.  
*kSWM\_PortPin\_P0\_29* port\_pin number P0\_29.  
*kSWM\_PortPin\_P0\_30* port\_pin number P0\_30.  
*kSWM\_PortPin\_P0\_31* port\_pin number P0\_31.  
*kSWM\_PortPin\_Reset* port\_pin reset number.

### 23.4.3 enum \_swm\_pinassignfixed\_select\_movable\_t

Enumerator

***kSWM\_PLU\_INPUT0*** Movable function as PLU\_INPUT0.  
***kSWM\_PLU\_INPUT1*** Movable function as PLU\_INPUT1.  
***kSWM\_PLU\_INPUT2*** Movable function as PLU\_INPUT2.  
***kSWM\_PLU\_INPUT3*** Movable function as PLU\_INPUT3.  
***kSWM\_PLU\_INPUT4*** Movable function as PLU\_INPUT4.  
***kSWM\_PLU\_INPUT5*** Movable function as PLU\_INPUT5.  
***kSWM\_PLU\_OUT0*** Movable function as PLU\_OUT0.  
***kSWM\_PLU\_OUT1*** Movable function as PLU\_OUT1.  
***kSWM\_PLU\_OUT2*** Movable function as PLU\_OUT2.  
***kSWM\_PLU\_OUT3*** Movable function as PLU\_OUT3.  
***kSWM\_PLU\_OUT4*** Movable function as PLU\_OUT4.  
***kSWM\_PLU\_OUT5*** Movable function as PLU\_OUT5.  
***kSWM\_PLU\_OUT6*** Movable function as PLU\_OUT6.  
***kSWM\_PLU\_OUT7*** Movable function as PLU\_OUT7.  
***kSWM\_PINASSINGNFIXED\_MOVABLE\_NUM\_FUNCS*** Movable function number.

### 23.4.4 enum \_swm\_select\_movable\_t

Enumerator

***kSWM\_USART0\_TXD*** Movable function as USART0\_TXD.  
***kSWM\_USART0\_RXD*** Movable function as USART0\_RXD.  
***kSWM\_USART0\_RTS*** Movable function as USART0\_RTS.  
***kSWM\_USART0\_CTS*** Movable function as USART0\_CTS.  
***kSWM\_USART0\_SCLK*** Movable function as USART0\_SCLK.  
***kSWM\_USART1\_TXD*** Movable function as USART1\_TXD.  
***kSWM\_USART1\_RXD*** Movable function as USART1\_RXD.  
***kSWM\_USART1\_SCLK*** Movable function as USART1\_SCLK.  
***kSWM\_SPI0\_SCK*** Movable function as SPI0\_SCK.  
***kSWM\_SPI0\_MOSI*** Movable function as SPI0\_MOSI.  
***kSWM\_SPI0\_MISO*** Movable function as SPI0\_MISO.  
***kSWM\_SPI0\_SSEL0*** Movable function as SPI0\_SSEL0.  
***kSWM\_SPI0\_SSEL1*** Movable function as SPI0\_SSEL1.  
***kSWM\_T0\_CAP\_CHN0*** Movable function as Timer Capture Channel 0.  
***kSWM\_T0\_CAP\_CHN1*** Movable function as Timer Capture Channel 1.  
***kSWM\_T0\_CAP\_CHN2*** Movable function as Timer Capture Channel 2.  
***kSWM\_T0\_MAT\_CHN0*** Movable function as Timer Match Channel 0.  
***kSWM\_T0\_MAT\_CHN1*** Movable function as Timer Match Channel 1.  
***kSWM\_T0\_MAT\_CHN2*** Movable function as Timer Match Channel 2.  
***kSWM\_T0\_MAT\_CHN3*** Movable function as Timer Match Channel 3.

*kSWM\_I2C0\_SDA* Movable function as I2C0\_SDA.  
*kSWM\_I2C0\_SCL* Movable function as I2C0\_SCL.  
*kSWM\_ACMP\_OUT* Movable function as ACMP\_OUT.  
*kSWM\_CLKOUT* Movable function as CLKOUT.  
*kSWM\_GPIO\_INT\_BMAT* Movable function as GPIO\_INT\_BMAT.  
*kSWM\_LVLSHFT\_IN0* Movable function as LVLSHFT\_IN0.  
*kSWM\_LVLSHFT\_IN1* Movable function as LVLSHFT\_IN1.  
*kSWM\_LVLSHFT\_OUT0* Movable function as LVLSHFT\_OUT0.  
*kSWM\_LVLSHFT\_OUT1* Movable function as LVLSHFT\_OUT1.  
*kSWM\_I2C1\_SDA* Movable function as I2C1\_SDA.  
*kSWM\_I2C1\_SCL* Movable function as I2C1\_SCL.  
*kSWM\_PLU\_CLKIN\_IN* Movable function as PLU\_CLKIN\_IN.  
*kSWM\_CAPT\_X0* Movable function as CAPT\_X0.  
*kSWM\_CAPT\_X1* Movable function as CAPT\_X1.  
*kSWM\_CAPT\_X2* Movable function as CAPT\_X2.  
*kSWM\_CAPT\_X3* Movable function as CAPT\_X3.  
*kSWM\_CAPT\_X4* Movable function as CAPT\_X4.  
*kSWM\_CAPT\_YL* Movable function as CAPT\_YL.  
*kSWM\_CAPT\_YH* Movable function as CAPT\_YH.  
*kSWM\_MOVABLE\_NUM\_FUNCS* Movable function number.

### 23.4.5 enum \_swm\_select\_fixed\_pin\_t

Enumerator

*kSWM\_ACMP\_INPUT1* Fixed-pin function as ACMP\_INPUT1.  
*kSWM\_ACMP\_INPUT2* Fixed-pin function as ACMP\_INPUT2.  
*kSWM\_ACMP\_INPUT3* Fixed-pin function as ACMP\_INPUT3.  
*kSWM\_ACMP\_INPUT4* Fixed-pin function as ACMP\_INPUT4.  
*kSWM\_SWCLK* Fixed-pin function as SWCLK.  
*kSWM\_SWDIO* Fixed-pin function as SWDIO.  
*kSWM\_RESETN* Fixed-pin function as RESETN.  
*kSWM\_CLKIN* Fixed-pin function as CLKIN.  
*kSWM\_WKCLKIN* Fixed-pin function as WKCLKIN.  
*kSWM\_VDDCMP* Fixed-pin function as VDDCMP.  
*kSWM\_ADC\_CHN0* Fixed-pin function as ADC\_CHN0.  
*kSWM\_ADC\_CHN1* Fixed-pin function as ADC\_CHN1.  
*kSWM\_ADC\_CHN2* Fixed-pin function as ADC\_CHN2.  
*kSWM\_ADC\_CHN3* Fixed-pin function as ADC\_CHN3.  
*kSWM\_ADC\_CHN4* Fixed-pin function as ADC\_CHN4.  
*kSWM\_ADC\_CHN5* Fixed-pin function as ADC\_CHN5.  
*kSWM\_ADC\_CHN6* Fixed-pin function as ADC\_CHN6.  
*kSWM\_ADC\_CHN7* Fixed-pin function as ADC\_CHN7.  
*kSWM\_ADC\_CHN8* Fixed-pin function as ADC\_CHN8.

*kSWM\_ADC\_CHN9* Fixed-pin function as ADC\_CHN9.  
*kSWM\_ADC\_CHN10* Fixed-pin function as ADC\_CHN10.  
*kSWM\_ADC\_CHN11* Fixed-pin function as ADC\_CHN11.  
*kSWM\_ACMP\_INPUT5* Fixed-pin function as ACMP\_INPUT5.  
*kSWM\_DAC\_OUT0* Fixed-pin function as DACOUT0.  
*kSWM\_FIXEDPIN\_NUM\_FUNCS* Fixed-pin function number.

## 23.5 Function Documentation

### 23.5.1 void SWM\_SetMovablePinSelect ( SWM\_Type \* *base*, swm\_select\_movable\_t *func*, swm\_port\_pin\_type\_t *swm\_port\_pin* )

This function will select a pin (designated by its GPIO port and bit numbers) to a function.

Parameters

|                     |  |
|---------------------|--|
| <i>base</i>         | SWM peripheral base address.                         |
| <i>func</i>         | any function name that is movable.                   |
| <i>swm_port_pin</i> | any pin which has a GPIO port number and bit number. |

### 23.5.2 void SWM\_SetFixedMovablePinSelect ( SWM\_Type \* *base*, swm\_select\_fixed\_movable\_t *func*, swm\_fixed\_port\_pin\_type\_t *swm\_port\_pin* )

This function will select a pin (designated by its GPIO port and bit numbers) to a function.

Parameters

|                     |  |
|---------------------|--|
| <i>base</i>         | SWM peripheral base address.                         |
| <i>func</i>         | any function name that is movable.                   |
| <i>swm_port_pin</i> | any pin which has a GPIO port number and bit number. |

### 23.5.3 void SWM\_SetFixedPinSelect ( SWM\_Type \* *base*, swm\_select\_fixed\_pin\_t *func*, bool *enable* )

This function will enable a fixed-pin function in PINENABLE0 or PINENABLE1.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | SWM peripheral base address.         |
| <i>func</i>   | any function name that is fixed pin. |
| <i>enable</i> | enable or disable.                   |

# Chapter 24

## SYSCON: System Configuration

### 24.1 Overview

The MCUXpresso SDK provides a peripheral clock and power driver for the SYSCON module of MCU-Xpresso SDK devices. For further details, see the corresponding chapter.

#### Files

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

#### Functions

- void [SYSCON\\_AttachSignal](#) (SYSCON\_Type \*base, uint32\_t index, [syscon\\_connection\\_t](#) connection)  
*Attaches a signal.*

#### Syscon multiplexing connections

- enum [\\_syscon\\_connection\\_t](#) { [kSYSCON\\_GpioPort0Pin0ToPintsel](#) = 0U + (PINTSEL\_ID << SYSCON\_SHIFT) }
- *SYSCON connections type.*
- typedef enum [\\_syscon\\_connection\\_t](#) [syscon\\_connection\\_t](#)  
*SYSCON connections type.*
- #define [PINTSEL\\_ID](#) 0x178U  
*Periphinmux IDs.*
- #define [SYSCON\\_SHIFT](#) 20U

#### Driver version

- #define [FSL\\_SYSON\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 1))  
*Group syscon driver version for SDK.*

### 24.2 Macro Definition Documentation

#### 24.2.1 #define FSL\_SYSON\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

Version 2.0.1.

## 24.3 Enumeration Type Documentation

### 24.3.1 enum \_syscon\_connection\_t

Enumerator

*kSYSCON\_GpioPort0Pin0ToPintsel* Pin Interrupt.

## 24.4 Function Documentation

### 24.4.1 void SYSCON\_AttachSignal ( SYSCON\_Type \* *base*, uint32\_t *index*, syscon\_connection\_t *connection* )

This function gates the SYSCON clock.

Parameters

|                   |   |
|-------------------|---|
| <i>base</i>       | Base address of the SYSCON peripheral.          |
| <i>index</i>      | Destination peripheral to attach the signal to. |
| <i>connection</i> | Selects connection.                             |

Return values

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

## Chapter 25

# WKT: Self-wake-up Timer

### 25.1 Overview

The MCUXpresso SDK provides a driver for the Self-wake-up Timer (WKT) of MCUXpresso SDK devices.

### 25.2 Function groups

The WKT driver supports operating the module as a time counter.

#### 25.2.1 Initialization and deinitialization

The function [WKT\\_Init\(\)](#) initializes the WKT with specified configurations. The function [WKT\\_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the WKT operating mode.

The function [WKT\\_Deinit\(\)](#) stops the WKT timers and disables the module clock.

#### 25.2.2 Read actual WKT counter value

The function [WKT\\_GetCounterValue\(\)](#) reads the current timer counting value. This function returns the real-time timer counting value, in a range from 0 to a timer period.

#### 25.2.3 Start and Stop timer operations

The function [WKT\\_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value, counts down to 0. When the timer reaches 0, it stops and generates a trigger pulse and sets the timeout interrupt flag.

The function [WKT\\_StopTimer\(\)](#) stops the timer counting.

#### 25.2.4 Status

Provides functions to get and clear the WKT status flags.

## 25.3 Typical use case

### 25.3.1 WKT tick example

Updates the WKT period and toggles an LED periodically. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/wkt`

#### Files

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

#### Data Structures

- struct [\\_wkt\\_config](#)  
*Describes WKT configuration structure. [More...](#)*

#### Typedefs

- typedef enum [\\_wkt\\_clock\\_source](#) [wkt\\_clock\\_source\\_t](#)  
*Describes WKT clock source.*
- typedef struct [\\_wkt\\_config](#) [wkt\\_config\\_t](#)  
*Describes WKT configuration structure.*
- typedef enum [\\_wkt\\_status\\_flags](#) [wkt\\_status\\_flags\\_t](#)  
*List of WKT flags.*

#### Enumerations

- enum [\\_wkt\\_clock\\_source](#) {  
[kWKT\\_DividedFROClockSource](#) = 0U,  
[kWKT\\_LowPowerClockSource](#) = 1U,  
[kWKT\\_ExternalClockSource](#) = 2U }  
*Describes WKT clock source.*
- enum [\\_wkt\\_status\\_flags](#) { [kWKT\\_AlarmFlag](#) = WKT\_CTRL\_ALARMFLAG\_MASK }  
*List of WKT flags.*

#### Driver version

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

#### Initialization and deinitialization

- void [WKT\\_Init](#) ([WKT\\_Type](#) \*base, const [wkt\\_config\\_t](#) \*config)  
*Ungates the WKT clock and configures the peripheral for basic operation.*
- void [WKT\\_Deinit](#) ([WKT\\_Type](#) \*base)  
*Gate the WKT clock.*
- static void [WKT\\_GetDefaultConfig](#) ([wkt\\_config\\_t](#) \*config)  
*Initializes the WKT configuration structure.*

## Read the counter value.

- static uint32\_t [WKT\\_GetCounterValue](#) (WKT\_Type \*base)  
*Read actual WKT counter value.*

## Status Interface

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

## Timer Start and Stop

- static void [WKT\\_StartTimer](#) (WKT\_Type \*base, uint32\_t count)  
*Starts the timer counting.*
- static void [WKT\\_StopTimer](#) (WKT\_Type \*base)  
*Stops the timer counting.*

## 25.4 Data Structure Documentation

### 25.4.1 struct \_wkt\_config

#### Data Fields

- [wkt\\_clock\\_source\\_t](#) clockSource  
*External or internal clock source select.*

## 25.5 Typedef Documentation

### 25.5.1 typedef enum \_wkt\_clock\_source wkt\_clock\_source\_t

### 25.5.2 typedef struct \_wkt\_config wkt\_config\_t

## 25.6 Enumeration Type Documentation

### 25.6.1 enum \_wkt\_clock\_source

#### Enumerator

- kWKT\_DividedFROClockSource* WKT clock sourced from the divided FRO clock.
- kWKT\_LowPowerClockSource* WKT clock sourced from the Low power clock Use this clock, LP-OSCN bit of DPDCTRL register must be enabled.
- kWKT\_ExternalClockSource* WKT clock sourced from the Low power clock Use this clock, WA-KECLKPAD\_DISABLE bit of DPDCTRL register must be enabled.

## 25.6.2 enum \_wkt\_status\_flags

Enumerator

*kWKT\_AlarmFlag* Alarm flag.

## 25.7 Function Documentation

### 25.7.1 void WKT\_Init ( WKT\_Type \* *base*, const wkt\_config\_t \* *config* )

Note

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

Parameters

|               |   |
|---------------|---|
| <i>base</i>   | WKT peripheral base address             |
| <i>config</i> | Pointer to user's WKT config structure. |

### 25.7.2 void WKT\_Deinit ( WKT\_Type \* *base* )

Parameters

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

### 25.7.3 static void WKT\_GetDefaultConfig ( wkt\_config\_t \* *config* ) [inline], [static]

This function initializes the WKT configuration structure to default values. The default values are as follows.

```
* config->clockSource = kWKT_DividedFROClockSource;
*
```

Parameters

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

See Also

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

**25.7.4 static uint32\_t WKT\_GetCounterValue ( WKT\_Type \* *base* ) [inline], [static]**

Parameters

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

**25.7.5 static uint32\_t WKT\_GetStatusFlags ( WKT\_Type \* *base* ) [inline], [static]**

Parameters

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

Returns

The status flags. This is the logical OR of members of the enumeration [wkt\\_status\\_flags\\_t](#)

**25.7.6 static void WKT\_ClearStatusFlags ( WKT\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

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

**25.7.7 static void WKT\_StartTimer ( WKT\_Type \* *base*, uint32\_t *count* ) [inline], [static]**

After calling this function, timer loads a count value, counts down to 0, then stops.

## Note

User can call the utility macros provided in `fsl_common.h` to convert to ticks Do not write to Counter register while the counting is in progress

## Parameters

|              |  |
|--------------|--|
| <i>base</i>  | WKT peripheral base address.                       |
| <i>count</i> | The value to be loaded into the WKT Count register |

### 25.7.8 static void WKT\_StopTimer ( WKT\_Type \* *base* ) [inline], [static]

This function Clears the counter and stops the timer from counting.

## Parameters

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

## Chapter 26

# WWDT: Windowed Watchdog Timer Driver

### 26.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

### 26.2 Function groups

#### 26.2.1 Initialization and deinitialization

The function [WWDT\\_Init\(\)](#) initializes the watchdog timer with specified configurations. The configurations include timeout value and whether to enable watchdog after init. The function [WWDT\\_GetDefaultConfig\(\)](#) gets the default configurations.

The function [WWDT\\_Deinit\(\)](#) disables the watchdog and the module clock.

#### 26.2.2 Status

Provides functions to get and clear the WWDT status.

#### 26.2.3 Interrupt

Provides functions to enable/disable WWDT interrupts and get current enabled interrupts.

#### 26.2.4 Watch dog Refresh

The function [WWDT\\_Refresh\(\)](#) feeds the WWDT.

### 26.3 Typical use case

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

#### Files

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

#### Data Structures

- struct [\\_wwdt\\_config](#)  
*Describes WWDT configuration structure. [More...](#)*

## Typedefs

- typedef struct `_wwdt_config wwdt_config_t`  
*Describes WWDT configuration structure.*

## Enumerations

- enum `_wwdt_status_flags_t` {  
`kWWDT_TimeoutFlag = WWDT_MOD_WDTOF_MASK,`  
`kWWDT_WarningFlag = WWDT_MOD_WDINT_MASK }`  
*WWDT status flags.*

## Driver version

- #define `FSL_WWDT_DRIVER_VERSION (MAKE_VERSION(2, 1, 9))`  
*Defines WWDT driver version.*

## Refresh sequence

- #define `WWDT_FIRST_WORD_OF_REFRESH (0xAAU)`  
*First word of refresh sequence.*
- #define `WWDT_SECOND_WORD_OF_REFRESH (0x55U)`  
*Second word of refresh sequence.*

## WWDT Initialization and De-initialization

- void `WWDT_GetDefaultConfig (wwdt_config_t *config)`  
*Initializes WWDT configure structure.*
- void `WWDT_Init (WWDT_Type *base, const wwdt_config_t *config)`  
*Initializes the WWDT.*
- void `WWDT_Deinit (WWDT_Type *base)`  
*Shuts down the WWDT.*

## WWDT Functional Operation

- static void `WWDT_Enable (WWDT_Type *base)`  
*Enables the WWDT module.*
- static void `WWDT_Disable (WWDT_Type *base)`  
*Disables the WWDT module.*
- static uint32\_t `WWDT_GetStatusFlags (WWDT_Type *base)`  
*Gets all WWDT status flags.*
- void `WWDT_ClearStatusFlags (WWDT_Type *base, uint32_t mask)`  
*Clear WWDT flag.*
- static void `WWDT_SetWarningValue (WWDT_Type *base, uint32_t warningValue)`  
*Set the WWDT warning value.*
- static void `WWDT_SetTimeoutValue (WWDT_Type *base, uint32_t timeoutCount)`  
*Set the WWDT timeout value.*
- static void `WWDT_SetWindowValue (WWDT_Type *base, uint32_t windowValue)`  
*Sets the WWDT window value.*
- void `WWDT_Refresh (WWDT_Type *base)`  
*Refreshes the WWDT timer.*

## 26.4 Data Structure Documentation

### 26.4.1 struct \_wwdt\_config

#### Data Fields

- bool [enableWwdt](#)  
*Enables or disables WWDT.*
- bool [enableWatchdogReset](#)  
*true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset*
- bool [enableWatchdogProtect](#)  
*true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time*
- bool [enableLockOscillator](#)  
*true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator*
- uint32\_t [windowValue](#)  
*Window value, set this to 0xFFFFFFFF if windowing is not in effect.*
- uint32\_t [timeoutValue](#)  
*Timeout value.*
- uint32\_t [warningValue](#)  
*Watchdog time counter value that will generate a warning interrupt.*
- uint32\_t [clockFreq\\_Hz](#)  
*Watchdog clock source frequency.*

#### Field Documentation

##### (1) uint32\_t \_wwdt\_config::warningValue

Set this to 0 for no warning

##### (2) uint32\_t \_wwdt\_config::clockFreq\_Hz

## 26.5 Macro Definition Documentation

### 26.5.1 #define FSL\_WWDT\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 9))

## 26.6 Typedef Documentation

### 26.6.1 typedef struct \_wwdt\_config wwdt\_config\_t

## 26.7 Enumeration Type Documentation

### 26.7.1 enum \_wwdt\_status\_flags\_t

This structure contains the WWDT status flags for use in the WWDT functions.

Enumerator

*kWWDT\_TimeoutFlag* Time-out flag, set when the timer times out.

*kWWDT\_WarningFlag* Warning interrupt flag, set when timer is below the value WDWARNINT.

## 26.8 Function Documentation

### 26.8.1 void WWDT\_GetDefaultConfig ( wwdt\_config\_t \* config )

This function initializes the WWDT configure structure to default value. The default value are:

```
* config->enableWwdt = true;
* config->enableWatchdogReset = false;
* config->enableWatchdogProtect = false;
* config->enableLockOscillator = false;
* config->windowValue = 0xFFFFFU;
* config->timeoutValue = 0xFFFFFU;
* config->warningValue = 0;
*
```

#### Parameters

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

#### See Also

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

### 26.8.2 void WWDT\_Init ( WWDT\_Type \* base, const wwdt\_config\_t \* config )

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

#### Example:

```
* wwdt_config_t config;
* WWDT_GetDefaultConfig(&config);
* config.timeoutValue = 0x7ffU;
* WWDT_Init(wwdt_base, &config);
*
```

#### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | WWDT peripheral base address |
| <i>config</i> | The configuration of WWDT    |

### 26.8.3 void WWDT\_Deinit ( WWDT\_Type \* base )

This function shuts down the WWDT.

Parameters

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

#### 26.8.4 static void WWDT\_Enable ( WWDT\_Type \* *base* ) [inline], [static]

This function write value into WWDT\_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

Parameters

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

#### 26.8.5 static void WWDT\_Disable ( WWDT\_Type \* *base* ) [inline], [static]

**Deprecated** Do not use this function. It will be deleted in next release version, for once the bit field of WDEN written with a 1, it can not be re-written with a 0.

This function write value into WWDT\_MOD register to disable the WWDT.

Parameters

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

#### 26.8.6 static uint32\_t WWDT\_GetStatusFlags ( WWDT\_Type \* *base* ) [inline], [static]

This function gets all status flags.

Example for getting Timeout Flag:

```
* uint32_t status;
* status = WWDT_GetStatusFlags(wwdt_base) &
*     kWWDT_TimeoutFlag;
*
```

## Parameters

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

## Returns

The status flags. This is the logical OR of members of the enumeration [\\_wwdt\\_status\\_flags\\_t](#)

### 26.8.7 void WWDT\_ClearStatusFlags ( WWDT\_Type \* *base*, uint32\_t *mask* )

This function clears WWDT status flag.

Example for clearing warning flag:

```
* WWDT_ClearStatusFlags(wwdt_base, kWWDTClearWarningFlag);
*
```

## Parameters

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

### 26.8.8 static void WWDT\_SetWarningValue ( WWDT\_Type \* *base*, uint32\_t *warningValue* ) [inline], [static]

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

## Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>base</i>         | WWDT peripheral base address |
| <i>warningValue</i> | WWDT warning value.          |

### 26.8.9 static void WWDT\_SetTimeoutValue ( WWDT\_Type \* *base*, uint32\_t *timeoutCount* ) [inline], [static]

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC

register. Thus the minimum time-out interval is  $TWDCLK * 256 * 4$ . If `enableWatchdogProtect` flag is true in `wwdt_config_t` config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the `WDTOF` flag.

Parameters

|                     |   |
|---------------------|---|
| <i>base</i>         | WWDT peripheral base address                  |
| <i>timeoutCount</i> | WWDT timeout value, count of WWDT clock tick. |

### 26.8.10 static void WWDT\_SetWindowValue ( WWDT\_Type \* *base*, uint32\_t *windowValue* ) [*inline*], [*static*]

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set *windowValue* to 0xFFFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | WWDT peripheral base address |
| <i>windowValue</i> | WWDT window value.           |

### 26.8.11 void WWDT\_Refresh ( WWDT\_Type \* *base* )

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

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

# Chapter 27

## Debug Console Lite

### 27.1 Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data.

### 27.2 Function groups

#### 27.2.1 Initialization

To initialize the debug console, call the [DbgConsole\\_Init\(\)](#) function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate, serial_port_type_t
    device, uint32_t clkSrcFreq);
```

Selects the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
    kSerialPort_None = 0U,
    kSerialPort_Uart = 1U,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. The debug console state is stored in the `debug_console_state_t` structure, such as shown here.

```
typedef struct DebugConsoleState
{
    uint8_t uartHandleBuffer[HAL_UART_HANDLE_SIZE];
    hal_uart_status_t (*putChar)(hal_uart_handle_t handle, const uint8_t *data, size_t length);
    hal_uart_status_t (*getChar)(hal_uart_handle_t handle, uint8_t *data, size_t length);
    serial_port_type_t type;
} debug_console_state_t;
```

This example shows how to call the [DbgConsole\\_Init\(\)](#) given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_USART_INSTANCE, BOARD_DEBUG_USART_BAUDRATE,
    BOARD_DEBUG_USART_TYPE,
    BOARD_DEBUG_USART_CLK_FREQ);
```

## 27.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype "`%[flags][width][.precision][length]specifier`", which is explained below

| flags   | Description   |
|---------|---|
| -       | Left-justified within the given field width. Right-justified is the default.  |
| +       | Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.  |
| (space) | If no sign is written, a blank space is inserted before the value.  |
| #       | Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed. |
| 0       | Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).   |

| Width    | Description  |
|----------|--|
| (number) | A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger. |
| *        | The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.  |

| <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                              |

| specifier | Description |
|-----------|-------------|
|-----------|-------------|

- Support a format specifier for SCANF following this prototype " %[\*][width][length]specifier", which is explained below

| * | Description  |
|---|--|
|   | An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument. |

| width | Description  |
|-------|--|
|       | This specifies the maximum number of characters to be read in the current reading operation. |

| length      | Description   |
|-------------|---|
| hh          | The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).   |
| h           | The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).  |
| l           | The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.           |
| ll          | The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s. |
| L           | The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).  |
| j or z or t | Not supported   |

| specifier              | Qualifying Input   | Type of argument |
|------------------------|--|------------------|
| c                      | Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end. | char *           |
| i                      | Integer: : Number optionally preceded with a + or - sign   | int *            |
| d                      | Decimal integer: Number optionally preceded with a + or - sign   | int *            |
| a, A, e, E, f, F, g, G | Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4                      | float *          |
| o                      | Octal Integer:   | int *            |
| s                      | String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).   | char *           |
| u                      | Unsigned decimal integer.  | unsigned int *   |

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE == DEBUGCONSOLE_DISABLE /* Disable debug console */
#define PRINTF
#define SCANF
#define PUTCHAR
#define GETCHAR
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_SDK /* Select printf, scanf, putchar, getchar of SDK
```

```

        version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#elif SDK_DEBUGCONSOLE == DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN /* Select printf, scanf, putchar, getchar of
    toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */

```

### 27.2.3 SDK\_DEBUGCONSOLE and SDK\_DEBUGCONSOLE\_UART

There are two macros `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` added to configure `PRINTF` and low level output peripheral.

- The macro `SDK_DEBUGCONSOLE` is used for frontend. Whether debug console redirect to toolchain or SDK or disabled, it decides which is the frontend of the debug console, Tool chain or SDK. The function can be set by the macro `SDK_DEBUGCONSOLE`.
- The macro `SDK_DEBUGCONSOLE_UART` is used for backend. It is used to decide whether provide low level IO implementation to toolchain `printf` and `scanf`. For example, within MCU-Xpresso, if the macro `SDK_DEBUGCONSOLE_UART` is defined, `__sys_write` and `__sys_readc` will be used when `__REDLIB__` is defined; `_write` and `_read` will be used in other cases. The macro does not specifically refer to the peripheral "UART". It refers to the external peripheral UART. So if the macro `SDK_DEBUGCONSOLE_UART` is not defined when tool-chain `printf` is calling, the semihosting will be used.

The following matrix shows the effects of `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` on `PRINTF` and `printf`. The green mark is the default setting of the debug console.

| <code>SDK_DEBUGCONSOLE</code>                    | <code>SDK_DEBUGCONSOLE_UART</code> | <code>PRINTF</code> | <code>printf</code> |
|--|------------------------------------|---------------------|---------------------|
| <code>DEBUGCONSOLE_-REDIRECT_TO_SDK</code>       | defined                            | UART                | UART                |
| <code>DEBUGCONSOLE_-REDIRECT_TO_SDK</code>       | undefined                          | UART                | semihost            |
| <code>DEBUGCONSOLE_-REDIRECT_TO_TOOLCHAIN</code> | defined                            | UART                | UART                |
| <code>DEBUGCONSOLE_-REDIRECT_TO_TOOLCHAIN</code> | undefined                          | semihost            | semihost            |
| <code>DEBUGCONSOLE_-DISABLE</code>               | defined                            | No output           | UART                |
| <code>DEBUGCONSOLE_-DISABLE</code>               | undefined                          | No output           | semihost            |

|                  |                       |        |        |
|------------------|-----------------------|--------|--------|
| SDK_DEBUGCONSOLE | SDK_DEBUGCONSOLE_UART | PRINTF | printf |
|------------------|-----------------------|--------|--------|

## 27.3 Typical use case

### Some examples use the PUTCHAR & GETCHAR function

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

### Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalent 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\r\nTime: %u ticks %2.5f milliseconds\r\n\r\nDONE\r\n", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

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

### Print out failure messages using MCUXpresso SDK `__assert_func`:

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

#### Note:

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

## Modules

- [Semihosting](#)

## Macros

- `#define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U`  
*Definition select redirect toolchain printf, scanf to uart or not.*
- `#define DEBUGCONSOLE_REDIRECT_TO_SDK 1U`  
*Select SDK version printf, scanf.*
- `#define DEBUGCONSOLE_DISABLE 2U`  
*Disable debugconsole function.*
- `#define SDK_DEBUGCONSOLE DEBUGCONSOLE_REDIRECT_TO_SDK`  
*Definition to select sdk or toolchain printf, scanf.*
- `#define PRINTF_FLOAT_ENABLE 0U`  
*Definition to printf the float number.*
- `#define SCANF_FLOAT_ENABLE 0U`  
*Definition to scanf the float number.*
- `#define PRINTF_ADVANCED_ENABLE 0U`  
*Definition to support advanced format specifier for printf.*
- `#define SCANF_ADVANCED_ENABLE 0U`  
*Definition to support advanced format specifier for scanf.*
- `#define PRINTF DbgConsole_Printf`  
*Definition to select redirect toolchain printf, scanf to uart or not.*

## Initialization

- `status_t DbgConsole_Init` (uint8\_t instance, uint32\_t baudRate, serial\_port\_type\_t device, uint32\_t clkSrcFreq)  
*Initializes the peripheral used for debug messages.*
- `status_t DbgConsole_Deinit` (void)  
*De-initializes the peripheral used for debug messages.*
- `status_t DbgConsole_EnterLowpower` (void)  
*Prepares to enter low power consumption.*
- `status_t DbgConsole_ExitLowpower` (void)  
*Restores from low power consumption.*
- `int DbgConsole_Printf` (const char \*fmt\_s,...)  
*Writes formatted output to the standard output stream.*
- `int DbgConsole_Vprintf` (const char \*fmt\_s, va\_list formatStringArg)  
*Writes formatted output to the standard output stream.*
- `int DbgConsole_Putchar` (int dbgConsoleCh)  
*Writes a character to stdout.*
- `int DbgConsole_Scanf` (char \*fmt\_s,...)  
*Reads formatted data from the standard input stream.*
- `int DbgConsole_Getchar` (void)  
*Reads a character from standard input.*

## 27.4 Macro Definition Documentation

### 27.4.1 #define DEBUGCONSOLE\_REDIRECT\_TO\_TOOLCHAIN 0U

Select toolchain printf and scanf.

### 27.4.2 #define DEBUGCONSOLE\_REDIRECT\_TO\_SDK 1U

### 27.4.3 #define DEBUGCONSOLE\_DISABLE 2U

### 27.4.4 #define SDK\_DEBUGCONSOLE DEBUGCONSOLE\_REDIRECT\_TO\_SDK

### 27.4.5 #define PRINTF\_FLOAT\_ENABLE 0U

### 27.4.6 #define SCANF\_FLOAT\_ENABLE 0U

### 27.4.7 #define PRINTF\_ADVANCED\_ENABLE 0U

### 27.4.8 #define SCANF\_ADVANCED\_ENABLE 0U

### 27.4.9 #define PRINTF DbgConsole\_Printf

if SDK\_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK\_DEBUGCONSOLE defined to 1,it represents select SDK version printf, scanf. if SDK\_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

## 27.5 Function Documentation

### 27.5.1 status\_t DbgConsole\_Init ( uint8\_t *instance*, uint32\_t *baudRate*, serial\_port\_type\_t *device*, uint32\_t *clkSrcFreq* )

Call this function to enable debug log messages to be output via the specified peripheral, frequency of peripheral source clock, and base address at the specified baud rate. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

---

|                   |   |
|-------------------|---|
| <i>instance</i>   | The instance of the module.If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UART adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART peripheral is LPUART1. If the uart_adapter.c is added to the current project, the UART peripheral is UART1. |
| <i>baudRate</i>   | The desired baud rate in bits per second.   |
| <i>device</i>     | Low level device type for the debug console, can be one of the following. <ul style="list-style-type: none"> <li>• kSerialPort_Uart.</li> </ul>   |
| <i>clkSrcFreq</i> | Frequency of peripheral source clock.   |

Returns

Indicates whether initialization was successful or not.

Return values

|                        |                        |
|------------------------|------------------------|
| <i>kStatus_Success</i> | Execution successfully |
| <i>kStatus_Fail</i>    | Execution failure      |

### 27.5.2 status\_t DbgConsole\_Deinit ( void )

Call this function to disable debug log messages to be output via the specified peripheral base address and at the specified baud rate.

Returns

Indicates whether de-initialization was successful or not.

### 27.5.3 status\_t DbgConsole\_EnterLowpower ( void )

This function is used to prepare to enter low power consumption.

Returns

Indicates whether de-initialization was successful or not.

**27.5.4 status\_t DbgConsole\_ExitLowpower ( void )**

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

**27.5.5 int DbgConsole\_Printf ( const char \* *fmt\_s*, ... )**

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

Parameters

|              |                        |
|--------------|------------------------|
| <i>fmt_s</i> | Format control string. |
|--------------|------------------------|

Returns

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

**27.5.6 int DbgConsole\_Vprintf ( const char \* *fmt\_s*, va\_list *formatStringArg* )**

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

Parameters

|                         |                        |
|-------------------------|------------------------|
| <i>fmt_s</i>            | Format control string. |
| <i>formatString-Arg</i> | Format arguments.      |

Returns

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

**27.5.7 int DbgConsole\_Putchar ( int *dbgConsoleCh* )**

Call this function to write a character to stdout.

## Parameters

|                     |                          |
|---------------------|--------------------------|
| <i>dbgConsoleCh</i> | Character to be written. |
|---------------------|--------------------------|

## Returns

Returns the character written.

**27.5.8 int DbgConsole\_Scanf ( char \* *fmt\_s*, ... )**

Call this function to read formatted data from the standard input stream.

## Parameters

|              |                        |
|--------------|------------------------|
| <i>fmt_s</i> | Format control string. |
|--------------|------------------------|

## Returns

Returns the number of fields successfully converted and assigned.

**27.5.9 int DbgConsole\_Getchar ( void )**

Call this function to read a character from standard input.

## Returns

Returns the character read.

## 27.6 Semihosting

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

### 27.6.1 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 ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

#### Step 2: Building the project

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

#### Step 3: Starting semihosting

1. Choose "Semihosting\_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting. Please Make sure the `SDK_DEBUGCONSOLE_UART` is not defined in project settings.
4. Start the project by choosing Project>Download and Debug.
5. Choose View>Terminal I/O to display the output from the I/O operations.

### 27.6.2 Guide Semihosting for Keil $\mu$ Vision

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

### 27.6.3 Guide Semihosting for MCUXpresso IDE

#### Step 1: Setting up the environment

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

#### Step 2: Building the project

1. Compile and link the project.

#### Step 3: Starting semihosting

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

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

### 27.6.4 Guide Semihosting for ARMGCC

#### Step 1: Setting up the environment

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

#### Add to "CMakeLists.txt"

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

**Step 2: Building the project**

1. Change "CMakeLists.txt":

**Change** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=nano.specs")"

**to** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=rdimon.specs")"

**Replace paragraph**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-common")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffunction-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fdata-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffreestanding")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-builtin")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mthumb")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mapcs")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --gc-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -static")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -z")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} muldefs")

**To**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --specs=rdimon.specs ")

**Remove**

target\_link\_libraries(semihosting\_ARMGCC.elf debug nosys)

2. Run "build\_debug.bat" to build project

### Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\trkr64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

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

**How to Reach Us:**

**Home Page:**

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

**Web Support:**

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

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

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions).

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, Processor Expert, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, Mbed, Mbed Enabled, and Vision are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2021 NXP B.V.

