

EFM32PG22

- BURTC - Backup RTC
 - BURTC_Init_TypeDef
- BUS - Bitfield Read/Write
- CHIP - Chip Errata Workarounds
- CMU - Clock Management Unit
 - CMU_LFXOInit_TypeDef
 - CMU_HFXOInit_TypeDef
 - CMU_DPLLInit_TypeDef
- CORE - Core Interrupt
 - dwt_cycle_counter_handle_t
 - CORE_nvicMask_t
- DBG - Debug
- EMU - Energy Management Unit
 - EMU_EM01Init_TypeDef
 - EMU_EM23Init_TypeDef
 - EMU_EM4Init_TypeDef
 - EMU_DCDCInit_TypeDef
- EUSART - Extended USART
 - EUSART_AdvancedInit_TypeDef
 - EUSART_UartInit_TypeDef
 - EUSART_IrDAInit_TypeDef
 - EUSART_PrsTriggerInit_TypeDef
 - EUSART_DaliInit_TypeDef
- Emlib
 - IADC - Incremental ADC
 - IADC_Init_t
 - IADC_Config_t
 - IADC_AllConfigs_t
 - IADC_InitScan_t
 - IADC_InitSingle_t
 - IADC_SingleInput_t
 - IADC_ScanTableEntry_t
 - IADC_ScanTable_t
 - IADC_Result_t
 - GPCRC - General Purpose CRC
 - GPCRC_Init_TypeDef
 - GPIO - General Purpose Input/Output
 - I2C - Inter-Integrated Circuit
 - I2C_Init_TypeDef
 - I2C_TransferSeq_TypeDef
 - LDMA - Linked DMA
 - LDMA_Descriptor_t
 - LDMA_Init_t

- LDMA_TransferCfg_t
- LETIMER - Low Energy Timer
 - LETIMER_Init_TypeDef
- MSC - Memory System Controller
 - MSC_ExecConfig_TypeDef
 - MSC_EccConfig_TypeDef
- PDM - Pulse Density Modulation
 - PDM_Init_TypeDef
- PRS - Peripheral Reflex System
- RAMFUNC - RAM Function Support
- RMU - Reset Management Unit
- RTCC - Real Timer Counter/Calendar
 - RTCC_Init_TypeDef
 - RTCC_CCChConf_TypeDef
- SE - Secure Element
 - SE_DataTransfer_t
 - SE_Command_t
 - Deprecated Functions
 - SE_OTPinit_t
 - SE_DebugStatus_t
 - SE_Status_t
- SMU - Security Management Unit
 - SMU_PrivilegedAccess_TypeDef
 - SMU_Init_TypeDef
- SYSTEM - System Utils
 - SYSTEM_ChipRevision_TypeDef
 - SYSTEM_CalAddrVal_TypeDef
- TIMER - Timer/Counter
 - TIMER_Init_TypeDef
 - TIMER_InitCC_TypeDef
 - TIMER_InitDTI_TypeDef
- USART - Synchronous/Asynchronous Serial
 - USART_InitAsync_TypeDef
 - USART_PrsTriggerInit_TypeDef
 - USART_InitSync_TypeDef
 - USART_InitIrDA_TypeDef
 - USART_InitI2s_TypeDef
- VERSION - Version Defines
- WDOG - Watchdog
 - WDOG_Init_TypeDef
- API Documentation

BURTC - Backup RTC

BURTC - Backup RTC

Backup Real Time Counter (BURTC) Peripheral API.

This module contains functions to control the BURTC peripheral of Silicon Labs 32-bit MCUs. The Backup Real Time Counter allows timekeeping in all energy modes. The Backup RTC is also available when the system is in backup mode.

Modules

[BURTC_Init_TypeDef](#)

Functions

void	BURTC_Init (const BURTC_Init_TypeDef *burtcInit) Initialize BURTC.
void	BURTC_Enable (bool enable) Enable or Disable BURTC peripheral.
void	BURTC_CompareSet (unsigned int comp, uint32_t value) Set BURTC compare channel.
uint32_t	BURTC_CompareGet (unsigned int comp) Get the BURTC compare value.
void	BURTC_CounterReset (void) Reset counter.
void	BURTC_Reset (void) Restore BURTC to reset state.
void	BURTC_IntClear (uint32_t flags) Clear one or more pending BURTC interrupts.
void	BURTC_IntDisable (uint32_t flags) Disable one or more BURTC interrupts.
void	BURTC_IntEnable (uint32_t flags) Enable one or more BURTC interrupts.
uint32_t	BURTC_IntGet (void) Get pending BURTC interrupt flags.
uint32_t	BURTC_IntGetEnabled (void) Get enabled and pending BURTC interrupt flags.
void	BURTC_IntSet (uint32_t flags) Set one or more pending BURTC interrupts from SW.
uint32_t	BURTC_Status (void) Status of BURTC RAM, timestamp and LP Mode.
void	BURTC_SyncWait (void) Wait for the BURTC to complete all synchronization of register changes and commands.
void	BURTC_Start (void) Start BURTC counter.
void	BURTC_Stop (void)

Stop the BURTC counter.

- uint32_t

BURTC_CounterGet(void)
Get BURTC counter.
- void

BURTC_Lock(void)
Lock BURTC registers, which will protect from writing new config settings.
- void

BURTC_Unlock(void)
Unlock BURTC registers, which will enable write access to change configuration.

Macros

- #define

burtcClkDiv_1 1
BURTC clock divisors.
- #define

burtcClkDiv_2 2
Divide clock by 2.
- #define

burtcClkDiv_4 4
Divide clock by 4.
- #define

burtcClkDiv_8 8
Divide clock by 8.
- #define

burtcClkDiv_16 16
Divide clock by 16.
- #define

burtcClkDiv_32 32
Divide clock by 32.
- #define

burtcClkDiv_64 64
Divide clock by 64.
- #define

burtcClkDiv_128 128
Divide clock by 128.
- #define

BURTC_INIT_DEFAULT undefined
Default configuration for BURTC init structure.

Function Documentation

BURTC_Init

```
void BURTC_Init (const BURTC_Init_TypeDef * burtcInit)
```

Initialize BURTC.

Parameters

Type	Direction	Argument Name	Description
const BURTC_Init_TypeDef *	[in]	burtcInit	A pointer to the BURTC initialization structure.

Configures the BURTC peripheral.

Note

- Before initialization, BURTC module must first be enabled by clearing the reset bit in the RMU, i.e.,

```
* RMU_ResetControl(rmuResetBU, rmuResetModeClear);  
*
```

Compare channel 0 must be configured outside this function, before initialization if enable is set to true. The counter will always be reset.

BURTC_Enable

```
void BURTC_Enable (bool enable)
```

Enable or Disable BURTC peripheral.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	true to enable, false to disable.

BURTC_CompareSet

```
void BURTC_CompareSet (unsigned int comp, uint32_t value)
```

Set BURTC compare channel.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	comp	Compare the channel index, must be 0 for current devices.
uint32_t	[in]	value	New compare value.

BURTC_CompareGet

```
uint32_t BURTC_CompareGet (unsigned int comp)
```

Get the BURTC compare value.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	comp	Compare the channel index value, must be 0 for Giant/Leopard Gecko.

Returns

- The currently configured value for this compare channel.

BURTC_CounterReset

```
void BURTC_CounterReset (void )
```

Reset counter.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

BURTC_Reset

```
void BURTC_Reset (void )
```

Restore BURTC to reset state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Before accessing the BURTC, BURSTEN in RMU->CTRL must be cleared. LOCK will not be reset to default value, as this will disable access to core BURTC registers.

BURTC_IntClear

```
void BURTC_IntClear (uint32_t flags)
```

Clear one or more pending BURTC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	BURTC interrupt sources to clear. Use a set of interrupt flags OR-ed together to clear multiple interrupt sources for the BURTC module (BURTC_IFS_nnn).

BURTC_IntDisable

```
void BURTC_IntDisable (uint32_t flags)
```

Disable one or more BURTC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	BURTC interrupt sources to disable. Use a set of interrupt flags OR-ed together to disable multiple interrupt sources for the BURTC module (BURTC_IFS_nnn).

BURTC_IntEnable

```
void BURTC_IntEnable (uint32_t flags)
```

Enable one or more BURTC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	BURTC interrupt sources to enable. Use a set of interrupt flags OR-ed together to set multiple interrupt sources for the BURTC module (BURTC_IFS_nnn).

Note

- Depending on use, a pending interrupt may already be set prior to enabling the interrupt. Consider using [BURTC_IntClear\(\)](#) prior to enabling if a pending interrupt should be ignored.

BURTC_IntGet

```
uint32_t BURTC_IntGet (void )
```

Get pending BURTC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function does not clear the event bits.

Returns

- Pending BURTC interrupt sources. Returns a set of interrupt flags OR-ed together for multiple interrupt sources in the BURTC module (BURTC_IFS_nnn).

BURTC_IntGetEnabled

```
uint32_t BURTC_IntGetEnabled (void )
```

Get enabled and pending BURTC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- The event bits are not cleared by the use of this function.

Returns

- Pending BURTC interrupt sources that is also enabled. Returns a set of interrupt flags OR-ed together for multiple interrupt sources in the BURTC module (BURTC_IFS_nnn).

BURTC_IntSet

```
void BURTC_IntSet (uint32_t flags)
```

Set one or more pending BURTC interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	BURTC interrupt sources to set to pending. Use a set of interrupt flags OR-ed together to set multiple interrupt sources for the BURTC module (BURTC_IFS_nnn).

BURTC_Status

```
uint32_t BURTC_Status (void )
```

Status of BURTC RAM, timestamp and LP Mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- A mask logically OR-ed status bits

BURTC_SyncWait

```
void BURTC_SyncWait (void )
```

Wait for the BURTC to complete all synchronization of register changes and commands.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

BURTC_Start

```
void BURTC_Start (void )
```

Start BURTC counter.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will send a start command to the BURTC peripheral. The BURTC peripheral will use some LF clock ticks before the command is executed. The [BURTC_SyncWait\(\)](#) function can be used to wait for the start command to be executed.

Note

- This function requires the BURTC to be enabled.

BURTC_Stop

```
void BURTC_Stop (void )
```

Stop the BURTC counter.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will send a stop command to the BURTC peripheral. The BURTC peripheral will use some LF clock ticks before the command is executed. The [BURTC_SyncWait\(\)](#) function can be used to wait for the stop command to be executed.

Note

- This function requires the BURTC to be enabled.

BURTC_CounterGet

```
uint32_t BURTC_CounterGet (void )
```

Get BURTC counter.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- BURTC counter value

BURTC_Lock

```
void BURTC_Lock (void )
```

Lock BURTC registers, which will protect from writing new config settings.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

BURTC_Unlock

```
void BURTC_Unlock (void )
```

Unlock BURTC registers, which will enable write access to change configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Macro Definition Documentation

burtcClkDiv_1

```
#define burtcClkDiv_1
```

Value:

```
1
```

BURTC clock divisors.

These values are valid for the BURTC prescaler. Divide clock by 1.

burtcClkDiv_2

```
#define burtcClkDiv_2
```

Value:

```
2
```

Divide clock by 2.

burtcClkDiv_4

```
#define burtcClkDiv_4
```

Value:

```
4
```

Divide clock by 4.

burtcClkDiv_8

```
#define burtcClkDiv_8
```

Value:

```
8
```

Divide clock by 8.

burtcClkDiv_16

```
#define burtcClkDiv_16
```

Value:

```
16
```

Divide clock by 16.

burtcClkDiv_32

```
#define burtcClkDiv_32
```

Value:

```
32
```

Divide clock by 32.

burtcClkDiv_64

```
#define burtcClkDiv_64
```

Value:

```
64
```

Divide clock by 64.

burtcClkDiv_128

```
#define burtcClkDiv_128
```

Value:

```
128
```

Divide clock by 128.

BURTC_INIT_DEFAULT

```
#define BURTC_INIT_DEFAULT
```

Value:

```
0 | {  
0 |     true,           \  
0 |     false,          \  
0 |     1,              \  
0 |     0,              \  
0 |     false,          \  
0 |     false,          \  
0 | }
```

Default configuration for BURTC init structure.

BURTC_Init_TypeDef

BURTC initialization structure for Series 2 devices.

Public Attributes

bool	start	Start BURTC after initialization.
bool	debugRun	If true, counter will keep running under debug halt.
uint32_t	clkDiv	Clock divider.
bool	compare0Top	Set if Compare Value 0 is also top value (counter restart)
bool	em4comp	Enable EM4 wakeup on compare match.
bool	em4overflow	Enable EM4 wakeup on counter overflow.

Public Attribute Documentation

start

```
bool BURTC_Init_TypeDef::start
```

Start BURTC after initialization.

debugRun

```
bool BURTC_Init_TypeDef::debugRun
```

If true, counter will keep running under debug halt.

clkDiv

```
uint32_t BURTC_Init_TypeDef::clkDiv
```

Clock divider.
Supported range is 1-32768

compare0Top

```
bool BURTC_Init_TypeDef::compare0Top
```

Set if Compare Value 0 is also top value (counter restart)

em4comp

```
bool BURTC_Init_TypeDef::em4comp
```

Enable EM4 wakeup on compare match.

em4overflow

```
bool BURTC_Init_TypeDef::em4overflow
```

Enable EM4 wakeup on counter overflow.

BUS - Bitfield Read/Write

BUS - Bitfield Read/Write

BUS register and RAM bit/field read/write API.

API to perform bit-band and field set/clear access to RAM and peripherals.

Functions

- void

BUS_RamBitWrite(volatile uint32_t *addr, unsigned int bit, unsigned int val)

Perform a single-bit write operation on a 32-bit word in RAM.
- unsigned int

BUS_RamBitRead(volatile const uint32_t *addr, unsigned int bit)

Perform a single-bit read operation on a 32-bit word in RAM.
- void

BUS_RegBitWrite(volatile uint32_t *addr, unsigned int bit, unsigned int val)

Perform a single-bit write operation on a peripheral register.
- unsigned int

BUS_RegBitRead(volatile const uint32_t *addr, unsigned int bit)

Perform a single-bit read operation on a peripheral register.
- void

BUS_RegMaskedSet(volatile uint32_t *addr, uint32_t mask)

Perform a masked set operation on a peripheral register address.
- void

BUS_RegMaskedClear(volatile uint32_t *addr, uint32_t mask)

Perform a masked clear operation on the peripheral register address.
- void

BUS_RegMaskedWrite(volatile uint32_t *addr, uint32_t mask, uint32_t val)

Perform peripheral register masked write.
- uint32_t

BUS_RegMaskedRead(volatile const uint32_t *addr, uint32_t mask)

Perform a peripheral register masked read.

Function Documentation

BUS_RamBitWrite

```
void BUS_RamBitWrite (volatile uint32_t * addr, unsigned int bit, unsigned int val)
```

Perform a single-bit write operation on a 32-bit word in RAM.

Parameters

Type	Direction	Argument Name	Description
volatile uint32_t *	[in]	addr	An ddress of a 32-bit word in RAM.
unsigned int	[in]	bit	A bit position to write, 0-31.
unsigned int	[in]	val	A value to set bit to, 0 or 1.

This function uses Cortex-M bit-banding hardware to perform an atomic read-modify-write operation on a single bit write on a 32-bit word in RAM. See the reference manual for more details about bit-banding.

Note

- This function is atomic on Cortex-M cores with bit-banding support. Bit- banding is a multi cycle read-modify-write bus operation. RAM bit-banding is performed using the memory alias region at BITBAND_RAM_BASE.

BUS_RamBitRead

```
unsigned int BUS_RamBitRead (volatile const uint32_t * addr, unsigned int bit)
```

Perform a single-bit read operation on a 32-bit word in RAM.

Parameters

Type	Direction	Argument Name	Description
volatile const uint32_t *	[in]	addr	RAM address.
unsigned int	[in]	bit	A bit position to read, 0-31.

This function uses Cortex-M bit-banding hardware to perform an atomic read operation on a single register bit. See the reference manual for more details about bit-banding.

Note

- This function is atomic on Cortex-M cores with bit-banding support. RAM bit-banding is performed using the memory alias region at BITBAND_RAM_BASE.

Returns

- The requested bit shifted to bit position 0 in the return value.

BUS_RegBitWrite

```
void BUS_RegBitWrite (volatile uint32_t * addr, unsigned int bit, unsigned int val)
```

Perform a single-bit write operation on a peripheral register.

Parameters

Type	Direction	Argument Name	Description
volatile uint32_t *	[in]	addr	A peripheral register address.
unsigned int	[in]	bit	A bit position to write, 0-31.
unsigned int	[in]	val	A value to set bit to, 0 or 1.

This function uses Cortex-M bit-banding hardware to perform an atomic read-modify-write operation on a single register bit. See the reference manual for more details about bit-banding.

Note

- This function is atomic on Cortex-M cores with bit-banding support. Bit-banding is a multi cycle read-modify-write bus operation. Peripheral register bit-banding is performed using the memory alias region at BITBAND_PER_BASE.

BUS_RegBitRead

```
unsigned int BUS_RegBitRead (volatile const uint32_t * addr, unsigned int bit)
```

Perform a single-bit read operation on a peripheral register.

Parameters

Type	Direction	Argument Name	Description
volatile const uint32_t *	[in]	addr	A peripheral register address.

Type	Direction	Argument Name	Description
unsigned int	[in]	bit	A bit position to read, 0-31.

This function uses Cortex-M bit-banding hardware to perform an atomic read operation on a single register bit. See the reference manual for more details about bit-banding.

Note

- This function is atomic on Cortex-M cores with bit-banding support. Peripheral register bit-banding is performed using the memory alias region at BITBAND_PER_BASE.

Returns

- The requested bit shifted to bit position 0 in the return value.

BUS_RegMaskedSet

```
void BUS_RegMaskedSet (volatile uint32_t * addr, uint32_t mask)
```

Perform a masked set operation on a peripheral register address.

Parameters

Type	Direction	Argument Name	Description
volatile uint32_t *	[in]	addr	A peripheral register address.
uint32_t	[in]	mask	A mask to set.

A peripheral register masked set provides a single-cycle and atomic set operation of a bit-mask in a peripheral register. All 1s in the mask are set to 1 in the register. All 0s in the mask are not changed in the register. RAMs and special peripherals are not supported. See the reference manual for more details about the peripheral register field set.

Note

- This function is single-cycle and atomic on cores with peripheral bit set and clear support. It uses the memory alias region at PER_BITSET_MEM_BASE.

BUS_RegMaskedClear

```
void BUS_RegMaskedClear (volatile uint32_t * addr, uint32_t mask)
```

Perform a masked clear operation on the peripheral register address.

Parameters

Type	Direction	Argument Name	Description
volatile uint32_t *	[in]	addr	A peripheral register address.
uint32_t	[in]	mask	A mask to clear.

A peripheral register masked clear provides a single-cycle and atomic clear operation of a bit-mask in a peripheral register. All 1s in the mask are set to 0 in the register. All 0s in the mask are not changed in the register. RAMs and special peripherals are not supported. See the reference manual for more details about the peripheral register field clear.

Note

- This function is single-cycle and atomic on cores with peripheral bit set and clear support. It uses the memory alias region at PER_BITCLR_MEM_BASE.

BUS_RegMaskedWrite

```
void BUS_RegMaskedWrite (volatile uint32_t * addr, uint32_t mask, uint32_t val)
```

Perform peripheral register masked write.

Parameters

Type	Direction	Argument Name	Description
volatile uint32_t *	[in]	addr	A peripheral register address.
uint32_t	[in]	mask	A peripheral register mask.
uint32_t	[in]	val	A peripheral register value. The value must be shifted to the correct bit position in the register corresponding to the field defined by the mask parameter. The register value must be contained in the field defined by the mask parameter. The register value is masked to prevent involuntary spillage.

This function first reads the peripheral register and updates only bits that are set in the mask with content of val. Typically, the mask is a bit-field in the register and the value val is within the mask.

Note

- The read-modify-write operation is executed in a critical section to guarantee atomicity. Note that atomicity can only be guaranteed if register is modified only by the core, and not by other peripherals (like DMA).

BUS_RegMaskedRead

```
uint32_t BUS_RegMaskedRead (volatile const uint32_t * addr, uint32_t mask)
```

Perform a peripheral register masked read.

Parameters

Type	Direction	Argument Name	Description
volatile const uint32_t *	[in]	addr	A peripheral register address.
uint32_t	[in]	mask	A peripheral register mask.

Read an unshifted and masked value from a peripheral register.

Note

- This operation is not hardware accelerated.

Returns

- An unshifted and masked register value.

CHIP - Chip Errata Workarounds

CHIP - Chip Errata Workarounds

Chip errata workaround APIs.
API to apply chip errata workarounds at initialization and reset.

Functions

- void

[CHIP_Init](#)(void)

Chip initialization routine for revision errata workarounds.
- void

[CHIP_Reset](#)(void)

Chip reset routine with errata workarounds.

Function Documentation

CHIP_Init

void CHIP_Init (void)

Chip initialization routine for revision errata workarounds.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function must be called immediately in main().

This initialization function configures the device to a state as similar to later revisions as possible to improve software compatibility with newer parts. See the device-specific errata for details.

CHIP_Reset

void CHIP_Reset (void)

Chip reset routine with errata workarounds.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function should be called to reset the chip. It does not return.

This function applies any errata workarounds needed to cleanly reset the device and then performs a system reset. See the device-specific errata for details.

CMU - Clock Management Unit

CMU - Clock Management Unit

Clock management unit (CMU) Peripheral API.

This module contains functions for the CMU peripheral of Silicon Labs 32-bit MCUs and SoCs. The CMU module controls oscillators, clocks gates, clock multiplexers, pre-scalers, calibration modules and wait-states.

Modules

[CMU_LFXOInit_TypeDef](#)

[CMU_HFXOInit_TypeDef](#)

[CMU_DPLLInit_TypeDef](#)

Enumerations

```
enum CMU\_HFRCODPLLFreq\_TypeDef {  
    cmuHFRCODPLLFreq_1MHz = 1000000U  
    cmuHFRCODPLLFreq_2MHz = 2000000U  
    cmuHFRCODPLLFreq_4MHz = 4000000U  
    cmuHFRCODPLLFreq_7MHz = 7000000U  
    cmuHFRCODPLLFreq_13MHz = 13000000U  
    cmuHFRCODPLLFreq_16MHz = 16000000U  
    cmuHFRCODPLLFreq_19MHz = 19000000U  
    cmuHFRCODPLLFreq_26MHz = 26000000U  
    cmuHFRCODPLLFreq_32MHz = 32000000U  
    cmuHFRCODPLLFreq_38MHz = 38000000U  
    cmuHFRCODPLLFreq_48MHz = 48000000U  
    cmuHFRCODPLLFreq_56MHz = 56000000U  
    cmuHFRCODPLLFreq_64MHz = 64000000U  
    cmuHFRCODPLLFreq_80MHz = 80000000U  
    cmuHFRCODPLLFreq_UserDefined = 0  
}
```

HFRCODPLL frequency bands.

```

enum    CMU_Clock_TypeDef {
    cmuClock_SYSCLK = (CMU_SYSCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_SYSTICK = (CMU_SYSTICK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_HCLK = (CMU_HCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_EXPCLK = (CMU_EXPCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_PCLK = (CMU_PCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_LSPCLK = (CMU_LSPCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_TRACECLK = (CMU_TRACECLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_EM01GRPACLK = (CMU_EM01GRPACLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_EM01GRPBCLK = (CMU_EM01GRPBCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_EUART0CLK = (CMU_EUART0CLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_IADCCCLK = (CMU_IADCCCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_EM23GRPACLK = (CMU_EM23GRPACLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_WDOG0CLK = (CMU_WDOG0CLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_RTCCCLK = (CMU_RTCCCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_EM4GRPACLK = (CMU_EM4GRPACLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_DPLLREFCLK = (CMU_DPLLREFCLK_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_CRYPTOAES = (CMU_CRYPTOAESCTRL_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CRYPTOAESCTRL_AESEN_SHIFT << CMU_EN_BIT_POS)
    cmuClock_CRYPTOPK = (CMU_CRYPTOAESCTRL_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CRYPTOAESCTRL_PKEN_SHIFT << CMU_EN_BIT_POS)
    cmuClock_CORE = (CMU_CORE_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_PDMREF = (CMU_PDMREF_BRANCH << CMU_CLK_BRANCH_POS)
    cmuClock_LDMA = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_LDMA_SHIFT
        << CMU_EN_BIT_POS)
    cmuClock_LDMAXBAR = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_LDMAXBAR_SHIFT << CMU_EN_BIT_POS)
    cmuClock_GPCRC = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_GPCRC_SHIFT << CMU_EN_BIT_POS)
    cmuClock_TIMER0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_TIMER0_SHIFT << CMU_EN_BIT_POS)
    cmuClock_TIMER1 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_TIMER1_SHIFT << CMU_EN_BIT_POS)
    cmuClock_TIMER2 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_TIMER2_SHIFT << CMU_EN_BIT_POS)
    cmuClock_TIMER3 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_TIMER3_SHIFT << CMU_EN_BIT_POS)
    cmuClock_TIMER4 = (CMU_CLKEN1_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN1_TIMER4_SHIFT
        << CMU_EN_BIT_POS)
    cmuClock_USART0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_USART0_SHIFT << CMU_EN_BIT_POS)
    cmuClock_USART1 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_USART1_SHIFT << CMU_EN_BIT_POS)
    cmuClock_IADCO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_IADCO_SHIFT
        << CMU_EN_BIT_POS)
    cmuClock_AMUXCPO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_AMUXCPO_SHIFT << CMU_EN_BIT_POS)
    cmuClock_LETIMER0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_LETIMER0_SHIFT << CMU_EN_BIT_POS)
    cmuClock_WDOG0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_WDOG0_SHIFT << CMU_EN_BIT_POS)
    cmuClock_I2CO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_I2CO_SHIFT <<
        CMU_EN_BIT_POS)
    cmuClock_I2C1 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_I2C1_SHIFT <<
        CMU_EN_BIT_POS)
    cmuClock_SYSCFG = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_SYSCFG_SHIFT << CMU_EN_BIT_POS)
    cmuClock_DPLL0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_DPLL0_SHIFT
        << CMU_EN_BIT_POS)
    cmuClock_HFRCO0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_HFRCO0_SHIFT << CMU_EN_BIT_POS)
    cmuClock_HFXO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_HFXO0_SHIFT
        << CMU_EN_BIT_POS)
    cmuClock_FSRCO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_FSRCO_SHIFT << CMU_EN_BIT_POS)
    cmuClock_LFRCO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_LFRCO_SHIFT << CMU_EN_BIT_POS)
    cmuClock_LFXO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_LFXO_SHIFT
        << CMU_EN_BIT_POS)
    cmuClock_ULFRCO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_ULFRCO_SHIFT << CMU_EN_BIT_POS)
    cmuClock_EUART0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_EUART0_SHIFT << CMU_EN_BIT_POS)
    cmuClock_PDM = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_PDM_SHIFT <<
        CMU_EN_BIT_POS)
    cmuClock_GPIO = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_GPIO_SHIFT
        << CMU_EN_BIT_POS)
    cmuClock_PRS = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_PRS_SHIFT <<
        CMU_EN_BIT_POS)
    cmuClock_BURAM = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_BURAM_SHIFT << CMU_EN_BIT_POS)
    cmuClock_BURTC = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) |
        (_CMU_CLKEN0_BURTC_SHIFT << CMU_EN_BIT_POS)
    cmuClock_RTCC = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_RTCC_SHIFT
        << CMU_EN_BIT_POS)
    cmuClock_DCDC = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_DCDC_SHIFT
        << CMU_EN_BIT_POS)
}

```

```

cmuClock_CRYPTOAACC =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_CRYPTOAACC_SHIFT
<< CMU_EN_BIT_POS)
cmuClock_SMU =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_SMU_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_ICACHE =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_ICACHE0_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_MSC =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_MSC_SHIFT <<
CMU_EN_BIT_POS)

```

```

}

```

Clock points in CMU clock-tree.

```

enum    CMU_Osc_TypeDef {

```

```

    cmuOsc_LFXO
    cmuOsc_LFRCO
    cmuOsc_FSRCO
    cmuOsc_HFXO
    cmuOsc_HFRCODPLL
    cmuOsc_ULFRCO

```

```

}

```

Oscillator types.

```

enum    CMU_Select_TypeDef {

```

```

    cmuSelect_Error
    cmuSelect_Disabled
    cmuSelect_FSRCO
    cmuSelect_HFXO
    cmuSelect_HFXORT
    cmuSelect_HFRCODPLL
    cmuSelect_HFRCODPLLRT
    cmuSelect_CLKINO
    cmuSelect_LFXO
    cmuSelect_LFRCO
    cmuSelect_ULFRCO
    cmuSelect_HCLK
    cmuSelect_SYSClk
    cmuSelect_HCLKDIV1024
    cmuSelect_EM01GRPACLK
    cmuSelect_EM23GRPACLK
    cmuSelect_EXPCLK
    cmuSelect_PRS
    cmuSelect_TEMPOSC
    cmuSelect_PFMOSC
    cmuSelect_BIASOSC

```

```

}

```

Selectable clock sources.

```

enum    CMU_DPLLEdgeSel_TypeDef {

```

```

    cmuDPLLEdgeSel_Fall = 0
    cmuDPLLEdgeSel_Rise = 1

```

```

}

```

DPLL reference clock edge detect selector.

```

enum    CMU_DPLLLockMode_TypeDef {

```

```

    cmuDPLLLockMode_Freq = _DPLL_CFG_MODE_FLL
    cmuDPLLLockMode_Phase = _DPLL_CFG_MODE_PLL

```

```

}

```

DPLL lock mode selector.

```
enum CMU_LfxoOscMode_TypeDef {
    cmuLfxoOscMode_Crystal = _LFXO_CFG_MODE_XTAL
    cmuLfxoOscMode_AcCoupledSine = _LFXO_CFG_MODE_BUFEXTCLK
    cmuLfxoOscMode_External = _LFXO_CFG_MODE_DIGEXTCLK
}
LFXO oscillator modes.
```

```
enum CMU_LfxoStartupDelay_TypeDef {
    cmuLfxoStartupDelay_2Cycles = _LFXO_CFG_TIMEOUT_CYCLES2
    cmuLfxoStartupDelay_256Cycles = _LFXO_CFG_TIMEOUT_CYCLES256
    cmuLfxoStartupDelay_1KCycles = _LFXO_CFG_TIMEOUT_CYCLES1K
    cmuLfxoStartupDelay_2KCycles = _LFXO_CFG_TIMEOUT_CYCLES2K
    cmuLfxoStartupDelay_4KCycles = _LFXO_CFG_TIMEOUT_CYCLES4K
    cmuLfxoStartupDelay_8KCycles = _LFXO_CFG_TIMEOUT_CYCLES8K
    cmuLfxoStartupDelay_16KCycles = _LFXO_CFG_TIMEOUT_CYCLES16K
    cmuLfxoStartupDelay_32KCycles = _LFXO_CFG_TIMEOUT_CYCLES32K
}
LFXO start-up timeout delay.
```

```
enum CMU_HfxoOscMode_TypeDef {
    cmuHfxoOscMode_Crystal = _HFXO_CFG_MODE_XTAL
    cmuHfxoOscMode_ExternalSine = _HFXO_CFG_MODE_EXTCLK
}
HFXO oscillator modes.
```

```
enum CMU_HfxoCbLsbTimeout_TypeDef {
    cmuHfxoCbLsbTimeout_8us = _HFXO_XTALCFG_TIMEOUTCBLSB_T8US
    cmuHfxoCbLsbTimeout_20us = _HFXO_XTALCFG_TIMEOUTCBLSB_T20US
    cmuHfxoCbLsbTimeout_41us = _HFXO_XTALCFG_TIMEOUTCBLSB_T41US
    cmuHfxoCbLsbTimeout_62us = _HFXO_XTALCFG_TIMEOUTCBLSB_T62US
    cmuHfxoCbLsbTimeout_83us = _HFXO_XTALCFG_TIMEOUTCBLSB_T83US
    cmuHfxoCbLsbTimeout_104us = _HFXO_XTALCFG_TIMEOUTCBLSB_T104US
    cmuHfxoCbLsbTimeout_125us = _HFXO_XTALCFG_TIMEOUTCBLSB_T125US
    cmuHfxoCbLsbTimeout_166us = _HFXO_XTALCFG_TIMEOUTCBLSB_T166US
    cmuHfxoCbLsbTimeout_208us = _HFXO_XTALCFG_TIMEOUTCBLSB_T208US
    cmuHfxoCbLsbTimeout_250us = _HFXO_XTALCFG_TIMEOUTCBLSB_T250US
    cmuHfxoCbLsbTimeout_333us = _HFXO_XTALCFG_TIMEOUTCBLSB_T333US
    cmuHfxoCbLsbTimeout_416us = _HFXO_XTALCFG_TIMEOUTCBLSB_T416US
    cmuHfxoCbLsbTimeout_833us = _HFXO_XTALCFG_TIMEOUTCBLSB_T833US
    cmuHfxoCbLsbTimeout_1250us = _HFXO_XTALCFG_TIMEOUTCBLSB_T1250US
    cmuHfxoCbLsbTimeout_2083us = _HFXO_XTALCFG_TIMEOUTCBLSB_T2083US
    cmuHfxoCbLsbTimeout_3750us = _HFXO_XTALCFG_TIMEOUTCBLSB_T3750US
}
HFXO core bias LSB change timeout.
```

```
enum CMU_HfxoSteadyStateTimeout_TypeDef {
    cmuHfxoSteadyStateTimeout_16us = _HFXO_XTALCFG_TIMEOUTSTEADY_T16US
    cmuHfxoSteadyStateTimeout_41us = _HFXO_XTALCFG_TIMEOUTSTEADY_T41US
    cmuHfxoSteadyStateTimeout_83us = _HFXO_XTALCFG_TIMEOUTSTEADY_T83US
    cmuHfxoSteadyStateTimeout_125us = _HFXO_XTALCFG_TIMEOUTSTEADY_T125US
    cmuHfxoSteadyStateTimeout_166us = _HFXO_XTALCFG_TIMEOUTSTEADY_T166US
    cmuHfxoSteadyStateTimeout_208us = _HFXO_XTALCFG_TIMEOUTSTEADY_T208US
    cmuHfxoSteadyStateTimeout_250us = _HFXO_XTALCFG_TIMEOUTSTEADY_T250US
    cmuHfxoSteadyStateTimeout_333us = _HFXO_XTALCFG_TIMEOUTSTEADY_T333US
    cmuHfxoSteadyStateTimeout_416us = _HFXO_XTALCFG_TIMEOUTSTEADY_T416US
    cmuHfxoSteadyStateTimeout_500us = _HFXO_XTALCFG_TIMEOUTSTEADY_T500US
    cmuHfxoSteadyStateTimeout_666us = _HFXO_XTALCFG_TIMEOUTSTEADY_T666US
    cmuHfxoSteadyStateTimeout_833us = _HFXO_XTALCFG_TIMEOUTSTEADY_T833US
    cmuHfxoSteadyStateTimeout_1666us = _HFXO_XTALCFG_TIMEOUTSTEADY_T1666US
    cmuHfxoSteadyStateTimeout_2500us = _HFXO_XTALCFG_TIMEOUTSTEADY_T2500US
    cmuHfxoSteadyStateTimeout_4166us = _HFXO_XTALCFG_TIMEOUTSTEADY_T4166US
    cmuHfxoSteadyStateTimeout_7500us = _HFXO_XTALCFG_TIMEOUTSTEADY_T7500US
}
HFXO steady state timeout.
```

```
enum CMU_HfxoCoreDegen_TypeDef {
    cmuHfxoCoreDegen_None = _HFXO_XTALCTRL_COREDGENANA_NONE
    cmuHfxoCoreDegen_33 = _HFXO_XTALCTRL_COREDGENANA_DGEN33
    cmuHfxoCoreDegen_50 = _HFXO_XTALCTRL_COREDGENANA_DGEN50
    cmuHfxoCoreDegen_100 = _HFXO_XTALCTRL_COREDGENANA_DGEN100
}
HFXO core degeneration control.
```

```
enum CMU_HfxoCtuneFixCap_TypeDef {
    cmuHfxoCtuneFixCap_None = _HFXO_XTALCTRL_CTUNEFIXANA_NONE
    cmuHfxoCtuneFixCap_Xi = _HFXO_XTALCTRL_CTUNEFIXANA_XI
    cmuHfxoCtuneFixCap_Xo = _HFXO_XTALCTRL_CTUNEFIXANA_XO
    cmuHfxoCtuneFixCap_Both = _HFXO_XTALCTRL_CTUNEFIXANA_BOTH
}
HFXO XI and XO pin fixed capacitor control.
```

```
enum CMU_Precision_TypeDef {
    cmuPrecisionDefault
    cmuPrecisionHigh
}
Oscillator precision modes.
```

Typedefs

```
typedef CMU_ClkDiv_TypeDef
uint32_t
Clock divider configuration.
```

Functions

```
uint32_t CMU_Calibrate(uint32_t cycles, CMU_Select_TypeDef ref)
Calibrate an oscillator.
```

```
void CMU_CalibrateConfig(uint32_t downCycles, CMU_Select_TypeDef downSel,
    CMU_Select_TypeDef upSel)
Configure clock calibration.
```

```
uint32_t CMU_CalibrateCountGet(void)
Get calibration count value.
```

```
void CMU_ClkOutPinConfig(uint32_t clkNo, CMU_Select_TypeDef sel, CMU_ClkDiv_TypeDef clkDiv,
    GPIO_Port_TypeDef port, unsigned int pin)
Direct a clock to a GPIO pin.
```

```
CMU_ClkDiv_TypeDef CMU_ClockDivGet(CMU_Clock_TypeDef clock)
Get clock divisor.
```

```
void CMU_ClockDivSet(CMU_Clock_TypeDef clock, CMU_ClkDiv_TypeDef div)
Set clock divisor.
```

```
void CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable)
Enable/disable a clock.
```

```
uint32_t CMU_ClockFreqGet(CMU_Clock_TypeDef clock)
Get clock frequency for a clock point.
```

```
CMU_Select_TypeDef CMU_ClockSelectGet(CMU_Clock_TypeDef clock)
Get currently selected reference clock used for a clock branch.
```

```
void CMU_ClockSelectSet(CMU_Clock_TypeDef clock, CMU_Select_TypeDef ref)
Select reference clock/oscillator used for a clock branch.
```

```
uint16_t CMU_LF_ClockPrecisionGet(CMU_Clock_TypeDef clock)
Gets the precision (in PPM) of the specified low frequency clock branch.
```

```
uint16_t CMU_HF_ClockPrecisionGet(CMU_Clock_TypeDef clock)
Gets the precision (in PPM) of the specified high frequency clock branch.
```

```
CMU_HFRCODPLLFreq_TypeDef CMU_HFRCODPLLBandGet(void)
Get HFRCODPLL band in use.
```

```
void CMU_HFRCODPLLBandSet(CMU_HFRCODPLLFreq_TypeDef freq)
Set HFRCODPLL band and the tuning value based on the value in the calibration table made during production.
```

bool	CMU_DPLLLock (const CMU_DPLLInit_TypeDef *init) Lock the DPLL to a given frequency.
void	CMU_HFXOInit (const CMU_HFXOInit_TypeDef *hfxoInit) Initialize all HFXO control registers.
sl_status_t	CMU_HFXOCTuneSet (uint32_t ctune) Set the HFXO crystal tuning capacitance.
uint32_t	CMU_HFXOCTuneGet (void) Get the HFXO crystal tuning capacitance.
void	CMU_HFXOCTuneDeltaSet (int32_t delta) Set the HFXO crystal tuning delta.
int32_t	CMU_HFXOCTuneDeltaGet (void) Get the HFXO crystal tuning delta.
void	CMU_HFXOCoreBiasCurrentCalibrate (void) Recalibrate the HFXO's Core Bias Current.
void	CMU_LFXOInit (const CMU_LFXOInit_TypeDef *lfxoInit) Initialize LFXO control registers.
void	CMU_LFXOPrecisionSet (uint16_t precision) Sets LFXO's crystal precision, in PPM.
uint16_t	CMU_LFXOPrecisionGet (void) Gets LFXO's crystal precision, in PPM.
void	CMU_HFXOPrecisionSet (uint16_t precision) Sets HFXO's crystal precision, in PPM.
uint16_t	CMU_HFXOPrecisionGet (void) Gets HFXO's crystal precision, in PPM.
uint32_t	CMU_OscillatorTuningGet (CMU_Osc_TypeDef osc) Get oscillator frequency tuning setting.
void	CMU_OscillatorTuningSet (CMU_Osc_TypeDef osc, uint32_t val) Set the oscillator frequency tuning control.
void	CMU_UpdateWaitStates (uint32_t freq, int vscale) Configure wait state settings necessary to switch to a given core clock frequency at a certain voltage scale level.
void	CMU_PCNTClockExternalSet (unsigned int instance, bool external) Select the PCNTn clock.
void	CMU_CalibrateCont (bool enable) Configure continuous calibration mode.
void	CMU_CalibrateStart (void) Start calibration.
void	CMU_CalibrateStop (void) Stop calibration counters.
void	CMU_DPLLUnlock (void) Unlock the DPLL.
void	CMU_IntClear (uint32_t flags) Clear one or more pending CMU interrupt flags.
void	CMU_IntDisable (uint32_t flags) Disable one or more CMU interrupt sources.

void	CMU_IntEnable (uint32_t flags) Enable one or more CMU interrupt sources.
uint32_t	CMU_IntGet (void) Get pending CMU interrupt sources.
uint32_t	CMU_IntGetEnabled (void) Get enabled and pending CMU interrupt flags.
void	CMU_IntSet (uint32_t flags) Set one or more pending CMU interrupt sources.
void	CMU_Lock (void) Lock CMU register access in order to protect registers contents against unintended modification.
void	CMU_OscillatorEnable (CMU_Osc_TypeDef osc, bool enable, bool wait) Enable/disable oscillator.
void	CMU_Unlock (void) Unlock CMU register access so that writing to registers is possible.
void	CMU_WdogLock (void) Lock WDOG register access in order to protect registers contents against unintended modification.
void	CMU_WdogUnlock (void) Unlock WDOG register access so that writing to registers is possible.
uint32_t	CMU_PrescToLog2 (uint32_t presc) Convert prescaler divider to a logarithmic value.

Macros

#define	CMU_CLOCK_SELECT_SET (clock, sel) Macro to set clock sources in the clock tree.
#define	_CMU_EM01GRPACLKCTRL_CLKSEL_DISABLED 0x00000000UL Disable clocks configuration.
#define	CMU_EM01GRPACLKCTRL_CLKSEL_DISABLED (_CMU_EM01GRPACLKCTRL_CLKSEL_DISABLED << 0) Shifted mode DISABLED for CMU_EM01GRPACLKCTRL.
#define	_CMU_EM01GRPBCLKCTRL_CLKSEL_DISABLED 0x00000000UL Mode DISABLED for CMU_EM01GRPBCLKCTRL
#define	CMU_EM01GRPBCLKCTRL_CLKSEL_DISABLED (_CMU_EM01GRPBCLKCTRL_CLKSEL_DISABLED << 0) Shifted mode DISABLED for CMU_EM01GRPBCLKCTRL.
#define	_CMU_EM23GRPACLKCTRL_CLKSEL_DISABLED 0x00000000UL Mode DISABLED for CMU_EM23GRPACLKCTRL
#define	CMU_EM23GRPACLKCTRL_CLKSEL_DISABLED (_CMU_EM23GRPACLKCTRL_CLKSEL_DISABLED << 0) Shifted mode DISABLED for CMU_EM23GRPACLKCTRL.
#define	_CMU_EM4GRPACLKCTRL_CLKSEL_DISABLED 0x00000000UL Mode DISABLED for CMU_EM4GRPACLKCTRL
#define	CMU_EM4GRPACLKCTRL_CLKSEL_DISABLED (_CMU_EM4GRPACLKCTRL_CLKSEL_DISABLED << 0) Shifted mode DISABLED for CMU_EM4GRPACLKCTRL.
#define	_CMU_WDOG0CLKCTRL_CLKSEL_DISABLED 0x00000000UL Mode DISABLED for CMU_WDOG0CLKCTRL
#define	CMU_WDOG0CLKCTRL_CLKSEL_DISABLED (_CMU_WDOG0CLKCTRL_CLKSEL_DISABLED << 0) Shifted mode DISABLED for CMU_WDOG0CLKCTRL

```
#define _CMU_WDOG1CLKCTRL_CLKSEL_DISABLED 0x00000000UL
Mode DISABLED for CMU_WDOG1CLKCTRL

#define CMU_WDOG1CLKCTRL_CLKSEL_DISABLED (_CMU_WDOG1CLKCTRL_CLKSEL_DISABLED << 0)
Shifted mode DISABLED for CMU_WDOG1CLKCTRL

#define _CMU_EUSARTOCLKCTRL_CLKSEL_DISABLED 0x00000000UL
Mode DISABLED for CMU_EUSARTOCLKCTRL

#define CMU_EUSARTOCLKCTRL_CLKSEL_DISABLED (_CMU_EUSARTOCLKCTRL_CLKSEL_DISABLED << 0)
Shifted mode DISABLED for CMU_EUSARTOCLKCTRL.

#define _CMU_SYSRTCCLKCTRL_CLKSEL_DISABLED 0x00000000UL
Mode DISABLED for CMU_SYSRTCCLKCTRL

#define CMU_SYSRTCCLKCTRL_CLKSEL_DISABLED (_CMU_SYSRTCCLKCTRL_CLKSEL_DISABLED << 0)
Shifted mode DISABLED for CMU_SYSRTCCLKCTRL.

#define CMU_HFRCODPLL_MIN cmuHFRCODPLLFreq_1MHz
HFRCODPLL maximum frequency.

#define CMU_HFRCODPLL_MAX cmuHFRCODPLLFreq_80MHz
HFRCODPLL minimum frequency.

#define CMU_LFXOINIT_DEFAULT undefined
Default LFXO initialization values for XTAL mode.

#define CMU_LFXOINIT_EXTERNAL_CLOCK undefined
Default LFXO initialization values for external clock mode.

#define CMU_LFXOINIT_EXTERNAL_SINE undefined
Default LFXO initialization values for external sine mode.

#define CMU_HFXOINIT_DEFAULT undefined
Default HFXO initialization values for XTAL mode.

#define CMU_HFXOINIT_EXTERNAL_SINE undefined
Default HFXO initialization values for external sine mode.

#define CMU_HFXOINIT_EXTERNAL_SINEPKDET undefined
Default HFXO initialization values for external sine mode with peak detector.

#define CMU_DPLL_LFXO_TO_40MHZ undefined
DPLL initialization values for 39,998,805 Hz using LFXO as reference clock, M=2 and N=3661.

#define CMU_DPLL_HFXO_TO_76_8MHZ undefined
DPLL initialization values for 76,800,000 Hz using HFXO as reference clock, M = 1919, N = 3839.

#define CMU_DPLL_HFXO_TO_80MHZ undefined
DPLL initialization values for 80,000,000 Hz using HFXO as reference clock, M = 1919, N = 3999.

#define CMU_DPLLINIT_DEFAULT undefined
Default configurations for DPLL initialization.
```

Enumeration Documentation

CMU_HFRCODPLLFreq_TypeDef

CMU_HFRCODPLLFreq_TypeDef

HFRCODPLL frequency bands.

Enumerator	
cmuHFRCODPLLFreq_1MHz	1MHz RC band.
cmuHFRCODPLLFreq_2MHz	2MHz RC band.

cmuHFRCODPLLFreq_4M0Hz	4MHz RC band.
cmuHFRCODPLLFreq_7M0Hz	7MHz RC band.
cmuHFRCODPLLFreq_13M0Hz	13MHz RC band.
cmuHFRCODPLLFreq_16M0Hz	16MHz RC band.
cmuHFRCODPLLFreq_19M0Hz	19MHz RC band.
cmuHFRCODPLLFreq_26M0Hz	26MHz RC band.
cmuHFRCODPLLFreq_32M0Hz	32MHz RC band.
cmuHFRCODPLLFreq_38M0Hz	38MHz RC band.
cmuHFRCODPLLFreq_48M0Hz	48MHz RC band.
cmuHFRCODPLLFreq_56M0Hz	56MHz RC band.
cmuHFRCODPLLFreq_64M0Hz	64MHz RC band.
cmuHFRCODPLLFreq_80M0Hz	80MHz RC band.
cmuHFRCODPLLFreq_UserDefined	

CMU_Clock_TypeDef

CMU_Clock_TypeDef

Clock points in CMU clock-tree.

Enumerator

cmuClock_SYSCLK	SYSTEM clock.
cmuClock_SYSTICK	SYSTICK clock.
cmuClock_HCLK	Core and AHB bus interface clock.
cmuClock_EXPCLK	Export clock.
cmuClock_PCLK	Peripheral APB bus interface clock.
cmuClock_LSPCLK	Low speed peripheral APB bus interface clock.
cmuClock_TRACECLK	Debug trace.
cmuClock_EM01GRPACLK	EM01GRPA clock.
cmuClock_EM01GRPBCLK	EM01GRPB clock.
cmuClock_EUART0CLK	EUART0 clock.
cmuClock_IADCCLK	IADC clock.
cmuClock_EM23GRPACLK	EM23GRPA clock.
cmuClock_WDOG0CLK	WDOG0 clock.
cmuClock_RTCCCLK	RTCC clock.
cmuClock_EM4GRPACLK	EM4GRPA clock.
cmuClock_DPLLREFCLK	DPLLREF clock.
cmuClock_CRYPTOAES	CRYPTOAES clock.
cmuClock_CRYPTOPK	CRYPTOPK clock.
cmuClock_CORE	Cortex-M33 core clock.
cmuClock_PDMREF	PDMREF clock.
cmuClock_LDMA	LDMA clock.
cmuClock_LDMAXBAR	LDMAXBAR clock.
cmuClock_GPCRC	GPCRC clock.

cmuClock_TIMER0	TIMER0 clock.
cmuClock_TIMER1	TIMER1 clock.
cmuClock_TIMER2	TIMER2 clock.
cmuClock_TIMER3	TIMER3 clock.
cmuClock_TIMER4	TIMER4 clock.
cmuClock_USART0	USART0 clock.
cmuClock_USART1	USART1 clock.
cmuClock_IADC0	IADC0 clock.
cmuClock_AMUXCP0	AMUXCP0 clock.
cmuClock_LETIMER0	LETIMER0 clock.
cmuClock_WDOG0	WDOG0 clock.
cmuClock_I2C0	I2C0 clock.
cmuClock_I2C1	I2C1 clock.
cmuClock_SYSCFG	SYSCFG clock.
cmuClock_DPLL0	DPLL0 clock.
cmuClock_HFRCO0	HFRCO0 clock.
cmuClock_HFXO	HFXO clock.
cmuClock_FSRCO	FSRCO clock.
cmuClock_LFRCO	LFRCO clock.
cmuClock_LFXO	LFXO clock.
cmuClock_ULFRCO	ULFRCO clock.
cmuClock_EUART0	EUART0 clock.
cmuClock_PDM	PDM clock.
cmuClock_GPIO	GPIO clock.
cmuClock_PRS	PRS clock.
cmuClock_BURAM	BURAM clock.
cmuClock_BURTC	BURTC clock.
cmuClock_RTCC	RTCC clock.
cmuClock_DCDC	DCDC clock.
cmuClock_CRYPTOAACC	CRYPTOAACC clock.
cmuClock_SMU	SMU clock.
cmuClock_ICACHE	ICACHE clock.
cmuClock_MSC	MSC clock.

CMU_Osc_TypeDef

CMU_Osc_TypeDef

Oscillator types.

Enumerator	
cmuOsc_LFXO	Low frequency crystal oscillator.
cmuOsc_LFRCO	Low frequency RC oscillator.

cmuOsc_FSRCO	Fast startup fixed frequency RC oscillator.
cmuOsc_HFXO	High frequency crystal oscillator.
cmuOsc_HFRCODPLL	High frequency RC and DPLL oscillator.
cmuOsc_ULFRCO	Ultra low frequency RC oscillator.

CMU_Select_TypeDef

CMU_Select_TypeDef

Selectable clock sources.

	Enumerator
cmuSelect_Error	Usage error.
cmuSelect_Disabled	Clock selector disabled.
cmuSelect_FSRCO	Fast startup fixed frequency RC oscillator.
cmuSelect_HFXO	High frequency crystal oscillator.
cmuSelect_HFXORT	Re-timed high frequency crystal oscillator.
cmuSelect_HFRCODPLL	High frequency RC and DPLL oscillator.
cmuSelect_HFRCODPLLRT	Re-timed high frequency RC and DPLL oscillator.
cmuSelect_CLKINO	External clock input.
cmuSelect_LFXO	Low frequency crystal oscillator.
cmuSelect_LFRCO	Low frequency RC oscillator.
cmuSelect_ULFRCO	Ultra low frequency RC oscillator.
cmuSelect_HCLK	Core and AHB bus interface clock.
cmuSelect_SYSClk	System clock.
cmuSelect_HCLKDIV1024	Prescaled HCLK frequency clock.
cmuSelect_EM01GRPACLK	EM01GRPA clock.
cmuSelect_EM23GRPACLK	EM23GRPA clock.
cmuSelect_EXPCLK	Pin export clock.
cmuSelect_PRS	PRS input as clock.
cmuSelect_TEMPOSC	Temperature oscillator.
cmuSelect_PFMOSC	PFM oscillator.
cmuSelect_BIASOSC	BIAS oscillator.

CMU_DPLLEdgeSel_TypeDef

CMU_DPLLEdgeSel_TypeDef

DPLL reference clock edge detect selector.

	Enumerator
cmuDPLLEdgeSel_Fall	Detect falling edge of reference clock.
cmuDPLLEdgeSel_Rise	Detect rising edge of reference clock.

CMU_DPLLLockMode_TypeDef

CMU_DPLLLockMode_TypeDef

DPLL lock mode selector.

Enumerator	
cmuDPLLLockMode_Freq	Frequency lock mode.
cmuDPLLLockMode_Phase	Phase lock mode.

CMU_LfxoOscMode_TypeDef

CMU_LfxoOscMode_TypeDef

LFXO oscillator modes.

Enumerator	
cmuLfxoOscMode_Crystal	Crystal oscillator.
cmuLfxoOscMode_AcCoupledSine	External AC coupled sine.
cmuLfxoOscMode_External	External digital clock.

CMU_LfxoStartupDelay_TypeDef

CMU_LfxoStartupDelay_TypeDef

LFXO start-up timeout delay.

Enumerator	
cmuLfxoStartupDelay_2Cycles	2 cycles start-up delay.
cmuLfxoStartupDelay_256Cycles	256 cycles start-up delay.
cmuLfxoStartupDelay_1KCycles	1K cycles start-up delay.
cmuLfxoStartupDelay_2KCycles	2K cycles start-up delay.
cmuLfxoStartupDelay_4KCycles	4K cycles start-up delay.
cmuLfxoStartupDelay_8KCycles	8K cycles start-up delay.
cmuLfxoStartupDelay_16KCycles	16K cycles start-up delay.
cmuLfxoStartupDelay_32KCycles	32K cycles start-up delay.

CMU_HfxoOscMode_TypeDef

CMU_HfxoOscMode_TypeDef

HFXO oscillator modes.

Enumerator	
cmuHfxoOscMode_Crystal	Crystal oscillator.
cmuHfxoOscMode_ExternalSine	External digital clock.

CMU_HfxoCbLsbTimeout_TypeDef

CMU_HfxoCbLsbTimeout_TypeDef

HFXO core bias LSB change timeout.

Enumerator	
cmuHfxoCbLsbTimeout_8us	8 us timeout.
cmuHfxoCbLsbTimeout_20us	20 us timeout.
cmuHfxoCbLsbTimeout_41us	41 us timeout.
cmuHfxoCbLsbTimeout_62us	62 us timeout.
cmuHfxoCbLsbTimeout_83us	83 us timeout.
cmuHfxoCbLsbTimeout_104us	104 us timeout.
cmuHfxoCbLsbTimeout_125us	125 us timeout.
cmuHfxoCbLsbTimeout_166us	166 us timeout.
cmuHfxoCbLsbTimeout_208us	208 us timeout.
cmuHfxoCbLsbTimeout_250us	250 us timeout.
cmuHfxoCbLsbTimeout_333us	333 us timeout.
cmuHfxoCbLsbTimeout_416us	416 us timeout.
cmuHfxoCbLsbTimeout_833us	833 us timeout.
cmuHfxoCbLsbTimeout_1250us	1250 us timeout.
cmuHfxoCbLsbTimeout_2083us	2083 us timeout.
cmuHfxoCbLsbTimeout_3750us	3750 us timeout.

CMU_HfxoSteadyStateTimeout_TypeDef

CMU_HfxoSteadyStateTimeout_TypeDef

HFXO steady state timeout.

Enumerator	
cmuHfxoSteadyStateTimeout_16us	16 us timeout.
cmuHfxoSteadyStateTimeout_41us	41 us timeout.
cmuHfxoSteadyStateTimeout_83us	83 us timeout.
cmuHfxoSteadyStateTimeout_125us	125 us timeout.
cmuHfxoSteadyStateTimeout_166us	166 us timeout.
cmuHfxoSteadyStateTimeout_208us	208 us timeout.
cmuHfxoSteadyStateTimeout_250us	250 us timeout.
cmuHfxoSteadyStateTimeout_333us	333 us timeout.
cmuHfxoSteadyStateTimeout_416us	416 us timeout.
cmuHfxoSteadyStateTimeout_500us	500 us timeout.
cmuHfxoSteadyStateTimeout_666us	666 us timeout.
cmuHfxoSteadyStateTimeout_833us	833 us timeout.
cmuHfxoSteadyStateTimeout_1666us	1666 us timeout.
cmuHfxoSteadyStateTimeout_2500us	2500 us timeout.
cmuHfxoSteadyStateTimeout_4166us	4166 us timeout.
cmuHfxoSteadyStateTimeout_7500us	7500 us timeout.

CMU_HfxoCoreDegen_TypeDef

CMU_HfxoCoreDegen_TypeDef

HFXO core degeneration control.

Enumerator	
cmuHfxoCoreDegen_None	No core degeneration.
cmuHfxoCoreDegen_33	Core degeneration control 33.
cmuHfxoCoreDegen_50	Core degeneration control 50.
cmuHfxoCoreDegen_100	Core degeneration control 100.

CMU_HfxoCtuneFixCap_TypeDef

CMU_HfxoCtuneFixCap_TypeDef

HFXO XI and XO pin fixed capacitor control.

Enumerator	
cmuHfxoCtuneFixCap_None	No fixed capacitors.
cmuHfxoCtuneFixCap_Xi	Fixed capacitor on XI pin.
cmuHfxoCtuneFixCap_Xo	Fixed capacitor on XO pin.
cmuHfxoCtuneFixCap_Both	Fixed capacitor on both pins.

CMU_Precision_TypeDef

CMU_Precision_TypeDef

Oscillator precision modes.

Enumerator	
cmuPrecisionDefault	Default precision mode.
cmuPrecisionHigh	High precision mode.

Typedef Documentation

CMU_ClkDiv_TypeDef

typedef uint32_t CMU_ClkDiv_TypeDef

Clock divider configuration.

Function Documentation

CMU_Calibrate

uint32_t CMU_Calibrate (uint32_t cycles, [CMU_Select_TypeDef](#) ref)

Calibrate an oscillator.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	cycles	The number of HCLK cycles to run calibration. Increasing this number increases precision, but the calibration will take more time.
CMU_Select_TypeDef	[in]	ref	The reference clock used to compare against HCLK.

Run a calibration of a selectable reference clock against HCLK. Please refer to the reference manual, CMU chapter, for further details.

Note

- This function will not return until calibration measurement is completed.

Returns

- The number of ticks the selected reference clock ticked while running cycles ticks of the HCLK clock.

CMU_CalibrateConfig

```
void CMU_CalibrateConfig (uint32_t downCycles, CMU\_Select\_TypeDef downSel, CMU\_Select\_TypeDef upSel)
```

Configure clock calibration.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	downCycles	The number of downSel clock cycles to run calibration. Increasing this number increases precision, but the calibration will take more time.
CMU_Select_TypeDef	[in]	downSel	The clock which will be counted down downCycles cycles.
CMU_Select_TypeDef	[in]	upSel	The reference clock, the number of cycles generated by this clock will be counted and added up, the result can be given with the CMU_CalibrateCountGet() function call.

Configure a calibration for a selectable clock source against another selectable reference clock. Refer to the reference manual, CMU chapter, for further details.

Note

- After configuration, a call to [CMU_CalibrateStart\(\)](#) is required, and the resulting calibration value can be read with the [CMU_CalibrateCountGet\(\)](#) function call.

CMU_CalibrateCountGet

```
uint32_t CMU_CalibrateCountGet (void )
```

Get calibration count value.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- If continuous calibration mode is active, calibration busy will almost always be off, and reading the value will be just needed, where the normal case would be that this function call has been triggered by the CALRDY interrupt flag.

Returns

- Calibration count, the number of UPSEL clocks (see [CMU_CalibrateConfig\(\)](#)) in the period of DOWNSEL oscillator clock cycles configured by a previous write operation to CMU->CALCNT.

CMU_ClkOutPinConfig

```
void CMU_ClkOutPinConfig (uint32_t clkNo, CMU\_Select\_TypeDef sel, CMU\_ClkDiv\_TypeDef clkDiv,
GPIO\_Port\_TypeDef port, unsigned int pin)
```

Direct a clock to a GPIO pin.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	clkNo	Selects between CLKOUT0, CLKOUT1 or CLKOUT2 outputs. Use values 0, 1 or 2.
CMU_Select_TypeDef	[in]	sel	Select clock source.
CMU_ClkDiv_TypeDef	[in]	clkDiv	Select a clock divisor (1..32). Only applicable when cmuSelect_EXPCLK is selected as clock source.
GPIO_Port_TypeDef	[in]	port	GPIO port.
unsigned int	[in]	pin	GPIO pin.

Note

- Refer to the reference manual and the datasheet for details on which GPIO port/pins that are available.

CMU_ClockDivGet

```
CMU\_ClkDiv\_TypeDef CMU_ClockDivGet (CMU\_Clock\_TypeDef clock)
```

Get clock divisor.

Parameters

Type	Direction	Argument Name	Description
CMU_Clock_TypeDef	[in]	clock	Clock point to get divisor for. Notice that not all clock points have a divisors. Please refer to CMU overview in reference manual.

Returns

- The current clock point divisor. 1 is returned if `clock` specifies a clock point without divisor.

CMU_ClockDivSet

```
void CMU_ClockDivSet (CMU_Clock_TypeDef clock, CMU_ClkDiv_TypeDef div)
```

Set clock divisor.

Parameters

Type	Direction	Argument Name	Description
CMU_Clock_TypeDef	[in]	clock	Clock point to set divisor for. Notice that not all clock points have a divisor, please refer to CMU overview in the reference manual.
CMU_ClkDiv_TypeDef	[in]	div	The clock divisor to use.

CMU_ClockEnable

```
void CMU_ClockEnable (CMU_Clock_TypeDef clock, bool enable)
```

Enable/disable a clock.

Parameters

Type	Direction	Argument Name	Description
CMU_Clock_TypeDef	[in]	clock	The clock to enable/disable.
bool	[in]	enable	- true - enable specified clock. - false - disable specified clock.

Module clocks are disabled after reset. If a module clock is disabled, the registers of that module are not accessible and accessing such registers will hardfault the Cortex core.

CMU_ClockFreqGet

```
uint32_t CMU_ClockFreqGet (CMU_Clock_TypeDef clock)
```

Get clock frequency for a clock point.

Parameters

Type	Direction	Argument Name	Description
CMU_Clock_TypeDef	[in]	clock	Clock point to fetch frequency for.

Returns

- The current frequency in Hz.

CMU_ClockSelectGet

```
CMU_Select_TypeDef CMU_ClockSelectGet (CMU_Clock_TypeDef clock)
```

Get currently selected reference clock used for a clock branch.

Parameters

Type	Direction	Argument Name	Description
CMU_Clock_TypeDef	[in]	clock	Clock branch to fetch selected ref. clock for.

Returns

- Reference clock used for clocking selected branch, [cmuSelect_Error](#) if invalid `clock` provided.

CMU_ClockSelectSet

```
void CMU_ClockSelectSet (CMU\_Clock\_TypeDef clock, CMU\_Select\_TypeDef ref)
```

Select reference clock/oscillator used for a clock branch.

Parameters

Type	Direction	Argument Name	Description
CMU_Clock_TypeDef	[in]	clock	Clock branch to select reference clock for.
CMU_Select_TypeDef	[in]	ref	Reference selected for clocking, please refer to reference manual for details on which reference is available for a specific clock branch.

CMU_LF_ClockPrecisionGet

```
uint16_t CMU_LF_ClockPrecisionGet (CMU\_Clock\_TypeDef clock)
```

Gets the precision (in PPM) of the specified low frequency clock branch.

Parameters

Type	Direction	Argument Name	Description
CMU_Clock_TypeDef	[in]	clock	Clock branch.

Returns

- Precision, in PPM, of the specified clock branch.

Note

- This function is only for internal usage.
- The current implementation of this function is used to determine if the clock has a precision ≤ 500 ppm or not (which is the minimum required for BLE). Future version of this function should provide more accurate precision numbers to allow for further optimizations from the stacks.

CMU_HF_ClockPrecisionGet

```
uint16_t CMU_HF_ClockPrecisionGet (CMU\_Clock\_TypeDef clock)
```

Gets the precision (in PPM) of the specified high frequency clock branch.

Parameters

Type	Direction	Argument Name	Description
CMU_Clock_TypeDef	[in]	clock	Clock branch.

Returns

- Precision, in PPM, of the specified clock branch.

Note

- This function is only for internal usage.
- The current implementation of this function is used to determine if the clock has a precision ≤ 500 ppm or not (which is the minimum required for BLE). Future version of this function should provide more accurate precision numbers to allow for further optimizations from the stacks.

CMU_HFRCODPLLBandGet

```
CMU_HFRCODPLLFreq_TypeDef CMU_HFRCODPLLBandGet (void )
```

Get HFRCODPLL band in use.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- HFRCODPLL band in use.

CMU_HFRCODPLLBandSet

```
void CMU_HFRCODPLLBandSet (CMU_HFRCODPLLFreq_TypeDef freq)
```

Set HFRCODPLL band and the tuning value based on the value in the calibration table made during production.

Parameters

Type	Direction	Argument Name	Description
CMU_HFRCODPLLFreq_TypeDef	[in]	freq	HFRCODPLL frequency band to activate.

CMU_DPLLLock

```
bool CMU_DPLLLock (const CMU_DPLLInit_TypeDef * init)
```

Lock the DPLL to a given frequency.

Parameters

Type	Direction	Argument Name	Description
const CMU_DPLLInit_TypeDef *	[in]	init	DPLL setup parameter struct.

The frequency is given by: $F_{out} = F_{ref} * (N+1) / (M+1)$.

Note

- This function does not check if the given N & M values will actually produce the desired target frequency.
N & M limitations:
 $300 < N \leq 4095$
 $0 \leq M \leq 4095$
Any peripheral running off HFRCODPLL should be switched to a lower frequency clock (if possible) prior to calling this function to avoid over-clocking.

Returns

- Returns false on invalid target frequency or DPLL locking error.

CMU_HFXOInit

```
void CMU_HFXOInit (const CMU_HFXOInit_TypeDef * hfxoInit)
```

Initialize all HFXO control registers.

Parameters

Type	Direction	Argument Name	Description
const CMU_HFXOInit_TypeDef *	[in]	hfxoInit	HFXO setup parameters.

Note

- HFXO configuration should be obtained from a configuration tool, app note or crystal datasheet. This function returns early if HFXO is already selected as SYSCLK.

CMU_HFXOCTuneSet

```
sl_status_t CMU_HFXOCTuneSet (uint32_t ctune)
```

Set the HFXO crystal tuning capacitance.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	ctune	The desired tuning capacitance value. Each step corresponds to approximately 80fF. Min value is 0. Max value is 255.

Returns

- SL_STATUS_OK if initialization parameter is valid. SL_STATUS_INVALID_PARAMETER if initialization parameter is invalid.

Note

- While the oscillator is running in steady operation state, it may be desirable to modify the tuning capacitance via CTUNEXIANA and CTUNEXOANA fields in the HFXO_XTALCTRL register. When tuning, care should be taken to make small changes to the CTUNE registers. Ideally, change the CTUNE registers by one LSB at a time and alternate between the XI and XO registers. Sufficient wait time for settling, on the order of TIMEOUTSTEADY, should pass before new frequency measurement is taken.

CMU_HFXOCTuneGet

```
uint32_t CMU_HFXOCTuneGet (void )
```

Get the HFXO crystal tuning capacitance.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The HFXO crystal tuning capacitance.

Note

- This function only returns the CTUNE XI value. The XO value can be different and can be found using the delta (difference between XI and XO). See [CMU_HFXOCTuneDeltaGet](#) to retrieve the delta value.

CMU_HFXOCTuneDeltaSet

```
void CMU_HFXOCTuneDeltaSet (int32_t delta)
```

Set the HFXO crystal tuning delta.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	delta	Chip dependent crystal capacitor bank delta between HFXO XI and XO.

Note

- The delta between XI and XO is applicable for the series 2 EFR32xG2x devices only.

CMU_HFXOCTuneDeltaGet

```
int32_t CMU_HFXOCTuneDeltaGet (void )
```

Get the HFXO crystal tuning delta.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Chip dependent crystal capacitor bank tuning delta.

CMU_HFXOCoreBiasCurrentCalibrate

```
void CMU_HFXOCoreBiasCurrentCalibrate (void )
```

Recalibrate the HFXO's Core Bias Current.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Care should be taken when using this function as it can cause disturbance on the HFXO frequency while the optimization is underway. It's recommended to only use this function when HFXO isn't being used. It's also a blocking function that can be time consuming.

CMU_LFXOInit

```
void CMU_LFXOInit (const CMU_LFXOInit_TypeDef * lfxoInit)
```

Initialize LFXO control registers.

Parameters

Type	Direction	Argument Name	Description
const CMU_LFXOInit_TypeDef *	[in]	lfxoInit	LFXO setup parameters

Note

- LFXO configuration should be obtained from a configuration tool, app note or crystal datasheet. This function disables the LFXO to ensure a valid state before update.

CMU_LFXOPrecisionSet

```
void CMU_LFXOPrecisionSet (uint16_t precision)
```

Sets LFXO's crystal precision, in PPM.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	precision	LFXO's crystal precision, in PPM.

Note

- LFXO precision should be obtained from a crystal datasheet.

CMU_LFXOPrecisionGet

```
uint16_t CMU_LFXOPrecisionGet (void )
```

Gets LFXO's crystal precision, in PPM.

Parameters

Type	Direction	Argument Name	Description
void	[in]		LFXO's crystal precision, in PPM.

CMU_HFXOPrecisionSet

```
void CMU_HFXOPrecisionSet (uint16_t precision)
```

Sets HFXO's crystal precision, in PPM.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	precision	HFXO's crystal precision, in PPM.

Note

HFXO precision should be obtained from a crystal datasheet.

CMU_HFXOPrecisionGet

```
uint16_t CMU_HFXOPrecisionGet (void )
```

Gets HFXO's crystal precision, in PPM.

Parameters

Type	Direction	Argument Name	Description
void	[in]		HFXO's crystal precision, in PPM.

CMU_OscillatorTuningGet

```
uint32_t CMU_OscillatorTuningGet (CMU_Osc_TypeDef osc)
```

Get oscillator frequency tuning setting.

Parameters

Type	Direction	Argument Name	Description
CMU_Osc_TypeDef	[in]	osc	Oscillator to get tuning value for.

Returns

- The oscillator frequency tuning setting in use.

CMU_OscillatorTuningSet

```
void CMU_OscillatorTuningSet (CMU_Osc_TypeDef osc, uint32_t val)
```

Set the oscillator frequency tuning control.

Parameters

Type	Direction	Argument Name	Description
CMU_Osc_TypeDef	[in]	osc	Oscillator to set tuning value for.
uint32_t	[in]	val	The oscillator frequency tuning setting to use.

Note

- Oscillator tuning is done during production, and the tuning value is automatically loaded after a reset. Changing the tuning value from the calibrated value is for more advanced use. Certain oscillators also have build-in tuning optimization.

CMU_UpdateWaitStates

```
void CMU_UpdateWaitStates (uint32_t freq, int vscale)
```

Configure wait state settings necessary to switch to a given core clock frequency at a certain voltage scale level.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	freq	The core clock frequency to configure wait-states.
int	[in]	vscale	The voltage scale to configure wait-states. Expected values are 0 or 1, higher number is lower voltage. • 0 = 1.1 V (VSCALE2) • 1 = 1.0 V (VSCALE1)

This function will set up the necessary flash wait states. Updating the wait state configuration must be done before increasing the clock frequency and it must be done after decreasing the clock frequency. Updating the wait state configuration must be done before core voltage is decreased and it must be done after a core voltage is increased.

CMU_PCNTClockExternalSet

```
void CMU_PCNTClockExternalSet (unsigned int instance, bool external)
```

Select the PCNTn clock.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	instance	PCNT instance number to set selected clock source for.
bool	[in]	external	Set to true to select the external clock, false to select EM23GRPACLK.

CMU_CalibrateCont

```
void CMU_CalibrateCont (bool enable)
```

Configure continuous calibration mode.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	If true, enables continuous calibration, if false disables continuous calibration.

CMU_CalibrateStart

```
void CMU_CalibrateStart (void )
```

Start calibration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This call is usually invoked after [CMU_CalibrateConfig\(\)](#) and possibly [CMU_CalibrateCont\(\)](#).

CMU_CalibrateStop

```
void CMU_CalibrateStop (void )
```

Stop calibration counters.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

CMU_DPLLUnlock

```
void CMU_DPLLUnlock (void )
```

Unlock the DPLL.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- The HFRCODPLL oscillator is not turned off.

CMU_IntClear

```
void CMU_IntClear (uint32_t flags)
```

Clear one or more pending CMU interrupt flags.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	CMU interrupt sources to clear.

CMU_IntDisable

```
void CMU_IntDisable (uint32_t flags)
```

Disable one or more CMU interrupt sources.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	CMU interrupt sources to disable.

CMU_IntEnable

```
void CMU_IntEnable (uint32_t flags)
```

Enable one or more CMU interrupt sources.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	CMU interrupt sources to enable.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. Consider using [CMU_IntClear\(\)](#) prior to enabling if such a pending interrupt should be ignored.

CMU_IntGet

```
uint32_t CMU_IntGet (void )
```

Get pending CMU interrupt sources.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- CMU interrupt sources pending.

CMU_IntGetEnabled

```
uint32_t CMU_IntGetEnabled (void )
```

Get enabled and pending CMU interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Useful for handling more interrupt sources in the same interrupt handler.

Note

- The event bits are not cleared by the use of this function.

Returns

- Pending and enabled CMU interrupt sources. The return value is the bitwise AND of
 - the enabled interrupt sources in CMU_IEN and
 - the pending interrupt flags CMU_IF

CMU_IntSet

```
void CMU_IntSet (uint32_t flags)
```

Set one or more pending CMU interrupt sources.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	CMU interrupt sources to set to pending.

CMU_Lock

```
void CMU_Lock (void )
```

Lock CMU register access in order to protect registers contents against unintended modification.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

See the reference manual for CMU registers that will be locked.

Note

- If locking the CMU registers, they must be unlocked prior to using any CMU API functions modifying CMU registers protected by the lock.

CMU_OscillatorEnable

```
void CMU_OscillatorEnable (CMU_Osc_TypeDef osc, bool enable, bool wait)
```

Enable/disable oscillator.

Parameters

Type	Direction	Argument Name	Description
CMU_Osc_TypeDef	[in]	osc	The oscillator to enable/disable.
bool	[in]	enable	• true - enable specified oscillator. • false - disable specified oscillator.
bool	[in]	wait	Only used if <code>enable</code> is true. • true - wait for oscillator start-up time to timeout before returning. • false - do not wait for oscillator start-up time to timeout before returning.

Note

- This is a dummy function to solve backward compatibility issues.

CMU_Unlock

```
void CMU_Unlock (void )
```

Unlock CMU register access so that writing to registers is possible.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

CMU_WdogLock

```
void CMU_WdogLock (void )
```

Lock WDOG register access in order to protect registers contents against unintended modification.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- If locking the WDOG registers, they must be unlocked prior to using any emlib API functions modifying registers protected by the lock.

CMU_WdogUnlock

```
void CMU_WdogUnlock (void )
```

Unlock WDOG register access so that writing to registers is possible.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

CMU_PrescToLog2

```
uint32_t CMU_PrescToLog2 (uint32_t presc)
```

Convert prescaler divider to a logarithmic value.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	presc	Prescaler value used to set the frequency divider. The divider is equal to ('presc' + 1). If a divider value is passed for 'presc', 'presc' will be equal to (divider - 1).

It only works for even numbers equal to 2^n .

Returns

- Logarithm base 2 (binary) value, i.e. exponent as used by fixed 2^n prescalers.

Macro Definition Documentation

CMU_CLOCK_SELECT_SET

```
#define CMU_CLOCK_SELECT_SET
```

Value:

```
(clock, sel)
```

Macro to set clock sources in the clock tree.

_CMU_EM01GRPACLKCTRL_CLKSEL_DISABLED

```
#define _CMU_EM01GRPACLKCTRL_CLKSEL_DISABLED
```

Value:

```
0x00000000UL
```

Disable clocks configuration.

Mode DISABLED for CMU_EM01GRPACLKCTRL

CMU_EM01GRPACLKCTRL_CLKSEL_DISABLED

```
#define CMU_EM01GRPACLKCTRL_CLKSEL_DISABLED
```

Value:

```
(_CMU_EM01GRPACLKCTRL_CLKSEL_DISABLED << 0)
```

Shifted mode DISABLED for CMU_EM01GRPACLKCTRL.

_CMU_EM01GRPBCLKCTRL_CLKSEL_DISABLED

```
#define _CMU_EM01GRPBCLKCTRL_CLKSEL_DISABLED
```

Value:

```
0x00000000UL
```

Mode DISABLED for CMU_EM01GRPBCLKCTRL

CMU_EM01GRPBCLKCTRL_CLKSEL_DISABLED

```
#define CMU_EM01GRPBCLKCTRL_CLKSEL_DISABLED
```

Value:

```
(_CMU_EM01GRPBCLKCTRL_CLKSEL_DISABLED << 0)
```

Shifted mode DISABLED for CMU_EM01GRPBCLKCTRL.

_CMU_EM23GRPACLKCTRL_CLKSEL_DISABLED

```
#define _CMU_EM23GRPACLKCTRL_CLKSEL_DISABLED
```

Value:

```
0x00000000UL
```

Mode DISABLED for CMU_EM23GRPACLKCTRL

CMU_EM23GRPACLKCTRL_CLKSEL_DISABLED

```
#define CMU_EM23GRPACLKCTRL_CLKSEL_DISABLED
```

Value:

```
(_CMU_EM23GRPACLKCTRL_CLKSEL_DISABLED << 0)
```

Shifted mode DISABLED for CMU_EM23GRPACLKCTRL.

_CMU_EM4GRPACLKCTRL_CLKSEL_DISABLED

```
#define _CMU_EM4GRPACLKCTRL_CLKSEL_DISABLED
```

Value:

```
0x00000000UL
```

Mode DISABLED for CMU_EM4GRPACLKCTRL

CMU_EM4GRPACLKCTRL_CLKSEL_DISABLED

```
#define CMU_EM4GRPACLKCTRL_CLKSEL_DISABLED
```

Value:

```
(_CMU_EM4GRPACLKCTRL_CLKSEL_DISABLED << 0)
```

Shifted mode DISABLED for CMU_EM4GRPACLKCTRL.

_CMU_WDOG0CLKCTRL_CLKSEL_DISABLED

```
#define _CMU_WDOG0CLKCTRL_CLKSEL_DISABLED
```

Value:

```
0x00000000UL
```

Mode DISABLED for CMU_WDOG0CLKCTRL

CMU_WDOG0CLKCTRL_CLKSEL_DISABLED

```
#define CMU_WDOG0CLKCTRL_CLKSEL_DISABLED
```

Value:

```
( _CMU_WDOG0CLKCTRL_CLKSEL_DISABLED << 0 )
```

Shifted mode DISABLED for CMU_WDOG0CLKCTRL

_CMU_WDOG1CLKCTRL_CLKSEL_DISABLED

```
#define _CMU_WDOG1CLKCTRL_CLKSEL_DISABLED
```

Value:

```
0x00000000UL
```

Mode DISABLED for CMU_WDOG1CLKCTRL

CMU_WDOG1CLKCTRL_CLKSEL_DISABLED

```
#define CMU_WDOG1CLKCTRL_CLKSEL_DISABLED
```

Value:

```
( _CMU_WDOG1CLKCTRL_CLKSEL_DISABLED << 0 )
```

Shifted mode DISABLED for CMU_WDOG1CLKCTRL

_CMU_EUSART0CLKCTRL_CLKSEL_DISABLED

```
#define _CMU_EUSART0CLKCTRL_CLKSEL_DISABLED
```

Value:

```
0x00000000UL
```

Mode DISABLED for CMU_EUSART0CLKCTRL

CMU_EUSART0CLKCTRL_CLKSEL_DISABLED

```
#define CMU_EUSART0CLKCTRL_CLKSEL_DISABLED
```

Value:

```
( _CMU_EUSART0CLKCTRL_CLKSEL_DISABLED << 0 )
```

Shifted mode DISABLED for CMU_EUSART0CLKCTRL.

_CMU_SYSRTC0CLKCTRL_CLKSEL_DISABLED

```
#define _CMU_SYSRTC0CLKCTRL_CLKSEL_DISABLED
```

Value:

```
0x00000000UL
```

Mode DISABLED for CMU_SYSRTC0CLKCTRL

CMU_SYSRTC0CLKCTRL_CLKSEL_DISABLED

```
#define CMU_SYSRTC0CLKCTRL_CLKSEL_DISABLED
```

Value:

```
(_CMU_SYSRTC0CLKCTRL_CLKSEL_DISABLED << 0)
```

Shifted mode DISABLED for CMU_SYSRTC0CLKCTRL.

CMU_HFRCODPLL_MIN

```
#define CMU_HFRCODPLL_MIN
```

Value:

```
cmuHFRCODPLLFreq_1M0Hz
```

HFRCODPLL maximum frequency.

CMU_HFRCODPLL_MAX

```
#define CMU_HFRCODPLL_MAX
```

Value:

```
cmuHFRCODPLLFreq_80M0Hz
```

HFRCODPLL minimum frequency.

CMU_LFXOINIT_DEFAULT

```
#define CMU_LFXOINIT_DEFAULT
```

Value:

```
0 | {
0 |     1,
0 |     38,
0 |     cmuLfxoStartupDelay_4KCycles,
0 |     cmuLfxoOscMode_Crystal,
0 |     false,
0 |     /* highAmplitudeEn */
0 |     true, /* agcEn */
0 |     false, /* failDetEM4WUEn */
0 |     false, /* failDetEn */
0 |     false, /* DisOndemand */
0 |     false, /* ForceEn */
0 |     false /* Lock registers */
0 | }
```

Default LFXO initialization values for XTAL mode.

CMU_LFXOINIT_EXTERNAL_CLOCK

```
#define CMU_LFXOINIT_EXTERNAL_CLOCK
```

Value:

```
0  {
0  0U,
0  0U,
0  cmuLfxoStartupDelay_2Cycles,
0  cmuLfxoOscMode_External,
0  false,
0  false,
0  false,
0  false,
0  false,
0  false,
0  false,
0  }
/* highAmplitudeEn */ \
/* agcEn */ \
/* failDetEM4WUEn */ \
/* failDetEn */ \
/* DisOndemand */ \
/* ForceEn */ \
/* Lock registers */ \
```

Default LFXO initialization values for external clock mode.

CMU_LFXOINIT_EXTERNAL_SINE

```
#define CMU_LFXOINIT_EXTERNAL_SINE
```

Value:

```
0  {
0  0U,
0  0U,
0  cmuLfxoStartupDelay_2Cycles,
0  cmuLfxoOscMode_AcCoupledSine,
0  false,
0  false,
0  false,
0  false,
0  false,
0  false,
0  false,
0  }
/* highAmplitudeEn */ \
/* agcEn */ \
/* failDetEM4WUEn */ \
/* failDetEn */ \
/* DisOndemand */ \
/* ForceEn */ \
/* Lock registers */ \
```

Default LFXO initialization values for external sine mode.

CMU_HFXOINIT_DEFAULT

```
#define CMU_HFXOINIT_DEFAULT
```

Value:

```
0  {
0  cmuHfxoCbLsbTimeout_416us,
0  cmuHfxoSteadyStateTimeout_833us, /* First lock */ \
0  cmuHfxoSteadyStateTimeout_83us, /* Subsequent locks */ \
0  0U, /* ctuneXoStartup */ \
0  0U, /* ctuneXiStartup */ \
0  32U, /* coreBiasStartup */ \
0  32U, /* imCoreBiasStartup */ \
0  cmuHfxoCoreDegen_None,
0  cmuHfxoCtuneFixCap_Both,
0  _HFXO_XTALCTRL_CTUNEXOANA_DEFAULT, /* ctuneXoAna */ \
0  _HFXO_XTALCTRL_CTUNEXIANA_DEFAULT, /* ctuneXiAna */ \
0  60U, /* coreBiasAna */ \
0  false, /* enXiDcBiasAna */ \
0  cmuHfxoOscMode_Crystal,
0  false, /* forceXo2GndAna */ \
0  false, /* forceXi2GndAna */ \
0  false, /* DisOndemand */ \
0  false, /* ForceEn */ \
```

```
0 | false          /* Lock registers          */ \
0 | }

```

Default HFXO initialization values for XTAL mode.

CMU_HFXOINIT_EXTERNAL_SINE

```
#define CMU_HFXOINIT_EXTERNAL_SINE

```

Value:

```
0 | {
0 | (CMU_HfxoCbLsbTimeout_TypeDef)0,      /* timeoutCbLsb          */ \
0 | (CMU_HfxoSteadyStateTimeout_TypeDef)0, /* timeoutSteady, first lock */ \
0 | (CMU_HfxoSteadyStateTimeout_TypeDef)0, /* timeoutSteady, subseq. locks */ \
0 | 0U,                                   /* ctuneXoStartup        */ \
0 | 0U,                                   /* ctuneXiStartup        */ \
0 | 0U,                                   /* coreBiasStartup       */ \
0 | 0U,                                   /* imCoreBiasStartup     */ \
0 | cmuHfxoCoreDegen_None,               \
0 | cmuHfxoCtuneFixCap_None,            \
0 | 0U,                                   /* ctuneXoAna           */ \
0 | 0U,                                   /* ctuneXiAna           */ \
0 | 0U,                                   /* coreBiasAna          */ \
0 | false, /* enXiDcBiasAna, false=DC true=AC coupling of signal */ \
0 | cmuHfxoOscMode_ExternalSine,        \
0 | false,                               /* forceXo2GndAna       */ \
0 | false,                               /* forceXi2GndAna (Never enable in sine mode) */ \
0 | false,                               /* DisOndemand          */ \
0 | false,                               /* ForceEn              */ \
0 | false,                               /* Lock registers       */ \
0 | }

```

Default HFXO initialization values for external sine mode.

CMU_HFXOINIT_EXTERNAL_SINEPKDET

```
#define CMU_HFXOINIT_EXTERNAL_SINEPKDET

```

Value:

```
0 | {
0 | (CMU_HfxoCbLsbTimeout_TypeDef)0,      /* timeoutCbLsb          */ \
0 | (CMU_HfxoSteadyStateTimeout_TypeDef)0, /* timeoutSteady, first lock */ \
0 | (CMU_HfxoSteadyStateTimeout_TypeDef)0, /* timeoutSteady, subseq. locks */ \
0 | 0U,                                   /* ctuneXoStartup        */ \
0 | 0U,                                   /* ctuneXiStartup        */ \
0 | 0U,                                   /* coreBiasStartup       */ \
0 | 0U,                                   /* imCoreBiasStartup     */ \
0 | cmuHfxoCoreDegen_None,               \
0 | cmuHfxoCtuneFixCap_None,            \
0 | 0U,                                   /* ctuneXoAna           */ \
0 | 0U,                                   /* ctuneXiAna           */ \
0 | 0U,                                   /* coreBiasAna          */ \
0 | false, /* enXiDcBiasAna, false=DC true=AC coupling of signal */ \
0 | cmuHfxoOscMode_ExternalSinePkDet,    \
0 | false,                               /* forceXo2GndAna       */ \
0 | false,                               /* forceXi2GndAna (Never enable in sine mode) */ \
0 | false,                               /* DisOndemand          */ \
0 | false,                               /* ForceEn              */ \
0 | false,                               /* Lock registers       */ \
0 | }

```

Default HFXO initialization values for external sine mode with peak detector.

CMU_DPLL_LFXO_TO_40MHZ

```
#define CMU_DPLL_LFXO_TO_40MHZ

```

Value:

```

0 | {
0 |     39998805,          /* Target frequency.          */ \
0 |     3661,              /* Factor N.                  */ \
0 |     2,                 /* Factor M.                  */ \
0 |     cmuSelect_LFXO,    /* Select LFXO as reference clock. */ \
0 |     cmuDPLLEdgeSel_Fall, /* Select falling edge of ref clock. */ \
0 |     cmuDPLLockMode_Freq, /* Use frequency lock mode.    */ \
0 |     true,              /* Enable automatic lock recovery. */ \
0 |     false              /* Don't enable dither function. */ \
0 | }

```

DPLL initialization values for 39,998,805 Hz using LFXO as reference clock, M=2 and N=3661.

CMU_DPLL_HFXO_TO_76_8MHZ

```
#define CMU_DPLL_HFXO_TO_76_8MHZ
```

Value:

```

0 | {
0 |     76800000,          /* Target frequency.          */ \
0 |     3839,              /* Factor N.                  */ \
0 |     1919,              /* Factor M.                  */ \
0 |     cmuSelect_HFXO,    /* Select HFXO as reference clock. */ \
0 |     cmuDPLLEdgeSel_Fall, /* Select falling edge of ref clock. */ \
0 |     cmuDPLLockMode_Freq, /* Use frequency lock mode.    */ \
0 |     true,              /* Enable automatic lock recovery. */ \
0 |     false              /* Don't enable dither function. */ \
0 | }

```

DPLL initialization values for 76,800,000 Hz using HFXO as reference clock, M = 1919, N = 3839.

CMU_DPLL_HFXO_TO_80MHZ

```
#define CMU_DPLL_HFXO_TO_80MHZ
```

Value:

```

0 | {
0 |     80000000,          /* Target frequency.          */ \
0 |     (4000 - 1),        /* Factor N.                  */ \
0 |     (1920 - 1),        /* Factor M.                  */ \
0 |     cmuSelect_HFXO,    /* Select HFXO as reference clock. */ \
0 |     cmuDPLLEdgeSel_Fall, /* Select falling edge of ref clock. */ \
0 |     cmuDPLLockMode_Freq, /* Use frequency lock mode.    */ \
0 |     true,              /* Enable automatic lock recovery. */ \
0 |     false              /* Don't enable dither function. */ \
0 | }

```

DPLL initialization values for 80,000,000 Hz using HFXO as reference clock, M = 1919, N = 3999.

CMU_DPLLINIT_DEFAULT

```
#define CMU_DPLLINIT_DEFAULT
```

Value:

```

0 | {
0 |     80000000,          /* Target frequency.          */ \
0 |     (4000 - 1),        /* Factor N.                  */ \
0 |     (1920 - 1),        /* Factor M.                  */ \
0 |     cmuSelect_HFXO,    /* Select HFXO as reference clock. */ \
0 | }

```

```
0 | cmuDPLLEdgeSel_Fall, /* Select falling edge of ref clock. */ \
0 | cmuDPLLLockMode_Freq, /* Use frequency lock mode. */ \
0 | true, /* Enable automatic lock recovery. */ \
0 | false /* Don't enable dither function. */ \
0 | }
```

Default configurations for DPLL initialization.

When using this macro you need to modify the N and M factor and the desired frequency to match the components placed on the board.

CMU_LFXOInit_TypeDef

LFXO initialization structure.

Initialization values should be obtained from a configuration tool, application note or crystal data sheet.

Public Attributes

uint8_t	gain	Startup gain.
uint8_t	capTune	Internal capacitance tuning.
CMU_LfxoStart upDelay_TypeDef	timeout	Startup delay.
CMU_LfxoOscM ode_TypeDef	mode	Oscillator mode.
bool	highAmplitudeEn	High amplitude enable.
bool	agcEn	AGC enable.
bool	failDetEM4WUEn	EM4 wakeup on failure enable.
bool	failDetEn	Oscillator failure detection enable.
bool	disOnDemand	Disable on-demand requests.
bool	forceEn	Force oscillator enable.
bool	regLock	Lock register access.

Public Attribute Documentation

gain

uint8_t CMU_LFXOInit_TypeDef::gain

Startup gain.

capTune

uint8_t CMU_LFXOInit_TypeDef::capTune

Internal capacitance tuning.

timeout

```
CMU_LfxoStartupDelay_TypeDef CMU_LFXOInit_TypeDef::timeout
```

Startup delay.

mode

```
CMU_LfxoOscMode_TypeDef CMU_LFXOInit_TypeDef::mode
```

Oscillator mode.

highAmplitudeEn

```
bool CMU_LFXOInit_TypeDef::highAmplitudeEn
```

High amplitude enable.

agcEn

```
bool CMU_LFXOInit_TypeDef::agcEn
```

AGC enable.

failDetEM4WUEn

```
bool CMU_LFXOInit_TypeDef::failDetEM4WUEn
```

EM4 wakeup on failure enable.

failDetEn

```
bool CMU_LFXOInit_TypeDef::failDetEn
```

Oscillator failure detection enable.

disOnDemand

```
bool CMU_LFXOInit_TypeDef::disOnDemand
```

Disable on-demand requests.

forceEn

```
bool CMU_LFXOInit_TypeDef::forceEn
```

Force oscillator enable.

regLock

```
bool CMU_LFXOInit_TypeDef::regLock
```

Lock register access.

CMU_HFXOInit_TypeDef

HFXO initialization structure.

Initialization values should be obtained from a configuration tool, application note or crystal data sheet.

Public Attributes

CMU_HfxoCbLsbTimeout_TypeDef	timeoutCbLsb	Core bias change timeout.
CMU_HfxoSteadyStateTimeout_TypeDef	timeoutSteadyFirstLock	Steady state timeout duration for first lock.
CMU_HfxoSteadyStateTimeout_TypeDef	timeoutSteady	Steady state timeout duration.
uint8_t	ctuneXoStartup	XO pin startup tuning capacitance.
uint8_t	ctuneXiStartup	XI pin startup tuning capacitance.
uint8_t	coreBiasStartup	Core bias startup current.
uint8_t	imCoreBiasStartup	Core bias intermediate startup current.
CMU_HfxoCoreDegen_TypeDef	coreDegenAna	Core degeneration control.
CMU_HfxoCtuneFixCap_TypeDef	ctuneFixAna	Fixed tuning capacitance on XI/XO.
uint8_t	ctuneXoAna	Tuning capacitance on XO.
uint8_t	ctuneXiAna	Tuning capacitance on XI.
uint8_t	coreBiasAna	Core bias current.
bool	enXiDcBiasAna	Enable XI internal DC bias.
CMU_HfxoOscMode_TypeDef	mode	Oscillator mode.
bool	forceXo2GndAna	Force XO pin to ground.
bool	forceXi2GndAna	Force XI pin to ground.
bool	disOnDemand	Disable on-demand requests.
bool	forceEn	Force oscillator enable.
bool	regLock	

Lock register
access.

Public Attribute Documentation

timeoutCbLsb

```
CMU_HfxoCbLsbTimeout_TypeDef CMU_HFXOInit_TypeDef::timeoutCbLsb
```

Core bias change timeout.

timeoutSteadyFirstLock

```
CMU_HfxoSteadyStateTimeout_TypeDef CMU_HFXOInit_TypeDef::timeoutSteadyFirstLock
```

Steady state timeout duration for first lock.

timeoutSteady

```
CMU_HfxoSteadyStateTimeout_TypeDef CMU_HFXOInit_TypeDef::timeoutSteady
```

Steady state timeout duration.

ctuneXoStartup

```
uint8_t CMU_HFXOInit_TypeDef::ctuneXoStartup
```

XO pin startup tuning capacitance.

ctuneXiStartup

```
uint8_t CMU_HFXOInit_TypeDef::ctuneXiStartup
```

XI pin startup tuning capacitance.

coreBiasStartup

```
uint8_t CMU_HFXOInit_TypeDef::coreBiasStartup
```

Core bias startup current.

imCoreBiasStartup

```
uint8_t CMU_HFXOInit_TypeDef::imCoreBiasStartup
```

Core bias intermediate startup current.

coreDegenAna

```
CMU_HfxoCoreDegen_TypeDef CMU_HFXOInit_TypeDef::coreDegenAna
```

Core degeneration control.

ctuneFixAna

```
CMU_HfxoCtuneFixCap_TypeDef CMU_HFXOInit_TypeDef::ctuneFixAna
```

Fixed tuning capacitance on XI/XO.

ctuneXoAna

```
uint8_t CMU_HFXOInit_TypeDef::ctuneXoAna
```

Tuning capacitance on XO.

ctuneXiAna

```
uint8_t CMU_HFXOInit_TypeDef::ctuneXiAna
```

Tuning capacitance on XI.

coreBiasAna

```
uint8_t CMU_HFXOInit_TypeDef::coreBiasAna
```

Core bias current.

enXiDcBiasAna

```
bool CMU_HFXOInit_TypeDef::enXiDcBiasAna
```

Enable XI internal DC bias.

mode

```
CMU_HfxoOscMode_TypeDef CMU_HFXOInit_TypeDef::mode
```

Oscillator mode.

forceXo2GndAna

```
bool CMU_HFXOInit_TypeDef::forceXo2GndAna
```

Force XO pin to ground.

forceXi2GndAna

```
bool CMU_HFXOInit_TypeDef::forceXi2GndAna
```

Force XI pin to ground.

disOnDemand

```
bool CMU_HFXOInit_TypeDef::disOnDemand
```

Disable on-demand requests.

forceEn

```
bool CMU_HFXOInit_TypeDef::forceEn
```

Force oscillator enable.

regLock

```
bool CMU_HFXOInit_TypeDef::regLock
```

Lock register access.

CMU_DPLLInit_TypeDef

DPLL initialization structure.

Frequency will be $F_{ref} * (N+1) / (M+1)$.

Public Attributes

uint32_t	frequency	PLL frequency value, max 80 MHz.
uint16_t	n	Factor N.
uint16_t	m	Factor M.
CMU_Select_TypeDef	refClk	Reference clock selector.
CMU_DPLLEdgeSel_TypeDef	edgeSel	Reference clock edge detect selector.
CMU_DPLLLockMode_TypeDef	lockMode	DPLL lock mode selector.
bool	autoRecover	Enable automatic lock recovery.
bool	ditherEn	Enable dither functionality.

Public Attribute Documentation

frequency

```
uint32_t CMU_DPLLInit_TypeDef::frequency
```

PLL frequency value, max 80 MHz.

n

```
uint16_t CMU_DPLLInit_TypeDef::n
```

Factor N.

300 <= N <= 4095

m

```
uint16_t CMU_DPLLInit_TypeDef::m
```

Factor M.

M <= 4095

refClk

```
CMU_Select_TypeDef CMU_DPLLInit_TypeDef::refClk
```

Reference clock selector.

edgeSel

```
CMU_DPLLEdgeSel_TypeDef CMU_DPLLInit_TypeDef::edgeSel
```

Reference clock edge detect selector.

lockMode

```
CMU_DPLLLockMode_TypeDef CMU_DPLLInit_TypeDef::lockMode
```

DPLL lock mode selector.

autoRecover

```
bool CMU_DPLLInit_TypeDef::autoRecover
```

Enable automatic lock recovery.

ditherEn

```
bool CMU_DPLLInit_TypeDef::ditherEn
```

Enable dither functionality.

CORE - Core Interrupt

CORE - Core Interrupt

Introduction

CORE interrupt API provides a simple and safe means to disable and enable interrupts to protect sections of code.

This is often referred to as "critical sections". This module provides support for three types of critical sections, each with different interrupt blocking capabilities.

- **CRITICAL section:** Inside a critical section, all interrupts are disabled (except for fault handlers). The PRIMASK register is always used for interrupt disable/enable.
- **ATOMIC section:** This type of section is configurable and the default method is to use PRIMASK. With BASEPRI configuration, interrupts with priority equal to or lower than a given configurable level are disabled. The interrupt disable priority level is defined at compile time. The BASEPRI register is not available for all architectures.
- **NVIC mask section:** Disable NVIC (external interrupts) on an individual manner.

em_core also has an API for manipulating RAM-based interrupt vector tables.

Compile-time Configuration

The following #defines are used to configure em_core:

```
// The interrupt priority level used inside ATOMIC sections.
#define CORE_ATOMIC_BASE_PRIORITY_LEVEL    3

// A method used for interrupt disable/enable within ATOMIC sections.
#define CORE_ATOMIC_METHOD                CORE_ATOMIC_METHOD_PRIMASK
```

If the default values do not support your needs, they can be overridden by supplying -D compiler flags on the compiler command line or by collecting all macro redefinitions in a file named emlib_config.h and then supplying -DEMLIB_USER_CONFIG on a compiler command line.

Note

- The default emlib configuration for ATOMIC section interrupt disable method is using PRIMASK, i.e., ATOMIC sections are implemented as CRITICAL sections.
- Due to architectural limitations Cortex-M0+ devices do not support ATOMIC type critical sections using the BASEPRI register. On M0+ devices ATOMIC section helper macros are available but they are implemented as CRITICAL sections using PRIMASK register.

Macro API

The primary em_core API is the macro API. Macro API will map to correct CORE functions according to the selected [CORE_ATOMIC_METHOD](#) and similar configurations (the full CORE API is of course also available). The most useful macros are as follows:

```
CORE_DECLARE_IRQ_STATE
CORE_ENTER_ATOMIC()
```

@n Used together to implement an ATOMIC section.

```
{
    CORE_DECLARE_IRQ_STATE;           // Storage for saving IRQ state prior to
                                      // atomic section entry.

    CORE_ENTER_ATOMIC();               // Enter atomic section.

    ...
    ... your code goes here ...
    ...

    CORE_EXIT_ATOMIC();                // Exit atomic section, IRQ state is restored.
}
```

@n A concatenation of all three macros above.

```
{
    CORE_ATOMIC_SECTION(
        ...
        ... your code goes here ...
        ...
    )
}
```

CORE_DECLARE_IRQ_STATE

CORE_ENTER_CRITICAL()

CORE_EXIT_CRITICAL()

@n These macros implement CRITICAL sections in a similar fashion as described above for ATOMIC sections.

CORE_DECLARE_NVIC_STATE

CORE_ENTER_NVIC()

CORE_EXIT_NVIC()

@n These macros implement NVIC mask sections in a similar fashion as described above for ATOMIC sections. See [Examples](#) for an example.

Refer to Macros or Macro Definition Documentation below for a full list of macros.

API reimplementation

Most of the functions in the API are implemented as weak functions. This means that it is easy to reimplement when special needs arise. Shown below is a reimplementation of CRITICAL sections suitable if FreeRTOS OS is used:

```
CORE_irqState_t CORE_EnterCritical(void)
{
    vPortEnterCritical();
    return 0;
}

void CORE_ExitCritical(CORE_irqState_t irqState)
{
    (void)irqState;
    vPortExitCritical();
}
```

Also note that CORE_Enter/ExitCritical() are not implemented as inline functions. As a result, reimplementations will be possible even when original implementations are inside a linked library.

Some RTOSes must be notified on interrupt handler entry and exit. Macros [CORE_INTERRUPT_ENTRY\(\)](#) and [CORE_INTERRUPT_EXIT\(\)](#) are suitable placeholders for inserting such code. Insert these macros in all your interrupt handlers and then override the default macro implementations. This is an example if uC/OS is used:

```
// In emlib_config.h:

#define CORE_INTERRUPT_ENTRY()    OSIntEnter()
#define CORE_INTERRUPT_EXIT()    OSIntExit()
```

Interrupt vector tables

When using RAM based interrupt vector tables it is the user's responsibility to allocate the table space correctly. The tables must be aligned as specified in the CPU reference manual.

[@n](#) Initialize a RAM based vector table by copying table entries from a source vector table to a target table. VTOR is set to the address of the target vector table.

[CORE_GetNvicRamTableHandler\(\)](#)

[@n](#) Use these functions to get or set the interrupt handler for a specific IRQn. They both use the interrupt vector table defined by the current VTOR register value.

Maximum Interrupt Disabled Time

The maximum time spent (in cycles) in critical and atomic sections can be measured for performance and interrupt latency analysis. To enable the timings, use the `SL_EMLIB_CORE_ENABLE_INTERRUPT_DISABLED_TIMING` configuration option. When enabled, the functions [CORE_get_max_time_critical_section\(\)](#) and [CORE_get_max_time_atomic_section\(\)](#) can be used to get the max timings since startup.

Examples

Implement an NVIC critical section:

```
{
    CORE_DECLARE_NVIC_ZEROMASK(mask); // A zero initialized NVIC disable mask

    // Set mask bits for IRQs to block in the NVIC critical section.
    // In many cases, you can create the disable mask once upon application
    // startup and use the mask globally throughout the application lifetime.
    CORE_NvicMaskSetIRQ(LEUART0_IRQn, &mask);
    CORE_NvicMaskSetIRQ(VCMP_IRQn, &mask);

    // Enter NVIC critical section with the disable mask
    CORE_NVIC_SECTION(&mask,
        ...
        ... your code goes here ...
        ...
    )
}
```

Porting from em_int

Existing code using `INT_Enable()` and `INT_Disable()` must be ported to the `em_core` API. While `em_int` used, a global counter to store the interrupt state, `em_core` uses a local variable. Any usage of `INT_Disable()`, therefore, needs to be replaced with a declaration of the interrupt state variable before entering the critical section.

Since the state variable is in local scope, the critical section exit needs to occur within the scope of the variable. If multiple nested critical sections are used, each needs to have its own state variable in its own scope.

In many cases, completely disabling all interrupts using `CRITICAL` sections might be more heavy-handed than needed. When porting, consider whether other types of sections, such as `ATOMIC` or `NVIC` mask, can be used to only disable a subset of the interrupts.

Replacing `em_int` calls with `em_core` function calls:

```
void func(void)
{
    // INT_Disable();
    CORE_DECLARE_IRQ_STATE;
    CORE_ENTER_ATOMIC();

    .
    .
    .

    // INT_Enable();
    CORE_EXIT_ATOMIC();
}
```

Modules

[dwt_cycle_counter_handle_t](#)

[CORE_nvicMask_t](#)

Typedefs

`typedef uint32_t` [CORE_irqState_t](#)
Storage for PRIMASK or BASEPRI value.

`typedef uint32_t` [CORE_irqState_t](#)
Storage for PRIMASK or BASEPRI value.

Functions

`void` [CORE_CriticalDisableIrq\(void\)](#)
Disable interrupts.

`void` [CORE_CriticalEnableIrq\(void\)](#)
Enable interrupts.

[CORE_irqState_t](#) [CORE_EnterCritical\(void\)](#)
Enter a `CRITICAL` section.

`void` [CORE_ExitCritical\(CORE_irqState_t irqState\)](#)
Exit a `CRITICAL` section.

`void` [CORE_YieldCritical\(void\)](#)
Brief interrupt enable/disable sequence to allow handling of pending interrupts.

`void` [CORE_AtomicDisableIrq\(void\)](#)
Disable interrupts.

`void` [CORE_AtomicEnableIrq\(void\)](#)
Enable interrupts.

<code>CORE_irqState_t</code>	<code>CORE_EnterAtomic(void)</code> Enter an ATOMIC section.
<code>void</code>	<code>CORE_ExitAtomic(CORE_irqState_t irqState)</code> Exit an ATOMIC section.
<code>void</code>	<code>CORE_YieldAtomic(void)</code> Brief interrupt enable/disable sequence to allow handling of pending interrupts.
<code>void</code>	<code>CORE_EnterNvicMask(CORE_nvicMask_t *nvicState, const CORE_nvicMask_t *disable)</code> Enter a NVIC mask section.
<code>void</code>	<code>CORE_NvicDisableMask(const CORE_nvicMask_t *disable)</code> Disable NVIC interrupts.
<code>void</code>	<code>CORE_NvicEnableMask(const CORE_nvicMask_t *enable)</code> Set current NVIC interrupt enable mask.
<code>void</code>	<code>CORE_YieldNvicMask(const CORE_nvicMask_t *enable)</code> Brief NVIC interrupt enable/disable sequence to allow handling of pending interrupts.
<code>void</code>	<code>CORE_NvicMaskSetIRQ(IRQn_Type irqN, CORE_nvicMask_t *mask)</code> Utility function to set an IRQn bit in a NVIC enable/disable mask.
<code>void</code>	<code>CORE_NvicMaskClearIRQ(IRQn_Type irqN, CORE_nvicMask_t *mask)</code> Utility function to clear an IRQn bit in a NVIC enable/disable mask.
<code>bool</code>	<code>CORE_InIrqContext(void)</code> Check whether the current CPU operation mode is handler mode.
<code>bool</code>	<code>CORE_IrqIsBlocked(IRQn_Type irqN)</code> Check if a specific interrupt is disabled or blocked.
<code>bool</code>	<code>CORE_IrqIsDisabled(void)</code> Check if interrupts are disabled.
<code>void</code>	<code>CORE_GetNvicEnabledMask(CORE_nvicMask_t *mask)</code> Get the current NVIC enable mask state.
<code>bool</code>	<code>CORE_GetNvicMaskDisableState(const CORE_nvicMask_t *mask)</code> Get NVIC disable state for a given mask.
<code>bool</code>	<code>CORE_NvicIRQDisabled(IRQn_Type irqN)</code> Check if an NVIC interrupt is disabled.
<code>void *</code>	<code>CORE_GetNvicRamTableHandler(IRQn_Type irqN)</code> Utility function to get the handler for a specific interrupt.
<code>void</code>	<code>CORE_SetNvicRamTableHandler(IRQn_Type irqN, void *handler)</code> Utility function to set the handler for a specific interrupt.
<code>void</code>	<code>CORE_InitNvicVectorTable(uint32_t *sourceTable, uint32_t sourceSize, uint32_t *targetTable, uint32_t targetSize, void *defaultHandler, bool overwriteActive)</code> Initialize an interrupt vector table by copying table entries from a source to a target table.
<code>void</code>	<code>cycle_counter_start(dwt_cycle_counter_handle_t *handle)</code> Start a recording.
<code>void</code>	<code>cycle_counter_stop(dwt_cycle_counter_handle_t *handle)</code> Stop a recording.
<code>uint32_t</code>	<code>CORE_get_max_time_critical_section(void)</code> Returns the max time spent in critical section.
<code>uint32_t</code>	<code>CORE_get_max_time_atomic_section(void)</code> Returns the max time spent in atomic section.

void [CORE_clear_max_time_critical_section](#)(void)
Clears the max time spent in atomic section.

void [CORE_clear_max_time_atomic_section](#)(void)
Clears the max time spent in atomic section.

Macros

#define [CORE_INTERRUPT_ENTRY](#) ()
Placeholder for optional interrupt handler entry code.

#define [CORE_INTERRUPT_EXIT](#) ()
Placeholder for optional interrupt handler exit code.

#define [CORE_ATOMIC_METHOD_PRIMASK](#) 0
Use PRIMASK register to disable interrupts in ATOMIC sections.

#define [CORE_ATOMIC_METHOD_BASEPRI](#) 1
Use BASEPRI register to disable interrupts in ATOMIC sections.

#define [CORE_ATOMIC_BASE_PRIORITY_LEVEL](#) 3
The interrupt priority level disabled within ATOMIC regions.

#define [CORE_DECLARE_IRQ_STATE](#) CORE_irqState_t irqState
Allocate storage for PRIMASK or BASEPRI value for use by CORE_ENTER/EXIT_ATOMIC() and CORE_ENTER/EXIT_CRITICAL() macros.

#define [CORE_CRITICAL_IRQ_DISABLE](#) ()
CRITICAL style interrupt disable.

#define [CORE_CRITICAL_IRQ_ENABLE](#) ()
CRITICAL style interrupt enable.

#define [CORE_CRITICAL_SECTION](#) (yourcode)
Convenience macro for implementing a CRITICAL section.

#define [CORE_ENTER_CRITICAL](#) ()
Enter CRITICAL section.

#define [CORE_EXIT_CRITICAL](#) ()
Exit CRITICAL section.

#define [CORE_YIELD_CRITICAL](#) ()
CRITICAL style yield.

#define [CORE_ATOMIC_IRQ_DISABLE](#) ()
ATOMIC style interrupt disable.

#define [CORE_ATOMIC_IRQ_ENABLE](#) ()
ATOMIC style interrupt enable.

#define [CORE_ATOMIC_SECTION](#) (yourcode)
Convenience macro for implementing an ATOMIC section.

#define [CORE_ENTER_ATOMIC](#) ()
Enter ATOMIC section.

#define [CORE_EXIT_ATOMIC](#) ()
Exit ATOMIC section.

#define [CORE_YIELD_ATOMIC](#) ()
ATOMIC style yield.

#define [CORE_IRQ_DISABLED](#) ()
Check if IRQ is disabled.

#define [CORE_IN_IRQ_CONTEXT](#) ()
Check if inside an IRQ handler.

```

#define CORE_DECLARE_IRQ_STATE
    Allocate storage for PRIMASK or BASEPRI value for use by CORE_ENTER/EXIT_ATOMIC() and
    CORE_ENTER/EXIT_CRITICAL() macros.

#define CORE_CRITICAL_IRQ_DISABLE ()
    CRITICAL style interrupt disable.

#define CORE_CRITICAL_IRQ_ENABLE ()
    CRITICAL style interrupt enable.

#define CORE_CRITICAL_SECTION (yourcode)
    Convenience macro for implementing a CRITICAL section.

#define CORE_ENTER_CRITICAL ()
    Enter CRITICAL section.

#define CORE_EXIT_CRITICAL ()
    Exit CRITICAL section.

#define CORE_YIELD_CRITICAL ()
    CRITICAL style yield.

#define CORE_ATOMIC_IRQ_DISABLE ()
    ATOMIC style interrupt disable.

#define CORE_ATOMIC_IRQ_ENABLE ()
    ATOMIC style interrupt enable.

#define CORE_ATOMIC_SECTION (yourcode)
    Convenience macro for implementing an ATOMIC section.

#define CORE_ENTER_ATOMIC ()
    Enter ATOMIC section.

#define CORE_EXIT_ATOMIC ()
    Exit ATOMIC section.

#define CORE_YIELD_ATOMIC ()
    ATOMIC style yield.

#define CORE_NVIC_REG_WORDS ((EXT_IRQ_COUNT + 31) / 32)
    Number of words in a NVIC mask set.

#define CORE_DEFAULT_VECTOR_TABLE_ENTRIES (EXT_IRQ_COUNT + 16)
    Number of entries in a default interrupt vector table.

#define CORE_INTERRUPT_HIGHEST_PRIORITY 0
    Highest priority for core interrupt.

#define CORE_INTERRUPT_DEFAULT_PRIORITY 5
    Default priority for core interrupt.

#define CORE_INTERRUPT_LOWEST_PRIORITY 7
    Lowest priority for core interrupt.

#define CORE_ATOMIC_METHOD_DEFAULT CORE_ATOMIC_METHOD_BASEPRI
    Default method to disable interrupts in ATOMIC sections.

#define CORE_ATOMIC_METHOD CORE_ATOMIC_METHOD_PRIMASK
    Specify which method to use when implementing ATOMIC sections.

#define CORE_DECLARE_NVIC_STATE CORE_nvicMask_t nvicState
    Allocate storage for NVIC interrupt masks for use by CORE_ENTER/EXIT_NVIC() macros.

#define CORE_DECLARE_NVIC_MASK (x)
    Allocate storage for NVIC interrupt masks.

#define CORE_DECLARE_NVIC_ZEROMASK (x)
    Allocate storage for and zero initialize NVIC interrupt mask.

```

```
#define CORE_NVIC_DISABLE (mask)
NVIC mask style interrupt disable.

#define CORE_NVIC_ENABLE (mask)
NVIC mask style interrupt enable.

#define CORE_NVIC_SECTION (mask, yourcode)
Convenience macro for implementing a NVIC mask section.

#define CORE_ENTER_NVIC (disable)
Enter NVIC mask section.

#define CORE_EXIT_NVIC ()
Exit NVIC mask section.

#define CORE_YIELD_NVIC (enable)
NVIC maks style yield.
```

Typedef Documentation

CORE_irqState_t

```
typedef uint32_t CORE_irqState_t
```

Storage for PRIMASK or BASEPRI value.

CORE_irqState_t

```
typedef uint32_t CORE_irqState_t
```

Storage for PRIMASK or BASEPRI value.

Function Documentation

CORE_CriticalDisableIrq

```
void CORE_CriticalDisableIrq (void )
```

Disable interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Disable all interrupts by setting PRIMASK. (Fault exception handlers will still be enabled).

CORE_CriticalEnableIrq

```
void CORE_CriticalEnableIrq (void )
```

Enable interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Enable interrupts by clearing PRIMASK.

CORE_EnterCritical

CORE_irqState_t CORE_EnterCritical (void)

Enter a CRITICAL section.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

When a CRITICAL section is entered, all interrupts (except fault handlers) are disabled.

Returns

- The value of PRIMASK register prior to the CRITICAL section entry.

CORE_ExitCritical

void CORE_ExitCritical (CORE_irqState_t irqState)

Exit a CRITICAL section.

Parameters

Type	Direction	Argument Name	Description
CORE_irqState_t	[in]	irqState	The interrupt priority blocking level to restore to PRIMASK when exiting the CRITICAL section. This value is usually the one returned by a prior call to CORE_EnterCritical().

CORE_YieldCritical

void CORE_YieldCritical (void)

Brief interrupt enable/disable sequence to allow handling of pending interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Usually used within a CRITICAL section.

CORE_AtomicDisableIrq

void CORE_AtomicDisableIrq (void)

Disable interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Disable interrupts with a priority lower or equal to [CORE_ATOMIC_BASE_PRIORITY_LEVEL](#). Sets core BASEPRI register to CORE_ATOMIC_BASE_PRIORITY_LEVEL.

Note

- If [CORE_ATOMIC_METHOD](#) is [CORE_ATOMIC_METHOD_PRIMASK](#), this function is identical to [CORE_CriticalDisableIrq\(\)](#).

CORE_AtomicEnableIrq

```
void CORE_AtomicEnableIrq (void )
```

Enable interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Enable interrupts by setting core BASEPRI register to 0.

Note

- If [CORE_ATOMIC_METHOD](#) is [CORE_ATOMIC_METHOD_BASEPRI](#) and PRIMASK is set (CPU is inside a CRITICAL section), interrupts will still be disabled after calling this function.
- If [CORE_ATOMIC_METHOD](#) is [CORE_ATOMIC_METHOD_PRIMASK](#), this function is identical to [CORE_CriticalEnableIrq\(\)](#).

CORE_EnterAtomic

```
CORE_irqState_t CORE_EnterAtomic (void )
```

Enter an ATOMIC section.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

When an ATOMIC section is entered, interrupts with priority lower or equal to [CORE_ATOMIC_BASE_PRIORITY_LEVEL](#) are disabled.

Note

- If [CORE_ATOMIC_METHOD](#) is [CORE_ATOMIC_METHOD_PRIMASK](#), this function is identical to [CORE_EnterCritical\(\)](#).

Returns

- The value of BASEPRI register prior to ATOMIC section entry.

CORE_ExitAtomic

```
void CORE_ExitAtomic (CORE_irqState_t irqState)
```

Exit an ATOMIC section.

Parameters

Type	Direction	Argument Name	Description
CORE_irqState_t	[in]	irqState	The interrupt priority blocking level to restore to BASEPRI when exiting the ATOMIC section. This value is usually the one returned by a prior call to CORE_EnterAtomic() .

Note

- If [CORE_ATOMIC_METHOD](#) is set to [CORE_ATOMIC_METHOD_PRIMASK](#), this function is identical to [CORE_ExitCritical\(\)](#).

CORE_YieldAtomic

```
void CORE_YieldAtomic (void )
```

Brief interrupt enable/disable sequence to allow handling of pending interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Usually used within an ATOMIC section.
- If [CORE_ATOMIC_METHOD](#) is [CORE_ATOMIC_METHOD_PRIMASK](#), this function is identical to [CORE_YieldCritical\(\)](#).

CORE_EnterNvicMask

```
void CORE_EnterNvicMask (CORE_nvicMask_t * nvicState, const CORE_nvicMask_t * disable)
```

Enter a NVIC mask section.

Parameters

Type	Direction	Argument Name	Description
CORE_nvicMask_t *	[out]	nvicState	Return NVIC interrupts enable mask prior to section entry.
const CORE_nvicMask_t *	[in]	disable	A mask specifying which NVIC interrupts to disable within the section.

When a NVIC mask section is entered, specified NVIC interrupts are disabled.

CORE_NvicDisableMask

```
void CORE_NvicDisableMask (const CORE_nvicMask_t * disable)
```

Disable NVIC interrupts.

Parameters

Type	Direction	Argument Name	Description
const CORE_nvicMask_t *	[in]	disable	A mask specifying which NVIC interrupts to disable.

CORE_NvicEnableMask

```
void CORE_NvicEnableMask (const CORE\_nvicMask\_t * enable)
```

Set current NVIC interrupt enable mask.

Parameters

Type	Direction	Argument Name	Description
const CORE_nvicMask_t *	[out]	enable	A mask specifying which NVIC interrupts are currently enabled.

CORE_YieldNvicMask

```
void CORE_YieldNvicMask (const CORE\_nvicMask\_t * enable)
```

Brief NVIC interrupt enable/disable sequence to allow handling of pending interrupts.

Parameters

Type	Direction	Argument Name	Description
const CORE_nvicMask_t *	[in]	enable	A mask specifying which NVIC interrupts to briefly enable.

Note

- Usually used within an NVIC mask section.

CORE_NvicMaskSetIRQ

```
void CORE_NvicMaskSetIRQ (IRQn_Type irqN, CORE\_nvicMask\_t * mask)
```

Utility function to set an IRQn bit in a NVIC enable/disable mask.

Parameters

Type	Direction	Argument Name	Description
IRQn_Type	[in]	irqN	The IRQn_Type enumerator for the interrupt.
CORE_nvicMask_t *	[inout]	mask	The mask to set the interrupt bit in.

CORE_NvicMaskClearIRQ

```
void CORE_NvicMaskClearIRQ (IRQn_Type irqN, CORE\_nvicMask\_t * mask)
```

Utility function to clear an IRQn bit in a NVIC enable/disable mask.

Parameters

Type	Direction	Argument Name	Description
Type	Direction	Argument Name	Description
IRQn_Type	[in]	irqN	The IRQn_Type enumerator for the interrupt.
CORE_nvicMask_t *	[inout]	mask	The mask to clear the interrupt bit in.

CORE_InIrqContext

```
bool CORE_InIrqContext (void )
```

Check whether the current CPU operation mode is handler mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if the CPU is in handler mode (currently executing an interrupt handler).
False if the CPU is in thread mode.

CORE_IrqIsBlocked

```
bool CORE_IrqIsBlocked (IRQn_Type irqN)
```

Check if a specific interrupt is disabled or blocked.

Parameters

Type	Direction	Argument Name	Description
IRQn_Type	[in]	irqN	The IRQn_Type enumerator for the interrupt to check.

Returns

- True if the interrupt is disabled or blocked.

CORE_IrqIsDisabled

```
bool CORE_IrqIsDisabled (void )
```

Check if interrupts are disabled.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if interrupts are disabled.

CORE_GetNvicEnabledMask

```
void CORE_GetNvicEnabledMask (CORE_nvicMask_t * mask)
```

Get the current NVIC enable mask state.

Parameters

Type	Direction	Argument Name	Description
CORE_nvicMask_t *	[out]	mask	The current NVIC enable mask.

CORE_GetNvicMaskDisableState

```
bool CORE_GetNvicMaskDisableState (const CORE_nvicMask_t * mask)
```

Get NVIC disable state for a given mask.

Parameters

Type	Direction	Argument Name	Description
const CORE_nvicMask_t *	[in]	mask	An NVIC mask to check.

Returns

- True if all NVIC interrupt mask bits are clear.

CORE_NvicIRQDisabled

```
bool CORE_NvicIRQDisabled (IRQn_Type irqN)
```

Check if an NVIC interrupt is disabled.

Parameters

Type	Direction	Argument Name	Description
IRQn_Type	[in]	irqN	The IRQn_Type enumerator for the interrupt to check.

Returns

- True if the interrupt is disabled.

CORE_GetNvicRamTableHandler

```
void * CORE_GetNvicRamTableHandler (IRQn_Type irqN)
```

Utility function to get the handler for a specific interrupt.

Parameters

Type	Direction	Argument Name	Description
IRQn_Type	[in]	irqN	The IRQn_Type enumerator for the interrupt.

Returns

- The handler address.

Note

- Uses the interrupt vector table defined by the current VTOR register value.

CORE_SetNvicRamTableHandler

```
void CORE_SetNvicRamTableHandler (IRQn_Type irqN, void * handler)
```

Utility function to set the handler for a specific interrupt.

Parameters

Type	Direction	Argument Name	Description
IRQn_Type	[in]	irqN	The IRQn_Type enumerator for the interrupt.
void *	[in]	handler	The handler address.

Note

- Uses the interrupt vector table defined by the current VTOR register value.

CORE_InitNvicVectorTable

```
void CORE_InitNvicVectorTable (uint32_t * sourceTable, uint32_t sourceSize, uint32_t * targetTable, uint32_t targetSize, void * defaultHandler, bool overwriteActive)
```

Initialize an interrupt vector table by copying table entries from a source to a target table.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[in]	sourceTable	The address of the source vector table.
uint32_t	[in]	sourceSize	A number of entries in the source vector table.
uint32_t *	[in]	targetTable	The address of the target (new) vector table.
uint32_t	[in]	targetSize	A number of entries in the target vector table.
void *	[in]	defaultHandler	An address of the interrupt handler used for target entries for which where there is no corresponding source entry (i.e., the target table is larger than the source table).
bool	[in]	overwriteActive	When true, a target table entry is always overwritten with the corresponding source entry. If false, a target table entry is only overwritten if it is zero. This makes it possible for an application to partly initialize a target table before passing it to this function.

Note

- This function will set a new VTOR register value.

cycle_counter_start

```
static void cycle_counter_start (dwt_cycle_counter_handle_t * handle)
```

Start a recording.

Parameters

Type	Direction	Argument Name	Description
dwt_cycle_counter_handle_t *	[in]	handle	Pointer to initialized counter handle.

Note

- SL_EMLIB_CORE_ENABLE_INTERRUPT_DISABLED_TIMING must be enabled.

cycle_counter_stop

```
static void cycle_counter_stop (dwt\_cycle\_counter\_handle\_t * handle)
```

Stop a recording.

Parameters

Type	Direction	Argument Name	Description
dwt_cycle_counter_handle_t *	[in]	handle	Pointer to initialized counter handle.

Note

- SL_EMLIB_CORE_ENABLE_INTERRUPT_DISABLED_TIMING must be enabled.

CORE_get_max_time_critical_section

```
uint32_t CORE_get_max_time_critical_section (void )
```

Returns the max time spent in critical section.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The max time spent in critical section.

Note

- SL_EMLIB_CORE_ENABLE_INTERRUPT_DISABLED_TIMING must be enabled.

CORE_get_max_time_atomic_section

```
uint32_t CORE_get_max_time_atomic_section (void )
```

Returns the max time spent in atomic section.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The max time spent in atomic section.

Note

SL_EMLIB_CORE_ENABLE_INTERRUPT_DISABLED_TIMING must be enabled.

CORE_clear_max_time_critical_section

```
void CORE_clear_max_time_critical_section (void )
```

Clears the max time spent in atomic section.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- SL_EMLIB_CORE_ENABLE_INTERRUPT_DISABLED_TIMING must be enabled.

CORE_clear_max_time_atomic_section

```
void CORE_clear_max_time_atomic_section (void )
```

Clears the max time spent in atomic section.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- SL_EMLIB_CORE_ENABLE_INTERRUPT_DISABLED_TIMING must be enabled.

Macro Definition Documentation

CORE_INTERRUPT_ENTRY

```
#define CORE_INTERRUPT_ENTRY
```

Placeholder for optional interrupt handler entry code.

This might be needed when working with an RTOS.

CORE_INTERRUPT_EXIT

```
#define CORE_INTERRUPT_EXIT
```

Placeholder for optional interrupt handler exit code.

This might be needed when working with an RTOS.

CORE_ATOMIC_METHOD_PRIMASK

```
#define CORE_ATOMIC_METHOD_PRIMASK
```

Value:

0

Use PRIMASK register to disable interrupts in ATOMIC sections.

CORE_ATOMIC_METHOD_BASEPRI

```
#define CORE_ATOMIC_METHOD_BASEPRI
```

Value:

1

Use BASEPRI register to disable interrupts in ATOMIC sections.

CORE_ATOMIC_BASE_PRIORITY_LEVEL

```
#define CORE_ATOMIC_BASE_PRIORITY_LEVEL
```

Value:

3

The interrupt priority level disabled within ATOMIC regions.

Interrupts with priority level equal to or lower than this definition will be disabled within ATOMIC regions.

CORE_DECLARE_IRQ_STATE

```
#define CORE_DECLARE_IRQ_STATE
```

Value:

```
CORE_irqState_t irqState
```

Allocate storage for PRIMASK or BASEPRI value for use by CORE_ENTER/EXIT_ATOMIC() and CORE_ENTER/EXIT_CRITICAL() macros.

CORE_CRITICAL_IRQ_DISABLE

```
#define CORE_CRITICAL_IRQ_DISABLE
```

Value:

()

CRITICAL style interrupt disable.

CORE_CRITICAL_IRQ_ENABLE

```
#define CORE_CRITICAL_IRQ_ENABLE
```

Value:

```
()
```

CRITICAL style interrupt enable.

CORE_CRITICAL_SECTION

```
#define CORE_CRITICAL_SECTION
```

Value:

```
0 | {
0 |     CORE_DECLARE_IRQ_STATE;
0 |     CORE_ENTER_CRITICAL();
0 |     {
0 |         yourcode
0 |     }
0 |     CORE_EXIT_CRITICAL();
0 | }
```

Convenience macro for implementing a CRITICAL section.

CORE_ENTER_CRITICAL

```
#define CORE_ENTER_CRITICAL
```

Value:

```
()
```

Enter CRITICAL section.

Assumes that a [CORE_DECLARE_IRQ_STATE](#) exist in scope.

CORE_EXIT_CRITICAL

```
#define CORE_EXIT_CRITICAL
```

Value:

```
()
```

Exit CRITICAL section.

Assumes that a [CORE_DECLARE_IRQ_STATE](#) exist in scope.

CORE_YIELD_CRITICAL

```
#define CORE_YIELD_CRITICAL
```

Value:

```
()
```

CRITICAL style yield.

CORE_ATOMIC_IRQ_DISABLE

```
#define CORE_ATOMIC_IRQ_DISABLE
```

Value:

```
()
```

ATOMIC style interrupt disable.

CORE_ATOMIC_IRQ_ENABLE

```
#define CORE_ATOMIC_IRQ_ENABLE
```

Value:

```
()
```

ATOMIC style interrupt enable.

CORE_ATOMIC_SECTION

```
#define CORE_ATOMIC_SECTION
```

Value:

```
0 | {                                \|
0 |   CORE_DECLARE_IRQ_STATE;        \|
0 |   CORE_ENTER_ATOMIC();           \|
0 |   {                               \|
0 |     yourcode                     \|
0 |   }                               \|
0 |   CORE_EXIT_ATOMIC();            \|
0 | }
```

Convenience macro for implementing an ATOMIC section.

CORE_ENTER_ATOMIC

```
#define CORE_ENTER_ATOMIC
```

Value:

```
()
```

Enter ATOMIC section.

Assumes that a [CORE_DECLARE_IRQ_STATE](#) exist in scope.

CORE_EXIT_ATOMIC

```
#define CORE_EXIT_ATOMIC
```

Value:

```
()
```

Exit ATOMIC section.

Assumes that a `CORE_DECLARE_IRQ_STATE` exist in scope.

CORE_YIELD_ATOMIC

```
#define CORE_YIELD_ATOMIC
```

Value:

```
()
```

ATOMIC style yield.

CORE_IRQ_DISABLED

```
#define CORE_IRQ_DISABLED
```

Value:

```
()
```

Check if IRQ is disabled.

CORE_IN_IRQ_CONTEXT

```
#define CORE_IN_IRQ_CONTEXT
```

Value:

```
()
```

Check if inside an IRQ handler.

CORE_DECLARE_IRQ_STATE

```
#define CORE_DECLARE_IRQ_STATE
```

Allocate storage for PRIMASK or BASEPRI value for use by `CORE_ENTER/EXIT_ATOMIC()` and `CORE_ENTER/EXIT_CRITICAL()` macros.

CORE_CRITICAL_IRQ_DISABLE

```
#define CORE_CRITICAL_IRQ_DISABLE
```

CRITICAL style interrupt disable.

CORE_CRITICAL_IRQ_ENABLE

```
#define CORE_CRITICAL_IRQ_ENABLE
```

CRITICAL style interrupt enable.

CORE_CRITICAL_SECTION

```
#define CORE_CRITICAL_SECTION
```

Value:

```
0 | {                                \|
0 |     CORE_DECLARE_IRQ_STATE;      \|
0 |     CORE_ENTER_CRITICAL();        \|
0 |     {                             \|
0 |         yourcode                  \|
0 |     }                             \|
0 |     CORE_EXIT_CRITICAL();         \|
0 | }
```

Convenience macro for implementing a CRITICAL section.

CORE_ENTER_CRITICAL

```
#define CORE_ENTER_CRITICAL
```

Enter CRITICAL section.

Assumes that a [CORE_DECLARE_IRQ_STATE](#) exist in scope.

CORE_EXIT_CRITICAL

```
#define CORE_EXIT_CRITICAL
```

Exit CRITICAL section.

Assumes that a [CORE_DECLARE_IRQ_STATE](#) exist in scope.

CORE_YIELD_CRITICAL

```
#define CORE_YIELD_CRITICAL
```

CRITICAL style yield.

CORE_ATOMIC_IRQ_DISABLE

```
#define CORE_ATOMIC_IRQ_DISABLE
```

ATOMIC style interrupt disable.

CORE_ATOMIC_IRQ_ENABLE

```
#define CORE_ATOMIC_IRQ_ENABLE
```

ATOMIC style interrupt enable.

CORE_ATOMIC_SECTION

```
#define CORE_ATOMIC_SECTION
```

Value:

```
0 | {
0 |     CORE_DECLARE_IRQ_STATE;
0 |     CORE_ENTER_ATOMIC();
0 |     {
0 |         yourcode
0 |     }
0 |     CORE_EXIT_ATOMIC();
0 | }
```

Convenience macro for implementing an ATOMIC section.

CORE_ENTER_ATOMIC

```
#define CORE_ENTER_ATOMIC
```

Enter ATOMIC section.

Assumes that a [CORE_DECLARE_IRQ_STATE](#) exist in scope.

CORE_EXIT_ATOMIC

```
#define CORE_EXIT_ATOMIC
```

Exit ATOMIC section.

Assumes that a [CORE_DECLARE_IRQ_STATE](#) exist in scope.

CORE_YIELD_ATOMIC

```
#define CORE_YIELD_ATOMIC
```

ATOMIC style yield.

CORE_NVIC_REG_WORDS

```
#define CORE_NVIC_REG_WORDS
```

Value:

```
((EXT_IRQ_COUNT + 31) / 32)
```

Number of words in a NVIC mask set.

CORE_DEFAULT_VECTOR_TABLE_ENTRIES

```
#define CORE_DEFAULT_VECTOR_TABLE_ENTRIES
```

Value:

```
(EXT_IRQ_COUNT + 16)
```

Number of entries in a default interrupt vector table.

CORE_INTERRUPT_HIGHEST_PRIORITY

```
#define CORE_INTERRUPT_HIGHEST_PRIORITY
```

Value:

```
0
```

Highest priority for core interrupt.

CORE_INTERRUPT_DEFAULT_PRIORITY

```
#define CORE_INTERRUPT_DEFAULT_PRIORITY
```

Value:

```
5
```

Default priority for core interrupt.

CORE_INTERRUPT_LOWEST_PRIORITY

```
#define CORE_INTERRUPT_LOWEST_PRIORITY
```

Value:

```
7
```

Lowest priority for core interrupt.

CORE_ATOMIC_METHOD_DEFAULT

```
#define CORE_ATOMIC_METHOD_DEFAULT
```

Value:

```
CORE_ATOMIC_METHOD_BASEPRI
```

Default method to disable interrupts in ATOMIC sections.

CORE_ATOMIC_METHOD

```
#define CORE_ATOMIC_METHOD
```

Value:

```
CORE_ATOMIC_METHOD_PRIMASK
```

Specify which method to use when implementing ATOMIC sections.

You can select between BASEPRI or PRIMASK method. Note

- On Cortex-M0+ devices only PRIMASK can be used.

CORE_DECLARE_NVIC_STATE

```
#define CORE_DECLARE_NVIC_STATE
```

Value:

```
CORE_nvicMask_t nvicState
```

Allocate storage for NVIC interrupt masks for use by CORE_ENTER/EXIT_NVIC() macros.

CORE_DECLARE_NVIC_MASK

```
#define CORE_DECLARE_NVIC_MASK
```

Value:

```
(x)
```

Allocate storage for NVIC interrupt masks.

CORE_DECLARE_NVIC_ZEROMASK

```
#define CORE_DECLARE_NVIC_ZEROMASK
```

Value:

```
(x)
```

Allocate storage for and zero initialize NVIC interrupt mask.

CORE_NVIC_DISABLE

```
#define CORE_NVIC_DISABLE
```

Value:

```
(mask)
```

NVIC mask style interrupt disable.

CORE_NVIC_ENABLE

```
#define CORE_NVIC_ENABLE
```

Value:

(mask)

NVIC mask style interrupt enable.

CORE_NVIC_SECTION

```
#define CORE_NVIC_SECTION
```

Value:

```
0 | {                                \
0 |     CORE_DECLARE_NVIC_STATE;      \
0 |     CORE_ENTER_NVIC(mask);        \
0 |     {                             \
0 |         yourcode                  \
0 |     }                             \
0 |     CORE_EXIT_NVIC();             \
0 | }
```

Convenience macro for implementing a NVIC mask section.

CORE_ENTER_NVIC

```
#define CORE_ENTER_NVIC
```

Value:

(disable)

Enter NVIC mask section.

Assumes that a [CORE_DECLARE_NVIC_STATE](#) exist in scope.

CORE_EXIT_NVIC

```
#define CORE_EXIT_NVIC
```

Value:

()

Exit NVIC mask section.

Assumes that a [CORE_DECLARE_NVIC_STATE](#) exist in scope.

CORE_YIELD_NVIC

```
#define CORE_YIELD_NVIC
```

Value:

(enable)

NVIC maks style yield.

dwt_cycle_counter_handle_t

A Cycle Counter Instance.

Public Attributes

uint32_t	start	Cycle counter at start of recording.
uint32_t	cycles	Cycles elapsed in last recording.
uint32_t	max	Max recorded cycles since last reset or init.

Public Attribute Documentation

start

```
uint32_t dwt_cycle_counter_handle_t::start
```

Cycle counter at start of recording.

cycles

```
uint32_t dwt_cycle_counter_handle_t::cycles
```

Cycles elapsed in last recording.

max

```
uint32_t dwt_cycle_counter_handle_t::max
```

Max recorded cycles since last reset or init.

CORE_nvicMask_t

Storage for NVIC interrupt masks.

Public Attributes

uint32_t [a](#)
Array of NVIC mask words.

Public Attribute Documentation

a

```
uint32_t CORE_nvicMask_t::a[CORE_NVIC_REG_WORDS]
```

Array of NVIC mask words.

DBG - Debug

DBG - Debug

Debug (DBG) Peripheral API.

This module contains functions to control the DBG peripheral of Silicon Labs 32-bit MCUs and SoCs. The Debug Interface is used to program and debug Silicon Labs devices.

Enumerations

```
enum   DBG_LockMode_TypeDef {
    dbgLockModeAllowErase = 1UL
    dbgLockModePermanent = 2UL
}
```

Lock modes.

Functions

```
bool   DBG_Connected(void)
    Check if a debugger is connected (and debug session activated).

void   DBG_SWOEnable(unsigned int location)
    Enable Serial Wire Output (SWO) pin.

void   DBG_EM2DebugEnable(bool enable)
    Enable or disable debug support while in EM2 mode.
```

Enumeration Documentation

DBG_LockMode_TypeDef

DBG_LockMode_TypeDef

Lock modes.

	Enumerator
dbgLockModeAllowErase	Lock debug access.
dbgLockModePermanent	Lock debug access permanently.

Function Documentation

DBG_Connected

bool DBG_Connected (void)

Check if a debugger is connected (and debug session activated).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Used to make run-time decisions depending on whether or not a debug session has been active since last reset, i.e., using a debug probe or similar. In some cases, special handling is required in that scenario.

Returns

- True if a debug session is active since last reset, otherwise false.

DBG_SWOEnable

```
void DBG_SWOEnable (unsigned int location)
```

Enable Serial Wire Output (SWO) pin.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	location	A pin location used for SWO pin on the application in use.

The SWO pin (sometimes denoted SWV, serial wire viewer) allows for miscellaneous output to be passed from the Cortex-M3 debug trace module to an external debug probe. By default, the debug trace module and pin output may be disabled.

Since the SWO pin is only useful when using a debugger, a suggested use of this function during startup may be:

```
* if (DBG_Connected())
* {
*   DBG_SWOEnable(1);
* }
*
```

By checking if the debugger is attached, a setup leading to a higher energy consumption when the debugger is attached can be avoided when not using a debugger.

Another alternative may be to set the debugger tool chain to configure the required setup (similar to the content of this function) by some sort of toolchain scripting during its attach/reset procedure. In that case, the above suggested code for enabling the SWO pin is not required in the application.

DBG_EM2DebugEnable

```
void DBG_EM2DebugEnable (bool enable)
```

Enable or disable debug support while in EM2 mode.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	Boolean true enables EM2 debug support, false disables.

Warnings

- Disabling debug support in EM2 will reduce current consumption with 1-2 uA, but some debuggers will have problems regaining control over a device which is in EM2 and has debug support disabled.

To remedy this, set the WSTK switch next to the battery holder to USB (powers down the EFR). Execute Simplicity Commander with command line parameters: `./commander.exe device recover` and then immediately move the switch to the AEM position. An additional `./commander.exe device masserase` command completes the recovery procedure.

EMU - Energy Management Unit

EMU - Energy Management Unit

Energy Management Unit (EMU) Peripheral API.

This module contains functions to control the EMU peripheral of Silicon Labs 32-bit MCUs and SoCs. The EMU handles the different low energy modes in Silicon Labs microcontrollers.

Modules

[EMU_EM0Init_TypeDef](#)

[EMU_EM23Init_TypeDef](#)

[EMU_EM4Init_TypeDef](#)

[EMU_DCDCInit_TypeDef](#)

Enumerations

```
enum EMU\_BODMode\_TypeDef {
    emuBODMode_Active
    emuBODMode_Inactive
}
BOD threshold setting selector, active or inactive mode.

enum EMU\_EM4State\_TypeDef {
    emuEM4Shutoff = 0
    emuEM4Hibernate = 1
}
EM4 modes.

enum EMU\_EM4PinRetention\_TypeDef {
    emuPinRetentionDisable = EMU_EM4CTRL_EM4IORETMODE_DISABLE
    emuPinRetentionEm4Exit = EMU_EM4CTRL_EM4IORETMODE_EM4EXIT
    emuPinRetentionLatch = EMU_EM4CTRL_EM4IORETMODE_SWUNLATCH
}
EM4 Pin Retention Type.

enum EMU\_PowerConfig\_TypeDef {
    emuPowerConfig_DcdcToDvdd
}
Power configurations.

enum EMU\_DcdcMode\_TypeDef {
    emuDcdcMode_Bypass = _DCDC_CTRL_MODE_BYPASS
    emuDcdcMode_Regulation = _DCDC_CTRL_MODE_DCDCREGULATION
}
DCDC mode.

enum EMU\_VreginCmpThreshold\_TypeDef {
    emuVreginCmpThreshold_2v0 = 0
    emuVreginCmpThreshold_2v1 = 1
    emuVreginCmpThreshold_2v2 = 2
    emuVreginCmpThreshold_2v3 = 3
}
VREGIN comparator threshold.
```

```
enum EMU_DcdcTonMaxTimeout_TypeDef {
    emuDcdcTonMaxTimeout_Off = _DCDC_CTRL_IPKTMAXCTRL_OFF
    emuDcdcTonMaxTimeout_0P35us = _DCDC_CTRL_IPKTMAXCTRL_TMAX_0P35us
    emuDcdcTonMaxTimeout_0P63us = _DCDC_CTRL_IPKTMAXCTRL_TMAX_0P63us
    emuDcdcTonMaxTimeout_0P91us = _DCDC_CTRL_IPKTMAXCTRL_TMAX_0P91us
    emuDcdcTonMaxTimeout_1P19us = _DCDC_CTRL_IPKTMAXCTRL_TMAX_1P19us
    emuDcdcTonMaxTimeout_1P47us = _DCDC_CTRL_IPKTMAXCTRL_TMAX_1P47us
    emuDcdcTonMaxTimeout_1P75us = _DCDC_CTRL_IPKTMAXCTRL_TMAX_1P75us
    emuDcdcTonMaxTimeout_2P03us = _DCDC_CTRL_IPKTMAXCTRL_TMAX_2P03us
}
DCDC Buck Ton max timeout.
```

```
enum EMU_DcdcDriveSpeed_TypeDef {
    emuDcdcDriveSpeed_BestEmi = _DCDC_EM01CTRL0_DRVSPEED_DEFAULT_SETTING
    emuDcdcDriveSpeed_Default = _DCDC_EM01CTRL0_DRVSPEED_DEFAULT_SETTING
    emuDcdcDriveSpeed_Intermediate = _DCDC_EM01CTRL0_DRVSPEED_DEFAULT_SETTING
    emuDcdcDriveSpeed_BestEfficiency = _DCDC_EM01CTRL0_DRVSPEED_DEFAULT_SETTING
}
DCDC Buck drive speed.
```

```
enum EMU_DcdcPeakCurrent_TypeDef {
    emuDcdcPeakCurrent_Load5mA = _DCDC_EM23CTRL0_IPKVAL_LOAD5MA
    emuDcdcPeakCurrent_Load10mA = _DCDC_EM23CTRL0_IPKVAL_LOAD10MA
    emuDcdcPeakCurrent_Load36mA = _DCDC_EM01CTRL0_IPKVAL_Load36mA
    emuDcdcPeakCurrent_Load40mA = _DCDC_EM01CTRL0_IPKVAL_Load40mA
    emuDcdcPeakCurrent_Load44mA = _DCDC_EM01CTRL0_IPKVAL_Load44mA
    emuDcdcPeakCurrent_Load48mA = _DCDC_EM01CTRL0_IPKVAL_Load48mA
    emuDcdcPeakCurrent_Load52mA = _DCDC_EM01CTRL0_IPKVAL_Load52mA
    emuDcdcPeakCurrent_Load56mA = _DCDC_EM01CTRL0_IPKVAL_Load56mA
    emuDcdcPeakCurrent_Load60mA = _DCDC_EM01CTRL0_IPKVAL_Load60mA
}
DCDC Buck peak current setting.
```

```
enum EMU_VScaleEM01_TypeDef {
    emuVScaleEM01_HighPerformance = _EMU_STATUS_VSCALE_VSCALE2
    emuVScaleEM01_LowPower = _EMU_STATUS_VSCALE_VSCALE1
}
Supported EM0/1 Voltage Scaling Levels.
```

```
enum EMU_VScaleEM23_TypeDef {
    emuVScaleEM23_FastWakeup = _EMU_CTRL_EM23VSCALE_VSCALE2
    emuVScaleEM23_LowPower = _EMU_CTRL_EM23VSCALE_VSCALE0
}
Supported EM2/3 Voltage Scaling Levels.
```

```
enum EMU_TempAvgNum_TypeDef {
    emuTempAvgNum_16 = _EMU_CTRL_TEMPAVGNUM_N16
    emuTempAvgNum_64 = _EMU_CTRL_TEMPAVGNUM_N64
}
Number of samples to use for temperature averaging.
```

Functions

```
void EMU_EM01Init(const EMU_EM01Init_TypeDef *em01Init)
    Update the EMU module with Energy Mode 0 and 1 configuration.
```

```
void EMU_EM23Init(const EMU_EM23Init_TypeDef *em23Init)
    Update the EMU module with Energy Mode 2 and 3 configuration.
```

```
void EMU_EM23PresleepHook(void)
    Energy mode 2/3 pre-sleep hook function.
```

```
void EMU_EM23PostsleepHook(void)
    Energy mode 2/3 post-sleep hook function.
```

void	EMU_EFPem23PresleepHook(void) EFP's Energy mode 2/3 pre-sleep hook function.
void	EMU_EFPem23PostsleepHook(void) EFP's Energy mode 2/3 post-sleep hook function.
void	EMU_EnterEM2(bool restore) Enter energy mode 2 (EM2).
void	EMU_EnterEM3(bool restore) Enter energy mode 3 (EM3).
void	EMU_Save(void) Save the CMU HF clock select state, oscillator enable, and voltage scaling (if available) before EMU_EnterEM2() or EMU_EnterEM3() are called with the restore parameter set to false.
void	EMU_Restore(void) Restore CMU HF clock select state, oscillator enable, and voltage scaling (if available) after EMU_EnterEM2() or EMU_EnterEM3() are called with the restore parameter set to false.
void	EMU_EM4Init(const EMU_EM4Init_TypeDef *em4Init) Update the EMU module with Energy Mode 4 configuration.
void	EMU_EM4PresleepHook(void) Energy mode 4 pre-sleep hook function.
void	EMU_EFPem4PresleepHook(void) EFP's Energy mode 4 pre-sleep hook function.
void	EMU_EnterEM4(void) Enter energy mode 4 (EM4).
void	EMU_EnterEM4Wait(void) Enter energy mode 4 (EM4).
void	EMU_EnterEM4H(void) Enter energy mode 4 hibernate (EM4H).
void	EMU_EnterEM4S(void) Enter energy mode 4 shutoff (EM4S).
void	EMU_MemPwrDown(uint32_t blocks) Power down memory block.
void	EMU_RamPowerDown(uint32_t start, uint32_t end) Power down RAM memory blocks.
void	EMU_RamPowerUp(void) Power up all available RAM memory blocks.
void	EMU_UpdateOscConfig(void) Update EMU module with CMU oscillator selection/enable status.
void	EMU_EFPem01VScale(EMU_VScaleEM01_TypeDef voltage) Energy mode 01 voltage scaling hook function.
void	EMU_VScaleEM01ByClock(uint32_t clockFrequency, bool wait) Voltage scale in EM0 and 1 by clock frequency.
void	EMU_VScaleEM01(EMU_VScaleEM01_TypeDef voltage, bool wait) Force voltage scaling in EM0 and 1 to a specific voltage level.
sl_status_t	EMU_DCDCModeSet(EMU_DcdcMode_TypeDef dcdcMode) Set DCDC regulator operating mode.
void	EMU_DCDCUpdatedHook(void) Indicate that the DCDC peripheral bus clock enable has changed allowing RAIL to react accordingly.

bool	EMU_DCDCInit (const EMU_DCDCInit_TypeDef *dcdcInit) Configure the DCDC regulator.
bool	EMU_DCDCPowerOff (void) Power off the DCDC regulator.
void	EMU_EM01PeakCurrentSet (const EMU_DcdcPeakCurrent_TypeDef peakCurrentEM01) Set EM01 mode Peak Current setting.
float	EMU_TemperatureGet (void) Get temperature in degrees Celsius.
void	EMU_EFPDirectModeEnable (bool enable) Enable/disable EFP Direct Mode.
void	EMU_EFPDriveDecoupleSet (bool enable) Set to enable EFP to drive Decouple voltage.
void	EMU_EFPDriveDvddSet (bool enable) Set to enable EFP to drive DVDD voltage.
void	EMU_DCDCLock (void) Lock DCDC registers in order to protect them against unintended modification.
void	EMU_DCDCUnlock (void) Unlock the DCDC so that writing to locked registers again is possible.
void	EMU_EnterEM1 (void) Enter energy mode 1 (EM1).
void	EMU_VScaleWait (void) Wait for voltage scaling to complete.
EMU_VScaleEMO1_TypeDef	EMU_VScaleGet (void) Get current voltage scaling level.
void	EMU_IntClear (uint32_t flags) Clear one or more pending EMU interrupts.
void	EMU_IntDisable (uint32_t flags) Disable one or more EMU interrupts.
void	EMU_IntEnable (uint32_t flags) Enable one or more EMU interrupts.
void	EMU_EFPIntDisable (uint32_t flags) Disable one or more EFP interrupts.
void	EMU_EFPIntEnable (uint32_t flags) Enable one or more EFP interrupts.
uint32_t	EMU_EFPIntGet (void) Get pending EMU EFP interrupt flags.
uint32_t	EMU_EFPIntGetEnabled (void) Get enabled and pending EMU EFP interrupt flags.
void	EMU_EFPIntSet (uint32_t flags) Set one or more pending EMU EFP interrupts.
void	EMU_EFPIntClear (uint32_t flags) Clear one or more pending EMU EFP interrupts.
uint32_t	EMU_IntGet (void) Get pending EMU interrupt flags.

uint32_t	EMU_IntGetEnabled(void) Get enabled and pending EMU interrupt flags.
void	EMU_IntSet(uint32_t flags) Set one or more pending EMU interrupts.
void	EMU_Lock(void) Lock EMU registers in order to protect them against unintended modification.
void	EMU_Unlock(void) Unlock the EMU so that writing to locked registers again is possible.
void	EMU_UnlatchPinRetention(void) When EM4 pin retention is set to emuPinRetentionLatch, then pins are retained through EM4 entry and wakeup.
bool	EMU_TemperatureReady(void) Temperature measurement ready status.
float	EMU_TemperatureAvgGet(void) Get averaged temperature in degrees Celsius.
void	EMU_TemperatureAvgRequest(EMU_TempAvgNum_TypeDef numSamples) Request averaged temperature.

Macros

#define	EMU_VSCALE_PRESENT Voltage scaling present.
#define	EMU_VSCALE_EM01_PRESENT Voltage scaling for EM01 present.
#define	EMU_SERIES2_DCDC_BUCK_PRESENT DC-DC buck converter present.
#define	EMU_EM01INIT_DEFAULT undefined Default initialization of EM0 and 1 configuration.
#define	EMU_EM23INIT_DEFAULT undefined Default initialization of EM2 and 3 configuration.
#define	EMU_EM4INIT_DEFAULT undefined Default initialization of EM4 configuration (Series 1 without VSCALE).
#define	EMU_DCDCINIT_DEFAULT undefined Default DCDC Buck initialization.
#define	EMU_TEMP_ZERO_C_IN_KELVIN (273.15f) Zero degrees Celcius in Kelvin.

Enumeration Documentation

EMU_BODMode_TypeDef

EMU_BODMode_TypeDef

BOD threshold setting selector, active or inactive mode.

Enumerator	
emuBODMode_Active	Configure BOD threshold for active mode.
emuBODMode_Inactive	Configure BOD threshold for inactive mode.

EMU_EM4State_TypeDef

EMU_EM4State_TypeDef

EM4 modes.

Enumerator	
emuEM4Shutoff	EM4 Shutoff.
emuEM4Hibernate	EM4 Hibernate.

EMU_EM4PinRetention_TypeDef

EMU_EM4PinRetention_TypeDef

EM4 Pin Retention Type.

Enumerator	
emuPinRetentionDisable	No Retention: Pads enter reset state when entering EM4.
emuPinRetentionEm4Exit	Retention through EM4: Pads enter reset state when exiting EM4.
emuPinRetentionLatch	Retention through EM4 and wakeup: call EMU_UnlatchPinRetention() to release pins from retention after EM4 wakeup.

EMU_PowerConfig_TypeDef

EMU_PowerConfig_TypeDef

Power configurations.

DCDC-to-DVDD is currently the only supported mode.

Enumerator	
emuPowerConfig_DcdcToDvdd	DCDC is connected to DVDD.

EMU_DcdcMode_TypeDef

EMU_DcdcMode_TypeDef

DCDC mode.

Enumerator	
emuDcdcMode_Bypass	DCDC regulator bypass.
emuDcdcMode_Regulation	DCDC regulator on.

EMU_VreginCmpThreshold_TypeDef

EMU_VreginCmpThreshold_TypeDef

VREGIN comparator threshold.

Enumerator

emuVreginCmpThreshold_2v0	Comparator threshold is 2.0V.
emuVreginCmpThreshold_2v1	Comparator threshold is 2.1V.
emuVreginCmpThreshold_2v2	Comparator threshold is 2.2V.
emuVreginCmpThreshold_2v3	Comparator threshold is 2.3V.

EMU_DcdcTonMaxTimeout_TypeDef

EMU_DcdcTonMaxTimeout_TypeDef

DCDC Buck Ton max timeout.

Enumerator	
emuDcdcTonMaxTimeout_Off	Ton max off.
emuDcdcTonMaxTimeout_OP35us	Ton max is 0.35us.
emuDcdcTonMaxTimeout_OP63us	Ton max is 0.63us.
emuDcdcTonMaxTimeout_OP91us	Ton max is 0.91us.
emuDcdcTonMaxTimeout_1P19us	Ton max is 1.19us.
emuDcdcTonMaxTimeout_1P47us	Ton max is 1.47us.
emuDcdcTonMaxTimeout_1P75us	Ton max is 1.75us.
emuDcdcTonMaxTimeout_2P03us	Ton max is 2.03us.

EMU_DcdcDriveSpeed_TypeDef

EMU_DcdcDriveSpeed_TypeDef

DCDC Buck drive speed.

Enumerator	
emuDcdcDriveSpeed_BestEmi	Recommend no options other than DEFAULT be used here, as there is no benefit.
emuDcdcDriveSpeed_Default	Recommend no options other than DEFAULT be used here, as there is no benefit.
emuDcdcDriveSpeed_Intermediate	Recommend no options other than DEFAULT be used here, as there is no benefit.
emuDcdcDriveSpeed_BestEfficiency	Recommend no options other than DEFAULT be used here, as there is no benefit.

EMU_DcdcPeakCurrent_TypeDef

EMU_DcdcPeakCurrent_TypeDef

DCDC Buck peak current setting.

Enumerator	
emuDcdcPeakCurrent_Load5mA	Load 5mA, peak current 90mA.
emuDcdcPeakCurrent_Load10mA	Load 10mA, peak current 150mA.
emuDcdcPeakCurrent_Load36mA	Load 36mA, peak current 90mA.

emuDcdcPeakCurrent_Load40mA	Load 40mA, peak current 100mA.
emuDcdcPeakCurrent_Load44mA	Load 44mA, peak current 110mA.
emuDcdcPeakCurrent_Load48mA	Load 48mA, peak current 120mA.
emuDcdcPeakCurrent_Load52mA	Load 52mA, peak current 130mA.
emuDcdcPeakCurrent_Load56mA	Load 56mA, peak current 140mA.
emuDcdcPeakCurrent_Load60mA	Load 60mA, peak current 150mA.

EMU_VScaleEM01_TypeDef

EMU_VScaleEM01_TypeDef

Supported EM0/1 Voltage Scaling Levels.

Enumerator	
emuVScaleEM01_HighPerformance	High-performance voltage level.
emuVScaleEM01_LowPower	Low-power optimized voltage level.

EMU_VScaleEM23_TypeDef

EMU_VScaleEM23_TypeDef

Supported EM2/3 Voltage Scaling Levels.

Enumerator	
emuVScaleEM23_FastWakeup	Fast-wakeup voltage level.
emuVScaleEM23_LowPower	Low-power optimized voltage level.

EMU_TempAvgNum_TypeDef

EMU_TempAvgNum_TypeDef

Number of samples to use for temperature averaging.

Enumerator	
emuTempAvgNum_16	16 samples used for temperature averaging.
emuTempAvgNum_64	64 samples used for temperature averaging.

Function Documentation

EMU_EM01Init

void EMU_EM01Init (const [EMU_EM01Init_TypeDef](#) * em01Init)

Update the EMU module with Energy Mode 0 and 1 configuration.

Parameters

Type	Direction	Argument Name	Description
const EMU_EM01Init_TypeDef *	[in]	em01Init	Energy Mode 0 and 1 configuration structure.

EMU_EM23Init

```
void EMU_EM23Init (const EMU_EM23Init_TypeDef * em23Init)
```

Update the EMU module with Energy Mode 2 and 3 configuration.

Parameters

Type	Direction	Argument Name	Description
const EMU_EM23Init_TypeDef *	[in]	em23Init	Energy Mode 2 and 3 configuration structure.

EMU_EM23PresleepHook

```
void EMU_EM23PresleepHook (void )
```

Energy mode 2/3 pre-sleep hook function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is called by [EMU_EnterEM2\(\)](#) and [EMU_EnterEM3\(\)](#) functions just prior to execution of the WFI instruction. The function implementation does not perform anything, but it is SL_WEAK so that it can be re-implemented in application code if actions are needed.

EMU_EM23PostsleepHook

```
void EMU_EM23PostsleepHook (void )
```

Energy mode 2/3 post-sleep hook function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is called by [EMU_EnterEM2\(\)](#) and [EMU_EnterEM3\(\)](#) functions just after wakeup from the WFI instruction. The function implementation does not perform anything, but it is SL_WEAK so that it can be re-implemented in application code if actions are needed.

EMU_EFPEM23PresleepHook

```
void EMU_EFPEM23PresleepHook (void )
```

EEP's Energy mode 2/3 pre-sleep hook function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is similar to [EMU_EM23PresleepHook\(\)](#) but is reserved for EFP usage.

Note

- The function is primarily meant to be used in systems with EFP circuitry. (EFP = Energy Friendly Pmic (PMIC = Power Management IC)). In such systems there is a need to drive certain signals to EFP pins to notify about energy mode transitions.

EMU_EFPEM23PostsleepHook

```
void EMU_EFPEM23PostsleepHook (void )
```

EFP's Energy mode 2/3 post-sleep hook function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is similar to [EMU_EM23PostsleepHook\(\)](#) but is reserved for EFP usage.

Note

- The function is primarily meant to be used in systems with EFP circuitry. (EFP = Energy Friendly Pmic (PMIC = Power Management IC)). In such systems there is a need to drive certain signals to EFP pins to notify about energy mode transitions.

EMU_EnterEM2

```
void EMU_EnterEM2 (bool restore)
```

Enter energy mode 2 (EM2).

Parameters

Type	Direction	Argument Name	Description
bool	[in]	restore	• true - save and restore oscillators, clocks and voltage scaling, see function details. • false - do not save and restore oscillators and clocks, see function details.

When entering EM2, high-frequency clocks are disabled, i.e., HFXO, HFRCO and AUXHFRCO (for AUXHFRCO, see exception note below). When re-entering EM0, HFRCO is re-enabled and the core will be clocked by the configured HFRCO band. This ensures a quick wakeup from EM2.

However, prior to entering EM2, the core may have been using another oscillator than HFRCO. The `restore` parameter gives the user the option to restore all HF oscillators according to state prior to entering EM2, as well as the clock used to clock the core. This restore procedure is handled by SW. However, since handled by SW, it will not be restored before completing the interrupt function(s) waking up the core!

Note

- If restoring core clock to use the HFXO oscillator, which has been disabled during EM2 mode, this function will stall until the oscillator has stabilized. Stalling time can be reduced by adding interrupt support detecting stable oscillator, and an asynchronous switch to the original oscillator. See CMU documentation. Such a feature is however outside the scope of the implementation in this function.
- If `ERRATA_FIX_EMU_E110_ENABLE` is active, the core's SLEEPONEXIT feature can not be used.
- This function is incompatible with the Power Manager module. When the Power Manager module is present, it must be the one deciding at which EM level the device sleeps to ensure the application properly works. Using both at the same time could lead to undefined behavior in the application.

- If HFXO is re-enabled by this function, and NOT used to clock the core, this function will not wait for HFXO to stabilize. This must be considered by the application if trying to use features relying on that oscillator upon return.
- If a debugger is attached, the AUXHFRCO will not be disabled if enabled upon entering EM2. It will thus remain enabled when returning to EM0 regardless of the `restore` parameter.
- If HFXO autostart and select is enabled by using `CMU_HFXOAutostartEnable()`, the automatic starting and selecting of the core clocks will be done, regardless of the `restore` parameter, when waking up on the wakeup sources corresponding to the autostart and select setting.
- If voltage scaling is supported, the restore parameter is true and the EM0 voltage scaling level is set higher than the EM2 level, then the EM0 level is also restored.
- On Series 2 Config 2 devices (EFRxG22), this function will also relock the DPLL if the DPLL is used and `restore` is true.

Note that the hardware will automatically update the HFRCO frequency in the case where voltage scaling is used in EM2/EM3 and not in EM0/EM1. When the restore argument to this function is true then software will restore the original HFRCO frequency after EM2/EM3 wake up. If the restore argument is false then the HFRCO frequency is 19 MHz when coming out of EM2/EM3 and all wait states are at a safe value.

- The `restore` option should only be used if all clock control is done via the CMU API.

EMU_EnterEM3

```
void EMU_EnterEM3 (bool restore)
```

Enter energy mode 3 (EM3).

Parameters

Type	Direction	Argument Name	Description
bool	[in]	restore	• true - save and restore oscillators, clocks and voltage scaling, see function details. • false - do not save and restore oscillators and clocks, see function details.

When entering EM3, the high-frequency clocks are disabled by hardware, i.e., HFXO, HFRCO, and AUXHFRCO (for AUXHFRCO, see exception note below). In addition, the low-frequency clocks, i.e., LFXO and LFRCO are disabled by software. When re-entering EM0, HFRCO is re-enabled and the core will be clocked by the configured HFRCO band. This ensures a quick wakeup from EM3.

However, prior to entering EM3, the core may have been using an oscillator other than HFRCO. The `restore` parameter gives the user the option to restore all HF/LF oscillators according to state prior to entering EM3, as well as the clock used to clock the core. This restore procedure is handled by software. However, since it is handled by software, it will not be restored before completing the interrupt function(s) waking up the core!

Note

- If restoring core clock to use an oscillator other than HFRCO, this function will stall until the oscillator has stabilized. Stalling time can be reduced by adding interrupt support detecting stable oscillator, and an asynchronous switch to the original oscillator. See CMU documentation. This feature is, however, outside the scope of the implementation in this function.
- If `ERRATA_FIX_EMU_E110_ENABLE` is active, the core's SLEEPONEXIT feature can't be used.
- This function is incompatible with the Power Manager module. When the Power Manager module is present, it must be the one deciding at which EM level the device sleeps to ensure the application properly works. Using both at the same time could lead to undefined behavior in the application.
- If HFXO/LFXO/LFRCO are re-enabled by this function, and NOT used to clock the core, this function will not wait for those oscillators to stabilize. This must be considered by the application if trying to use features relying on

those oscillators upon return.

- If a debugger is attached, the AUXHFRCO will not be disabled if enabled upon entering EM3. It will, therefore, remain enabled when returning to EM0 regardless of the `restore` parameter.
- If voltage scaling is supported, the restore parameter is true and the EM0 voltage scaling level is set higher than the EM3 level, then the EM0 level is also restored.
- On Series 2 Config 2 devices (EFRxG22), this function will also relock the DPLL if the DPLL is used and `restore` is true.
- The `restore` option should only be used if all clock control is done via the CMU API.

EMU_Save

```
void EMU_Save (void )
```

Save the CMU HF clock select state, oscillator enable, and voltage scaling (if available) before [EMU_EnterEM2\(\)](#) or [EMU_EnterEM3\(\)](#) are called with the restore parameter set to false.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Calling this function is equivalent to calling [EMU_EnterEM2\(\)](#) or [EMU_EnterEM3\(\)](#) with the restore parameter set to true, but it allows the state to be saved without going to sleep. The state can be restored manually by calling [EMU_Restore\(\)](#).

EMU_Restore

```
void EMU_Restore (void )
```

Restore CMU HF clock select state, oscillator enable, and voltage scaling (if available) after [EMU_EnterEM2\(\)](#) or [EMU_EnterEM3\(\)](#) are called with the restore parameter set to false.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Calling this function is equivalent to calling [EMU_EnterEM2\(\)](#) or [EMU_EnterEM3\(\)](#) with the restore parameter set to true, but it allows the application to evaluate the wakeup reason before restoring state.

EMU_EM4Init

```
void EMU_EM4Init (const EMU_EM4Init_TypeDef * em4Init)
```

Update the EMU module with Energy Mode 4 configuration.

Parameters

Type	Direction	Argument Name	Description
const EMU_EM4Init_TypeDef *	[in]	em4Init	Energy Mode 4 configuration structure.

EMU_EM4PresleepHook

```
void EMU_EM4PresleepHook (void )
```

Energy mode 4 pre-sleep hook function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is called by [EMU_EnterEM4\(\)](#) just prior to the sequence of writes to put the device in EM4. The function implementation does not perform anything, but it is SL_WEAK so that it can be re-implemented in application code if actions are needed.

EMU_EFPEM4PresleepHook

```
void EMU_EFPEM4PresleepHook (void )
```

EFP's Energy mode 4 pre-sleep hook function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is similar to [EMU_EM4PresleepHook\(\)](#) but is reserved for EFP usage.

Note

- The function is primarily meant to be used in systems with EFP circuitry. (EFP = Energy Friendly Pmic (PMIC = Power Management IC)). In such systems there is a need to drive certain signals to EFP pins to notify about energy mode transitions.

EMU_EnterEM4

```
__NO_RETURN void EMU_EnterEM4 (void )
```

Enter energy mode 4 (EM4).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function never returns. It waits after the EM4 entry request to make sure the CPU is properly shutdown by calling WFI.

Note

- Only a power on reset or external reset pin can wake the device from EM4.

EMU_EnterEM4Wait

```
void EMU_EnterEM4Wait (void )
```

Enter energy mode 4 (EM4).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function never returns. It waits after the EM4 entry request to make sure the CPU is properly shutdown by calling WFI.

Note

- Only a power on reset or external reset pin can wake the device from EM4.

EMU_EnterEM4H

```
void EMU_EnterEM4H (void )
```

Enter energy mode 4 hibernate (EM4H).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function never returns. It waits after the EM4 entry request to make sure the CPU is properly shutdown by calling WFI.

Note

- Retention of clocks and GPIO in EM4 can be configured using [EMU_EM4Init](#) before calling this function.

EMU_EnterEM4S

```
void EMU_EnterEM4S (void )
```

Enter energy mode 4 shutoff (EM4S).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function never returns. It waits after the EM4 entry request to make sure the CPU is properly shutdown by calling WFI.

Note

- Retention of clocks and GPIO in EM4 can be configured using [EMU_EM4Init](#) before calling this function.

EMU_MemPwrDown

```
void EMU_MemPwrDown (uint32_t blocks)
```

Power down memory block.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	blocks	Specifies a logical OR of bits indicating memory blocks to power down. Bit 0 selects block 1, bit 1 selects block 2, and so on. Memory block 0 cannot be disabled. See the reference manual for available memory blocks for a device.

Note

- Only a POR reset can power up the specified memory block(s) after power down.

EMU_RamPowerDown

```
void EMU_RamPowerDown (uint32_t start, uint32_t end)
```

Power down RAM memory blocks.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	start	The start address of the RAM region to power down. This address is inclusive.
uint32_t	[in]	end	The end address of the RAM region to power down. This address is exclusive. If this parameter is 0, all RAM blocks contained in the region from start to the upper RAM address will be powered down.

This function will power down all the RAM blocks that are within a given range. The RAM block layout is different between device families, so this function can be used in a generic way to power down a RAM memory region which is known to be unused.

This function will only power down blocks which are completely enclosed by the memory range given by [start, end).

This is an example to power down all RAM blocks except the first one. The first RAM block is special in that it cannot be powered down by the hardware. The size of the first RAM block is device-specific. See the reference manual to find the RAM block sizes.

```
EMU_RamPowerDown(SRAM_BASE, SRAM_BASE + SRAM_SIZE);
```

Note

- Only a reset can power up the specified memory block(s) after power down on a series 0 device. The specified memory block(s) will stay off until a call to [EMU_RamPowerUp\(\)](#) is done on series 1/2.

EMU_RamPowerUp

```
void EMU_RamPowerUp (void )
```

Power up all available RAM memory blocks.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will power up all the RAM blocks on a device, this means that the RAM blocks are retained in EM2/EM3. Note that this functionality is not supported on Series 0 devices. Only a reset will power up the RAM

blocks on a series 0 device.

EMU_UpdateOscConfig

```
void EMU_UpdateOscConfig (void )
```

Update EMU module with CMU oscillator selection/enable status.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

EMU_EFPEM01VScale

```
void EMU_EFPEM01VScale (EMU_VScaleEM01_TypeDef voltage)
```

Energy mode 01 voltage scaling hook function.

Parameters

Type	Direction	Argument Name	Description
EMU_VScaleEM01_TypeDef	[in]	voltage	Voltage scaling level requested.

This function is called by EMU_VScaleEM01 to let EFP know that voltage scaling is requested.

EMU_VScaleEM01ByClock

```
void EMU_VScaleEM01ByClock (uint32_t clockFrequency, bool wait)
```

Voltage scale in EM0 and 1 by clock frequency.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	clockFrequency	Use CMSIS HF clock if 0 or override to custom clock. Providing a custom clock frequency is required if using a non-standard HFXO frequency.
bool	[in]	wait	Wait for scaling to complete.

Note

- This function is primarily needed by the [CMU - Clock Management Unit](#).

EMU_VScaleEM01

```
void EMU_VScaleEM01 (EMU_VScaleEM01_TypeDef voltage, bool wait)
```

Force voltage scaling in EM0 and 1 to a specific voltage level.

Parameters

Type	Direction	Argument Name	Description
EMU_VScaleEM01_TypeDef	[in]	voltage	Target VSCALE voltage level.
bool	[in]	wait	Wait for scaling to complete.

Note

- This function is useful for upscaling before programming Flash from [MSC - Memory System Controller](#) and downscaling after programming is done. Flash programming is only supported at [emuVScaleEM01_HighPerformance](#).
- This function ignores vScaleEM01LowPowerVoltageEnable set from [EMU_EM01Init\(\)](#).

EMU_DCDCModeSet

```
sl_status_t EMU_DCDCModeSet (EMU\_DcdcMode\_TypeDef dcdcMode)
```

Set DCDC regulator operating mode.

Parameters

Type	Direction	Argument Name	Description
EMU_DcdcMode_TypeDef	[in]	dcdcMode	DCDC mode.

Returns

- Returns the status of the DCDC mode set operation.

```
* SL_STATUS_OK - Operation completed successfully.
* SL_STATUS_TIMEOUT - Operation EMU DCDC set mode timeout.
*
```

EMU_DCDCUpdatedHook

```
void EMU_DCDCUpdatedHook (void )
```

Indicate that the DCDC peripheral bus clock enable has changed allowing RAIL to react accordingly.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is called after DCDC has been enabled or disabled. The function implementation does not perform anything, but it is SL_WEAK so that it can use the RAIL version if needed.

EMU_DCDCInit

```
bool EMU_DCDCInit (const EMU\_DCDCInit\_TypeDef * dcdcInit)
```

Configure the DCDC regulator.

Parameters

Type	Direction	Argument Name	Description
const EMU_DCDCInit_TypeDef *	[in]	dcdcInit	The DCDC initialization structure.

Returns

- True if initialization parameters are valid.

EMU_DCDCPowerOff

```
bool EMU_DCDCPowerOff (void )
```

Power off the DCDC regulator.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Returns true.

EMU_EM01PeakCurrentSet

```
void EMU_EM01PeakCurrentSet (const EMU\_DcdcPeakCurrent\_TypeDef peakCurrentEM01)
```

Set EM01 mode Peak Current setting.

Parameters

Type	Direction	Argument Name	Description
const EMU_DcdcPeakCurrent_TypeDef	[in]	peakCurrentEM01	Peak load current coefficient in EM01 mode.

EMU_TemperatureGet

```
float EMU_TemperatureGet (void )
```

Get temperature in degrees Celsius.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Temperature in degrees Celsius

EMU_EFPDirectModeEnable

```
void EMU_EFPDirectModeEnable (bool enable)
```

Enable/disable EFP Direct Mode.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	True to enable direct mode.

EMU_EFPDriveDecoupleSet

```
void EMU_EFPDriveDecoupleSet (bool enable)
```

Set to enable EFP to drive Decouple voltage.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	True to enable EFP to drive Decouple voltage.

Once set, internal LDO will be disabled, and the EMU will control EFP for voltage-scaling. Note that because this bit disables the internal LDO powering the core, it should not be set until after EFP's DECOUPLE output has been configured and enabled.

EMU_EFPDriveDvddSet

```
void EMU_EFPDriveDvddSet (bool enable)
```

Set to enable EFP to drive DVDD voltage.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	True to enable EFP to drive DVDD voltage.

Set this if EFP's DCDC output is powering DVDD supply. This mode assumes that internal DCDC is not being used.

EMU_DCDCLock

```
void EMU_DCDCLock (void )
```

Lock DCDC registers in order to protect them against unintended modification.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

EMU_DCDCUnlock

```
void EMU_DCDCUnlock (void )
```

Unlock the DCDC so that writing to locked registers again is possible.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

EMU_EnterEM1

```
void EMU_EnterEM1 (void )
```

Enter energy mode 1 (EM1).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function is incompatible with the Power Manager module. When the Power Manager module is present, it must be the one deciding at which EM level the device sleeps to ensure the application properly works. Using both at the same time could lead to undefined behavior in the application.

EMU_VScaleWait

```
void EMU_VScaleWait (void )
```

Wait for voltage scaling to complete.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

EMU_VScaleGet

```
EMU_VScaleEM01_TypeDef EMU_VScaleGet (void )
```

Get current voltage scaling level.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Current voltage scaling level.

EMU_IntClear

```
void EMU_IntClear (uint32_t flags)
```

Clear one or more pending EMU interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	Pending EMU interrupt sources to clear. Use one or more valid interrupt flags for the EMU module (EMU_IFC_nnn or EMU_IF_nnn).

EMU_IntDisable

```
void EMU_IntDisable (uint32_t flags)
```

Disable one or more EMU interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	EMU interrupt sources to disable. Use one or more valid interrupt flags for the EMU module (EMU_IEN_nnn).

EMU_IntEnable

```
void EMU_IntEnable (uint32_t flags)
```

Enable one or more EMU interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	EMU interrupt sources to enable. Use one or more valid interrupt flags for the EMU module (EMU_IEN_nnn).

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [EMU_IntClear\(\)](#) prior to enabling the interrupt.

EMU_EFPIntDisable

```
void EMU_EFPIntDisable (uint32_t flags)
```

Disable one or more EFP interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	EFP interrupt sources to disable. Use one or more valid interrupt flags for the EFP module (EFPIENnnn).

EMU_EFPIntEnable

```
void EMU_EFPIntEnable (uint32_t flags)
```

Enable one or more EFP interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	EFP interrupt sources to enable. Use one or more valid interrupt flags for the EFP module (EFPIENnnn).

EMU_EFPIntGet

```
uint32_t EMU_EFPIntGet (void )
```

Get pending EMU EFP interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Event bits are not cleared by the use of this function.

Returns

- EMU EFP interrupt sources pending. .

EMU_EFPIntGetEnabled

```
uint32_t EMU_EFPIntGetEnabled (void )
```

Get enabled and pending EMU EFP interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

- Pending and enabled EMU EFP interrupt sources Return value is the bitwise AND of
 - the enabled interrupt sources in EMU_EFPIEN and
 - the pending interrupt flags EMU_EFPIF.

EMU_EFPIntSet

```
void EMU_EFPIntSet (uint32_t flags)
```

Set one or more pending EMU EFP interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	EMU EFP interrupt sources to set to pending. Use one or more valid interrupt flags for the EMU EFP module (EMU_EFPIFSnnn).

EMU_EFPIntClear

```
void EMU_EFPIntClear (uint32_t flags)
```

Clear one or more pending EMU EFP interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	Pending EMU EFP interrupt sources to clear. Use one or more valid interrupt flags for the EMU EFP module.

EMU_IntGet

```
uint32_t EMU_IntGet (void )
```

Get pending EMU interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Event bits are not cleared by the use of this function.

Returns

- EMU interrupt sources pending. Returns one or more valid interrupt flags for the EMU module (EMU_IF_nnn).

EMU_IntGetEnabled

```
uint32_t EMU_IntGetEnabled (void )
```

Get enabled and pending EMU interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

- Pending and enabled EMU interrupt sources Return value is the bitwise AND of
 - the enabled interrupt sources in EMU_IEN and

the pending interrupt flags EMU_IF.

EMU_IntSet

```
void EMU_IntSet (uint32_t flags)
```

Set one or more pending EMU interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	EMU interrupt sources to set to pending. Use one or more valid interrupt flags for the EMU module (EMU_IFS_nnn).

EMU_Lock

```
void EMU_Lock (void )
```

Lock EMU registers in order to protect them against unintended modification.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- If locking EMU registers, they must be unlocked prior to using any EMU API functions modifying EMU registers, excluding interrupt control and regulator control if the architecture has a EMU_PWRCTRL register. An exception to this is the energy mode entering API (EMU_EnterEMn()), which can be used when the EMU registers are locked.

EMU_Unlock

```
void EMU_Unlock (void )
```

Unlock the EMU so that writing to locked registers again is possible.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

EMU_UnlatchPinRetention

```
void EMU_UnlatchPinRetention (void )
```

When EM4 pin retention is set to emuPinRetentionLatch, then pins are retained through EM4 entry and wakeup.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The pin state is released by calling this function. The feature allows peripherals or GPIO to be re-initialized after EM4 exit (reset), and when initialization is done, this function can release pins and return control to the peripherals or GPIO.

EMU_TemperatureReady

```
bool EMU_TemperatureReady (void )
```

Temperature measurement ready status.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if temperature measurement is ready

EMU_TemperatureAvgGet

```
float EMU_TemperatureAvgGet (void )
```

Get averaged temperature in degrees Celsius.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- An averaged temperature measurement must first be requested by calling [EMU_TemperatureAvgRequest\(\)](#) and waiting for the TEMPAVG interrupt flag to go high.

Returns

- Averaged temperature

EMU_TemperatureAvgRequest

```
void EMU_TemperatureAvgRequest (EMU_TempAvgNum_TypeDef numSamples)
```

Request averaged temperature.

Parameters

Type	Direction	Argument Name	Description
EMU_TempAvgNum_TypeDef	[in]	numSamples	Number of temperature samples to average

Note

- EMU must be unlocked by calling [EMU_Unlock\(\)](#) before this function can be called.

Macro Definition Documentation

EMU_VSCALE_PRESENT

```
#define EMU_VSCALE_PRESENT
```

Voltage scaling present.

EMU_VSCALE_EM01_PRESENT

```
#define EMU_VSCALE_EM01_PRESENT
```

Voltage scaling for EM01 present.

EMU_SERIES2_DCDC_BUCK_PRESENT

```
#define EMU_SERIES2_DCDC_BUCK_PRESENT
```

DC-DC buck converter present.

EMU_EM01INIT_DEFAULT

```
#define EMU_EM01INIT_DEFAULT
```

Value:

```
0 | {
0 |     false
0 | } /* Do not scale down in EM0/1.*/ \
```

Default initialization of EM0 and 1 configuration.

EMU_EM23INIT_DEFAULT

```
#define EMU_EM23INIT_DEFAULT
```

Value:

```
0 | {
0 |     false,
0 |     emuVScaleEM23_FastWakeup,
0 | } /* Reduced voltage regulator drive strength in EM2/3.*/ \
/* Do not scale down in EM2/3. */ \
```

Default initialization of EM2 and 3 configuration.

EMU_EM4INIT_DEFAULT

```
#define EMU_EM4INIT_DEFAULT
```

Value:

```
0 | {
0 |     \
```

```

0 | false,          /* Retain LFX0 configuration upon EM4 entry. */ \
0 | false,          /* Retain LFRC0 configuration upon EM4 entry. */ \
0 | false,          /* Retain ULFR0 configuration upon EM4 entry. */ \
0 | emuEM4Shutoff,  /* Use EM4 shutoff state. */ \
0 | emuPinRetentionDisable, /* Do not retain pins in EM4. */ \
0 | }

```

Default initialization of EM4 configuration (Series 1 without VSCALE).

EMU_DCDCINIT_DEFAULT

```
#define EMU_DCDCINIT_DEFAULT
```

Value:

```

0 | {
0 |     emuDcdcMode_Regulation,      /**< DCDC regulator on. */ \
0 |     emuVreginCmpThreshold_2v3,  /**< 2.3V VREGIN comparator threshold. */ \
0 |     emuDcdcTonMaxTimeout_1P19us, /**< Ton max is 1.19us. */ \
0 |     true,                       /**< Enable DCM only mode. */ \
0 |     emuDcdcDriveSpeed_Default,  /**< Default efficiency in EM0/1. */ \
0 |     emuDcdcDriveSpeed_Default,  /**< Default efficiency in EM2/3. */ \
0 |     emuDcdcPeakCurrent_Load60mA, /**< Default peak current in EM0/1. */ \
0 |     emuDcdcPeakCurrent_Load5mA  /**< Default peak current in EM2/3. */ \
0 | }

```

Default DCDC Buck initialization.

EMU_TEMP_ZERO_C_IN_KELVIN

```
#define EMU_TEMP_ZERO_C_IN_KELVIN
```

Value:

```
(273.15f)
```

Zero degrees Celcius in Kelvin.

EMU_EM01Init_TypeDef

EM0 and 1 initialization structure.

Voltage scaling is applied when the core clock frequency is changed from [CMU - Clock Management Unit](#). EM0 and 1 emuVScaleEM01_HighPerformance is always enabled.

Public Attributes

bool [vScaleEM01LowPowerVoltageEnable](#)
EM0/1 low power voltage status.

Public Attribute Documentation

vScaleEM01LowPowerVoltageEnable

```
bool EMU_EM01Init_TypeDef::vScaleEM01LowPowerVoltageEnable
```

EM0/1 low power voltage status.

EMU_EM23Init_TypeDef

EM2 and 3 initialization structure.

Public Attributes

bool	em23VregFullEn
	Enable full VREG drive strength in EM2/3.
EMU_VScaleEM23_TypeDef	vScaleEM23Voltage
	EM2/3 voltage scaling level.

Public Attribute Documentation

em23VregFullEn

```
bool EMU_EM23Init_TypeDef::em23VregFullEn
```

Enable full VREG drive strength in EM2/3.

vScaleEM23Voltage

```
EMU_VScaleEM23_TypeDef EMU_EM23Init_TypeDef::vScaleEM23Voltage
```

EM2/3 voltage scaling level.

EMU_EM4Init_TypeDef

EM4 initialization structure.

Public Attributes

	bool	retainLfxx	Disable LFXO upon EM4 entry.
	bool	retainLfrco	Disable LFRCO upon EM4 entry.
	bool	retainUlfrco	Disable ULFRCO upon EM4 entry.
EMU_EM4State_TypeDef		em4State	Hibernate or shutoff EM4 state.
EMU_EM4PinRetention_TypeDef		pinRetentionMode	EM4 pin retention mode.

Public Attribute Documentation

retainLfxx

```
bool EMU_EM4Init_TypeDef::retainLfxx
```

Disable LFXO upon EM4 entry.

retainLfrco

```
bool EMU_EM4Init_TypeDef::retainLfrco
```

Disable LFRCO upon EM4 entry.

retainUlfrco

```
bool EMU_EM4Init_TypeDef::retainUlfrco
```

Disable ULFRCO upon EM4 entry.

em4State

```
EMU_EM4State_TypeDef EMU_EM4Init_TypeDef::em4State
```

Hibernate or shutoff EM4 state.

pinRetentionMode

EMU_EM4PinRetention_TypeDef EMU_EM4Init_TypeDef::pinRetentionMode

EM4 pin retention mode.

EMU_DCDCInit_TypeDef

DCDC regulator initialization structure.

Public Attributes

EMU_DcdcMode_TypeDef	mode	DCDC mode.
EMU_VreginCmpThreshold_TypeDef	cmpThreshold	VREGIN comparator threshold.
EMU_DcdcTonMaxTimeout_TypeDef	tonMax	Ton max timeout control.
bool	dcmOnlyEn	DCM only mode enable.
EMU_DcdcDriveSpeed_TypeDef	driveSpeedEM01	DCDC drive speed in EM0/1.
EMU_DcdcDriveSpeed_TypeDef	driveSpeedEM23	DCDC drive speed in EM2/3.
EMU_DcdcPeakCurrent_TypeDef	peakCurrentEM01	EM0/1 peak current setting.
EMU_DcdcPeakCurrent_TypeDef	peakCurrentEM23	EM2/3 peak current setting.

Public Attribute Documentation

mode

EMU_DcdcMode_TypeDef EMU_DCDCInit_TypeDef::mode

DCDC mode.

cmpThreshold

EMU_VreginCmpThreshold_TypeDef EMU_DCDCInit_TypeDef::cmpThreshold

VREGIN comparator threshold.

tonMax

EMU_DcdcTonMaxTimeout_TypeDef EMU_DCDCInit_TypeDef::tonMax

Ton max timeout control.

dcmOnlyEn

```
bool EMU_DCDCInit_TypeDef::dcmOnlyEn
```

DCM only mode enable.

driveSpeedEM01

```
EMU_DcdcDriveSpeed_TypeDef EMU_DCDCInit_TypeDef::driveSpeedEM01
```

DCDC drive speed in EM0/1.

driveSpeedEM23

```
EMU_DcdcDriveSpeed_TypeDef EMU_DCDCInit_TypeDef::driveSpeedEM23
```

DCDC drive speed in EM2/3.

peakCurrentEM01

```
EMU_DcdcPeakCurrent_TypeDef EMU_DCDCInit_TypeDef::peakCurrentEM01
```

EM0/1 peak current setting.

peakCurrentEM23

```
EMU_DcdcPeakCurrent_TypeDef EMU_DCDCInit_TypeDef::peakCurrentEM23
```

EM2/3 peak current setting.

EUSART - Extended USART

EUSART - Extended USART

Extended Universal Synchronous/Asynchronous Receiver/Transmitter.

- [Introduction](#)
- [Example](#)
- [EM2 guidelines for non EM2-Capable instances](#)

Introduction

This module contains functions to control the Enhanced Universal Synchronous / Asynchronous Receiver / Transmitter controller(s) (EUSART) peripheral of Silicon Labs' 32-bit MCUs and SoCs. EUSART can be used as a UART and can, therefore, be connected to an external transceiver to communicate with another host using the serial link.

It supports full duplex asynchronous UART communication as well as RS-485, SPI, MicroWire, and 3-wire. It can also interface with ISO7816 Smart-Cards, and IrDA devices.

EUSART has a wide selection of operating modes, frame formats, and baud rates. All features are supported through the API of this module.

This module does not support DMA configuration. UARTDRV and SPIDRV drivers provide full support for DMA and more.

Example

EUSART Async TX example:

```
{
    EUSART_UartInit_TypeDef init = EUSART_UART_INIT_DEFAULT_HF;

    // Configure the clocks.
    CMU_ClockSelectSet(cmuClock_EUSART0CLK, cmuSelect_EM01GRPCCLK);
    CMU_ClockEnable(cmuClock_EUSART0CLK, true);
    // Initialize the EUSART
    EUSART_UartInitHf(EUSART0, &init);
    EUSART_Tx(EUSART0, data);
}
```

EUSART Sync SPI Transaction example:

```

{
    EUSART_SpiInit_TypeDef init_master = EUSART_SPI_MASTER_INIT_DEFAULT_HF;

    // Configure the clocks.
    CMU_ClockSelectSet(cmuClock_EM01GRPCCLK, cmuSelect_HFRCODPLL);
    CMU_ClockEnable(cmuClock_EUSART1, true);
    CMU_ClockEnable(cmuClock_GPIO, true);

    //Configure the SPI ports
    GPIO_PinModeSet(sclk_port, sclk_pin, gpioModePushPull, 0);
    GPIO_PinModeSet(mosi_port, mosi_pin, gpioModePushPull, 0);
    GPIO_PinModeSet(mosi_port, miso_pin, gpioModeInput, 0);

    // Connect EUSART to ports
    GPIO->EUSARTROUTE[EUSART_NUM(EUSART1)].TXROUTE = (mosi_port << _GPIO_EUSART_TXROUTE_PORT_SHIFT)
                                                    | (mosi_pin << _GPIO_EUSART_TXROUTE_PIN_SHIFT);
    GPIO->EUSARTROUTE[EUSART_NUM(EUSART1)].RXROUTE = (miso_port << _GPIO_EUSART_RXROUTE_PORT_SHIFT)
                                                    | (miso_pin << _GPIO_EUSART_RXROUTE_PIN_SHIFT);
    GPIO->EUSARTROUTE[EUSART_NUM(EUSART1)].SCLKROUTE = (sclk_port << _GPIO_EUSART_SCLKROUTE_PORT_SHIFT)
                                                    | (sclk_pin << _GPIO_EUSART_SCLKROUTE_PIN_SHIFT);
    GPIO->EUSARTROUTE[EUSART_NUM(EUSART1)].ROUTEEN = GPIO_EUSART_ROUTEEN_TXPEN | GPIO_EUSART_ROUTEEN_SCLKPEN;

    // Initialize the EUSART
    EUSART_SpiInit(EUSART1, &init_master);
    EUSART_Spi_TxRx(EUSART1, data);
}

```

EM2 guidelines for non EM2-Capable instances

Note

- EUSART instances located in the PD1 power domain are non EM2-capable. The EUSART_EM2_CAPABLE() and EUSART_NOT_EM2_CAPABLE() macros can be used to determine whether or not a EUSART instance is EM2-Capable.

Follow these steps when entering in EM2:

1. Wait for the current transaction to complete with TXCIF interrupt
2. Disable TX and RX using TXDIS and RXDIS cmd
3. Poll for EUSARTn_SYNCBUSY.TXDIS and EUSARTn_SYNCBUSY.RXDIS to go low
4. Wait for EUSARTn_STATUS.TXENS and EUSARTn_STATUS.RXENS to go low
5. Disable SCLKPEN and CSPEN in GPIO if they were previously enabled
6. Enter EM2

On wakeup from EM2, EUSART transmitter/receiver and relevant GPIO (SCLKPEN and CSPEN) must be re-enabled. For example:

```

{
    // Enable TX and RX
    EUSART_Enable(EUSART0, eusartEnable);
    BUS_RegMaskedWrite(&GPIO->EUSARTROUTE[EUSART_NUM(EUSART0)].ROUTEEN,
                      _GPIO_EUSART_ROUTEEN_TXPEN_MASK | _GPIO_EUSART_ROUTEEN_SCLKPEN_MASK,
                      GPIO_EUSART_ROUTEEN_TXPEN | GPIO_EUSART_ROUTEEN_SCLKPEN);
}

```

Modules

[EUSART_AdvancedInit_TypeDef](#)

[EUSART_UartInit_TypeDef](#)
[EUSART_IrDAInit_TypeDef](#)
[EUSART_PrsTriggerInit_TypeDef](#)
[EUSART_DaliInit_TypeDef](#)

Enumerations

```
enum    EUSART_Enable_TypeDef {
    eusartDisable = 0x0
    eusartEnableRx = (EUSART_CMD_RXEN | EUSART_CMD_TXDIS)
    eusartEnableTx = (EUSART_CMD_TXEN | EUSART_CMD_RXDIS)
    eusartEnable = (EUSART_CMD_RXEN | EUSART_CMD_TXEN)
}
Enable selection.

enum    EUSART_Databits_TypeDef {
    eusartDataBits7 = EUSART_FRAMECFG_DATABITS_SEVEN
    eusartDataBits8 = EUSART_FRAMECFG_DATABITS_EIGHT
    eusartDataBits9 = EUSART_FRAMECFG_DATABITS_NINE
}
Data bit selection.

enum    EUSART_Parity_TypeDef {
    eusartNoParity = EUSART_FRAMECFG_PARITY_NONE
    eusartEvenParity = EUSART_FRAMECFG_PARITY_EVEN
    eusartOddParity = EUSART_FRAMECFG_PARITY_ODD
}
Parity selection.

enum    EUSART_Stopbits_TypeDef {
    eusartStopbitsOp5 = EUSART_FRAMECFG_STOPBITS_HALF
    eusartStopbits1p5 = EUSART_FRAMECFG_STOPBITS_ONEANDAHALF
    eusartStopbits1 = EUSART_FRAMECFG_STOPBITS_ONE
    eusartStopbits2 = EUSART_FRAMECFG_STOPBITS_TWO
}
Stop bits selection.

enum    EUSART_OVS_TypeDef {
    eusartOVS16 = EUSART_CFG0_OVS_X16
    eusartOVS8 = EUSART_CFG0_OVS_X8
    eusartOVS6 = EUSART_CFG0_OVS_X6
    eusartOVS4 = EUSART_CFG0_OVS_X4
    eusartOVS0 = EUSART_CFG0_OVS_DISABLE
}
Oversampling selection, used for asynchronous operation.

enum    EUSART_HwFlowControl_TypeDef {
    eusartHwFlowControlNone = 0
    eusartHwFlowControlCts
    eusartHwFlowControlRts
    eusartHwFlowControlCtsAndRts
}
HW flow control config.

enum    EUSART_LoopbackEnable_TypeDef {
    eusartLoopbackEnable = EUSART_CFG0_LOOPBK
    eusartLoopbackDisable = _EUSART_CFG0_RESETVALUE
}
Loopback enable.
```

```

enum    EUSART_MajorityVote_TypeDef {
        eusartMajorityVoteEnable = EUSART_CFG0_MVDIS_DEFAULT
        eusartMajorityVoteDisable = EUSART_CFG0_MVDIS
    }
    Majority vote enable.

enum    EUSART_BlockRx_TypeDef {
        eusartBlockRxEnable = EUSART_CMD_RXBLOCKEN
        eusartBlockRxDisable = EUSART_CMD_RXBLOCKDIS
    }
    Block reception enable.

enum    EUSART_TristateTx_TypeDef {
        eusartTristateTxEnable = EUSART_CMD_TXTRIEN
        eusartTristateTxDisable = EUSART_CMD_TXTRIDIS
    }
    TX output tristate enable.

enum    EUSART_IrDARxFilterEnable_TypeDef {
        eusartIrDARxFilterEnable = EUSART_IRHFCFG_IRHFFILT_ENABLE
        eusartIrDARxFilterDisable = EUSART_IRHFCFG_IRHFFILT_DISABLE
    }
    IrDA filter enable.

enum    EUSART_IrDAPulseWidth_Typedef {
        eusartIrDAPulseWidthOne = EUSART_IRHFCFG_IRHFPW_ONE
        eusartIrDAPulseWidthTwo = EUSART_IRHFCFG_IRHFPW_TWO
        eusartIrDAPulseWidthThree = EUSART_IRHFCFG_IRHFPW_THREE
        eusartIrDAPulseWidthFour = EUSART_IRHFCFG_IRHFPW_FOUR
    }
    Pulse width selection for IrDA mode.

enum    EUSART_PrsTriggerEnable_TypeDef {
        eusartPrsTriggerDisable = 0x0
        eusartPrsTriggerEnableRx = EUSART_TRIGCTRL_RXTEN
        eusartPrsTriggerEnableTx = EUSART_TRIGCTRL_TXTEN
        eusartPrsTriggerEnableRxCx = (EUSART_TRIGCTRL_RXTEN | EUSART_TRIGCTRL_TXTEN)
    }
    PRS trigger enable.

enum    EUSART_InvertIO_TypeDef {
        eusartInvertIODisable = (EUSART_CFG0_RXINV_DISABLE | EUSART_CFG0_TXINV_DISABLE)
        eusartInvertRxEnable = EUSART_CFG0_RXINV_ENABLE
        eusartInvertTxEnable = EUSART_CFG0_TXINV_ENABLE
        eusartInvertIOEnable = (EUSART_CFG0_RXINV_ENABLE | EUSART_CFG0_TXINV_ENABLE)
    }
    IO polarity selection.

enum    EUSART_AutoTxDelay_TypeDef {
        eusartAutoTxDelayNone = EUSART_TIMINGCFG_TXDELAY_NONE
        eusartAutoTxDelaySingle = EUSART_TIMINGCFG_TXDELAY_SINGLE
        eusartAutoTxDelayDouble = EUSART_TIMINGCFG_TXDELAY_DOUBLE
        eusartAutoTxDelayTripple = EUSART_TIMINGCFG_TXDELAY_TRIPPLE
    }
    Auto TX delay transmission.

enum    EUSART_RxFifoWatermark_TypeDef {
        eusartRxFiFoWatermark1Frame = EUSART_CFG1_RXFIW_ONEFRAME
        eusartRxFiFoWatermark2Frame = EUSART_CFG1_RXFIW_TWOFRAMES
        eusartRxFiFoWatermark3Frame = EUSART_CFG1_RXFIW_THREEFRAMES
        eusartRxFiFoWatermark4Frame = EUSART_CFG1_RXFIW_FOURFRAMES
    }
    RX FIFO Interrupt ans Status Watermark.

```

```
enum EUSART\_TxFifoWatermark\_TypeDef {
    eusartTxFifoWatermark1Frame = EUSART_CFG1_TXFIW_ONEFRAME
    eusartTxFifoWatermark2Frame = EUSART_CFG1_TXFIW_TWOFRAMES
    eusartTxFifoWatermark3Frame = EUSART_CFG1_TXFIW_THREEFRAMES
    eusartTxFifoWatermark4Frame = EUSART_CFG1_TXFIW_FOURFRAMES
}
TX FIFO Interrupt and Status Watermark.
```

Typedefs

```
typedef uint8_t EUSART\_PrsChannel\_TypeDef
PRS Channel type.
```

Functions

```
void EUSART\_UartInitHf(EUSART_TypeDef *eusart, const EUSART_UartInit_TypeDef *init)
Initialize EUSART when used in UART mode with the high frequency clock.

void EUSART\_UartInitLf(EUSART_TypeDef *eusart, const EUSART_UartInit_TypeDef *init)
Initialize EUSART when used in UART mode with the low frequency clock.

void EUSART\_IrDAInit(EUSART_TypeDef *eusart, const EUSART_IrDAInit_TypeDef *irdaInit)
Initialize EUSART when used in IrDA mode with the high or low frequency clock.

void EUSART\_Reset(EUSART_TypeDef *eusart)
Configure EUSART to its reset state.

void EUSART\_Enable(EUSART_TypeDef *eusart, EUSART_Enable_TypeDef enable)
Enable/disable EUSART receiver and/or transmitter.

uint8_t EUSART\_Rx(EUSART_TypeDef *eusart)
Receive one 8 bit frame, (or part of 9 bit frame).

uint16_t EUSART\_RxExt(EUSART_TypeDef *eusart)
Receive one 8-16 bit frame with extended information.

void EUSART\_Tx(EUSART_TypeDef *eusart, uint8_t data)
Transmit one frame.

void EUSART\_TxExt(EUSART_TypeDef *eusart, uint16_t data)
Transmit one 8-9 bit frame with extended control.

void EUSART\_BaudrateSet(EUSART_TypeDef *eusart, uint32_t refFreq, uint32_t baudrate)
Configure the baudrate (or as close as possible to a specified baudrate).

uint32_t EUSART\_BaudrateGet(EUSART_TypeDef *eusart)
Get the current baudrate.

void EUSART\_RxBlock(EUSART_TypeDef *eusart, EUSART_BlockRx_TypeDef enable)
Enable/Disable reception operation until the configured start frame is received.

void EUSART\_TxTristateSet(EUSART_TypeDef *eusart, EUSART_TristateTx_TypeDef enable)
Enable/Disable the tristating of the transmitter output.

void EUSART\_PrsTriggerEnable(EUSART_TypeDef *eusart, const EUSART_PrsTriggerInit_TypeDef *init)
Initialize the automatic enabling of transmissions and/or reception using the PRS as a trigger.

uint32_t EUSART\_StatusGet(EUSART_TypeDef *eusart)
Get EUSART STATUS register.

void EUSART\_IntClear(EUSART_TypeDef *eusart, uint32_t flags)
Clear one or more pending EUSART interrupts.
```

void	EUSART_IntDisable (EUSART_TypeDef *eusart, uint32_t flags) Disable one or more EUSART interrupts.
void	EUSART_IntEnable (EUSART_TypeDef *eusart, uint32_t flags) Enable one or more EUSART interrupts.
uint32_t	EUSART_IntGet (EUSART_TypeDef *eusart) Get pending EUSART interrupt flags.
uint32_t	EUSART_IntGetEnabled (EUSART_TypeDef *eusart) Get enabled and pending EUSART interrupt flags.
void	EUSART_IntSet (EUSART_TypeDef *eusart, uint32_t flags) Set one or more pending EUSART interrupts from SW.

Macros

#define	EUSART0_FIFO_DEPTH 4 Define EUSART FIFO Depth information.
#define	EUSART1_FIFO_DEPTH EUSART0_FIFO_DEPTH
#define	EUSART2_FIFO_DEPTH EUSART0_FIFO_DEPTH
#define	EUSART3_FIFO_DEPTH EUSART0_FIFO_DEPTH
#define	EUSART4_FIFO_DEPTH EUSART0_FIFO_DEPTH
#define	EUSART_FIFO_DEPTH (n)
#define	EUSART_UART_INIT_DEFAULT_HF undefined Default configuration for EUSART initialization structure in UART mode with high-frequency clock.
#define	EUSART_DEFAULT_START_FRAME 0x00u Default start frame configuration, i.e. feature disabled.
#define	EUSART_ADVANCED_INIT_DEFAULT undefined Default configuration for EUSART advanced initialization structure.
#define	EUSART_UART_INIT_DEFAULT_LF undefined Default configuration for EUSART initialization structure in UART mode with low-frequency clock.
#define	EUSART_IRDA_INIT_DEFAULT_HF undefined Default configuration for EUSART initialization structure in IrDA mode with high-frequency clock.
#define	EUSART_IRDA_INIT_DEFAULT_LF undefined Default configuration for EUSART initialization structure in IrDA mode with low-frequency clock.

Enumeration Documentation

EUSART_Enable_TypeDef

EUSART_Enable_TypeDef

Enable selection.

Enumerator	
eusartDisable	Disable the peripheral.
eusartEnableRx	Enable receiver only, transmitter disabled.
eusartEnableTx	Enable transmitter only, receiver disabled.
eusartEnable	Enable both receiver and transmitter.

EUSART_Databits_TypeDef

EUSART_Databits_TypeDef

Data bit selection.

Enumerator	
eusartDataBits7	7 data bits.
eusartDataBits8	8 data bits.
eusartDataBits9	9 data bits.

EUSART_Parity_TypeDef

EUSART_Parity_TypeDef

Parity selection.

Enumerator	
eusartNoParity	No parity.
eusartEvenParity	Even parity.
eusartOddParity	Odd parity.

EUSART_Stopbits_TypeDef

EUSART_Stopbits_TypeDef

Stop bits selection.

Enumerator	
eusartStopbits0p5	0.5 stop bits.
eusartStopbits1p5	1.5 stop bits.
eusartStopbits1	1 stop bits.
eusartStopbits2	2 stop bits.

EUSART_OVS_TypeDef

EUSART_OVS_TypeDef

Oversampling selection, used for asynchronous operation.

Enumerator	
eusartOVS16	16x oversampling (normal).
eusartOVS8	8x oversampling.
eusartOVS6	6x oversampling.
eusartOVS4	4x oversampling.
eusartOVS0	Oversampling disabled.

EUSART_HwFlowControl_TypeDef

EUSART_HwFlowControl_TypeDef

HW flow control config.

Enumerator

eusartHwFlowControlNone	No HW Flow Control.
eusartHwFlowControlCts	CTS HW Flow Control.
eusartHwFlowControlRts	RTS HW Flow Control.
eusartHwFlowControlCtsAndRts	CTS and RTS HW Flow Control.

EUSART_LoopbackEnable_TypeDef

EUSART_LoopbackEnable_TypeDef

Loopback enable.

Enumerator

eusartLoopbackEnable	Enable loopback.
eusartLoopbackDisable	Disable loopback.

EUSART_MajorityVote_TypeDef

EUSART_MajorityVote_TypeDef

Majority vote enable.

Enumerator

eusartMajorityVoteEnable	Enable majority vote for 16x, 8x and 6x oversampling modes.
eusartMajorityVoteDisable	Disable majority vote for 16x, 8x and 6x oversampling modes.

EUSART_BlockRx_TypeDef

EUSART_BlockRx_TypeDef

Block reception enable.

Enumerator

eusartBlockRxEnable	Block reception enable, resulting in all incoming frames being discarded.
eusartBlockRxDisable	Block reception disable, resulting in all incoming frames being loaded into the RX FIFO.

EUSART_TristateTx_TypeDef

EUSART_TristateTx_TypeDef

TX output tristate enable.

Enumerator

eusartTristateTxEnable	Tristates the transmitter output.
------------------------	-----------------------------------

eusartTristateTxDisable	Disables tristating of the transmitter output.
-------------------------	--

EUSART_IrDARxFilterEnable_TypeDef

EUSART_IrDARxFilterEnable_TypeDef	
IrDA filter enable.	
Enumerator	
eusartIrDARxFilterEnable	Enable filter on demodulator.
eusartIrDARxFilterDisable	Disable filter on demodulator.

EUSART_IrDAPulseWidth_Typedef

EUSART_IrDAPulseWidth_Typedef	
Pulse width selection for IrDA mode.	
Enumerator	
eusartIrDAPulseWidthOne	IrDA pulse width is 1/16 for OVS=X16 and 1/8 for OVS=X8.
eusartIrDAPulseWidthTwo	IrDA pulse width is 2/16 for OVS=X16 and 2/8 for OVS=X8.
eusartIrDAPulseWidthThree	IrDA pulse width is 3/16 for OVS=X16 and 3/8 for OVS=X8.
eusartIrDAPulseWidthFour	IrDA pulse width is 4/16 for OVS=X16 and 4/8 for OVS=X8.

EUSART_PrSTriggerEnable_TypeDef

EUSART_PrSTriggerEnable_TypeDef	
PRS trigger enable.	
Enumerator	
eusartPrSTriggerDisable	Disable trigger on both receiver and transmitter.
eusartPrSTriggerEnableRx	Enable receive trigger only, transmit disabled.
eusartPrSTriggerEnableTx	Enable transmit trigger only, receive disabled.
eusartPrSTriggerEnableRXTx	Enable trigger on both receive and transmit.

EUSART_InvertIO_TypeDef

EUSART_InvertIO_TypeDef	
IO polarity selection.	
Enumerator	
eusartInvertIODisable	Disable inversion on both RX and TX signals.
eusartInvertRxEnable	Invert RX signal, before receiver.
eusartInvertTxEnable	Invert TX signal, after transmitter.
eusartInvertIOEnable	Enable trigger on both receive and transmit.

EUSART_AutoTxDelay_TypeDef

EUSART_AutoTxDelay_TypeDef

Auto TX delay transmission.

Enumerator	
eusartAutoTxDelayNone	Frames are transmitted immediately.
eusartAutoTxDelaySingle	Transmission of new frames is delayed by a single bit period.
eusartAutoTxDelayDouble	Transmission of new frames is delayed by a two bit periods.
eusartAutoTxDelayTripple	Transmission of new frames is delayed by a three bit periods.

EUSART_RxFifoWatermark_TypeDef

EUSART_RxFifoWatermark_TypeDef

RX FIFO Interrupt ans Status Watermark.

Enumerator	
eusartRxFiFoWatermark1Frame	
eusartRxFiFoWatermark2Frame	
eusartRxFiFoWatermark3Frame	
eusartRxFiFoWatermark4Frame	

EUSART_TxFifoWatermark_TypeDef

EUSART_TxFifoWatermark_TypeDef

TX FIFO Interrupt and Status Watermark.

Enumerator	
eusartTxFiFoWatermark1Frame	
eusartTxFiFoWatermark2Frame	
eusartTxFiFoWatermark3Frame	
eusartTxFiFoWatermark4Frame	

Typedef Documentation

EUSART_PrsChannel_TypeDef

typedef uint8_t EUSART_PrsChannel_TypeDef

PRS Channel type.

Function Documentation

EUSART_UartInitHf

void EUSART_UartInitHf (EUSART_TypeDef * eusart, const [EUSART_Uartinit_TypeDef](#) * init)

Initialize EUSART when used in UART mode with the high frequency clock.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
const EUSART_UartInit_TypeDef *	N/A	init	A pointer to the initialization structure.

Initialize EUSART when used in UART mode with the high frequency clock.

EUSART_UartInitLf

```
void EUSART_UartInitLf (EUSART_TypeDef * eusart, const EUSART_UartInit_TypeDef * init)
```

Initialize EUSART when used in UART mode with the low frequency clock.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
const EUSART_UartInit_TypeDef *	N/A	init	A pointer to the initialization structure.

Initialize EUSART when used in UART mode with the low frequency clock.

Note

- (1) When EUSART oversampling is set to eusartOVS0 (Disable), the peripheral clock frequency must be at least three times higher than the chosen baud rate. In LF, max input clock is 32768 (LFXO or LFRCO), thus $32768 / 3 \sim 9600$ baudrate.

EUSART_IrDAInit

```
void EUSART_IrDAInit (EUSART_TypeDef * eusart, const EUSART_IrDAInit_TypeDef * irdaInit)
```

Initialize EUSART when used in IrDA mode with the high or low frequency clock.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
const EUSART_IrDAInit_TypeDef *	N/A	irdaInit	A pointer to the initialization structure.

Initialize EUSART when used in IrDA mode with the high or low frequency clock.

EUSART_Reset

```
void EUSART_Reset (EUSART_TypeDef * eusart)
```

Configure EUSART to its reset state.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.

Configure EUSART to its reset state.

EUSART_Enable

```
void EUSART_Enable (EUSART_TypeDef * eusart, EUSART_Enable_TypeDef enable)
```

Enable/disable EUSART receiver and/or transmitter.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
EUSART_Enable_TypeDef	N/A	enable	Select the status for the receiver and transmitter.

Enable/disable EUSART receiver and/or transmitter.

EUSART_Rx

```
uint8_t EUSART_Rx (EUSART_TypeDef * eusart)
```

Receive one 8 bit frame, (or part of 9 bit frame).

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.

Note

- This function is normally used to receive one frame when operating with frame length of 8 bits. See [EUSART_RxExt\(\)](#) for reception of 9 bit frames. Notice that possible parity/stop bits are not considered a part of the specified frame bit length.
- This function will stall if buffer is empty until data is received.

Returns

- Data received.

Receive one 8 bit frame, (or part of 9 bit frame).

Note

- (1) Handles the case where the RX Fifo Watermark has been set to N frames, and when N is greater than one. Attempt to read a frame from the RX Fifo. If the read is unsuccessful (i.e. no frames in the RX fifo), the RXFU interrupt flag is set. If the flag is set, wait to read again until the RXFL status flag is set, indicating there are N frames in the RX Fifo, where N is equal to the RX watermark level. Once there are N frames in the Fifo, read and return one frame. For consecutive N-1 reads there will be data available in the Fifo. Therefore, the RXUF interrupt will not be triggered eliminating delays between reads and sending N data frames in "bursts".

EUSART_RxExt

```
uint16_t EUSART_RxExt (EUSART_TypeDef * eusart)
```

Receive one 8-16 bit frame with extended information.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.

Note

- This function is normally used to receive one frame and additional RX status information.
- This function will stall if buffer is empty until data is received.

Returns

- Data received and receive status.

Receive one 8-16 bit frame with extended information.

EUSART_Tx

```
void EUSART_Tx (EUSART_TypeDef * eusart, uint8_t data)
```

Transmit one frame.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
uint8_t	N/A	data	Data to transmit.

Note

- Depending on the frame length configuration, 8 (least significant) bits from `data` are transmitted. If the frame length is 9, 8 bits are transmitted from `data`. See [EUSART_TxExt\(\)](#) for transmitting 9 bit frame with full control of all 9 bits.
- This function will stall if the 4 frame FIFO is full, until the buffer becomes available.

Transmit one frame.

EUSART_TxExt

```
void EUSART_TxExt (EUSART_TypeDef * eusart, uint16_t data)
```

Transmit one 8-9 bit frame with extended control.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
uint16_t	N/A	data	Data to transmit.

Note

- Possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.
- This function will stall if buffer is full until the buffer becomes available.

Transmit one 8-9 bit frame with extended control.

EUSART_BaudrateSet

```
void EUSART_BaudrateSet (EUSART_TypeDef * eusart, uint32_t refFreq, uint32_t baudrate)
```

Configure the baudrate (or as close as possible to a specified baudrate).

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
uint32_t	N/A	refFreq	The EUSART reference clock frequency in Hz that will be used. If set to 0, the currently configured peripheral clock is used.
uint32_t	N/A	baudrate	A baudrate to try to achieve.

Configure the baudrate (or as close as possible to a specified baudrate).

Note

- (1) When the oversampling is disabled, the peripheral clock frequency must be at least three times higher than the chosen baud rate.

EUSART_BaudrateGet

```
uint32_t EUSART_BaudrateGet (EUSART_TypeDef * eusart)
```

Get the current baudrate.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.

Returns

- The current baudrate.

Get the current baudrate.

EUSART_RxBlock

```
void EUSART_RxBlock (EUSART_TypeDef * eusart, EUSART_BlockRx_TypeDef enable)
```

Enable/Disable reception operation until the configured start frame is received.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
EUSART_BlockRx_TypeDef	N/A	enable	Select the receiver blocking status.

Enable/Disable reception operation until the configured start frame is received.

EUSART_TxTristateSet

```
void EUSART_TxTristateSet (EUSART_TypeDef * eusart, EUSART\_TristateTx\_TypeDef enable)
```

Enable/Disable the tristating of the transmitter output.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
EUSART_TristateTx_TypeDef	N/A	enable	Select the transmitter tristate status.

Enable/Disable the tristating of the transmitter output.

EUSART_PrTriggeEnable

```
void EUSART_PrTriggeEnable (EUSART_TypeDef * eusart, const EUSART\_PrTriggeInit\_TypeDef * init)
```

Initialize the automatic enabling of transmissions and/or reception using the PRS as a trigger.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
const EUSART_PrTriggeInit_TypeDef *	N/A	init	Pointer to the initialization structure.

Note

- Initialize EUSART with [EUSART_UartInitHf\(\)](#) or [EUSART_UartInitLf\(\)](#) before enabling the PRS trigger.

Initialize the automatic enabling of transmissions and/or reception using the PRS as a trigger.

EUSART_StatusGet

```
uint32_t EUSART_StatusGet (EUSART_TypeDef * eusart)
```

Get EUSART STATUS register.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.

Returns

- STATUS register value.

EUSART_IntClear

```
void EUSART_IntClear (EUSART_TypeDef * eusart, uint32_t flags)
```

Clear one or more pending EUSART interrupts.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
uint32_t	N/A	flags	Pending EUSART interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for EUSART module (EUSART_IF_nnn).

EUSART_IntDisable

```
void EUSART_IntDisable (EUSART_TypeDef * eusart, uint32_t flags)
```

Disable one or more EUSART interrupts.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
uint32_t	N/A	flags	Pending EUSART interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for EUSART module (EUSART_IF_nnn).

EUSART_IntEnable

```
void EUSART_IntEnable (EUSART_TypeDef * eusart, uint32_t flags)
```

Enable one or more EUSART interrupts.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
uint32_t	N/A	flags	Pending EUSART interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for EUSART module (EUSART_IF_nnn).

EUSART_IntGet

```
uint32_t EUSART_IntGet (EUSART_TypeDef * eusart)
```

Get pending EUSART interrupt flags.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.

Returns

- Pending EUSART interrupt sources.

EUSART_IntGetEnabled

```
uint32_t EUSART_IntGetEnabled (EUSART_TypeDef * eusart)
```

Get enabled and pending EUSART interrupt flags.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.

Useful for handling more interrupt sources in the same interrupt handler.

Returns

- Pending and enabled EUSART interrupt sources.

EUSART_IntSet

```
void EUSART_IntSet (EUSART_TypeDef * eusart, uint32_t flags)
```

Set one or more pending EUSART interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
EUSART_TypeDef *	N/A	eusart	Pointer to the EUSART peripheral register block.
uint32_t	N/A	flags	Interrupt source(s) to set to pending. Use a bitwise logic OR combination of valid interrupt flags for EUSART module (EUSART_IF_nnn).

Macro Definition Documentation

EUSART0_FIFO_DEPTH

```
#define EUSART0_FIFO_DEPTH
```

Value:

4

Define EUSART FIFO Depth information.

EUSART1_FIFO_DEPTH

```
#define EUSART1_FIFO_DEPTH
```

Value:

EUSART0_FIFO_DEPTH

EUSART2_FIFO_DEPTH

```
#define EUSART2_FIFO_DEPTH
```

Value:

```
EUSART0_FIFO_DEPTH
```

EUSART3_FIFO_DEPTH

```
#define EUSART3_FIFO_DEPTH
```

Value:

```
EUSART0_FIFO_DEPTH
```

EUSART4_FIFO_DEPTH

```
#define EUSART4_FIFO_DEPTH
```

Value:

```
EUSART0_FIFO_DEPTH
```

EUSART_FIFO_DEPTH

```
#define EUSART_FIFO_DEPTH
```

Value:

```
0 | ((n) == 0) ? EUSART0_FIFO_DEPTH \
0 | : ((n) == 1) ? EUSART1_FIFO_DEPTH \
0 | : ((n) == 2) ? EUSART2_FIFO_DEPTH \
0 | : ((n) == 3) ? EUSART3_FIFO_DEPTH \
0 | : ((n) == 4) ? EUSART4_FIFO_DEPTH \
0 | : 0x0UL)
```

EUSART_UART_INIT_DEFAULT_HF

```
#define EUSART_UART_INIT_DEFAULT_HF
```

Value:

```
0 | {
0 |     eusartEnable,          /* Enable RX/TX when initialization completed. */ \
0 |     0,                    /* Use current configured reference clock for configuring baud rate.*/ \
0 |     115200,               /* 115200 bits/s. */ \
0 |     eusartOVS16,          /* Oversampling x16. */ \
0 |     eusartDataBits8,      /* 8 data bits. */ \
0 |     eusartNoParity,       /* No parity. */ \
0 |     eusartStopbits1,      /* 1 stop bit. */ \
0 |     eusartMajorityVoteEnable, /* Majority vote enabled. */ \
0 |     eusartLoopbackDisable, /* Loop back disabled. */ \
0 |     NULL,                 /* Default advanced settings. */ \
0 | }
```

Default configuration for EUSART initialization structure in UART mode with high-frequency clock.

EUSART_DEFAULT_START_FRAME

```
#define EUSART_DEFAULT_START_FRAME
```

Value:

```
0x00u
```

Default start frame configuration, i.e. feature disabled.

EUSART_ADVANCED_INIT_DEFAULT

```
#define EUSART_ADVANCED_INIT_DEFAULT
```

Value:

```

0  {
0  eusartHwFlowControlNone,      /* Flow control disabled. */
0  false,                      /* Collision detection disabled. */
0  false,                      /* Data is sent with the least significant bit first. */
0  eusartInvertIODisable,      /* RX and TX signal active high. */
0  false,                      /* No DMA wake up on reception. */
0  false,                      /* No DMA wake up on transmission. */
0  false,                      /* Halt DMA on error disabled. */
0  EUSART_DEFAULT_START_FRAME, /* No start frame. */
0  false,                      /* TX auto tristate disabled. */
0  false,                      /* Do not use PRS signal as RX signal. */
0  (EUSART_PrsChannel_TypeDef) 0u, /* EUSART RX connected to prs channel 0. */
0  false,                      /* Multiprocessor mode disabled. */
0  false,                      /* Multiprocessor address bit : 0. */
0  eusartAutoTxDelayNone,      /* Frames are transmitted immediately */
0  eusartRxFiFoWatermark1Frame, /* RXFL status/IF set when RX FIFO has at least one frame in it */
0  eusartTxFiFoWatermark1Frame, /* TXFL status/IF set when TX FIFO has space for at least one more frame */
0  }

```

Default configuration for EUSART advanced initialization structure.

EUSART_UART_INIT_DEFAULT_LF

```
#define EUSART_UART_INIT_DEFAULT_LF
```

Value:

```

0  {
0  eusartEnable,              /* Enable RX/TX when initialization completed. */
0  0,                        /* Use current configured reference clock for configuring baud rate. */
0  9600,                     /* 9600 bits/s. */
0  eusart0VS0,               /* Oversampling disabled. */
0  eusartDataBits8,          /* 8 data bits. */
0  eusartNoParity,           /* No parity. */
0  eusartStopbits1,          /* 1 stop bit. */
0  eusartMajorityVoteDisable, /* Majority vote enabled. */
0  eusartLoopbackDisable,    /* Loop back disabled. */
0  NULL,                     /* Default advanced settings. */
0  }

```

Default configuration for EUSART initialization structure in UART mode with low-frequency clock.

EUSART_IRDA_INIT_DEFAULT_HF

```
#define EUSART_IRDA_INIT_DEFAULT_HF
```

Value:

```

0  {
0  EUSART_UART_INIT_DEFAULT_HF, /* Default high frequency configuration. */ \
0  false,                       /* Disable IrDA low frequency mode. */ \
0  eusartIrDARxFilterDisable,   /* RX Filter disabled. */ \
0  eusartIrDAPulseWidthOne,     /* Pulse width is set to 1/16. */ \
0  }

```

Default configuration for EUSART initialization structure in IrDA mode with high-frequency clock.

EUSART_IRDA_INIT_DEFAULT_LF

```
#define EUSART_IRDA_INIT_DEFAULT_LF
```

Value:

```

0  {
0  {
0  eusartEnableRx,             /* Enable RX when initialization completed (TX not allowed). */ \
0  0,                         /* Use current configured reference clock for configuring baud rate.*/ \
0  9600,                       /* 9600 bits/s. */ \
0  eusartOVS0,                 /* Oversampling disabled. */ \
0  eusartDataBits8,           /* 8 data bits. */ \
0  eusartNoParity,            /* No parity. */ \
0  eusartStopbits1,           /* 1 stop bit. */ \
0  eusartMajorityVoteDisable, /* Majority vote enabled. */ \
0  eusartLoopbackDisable,     /* Loop back disabled. */ \
0  NULL,                      /* Default advanced settings. */ \
0  },
0  true,                      /* Enable IrDA low frequency mode. */ \
0  eusartIrDARxFilterDisable, /* RX Filter disabled. */ \
0  eusartIrDAPulseWidthOne,   /* Pulse width is set to 1. */ \
0  }

```

Default configuration for EUSART initialization structure in IrDA mode with low-frequency clock.

EUSART_AdvancedInit_TypeDef

Advanced initialization structure.

Public Attributes

EUSART_HwFlowControl_TypeDef	hwFlowControl	Hardware flow control mode.
	bool	collisionDetectEnable Enable the collision Detection feature.
	bool	msbFirst If true, data will be send with most significant bit first.
EUSART_InvertIO_TypeDef	invertIO	Enable inversion of RX and/or TX signals.
	bool	dmaWakeUpOnRx Enable the automatic wake up from EM2 to EM1 for DMA RX operation.
	bool	dmaWakeUpOnTx Enable the automatic wake up from EM2 to EM1 for DMA TX operation.
	bool	dmaHaltOnError Enable DMA requests blocking while framing or parity errors.
	uint8_t	startFrame Start frame that will enable RX operation. 0x00 Disable this feature.
	bool	txAutoTristate Enable automatic tristating of transmistter output when there is nothing to transmit.
	bool	prsRxEnable Enable EUSART capability to use a PRS channel as an input data line for the receiver.
EUSART_PrsChannel_TypeDef	prsRxChannel	PRS Channel used to transmit data from PRS to the EUSART.
	bool	multiProcessorEnable Enable Multiprocessor mode. Address and data filtering using the 9th bit.
	bool	multiProcessorAddressBitHigh Multiprocessor address bit value. If true, 9th bit of address frame must bit 1, 0 otherwise.
EUSART_AutoTxDelay_TypeDef	autoTxDelay	Auto TX delay before new transfers. Frames sent back-to-back are not delayed.
EUSART_RxFifoWatermark_TypeDef	RxFifoWatermark	Interrupt and status level of the Receive FIFO.
EUSART_TxFifoWatermark_TypeDef	TxFifoWatermark	Interrupt and status level of the Transmit FIFO.

Public Attribute Documentation

hwFlowControl

```
EUSART_HwFlowControl_TypeDef EUSART_AdvancedInit_TypeDef::hwFlowControl
```

Hardware flow control mode.

collisionDetectEnable

```
bool EUSART_AdvancedInit_TypeDef::collisionDetectEnable
```

Enable the collision Detection feature.

Internal (setting loopbackEnable) or external loopback must be done to use this feature.

msbFirst

```
bool EUSART_AdvancedInit_TypeDef::msbFirst
```

If true, data will be send with most significant bit first.

invertIO

```
EUSART_InvertIO_TypeDef EUSART_AdvancedInit_TypeDef::invertIO
```

Enable inversion of RX and/or TX signals.

dmaWakeUpOnRx

```
bool EUSART_AdvancedInit_TypeDef::dmaWakeUpOnRx
```

Enable the automatic wake up from EM2 to EM1 for DMA RX operation.

dmaWakeUpOnTx

```
bool EUSART_AdvancedInit_TypeDef::dmaWakeUpOnTx
```

Enable the automatic wake up from EM2 to EM1 for DMA TX operation.

dmaHaltOnError

```
bool EUSART_AdvancedInit_TypeDef::dmaHaltOnError
```

Enable DMA requests blocking while framing or parity errors.

startFrame

```
uint8_t EUSART_AdvancedInit_TypeDef::startFrame
```

Start frame that will enable RX operation. 0x00 Disable this feature.

txAutoTristate

```
bool EUSART_AdvancedInit_TypeDef::txAutoTristate
```

Enable automatic tristating of transmitter output when there is nothing to transmit.

prsRxEnable

```
bool EUSART_AdvancedInit_TypeDef::prsRxEnable
```

Enable EUSART capability to use a PRS channel as an input data line for the receiver.

The configured RX GPIO signal won't be routed to the EUSART receiver.

prsRxChannel

```
EUSART_PrsChannel_TypeDef EUSART_AdvancedInit_TypeDef::prsRxChannel
```

PRS Channel used to transmit data from PRS to the EUSART.

multiProcessorEnable

```
bool EUSART_AdvancedInit_TypeDef::multiProcessorEnable
```

Enable Multiprocessor mode. Address and data filtering using the 9th bit.

multiProcessorAddressBitHigh

```
bool EUSART_AdvancedInit_TypeDef::multiProcessorAddressBitHigh
```

Multiprocessor address bit value. If true, 9th bit of address frame must bit 1, 0 otherwise.

autoTxDelay

```
EUSART_AutoTxDelay_TypeDef EUSART_AdvancedInit_TypeDef::autoTxDelay
```

Auto TX delay before new transfers. Frames sent back-to-back are not delayed.

RxFifoWatermark

EUSART_RxFifoWatermark_TypeDef EUSART_AdvancedInit_TypeDef::RxFifoWatermark

Interrupt and status level of the Receive FIFO.

TxFifoWatermark

EUSART_TxFifoWatermark_TypeDef EUSART_AdvancedInit_TypeDef::TxFifoWatermark

Interrupt and status level of the Transmit FIFO.

EUSART_UartInit_TypeDef

Initialization structure.

Public Attributes

EUSART_Enable_TypeDef	enable Specifies whether TX and/or RX will be enabled when initialization completes.
uint32_t	refFreq EUSART reference clock assumed when configuring baud rate setup.
uint32_t	baudrate Desired baud rate.
EUSART_OVS_TypeDef	oversampling Oversampling used.
EUSART_Databits_TypeDef	databits Number of data bits in frame.
EUSART_Parity_TypeDef	parity Parity mode to use.
EUSART_Stopbits_TypeDef	stopbits Number of stop bits to use.
EUSART_MajorityVote_TypeDef	majorityVote Majority Vote can be disabled for 16x, 8x and 6x oversampling modes.
EUSART_LoopbackEnable_TypeDef	loopbackEnable Enable Loop Back configuration.
EUSART_AdvancedInit_TypeDef*	advancedSettings Advanced initialization structure pointer. It can be NULL.

Public Attribute Documentation

enable

```
EUSART_Enable_TypeDef EUSART_UartInit_TypeDef::enable
```

Specifies whether TX and/or RX will be enabled when initialization completes.

refFreq

```
uint32_t EUSART_UartInit_TypeDef::refFreq
```

EUSART reference clock assumed when configuring baud rate setup.
Set to 0 if using currently configured reference clock.

baudrate

```
uint32_t EUSART_UartInit_TypeDef::baudrate
```

Desired baud rate.

If set to 0, Auto Baud feature is enabled and the EUSART will wait for (0x55) frame to detect the Baudrate.

oversampling

```
EUSART_OVS_TypeDef EUSART_UartInit_TypeDef::oversampling
```

Oversampling used.

databits

```
EUSART_Databits_TypeDef EUSART_UartInit_TypeDef::databits
```

Number of data bits in frame.

parity

```
EUSART_Parity_TypeDef EUSART_UartInit_TypeDef::parity
```

Parity mode to use.

stopbits

```
EUSART_Stopbits_TypeDef EUSART_UartInit_TypeDef::stopbits
```

Number of stop bits to use.

majorityVote

```
EUSART_MajorityVote_TypeDef EUSART_UartInit_TypeDef::majorityVote
```

Majority Vote can be disabled for 16x, 8x and 6x oversampling modes.

loopbackEnable

```
EUSART_LoopbackEnable_TypeDef EUSART_UartInit_TypeDef::loopbackEnable
```

Enable Loop Back configuration.

advancedSettings

```
EUSART_AdvancedInit_TypeDef* EUSART_UartInit_TypeDef::advancedSettings
```

Advanced initialization structure pointer. It can be NULL.

EUSART_IrDAInit_TypeDef

IrDA Initialization structure.

Public Attributes

EUSART_UartInit_TypeDef	init General EUSART initialization structure.
bool	irDALowFrequencyEnable Enable the IrDA low frequency mode. Only RX operation are enabled.
EUSART_IrDARxFilterEnable_TypeDef	irDARxFilterEnable Set to enable filter on IrDA demodulator.
EUSART_IrDAPulseWidth_TypeDef	irDAPulseWidth Configure the pulse width generated by the IrDA modulator as a fraction of the configured EUSART bit period.

Public Attribute Documentation

init

```
EUSART_UartInit_TypeDef EUSART_IrDAInit_TypeDef::init
```

General EUSART initialization structure.

irDALowFrequencyEnable

```
bool EUSART_IrDAInit_TypeDef::irDALowFrequencyEnable
```

Enable the IrDA low frequency mode. Only RX operation are enabled.

irDARxFilterEnable

```
EUSART_IrDARxFilterEnable_TypeDef EUSART_IrDAInit_TypeDef::irDARxFilterEnable
```

Set to enable filter on IrDA demodulator.

irDAPulseWidth

```
EUSART_IrDAPulseWidth_TypeDef EUSART_IrDAInit_TypeDef::irDAPulseWidth
```

Configure the pulse width generated by the IrDA modulator as a fraction of the configured EUSART bit period.

EUSART_PrTriggerInit_TypeDef

PRS Trigger initialization structure.

Public Attributes

EUSART_PrTriggerEnable_TypeDef

prs_trigger_enable

PRS to EUSART trigger mode.

EUSART_PrChannel_TypeDef

prs_trigger_channel

PRS channel to be used to trigger auto transmission.

Public Attribute Documentation

prs_trigger_enable

EUSART_PrTriggerEnable_TypeDef EUSART_PrTriggerInit_TypeDef::prs_trigger_enable

PRS to EUSART trigger mode.

prs_trigger_channel

EUSART_PrChannel_TypeDef EUSART_PrTriggerInit_TypeDef::prs_trigger_channel

PRS channel to be used to trigger auto transmission.

EUSART_DaliInit_TypeDef

DALI Initialization structure.

Public Attributes

EUSART_UartInit_TypeDef	init General EUSART initialization structure.
bool	daliLowFrequencyEnable Enable the DALI low frequency mode.

Public Attribute Documentation

init

EUSART_UartInit_TypeDef EUSART_DaliInit_TypeDef::init

General EUSART initialization structure.

daliLowFrequencyEnable

bool EUSART_DaliInit_TypeDef::daliLowFrequencyEnable

Enable the DALI low frequency mode.

Emlib

Emlib

Modules

[IADC - Incremental ADC](#)

IADC - Incremental ADC

IADC - Incremental ADC

Incremental Analog to Digital Converter (IADC) Peripheral API.

This module contains functions to control the IADC peripheral of Silicon Labs 32-bit MCUs and SoCs. The IADC is used to convert analog signals into a digital representation.

Modules

[IADC_Init_t](#)

[IADC_Config_t](#)

[IADC_AllConfigs_t](#)

[IADC_InitScan_t](#)

[IADC_InitSingle_t](#)

[IADC_SingleInput_t](#)

[IADC_ScanTableEntry_t](#)

[IADC_ScanTable_t](#)

[IADC_Result_t](#)

Enumerations

```
enum    IADC_Warmup_t {
    iadcWarmupNormal = _IADC_CTRL_WARMUPMODE_NORMAL
    iadcWarmupKeepInStandby = _IADC_CTRL_WARMUPMODE_KEEPISTANDBY
    iadcWarmupKeepWarm = _IADC_CTRL_WARMUPMODE_KEEPPWARM
}
Warm-up mode.

enum    IADC_Alignment_t {
    iadcAlignRight12 = _IADC_SCANFIFOCFG_ALIGNMENT_RIGHT12
    iadcAlignLeft12 = _IADC_SCANFIFOCFG_ALIGNMENT_LEFT12
    iadcAlignRight16 = _IADC_SCANFIFOCFG_ALIGNMENT_RIGHT16
    iadcAlignLeft16 = _IADC_SCANFIFOCFG_ALIGNMENT_LEFT16
    iadcAlignRight20 = _IADC_SCANFIFOCFG_ALIGNMENT_RIGHT20
    iadcAlignLeft20 = _IADC_SCANFIFOCFG_ALIGNMENT_LEFT20
}
IADC result alignment.
```

```
enum IADC_NegInput_t {
    iadcNegInputGnd = (_IADC_SCAN_PORTNEG_GND << (_IADC_SCAN_PORTNEG_SHIFT -
_IADC_SCAN_PINNEG_SHIFT)) | 1
    iadcNegInputGndaux = (_IADC_SCAN_PORTNEG_GND << (_IADC_SCAN_PORTNEG_SHIFT -
_IADC_SCAN_PINNEG_SHIFT))
    iadcNegInputPortAPin0 = (_IADC_SCAN_PORTNEG_PORTA << (_IADC_SCAN_PORTNEG_SHIFT -
_IADC_SCAN_PINNEG_SHIFT))
    iadcNegInputPortAPin1
    iadcNegInputPortAPin2
    iadcNegInputPortAPin3
    iadcNegInputPortAPin4
    iadcNegInputPortAPin5
    iadcNegInputPortAPin6
    iadcNegInputPortAPin7
    iadcNegInputPortAPin8
    iadcNegInputPortAPin9
    iadcNegInputPortAPin10
    iadcNegInputPortAPin11
    iadcNegInputPortAPin12
    iadcNegInputPortAPin13
    iadcNegInputPortAPin14
    iadcNegInputPortAPin15
    iadcNegInputPortBPin0
    iadcNegInputPortBPin1
    iadcNegInputPortBPin2
    iadcNegInputPortBPin3
    iadcNegInputPortBPin4
    iadcNegInputPortBPin5
    iadcNegInputPortBPin6
    iadcNegInputPortBPin7
    iadcNegInputPortBPin8
    iadcNegInputPortBPin9
    iadcNegInputPortBPin10
    iadcNegInputPortBPin11
    iadcNegInputPortBPin12
    iadcNegInputPortBPin13
    iadcNegInputPortBPin14
    iadcNegInputPortBPin15
    iadcNegInputPortCPin0
    iadcNegInputPortCPin1
    iadcNegInputPortCPin2
    iadcNegInputPortCPin3
    iadcNegInputPortCPin4
    iadcNegInputPortCPin5
    iadcNegInputPortCPin6
    iadcNegInputPortCPin7
    iadcNegInputPortCPin8
    iadcNegInputPortCPin9
    iadcNegInputPortCPin10
    iadcNegInputPortCPin11
    iadcNegInputPortCPin12
    iadcNegInputPortCPin13
    iadcNegInputPortCPin14
    iadcNegInputPortCPin15
    iadcNegInputPortDPin0
    iadcNegInputPortDPin1
    iadcNegInputPortDPin2
    iadcNegInputPortDPin3
    iadcNegInputPortDPin4
    iadcNegInputPortDPin5
    iadcNegInputPortDPin6
    iadcNegInputPortDPin7
    iadcNegInputPortDPin8
    iadcNegInputPortDPin9
    iadcNegInputPortDPin10
    iadcNegInputPortDPin11
    iadcNegInputPortDPin12
    iadcNegInputPortDPin13
    iadcNegInputPortDPin14
    iadcNegInputPortDPin15
}
IADC negative input selection.
```

```

enum IADC_PosInput_t {
    iadcPosInputGnd = (_IADC_SCAN_PORTPOS_GND << (_IADC_SCAN_PORTPOS_SHIFT -
_IADC_SCAN_PINPOS_SHIFT))
    iadcPosInputAvdd = (_IADC_SCAN_PORTPOS_SUPPLY << (_IADC_SCAN_PORTPOS_SHIFT -
_IADC_SCAN_PINPOS_SHIFT)) | 0
    iadcPosInputVddio = (_IADC_SCAN_PORTPOS_SUPPLY << (_IADC_SCAN_PORTPOS_SHIFT -
_IADC_SCAN_PINPOS_SHIFT)) | 1
    iadcPosInputVss = (_IADC_SCAN_PORTPOS_SUPPLY << (_IADC_SCAN_PORTPOS_SHIFT -
_IADC_SCAN_PINPOS_SHIFT)) | 2
    iadcPosInputVssaux = (_IADC_SCAN_PORTPOS_SUPPLY << (_IADC_SCAN_PORTPOS_SHIFT -
_IADC_SCAN_PINPOS_SHIFT)) | 3
    iadcPosInputDvdd = (_IADC_SCAN_PORTPOS_SUPPLY << (_IADC_SCAN_PORTPOS_SHIFT -
_IADC_SCAN_PINPOS_SHIFT)) | 4
    iadcPosInputDecouple = (_IADC_SCAN_PORTPOS_SUPPLY << (_IADC_SCAN_PORTPOS_SHIFT -
_IADC_SCAN_PINPOS_SHIFT)) | 7
    iadcPosInputPortAPin0 = (_IADC_SCAN_PORTPOS_PORTA << (_IADC_SCAN_PORTPOS_SHIFT -
_IADC_SCAN_PINPOS_SHIFT))
    iadcPosInputPortAPin1
    iadcPosInputPortAPin2
    iadcPosInputPortAPin3
    iadcPosInputPortAPin4
    iadcPosInputPortAPin5
    iadcPosInputPortAPin6
    iadcPosInputPortAPin7
    iadcPosInputPortAPin8
    iadcPosInputPortAPin9
    iadcPosInputPortAPin10
    iadcPosInputPortAPin11
    iadcPosInputPortAPin12
    iadcPosInputPortAPin13
    iadcPosInputPortAPin14
    iadcPosInputPortAPin15
    iadcPosInputPortBPin0
    iadcPosInputPortBPin1
    iadcPosInputPortBPin2
    iadcPosInputPortBPin3
    iadcPosInputPortBPin4
    iadcPosInputPortBPin5
    iadcPosInputPortBPin6
    iadcPosInputPortBPin7
    iadcPosInputPortBPin8
    iadcPosInputPortBPin9
    iadcPosInputPortBPin10
    iadcPosInputPortBPin11
    iadcPosInputPortBPin12
    iadcPosInputPortBPin13
    iadcPosInputPortBPin14
    iadcPosInputPortBPin15
    iadcPosInputPortCPin0
    iadcPosInputPortCPin1
    iadcPosInputPortCPin2
    iadcPosInputPortCPin3
    iadcPosInputPortCPin4
    iadcPosInputPortCPin5
    iadcPosInputPortCPin6
    iadcPosInputPortCPin7
    iadcPosInputPortCPin8
    iadcPosInputPortCPin9
    iadcPosInputPortCPin10
    iadcPosInputPortCPin11
    iadcPosInputPortCPin12
    iadcPosInputPortCPin13
    iadcPosInputPortCPin14
    iadcPosInputPortCPin15
    iadcPosInputPortDPin0
    iadcPosInputPortDPin1
    iadcPosInputPortDPin2
    iadcPosInputPortDPin3
    iadcPosInputPortDPin4
    iadcPosInputPortDPin5
    iadcPosInputPortDPin6
    iadcPosInputPortDPin7
    iadcPosInputPortDPin8
    iadcPosInputPortDPin9
    iadcPosInputPortDPin10
    iadcPosInputPortDPin11
    iadcPosInputPortDPin12
    iadcPosInputPortDPin13
    iadcPosInputPortDPin14
    iadcPosInputPortDPin15
}

```

IADC positive port selection.

```

enum IADC_Cmd_t {
    iadcCmdStartSingle = IADC_CMD_SINGLESTART
    iadcCmdStopSingle = IADC_CMD_SINGLESTOP
    iadcCmdStartScan = IADC_CMD_SCANSTART
    iadcCmdStopScan = IADC_CMD_SCANSTOP
    iadcCmdEnableTimer = IADC_CMD_TIMEREN
    iadcCmdDisableTimer = IADC_CMD_TIMERDIS
}
IADC Commands.

enum IADC_CfgAdcMode_t {
    iadcCfgModeNormal = _IADC_CFG_ADCMODE_NORMAL
}
IADC Configuration.

enum IADC_CfgOsrHighSpeed_t {
    iadcCfgOsrHighSpeed2x = _IADC_CFG_OSRHS_HISPD2
    iadcCfgOsrHighSpeed4x = _IADC_CFG_OSRHS_HISPD4
    iadcCfgOsrHighSpeed8x = _IADC_CFG_OSRHS_HISPD8
    iadcCfgOsrHighSpeed16x = _IADC_CFG_OSRHS_HISPD16
    iadcCfgOsrHighSpeed32x = _IADC_CFG_OSRHS_HISPD32
    iadcCfgOsrHighSpeed64x = _IADC_CFG_OSRHS_HISPD64
}
IADC Over sampling rate for high speed.

enum IADC_CfgAnalogGain_t {
    iadcCfgAnalogGain0P5x = _IADC_CFG_ANALOGGAIN_ANAGAIN0P5
    iadcCfgAnalogGain1x = _IADC_CFG_ANALOGGAIN_ANAGAIN1
    iadcCfgAnalogGain2x = _IADC_CFG_ANALOGGAIN_ANAGAIN2
    iadcCfgAnalogGain3x = _IADC_CFG_ANALOGGAIN_ANAGAIN3
    iadcCfgAnalogGain4x = _IADC_CFG_ANALOGGAIN_ANAGAIN4
}
IADC Analog Gain.

enum IADC_CfgReference_t {
    iadcCfgReferenceInt1V2 = _IADC_CFG_REFSEL_VBGR
    iadcCfgReferenceExt1V25 = _IADC_CFG_REFSEL_VREF
    iadcCfgReferenceVddx = _IADC_CFG_REFSEL_VDDX
    iadcCfgReferenceVddX0P8Buf = _IADC_CFG_REFSEL_VDDX0P8BUF
}
IADC Reference.

enum IADC_CfgTwosComp_t {
    iadcCfgTwosCompAuto = _IADC_CFG_TWOSCOMPL_AUTO
    iadcCfgTwosCompUnipolar = _IADC_CFG_TWOSCOMPL_FORCEUNIPOLAR
    iadcCfgTwosCompBipolar = _IADC_CFG_TWOSCOMPL_FORCEBIPOLAR
}
IADC Two's complement results.

enum IADC_TriggerSel_t {
    iadcTriggerSelImmediate = _IADC_TRIGGER_SCANTRIGSEL_IMMEDIATE
    iadcTriggerSelTimer = _IADC_TRIGGER_SCANTRIGSEL_TIMER
    iadcTriggerSelPrs0SameClk = _IADC_TRIGGER_SCANTRIGSEL_PRSCLKGRP
    iadcTriggerSelPrs0PosEdge = _IADC_TRIGGER_SCANTRIGSEL_PRSPOS
    iadcTriggerSelPrs0NegEdge = _IADC_TRIGGER_SCANTRIGSEL_PRSNEG
}
IADC trigger action.

enum IADC_TriggerAction_t {
    iadcTriggerActionOnce = _IADC_TRIGGER_SCANTRIGACTION_ONCE
    iadcTriggerActionContinuous = _IADC_TRIGGER_SCANTRIGACTION_CONTINUOUS
}
IADC trigger action.

```

```
enum IADC_FifoCfgDvl_t {
    iadcFifoCfgDvl1 = _IADC_SCANFIFOCFG_DVL_VALID1
    iadcFifoCfgDvl2 = _IADC_SCANFIFOCFG_DVL_VALID2
    iadcFifoCfgDvl3 = _IADC_SCANFIFOCFG_DVL_VALID3
    iadcFifoCfgDvl4 = _IADC_SCANFIFOCFG_DVL_VALID4
}
IADC data valid level before requesting DMA transfer.
```

```
enum IADC_DigitalAveraging_t {
    iadcDigitalAverage1 = _IADC_CFG_DIGAVG_AVG1
    iadcDigitalAverage2 = _IADC_CFG_DIGAVG_AVG2
    iadcDigitalAverage4 = _IADC_CFG_DIGAVG_AVG4
    iadcDigitalAverage8 = _IADC_CFG_DIGAVG_AVG8
    iadcDigitalAverage16 = _IADC_CFG_DIGAVG_AVG16
}
IADC digital averaging function.
```

Functions

```
void IADC_init(IADC_TypeDef *iadc, const IADC_Init_t *init, const IADC_AllConfigs_t *allConfigs)
Initialize IADC.
```

```
void IADC_initScan(IADC_TypeDef *iadc, const IADC_InitScan_t *init, const IADC_ScanTable_t *scanTable)
Initialize IADC scan sequence.
```

```
void IADC_initSingle(IADC_TypeDef *iadc, const IADC_InitSingle_t *init, const IADC_SingleInput_t *input)
Initialize single IADC conversion.
```

```
void IADC_updateSingleInput(IADC_TypeDef *iadc, const IADC_SingleInput_t *input)
Update IADC single input selection.
```

```
void IADC_setScanMask(IADC_TypeDef *iadc, uint32_t mask)
Set mask of IADC scan table entries to include in scan.
```

```
void IADC_updateScanEntry(IADC_TypeDef *iadc, uint8_t id, IADC_ScanTableEntry_t *entry)
Add/update entry in scan table.
```

```
void IADC_reset(IADC_TypeDef *iadc)
Reset IADC to same state as after a HW reset.
```

```
uint8_t IADC_calcTimebase(IADC_TypeDef *iadc, uint32_t srcClkFreq)
Calculate timebase value in order to get a timebase providing at least 1us.
```

```
uint8_t IADC_calcSrcClkPrescale(IADC_TypeDef *iadc, uint32_t srcClkFreq, uint32_t cmuClkFreq)
Calculate prescaler for CLK_SRC_ADC high speed clock.
```

```
uint32_t IADC_calcAdcClkPrescale(IADC_TypeDef *iadc, uint32_t adcClkFreq, uint32_t cmuClkFreq,
IADC_CfgAdcMode_t adcMode, uint8_t srcClkPrescaler)
Calculate prescaler for ADC_CLK clock.
```

```
IADC_Result_t IADC_pullSingleFifoResult(IADC_TypeDef *iadc)
Pull result from single data FIFO.
```

```
IADC_Result_t IADC_readSingleResult(IADC_TypeDef *iadc)
Read most recent single conversion result.
```

```
IADC_Result_t IADC_pullScanFifoResult(IADC_TypeDef *iadc)
Pull result from scan data FIFO.
```

```
IADC_Result_t IADC_readScanResult(IADC_TypeDef *iadc)
Read most recent scan conversion result.
```

```
uint32_t IADC_getReferenceVoltage(IADC_CfgReference_t reference)
Get reference voltage selection.
```

uint32_t	IADC_pullSingleFifoData (IADC_TypeDef *iadc) Pull data from single data FIFO.
uint32_t	IADC_readSingleData (IADC_TypeDef *iadc) Read most recent single conversion data.
uint32_t	IADC_pullScanFifoData (IADC_TypeDef *iadc) Pull data from scan data FIFO.
uint32_t	IADC_readScanData (IADC_TypeDef *iadc) Read most recent scan conversion data.
void	IADC_clearInt (IADC_TypeDef *iadc, uint32_t flags) Clear one or more pending IADC interrupts.
void	IADC_disableInt (IADC_TypeDef *iadc, uint32_t flags) Disable one or more IADC interrupts.
void	IADC_enableInt (IADC_TypeDef *iadc, uint32_t flags) Enable one or more IADC interrupts.
uint32_t	IADC_getInt (IADC_TypeDef *iadc) Get pending IADC interrupt flags.
uint32_t	IADC_getEnabledInt (IADC_TypeDef *iadc) Get enabled and pending IADC interrupt flags.
void	IADC_setInt (IADC_TypeDef *iadc, uint32_t flags) Set one or more pending IADC interrupts from SW.
void	IADC_command (IADC_TypeDef *iadc, IADC_Cmd_t cmd) Start/stop scan sequence, single conversion and/or timer.
uint32_t	IADC_getScanMask (IADC_TypeDef *iadc) Get the scan mask currently used in the IADC.
uint32_t	IADC_getStatus (IADC_TypeDef *iadc) Get status bits of IADC.
uint8_t	IADC_getSingleFifoCnt (IADC_TypeDef *iadc) Get the number of elements in the IADC single FIFO.
uint8_t	IADC_getScanFifoCnt (IADC_TypeDef *iadc) Get the number of elements in the IADC scan FIFO.
IADC_NegInput_t	IADC_portPinToNegInput (GPIO_Port_TypeDef port, uint8_t pin) Convert the GPIO port/pin to IADC negative input selection.
IADC_PosInput_t	IADC_portPinToPosInput (GPIO_Port_TypeDef port, uint8_t pin) Convert the GPIO port/pin to IADC positive input selection.

Macros

#define	IADC_INIT_DEFAULT undefined Default config for IADC init structure.
#define	IADC_CONFIG_DEFAULT undefined Default IADC config structure.
#define	IADC_ALLCONFIGS_DEFAULT undefined Default IADC structure for all configs.
#define	IADC_INITSCAN_DEFAULT undefined Default config for IADC scan init structure.

```
#define IADC_INITSINGLE_DEFAULT undefined
Default config for IADC single init structure.

#define IADC_SINGLEINPUT_DEFAULT undefined
Default config for IADC single input structure.

#define IADC_SCANTABLEENTRY_DEFAULT undefined
Default config for IADC scan table entry structure.

#define IADC_SCANTABLE_DEFAULT undefined
Default IADC structure for scan table.
```

Enumeration Documentation

IADC_Warmup_t

IADC_Warmup_t

Warm-up mode.

Enumerator	
iadcWarmupNormal	IADC shutdown after each conversion.
iadcWarmupKeepInStandby	ADC is kept in standby mode between conversion.
iadcWarmupKeepWarm	ADC and reference selected for scan mode kept warmup, allowing continuous conversion.

IADC_Alignment_t

IADC_Alignment_t

IADC result alignment.

Enumerator	
iadcAlignRight12	IADC results 12-bit right aligned
iadcAlignLeft12	IADC results 12-bit left aligned
iadcAlignRight16	IADC results 16-bit right aligned
iadcAlignLeft16	IADC results 16-bit left aligned
iadcAlignRight20	IADC results 20-bit right aligned
iadcAlignLeft20	IADC results 20-bit left aligned

IADC_NegInput_t

IADC_NegInput_t

IADC negative input selection.

Enumerator	
iadcNegInputGnd	Ground
iadcNegInputGndaux	Ground using even mux.
iadcNegInputPortAPin0	GPIO port A pin 0.
iadcNegInputPortAPin1	GPIO port A pin 1.

iadcNegInputPortAPin2	GPIO port A pin 2.
iadcNegInputPortAPin3	GPIO port A pin 3.
iadcNegInputPortAPin4	GPIO port A pin 4.
iadcNegInputPortAPin5	GPIO port A pin 5.
iadcNegInputPortAPin6	GPIO port A pin 6.
iadcNegInputPortAPin7	GPIO port A pin 7.
iadcNegInputPortAPin8	GPIO port A pin 8.
iadcNegInputPortAPin9	GPIO port A pin 9.
iadcNegInputPortAPin10	GPIO port A pin 10.
iadcNegInputPortAPin11	GPIO port A pin 11.
iadcNegInputPortAPin12	GPIO port A pin 12.
iadcNegInputPortAPin13	GPIO port A pin 13.
iadcNegInputPortAPin14	GPIO port A pin 14.
iadcNegInputPortAPin15	GPIO port A pin 15.
iadcNegInputPortBPin0	GPIO port B pin 0.
iadcNegInputPortBPin1	GPIO port B pin 1.
iadcNegInputPortBPin2	GPIO port B pin 2.
iadcNegInputPortBPin3	GPIO port B pin 3.
iadcNegInputPortBPin4	GPIO port B pin 4.
iadcNegInputPortBPin5	GPIO port B pin 5.
iadcNegInputPortBPin6	GPIO port B pin 6.
iadcNegInputPortBPin7	GPIO port B pin 7.
iadcNegInputPortBPin8	GPIO port B pin 8.
iadcNegInputPortBPin9	GPIO port B pin 9.
iadcNegInputPortBPin10	GPIO port B pin 10.
iadcNegInputPortBPin11	GPIO port B pin 11.
iadcNegInputPortBPin12	GPIO port B pin 12.
iadcNegInputPortBPin13	GPIO port B pin 13.
iadcNegInputPortBPin14	GPIO port B pin 14.
iadcNegInputPortBPin15	GPIO port B pin 15.
iadcNegInputPortCPin0	GPIO port C pin 0.
iadcNegInputPortCPin1	GPIO port C pin 1.
iadcNegInputPortCPin2	GPIO port C pin 2.
iadcNegInputPortCPin3	GPIO port C pin 3.
iadcNegInputPortCPin4	GPIO port C pin 4.
iadcNegInputPortCPin5	GPIO port C pin 5.
iadcNegInputPortCPin6	GPIO port C pin 6.
iadcNegInputPortCPin7	GPIO port C pin 7.
iadcNegInputPortCPin8	GPIO port C pin 8.
iadcNegInputPortCPin9	GPIO port C pin 9.
iadcNegInputPortCPin10	GPIO port C pin 10.
iadcNegInputPortCPin11	GPIO port C pin 11.
iadcNegInputPortCPin12	GPIO port C pin 12.

iadcNegInputPortCPin13	GPIO port C pin 13.
iadcNegInputPortCPin14	GPIO port C pin 14.
iadcNegInputPortCPin15	GPIO port C pin 15.
iadcNegInputPortDPin0	GPIO port D pin 0.
iadcNegInputPortDPin1	GPIO port D pin 1.
iadcNegInputPortDPin2	GPIO port D pin 2.
iadcNegInputPortDPin3	GPIO port D pin 3.
iadcNegInputPortDPin4	GPIO port D pin 4.
iadcNegInputPortDPin5	GPIO port D pin 5.
iadcNegInputPortDPin6	GPIO port D pin 6.
iadcNegInputPortDPin7	GPIO port D pin 7.
iadcNegInputPortDPin8	GPIO port D pin 8.
iadcNegInputPortDPin9	GPIO port D pin 9.
iadcNegInputPortDPin10	GPIO port D pin 10.
iadcNegInputPortDPin11	GPIO port D pin 11.
iadcNegInputPortDPin12	GPIO port D pin 12.
iadcNegInputPortDPin13	GPIO port D pin 13.
iadcNegInputPortDPin14	GPIO port D pin 14.
iadcNegInputPortDPin15	GPIO port D pin 15.

IADC_PosInput_t

IADC_PosInput_t

IADC positive port selection.

	Enumerator
iadcPosInputGnd	Ground
iadcPosInputAvdd	Avdd / 4
iadcPosInputVddio	Vddio / 4
iadcPosInputVss	Vss
iadcPosInputVssaux	Vss
iadcPosInputDvdd	Dvdd / 4
iadcPosInputDecouple	Decouple
iadcPosInputPortAPin0	GPIO port A pin 0.
iadcPosInputPortAPin1	GPIO port A pin 1.
iadcPosInputPortAPin2	GPIO port A pin 2.
iadcPosInputPortAPin3	GPIO port A pin 3.
iadcPosInputPortAPin4	GPIO port A pin 4.
iadcPosInputPortAPin5	GPIO port A pin 5.
iadcPosInputPortAPin6	GPIO port A pin 6.
iadcPosInputPortAPin7	GPIO port A pin 7.
iadcPosInputPortAPin8	GPIO port A pin 8.

iadcPosInputPortAPin9	GPIO port A pin 9.
iadcPosInputPortAPin10	GPIO port A pin 10.
iadcPosInputPortAPin11	GPIO port A pin 11.
iadcPosInputPortAPin12	GPIO port A pin 12.
iadcPosInputPortAPin13	GPIO port A pin 13.
iadcPosInputPortAPin14	GPIO port A pin 14.
iadcPosInputPortAPin15	GPIO port A pin 15.
iadcPosInputPortBPin0	GPIO port B pin 0.
iadcPosInputPortBPin1	GPIO port B pin 1.
iadcPosInputPortBPin2	GPIO port B pin 2.
iadcPosInputPortBPin3	GPIO port B pin 3.
iadcPosInputPortBPin4	GPIO port B pin 4.
iadcPosInputPortBPin5	GPIO port B pin 5.
iadcPosInputPortBPin6	GPIO port B pin 6.
iadcPosInputPortBPin7	GPIO port B pin 7.
iadcPosInputPortBPin8	GPIO port B pin 8.
iadcPosInputPortBPin9	GPIO port B pin 9.
iadcPosInputPortBPin10	GPIO port B pin 10.
iadcPosInputPortBPin11	GPIO port B pin 11.
iadcPosInputPortBPin12	GPIO port B pin 12.
iadcPosInputPortBPin13	GPIO port B pin 13.
iadcPosInputPortBPin14	GPIO port B pin 14.
iadcPosInputPortBPin15	GPIO port B pin 15.
iadcPosInputPortCPin0	GPIO port C pin 0.
iadcPosInputPortCPin1	GPIO port C pin 1.
iadcPosInputPortCPin2	GPIO port C pin 2.
iadcPosInputPortCPin3	GPIO port C pin 3.
iadcPosInputPortCPin4	GPIO port C pin 4.
iadcPosInputPortCPin5	GPIO port C pin 5.
iadcPosInputPortCPin6	GPIO port C pin 6.
iadcPosInputPortCPin7	GPIO port C pin 7.
iadcPosInputPortCPin8	GPIO port C pin 8.
iadcPosInputPortCPin9	GPIO port C pin 9.
iadcPosInputPortCPin10	GPIO port C pin 10.
iadcPosInputPortCPin11	GPIO port C pin 11.
iadcPosInputPortCPin12	GPIO port C pin 12.
iadcPosInputPortCPin13	GPIO port C pin 13.
iadcPosInputPortCPin14	GPIO port C pin 14.
iadcPosInputPortCPin15	GPIO port C pin 15.
iadcPosInputPortDPin0	GPIO port D pin 0.
iadcPosInputPortDPin1	GPIO port D pin 1.
iadcPosInputPortDPin2	GPIO port D pin 2.
iadcPosInputPortDPin3	GPIO port D pin 3.

iadcPosInputPortDPin4	GPIO port D pin 4.
iadcPosInputPortDPin5	GPIO port D pin 5.
iadcPosInputPortDPin6	GPIO port D pin 6.
iadcPosInputPortDPin7	GPIO port D pin 7.
iadcPosInputPortDPin8	GPIO port D pin 8.
iadcPosInputPortDPin9	GPIO port D pin 9.
iadcPosInputPortDPin10	GPIO port D pin 10.
iadcPosInputPortDPin11	GPIO port D pin 11.
iadcPosInputPortDPin12	GPIO port D pin 12.
iadcPosInputPortDPin13	GPIO port D pin 13.
iadcPosInputPortDPin14	GPIO port D pin 14.
iadcPosInputPortDPin15	GPIO port D pin 15.

IADC_Cmd_t

IADC_Cmd_t

IADC Commands.

Enumerator	
iadcCmdStartSingle	Start single queue
iadcCmdStopSingle	Stop single queue
iadcCmdStartScan	Start scan queue
iadcCmdStopScan	Stop scan queue
iadcCmdEnableTimer	Enable Timer
iadcCmdDisableTimer	Disable Timer

IADC_CfgAdcMode_t

IADC_CfgAdcMode_t

IADC Configuration.

Enumerator	
iadcCfgModeNormal	Normal mode

IADC_CfgOsrHighSpeed_t

IADC_CfgOsrHighSpeed_t

IADC Over sampling rate for high speed.

Enumerator	
iadcCfgOsrHighSpeed2x	High speed oversampling of 2x.
iadcCfgOsrHighSpeed4x	High speed oversampling of 4x.
iadcCfgOsrHighSpeed8x	High speed oversampling of 8x.

iadcCfgOsrHighSpeed16x	High speed oversampling of 16x.
iadcCfgOsrHighSpeed32x	High speed oversampling of 32x.
iadcCfgOsrHighSpeed64x	High speed oversampling of 64x.

IADC_CfgAnalogGain_t

IADC_CfgAnalogGain_t

IADC Analog Gain.

Enumerator	
iadcCfgAnalogGain0P5x	Analog gain of 0.5x.
iadcCfgAnalogGain1x	Analog gain of 1x.
iadcCfgAnalogGain2x	Analog gain of 2x.
iadcCfgAnalogGain3x	Analog gain of 3x.
iadcCfgAnalogGain4x	Analog gain of 4x.

IADC_CfgReference_t

IADC_CfgReference_t

IADC Reference.

Enumerator	
iadcCfgReferenceInt1V2	Internal 1.2V Band Gap Reference (buffered) to ground.
iadcCfgReferenceExt1V25	External reference (unbuffered) VREFP to VREFN.
iadcCfgReferenceVddx	VDDX (unbuffered) to ground.
iadcCfgReferenceVddX0P8Buf	0.8 * VDDX (buffered) to ground.

IADC_CfgTwosComp_t

IADC_CfgTwosComp_t

IADC Two's complement results.

Enumerator	
iadcCfgTwosCompAuto	Automatic.
iadcCfgTwosCompUnipolar	All results in unipolar format.
iadcCfgTwosCompBipolar	All results in bipolar (2's complement) format.

IADC_TriggerSel_t

IADC_TriggerSel_t

IADC trigger action.

Enumerator

iadcTriggerSelImmediate	Start single/scan queue immediately.
iadcTriggerSelTimer	Timer starts single/scan queue.
iadcTriggerSelPrs0SameClk	PRS0 from timer in same clock group starts single/scan queue
iadcTriggerSelPrs0PosEdge	PRS0 positive edge starts single/scan queue
iadcTriggerSelPrs0NegEdge	PRS0 negative edge starts single/scan queue

IADC_TriggerAction_t

IADC_TriggerAction_t

IADC trigger action.

Enumerator	
iadcTriggerActionOnce	Convert single/scan queue once per trigger
iadcTriggerActionContinuous	Convert single/scan queue continuously

IADC_FifoCfgDvl_t

IADC_FifoCfgDvl_t

IADC data valid level before requesting DMA transfer.

Enumerator	
iadcFifoCfgDvl1	Data valid level is 1 before requesting DMA transfer.
iadcFifoCfgDvl2	Data valid level is 2 before requesting DMA transfer.
iadcFifoCfgDvl3	Data valid level is 3 before requesting DMA transfer.
iadcFifoCfgDvl4	Data valid level is 4 before requesting DMA transfer.

IADC_DigitalAveraging_t

IADC_DigitalAveraging_t

IADC digital averaging function.

Enumerator	
iadcDigitalAverage1	Average over 1 sample (no averaging).
iadcDigitalAverage2	Average over 2 samples.
iadcDigitalAverage4	Average over 4 samples.
iadcDigitalAverage8	Average over 8 samples.
iadcDigitalAverage16	Average over 16 samples.

Function Documentation

IADC_init

```
void IADC_init (IADC_TypeDef * iadc, const IADC_Init_t * init, const IADC_AllConfigs_t * allConfigs)
```

Initialize IADC.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
const IADC_Init_t *	[in]	init	Pointer to IADC initialization structure.
const IADC_AllConfigs_t *	[in]	allConfigs	Pointer to structure holding all configs.

Initializes common parts for both single conversion and scan sequence. In addition, single and/or scan control configuration must be done, please refer to [IADC_initSingle\(\)](#) and [IADC_initScan\(\)](#) respectively.

Note

- This function will stop any ongoing conversions.

IADC_initScan

```
void IADC_initScan (IADC_TypeDef * iadc, const IADC_InitScan_t * init, const IADC_ScanTable_t * scanTable)
```

Initialize IADC scan sequence.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
const IADC_InitScan_t *	[in]	init	Pointer to IADC initialization structure.
const IADC_ScanTable_t *	[in]	scanTable	Pointer to IADC scan table structure.

This function will configure scan mode and set up entries in the scan table. The scan table mask can be updated by calling [IADC_updateScanMask](#).

Note

- This function will stop any ongoing conversions.
- If an even numbered pin is selected for the positive input, the negative input must use an odd numbered pin and vice versa.

IADC_initSingle

```
void IADC_initSingle (IADC_TypeDef * iadc, const IADC_InitSingle_t * init, const IADC_SingleInput_t * input)
```

Initialize single IADC conversion.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
const IADC_InitSingle_t *	[in]	init	Pointer to IADC single initialization structure.
const IADC_SingleInput_t *	[in]	input	Pointer to IADC single input selection initialization structure.

This function will initialize the single conversion and configure the single input selection.

Note

- This function will stop any ongoing conversions.
-

If an even numbered pin is selected for the positive input, the negative input must use an odd numbered pin and vice versa.

IADC_updateSingleInput

```
void IADC_updateSingleInput (IADC_TypeDef * iadc, const IADC_SingleInput_t * input)
```

Update IADC single input selection.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
const IADC_SingleInput_t *	[in]	input	Pointer to single input selection structure.

This function updates the single input selection. The function can be called while single and/or scan conversions are ongoing and the new input configuration will take place on the next single conversion.

Note

- If an even numbered pin is selected for the positive input, the negative input must use an odd numbered pin and vice versa.

IADC_setScanMask

```
void IADC_setScanMask (IADC_TypeDef * iadc, uint32_t mask)
```

Set mask of IADC scan table entries to include in scan.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint32_t	[in]	mask	Mask of scan table entries to include in scan.

Set mask of scan table entries to include in next scan. This function can be called while scan conversions are ongoing, but the new scan mask will take effect once the ongoing scan is completed.

IADC_updateScanEntry

```
void IADC_updateScanEntry (IADC_TypeDef * iadc, uint8_t id, IADC_ScanTableEntry_t * entry)
```

Add/update entry in scan table.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint8_t	[in]	id	ID of scan table entry to add.
IADC_ScanTableEntry_t *	[in]	entry	Pointer to scan table entry structure.

This function will update or add an entry in the scan table with a specific ID.

Note

This function will stop any ongoing conversions.

IADC_reset

```
void IADC_reset (IADC_TypeDef * iadc)
```

Reset IADC to same state as after a HW reset.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

IADC_calcTimebase

```
uint8_t IADC_calcTimebase (IADC_TypeDef * iadc, uint32_t srcClkFreq)
```

Calculate timebase value in order to get a timebase providing at least 1us.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint32_t	[in]	srcClkFreq	Frequency in Hz of reference CLK_SRC_ADC clock. Set to 0 to derive srcClkFreq from CLK_CMU_ADC and prescaler HSCLKRATE.

Returns

- Timebase value to use for IADC in order to achieve at least 1 us.

IADC_calcSrcClkPrescale

```
uint8_t IADC_calcSrcClkPrescale (IADC_TypeDef * iadc, uint32_t srcClkFreq, uint32_t cmuClkFreq)
```

Calculate prescaler for CLK_SRC_ADC high speed clock.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint32_t	[in]	srcClkFreq	CLK_SRC_ADC frequency wanted. The frequency will automatically be adjusted to be within valid range according to reference manual.
uint32_t	[in]	cmuClkFreq	Frequency in Hz of reference CLK_CMU_ADC. Set to 0 to use currently defined CMU clock setting for the IADC.

The IADC high speed clock is given by: $\text{CLK_SRC_ADC} / (\text{srcClkPrescaler} + 1)$.

Returns

- Divider value to use for IADC in order to achieve a high speed clock value $\leq$ `srcClkFreq`.

IADC_calcAdcClkPrescale

```
uint32_t IADC_calcAdcClkPrescale (IADC_TypeDef * iadc, uint32_t adcClkFreq, uint32_t cmuClkFreq,
IADC_CfgAdcMode_t adcMode, uint8_t srcClkPrescaler)
```

Calculate prescaler for ADC_CLK clock.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint32_t	[in]	adcClkFreq	ADC_CLK frequency wanted. The frequency will automatically be adjusted to be within valid range according to reference manual.
uint32_t	[in]	cmuClkFreq	Frequency in Hz of CLK_CMU_ADC Set to 0 to use currently defined IADC clock setting (in CMU).
IADC_CfgAdcMode_t	[in]	adcMode	Mode for IADC config.
uint8_t	[in]	srcClkPrescaler	Precaler setting for ADC_CLK

The ADC_CLK is given by: $\text{CLK_SRC_ADC} / (\text{adcClkprescale} + 1)$.

Returns

- Divider value to use for IADC in order to achieve a ADC_CLK frequency $\leq$ `adcClkFreq`.

IADC_pullSingleFifoResult

```
IADC_Result_t IADC_pullSingleFifoResult (IADC_TypeDef * iadc)
```

Pull result from single data FIFO.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

The result struct includes both the data and the ID (0x20) if showId was set when initializing single mode.

Note

- Check data valid flag before calling this function.

Returns

- Single conversion result struct holding data and id.

IADC_readSingleResult

```
IADC_Result_t IADC_readSingleResult (IADC_TypeDef * iadc)
```

Read most recent single conversion result.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

The result struct includes both the data and the ID (0x20) if showId was set when initializing single mode. Calling this function will not affect the state of the single data FIFO.

Note

- Check data valid flag before calling this function.

Returns

- Single conversion result struct holding data and id.

IADC_pullScanFifoResult

```
IADC_Result_t IADC_pullScanFifoResult (IADC_TypeDef * iadc)
```

Pull result from scan data FIFO.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

The result struct includes both the data and the ID (0x20) if showId was set when initializing scan entry.

Note

- Check data valid flag before calling this function.

Returns

- Scan conversion result struct holding data and id.

IADC_readScanResult

```
IADC_Result_t IADC_readScanResult (IADC_TypeDef * iadc)
```

Read most recent scan conversion result.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

The result struct includes both the data and the ID (0x20) if showId was set when initializing scan entry. Calling this function will not affect the state of the scan data FIFO.

Note

- Check data valid flag before calling this function.

Returns

- Scan conversion result struct holding data and id.

IADC_getReferenceVoltage

```
uint32_t IADC_getReferenceVoltage (IADC_CfgReference_t reference)
```

Get reference voltage selection.

Parameters

Type	Direction	Argument Name	Description
IADC_CfgReference_t	[in]	reference	IADC Reference selection.

Returns

- IADC reference voltage in millivolts.

IADC_pullSingleFifoData

```
uint32_t IADC_pullSingleFifoData (IADC_TypeDef * iadc)
```

Pull data from single data FIFO.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

If showId was set when initializing single mode, the results will contain the ID (0x20).

Note

- Check data valid flag before calling this function.

Returns

- Single conversion data.

IADC_readSingleData

```
uint32_t IADC_readSingleData (IADC_TypeDef * iadc)
```

Read most recent single conversion data.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

If showId was set when initializing single mode, the data will contain the ID (0x20). Calling this function will not affect the state of the single data FIFO.

Note

- Check data valid flag before calling this function.

Returns

- Single conversion data.

IADC_pullScanFifoData

```
uint32_t IADC_pullScanFifoData (IADC_TypeDef * iadc)
```

Pull data from scan data FIFO.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

If showId was set for the scan entry initialization, the data will contain the ID of the scan entry.

Note

- Check data valid flag before calling this function.

Returns

- Scan conversion data.

IADC_readScanData

```
uint32_t IADC_readScanData (IADC_TypeDef * iadc)
```

Read most recent scan conversion data.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

If showId was set for the scan entry initialization, the data will contain the ID of the scan entry. Calling this function will not affect the state of the scan data FIFO.

Note

- Check data valid flag before calling this function.

Returns

- Scan conversion data.

IADC_clearInt

```
void IADC_clearInt (IADC_TypeDef * iadc, uint32_t flags)
```

Clear one or more pending IADC interrupts.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint32_t	[in]	flags	Pending IADC interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for the IADC module (IADC_IF_nnn).

IADC_disableInt

```
void IADC_disableInt (IADC_TypeDef * iadc, uint32_t flags)
```

Disable one or more IADC interrupts.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint32_t	[in]	flags	IADC interrupt sources to disable. Use a bitwise logic OR combination of valid interrupt flags for the IADC module (IADC_IF_nnn).

IADC_enableInt

```
void IADC_enableInt (IADC_TypeDef * iadc, uint32_t flags)
```

Enable one or more IADC interrupts.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint32_t	[in]	flags	IADC interrupt sources to enable. Use a bitwise logic OR combination of valid interrupt flags for the IADC module (IADC_IF_nnn).

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. Consider using IADC_intClear() prior to enabling if such a pending interrupt should be ignored.

IADC_getInt

```
uint32_t IADC_getInt (IADC_TypeDef * iadc)
```

Get pending IADC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

Note

- The event bits are not cleared by the use of this function.

Returns

- IADC interrupt sources pending. A bitwise logic OR combination of valid interrupt flags for the IADC module (IADC_IF_nnn).

IADC_getEnabledInt

```
uint32_t IADC_getEnabledInt (IADC_TypeDef * iadc)
```

Get enabled and pending IADC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

- Pending and enabled IADC interrupt sources. The return value is the bitwise AND combination of
 - the OR combination of enabled interrupt sources in IADCx_IEN_nnn register (IADCx_IEN_nnn) and
 - the OR combination of valid interrupt flags of the IADC module (IADCx_IF_nnn).

IADC_setInt

```
void IADC_setInt (IADC_TypeDef * iadc, uint32_t flags)
```

Set one or more pending IADC interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
uint32_t	[in]	flags	IADC interrupt sources to set to pending. Use a bitwise logic OR combination of valid interrupt flags for the IADC module (IADC_IF_nnn).

IADC_command

```
void IADC_command (IADC_TypeDef * iadc, IADC_Cmd_t cmd)
```

Start/stop scan sequence, single conversion and/or timer.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.
IADC_Cmd_t	[in]	cmd	Command to be performed.

IADC_getScanMask

```
uint32_t IADC_getScanMask (IADC_TypeDef * iadc)
```

Get the scan mask currently used in the IADC.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

Returns

- Mask of scan table entries currently included in scan.

IADC_getStatus

```
uint32_t IADC_getStatus (IADC_TypeDef * iadc)
```

Get status bits of IADC.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

Returns

- IADC status bits

IADC_getSingleFifoCnt

```
uint8_t IADC_getSingleFifoCnt (IADC_TypeDef * iadc)
```

Get the number of elements in the IADC single FIFO.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

Returns

- Number of elements in single FIFO

IADC_getScanFifoCnt

```
uint8_t IADC_getScanFifoCnt (IADC_TypeDef * iadc)
```

Get the number of elements in the IADC scan FIFO.

Parameters

Type	Direction	Argument Name	Description
IADC_TypeDef *	[in]	iadc	Pointer to IADC peripheral register block.

Returns

- Number of elements in scan FIFO

IADC_portPinToNegInput

```
IADC_NegInput_t IADC_portPinToNegInput (GPIO_Port_TypeDef port, uint8_t pin)
```

Convert the GPIO port/pin to IADC negative input selection.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	GPIO port
uint8_t	[in]	pin	GPIO in

Returns

- IADC negative input selection

IADC_portPinToPosInput

IADC_PosInput_t IADC_portPinToPosInput ([GPIO_Port_TypeDef](#) port, uint8_t pin)

Convert the GPIO port/pin to IADC positive input selection.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	GPIO port
uint8_t	[in]	pin	GPIO in

Returns

- IADC positive input selection

Macro Definition Documentation

IADC_INIT_DEFAULT

#define IADC_INIT_DEFAULT

Value:

```
0  {
0      false,                /* IADC clock not disabled on PRS0 */ \
0      false,                /* IADC clock not disabled on PRS1 */ \
0      false,                /* Do not halt during debug */ \
0      iadcWarmupNormal,     /* IADC shutdown after each conversion. */ \
0      0,                    /* Calculate timebase. */ \
0      0,                    /* Max IADC clock rate. */ \
0      _IADC_TIMER_TIMER_DEFAULT, /* Use HW default value. */ \
0      _IADC_CMPTHR_ADGT_DEFAULT, /* Use HW default value. */ \
0      _IADC_CMPTHR_ADLT_DEFAULT, /* Use HW default value. */ \
0  }
```

Default config for IADC init structure.

IADC_CONFIG_DEFAULT

#define IADC_CONFIG_DEFAULT

Value:

```
0  {
0      iadcCfgModeNormal,    /* Normal mode for IADC. */ \
0      iadcCfgOsrHighSpeed2x, /* 2x high speed over sampling. */ \
0      iadcCfgAnalogGain1x,  /* 1x analog gain. */ \
0      iadcCfgReferenceInt1V2, /* Internal 1.2V band gap reference. */ \
0      iadcCfgTwosCompAuto,   /* Automatic Two's Complement. */ \
0      0,                    /* Max IADC analog clock rate. */ \
0      1210,                 /* Vref expressed in millivolts. */ \
0  }
```

```

0 | iadcDigitalAverage1          /* No averaging. */          \
0 | }

```

Default IADC config structure.

IADC_ALLCONFIGS_DEFAULT

```
#define IADC_ALLCONFIGS_DEFAULT
```

Value:

```

0 | {
0 | {
0 |     IADC_CONFIG_DEFAULT,
0 |     IADC_CONFIG_DEFAULT
0 | }
0 | }

```

Default IADC structure for all configs.

IADC_INITSCAN_DEFAULT

```
#define IADC_INITSCAN_DEFAULT
```

Value:

```

0 | {
0 |     iadcAlignRight12,          /* Results 12-bit right aligned */      \
0 |     false,                    /* Do not show ID in result */          \
0 |     iadcFifoCfgDvl14,         /* Use HW default value. */            \
0 |     false,                    /* Do not wake up DMA on scan FIFO DVL */ \
0 |     iadcTriggerSelImmediate,   /* Start scan immediately on trigger */  \
0 |     iadcTriggerActionOnce,     /* Convert once on scan trigger */      \
0 |     false                     /* Do not start scan queue */           \
0 | }

```

Default config for IADC scan init structure.

IADC_INITSINGLE_DEFAULT

```
#define IADC_INITSINGLE_DEFAULT
```

Value:

```

0 | {
0 |     iadcAlignRight12,          /* Results 12-bit right aligned */      \
0 |     false,                    /* Do not show ID in result */          \
0 |     iadcFifoCfgDvl14,         /* Use HW default value. */            \
0 |     false,                    /* Do not wake up DMA on single FIFO DVL */ \
0 |     iadcTriggerSelImmediate,   /* Start single immediately on trigger */ \
0 |     iadcTriggerActionOnce,     /* Convert once on single trigger */    \
0 |     false,                    /* No tailgating */                    \
0 |     false                     /* Do not start single queue */         \
0 | }

```

Default config for IADC single init structure.

IADC_SINGLEINPUT_DEFAULT

```
#define IADC_SINGLEINPUT_DEFAULT
```

Value:

```
0 | {
0 |     iadcNegInputGnd,      /* Negative input GND */ \
0 |     iadcPosInputGnd,     /* Positive input GND */ \
0 |     0,                  /* Config 0 */ \
0 |     false               /* Do not compare results */ \
0 | }
0 |
```

Default config for IADC single input structure.

IADC_SCANTABLEENTRY_DEFAULT

```
#define IADC_SCANTABLEENTRY_DEFAULT
```

Value:

```
0 | {
0 |     iadcNegInputGnd, /* Negative input GND */ \
0 |     iadcPosInputGnd, /* Positive input GND */ \
0 |     0,              /* Config 0 */ \
0 |     false,         /* Do not compare results */ \
0 |     false          /* Do not include in scan */ \
0 | }
0 |
```

Default config for IADC scan table entry structure.

IADC_SCANTABLE_DEFAULT

```
#define IADC_SCANTABLE_DEFAULT
```

Value:

```
0 | {
0 |     {
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |         IADC_SCANTABLEENTRY_DEFAULT, \
0 |     }
0 | }
0 |
```

Default IADC structure for scan table.

IADC_Init_t

IADC init structure, common for single conversion and scan sequence.

Public Attributes

	bool	iadcClkSuspend0 Suspend IADC_CLK when in scan mode until PRS trigger.
	bool	iadcClkSuspend1 Suspend IADC_CLK when in single mode until PRS trigger.
	bool	debugHalt Halt IADC during debug mode.
IADC_Warmup_t	warmup	IADC warmup mode.
	uint8_t	timebase IADC clock cycles (timebase+1) corresponding to 1us.
	uint8_t	srcClkPrescale User requested source clock divider (prescale+1) which will be used if the calculated prescaler value is less.
	uint16_t	timerCycles Number of ADC_CLK cycles per TIMER event.
	uint16_t	greaterThanEqualThres Digital window comparator greater-than or equal threshold.
	uint16_t	lessThanEqualThres Digital window comparator less-than or equal threshold.

Public Attribute Documentation

iadcClkSuspend0

```
bool IADC_Init_t::iadcClkSuspend0
```

Suspend IADC_CLK when in scan mode until PRS trigger.

iadcClkSuspend1

```
bool IADC_Init_t::iadcClkSuspend1
```

Suspend IADC_CLK when in single mode until PRS trigger.

debugHalt

```
bool IADC_Init_t::debugHalt
```

Halt IADC during debug mode.

warmup

```
IADC_Warmup_t IADC_Init_t::warmup
```

IADC warmup mode.

timebase

```
uint8_t IADC_Init_t::timebase
```

IADC clock cycles (timebase+1) corresponding to 1us.

Used as time reference for IADC delays, e.g. warmup. If the user sets timebase to 0, then IADC_Init() will calculate the timebase using the currently defined CMU clock setting for the IADC.

srcClkPrescale

```
uint8_t IADC_Init_t::srcClkPrescale
```

User requested source clock divider (prescale+1) which will be used if the calculated prescaler value is less.

timerCycles

```
uint16_t IADC_Init_t::timerCycles
```

Number of ADC_CLK cycles per TIMER event.

greaterThanEqualThres

```
uint16_t IADC_Init_t::greaterThanEqualThres
```

Digital window comparator greater-than or equal threshold.

lessThanEqualThres

```
uint16_t IADC_Init_t::lessThanEqualThres
```

Digital window comparator less-than or equal threshold.

IADC_Config_t

IADC config structure.

Public Attributes

<code>IADC_CfgAdcMode_t</code>	<code>adcMode</code> IADC mode; Normal, High speed or High Accuracy.
<code>IADC_CfgOsrHighSpeed_t</code>	<code>osrHighSpeed</code> Over sampling ratio for High Speed and Normal modes.
<code>IADC_CfgAnalogGain_t</code>	<code>analogGain</code> Analog gain.
<code>IADC_CfgReference_t</code>	<code>reference</code> Reference selection.
<code>IADC_CfgTwosComp_t</code>	<code>twosComplement</code> Two's complement reporting.
<code>uint32_t</code>	<code>adcClkPrescale</code> ADC_CLK divider (prescale+1).
<code>uint32_t</code>	<code>vRef</code> Vref magnitude expressed in millivolts.
<code>IADC_DigitalAveraging_t</code>	<code>digAvg</code> Digital average mode.

Public Attribute Documentation

adcMode

`IADC_CfgAdcMode_t IADC_Config_t::adcMode`

IADC mode; Normal, High speed or High Accuracy.

osrHighSpeed

`IADC_CfgOsrHighSpeed_t IADC_Config_t::osrHighSpeed`

Over sampling ratio for High Speed and Normal modes.

analogGain

`IADC_CfgAnalogGain_t IADC_Config_t::analogGain`

Analog gain.

reference

```
IADC_CfgReference_t IADC_Config_t::reference
```

Reference selection.

twosComplement

```
IADC_CfgTwosComp_t IADC_Config_t::twosComplement
```

Two's complement reporting.

adcClkPrescale

```
uint32_t IADC_Config_t::adcClkPrescale
```

ADC_CLK divider (prescale+1).

vRef

```
uint32_t IADC_Config_t::vRef
```

Vref magnitude expressed in millivolts.

digAvg

```
IADC_DigitalAveraging_t IADC_Config_t::digAvg
```

Digital average mode.

IADC_AllConfigs_t

Structure for all IADC configs.

Public Attributes

[IADC_Config_t](#) [configs](#)
All IADC configs.

Public Attribute Documentation

configs

IADC_Config_t IADC_AllConfigs_t::configs[IADC0_CONFIGNUM]

All IADC configs.

IADC_InitScan_t

IADC scan init structure.

Public Attributes

<code>IADC_Alignment_t</code>	<code>alignment</code> Alignment of data in FIFO.
<code>bool</code>	<code>showId</code> Tag FIFO entry with scan table entry id.
<code>IADC_FifoCfgDvl_t</code>	<code>dataValidLevel</code> Data valid level before requesting DMA transfer.
<code>bool</code>	<code>fifoDmaWakeup</code> Wake-up DMA when FIFO reaches data valid level.
<code>IADC_TriggerSel_t</code>	<code>triggerSelect</code> Trigger selection.
<code>IADC_TriggerAction_t</code>	<code>triggerAction</code> Trigger action.
<code>bool</code>	<code>start</code> Start scan immediately.

Public Attribute Documentation

alignment

```
IADC_Alignment_t IADC_InitScan_t::alignment
```

Alignment of data in FIFO.

showId

```
bool IADC_InitScan_t::showId
```

Tag FIFO entry with scan table entry id.

dataValidLevel

```
IADC_FifoCfgDvl_t IADC_InitScan_t::dataValidLevel
```

Data valid level before requesting DMA transfer.

fifoDmaWakeup

```
bool IADC_InitScan_t::fifoDmaWakeup
```

Wake-up DMA when FIFO reaches data valid level.

triggerSelect

```
IADC_TriggerSel_t IADC_InitScan_t::triggerSelect
```

Trigger selection.

triggerAction

```
IADC_TriggerAction_t IADC_InitScan_t::triggerAction
```

Trigger action.

start

```
bool IADC_InitScan_t::start
```

Start scan immediately.

IADC_InitSingle_t

IADC single init structure.

Public Attributes

IADC_Alignment_t	alignment Alignment of data in FIFO.
bool	showId Tag FIFO entry with single indicator (0x20).
IADC_FifoCfgDvl_t	dataValidLevel Data valid level before requesting DMA transfer.
bool	fifoDmaWakeup Wake-up DMA when FIFO reaches data valid level.
IADC_TriggerSel_t	triggerSelect Trigger selection.
IADC_TriggerAction_t	triggerAction Trigger action.
bool	singleTailgate If true, wait until end of SCAN queue before single queue warmup and conversion.
bool	start Start scan immediately.

Public Attribute Documentation

alignment

IADC_Alignment_t IADC_InitSingle_t::alignment

Alignment of data in FIFO.

showId

bool IADC_InitSingle_t::showId

Tag FIFO entry with single indicator (0x20).

dataValidLevel

IADC_FifoCfgDvl_t IADC_InitSingle_t::dataValidLevel

Data valid level before requesting DMA transfer.

fifoDmaWakeup

```
bool IADC_InitSingle_t::fifoDmaWakeup
```

Wake-up DMA when FIFO reaches data valid level.

triggerSelect

```
IADC_TriggerSel_t IADC_InitSingle_t::triggerSelect
```

Trigger selection.

triggerAction

```
IADC_TriggerAction_t IADC_InitSingle_t::triggerAction
```

Trigger action.

singleTailgate

```
bool IADC_InitSingle_t::singleTailgate
```

If true, wait until end of SCAN queue before single queue warmup and conversion.

start

```
bool IADC_InitSingle_t::start
```

Start scan immediately.

IADC_SingleInput_t

IADC single input selection structure.

Public Attributes

<code>IADC_NegInput_t</code>	<code>negInput</code> Port/pin input for the negative side of the ADC.
<code>IADC_PosInput_t</code>	<code>posInput</code> Port/pin input for the positive side of the ADC.
<code>uint8_t</code>	<code>configId</code> Configuration id.
<code>bool</code>	<code>compare</code> Perform digital window comparison on the result from this entry.

Public Attribute Documentation

negInput

`IADC_NegInput_t IADC_SingleInput_t::negInput`

Port/pin input for the negative side of the ADC.

posInput

`IADC_PosInput_t IADC_SingleInput_t::posInput`

Port/pin input for the positive side of the ADC.

configId

`uint8_t IADC_SingleInput_t::configId`

Configuration id.

compare

`bool IADC_SingleInput_t::compare`

Perform digital window comparison on the result from this entry.

IADC_ScanTableEntry_t

IADC scan table entry structure.

Public Attributes

<code>IADC_NegInput_t</code>	<code>negInput</code> Port/pin input for the negative side of the ADC.
<code>IADC_PosInput_t</code>	<code>posInput</code> Port/pin input for the positive side of the ADC.
<code>uint8_t</code>	<code>configId</code> Configuration id.
<code>bool</code>	<code>compare</code> Perform digital window comparison on the result from this entry.
<code>bool</code>	<code>includeInScan</code> Include this scan table entry in scan operation.

Public Attribute Documentation

negInput

`IADC_NegInput_t IADC_ScanTableEntry_t::negInput`

Port/pin input for the negative side of the ADC.

posInput

`IADC_PosInput_t IADC_ScanTableEntry_t::posInput`

Port/pin input for the positive side of the ADC.

configId

`uint8_t IADC_ScanTableEntry_t::configId`

Configuration id.

compare

`bool IADC_ScanTableEntry_t::compare`

Perform digital window comparison on the result from this entry.

includeInScan

```
bool IADC_ScanTableEntry_t::includeInScan
```

Include this scan table entry in scan operation.

IADC_ScanTable_t

Structure for IADC scan table.

Public Attributes

[IADC_ScanTableEntry_t](#) [entries](#)
IADC scan table entries.

Public Attribute Documentation

entries

IADC_ScanTableEntry_t IADC_ScanTable_t::entries[IADCO_ENTRIES]

IADC scan table entries.

IADC_Result_t

Structure holding IADC result, including data and ID.

Public Attributes

- uint32_t

data

ADC sample data.
- uint8_t

id

ID of FIFO entry; Scan table entry id or single indicator (0x20).

Public Attribute Documentation

data

uint32_t IADC_Result_t::data

ADC sample data.

id

uint8_t IADC_Result_t::id

ID of FIFO entry; Scan table entry id or single indicator (0x20).

GPCRC - General Purpose CRC

GPCRC - General Purpose CRC

General Purpose Cyclic Redundancy Check (GPCRC) API.

The GPCRC API functions provide full support for the GPCRC peripheral.

The GPCRC module is a peripheral that implements a Cyclic Redundancy Check (CRC) function. It supports a fixed 32-bit polynomial and a user configurable 16-bit polynomial. The fixed 32-bit polynomial is the commonly used IEEE 802.3 polynomial 0x04C11DB7.

When using a 16-bit polynomial it is up to the user to choose a polynomial that fits the application. Commonly used 16-bit polynomials are 0x1021 (CCITT-16), 0x3D65 (IEC16-MBus), and 0x8005 (ZigBee, 802.15.4, and USB). See this link for other polynomials: https://en.wikipedia.org/wiki/Cyclic_redundancy_check

Before a CRC calculation can begin, call the [GPCRC_Start](#) function. This function will reset CRC calculation by copying the configured initialization value over to the CRC data register.

There are two ways of sending input data to the GPCRC. Either write the input data into the input data register using input functions [GPCRC_InputU32](#), [GPCRC_InputU16](#) and [GPCRC_InputU8](#), or the user can configure [LDMA - Linked DMA](#) to transfer data directly to one of the GPCRC input data registers.

Examples of GPCRC usage:

A CRC-32 Calculation:

```
#if !defined(_SILICON_LABS_32B_SERIES_2)
/* The GPCRC is a high frequency peripheral so we need to enable the
 * HPER clock in addition to the GPCRC clock. */
CMU_ClockEnable(cmuClock_HPER, true);
CMU_ClockEnable(cmuClock_GPCRC, true);
#endif

uint32_t checksum;

/* Initialize GPCRC, 32-bit fixed polynomial is default */
GPCRC_Init_TypeDef init = GPCRC_INIT_DEFAULT;
init.initValue = 0xFFFFFFFF; // Standard CRC-32 init value
GPCRC_Init(GPCRC, &init);

GPCRC_Start(GPCRC);
GPCRC_InputU8(GPCRC, 0xC5);
/* According to the CRC-32 specification, the end result should be inverted */
checksum = ~GPCRC_DataRead(GPCRC);
/* The checksum is now 0x390CD9B2 */
```

A CRC-16 Calculation:

```
#if !defined(_SILICON_LABS_32B_SERIES_2)
/* The GPCRC is a high frequency peripheral so we need to enable the
 * HUPER clock in addition to the GPCRC clock. */
CMU_ClockEnable(cmuClock_HUPER, true);
CMU_ClockEnable(cmuClock_GPCRC, true);
#endif

uint16_t checksum;

/* Initialize GPCRC with the 16-bit polynomial 0x8005. */
GPCRC_Init_TypeDef init = GPCRC_INIT_DEFAULT;
init.crcPoly = 0x8005;
GPCRC_Init(GPCRC, &init);

GPCRC_Start(GPCRC);
GPCRC_InputU8(GPCRC, 0xAB);
checksum = (uint16_t) GPCRC_DataRead(GPCRC);
/* The checksum is now 0xBF41 */
```

A CRC-CCITT calculation:

```
#if !defined(_SILICON_LABS_32B_SERIES_2)
/* The GPCRC is a high frequency peripheral so we need to enable the
 * HUPER clock in addition to the GPCRC clock. */
CMU_ClockEnable(cmuClock_HUPER, true);
CMU_ClockEnable(cmuClock_GPCRC, true);
#endif

/* Initialize GPCRC with the 16-bit CRC-CCIT polynomial 0x1021 and use
 * the initial value of 0xFFFF. */
GPCRC_Init_TypeDef init = GPCRC_INIT_DEFAULT;
init.crcPoly = 0x1021;
init.initValue = 0xFFFF;
init.reverseBits = true;
GPCRC_Init(GPCRC, &init);

char * input = "123456789";

GPCRC_Start(GPCRC);
for (size_t i = 0; i < strlen(input); i++) {
    GPCRC_InputU8(GPCRC, (uint8_t) input[i]);
}

uint16_t checksum = (uint16_t) GPCRC_DataReadBitReversed(GPCRC);
/* checksum is now 0x29B1 */
```

Modules

[GPCRC_Init_TypeDef](#)

Functions

- void [GPCRC_Init](#)(GPCRC_TypeDef *gpcrc, const GPCRC_Init_TypeDef *init)
Initialize the General Purpose Cyclic Redundancy Check (GPCRC) module.
- void [GPCRC_Reset](#)(GPCRC_TypeDef *gpcrc)
Reset GPCRC registers to the hardware reset state.
- void [GPCRC_Enable](#)(GPCRC_TypeDef *gpcrc, bool enable)
Enable/disable GPCRC.
- void [GPCRC_Start](#)(GPCRC_TypeDef *gpcrc)
Issue a command to initialize the CRC calculation.

void	GPCRC_InitValueSet (GPCRC_TypeDef *gpcrc, uint32_t initValue)	Set the initialization value of the CRC.
void	GPCRC_InputU32 (GPCRC_TypeDef *gpcrc, uint32_t data)	Write a 32-bit value to the input data register of the CRC.
void	GPCRC_InputU16 (GPCRC_TypeDef *gpcrc, uint16_t data)	Write a 16-bit value to the input data register of the CRC.
void	GPCRC_InputU8 (GPCRC_TypeDef *gpcrc, uint8_t data)	Write an 8-bit value to the CRC input data register.
uint32_t	GPCRC_DataRead (GPCRC_TypeDef *gpcrc)	Read the CRC data register.
uint32_t	GPCRC_DataReadBitReversed (GPCRC_TypeDef *gpcrc)	Read the data register of the CRC bit reversed.
uint32_t	GPCRC_DataReadByteReversed (GPCRC_TypeDef *gpcrc)	Read the data register of the CRC byte reversed.

Macros

#define	GPCRC_INIT_DEFAULT undefined	Default configuration for GPCRC_Init_TypeDef structure.
---------	--	---

Function Documentation

GPCRC_Init

```
void GPCRC_Init (GPCRC_TypeDef * gpcrc, const GPCRC\_Init\_TypeDef * init)
```

Initialize the General Purpose Cyclic Redundancy Check (GPCRC) module.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	A pointer to the GPCRC peripheral register block.
const GPCRC_Init_TypeDef *	[in]	init	A pointer to the initialization structure used to configure the GPCRC.

Use this function to configure the operational parameters of the GPCRC, such as the polynomial to use and how the input should be preprocessed before entering the CRC calculation.

Note

- This function will not copy the initialization value to the data register to prepare for a new CRC calculation. Either call [GPCRC_Start](#) before each calculation or by use the autoInit functionality.

GPCRC_Reset

```
void GPCRC_Reset (GPCRC_TypeDef * gpcrc)
```

Reset GPCRC registers to the hardware reset state.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	A pointer to the GPCRC peripheral register block.

Note

- The data registers are not reset by this function.

GPCRC_Enable

```
void GPCRC_Enable (GPCRC_TypeDef * gpcrc, bool enable)
```

Enable/disable GPCRC.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Pointer to GPCRC peripheral register block.
bool	[in]	enable	True to enable GPCRC, false to disable.

GPCRC_Start

```
void GPCRC_Start (GPCRC_TypeDef * gpcrc)
```

Issue a command to initialize the CRC calculation.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Pointer to GPCRC peripheral register block.

Issues the command INIT in GPCRC_CMD that initializes the CRC calculation by writing the initial values to the DATA register.

GPCRC_InitValueSet

```
void GPCRC_InitValueSet (GPCRC_TypeDef * gpcrc, uint32_t initValue)
```

Set the initialization value of the CRC.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Value to use to initialize a CRC calculation. This value is moved into the data register when calling GPCRC_Start
uint32_t	[in]	initValue	Pointer to GPCRC peripheral register block.

GPCRC_InputU32

```
void GPCRC_InputU32 (GPCRC_TypeDef * gpcrc, uint32_t data)
```

Write a 32-bit value to the input data register of the CRC.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Pointer to GPCRC peripheral register block.
uint32_t	[in]	data	Data to be written to the input data register.

Use this function to write a 32-bit input data to the CRC. CRC calculation is based on the provided input data using the configured CRC polynomial.

GPCRC_InputU16

```
void GPCRC_InputU16 (GPCRC_TypeDef * gpcrc, uint16_t data)
```

Write a 16-bit value to the input data register of the CRC.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Pointer to GPCRC peripheral register block.
uint16_t	[in]	data	Data to be written to the input data register.

Use this function to write a 16 bit input data to the CRC. CRC calculation is based on the provided input data using the configured CRC polynomial.

GPCRC_InputU8

```
void GPCRC_InputU8 (GPCRC_TypeDef * gpcrc, uint8_t data)
```

Write an 8-bit value to the CRC input data register.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Pointer to GPCRC peripheral register block.
uint8_t	[in]	data	Data to be written to the input data register.

Use this function to write an 8-bit input data to the CRC. CRC calculation is based on the provided input data using the configured CRC polynomial.

GPCRC_DataRead

```
uint32_t GPCRC_DataRead (GPCRC_TypeDef * gpcrc)
```

Read the CRC data register.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Pointer to GPCRC peripheral register block.

Use this function to read the calculated CRC value.

Returns

- Content of the CRC data register.

GPCRC_DataReadBitReversed

```
uint32_t GPCRC_DataReadBitReversed (GPCRC_TypeDef * gpcrc)
```

Read the data register of the CRC bit reversed.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Pointer to GPCRC peripheral register block.

Use this function to read the calculated CRC value bit reversed. When using a 32-bit polynomial, bits [31:0] are reversed, when using a 16-bit polynomial, bits [15:0] are reversed.

Returns

- Content of the CRC data register bit reversed.

GPCRC_DataReadByteReversed

```
uint32_t GPCRC_DataReadByteReversed (GPCRC_TypeDef * gpcrc)
```

Read the data register of the CRC byte reversed.

Parameters

Type	Direction	Argument Name	Description
GPCRC_TypeDef *	[in]	gpcrc	Pointer to GPCRC peripheral register block.

Use this function to read the calculated CRC value byte reversed.

Returns

- Content of the CRC data register byte reversed.

Macro Definition Documentation

GPCRC_INIT_DEFAULT

```
#define GPCRC_INIT_DEFAULT
```

Value:

```
0 | {
0 |     0x04C11DB7UL,      /* CRC32 Polynomial value. */
0 |     0x00000000UL,      /* Initialization value. */
0 |     false,             /* Byte order is normal. */
0 |     false,             /* Bit order is not reversed on output. */
0 |     false,             /* Disable byte mode. */
0 |     false,             /* Disable automatic initialization on data read. */
0 |     true,              /* Enable GPCRC. */
0 | }
```

Default configuration for [GPCRC_Init_TypeDef](#) structure.

GPCRC_Init_TypeDef

CRC initialization structure.

Public Attributes

uint32_t	crcPoly	CRC polynomial value.
uint32_t	initValue	CRC initialization value.
bool	reverseByteOrder	Reverse byte order.
bool	reverseBits	Reverse bits within each input byte.
bool	enableByteMode	Enable/disable byte mode.
bool	autoInit	Enable automatic initialization by re-seeding the CRC result based on the init value after reading one of the CRC data registers.
bool	enable	Enable/disable GPCRC when initialization is completed.

Public Attribute Documentation

crcPoly

```
uint32_t GPCRC_Init_TypeDef::crcPoly
```

CRC polynomial value.

GPCRC supports either a fixed 32-bit polynomial or a user-configurable 16 bit polynomial. The fixed 32-bit polynomial is the one used in IEEE 802.3, which has the value 0x04C11DB7. To use the 32-bit fixed polynomial, assign 0x04C11DB7 to the `crcPoly` field. To use a 16-bit polynomial, assign a value to `crcPoly` where the upper 16 bits are zero.

The polynomial should be written in normal bit order. For instance, to use the CRC-16 polynomial $X^{16} + X^{15} + X^2 + 1$, first convert it to hex representation and remove the highest order term of the polynomial. This will give 0x8005 as the value to write into `crcPoly`.

initValue

```
uint32_t GPCRC_Init_TypeDef::initValue
```

CRC initialization value.

This value is assigned to the GPCRC_INIT register. The `initValue` is loaded into the data register when calling the [GPCRC_Start](#) function or when one of the data registers are read while [autoInit](#) is enabled.

reverseByteOrder

```
bool GPCRC_Init_TypeDef::reverseByteOrder
```

Reverse byte order.

This has an effect when sending a 32-bit word or 16-bit half word input to the CRC calculation. When set to true, the input bytes are reversed before entering the CRC calculation. When set to false, the input bytes stay in the same order.

reverseBits

```
bool GPCRC_Init_TypeDef::reverseBits
```

Reverse bits within each input byte.

This setting enables or disables byte level bit reversal. When byte-level bit reversal is enabled, then each byte of input data will be reversed before entering CRC calculation.

enableByteMode

```
bool GPCRC_Init_TypeDef::enableByteMode
```

Enable/disable byte mode.

When byte mode is enabled, then all input is treated as single byte input even though the input is a 32-bit word or a 16-bit half word. Only the least significant byte of the data-word will be used for CRC calculation for all writes.

autoInit

```
bool GPCRC_Init_TypeDef::autoInit
```

Enable automatic initialization by re-seeding the CRC result based on the init value after reading one of the CRC data registers.

enable

```
bool GPCRC_Init_TypeDef::enable
```

Enable/disable GPCRC when initialization is completed.

GPIO - General Purpose Input/Output

GPIO - General Purpose Input/Output

General Purpose Input/Output (GPIO) API.

This module contains functions to control the GPIO peripheral of Silicon Labs 32-bit MCUs and SoCs. The GPIO peripheral is used for pin configuration and direct pin manipulation and sensing as well as routing for peripheral pin connections.

Enumerations

```
enum GPIO_Port_TypeDef {
    gpioPortA = 0
    gpioPortB = 1
    gpioPortC = 2
    gpioPortD = 3
}
GPIO ports IDs.

enum GPIO_Mode_TypeDef {
    gpioModeDisabled = _GPIO_P_MODEL_MODE0_DISABLED
    gpioModeInput = _GPIO_P_MODEL_MODE0_INPUT
    gpioModeInputPull = _GPIO_P_MODEL_MODE0_INPUTPULL
    gpioModeInputPullFilter = _GPIO_P_MODEL_MODE0_INPUTPULLFILTER
    gpioModePushPull = _GPIO_P_MODEL_MODE0_PUSHPULL
    gpioModePushPullAlternate = _GPIO_P_MODEL_MODE0_PUSHPULLALT
    gpioModeWiredOr = _GPIO_P_MODEL_MODE0_WIREDOR
    gpioModeWiredOrPullDown = _GPIO_P_MODEL_MODE0_WIREDORPULLDOWN
    gpioModeWiredAnd = _GPIO_P_MODEL_MODE0_WIREDAND
    gpioModeWiredAndFilter = _GPIO_P_MODEL_MODE0_WIREDANDFILTER
    gpioModeWiredAndPullUp = _GPIO_P_MODEL_MODE0_WIREDANDPULLUP
    gpioModeWiredAndPullUpFilter = _GPIO_P_MODEL_MODE0_WIREDANDPULLUPFILTER
    gpioModeWiredAndAlternate = _GPIO_P_MODEL_MODE0_WIREDANDALT
    gpioModeWiredAndAlternateFilter = _GPIO_P_MODEL_MODE0_WIREDANDALTFILTER
    gpioModeWiredAndAlternatePullUp = _GPIO_P_MODEL_MODE0_WIREDANDALTPULLUP
    gpioModeWiredAndAlternatePullUpFilter = _GPIO_P_MODEL_MODE0_WIREDANDALTPULLUPFILTER
}
Pin mode.
```

Functions

```
void GPIO_DbgLocationSet(unsigned int location)
    Sets the pin location of the debug pins (Serial Wire interface).

void GPIO_DbgSWDClkEnable(bool enable)
    Enable/disable serial wire clock pin.

void GPIO_DbgSWDIOEnable(bool enable)
    Enable/disable serial wire data I/O pin.

void GPIO_DbgSWOEnable(bool enable)
    Enable/Disable serial wire output pin.

void GPIO_EM4DisablePinWakeup(uint32_t pinmask)
    Disable GPIO pin wake-up from EM4.

void GPIO_EM4EnablePinWakeup(uint32_t pinmask, uint32_t polaritymask)
    Enable GPIO pin wake-up from EM4.
```

uint32_t	GPIO_EM4GetPinWakeupCause (void) Check which GPIO pin(s) that caused a wake-up from EM4.
void	GPIO_EM4SetPinRetention (bool enable) Enable GPIO pin retention of output enable, output value, pull enable, and pull direction in EM4.
void	GPIO_ExtIntConfig (GPIO_Port_TypeDef port, unsigned int pin, unsigned int intNo, bool risingEdge, bool fallingEdge, bool enable) Configure the GPIO external pin interrupt.
void	GPIO_EM4WUExtIntConfig (GPIO_Port_TypeDef port, unsigned int pin, uint32_t intNo, bool polarity, bool enable) Configure EM4WU pins as external level-sensitive interrupts.
void	GPIO_InputSenseSet (uint32_t val, uint32_t mask) Enable/disable input sensing.
void	GPIO_IntClear (uint32_t flags) Clear one or more pending GPIO interrupts.
void	GPIO_IntDisable (uint32_t flags) Disable one or more GPIO interrupts.
void	GPIO_IntEnable (uint32_t flags) Enable one or more GPIO interrupts.
uint32_t	GPIO_EnabledIntGet (void) Get enabled GPIO interrupts.
uint32_t	GPIO_IntGet (void) Get pending GPIO interrupts.
uint32_t	GPIO_IntGetEnabled (void) Get enabled and pending GPIO interrupt flags.
void	GPIO_IntSet (uint32_t flags) Set one or more pending GPIO interrupts from SW.
void	GPIO_Lock (void) Lock the GPIO configuration.
unsigned int	GPIO_PinInGet (GPIO_Port_TypeDef port, unsigned int pin) Read the pad value for a single pin in a GPIO port.
GPIO_Mode_TypeDef	GPIO_PinModeGet (GPIO_Port_TypeDef port, unsigned int pin) Get the mode for a GPIO pin.
void	GPIO_PinModeSet (GPIO_Port_TypeDef port, unsigned int pin, GPIO_Mode_TypeDef mode, unsigned int out) Set the mode for a GPIO pin.
void	GPIO_PinOutClear (GPIO_Port_TypeDef port, unsigned int pin) Set a single pin in GPIO data out port register to 0.
unsigned int	GPIO_PinOutGet (GPIO_Port_TypeDef port, unsigned int pin) Get current setting for a pin in a GPIO port data out register.
void	GPIO_PinOutSet (GPIO_Port_TypeDef port, unsigned int pin) Set a single pin in GPIO data out register to 1.
void	GPIO_PinOutToggle (GPIO_Port_TypeDef port, unsigned int pin) Toggle a single pin in GPIO port data out register.
uint32_t	GPIO_PortInGet (GPIO_Port_TypeDef port) Read the pad values for GPIO port.
void	GPIO_PortOutClear (GPIO_Port_TypeDef port, uint32_t pins) Set bits in DOUT register for a port to 0.

- uint32_t

GPIO_PortOutGet(GPIO_Port_TypeDef port)

Get the current setting for a GPIO port data out register.
- void

GPIO_PortOutSet(GPIO_Port_TypeDef port, uint32_t pins)

Set bits GPIO data out register to 1.
- void

GPIO_PortOutSetVal(GPIO_Port_TypeDef port, uint32_t val, uint32_t mask)

Set GPIO port data out register.
- void

GPIO_PortOutToggle(GPIO_Port_TypeDef port, uint32_t pins)

Toggle pins in GPIO port data out register.
- void

GPIO_SlewrateSet(GPIO_Port_TypeDef port, uint32_t slewrate, uint32_t slewrateAlt)

Set slewrate for pins on a GPIO port.
- void

GPIO_Unlock(void)

Unlock the GPIO configuration.

Enumeration Documentation

GPIO_Port_TypeDef

GPIO_Port_TypeDef

GPIO ports IDs.

Enumerator	
gpioPortA	Port A.
gpioPortB	Port B.
gpioPortC	Port C.
gpioPortD	Port D.

GPIO_Mode_TypeDef

GPIO_Mode_TypeDef

Pin mode.

For more details on each mode, refer to the reference manual.

Enumerator	
gpioModeDisabled	Input disabled.
gpioModeInput	Input enabled.
gpioModeInputPull	Input enabled.
gpioModeInputPullFilter	Input enabled with filter.
gpioModePushPull	Push-pull output.
gpioModePushPullAlternate	Push-pull using alternate control.
gpioModeWiredOr	Wired-or output.
gpioModeWiredOrPullDown	Wired-or output with pull-down.
gpioModeWiredAnd	Open-drain output.
gpioModeWiredAndFilter	Open-drain output with filter.
gpioModeWiredAndPullUp	Open-drain output with pull-up.
gpioModeWiredAndPullUpFilter	Open-drain output with filter and pull-up.
gpioModeWiredAndAlternate	Open-drain output using alternate control.

gpioModeWiredAndAlternateFilter	Open-drain output using alternate control with filter.
gpioModeWiredAndAlternatePullUp	Open-drain output using alternate control with pull-up.
gpioModeWiredAndAlternatePullUpFilter	Open-drain output using alternate control with filter and pull-up.

Function Documentation

GPIO_DbgLocationSet

```
void GPIO_DbgLocationSet (unsigned int location)
```

Sets the pin location of the debug pins (Serial Wire interface).

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	location	The debug pin location to use (0-3).

Note

- Changing the pins used for debugging uncontrolled, may result in a lockout.

GPIO_DbgSWDClkEnable

```
void GPIO_DbgSWDClkEnable (bool enable)
```

Enable/disable serial wire clock pin.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	• false - disable serial wire clock. • true - enable serial wire clock (default after reset).

Note

- Disabling SWDClk will disable the debug interface, which may result in a lockout if done early in startup (before debugger is able to halt core).

GPIO_DbgSWDIOEnable

```
void GPIO_DbgSWDIOEnable (bool enable)
```

Enable/disable serial wire data I/O pin.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	• false - disable serial wire data pin. • true - enable serial wire data pin (default after reset).

Note

Disabling SWDClk will disable the debug interface, which may result in a lockout if done early in startup (before debugger is able to halt core).

GPIO_DbgSWOEnable

```
void GPIO_DbgSWOEnable (bool enable)
```

Enable/Disable serial wire output pin.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	- false - disable serial wire viewer pin (default after reset). - true - enable serial wire viewer pin.

Note

- Enabling this pin is not sufficient to fully enable serial wire output, which is also dependent on issues outside the GPIO module. Refer to [DBG_SWOEnable\(\)](#).

Warnings

- If debug port is locked, SWO pin is not disabled automatically. To avoid information leakage through SWO, disable SWO pin after locking debug port.

GPIO_EM4DisablePinWakeup

```
void GPIO_EM4DisablePinWakeup (uint32_t pinmask)
```

Disable GPIO pin wake-up from EM4.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	pinmask	Bit mask containing the bitwise logic OR of which GPIO pin(s) to disable. Refer to Reference Manuals for pinmask to GPIO port/pin mapping.

GPIO_EM4EnablePinWakeup

```
void GPIO_EM4EnablePinWakeup (uint32_t pinmask, uint32_t polaritymask)
```

Enable GPIO pin wake-up from EM4.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	pinmask	A bitmask containing the bitwise logic OR of which GPIO pin(s) to enable. See Reference Manuals for a pinmask to the GPIO port/pin mapping.
uint32_t	[in]	polaritymask	A bitmask containing the bitwise logic OR of GPIO pin(s) wake-up polarity. See Reference Manuals for pinmask-to-GPIO port/pin mapping.

When the function exits, EM4 mode can be safely entered.

Note

- It is assumed that the GPIO pin modes are set correctly. Valid modes are [gpioModeInput](#) and [gpioModeInputPull](#).

GPIO_EM4GetPinWakeupCause

```
uint32_t GPIO_EM4GetPinWakeupCause (void )
```

Check which GPIO pin(s) that caused a wake-up from EM4.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Bit mask containing the bitwise logic OR of which GPIO pin(s) caused the wake-up. Refer to Reference Manuals for pinmask to GPIO port/pin mapping.

GPIO_EM4SetPinRetention

```
void GPIO_EM4SetPinRetention (bool enable)
```

Enable GPIO pin retention of output enable, output value, pull enable, and pull direction in EM4.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	• true - enable EM4 pin retention. • false - disable EM4 pin retention.

Note

- On series 0 devices EM4 gpio retention can either be turned on or off. On series 1 devices there are three EM4 GPIO retention modes available. These modes are "Disabled", "EM4EXIT" and "SWUNLATCH". Use the [EMU_EM4Init\(\)](#) to configure the GPIO retention mode on a series 1 device.

The behavior of this function depends on the configured GPIO retention mode. If the GPIO retention mode is configured to be "SWUNLATCH" then this function will not change anything. If the retention mode is anything else then this function will set the GPIO retention mode to "EM4EXIT" when the enable argument is true, and "Disabled" when false.

GPIO_ExtIntConfig

```
void GPIO_ExtIntConfig (GPIO_Port_TypeDef port, unsigned int pin, unsigned int intNo, bool risingEdge, bool fallingEdge, bool enable)
```

Configure the GPIO external pin interrupt.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The port to associate with the <code>pin</code> .
unsigned int	[in]	pin	The pin number on the port.
unsigned int	[in]	intNo	The interrupt number to trigger.
bool	[in]	risingEdge	Set to true if the interrupt will be enabled on the rising edge. Otherwise, false.
bool	[in]	fallingEdge	Set to true if the interrupt will be enabled on the falling edge. Otherwise, false.
bool	[in]	enable	Set to true if the interrupt will be enabled after the configuration is complete. False to leave disabled. See GPIO_IntDisable() and GPIO_IntEnable() .

It is recommended to disable interrupts before configuring the GPIO pin interrupt. See [GPIO_IntDisable\(\)](#) for more information.

The GPIO interrupt handler must be in place before enabling the interrupt.

Notice that any pending interrupt for the selected interrupt is cleared by this function.

Note

- On series 0 devices, the pin number parameter is not used. The pin number used on these devices is hardwired to the interrupt with the same number.
On series 1 devices, the pin number can be selected freely within a group. Interrupt numbers are divided into 4 groups (`intNo / 4`) and valid pin number within the interrupt groups are: 0: pins 0-3 (interrupt number 0-3) 1: pins 4-7 (interrupt number 4-7) 2: pins 8-11 (interrupt number 8-11) 3: pins 12-15 (interrupt number 12-15)

GPIO_EM4WUExtIntConfig

```
void GPIO_EM4WUExtIntConfig (GPIO\_Port\_TypeDef port, unsigned int pin, uint32_t intNo, bool polarity, bool enable)
```

Configure EM4WU pins as external level-sensitive interrupts.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The port to associate with the <code>pin</code> .
unsigned int	[in]	pin	The pin number on the port.
uint32_t	[in]	intNo	The EM4WU interrupt number to trigger.
bool	[in]	polarity	true = Active high level-sensitive interrupt. false = Active low level-sensitive interrupt.
bool	[in]	enable	Set to true if the interrupt will be enabled after the configuration is complete. False to leave disabled. See GPIO_IntDisable() and GPIO_IntEnable() .

It is recommended to disable interrupts before configuring the GPIO pin interrupt. See [GPIO_IntDisable\(\)](#) for more information.

The GPIO interrupt handler must be in place before enabling the interrupt.

Notice that any pending interrupt for the selected interrupt is cleared by this function.

Note

The selected port/pin must be mapped to an existant EM4WU interrupt. Each EM4WU signal is connected to a fixed pin. Refer to the Alternate Function Table in the device Datasheet for the location of each EM4WU signal. For example, on xG22 device, the interrupt of EM4WU6 is fixed to pin PC00.

GPIO_InputSenseSet

```
void GPIO_InputSenseSet (uint32_t val, uint32_t mask)
```

Enable/disable input sensing.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	val	Bitwise logic OR of one or more of: - GPIO_INSENSE_INT - interrupt input sensing. - GPIO_INSENSE_PRS - peripheral reflex system input sensing.
uint32_t	[in]	mask	Mask containing bitwise logic OR of bits similar as for <code>val</code> used to indicate which input sense options to disable/enable.

Disabling input sensing if not used, can save some energy consumption.

GPIO_IntClear

```
void GPIO_IntClear (uint32_t flags)
```

Clear one or more pending GPIO interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	Bitwise logic OR of GPIO interrupt sources to clear.

GPIO_IntDisable

```
void GPIO_IntDisable (uint32_t flags)
```

Disable one or more GPIO interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	GPIO interrupt sources to disable.

GPIO_IntEnable

```
void GPIO_IntEnable (uint32_t flags)
```

Enable one or more GPIO interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	GPIO interrupt sources to enable.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [GPIO_IntClear\(\)](#) prior to enabling the interrupt.

GPIO_EnabledIntGet

```
uint32_t GPIO_EnabledIntGet (void )
```

Get enabled GPIO interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Enabled GPIO interrupt sources.

GPIO_IntGet

```
uint32_t GPIO_IntGet (void )
```

Get pending GPIO interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- GPIO interrupt sources pending.

GPIO_IntGetEnabled

```
uint32_t GPIO_IntGetEnabled (void )
```

Get enabled and pending GPIO interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

- Pending and enabled GPIO interrupt sources. The return value is the bitwise AND combination of

- the OR combination of enabled interrupt sources in GPIO_IEN register and
- the OR combination of valid interrupt flags in GPIO_IF register.

GPIO_IntSet

```
void GPIO_IntSet (uint32_t flags)
```

Set one or more pending GPIO interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	GPIO interrupt sources to set to pending.

GPIO_Lock

```
void GPIO_Lock (void )
```

Lock the GPIO configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

GPIO_PinInGet

```
unsigned int GPIO_PinInGet (GPIO_Port_TypeDef port, unsigned int pin)
```

Read the pad value for a single pin in a GPIO port.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
unsigned int	[in]	pin	The pin number to read.

Returns

- The pin value, 0 or 1.

GPIO_PinModeGet

```
GPIO_Mode_TypeDef GPIO_PinModeGet (GPIO_Port_TypeDef port, unsigned int pin)
```

Get the mode for a GPIO pin.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
unsigned int	[in]	pin	The pin number in the port.

Returns

- The pin mode.

GPIO_PinModeSet

```
void GPIO_PinModeSet (GPIO_Port_TypeDef port, unsigned int pin, GPIO_Mode_TypeDef mode, unsigned int out)
```

Set the mode for a GPIO pin.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
unsigned int	[in]	pin	The pin number in the port.
GPIO_Mode_TypeDef	[in]	mode	The desired pin mode.
unsigned int	[in]	out	A value to set for the pin in the DOUT register. The DOUT setting is important for some input mode configurations to determine the pull-up/down direction.

GPIO_PinOutClear

```
void GPIO_PinOutClear (GPIO_Port_TypeDef port, unsigned int pin)
```

Set a single pin in GPIO data out port register to 0.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
unsigned int	[in]	pin	The pin to set.

Note

- To ensure that the setting takes effect on the output pad, the pin must be configured properly. If not, it will take effect whenever the pin has been properly configured.

GPIO_PinOutGet

```
unsigned int GPIO_PinOutGet (GPIO_Port_TypeDef port, unsigned int pin)
```

Get current setting for a pin in a GPIO port data out register.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
unsigned int	[in]	pin	The pin to get setting for.

Returns

- The DOUT setting for the requested pin, 0 or 1.

GPIO_PinOutSet

```
void GPIO_PinOutSet (GPIO_Port_TypeDef port, unsigned int pin)
```

Set a single pin in GPIO data out register to 1.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
unsigned int	[in]	pin	The pin to set.

Note

- To ensure that the setting takes effect on the output pad, the pin must be configured properly. If not, it will take effect whenever the pin has been properly configured.

GPIO_PinOutToggle

```
void GPIO_PinOutToggle (GPIO_Port_TypeDef port, unsigned int pin)
```

Toggle a single pin in GPIO port data out register.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
unsigned int	[in]	pin	The pin to toggle.

Note

- To ensure that the setting takes effect on the output pad, the pin must be configured properly. If not, it will take effect whenever the pin has been properly configured.

GPIO_PortInGet

```
uint32_t GPIO_PortInGet (GPIO_Port_TypeDef port)
```

Read the pad values for GPIO port.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.

Returns

- The pad values for the GPIO port.

GPIO_PortOutClear

```
void GPIO_PortOutClear (GPIO_Port_TypeDef port, uint32_t pins)
```

Set bits in DOUT register for a port to 0.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
uint32_t	[in]	pins	Bit mask for bits to clear in DOUT register.

Note

- To ensure that the setting takes effect on the output pad, the pin must be configured properly. If not, it will take effect whenever the pin has been properly configured.

GPIO_PortOutGet

```
uint32_t GPIO_PortOutGet (GPIO\_Port\_TypeDef port)
```

Get the current setting for a GPIO port data out register.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.

Returns

- The data out setting for the requested port.

GPIO_PortOutSet

```
void GPIO_PortOutSet (GPIO\_Port\_TypeDef port, uint32_t pins)
```

Set bits GPIO data out register to 1.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
uint32_t	[in]	pins	Bit mask for bits to set to 1 in DOUT register.

Note

- To ensure that the setting takes effect on the respective output pads, the pins must be configured properly. If not, it will take effect whenever the pin has been properly configured.

GPIO_PortOutSetVal

```
void GPIO_PortOutSetVal (GPIO\_Port\_TypeDef port, uint32_t val, uint32_t mask)
```

Set GPIO port data out register.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
uint32_t	[in]	val	Value to write to port data out register.
uint32_t	[in]	mask	Mask indicating which bits to modify.

Note

- To ensure that the setting takes effect on the respective output pads, the pins must be configured properly. If not, it will take effect whenever the pin has been properly configured.

GPIO_PortOutToggle

```
void GPIO_PortOutToggle (GPIO_Port_TypeDef port, uint32_t pins)
```

Toggle pins in GPIO port data out register.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to access.
uint32_t	[in]	pins	Bit mask with pins to toggle.

Note

- To ensure that the setting takes effect on the output pad, the pin must be configured properly. If not, it will take effect whenever the pin has been properly configured.

GPIO_SlewrateSet

```
void GPIO_SlewrateSet (GPIO_Port_TypeDef port, uint32_t slewrate, uint32_t slewrateAlt)
```

Set slewrate for pins on a GPIO port.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The GPIO port to configure.
uint32_t	[in]	slewrate	The slewrate to configure for pins on this GPIO port.
uint32_t	[in]	slewrateAlt	The slewrate to configure for pins using alternate modes on this GPIO port.

GPIO_Unlock

```
void GPIO_Unlock (void )
```

Unlock the GPIO configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

I2C - Inter-Integrated Circuit

I2C - Inter-Integrated Circuit

Inter-integrated Circuit (I2C) Peripheral API.

This module contains functions to control the I2C peripheral of Silicon Labs 32-bit MCUs and SoCs. The I2C interface allows communication on I2C buses with the lowest energy consumption possible.

Modules

[I2C_Init_TypeDef](#)

[I2C_TransferSeq_TypeDef](#)

Enumerations

```
enum    I2C_ClockHLR_TypeDef {
        i2cClockHLRStandard = _I2C_CTRL_CLHR_STANDARD
        i2cClockHLRAsymetric = _I2C_CTRL_CLHR_ASYMMETRIC
        i2cClockHLRFast = _I2C_CTRL_CLHR_FAST
    }
    Clock low to high ratio settings.

enum    I2C_TransferReturn_TypeDef {
        i2cTransferInProgress = 1
        i2cTransferDone = 0
        i2cTransferNack = -1
        i2cTransferBusErr = -2
        i2cTransferArbLost = -3
        i2cTransferUsageFault = -4
        i2cTransferSwFault = -5
    }
    Return codes for single Controller mode transfer function.
```

Functions

```
uint32_t I2C_BusFreqGet(I2C_TypeDef *i2c)
    Get the current configured I2C bus frequency.

void I2C_BusFreqSet(I2C_TypeDef *i2c, uint32_t freqRef, uint32_t freqScl, I2C_ClockHLR_TypeDef
i2cMode)
    Set the I2C bus frequency.

void I2C_Enable(I2C_TypeDef *i2c, bool enable)
    Enable/disable I2C.

void I2C_Init(I2C_TypeDef *i2c, const I2C_Init_TypeDef *init)
    Initialize I2C.

void I2C_Reset(I2C_TypeDef *i2c)
    Reset I2C to the same state that it was in after a hardware reset.

I2C_TransferReturn_TypeDef I2C_Transfer(I2C_TypeDef *i2c)
    Continue an initiated I2C transfer (single master mode only).

I2C_TransferReturn_TypeDef I2C_TransferInit(I2C_TypeDef *i2c, I2C_TransferSeq_TypeDef *seq)
    Prepare and start an I2C transfer (single master mode only).
```

void	I2C_IntClear (I2C_TypeDef *i2c, uint32_t flags) Clear one or more pending I2C interrupts.
void	I2C_IntDisable (I2C_TypeDef *i2c, uint32_t flags) Disable one or more I2C interrupts.
void	I2C_IntEnable (I2C_TypeDef *i2c, uint32_t flags) Enable one or more I2C interrupts.
uint32_t	I2C_IntGet (I2C_TypeDef *i2c) Get pending I2C interrupt flags.
uint32_t	I2C_IntGetEnabled (I2C_TypeDef *i2c) Get enabled and pending I2C interrupt flags.
void	I2C_IntSet (I2C_TypeDef *i2c, uint32_t flags) Set one or more pending I2C interrupts from SW.
uint8_t	I2C_SlaveAddressGet (I2C_TypeDef *i2c) Get Target address used for I2C peripheral (when operating in Target mode).
void	I2C_SlaveAddressSet (I2C_TypeDef *i2c, uint8_t addr) Set Target address to use for I2C peripheral (when operating in Target mode).
uint8_t	I2C_SlaveAddressMaskGet (I2C_TypeDef *i2c) Get Target address mask used for I2C peripheral (when operating in Target mode).
void	I2C_SlaveAddressMaskSet (I2C_TypeDef *i2c, uint8_t mask) Set Target address mask used for I2C peripheral (when operating in Target mode).

Macros

#define	I2C_FREQ_STANDARD_MAX 100000 Standard mode max frequency assuming using 4:4 ratio for Nlow:Nhigh.
#define	I2C_FREQ_FAST_MAX 392157 Fast mode max frequency assuming using 6:3 ratio for Nlow:Nhigh.
#define	I2C_FREQ_FASTPLUS_MAX 987167 Fast mode+ max frequency assuming using 11:6 ratio for Nlow:Nhigh.
#define	I2C_FLAG_WRITE 0x0001 Indicate plain write sequence: S+ADDR(W)+DATA0+P.
#define	I2C_FLAG_READ 0x0002 Indicate plain read sequence: S+ADDR(R)+DATA0+P.
#define	I2C_FLAG_WRITE_READ 0x0004 Indicate combined write/read sequence: S+ADDR(W)+DATA0+Sr+ADDR(R)+DATA1+P.
#define	I2C_FLAG_WRITE_WRITE 0x0008 Indicate write sequence using two buffers: S+ADDR(W)+DATA0+DATA1+P.
#define	I2C_FLAG_10BIT_ADDR 0x0010 Use 10 bit address.
#define	I2C_INIT_DEFAULT undefined Suggested default configuration for I2C initialization structure.

Enumeration Documentation

I2C_ClockHLR_TypeDef

I2C_ClockHLR_TypeDef

Clock low to high ratio settings.

Enumerator	
i2cClockHLRStandard	Ratio is 4:4.
i2cClockHLRAsymmetric	Ratio is 6:3.
i2cClockHLRFast	Ratio is 11:3.

I2C_TransferReturn_TypeDef

I2C_TransferReturn_TypeDef

Return codes for single Controller mode transfer function.

Enumerator	
i2cTransferInProgress	Transfer in progress.
i2cTransferDone	Transfer completed successfully.
i2cTransferNack	NACK received during transfer.
i2cTransferBusErr	Bus error during transfer (misplaced START/STOP).
i2cTransferArbLost	Arbitration lost during transfer.
i2cTransferUsageFault	Usage fault.
i2cTransferSwFault	SW fault.

Function Documentation

I2C_BusFreqGet

```
uint32_t I2C_BusFreqGet (I2C_TypeDef * i2c)
```

Get the current configured I2C bus frequency.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	A pointer to the I2C peripheral register block.

This frequency is only relevant when acting as master.

Note

- The actual frequency is a real number, this function returns a rounded down (truncated) integer value.

Returns

- The current I2C frequency in Hz.

I2C_BusFreqSet

```
void I2C_BusFreqSet (I2C_TypeDef * i2c, uint32_t freqRef, uint32_t freqScl, I2C_ClockHLR_TypeDef i2cMode)
```

Set the I2C bus frequency.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	A pointer to the I2C peripheral register block.
uint32_t	[in]	freqRef	An I2C reference clock frequency in Hz that will be used. If set to 0, HFPERCLK / HFPERCCLK clock is used. Setting it to a higher than actual configured value has the consequence of reducing the real I2C frequency.
uint32_t	[in]	freqScl	A bus frequency to set (bus speed may be lower due to integer prescaling). Safe (according to the I2C specification) maximum frequencies for standard fast and fast+ modes are available using I2C_FREQ_ defines. (Using I2C_FREQ_ defines requires corresponding setting of type .) The slowest slave device on a bus must always be considered.
I2C_ClockHLR_TypeDef	[in]	i2cMode	A clock low-to-high ratio type to use. If not using i2cClockHLRStandard, make sure all devices on the bus support the specified mode. Using a non-standard ratio is useful to achieve a higher bus clock in fast and fast+ modes.

The bus frequency is only relevant when acting as master. The bus frequency should not be set higher than the maximum frequency accepted by the slowest device on the bus.

Notice that, due to asymmetric requirements on low and high I2C clock cycles in the I2C specification, the maximum frequency allowed to comply with the specification may be somewhat lower than expected.

See the reference manual, details on I2C clock generation, for maximum allowed theoretical frequencies for different modes.

I2C_Enable

```
void I2C_Enable (I2C_TypeDef * i2c, bool enable)
```

Enable/disable I2C.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	A pointer to the I2C peripheral register block.
bool	[in]	enable	True to enable counting, false to disable.

Note

- After enabling the I2C (from being disabled), the I2C is in BUSY state.

I2C_Init

```
void I2C_Init (I2C_TypeDef * i2c, const I2C_Init_TypeDef * init)
```

Initialize I2C.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	A pointer to the I2C peripheral register block.
const I2C_Init_TypeDef *	[in]	init	A pointer to the I2C initialization structure.

I2C_Reset

```
void I2C_Reset (I2C_TypeDef * i2c)
```

Reset I2C to the same state that it was in after a hardware reset.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	A pointer to the I2C peripheral register block.

Note

- The ROUTE register is NOT reset by this function to allow for centralized setup of this feature.

I2C_Transfer

```
I2C_TransferReturn_TypeDef I2C_Transfer (I2C_TypeDef * i2c)
```

Continue an initiated I2C transfer (single master mode only).

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	A pointer to the I2C peripheral register block.

This function is used repeatedly after a [I2C_TransferInit\(\)](#) to complete a transfer. It may be used in polled mode as the below example shows:

```
I2C_TransferReturn_TypeDef ret;

// Do a polled transfer
ret = I2C_TransferInit(I2C0, seq);
while (ret == i2cTransferInProgress)
{
    ret = I2C_Transfer(I2C0);
}
```

It may also be used in interrupt driven mode, where this function is invoked from the interrupt handler. Notice that, if used in interrupt mode, NVIC interrupts must be configured and enabled for the I2C bus used. I2C peripheral specific interrupts are managed by this software.

Note

- Only single master mode is supported.

Returns

- Returns status for an ongoing transfer.
 - [i2cTransferInProgress](#) - indicates that transfer not finished.
 - [i2cTransferDone](#) - transfer completed successfully.
 - otherwise some sort of error has occurred.

I2C_TransferInit

```
I2C_TransferReturn_TypeDef I2C_TransferInit (I2C_TypeDef * i2c, I2C_TransferSeq_TypeDef * seq)
```

Prepare and start an I2C transfer (single master mode only).

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	A pointer to the I2C peripheral register block.
I2C_TransferSeq_TypeDef *	[in]	seq	A pointer to the sequence structure defining the I2C transfer to take place. The referenced structure must exist until the transfer has fully completed.

This function must be invoked to start an I2C transfer sequence. To complete the transfer, [I2C_Transfer\(\)](#) must be used either in polled mode or by adding a small driver wrapper using interrupts.

Note

- Only single master mode is supported.

Returns

- Returns the status for an ongoing transfer:
 - [i2cTransferInProgress](#) - indicates that the transfer is not finished.
 - Otherwise, an error has occurred.

I2C_IntClear

```
void I2C_IntClear (I2C_TypeDef * i2c, uint32_t flags)
```

Clear one or more pending I2C interrupts.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.
uint32_t	[in]	flags	Pending I2C interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for the I2C module (I2C_IF_nnn).

I2C_IntDisable

```
void I2C_IntDisable (I2C_TypeDef * i2c, uint32_t flags)
```

Disable one or more I2C interrupts.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.
uint32_t	[in]	flags	I2C interrupt sources to disable. Use a bitwise logic OR combination of valid interrupt flags for the I2C module (I2C_IF_nnn).

I2C_IntEnable

```
void I2C_IntEnable (I2C_TypeDef * i2c, uint32_t flags)
```

Enable one or more I2C interrupts.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.
uint32_t	[in]	flags	I2C interrupt sources to enable. Use a bitwise logic OR combination of valid interrupt flags for the I2C module (I2C_IF_nnn).

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [I2C_IntClear\(\)](#) prior to enabling the interrupt.

I2C_IntGet

```
uint32_t I2C_IntGet (I2C_TypeDef * i2c)
```

Get pending I2C interrupt flags.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.

Note

- Event bits are not cleared by the use of this function.

Returns

- I2C interrupt sources pending. A bitwise logic OR combination of valid interrupt flags for the I2C module (I2C_IF_nnn).

I2C_IntGetEnabled

```
uint32_t I2C_IntGetEnabled (I2C_TypeDef * i2c)
```

Get enabled and pending I2C interrupt flags.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

Pending and enabled I2C interrupt sources Return value is the bitwise AND of

- the enabled interrupt sources in I2Cn_IEN and
- the pending interrupt flags I2Cn_IF

I2C_IntSet

```
void I2C_IntSet (I2C_TypeDef * i2c, uint32_t flags)
```

Set one or more pending I2C interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.
uint32_t	[in]	flags	I2C interrupt sources to set to pending. Use a bitwise logic OR combination of valid interrupt flags for the I2C module (I2C_IF_nnn).

I2C_SlaveAddressGet

```
uint8_t I2C_SlaveAddressGet (I2C_TypeDef * i2c)
```

Get Target address used for I2C peripheral (when operating in Target mode).

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.

For 10-bit addressing mode, the address is split in two bytes, and only the first byte setting is fetched, effectively only controlling the 2 most significant bits of the 10-bit address. Full handling of 10-bit addressing in Target mode requires additional SW handling.

Returns

- I2C Target address in use. The 7 most significant bits define the actual address, the least significant bit is reserved and always returned as 0.

I2C_SlaveAddressSet

```
void I2C_SlaveAddressSet (I2C_TypeDef * i2c, uint8_t addr)
```

Set Target address to use for I2C peripheral (when operating in Target mode).

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.
uint8_t	[in]	addr	I2C Target address to use. The 7 most significant bits define the actual address, the least significant bit is reserved and always set to 0.

For 10-bit addressing mode, the address is split in two bytes, and only the first byte is set, effectively only controlling the 2 most significant bits of the 10-bit address. Full handling of 10-bit addressing in Target mode requires additional SW handling.

I2C_SlaveAddressMaskGet

```
uint8_t I2C_SlaveAddressMaskGet (I2C_TypeDef * i2c)
```

Get Target address mask used for I2C peripheral (when operating in Target mode).

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.

The address mask defines how the comparator works. A bit position with value 0 means that the corresponding Target address bit is ignored during comparison (don't care). A bit position with value 1 means that the corresponding Target address bit must match.

For 10-bit addressing mode, the address is split in two bytes, and only the mask for the first address byte is fetched, effectively only controlling the 2 most significant bits of the 10-bit address.

Returns

- I2C Target address mask in use. The 7 most significant bits define the actual address mask, the least significant bit is reserved and always returned as 0.

I2C_SlaveAddressMaskSet

```
void I2C_SlaveAddressMaskSet (I2C_TypeDef * i2c, uint8_t mask)
```

Set Target address mask used for I2C peripheral (when operating in Target mode).

Parameters

Type	Direction	Argument Name	Description
I2C_TypeDef *	[in]	i2c	Pointer to I2C peripheral register block.
uint8_t	[in]	mask	I2C Target address mask to use. The 7 most significant bits define the actual address mask, the least significant bit is reserved and should be 0.

The address mask defines how the comparator works. A bit position with value 0 means that the corresponding Target address bit is ignored during comparison (don't care). A bit position with value 1 means that the corresponding Target address bit must match.

For 10-bit addressing mode, the address is split in two bytes, and only the mask for the first address byte is set, effectively only controlling the 2 most significant bits of the 10-bit address.

Macro Definition Documentation

I2C_FREQ_STANDARD_MAX

```
#define I2C_FREQ_STANDARD_MAX
```

Value:

```
100000
```

Standard mode max frequency assuming using 4:4 ratio for Nlow:Nhigh.

From I2C specification: Min Tlow = 4.7us, min Thigh = 4.0us, max Trise=1.0us, max Tfall=0.3us. Since ratio is 4:4, have to use worst case value of Tlow or Thigh as base.

$$1/(T_{low} + T_{high} + 1\mu s + 0.3\mu s) = 1/(4.7 + 4.7 + 1.3)\mu s = 93458\text{Hz}$$
 Note

- Due to chip characteristics, max value is somewhat reduced.

I2C_FREQ_FAST_MAX

```
#define I2C_FREQ_FAST_MAX
```

Value:

392157

Fast mode max frequency assuming using 6:3 ratio for Nlow:Nhigh.

From I2C specification: Min Tlow = 1.3us, min Thigh = 0.6us, max Trise=0.3us, max Tfall=0.3us. Since ratio is 6:3, have to use worst case value of Tlow or 2xThigh as base.

$$1/(T_{low} + T_{high} + 0.3\mu s + 0.3\mu s) = 1/(1.3 + 0.65 + 0.6)\mu s = 392157\text{Hz}$$

I2C_FREQ_FASTPLUS_MAX

```
#define I2C_FREQ_FASTPLUS_MAX
```

Value:

987167

Fast mode+ max frequency assuming using 11:6 ratio for Nlow:Nhigh.

From I2C specification: Min Tlow = 0.5us, min Thigh = 0.26us, max Trise=0.12us, max Tfall=0.12us. Since ratio is 11:6, have to use worst case value of Tlow or (11/6)xThigh as base.

$$1/(T_{low} + T_{high} + 0.12\mu s + 0.12\mu s) = 1/(0.5 + 0.273 + 0.24)\mu s = 987167\text{Hz}$$

I2C_FLAG_WRITE

```
#define I2C_FLAG_WRITE
```

Value:

0x0001

Indicate plain write sequence: S+ADDR(W)+DATA0+P.

- S - Start
- ADDR(W) - address with W/R bit cleared
- DATA0 - Data taken from buffer with index 0
- P - Stop

I2C_FLAG_READ

```
#define I2C_FLAG_READ
```

Value:

0x0002

Indicate plain read sequence: S+ADDR(R)+DATA0+P.

- S - Start
- ADDR(R) - Address with W/R bit set
- DATA0 - Data read into buffer with index 0
- P - Stop

I2C_FLAG_WRITE_READ

#define I2C_FLAG_WRITE_READ

Value:

0x0004

Indicate combined write/read sequence: S+ADDR(W)+DATA0+Sr+ADDR(R)+DATA1+P.

- S - Start
- Sr - Repeated start
- ADDR(W) - Address with W/R bit cleared
- ADDR(R) - Address with W/R bit set
- DATAn - Data written from/read into buffer with index n
- P - Stop

I2C_FLAG_WRITE_WRITE

#define I2C_FLAG_WRITE_WRITE

Value:

0x0008

Indicate write sequence using two buffers: S+ADDR(W)+DATA0+DATA1+P.

- S - Start
- ADDR(W) - Address with W/R bit cleared
- DATAn - Data written from buffer with index n
- P - Stop

I2C_FLAG_10BIT_ADDR

#define I2C_FLAG_10BIT_ADDR

Value:

0x0010

Use 10 bit address.

I2C_INIT_DEFAULT

```
#define I2C_INIT_DEFAULT
```

Value:

```
0  {
0  true,          /* Enable when initialization done. */ \
0  true,          /* Set to Controller mode. */         \
0  0,             /* Use currently configured reference clock. */ \
0  I2C_FREQ_STANDARD_MAX, /* Set to standard rate assuring being */ \
0  /*             within I2C specification. */ \
0  i2cClockHLRStandard /* Set to use 4:4 low/high duty cycle. */ \
0  }
```

Suggested default configuration for I2C initialization structure.

I2C_Init_TypeDef

I2C initialization structure.

Public Attributes

bool	enable	Enable I2C peripheral when initialization completed.
bool	master	Set to Controller (true) or Target (false) mode.
uint32_t	refFreq	I2C reference clock assumed when configuring bus frequency setup.
uint32_t	freq	(Max) I2C bus frequency to use.
I2C_ClockHLR_TypeDef	clhr	Clock low/high ratio control.

Public Attribute Documentation

enable

```
bool I2C_Init_TypeDef::enable
```

Enable I2C peripheral when initialization completed.

master

```
bool I2C_Init_TypeDef::master
```

Set to Controller (true) or Target (false) mode.

refFreq

```
uint32_t I2C_Init_TypeDef::refFreq
```

I2C reference clock assumed when configuring bus frequency setup.

Set it to 0 if currently configured reference clock will be used This parameter is only applicable if operating in Controller mode.

freq

```
uint32_t I2C_Init_TypeDef::freq
```

(Max) I2C bus frequency to use.

This parameter is only applicable if operating in Controller mode.

clhr

```
I2C_ClockHLR_TypeDef I2C_Init_TypeDef::clhr
```

Clock low/high ratio control.

I2C_TransferSeq_TypeDef

Master mode transfer message structure used to define a complete I2C transfer sequence (from start to stop).

The structure allows for defining the following types of sequences (refer to defines for sequence details):

- [I2C_FLAG_READ](#) - Data read into buf[0].data
- [I2C_FLAG_WRITE](#) - Data written from buf[0].data
- [I2C_FLAG_WRITE_READ](#) - Data written from buf[0].data and read into buf[1].data
- [I2C_FLAG_WRITE_WRITE](#) - Data written from buf[0].data and buf[1].data

Public Attributes

uint16_t	addr	Address to use after (repeated) start.
uint16_t	flags	Flags defining sequence type and details, see I2C_FLAG_ defines.
uint8_t *	data	Buffer used for data to transmit/receive, must be <code>len</code> long.
uint16_t	len	Number of bytes in <code>data</code> to send or receive.
struct I2C_TransferSeq_TypeDef::@0	buf	Buffers used to hold data to send from or receive into, depending on sequence type.

Public Attribute Documentation

addr

```
uint16_t I2C_TransferSeq_TypeDef::addr
```

Address to use after (repeated) start.

Layout details, A = Address bit, X = don't care bit (set to 0):

- 7 bit address - Use format AAAA AAAX
- 10 bit address - Use format XXXX XAAX AAAA AAAA

flags

```
uint16_t I2C_TransferSeq_TypeDef::flags
```

Flags defining sequence type and details, see I2C_FLAG_ defines.

data

```
uint8_t* I2C_TransferSeq_TypeDef::data
```

Buffer used for data to transmit/receive, must be `len` long.

len

```
uint16_t I2C_TransferSeq_TypeDef::len
```

Number of bytes in `data` to send or receive.

Notice that when receiving data to this buffer, at least 1 byte must be received. Setting `len` to 0 in the receive case is considered a usage fault. Transmitting 0 bytes is legal, in which case only the address is transmitted after the start condition.

buf

```
struct I2C_TransferSeq_TypeDef::@0 I2C_TransferSeq_TypeDef::buf[2]
```

Buffers used to hold data to send from or receive into, depending on sequence type.

LDMA - Linked DMA

LDMA - Linked DMA

Linked Direct Memory Access (LDMA) Peripheral API.

LDMA API functions provide full support for the LDMA peripheral.

LDMA supports these DMA transfer types:

- Memory to memory.
- Memory to peripheral.
- Peripheral to memory.
- Peripheral to peripheral.
- Constant value to memory.

LDMA supports linked lists of DMA descriptors allowing:

- Circular and ping-pong buffer transfers.
- Scatter-gather transfers.
- Looped transfers.

LDMA has some advanced features:

- Intra-channel synchronization (SYNC), allowing hardware events to pause and restart a DMA sequence.
- Immediate-write (WRI), allowing DMA to write a constant anywhere in the memory map.
- Complex flow control allowing if-else constructs.

Basic understanding of LDMA controller is assumed. Please refer to the reference manual for further details. The LDMA examples described in the reference manual are particularly helpful in understanding LDMA operations.

In order to use the DMA controller, the initialization function `LDMA_Init()` must have been executed once (normally during system initialization).

DMA transfers are initiated by a call to `LDMA_StartTransfer()`, transfer properties are controlled by the contents of `LDMA_TransferCfg_t` and `LDMA_Descriptor_t` structure parameters. The `LDMA_Descriptor_t` structure parameter may be a pointer to an array of descriptors, descriptors in array should be linked together as needed.

Transfer and descriptor initialization macros are provided for the most common transfer types. Due to the flexibility of LDMA peripheral, only a small subset of all possible initializer macros are provided, users should create new ones when needed.

Examples of LDMA usage:

A simple memory to memory transfer:

```
/* A single transfer of 4 half words. */
static const LDMA_TransferCfg_t transferCfg = LDMA_TRANSFER_CFG_MEMORY();
static const LDMA_Descriptor_t desc = LDMA_DESCRIPTOR_SINGLE_M2M_HALF(src, dst, 4);

void ldmaTransfer(void)
{
    /* Initialize the LDMA with default values. */
    LDMA_Init_t init = LDMA_INIT_DEFAULT;
    LDMA_Init(&init);

    /* Start the memory transfer */
    LDMA_StartTransfer(0, &transferCfg, &desc);
}
```

A linked list of three memory to memory transfers:

```
/* A transfer consisting of 3 descriptors linked together and each descriptor
 * transfers 4 words from the source to the destination. */
static const LDMA_TransferCfg_t transferCfg1 = LDMA_TRANSFER_CFG_MEMORY();
static const LDMA_Descriptor_t descList[] =
{
    LDMA_DESCRIPTOR_LINKREL_M2M_WORD(src, dst, 4, 1),
    LDMA_DESCRIPTOR_LINKREL_M2M_WORD(src + 4, dst + 4, 4, 1),
    LDMA_DESCRIPTOR_SINGLE_M2M_WORD(src + 8, dst + 8, 4)
};

void ldmaLinkedTransfer(void)
{
    /* Initialize the LDMA with default values. */
    LDMA_Init_t init = LDMA_INIT_DEFAULT;
    LDMA_Init(&init);

    /* Start the linked memory transfer */
    LDMA_StartTransfer(0, &transferCfg1, &descList[0]);
}
```

DMA from serial port peripheral to memory:

```

#if !defined(USART1) && defined(USART0)
/* The LDMA transfer should be triggered by the USART0 RX data available signal. */
static const LDMA_TransferCfg_t usart0RxTransfer =
    LDMA_TRANSFER_CFG_PERIPHERAL(ldmaPeripheralSignal_USART0_RXDATAV);

/* Transfer 4 bytes from the USART0 RX FIFO to memory. */
static const LDMA_Descriptor_t usart0RxDesc =
    LDMA_DESCRIPTOR_SINGLE_P2M_BYTE(&USART0->RXDATA, // Peripheral address
                                     dst,             // Destination (SRAM)
                                     4);              // Number of byte transfers

void ldmaPeripheralTransfer(void)
{
    /* Initialize the LDMA with default values. */
    LDMA_Init_t init = LDMA_INIT_DEFAULT;
    LDMA_Init(&init);

    /* Start the peripheral transfer which is triggered by the USART0
     * peripheral RXDATAV signal. */
    LDMA_StartTransfer(0, &usart0RxTransfer, &usart0RxDesc);
}
#elif defined(USART1)
/* The LDMA transfer should be triggered by the USART1 RX data available signal. */
static const LDMA_TransferCfg_t usart1RxTransfer =
    LDMA_TRANSFER_CFG_PERIPHERAL(ldmaPeripheralSignal_USART1_RXDATAV);

/* Transfer 4 bytes from the USART1 RX FIFO to memory. */
static const LDMA_Descriptor_t usart1RxDesc =
    LDMA_DESCRIPTOR_SINGLE_P2M_BYTE(&USART1->RXDATA, // Peripheral address
                                     dst,             // Destination (SRAM)
                                     4);              // Number of byte transfers

void ldmaPeripheralTransfer(void)
{
    /* Initialize the LDMA with default values. */
    LDMA_Init_t init = LDMA_INIT_DEFAULT;
    LDMA_Init(&init);

    /* Start the peripheral transfer which is triggered by the USART1
     * peripheral RXDATAV signal. */
    LDMA_StartTransfer(0, &usart1RxTransfer, &usart1RxDesc);
}
#elif defined(EUSART0)
/* The LDMA transfer should be triggered by the EUSART0 RX data available signal. */
static const LDMA_TransferCfg_t eusart0RxTransfer =
    LDMA_TRANSFER_CFG_PERIPHERAL(ldmaPeripheralSignal_EUSART0_RXFL);

/* Transfer 4 bytes from the EUSART0 RX FIFO to memory. */
static const LDMA_Descriptor_t eusart0RxDesc =
    LDMA_DESCRIPTOR_SINGLE_P2M_BYTE(&EUSART0->RXDATA, // Peripheral address
                                     dst,             // Destination (SRAM)
                                     4);              // Number of byte transfers

void ldmaPeripheralTransfer(void)
{
    /* Initialize the LDMA with default values. */
    LDMA_Init_t init = LDMA_INIT_DEFAULT;
    LDMA_Init(&init);

    /* Start the peripheral transfer which is triggered by the EUSART0
     * peripheral RXFL signal. */
    LDMA_StartTransfer(0, &eusart0RxTransfer, &eusart0RxDesc);
}

#endif

```

Ping-pong DMA from serial port peripheral to memory:

```

#if !defined(USART1) && defined(USART0)
/* Two buffers used in the ping-pong transfer from USART0. */
static volatile uint8_t buffer0[5];
static volatile uint8_t buffer1[5];

/* The LDMA transfer should be triggered by the USART0 RX data available signal. */
static const LDMA_TransferCfg_t usart0Transfer =
    LDMA_TRANSFER_CFG_PERIPHERAL(ldmaPeripheralSignal_USART0_RXDATAV);

/* Both descriptors transfer 5 bytes from the USART0 rx data register into
 * one of the buffers. Note that the first descriptor uses a relative address
 * of 1 to link to the next descriptor, while the last descriptor uses a
 * relative address of -1 to link to the first descriptor. */
static const LDMA_Descriptor_t rxLoop[] =
{
    LDMA_DESCRIPTOR_LINKREL_P2M_BYTE(&USART0->RXDATA, buffer0, 5, 1),
    LDMA_DESCRIPTOR_LINKREL_P2M_BYTE(&USART0->RXDATA, buffer1, 5, -1)
};

void ldmaPingPongTransfer(void)
{
    /* Initialize the LDMA with default values. */
    LDMA_Init_t init = LDMA_INIT_DEFAULT;
    LDMA_Init(&init);

    /* Start the peripheral transfer which is triggered by the USART0
     * peripheral RXDATAV signal. */
    LDMA_StartTransfer(0, &usart0Transfer, &rxLoop[0]);
}

#elif defined(USART1)
/* Two buffers used in the ping-pong transfer from USART1. */
static volatile uint8_t buffer0[5];
static volatile uint8_t buffer1[5];

/* The LDMA transfer should be triggered by the USART1 RX data available signal. */
static const LDMA_TransferCfg_t usart1Transfer =
    LDMA_TRANSFER_CFG_PERIPHERAL(ldmaPeripheralSignal_USART1_RXDATAV);

/* Both descriptors transfer 5 bytes from the USART1 rx data register into
 * one of the buffers. Note that the first descriptor uses a relative address
 * of 1 to link to the next descriptor, while the last descriptor uses a
 * relative address of -1 to link to the first descriptor. */
static const LDMA_Descriptor_t rxLoop[] =
{
    LDMA_DESCRIPTOR_LINKREL_P2M_BYTE(&USART1->RXDATA, buffer0, 5, 1),
    LDMA_DESCRIPTOR_LINKREL_P2M_BYTE(&USART1->RXDATA, buffer1, 5, -1)
};

void ldmaPingPongTransfer(void)
{
    /* Initialize the LDMA with default values. */
    LDMA_Init_t init = LDMA_INIT_DEFAULT;
    LDMA_Init(&init);

    /* Start the peripheral transfer which is triggered by the USART1
     * peripheral RXDATAV signal. */
    LDMA_StartTransfer(0, &usart1Transfer, &rxLoop[0]);
}

#elif defined(EUSART0)
/* Two buffers used in the ping-pong transfer from EUSART0. */
static volatile uint8_t buffer0[5];
static volatile uint8_t buffer1[5];

/* The LDMA transfer should be triggered by the EUSART0 RX data available signal. */
static const LDMA_TransferCfg_t eusart0Transfer =
    LDMA_TRANSFER_CFG_PERIPHERAL(ldmaPeripheralSignal_EUSART0_RXFL);

```

```

/* Both descriptors transfer 5 bytes from the EUSART0 rx data register into
 * one of the buffers. Note that the first descriptor uses a relative address
 * of 1 to link to the next descriptor, while the last descriptor uses a
 * relative address of -1 to link to the first descriptor. */
static const LDMA_Descriptor_t rxLoop[]={LDMA_DESCRIPTOR_LINKREL_P2M_BYTE(&EUSART0->RXDATA,
buffer0,5,1),LDMA_DESCRIPTOR_LINKREL_P2M_BYTE(&EUSART0->RXDATA, buffer1,5,-1)};

void ldmaPingPongTransfer(void){/* Initialize the LDMA with default values. */
    LDMA_Init_t init = LDMA_INIT_DEFAULT;LDMA_Init(&init);/* Start the peripheral transfer which is triggered by the
EUSART0
    * peripheral RXFL signal. */LDMA_StartTransfer(0,&eusart0Transfer,&rxLoop[0]);}
#endif

```

Note

- LDMA module does not implement LDMA interrupt handler. A template for an LDMA IRQ handler is included here as an example.

```

/* Template for an LDMA IRQ handler. */
void LDMA_IRQHandler(void)
{
    uint32_t ch;
    /* Get all pending and enabled interrupts. */
    uint32_t pending = LDMA_IntGetEnabled();

    /* Loop here on an LDMA error to enable debugging. */
    while (pending & LDMA_IF_ERROR) {
    }

    /* Iterate over all LDMA channels. */
    for (ch = 0; ch < DMA_CHAN_COUNT; ch++) {
        uint32_t mask = 0x1 << ch;
        if (pending & mask) {
            /* Clear interrupt flag. */

#ifdef LDMA_HAS_SET_CLEAR
            LDMA->IF_CLR = mask;
#else
            LDMA->IFC = mask;
#endif

            /* Do more stuff here, execute callbacks etc. */
        }
    }
}

```

Modules

[LDMA_Descriptor_t](#)

[LDMA_Init_t](#)

[LDMA_TransferCfg_t](#)

Enumerations

```
enum LDMA_CtrlBlockSize_t {
    IdmaCtrlBlockSizeUnit1 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT1
    IdmaCtrlBlockSizeUnit2 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT2
    IdmaCtrlBlockSizeUnit3 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT3
    IdmaCtrlBlockSizeUnit4 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT4
    IdmaCtrlBlockSizeUnit6 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT6
    IdmaCtrlBlockSizeUnit8 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT8
    IdmaCtrlBlockSizeUnit16 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT16
    IdmaCtrlBlockSizeUnit32 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT32
    IdmaCtrlBlockSizeUnit64 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT64
    IdmaCtrlBlockSizeUnit128 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT128
    IdmaCtrlBlockSizeUnit256 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT256
    IdmaCtrlBlockSizeUnit512 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT512
    IdmaCtrlBlockSizeUnit1024 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT1024
    IdmaCtrlBlockSizeAll = _LDMA_CH_CTRL_BLOCKSIZE_ALL
}
Controls the number of unit data transfers per arbitration cycle, providing a means to balance DMA channels'
load on the controller.
```

```
enum LDMA_CtrlStructType_t {
    IdmaCtrlStructTypeXfer = _LDMA_CH_CTRL_STRUCTTYPE_TRANSFER
    IdmaCtrlStructTypeSync = _LDMA_CH_CTRL_STRUCTTYPE_SYNCHRONIZE
    IdmaCtrlStructTypeWrite = _LDMA_CH_CTRL_STRUCTTYPE_WRITE
}
DMA structure type.
```

```
enum LDMA_CtrlReqMode_t {
    IdmaCtrlReqModeBlock = _LDMA_CH_CTRL_REQMODE_BLOCK
    IdmaCtrlReqModeAll = _LDMA_CH_CTRL_REQMODE_ALL
}
DMA transfer block or cycle selector.
```

```
enum LDMA_CtrlSrcInc_t {
    IdmaCtrlSrcIncOne = _LDMA_CH_CTRL_SRCINC_ONE
    IdmaCtrlSrcIncTwo = _LDMA_CH_CTRL_SRCINC_TWO
    IdmaCtrlSrcIncFour = _LDMA_CH_CTRL_SRCINC_FOUR
    IdmaCtrlSrcIncNone = _LDMA_CH_CTRL_SRCINC_NONE
}
Source address increment unit size.
```

```
enum LDMA_CtrlSize_t {
    IdmaCtrlSizeByte = _LDMA_CH_CTRL_SIZE_BYTE
    IdmaCtrlSizeHalf = _LDMA_CH_CTRL_SIZE_HALFWORD
    IdmaCtrlSizeWord = _LDMA_CH_CTRL_SIZE_WORD
}
DMA transfer unit size.
```

```
enum LDMA_CtrlDstInc_t {
    IdmaCtrlDstIncOne = _LDMA_CH_CTRL_DSTINC_ONE
    IdmaCtrlDstIncTwo = _LDMA_CH_CTRL_DSTINC_TWO
    IdmaCtrlDstIncFour = _LDMA_CH_CTRL_DSTINC_FOUR
    IdmaCtrlDstIncNone = _LDMA_CH_CTRL_DSTINC_NONE
}
Destination address increment unit size.
```

```
enum LDMA_CtrlSrcAddrMode_t {
    IdmaCtrlSrcAddrModeAbs = _LDMA_CH_CTRL_SRCMODE_ABSOLUTE
    IdmaCtrlSrcAddrModeRel = _LDMA_CH_CTRL_SRCMODE_RELATIVE
}
Source addressing mode.
```

```
enum LDMA_CtrlDstAddrMode_t {
    IdmaCtrlDstAddrModeAbs = _LDMA_CH_CTRL_DSTMODE_ABSOLUTE
    IdmaCtrlDstAddrModeRel = _LDMA_CH_CTRL_DSTMODE_RELATIVE
}
Destination addressing mode.
```

```
enum LDMA_LinkMode_t {  
    ldmaLinkModeAbs = _LDMA_CH_LINK_LINKMODE_ABSOLUTE  
    ldmaLinkModeRel = _LDMA_CH_LINK_LINKMODE_RELATIVE  
}  
DMA link load address mode.  
  
enum LDMA_CfgArbSlots_t {  
    ldmaCfgArbSlotsAs1 = _LDMA_CH_CFG_ARBSLOTS_ONE  
    ldmaCfgArbSlotsAs2 = _LDMA_CH_CFG_ARBSLOTS_TWO  
    ldmaCfgArbSlotsAs4 = _LDMA_CH_CFG_ARBSLOTS_FOUR  
    ldmaCfgArbSlotsAs8 = _LDMA_CH_CFG_ARBSLOTS_EIGHT  
}  
Insert extra arbitration slots to increase channel arbitration priority.  
  
enum LDMA_CfgSrcIncSign_t {  
    ldmaCfgSrcIncSignPos = _LDMA_CH_CFG_SRCINCSIGN_POSITIVE  
    ldmaCfgSrcIncSignNeg = _LDMA_CH_CFG_SRCINCSIGN_NEGATIVE  
}  
Source address increment sign.  
  
enum LDMA_CfgDstIncSign_t {  
    ldmaCfgDstIncSignPos = _LDMA_CH_CFG_DSTINCSIGN_POSITIVE  
    ldmaCfgDstIncSignNeg = _LDMA_CH_CFG_DSTINCSIGN_NEGATIVE  
}  
Destination address increment sign.
```

```
enum LDMA_PeripheralSignal_t {
    IdmaPeripheralSignal_NONE = LDMAXBAR_CH_REQSEL_SOURCESEL_NONE
    IdmaPeripheralSignal_LDMAXBAR_PRSREQ0 =
        LDMAXBAR_CH_REQSEL_SIGSEL_LDMAXBARPRSREQ0 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_LDMAXBAR
    IdmaPeripheralSignal_LDMAXBAR_PRSREQ1 = LDMAXBAR_CH_REQSEL_SIGSEL_LDMAXBARPRSREQ1
        | LDMAXBAR_CH_REQSEL_SOURCESEL_LDMAXBAR
    IdmaPeripheralSignal_TIMER0_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER0CC0 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER0
    IdmaPeripheralSignal_TIMER0_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER0CC1 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER0
    IdmaPeripheralSignal_TIMER0_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER0CC2 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER0
    IdmaPeripheralSignal_TIMER0_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER0UFOF |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER0
    IdmaPeripheralSignal_TIMER1_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER1CC0 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER1
    IdmaPeripheralSignal_TIMER1_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER1CC1 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER1
    IdmaPeripheralSignal_TIMER1_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER1CC2 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER1
    IdmaPeripheralSignal_TIMER1_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER1UFOF |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER1
    IdmaPeripheralSignal_USART0_RXDATAV = LDMAXBAR_CH_REQSEL_SIGSEL_USART0RXDATAV |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
    IdmaPeripheralSignal_USART0_RXDATAVRIGHT =
        LDMAXBAR_CH_REQSEL_SIGSEL_USART0RXDATAVRIGHT |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
    IdmaPeripheralSignal_USART0_TXBL = LDMAXBAR_CH_REQSEL_SIGSEL_USART0TXBL |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
    IdmaPeripheralSignal_USART0_TXBLRIGHT = LDMAXBAR_CH_REQSEL_SIGSEL_USART0TXBLRIGHT |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
    IdmaPeripheralSignal_USART0_TXEMPTY = LDMAXBAR_CH_REQSEL_SIGSEL_USART0TXEMPTY |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
    IdmaPeripheralSignal_USART1_RXDATAV = LDMAXBAR_CH_REQSEL_SIGSEL_USART1RXDATAV |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART1
    IdmaPeripheralSignal_USART1_RXDATAVRIGHT =
        LDMAXBAR_CH_REQSEL_SIGSEL_USART1RXDATAVRIGHT |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART1
    IdmaPeripheralSignal_USART1_TXBL = LDMAXBAR_CH_REQSEL_SIGSEL_USART1TXBL |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART1
    IdmaPeripheralSignal_USART1_TXBLRIGHT = LDMAXBAR_CH_REQSEL_SIGSEL_USART1TXBLRIGHT |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART1
    IdmaPeripheralSignal_USART1_TXEMPTY = LDMAXBAR_CH_REQSEL_SIGSEL_USART1TXEMPTY |
        LDMAXBAR_CH_REQSEL_SOURCESEL_USART1
    IdmaPeripheralSignal_I2C0_RXDATAV = LDMAXBAR_CH_REQSEL_SIGSEL_I2C0RXDATAV |
        LDMAXBAR_CH_REQSEL_SOURCESEL_I2C0
    IdmaPeripheralSignal_I2C0_TXBL = LDMAXBAR_CH_REQSEL_SIGSEL_I2C0TXBL |
        LDMAXBAR_CH_REQSEL_SOURCESEL_I2C0
    IdmaPeripheralSignal_I2C1_RXDATAV = LDMAXBAR_CH_REQSEL_SIGSEL_I2C1RXDATAV |
        LDMAXBAR_CH_REQSEL_SOURCESEL_I2C1
    IdmaPeripheralSignal_I2C1_TXBL = LDMAXBAR_CH_REQSEL_SIGSEL_I2C1TXBL |
        LDMAXBAR_CH_REQSEL_SOURCESEL_I2C1
    IdmaPeripheralSignal_PDM_RXDATAV = LDMAXBAR_CH_REQSEL_SIGSEL_PDMRXDATAV |
        LDMAXBAR_CH_REQSEL_SOURCESEL_PDM
    IdmaPeripheralSignal_IADC0_IADC_SCAN = LDMAXBAR_CH_REQSEL_SIGSEL_IADC0IADC_SCAN |
        LDMAXBAR_CH_REQSEL_SOURCESEL_IADC0
    IdmaPeripheralSignal_IADC0_IADC_SINGLE = LDMAXBAR_CH_REQSEL_SIGSEL_IADC0IADC_SINGLE
        | LDMAXBAR_CH_REQSEL_SOURCESEL_IADC0
    IdmaPeripheralSignal_MSC_WDATA = LDMAXBAR_CH_REQSEL_SIGSEL_MSCWDATA |
        LDMAXBAR_CH_REQSEL_SOURCESEL_MSC
    IdmaPeripheralSignal_TIMER2_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER2CC0 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER2
    IdmaPeripheralSignal_TIMER2_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER2CC1 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER2
    IdmaPeripheralSignal_TIMER2_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER2CC2 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER2
    IdmaPeripheralSignal_TIMER2_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER2UFOF |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER2
    IdmaPeripheralSignal_TIMER3_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER3CC0 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER3
    IdmaPeripheralSignal_TIMER3_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER3CC1 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER3
    IdmaPeripheralSignal_TIMER3_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER3CC2 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER3
    IdmaPeripheralSignal_TIMER3_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER3UFOF |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER3
    IdmaPeripheralSignal_TIMER4_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER4CC0 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER4
    IdmaPeripheralSignal_TIMER4_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER4CC1 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER4
    IdmaPeripheralSignal_TIMER4_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER4CC2 |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER4
    IdmaPeripheralSignal_TIMER4_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER4UFOF |
        LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER4
    IdmaPeripheralSignal_EUART0_RXFL = LDMAXBAR_CH_REQSEL_SIGSEL_EUART0RXFL |
        LDMAXBAR_CH_REQSEL_SOURCESEL_EUART0
}
```

```
ldmaPeripheralSignal_EUART0_TXFL =
LDMAXBAR_CH_REQSEL_SIGSEL_EUART0TXFL
|
LDMAXBAR_CH_REQSEL_SOURCESEL_EUART0
}
Peripherals that can trigger LDMA transfers.
```

Functions

void	LDMA_DeInit (void)	De-initialize the LDMA controller.
void	LDMA_EnableChannelRequest (int ch, bool enable)	Enable or disable an LDMA channel request.
void	LDMA_Init (const LDMA_Init_t *init)	Initialize the LDMA controller.
void	LDMA_StartTransfer (int ch, const LDMA_TransferCfg_t *transfer, const LDMA_Descriptor_t *descriptor)	Start a DMA transfer.
void	LDMA_StopTransfer (int ch)	Stop a DMA transfer.
bool	LDMA_TransferDone (int ch)	Check if a DMA transfer has completed.
uint32_t	LDMA_TransferRemainingCount (int ch)	Get the number of items remaining in a transfer.
bool	LDMA_ChannelEnabled (int ch)	Check if a certain channel is enabled.
void	LDMA_IntClear (uint32_t flags)	Clear one or more pending LDMA interrupts.
void	LDMA_IntDisable (uint32_t flags)	Disable one or more LDMA interrupts.
void	LDMA_IntEnable (uint32_t flags)	Enable one or more LDMA interrupts.
uint32_t	LDMA_IntGet (void)	Get pending LDMA interrupt flags.
uint32_t	LDMA_IntGetEnabled (void)	Get enabled and pending LDMA interrupt flags.
void	LDMA_IntSet (uint32_t flags)	Set one or more pending LDMA interrupts.

Macros

#define	LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD 4	Size in words of a non-extended DMA descriptor.
#define	LDMA_DESCRIPTOR_EXTEND_SIZE_WORD 7	Size in words of an extended DMA descriptor.
#define	LDMA_DESCRIPTOR_MAX_XFER_SIZE (((_LDMA_CH_CTRL_XFERCNT_MASK >> _LDMA_CH_CTRL_XFERCNT_SHIFT) + 1))	Maximum transfer size possible per descriptor.

```
#define LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR (addr)
Converts a LDMA_Descriptor_t pointer to the value suitable to write to the linkAddr field of a
LDMA_Descriptor_t.

#define LDMA_DESCRIPTOR_LINKABS_LINKADDR_TO_ADDR (linkAddr)
Converts a LDMA_Descriptor_t linkAddr field value back to a LDMA_Descriptor_t pointer.

#define LDMA_INIT_DEFAULT undefined
Default DMA initialization structure.

#define LDMA_TRANSFER_CFG_MEMORY ()
Generic DMA transfer configuration for memory to memory transfers.

#define LDMA_TRANSFER_CFG_MEMORY_LOOP (loopCnt)
Generic DMA transfer configuration for looped memory to memory transfers.

#define LDMA_TRANSFER_CFG_PERIPHERAL (signal)
Generic DMA transfer configuration for memory to/from peripheral transfers.

#define LDMA_TRANSFER_CFG_PERIPHERAL_LOOP (signal, loopCnt)
Generic DMA transfer configuration for looped memory to/from peripheral transfers.

#define LDMA_DESCRIPTOR_SINGLE_M2M_WORD (src, dest, count)
DMA descriptor initializer for single memory to memory word transfer.

#define LDMA_DESCRIPTOR_SINGLE_M2M_HALF (src, dest, count)
DMA descriptor initializer for single memory to memory half-word transfer.

#define LDMA_DESCRIPTOR_SINGLE_M2M_BYTE (src, dest, count)
DMA descriptor initializer for single memory to memory byte transfer.

#define LDMA_DESCRIPTOR_LINKABS_M2M_WORD (src, dest, count)
DMA descriptor initializer for linked memory to memory word transfer.

#define LDMA_DESCRIPTOR_LINKABS_M2M_HALF (src, dest, count)
DMA descriptor initializer for linked memory to memory half-word transfer.

#define LDMA_DESCRIPTOR_LINKABS_M2M_BYTE (src, dest, count)
DMA descriptor initializer for linked memory to memory byte transfer.

#define LDMA_DESCRIPTOR_LINKREL_M2M_WORD (src, dest, count, linkjmp)
DMA descriptor initializer for linked memory to memory word transfer.

#define LDMA_DESCRIPTOR_LINKREL_M2M_HALF (src, dest, count, linkjmp)
DMA descriptor initializer for linked memory to memory half-word transfer.

#define LDMA_DESCRIPTOR_LINKREL_M2M_BYTE (src, dest, count, linkjmp)
DMA descriptor initializer for linked memory to memory byte transfer.

#define LDMA_DESCRIPTOR_SINGLE_P2M_BYTE (src, dest, count)
DMA descriptor initializer for byte transfers from a peripheral to memory.

#define LDMA_DESCRIPTOR_SINGLE_P2P_BYTE (src, dest, count)
DMA descriptor initializer for byte transfers from a peripheral to a peripheral.

#define LDMA_DESCRIPTOR_SINGLE_M2P_BYTE (src, dest, count)
DMA descriptor initializer for byte transfers from memory to a peripheral.

#define LDMA_DESCRIPTOR_LINKREL_P2M_BYTE (src, dest, count, linkjmp)
DMA descriptor initializer for byte transfers from a peripheral to memory.

#define LDMA_DESCRIPTOR_LINKREL_P2M_WORD (src, dest, count, linkjmp)
DMA descriptor initializer for word transfers from a peripheral to memory.

#define LDMA_DESCRIPTOR_LINKREL_M2P_BYTE (src, dest, count, linkjmp)
DMA descriptor initializer for byte transfers from memory to a peripheral.
```

```
#define LDMA_DESCRIPTOR_SINGLE_WRITE (value, address)
DMA descriptor initializer for Immediate WRITE transfer.

#define LDMA_DESCRIPTOR_LINKABS_WRITE (value, address)
DMA descriptor initializer for Immediate WRITE transfer.

#define LDMA_DESCRIPTOR_LINKREL_WRITE (value, address, linkjmp)
DMA descriptor initializer for Immediate WRITE transfer.

#define LDMA_DESCRIPTOR_SINGLE_SYNC (set, clr, matchValue, matchEnable)
DMA descriptor initializer for SYNC transfer.

#define LDMA_DESCRIPTOR_LINKABS_SYNC (set, clr, matchValue, matchEnable)
DMA descriptor initializer for SYNC transfer.

#define LDMA_DESCRIPTOR_LINKREL_SYNC (set, clr, matchValue, matchEnable, linkjmp)
DMA descriptor initializer for SYNC transfer.
```

Enumeration Documentation

LDMA_CtrlBlockSize_t

LDMA_CtrlBlockSize_t

Controls the number of unit data transfers per arbitration cycle, providing a means to balance DMA channels' load on the controller.

	Enumerator
ldmaCtrlBlockSizeUnit1	One transfer per arbitration.
ldmaCtrlBlockSizeUnit2	Two transfers per arbitration.
ldmaCtrlBlockSizeUnit3	Three transfers per arbitration.
ldmaCtrlBlockSizeUnit4	Four transfers per arbitration.
ldmaCtrlBlockSizeUnit6	Six transfers per arbitration.
ldmaCtrlBlockSizeUnit8	Eight transfers per arbitration.
ldmaCtrlBlockSizeUnit16	16 transfers per arbitration.
ldmaCtrlBlockSizeUnit32	32 transfers per arbitration.
ldmaCtrlBlockSizeUnit64	64 transfers per arbitration.
ldmaCtrlBlockSizeUnit128	128 transfers per arbitration.
ldmaCtrlBlockSizeUnit256	256 transfers per arbitration.
ldmaCtrlBlockSizeUnit512	512 transfers per arbitration.
ldmaCtrlBlockSizeUnit1024	1024 transfers per arbitration.
ldmaCtrlBlockSizeAll	Lock arbitration during transfer.

LDMA_CtrlStructType_t

LDMA_CtrlStructType_t

DMA structure type.

	Enumerator
ldmaCtrlStructTypeXfer	TRANSFER transfer type.
ldmaCtrlStructTypeSync	SYNCHRONIZE transfer type.
ldmaCtrlStructTypeWrite	WRITE transfer type.

LDMA_CtrlReqMode_t

LDMA_CtrlReqMode_t

DMA transfer block or cycle selector.

Enumerator

ldmaCtrlReqModeBlock	Each DMA request trigger transfer of one block.
ldmaCtrlReqModeAll	A DMA request trigger transfer of a complete cycle.

LDMA_CtrlSrcInc_t

LDMA_CtrlSrcInc_t

Source address increment unit size.

Enumerator

ldmaCtrlSrcIncOne	Increment source address by one unit data size.
ldmaCtrlSrcIncTwo	Increment source address by two unit data sizes.
ldmaCtrlSrcIncFour	Increment source address by four unit data sizes.
ldmaCtrlSrcIncNone	Do not increment source address.

LDMA_CtrlSize_t

LDMA_CtrlSize_t

DMA transfer unit size.

Enumerator

ldmaCtrlSizeByte	Each unit transfer is a byte.
ldmaCtrlSizeHalf	Each unit transfer is a half-word.
ldmaCtrlSizeWord	Each unit transfer is a word.

LDMA_CtrlDstInc_t

LDMA_CtrlDstInc_t

Destination address increment unit size.

Enumerator

ldmaCtrlDstIncOne	Increment destination address by one unit data size.
ldmaCtrlDstIncTwo	Increment destination address by two unit data sizes.
ldmaCtrlDstIncFour	Increment destination address by four unit data sizes.
ldmaCtrlDstIncNone	Do not increment destination address.

LDMA_CtrlSrcAddrMode_t

LDMA_CtrlSrcAddrMode_t

Source addressing mode.

Enumerator	
ldmaCtrlSrcAddrModeAbs	Address fetched from a linked structure is absolute.
ldmaCtrlSrcAddrModeRel	Address fetched from a linked structure is relative.

LDMA_CtrlDstAddrMode_t

LDMA_CtrlDstAddrMode_t

Destination addressing mode.

Enumerator	
ldmaCtrlDstAddrModeAbs	Address fetched from a linked structure is absolute.
ldmaCtrlDstAddrModeRel	Address fetched from a linked structure is relative.

LDMA_LinkMode_t

LDMA_LinkMode_t

DMA link load address mode.

Enumerator	
ldmaLinkModeAbs	Link address is an absolute address value.
ldmaLinkModeRel	Link address is a two's complement relative address.

LDMA_CfgArbSlots_t

LDMA_CfgArbSlots_t

Insert extra arbitration slots to increase channel arbitration priority.

Enumerator	
ldmaCfgArbSlotsAs1	One arbitration slot selected.
ldmaCfgArbSlotsAs2	Two arbitration slots selected.
ldmaCfgArbSlotsAs4	Four arbitration slots selected.
ldmaCfgArbSlotsAs8	Eight arbitration slots selected.

LDMA_CfgSrcIncSign_t

LDMA_CfgSrcIncSign_t

Source address increment sign.

Enumerator	
ldmaCfgSrcIncSignPos	Increment source address.
ldmaCfgSrcIncSignNeg	Decrement source address.

LDMA_CfgDstIncSign_t

LDMA_CfgDstIncSign_t

Destination address increment sign.

	Enumerator
ldmaCfgDstIncSignPos	Increment destination address.
ldmaCfgDstIncSignNeg	Decrement destination address.

LDMA_PeripheralSignal_t

LDMA_PeripheralSignal_t

Peripherals that can trigger LDMA transfers.

	Enumerator
ldmaPeripheralSignal_NONE	No peripheral selected for DMA triggering.
ldmaPeripheralSignal_LDMAXBAR_PRSREQ0	Trigger on PRS REQ0.
ldmaPeripheralSignal_LDMAXBAR_PRSREQ1	Trigger on PRS REQ1.
ldmaPeripheralSignal_TIMER0_CC0	Trigger on TIMER0_CC0.
ldmaPeripheralSignal_TIMER0_CC1	Trigger on TIMER0_CC1.
ldmaPeripheralSignal_TIMER0_CC2	Trigger on TIMER0_CC2.
ldmaPeripheralSignal_TIMER0_UFOF	Trigger on TIMER0_UFOF.
ldmaPeripheralSignal_TIMER1_CC0	Trigger on TIMER1_CC0.
ldmaPeripheralSignal_TIMER1_CC1	Trigger on TIMER1_CC1.
ldmaPeripheralSignal_TIMER1_CC2	Trigger on TIMER1_CC2.
ldmaPeripheralSignal_TIMER1_UFOF	Trigger on TIMER1_UFOF.
ldmaPeripheralSignal_USART0_RXDATAV	Trigger on USART0_RXDATAV.
ldmaPeripheralSignal_USART0_RXDATAVRIGHT	Trigger on USART0_RXDATAVRIGHT.
ldmaPeripheralSignal_USART0_TXBL	Trigger on USART0_TXBL.
ldmaPeripheralSignal_USART0_TXBLRIGHT	Trigger on USART0_TXBLRIGHT.
ldmaPeripheralSignal_USART0_TXEMPTY	Trigger on USART0_TXEMPTY.
ldmaPeripheralSignal_USART1_RXDATAV	Trigger on USART1_RXDATAV.
ldmaPeripheralSignal_USART1_RXDATAVRIGHT	Trigger on USART1_RXDATAVRIGHT.
ldmaPeripheralSignal_USART1_TXBL	Trigger on USART1_TXBL.
ldmaPeripheralSignal_USART1_TXBLRIGHT	Trigger on USART1_TXBLRIGHT.
ldmaPeripheralSignal_USART1_TXEMPTY	Trigger on USART1_TXEMPTY.
ldmaPeripheralSignal_I2C0_RXDATAV	Trigger on I2C0_RXDATAV.
ldmaPeripheralSignal_I2C0_TXBL	Trigger on I2C0_TXBL.
ldmaPeripheralSignal_I2C1_RXDATAV	Trigger on I2C1_RXDATAV.
ldmaPeripheralSignal_I2C1_TXBL	Trigger on I2C1_TXBL.
ldmaPeripheralSignal_PDM_RXDATAV	Trigger on PDM_RXDATAV.
ldmaPeripheralSignal_IADC0_IADC_SCAN	Trigger on IADC0_IADC_SCAN.
ldmaPeripheralSignal_IADC0_IADC_SINGLE	Trigger on IADC0_IADC_SINGLE.
ldmaPeripheralSignal_MSC_WDATA	Trigger on MSC_WDATA.

ldmaPeripheralSignal_TIMER2_CC0	Trigger on TIMER2_CC0.
ldmaPeripheralSignal_TIMER2_CC1	Trigger on TIMER2_CC1.
ldmaPeripheralSignal_TIMER2_CC2	Trigger on TIMER2_CC2.
ldmaPeripheralSignal_TIMER2_UFOF	Trigger on TIMER2_UFOF.
ldmaPeripheralSignal_TIMER3_CC0	Trigger on TIMER3_CC0.
ldmaPeripheralSignal_TIMER3_CC1	Trigger on TIMER3_CC1.
ldmaPeripheralSignal_TIMER3_CC2	Trigger on TIMER3_CC2.
ldmaPeripheralSignal_TIMER3_UFOF	Trigger on TIMER3_UFOF.
ldmaPeripheralSignal_TIMER4_CC0	Trigger on TIMER4_CC0.
ldmaPeripheralSignal_TIMER4_CC1	Trigger on TIMER4_CC1.
ldmaPeripheralSignal_TIMER4_CC2	Trigger on TIMER4_CC2.
ldmaPeripheralSignal_TIMER4_UFOF	Trigger on TIMER4_UFOF.
ldmaPeripheralSignal_EUART0_RXFL	Trigger on EUART0_RXFL.
ldmaPeripheralSignal_EUART0_TXFL	Trigger on EUART0_TXFL.

Function Documentation

LDMA_DeInit

```
void LDMA_DeInit (void )
```

De-initialize the LDMA controller.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

LDMA interrupts are disabled and the LDMA clock is stopped.

LDMA_EnableChannelRequest

```
void LDMA_EnableChannelRequest (int ch, bool enable)
```

Enable or disable an LDMA channel request.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	LDMA channel to enable or disable requests.
bool	[in]	enable	If 'true', the request will be enabled. If 'false', the request will be disabled.

Use this function to enable or disable an LDMA channel request. This will prevent the LDMA from proceeding after its current transaction if disabled.

LDMA_Init

```
void LDMA_Init (const LDMA_Init_t * init)
```

Initialize the LDMA controller.

Parameters

Type	Direction	Argument Name	Description
const LDMA_Init_t *	[in]	init	A pointer to the initialization structure used to configure the LDMA.

This function will disable all the LDMA channels and enable the LDMA bus clock in the CMU. This function will also enable the LDMA IRQ in the NVIC and set the LDMA IRQ priority to a user-configurable priority. The LDMA interrupt priority is configured using the [LDMA_Init_t](#) structure.

Note

- Since this function enables the LDMA IRQ, always add a custom LDMA_IRQHandler to the application to handle any interrupts from LDMA.

LDMA_StartTransfer

```
void LDMA_StartTransfer (int ch, const LDMA\_TransferCfg\_t * transfer, const LDMA\_Descriptor\_t * descriptor)
```

Start a DMA transfer.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	A DMA channel.
const LDMA_TransferCfg_t *	[in]	transfer	The initialization structure used to configure the transfer.
const LDMA_Descriptor_t *	[in]	descriptor	The transfer descriptor, which can be an array of descriptors linked together. Each descriptor's fields stored in RAM will be loaded into the certain hardware registers at the proper time to perform the DMA transfer.

LDMA_StopTransfer

```
void LDMA_StopTransfer (int ch)
```

Stop a DMA transfer.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	A DMA channel to stop.

Note

- The DMA will complete the current AHB burst transfer before stopping.

LDMA_TransferDone

```
bool LDMA_TransferDone (int ch)
```

Check if a DMA transfer has completed.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	A DMA channel to check.

Returns

- True if transfer has completed, false if not.

LDMA_TransferRemainingCount

```
uint32_t LDMA_TransferRemainingCount (int ch)
```

Get the number of items remaining in a transfer.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	The channel number of the transfer to check.

Note

- This function does not take into account that a DMA transfer with a chain of linked transfers might be ongoing. It will only check the count for the current transfer.

Returns

- A number of items remaining in the transfer.

LDMA_ChannelEnabled

```
bool LDMA_ChannelEnabled (int ch)
```

Check if a certain channel is enabled.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	LDMA channel to check.

Returns

- return true if the LDMA channel is enabled and false if the channel is not enabled.

LDMA_IntClear

```
void LDMA_IntClear (uint32_t flags)
```

Clear one or more pending LDMA interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	Pending LDMA interrupt sources to clear. Use one or more valid interrupt flags for the LDMA module. The flags are LDMA_IFC_ERROR and one done flag for each channel.

LDMA_IntDisable

```
void LDMA_IntDisable (uint32_t flags)
```

Disable one or more LDMA interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	LDMA interrupt sources to disable. Use one or more valid interrupt flags for LDMA module. The flags are LDMA_IEN_ERROR and one done flag for each channel.

LDMA_IntEnable

```
void LDMA_IntEnable (uint32_t flags)
```

Enable one or more LDMA interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	LDMA interrupt sources to enable. Use one or more valid interrupt flags for LDMA module. The flags are LDMA_IEN_ERROR and one done flag for each channel.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [LDMA_IntClear\(\)](#) prior to enabling the interrupt.

LDMA_IntGet

```
uint32_t LDMA_IntGet (void )
```

Get pending LDMA interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Event bits are not cleared by the use of this function.

Returns

-

LDMA interrupt sources pending. Returns one or more valid interrupt flags for LDMA module. The flags are LDMA_IF_ERROR and one flag for each LDMA channel.

LDMA_IntGetEnabled

```
uint32_t LDMA_IntGetEnabled (void )
```

Get enabled and pending LDMA interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

- Pending and enabled LDMA interrupt sources Return value is the bitwise AND of
 - the enabled interrupt sources in LDMA_IEN and
 - the pending interrupt flags LDMA_IF

LDMA_IntSet

```
void LDMA_IntSet (uint32_t flags)
```

Set one or more pending LDMA interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	LDMA interrupt sources to set to pending. Use one or more valid interrupt flags for LDMA module. The flags are LDMA_IFS_ERROR and one done flag for each LDMA channel.

Macro Definition Documentation

LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD

```
#define LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD
```

Value:

4

Size in words of a non-extended DMA descriptor.

LDMA_DESCRIPTOR_EXTEND_SIZE_WORD

```
#define LDMA_DESCRIPTOR_EXTEND_SIZE_WORD
```

Value:

```
7
```

Size in words of an extended DMA descriptor.

LDMA_DESCRIPTOR_MAX_XFER_SIZE

```
#define LDMA_DESCRIPTOR_MAX_XFER_SIZE
```

Value:

```
(((_LDMA_CH_CTRL_XFERCNT_MASK >> _LDMA_CH_CTRL_XFERCNT_SHIFT) + 1))
```

Maximum transfer size possible per descriptor.

LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR

```
#define LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR
```

Value:

```
(addr)
```

Converts a [LDMA_Descriptor_t](#) pointer to the value suitable to write to the linkAddr field of a [LDMA_Descriptor_t](#).

LDMA_DESCRIPTOR_LINKABS_LINKADDR_TO_ADDR

```
#define LDMA_DESCRIPTOR_LINKABS_LINKADDR_TO_ADDR
```

Value:

```
(linkAddr)
```

Converts a [LDMA_Descriptor_t](#) linkAddr field value back to a [LDMA_Descriptor_t](#) pointer.

LDMA_INIT_DEFAULT

```
#define LDMA_INIT_DEFAULT
```

Value:

```
0 | {
0 |     .ldmaInitCtrlNumFixed      = _LDMA_CTRL_NUMFIXED_DEFAULT, /* Fixed priority arbitration.*/ \
0 |     .ldmaInitCtrlSyncPrsClrEn = 0, /* No PRS Synctrig clear enable*/ \
0 |     .ldmaInitCtrlSyncPrsSetEn = 0, /* No PRS Synctrig set enable. */ \
0 |     .ldmaInitIrqPriority       = 3 /* IRQ priority level 3. */ \
0 | }
```

Default DMA initialization structure.

LDMA_TRANSFER_CFG_MEMORY

```
#define LDMA_TRANSFER_CFG_MEMORY
```

Value:

```
{
  0, 0, 0, 0, 0, \
  false, false, ldmaCfgArbSlotsAs1, \
  ldmaCfgSrcIncSignPos, ldmaCfgDstIncSignPos, 0 \
}
```

Generic DMA transfer configuration for memory to memory transfers.

LDMA_TRANSFER_CFG_MEMORY_LOOP

```
#define LDMA_TRANSFER_CFG_MEMORY_LOOP
```

Value:

```
{
  0, 0, 0, 0, 0, \
  false, false, ldmaCfgArbSlotsAs1, \
  ldmaCfgSrcIncSignPos, ldmaCfgDstIncSignPos, \
  loopCnt \
}
```

Generic DMA transfer configuration for looped memory to memory transfers.

LDMA_TRANSFER_CFG_PERIPHERAL

```
#define LDMA_TRANSFER_CFG_PERIPHERAL
```

Value:

```
{
  signal, 0, 0, 0, 0, \
  false, false, ldmaCfgArbSlotsAs1, \
  ldmaCfgSrcIncSignPos, ldmaCfgDstIncSignPos, 0 \
}
```

Generic DMA transfer configuration for memory to/from peripheral transfers.

LDMA_TRANSFER_CFG_PERIPHERAL_LOOP

```
#define LDMA_TRANSFER_CFG_PERIPHERAL_LOOP
```

Value:

```
{
  signal, 0, 0, 0, 0, \
  false, false, ldmaCfgArbSlotsAs1, \
  ldmaCfgSrcIncSignPos, ldmaCfgDstIncSignPos, loopCnt \
}
```

Generic DMA transfer configuration for looped memory to/from peripheral transfers.

LDMA_DESCRIPTOR_SINGLE_M2M_WORD

```
#define LDMA_DESCRIPTOR_SINGLE_M2M_WORD
```

Value:

```
0  {
0      .xfer =
0      {
0          .structType = ldmaCtrlStructTypeXfer,
0          .structReq  = 1,
0          .xferCnt    = (count) - 1,
0          .byteSwap   = 0,
0          .blockSize  = ldmaCtrlBlockSizeUnit1,
0          .doneIfs    = 1,
0          .reqMode     = ldmaCtrlReqModeAll,
0          .decLoopCnt = 0,
0          .ignoreSrec = 0,
0          .srcInc      = ldmaCtrlSrcIncOne,
0          .size        = ldmaCtrlSizeWord,
0          .dstInc      = ldmaCtrlDstIncOne,
0          .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0          .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0          .srcAddr     = (uint32_t)(src),
0          .dstAddr     = (uint32_t)(dest),
0          .linkMode    = 0,
0          .link        = 0,
0          .linkAddr    = 0
0      }
0  }
```

DMA descriptor initializer for single memory to memory word transfer.

LDMA_DESCRIPTOR_SINGLE_M2M_HALF

```
#define LDMA_DESCRIPTOR_SINGLE_M2M_HALF
```

Value:

```
0  {
0      .xfer =
0      {
0          .structType = ldmaCtrlStructTypeXfer,
0          .structReq  = 1,
0          .xferCnt    = (count) - 1,
0          .byteSwap   = 0,
0          .blockSize  = ldmaCtrlBlockSizeUnit1,
0          .doneIfs    = 1,
0          .reqMode     = ldmaCtrlReqModeAll,
0          .decLoopCnt = 0,
0          .ignoreSrec = 0,
0          .srcInc      = ldmaCtrlSrcIncOne,
0          .size        = ldmaCtrlSizeHalf,
0          .dstInc      = ldmaCtrlDstIncOne,
0          .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0          .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0          .srcAddr     = (uint32_t)(src),
0          .dstAddr     = (uint32_t)(dest),
0          .linkMode    = 0,
0          .link        = 0,
0          .linkAddr    = 0
0      }
0  }
```

DMA descriptor initializer for single memory to memory half-word transfer.

LDMA_DESCRIPTOR_SINGLE_M2M_BYTE

```
#define LDMA_DESCRIPTOR_SINGLE_M2M_BYTE
```

Value:

```

0      {
0      {
0          .xfer =
0          {
0              .structType = ldmaCtrlStructTypeXfer,
0              .structReq  = 1,
0              .xferCnt    = (count) - 1,
0              .byteSwap   = 0,
0              .blockSize  = ldmaCtrlBlockSizeUnit1,
0              .doneIfs    = 1,
0              .reqMode    = ldmaCtrlReqModeAll,
0              .decLoopCnt = 0,
0              .ignoreSrec = 0,
0              .srcInc     = ldmaCtrlSrcIncOne,
0              .size      = ldmaCtrlSizeByte,
0              .dstInc     = ldmaCtrlDstIncOne,
0              .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0              .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0              .srcAddr    = (uint32_t)(src),
0              .dstAddr    = (uint32_t)(dest),
0              .linkMode   = 0,
0              .link       = 0,
0              .linkAddr   = 0
0          }
0      }
0  }

```

DMA descriptor initializer for single memory to memory byte transfer.

LDMA_DESCRIPTOR_LINKABS_M2M_WORD

```
#define LDMA_DESCRIPTOR_LINKABS_M2M_WORD
```

Value:

```

0      {
0      {
0          .xfer =
0          {
0              .structType = ldmaCtrlStructTypeXfer,
0              .structReq  = 1,
0              .xferCnt    = (count) - 1,
0              .byteSwap   = 0,
0              .blockSize  = ldmaCtrlBlockSizeUnit1,
0              .doneIfs    = 0,
0              .reqMode    = ldmaCtrlReqModeAll,
0              .decLoopCnt = 0,
0              .ignoreSrec = 0,
0              .srcInc     = ldmaCtrlSrcIncOne,
0              .size      = ldmaCtrlSizeWord,
0              .dstInc     = ldmaCtrlDstIncOne,
0              .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0              .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0              .srcAddr    = (uint32_t)(src),
0              .dstAddr    = (uint32_t)(dest),
0              .linkMode   = ldmaLinkModeAbs,
0              .link       = 1,
0              .linkAddr   = 0 /* Must be set runtime ! */
0          }
0      }
0  }

```

DMA descriptor initializer for linked memory to memory word transfer.

Link address must be an absolute address. Note

- The linkAddr member of the transfer descriptor is not initialized. linkAddr must be initialized by using the proper bits right-shift to get the correct bits from the absolute address. LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR macro should be used for that operation:

```
desc.linkAddr = LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR(&next_desc);
```

The opposite bit shift (left) must be done with LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR if linkAddr is read.

LDMA_DESCRIPTOR_LINKABS_M2M_HALF

```
#define LDMA_DESCRIPTOR_LINKABS_M2M_HALF
```

Value:

```
0  {
0      .xfer =
0      {
0          .structType = ldmaCtrlStructTypeXfer,
0          .structReq  = 1,
0          .xferCnt    = (count) - 1,
0          .byteSwap   = 0,
0          .blockSize  = ldmaCtrlBlockSizeUnit1,
0          .doneIfs    = 0,
0          .reqMode     = ldmaCtrlReqModeAll,
0          .decLoopCnt = 0,
0          .ignoreSrec = 0,
0          .srcInc      = ldmaCtrlSrcIncOne,
0          .size        = ldmaCtrlSizeHalf,
0          .dstInc      = ldmaCtrlDstIncOne,
0          .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0          .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0          .srcAddr     = (uint32_t)(src),
0          .dstAddr     = (uint32_t)(dest),
0          .linkMode    = ldmaLinkModeAbs,
0          .link        = 1,
0          .linkAddr    = 0 /* Must be set runtime ! */
0      }
0  }
```

DMA descriptor initializer for linked memory to memory half-word transfer.

Link address must be an absolute address. Note

- The linkAddr member of the transfer descriptor is not initialized. linkAddr must be initialized by using the proper bits right-shift to get the correct bits from the absolute address.

LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR macro should be used for that operation:

```
desc.linkAddr = LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR(&next_desc);
```

The opposite bit shift (left) must be done with LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR if linkAddr is read.

LDMA_DESCRIPTOR_LINKABS_M2M_BYTE

```
#define LDMA_DESCRIPTOR_LINKABS_M2M_BYTE
```

Value:

```
0  {
0      .xfer =
0      {
0          .structType = ldmaCtrlStructTypeXfer,
0          .structReq  = 1,
0          .xferCnt    = (count) - 1,
0          .byteSwap   = 0,
0          .blockSize  = ldmaCtrlBlockSizeUnit1,
0          .doneIfs    = 0,
0          .reqMode     = ldmaCtrlReqModeAll,
0          .decLoopCnt = 0,
0          .ignoreSrec = 0,
0          .srcInc      = ldmaCtrlSrcIncOne,
0          .size        = ldmaCtrlSizeByte,
0          .dstInc      = ldmaCtrlDstIncOne,
0          .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0          .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0          .srcAddr     = (uint32_t)(src),
0          .dstAddr     = (uint32_t)(dest),
0          .linkMode    = ldmaLinkModeAbs,
0          .link        = 1,
0          .linkAddr    = 0 /* Must be set runtime ! */
0      }
0  }
```

DMA descriptor initializer for linked memory to memory byte transfer.

Link address must be an absolute address. Note

- The linkAddr member of the transfer descriptor is not initialized. linkAddr must be initialized by using the proper bits right-shift to get the correct bits from the absolute address.

LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR macro should be used for that operation:

```
desc.linkAddr = LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR(&next_desc);
```

The opposite bit shift (left) must be done with LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR if linkAddr is read.

LDMA_DESCRIPTOR_LINKREL_M2M_WORD

```
#define LDMA_DESCRIPTOR_LINKREL_M2M_WORD
```

Value:

```
0 | {
0 |     .xfer =
0 |     {
0 |         .structType = ldmaCtrlStructTypeXfer,
0 |         .structReq  = 1,
0 |         .xferCnt    = (count) - 1,
0 |         .byteSwap   = 0,
0 |         .blockSize  = ldmaCtrlBlockSizeUnit1,
0 |         .doneIfs    = 0,
0 |         .reqMode    = ldmaCtrlReqModeAll,
0 |         .decLoopCnt = 0,
0 |         .ignoreSrec = 0,
0 |         .srcInc     = ldmaCtrlSrcIncOne,
0 |         .size       = ldmaCtrlSizeWord,
0 |         .dstInc     = ldmaCtrlDstIncOne,
0 |         .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0 |         .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0 |         .srcAddr     = (uint32_t)(src),
0 |         .dstAddr     = (uint32_t)(dest),
0 |         .linkMode    = ldmaLinkModeRel,
0 |         .link        = 1,
0 |         .linkAddr    = (linkjmp) * LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD
0 |     }
0 | }
```

DMA descriptor initializer for linked memory to memory word transfer.

Link address is a relative address. Note

- The linkAddr member of the transfer descriptor is initialized to 4 (regular descriptor) or 7 (extended descriptor), assuming that the next descriptor immediately follows this descriptor (in memory).

LDMA_DESCRIPTOR_LINKREL_M2M_HALF

```
#define LDMA_DESCRIPTOR_LINKREL_M2M_HALF
```

Value:

```
0 | {
0 |     .xfer =
0 |     {
0 |         .structType = ldmaCtrlStructTypeXfer,
0 |         .structReq  = 1,
0 |         .xferCnt    = (count) - 1,
0 |         .byteSwap   = 0,
0 |         .blockSize  = ldmaCtrlBlockSizeUnit1,
0 |         .doneIfs    = 0,
0 |         .reqMode    = ldmaCtrlReqModeAll,
0 |         .decLoopCnt = 0,
0 |         .ignoreSrec = 0,
0 |         .srcInc     = ldmaCtrlSrcIncOne,
0 |         .size       = ldmaCtrlSizeHalf,
0 |     }
0 | }
```

```

0 |         .dstInc      = ldmaCtrlDstIncOne,          \
0 |         .srcAddrMode = ldmaCtrlSrcAddrModeAbs,     \
0 |         .dstAddrMode = ldmaCtrlDstAddrModeAbs,     \
0 |         .srcAddr     = (uint32_t)(src),            \
0 |         .dstAddr     = (uint32_t)(dest),           \
0 |         .linkMode    = ldmaLinkModeRel,           \
0 |         .link        = 1,                          \
0 |         .linkAddr    = (linkjmp) * LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD \
0 |     }
0 | }

```

DMA descriptor initializer for linked memory to memory half-word transfer.

Link address is a relative address. Note

- The linkAddr member of the transfer descriptor is initialized to 4 (regular descriptor) or 7 (extended descriptor), assuming that the next descriptor immediately follows this descriptor (in memory).

LDMA_DESCRIPTOR_LINKREL_M2M_BYTE

```
#define LDMA_DESCRIPTOR_LINKREL_M2M_BYTE
```

Value:

```

0 | {
0 |     .xfer =
0 |     {
0 |         .structType = ldmaCtrlStructTypeXfer,
0 |         .structReq  = 1,
0 |         .xferCnt    = (count) - 1,
0 |         .byteSwap   = 0,
0 |         .blockSize  = ldmaCtrlBlockSizeUnit1,
0 |         .doneIfs    = 0,
0 |         .reqMode    = ldmaCtrlReqModeAll,
0 |         .decLoopCnt = 0,
0 |         .ignoreSrec = 0,
0 |         .srcInc     = ldmaCtrlSrcIncOne,
0 |         .size       = ldmaCtrlSizeByte,
0 |         .dstInc     = ldmaCtrlDstIncOne,
0 |         .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0 |         .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0 |         .srcAddr     = (uint32_t)(src),
0 |         .dstAddr     = (uint32_t)(dest),
0 |         .linkMode    = ldmaLinkModeRel,
0 |         .link        = 1,
0 |         .linkAddr    = (linkjmp) * LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD \
0 |     }
0 | }

```

DMA descriptor initializer for linked memory to memory byte transfer.

Link address is a relative address. Note

- The linkAddr member of the transfer descriptor is initialized to 4 (regular descriptor) or 7 (extended descriptor), assuming that the next descriptor immediately follows this descriptor (in memory).

LDMA_DESCRIPTOR_SINGLE_P2M_BYTE

```
#define LDMA_DESCRIPTOR_SINGLE_P2M_BYTE
```

Value:

```

0 | {
0 |     .xfer =
0 |     {
0 |         .structType = ldmaCtrlStructTypeXfer,
0 |         .structReq  = 0,
0 |         .xferCnt    = (count) - 1,
0 |         .byteSwap   = 0,
0 |         .blockSize  = ldmaCtrlBlockSizeUnit1,
0 |         .doneIfs    = 1,
0 |     }
0 | }

```

```

0      .reqMode      = ldmaCtrlReqModeBlock,      \
0      .decLoopCnt   = 0,                        \
0      .ignoreSrec   = 0,                        \
0      .srcInc       = ldmaCtrlSrcIncNone,        \
0      .size         = ldmaCtrlSizeByte,         \
0      .dstInc       = ldmaCtrlDstIncOne,         \
0      .srcAddrMode  = ldmaCtrlSrcAddrModeAbs,    \
0      .dstAddrMode  = ldmaCtrlDstAddrModeAbs,    \
0      .srcAddr      = (uint32_t)(src),           \
0      .dstAddr      = (uint32_t)(dest),          \
0      .linkMode     = 0,                        \
0      .link         = 0,                        \
0      .linkAddr     = 0,                        \
0  }
0  }

```

DMA descriptor initializer for byte transfers from a peripheral to memory.

LDMA_DESCRIPTOR_SINGLE_P2P_BYTE

```
#define LDMA_DESCRIPTOR_SINGLE_P2P_BYTE
```

Value:

```

0      {
0      .xfer =
0      {
0      .structType   = ldmaCtrlStructTypeXfer,    \
0      .structReq    = 0,                        \
0      .xferCnt      = (count) - 1,              \
0      .byteSwap     = 0,                        \
0      .blockSize    = ldmaCtrlBlockSizeUnit1,   \
0      .doneIfs      = 1,                        \
0      .reqMode      = ldmaCtrlReqModeBlock,    \
0      .decLoopCnt   = 0,                        \
0      .ignoreSrec   = 0,                        \
0      .srcInc       = ldmaCtrlSrcIncNone,        \
0      .size         = ldmaCtrlSizeByte,         \
0      .dstInc       = ldmaCtrlDstIncNone,        \
0      .srcAddrMode  = ldmaCtrlSrcAddrModeAbs,    \
0      .dstAddrMode  = ldmaCtrlDstAddrModeAbs,    \
0      .srcAddr      = (uint32_t)(src),           \
0      .dstAddr      = (uint32_t)(dest),          \
0      .linkMode     = 0,                        \
0      .link         = 0,                        \
0      .linkAddr     = 0,                        \
0  }
0  }

```

DMA descriptor initializer for byte transfers from a peripheral to a peripheral.

LDMA_DESCRIPTOR_SINGLE_M2P_BYTE

```
#define LDMA_DESCRIPTOR_SINGLE_M2P_BYTE
```

Value:

```

0      {
0      .xfer =
0      {
0      .structType   = ldmaCtrlStructTypeXfer,    \
0      .structReq    = 0,                        \
0      .xferCnt      = (count) - 1,              \
0      .byteSwap     = 0,                        \
0      .blockSize    = ldmaCtrlBlockSizeUnit1,   \
0      .doneIfs      = 1,                        \
0      .reqMode      = ldmaCtrlReqModeBlock,    \
0      .decLoopCnt   = 0,                        \
0      .ignoreSrec   = 0,                        \
0      .srcInc       = ldmaCtrlSrcIncOne,         \
0      .size         = ldmaCtrlSizeByte,         \
0      .dstInc       = ldmaCtrlDstIncNone,        \
0      .srcAddrMode  = ldmaCtrlSrcAddrModeAbs,    \
0      .dstAddrMode  = ldmaCtrlDstAddrModeAbs,    \
0      .srcAddr      = (uint32_t)(src),           \
0      .dstAddr      = (uint32_t)(dest),          \
0  }
0  }

```

```

0 |         .linkMode    = 0,           \
0 |         .link        = 0,           \
0 |         .linkAddr    = 0,           \
0 |     }                          \
0 | }

```

DMA descriptor initializer for byte transfers from memory to a peripheral.

LDMA_DESCRIPTOR_LINKREL_P2M_BYTE

```
#define LDMA_DESCRIPTOR_LINKREL_P2M_BYTE
```

Value:

```

0 | {
0 |     .xfer =
0 |     {
0 |         .structType    = ldmaCtrlStructTypeXfer,
0 |         .structReq     = 0,
0 |         .xferCnt       = (count) - 1,
0 |         .byteSwap      = 0,
0 |         .blockSize     = ldmaCtrlBlockSizeUnit1,
0 |         .doneIfs       = 1,
0 |         .reqMode       = ldmaCtrlReqModeBlock,
0 |         .decLoopCnt    = 0,
0 |         .ignoreSrec    = 0,
0 |         .srcInc        = ldmaCtrlSrcIncNone,
0 |         .size          = ldmaCtrlSizeByte,
0 |         .dstInc        = ldmaCtrlDstIncOne,
0 |         .srcAddrMode   = ldmaCtrlSrcAddrModeAbs,
0 |         .dstAddrMode   = ldmaCtrlDstAddrModeAbs,
0 |         .srcAddr       = (uint32_t)(src),
0 |         .dstAddr       = (uint32_t)(dest),
0 |         .linkMode      = ldmaLinkModeRel,
0 |         .link          = 1,
0 |         .linkAddr      = (linkjmp) * LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD \
0 |     }
0 | }

```

DMA descriptor initializer for byte transfers from a peripheral to memory.

LDMA_DESCRIPTOR_LINKREL_P2M_WORD

```
#define LDMA_DESCRIPTOR_LINKREL_P2M_WORD
```

Value:

```

0 | {
0 |     .xfer =
0 |     {
0 |         .structType    = ldmaCtrlStructTypeXfer,
0 |         .structReq     = 0,
0 |         .xferCnt       = (count) - 1,
0 |         .byteSwap      = 0,
0 |         .blockSize     = ldmaCtrlBlockSizeUnit1,
0 |         .doneIfs       = 1,
0 |         .reqMode       = ldmaCtrlReqModeBlock,
0 |         .decLoopCnt    = 0,
0 |         .ignoreSrec    = 0,
0 |         .srcInc        = ldmaCtrlSrcIncNone,
0 |         .size          = ldmaCtrlSizeWord,
0 |         .dstInc        = ldmaCtrlDstIncOne,
0 |         .srcAddrMode   = ldmaCtrlSrcAddrModeAbs,
0 |         .dstAddrMode   = ldmaCtrlDstAddrModeAbs,
0 |         .srcAddr       = (uint32_t)(src),
0 |         .dstAddr       = (uint32_t)(dest),
0 |         .linkMode      = ldmaLinkModeRel,
0 |         .link          = 1,
0 |         .linkAddr      = (linkjmp) * LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD \
0 |     }
0 | }

```

DMA descriptor initializer for word transfers from a peripheral to memory.

LDMA_DESCRIPTOR_LINKREL_M2P_BYTE

```
#define LDMA_DESCRIPTOR_LINKREL_M2P_BYTE
```

Value:

```
0  {
0      .xfer =
0      {
0          .structType = ldmaCtrlStructTypeXfer,
0          .structReq  = 0,
0          .xferCnt    = (count) - 1,
0          .byteSwap   = 0,
0          .blockSize  = ldmaCtrlBlockSizeUnit1,
0          .doneIfs    = 1,
0          .reqMode    = ldmaCtrlReqModeBlock,
0          .decLoopCnt = 0,
0          .ignoreSrec = 0,
0          .srcInc     = ldmaCtrlSrcIncOne,
0          .size       = ldmaCtrlSizeByte,
0          .dstInc     = ldmaCtrlDstIncNone,
0          .srcAddrMode = ldmaCtrlSrcAddrModeAbs,
0          .dstAddrMode = ldmaCtrlDstAddrModeAbs,
0          .srcAddr    = (uint32_t)(src),
0          .dstAddr    = (uint32_t)(dest),
0          .linkMode    = ldmaLinkModeRel,
0          .link        = 1,
0          .linkAddr    = (linkjmp) * LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD
0      }
0  }
```

DMA descriptor initializer for byte transfers from memory to a peripheral.

LDMA_DESCRIPTOR_SINGLE_WRITE

```
#define LDMA_DESCRIPTOR_SINGLE_WRITE
```

Value:

```
0  {
0      .wri =
0      {
0          .structType = ldmaCtrlStructTypeWrite,
0          .structReq  = 1,
0          .xferCnt    = 0,
0          .byteSwap   = 0,
0          .blockSize  = 0,
0          .doneIfs    = 1,
0          .reqMode    = 0,
0          .decLoopCnt = 0,
0          .ignoreSrec = 0,
0          .srcInc     = 0,
0          .size       = 0,
0          .dstInc     = 0,
0          .srcAddrMode = 0,
0          .dstAddrMode = 0,
0          .immVal     = (value),
0          .dstAddr    = (uint32_t)(address),
0          .linkMode    = 0,
0          .link        = 0,
0          .linkAddr    = 0
0      }
0  }
```

DMA descriptor initializer for Immediate WRITE transfer.

LDMA_DESCRIPTOR_LINKABS_WRITE

```
#define LDMA_DESCRIPTOR_LINKABS_WRITE
```

Value:

```

0  {
1      .wri =
2      {
3          .structType    = ldmaCtrlStructTypewrite,
4          .structReq     = 1,
5          .xferCnt       = 0,
6          .byteSwap      = 0,
7          .blockSize     = 0,
8          .doneIfs       = 0,
9          .reqMode       = 0,
10         .decLoopCnt     = 0,
11         .ignoreSrec     = 0,
12         .srcInc         = 0,
13         .size           = 0,
14         .dstInc         = 0,
15         .srcAddrMode    = 0,
16         .dstAddrMode    = 0,
17         .immVal         = (value),
18         .dstAddr        = (uint32_t)(address),
19         .linkMode       = ldmaLinkModeAbs,
20         .link           = 1,
21         .linkAddr       = 0 /* Must be set runtime ! */
22     }
23 }

```

DMA descriptor initializer for Immediate WRITE transfer.

Link address must be an absolute address. Note

- The linkAddr member of the transfer descriptor is not initialized. linkAddr must be initialized by using the proper bits right-shift to get the correct bits from the absolute address.

LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR should be used for that operation:

```
desc.linkAddr = LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR(&next_desc);
```

The opposite bit shift (left) must be done with `LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR` if `linkAddr` is read.

LDMA_DESCRIPTOR_LINKREL_WRITE

```
#define LDMA_DESCRIPTOR_LINKREL_WRITE
```

Value:

```

0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9
0 10
0 11
0 12
0 13
0 14
0 15
0 16
0 17
0 18
0 19
0 20
0 21
0 22
0 23
0 24
0 25
0 26
0 27
0 28
0 29
0 30
0 31
0 32
0 33
0 34
0 35
0 36
0 37
0 38
0 39
0 40
0 41
0 42
0 43
0 44
0 45
0 46
0 47
0 48
0 49
0 50
0 51
0 52
0 53
0 54
0 55
0 56
0 57
0 58
0 59
0 60
0 61
0 62
0 63
0 64
0 65
0 66
0 67
0 68
0 69
0 70
0 71
0 72
0 73
0 74
0 75
0 76
0 77
0 78
0 79
0 80
0 81
0 82
0 83
0 84
0 85
0 86
0 87
0 88
0 89
0 90
0 91
0 92
0 93
0 94
0 95
0 96
0 97
0 98
0 99
0 100
0 101
0 102
0 103
0 104
0 105
0 106
0 107
0 108
0 109
0 110
0 111
0 112
0 113
0 114
0 115
0 116
0 117
0 118
0 119
0 120
0 121
0 122
0 123
0 124
0 125
0 126
0 127
0 128
0 129
0 130
0 131
0 132
0 133
0 134
0 135
0 136
0 137
0 138
0 139
0 140
0 141
0 142
0 143
0 144
0 145
0 146
0 147
0 148
0 149
0 150
0 151
0 152
0 153
0 154
0 155
0 156
0 157
0 158
0 159
0 160
0 161
0 162
0 163
0 164
0 165
0 166
0 167
0 168
0 169
0 170
0 171
0 172
0 173
0 174
0 175
0 176
0 177
0 178
0 179
0 180
0 181
0 182
0 183
0 184
0 185
0 186
0 187
0 188
0 189
0 190
0 191
0 192
0 193
0 194
0 195
0 196
0 197
0 198
0 199
0 200
0 201
0 202
0 203
0 204
0 205
0 206
0 207
0 208
0 209
0 210
0 211
0 212
0 213
0 214
0 215
0 216
0 217
0 218
0 219
0 220
0 221
0 222
0 223
0 224
0 225
0 226
0 227
0 228
0 229
0 230
0 231
0 232
0 233
0 234
0 235
0 236
0 237
0 238
0 239
0 240
0 241
0 242
0 243
0 244
0 245
0 246
0 247
0 248
0 249
0 250
0 251
0 252
0 253
0 254
0 255
0 256
0 257
0 258
0 259
0 260
0 261
0 262
0 263
0 264
0 265
0 266
0 267
0 268
0 269
0 270
0 271
0 272
0 273
0 274
0 275
0 276
0 277
0 278
0 279
0 280
0 281
0 282
0 283
0 284
0 285
0 286
0 287
0 288
0 289
0 290
0 291
0 292
0 293
0 294
0 295
0 296
0 297
0 298
0 299
0 300
0 301
0 302
0 303
0 304
0 305
0 306
0 307
0 308
0 309
0 310
0 311
0 312
0 313
0 314
0 315
0 316
0 317
0 318
0 319
0 320
0 321
0 322
0 323
0 324
0 325
0 326
0 327
0 328
0 329
0 330
0 331
0 332
0 333
0 334
0 335
0 336
0 337
0 338
0 339
0 340
0 341
0 342
0 343
0 344
0 345
0 346
0 347
0 348
0 349
0 350
0 351
0 352
0 353
0 354
0 355
0 356
0 357
0 358
0 359
0 360
0 361
0 362
0 363
0 364
0 365
0 366
0 367
0 368
0 369
0 370
0 371
0 372
0 373
0 374
0 375
0 376
0 377
0 378
0 379
0 380
0 381
0 382
0 383
0 384
0 385
0 386
0 387
0 388
0 389
0 390
0 391
0 392
0 393
0 394
0 395
0 396
0 397
0 398
0 399
0 400
0 401
0 402
0 403
0 404
0 405
0 406
0 407
0 408
0 409
0 410
0 411
0 412
0 413
0 414
0 415
0 416
0 417
0 418
0 419
0 420
0 421
0 422
0 423
0 424
0 425
0 426
0 427
0 428
0 429
0 430
0 431
0 432
0 433
0 434
0 435
0 436
0 437
0 438
0 439
0 440
0 441
0 442
0 443
0 444
0 445
0 446
0 447
0 448
0 449
0 450
0 451
0 452
0 453
0 454
0 455
0 456
0 457
0 458
0 459
0 460
0 461
0 462
0 463
0 464
0 465
0 466
0 467
0 468
0 469
0 470
0 471
0 472
0 473
0 474
0 475
0 476
0 477
0 478
0 479
0 480
0 481
0 482
0 483
0 484
0 485
0 486
0 487
0 488
0 489
0 490
0 491
0 492
0 493
0 494
0 495
0 496
0 497
0 498
0 499
0 500
0 501
0 502
0 503
0 504
0 505
0 506
0 507
0 508
0 509
0 510
0 511
0 512
0 513
0 514
0 515
0 516
0 517
0 518
0 519
0 520
0 521
0 522
0 523
0 524
0 525
0 526
0 527
0 528
0 529
0 530
0 531
0 532
0 533
0 534
0 535
0 536
0 537
0 538
0 539
0 540
0 541
0 542
0 543
0 544
0 545
0 546
0 547
0 548
0 549
0 550
0 551
0 552
0 553
0 554
0 555
0 556
0 557
0 558
0 559
0 560
0 561
0 562
0 563
0 564
0 565
0 566
0 567
0 568
0 569
0 570
0 571
0 572
0 573
0 574
0 575
0 576
0 577
0 578
0 579
0 580
0 581
0 582
0 583
0 584
0 585
0 586
0 587
0 588
0 589
0 590
0 591
0 592
0 593
0 594
0 595
0 596
0 597
0 598
0 599
0 600
0 601
0 602
0 603
0 604
0 605
0 606
0 607
0 608
0 609
0 610
0 611
0 612
0 613
0 614
0 615
0 616
0 617
0 618
0 619
0 620
0 621
0 622
0 623
0 624
0 625
0 626
0 627
0 628
0 629
0 630
0 631
0 632
0 633
0 634
0 635
0 636
0 637
0 638
0 639
0 640
0 641
0 642
0 643
0 644
0 645
0 646
0 647
0 648
0 649
0 650
0 651
0 652
0 653
0 654
0 655
0 656
0 657
0 658
0 659
0 660
0 661
0 662
0 663
0 664
0 665
0 666
0 667
0 668
0 669
0 670
0 671
0 672
0 673
0 674
0 675
0 676
0 677
0 678
0 679
0 680
0 681
0 682
0 683
0 684
0 685
0 686
0 687
0 688
0 689
0 690
0 691
0 692
0 693
0 694
0 695
0 696
0 697
0 698
0 699
0 7
```

DMA descriptor initializer for Immediate WRITE transfer.

LDMA_DESCRIPTOR_SINGLE_SYNC

```
#define LDMA_DESCRIPTOR_SINGLE_SYNC
```

Value:

```
0  {
0      .sync =
0      {
0          .structType = ldmaCtrlStructTypeSync,
0          .structReq  = 1,
0          .xferCnt    = 0,
0          .byteSwap   = 0,
0          .blockSize  = 0,
0          .doneIfs    = 1,
0          .reqMode     = 0,
0          .decLoopCnt = 0,
0          .ignoreSrec = 0,
0          .srcInc      = 0,
0          .size        = 0,
0          .dstInc      = 0,
0          .srcAddrMode = 0,
0          .dstAddrMode = 0,
0          .syncSet     = (set),
0          .syncClr     = (clr),
0          .matchVal    = (matchValue),
0          .matchEn     = (matchEnable),
0          .linkMode    = 0,
0          .link         = 0,
0          .linkAddr    = 0
0      }
0  }
```

DMA descriptor initializer for SYNC transfer.

LDMA_DESCRIPTOR_LINKABS_SYNC

```
#define LDMA_DESCRIPTOR_LINKABS_SYNC
```

Value:

```
0  {
0      .sync =
0      {
0          .structType = ldmaCtrlStructTypeSync,
0          .structReq  = 1,
0          .xferCnt    = 0,
0          .byteSwap   = 0,
0          .blockSize  = 0,
0          .doneIfs    = 0,
0          .reqMode     = 0,
0          .decLoopCnt = 0,
0          .ignoreSrec = 0,
0          .srcInc      = 0,
0          .size        = 0,
0          .dstInc      = 0,
0          .srcAddrMode = 0,
0          .dstAddrMode = 0,
0          .syncSet     = (set),
0          .syncClr     = (clr),
0          .matchVal    = (matchValue),
0          .matchEn     = (matchEnable),
0          .linkMode    = ldmaLinkModeAbs,
0          .link         = 1,
0          .linkAddr    = 0 /* Must be set runtime ! */
0      }
0  }
```

DMA descriptor initializer for SYNC transfer.

Link address must be an absolute address. Note

- The linkAddr member of the transfer descriptor is not initialized. linkAddr must be initialized by using the proper bits right-shift to get the correct bits from the absolute address.

LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR should be used for that operation:

```
desc.linkAddr = LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR(&next_desc);
```

The opposite bit shift (left) must be done with `LDMA_DESCRIPTOR_LINKABS_ADDR_TO_LINKADDR` if `linkAddr` is read.

LDMA_DESCRIPTOR_LINKREL_SYNC

```
#define LDMA_DESCRIPTOR_LINKREL_SYNC
```

Value:

```
0  {
0  .sync =
0  {
0      .structType = ldmaCtrlStructTypeSync,
0      .structReq  = 1,
0      .xferCnt    = 0,
0      .byteSwap   = 0,
0      .blockSize  = 0,
0      .doneIfs    = 0,
0      .reqMode    = 0,
0      .decLoopCnt = 0,
0      .ignoreSrec = 0,
0      .srcInc     = 0,
0      .size       = 0,
0      .dstInc     = 0,
0      .srcAddrMode = 0,
0      .dstAddrMode = 0,
0      .syncSet    = (set),
0      .syncClr    = (clr),
0      .matchVal   = (matchValue),
0      .matchEn    = (matchEnable),
0      .linkMode   = ldmaLinkModeRel,
0      .link       = 1,
0      .linkAddr   = (linkjmp) * LDMA_DESCRIPTOR_NON_EXTEND_SIZE_WORD
0  }
0 }
```

DMA descriptor initializer for SYNC transfer.

LDMA_Descriptor_t

DMA descriptor.

The LDMA DMA controller supports three different DMA descriptors. Each consists of four WORDs which map directly onto HW control registers for a given DMA channel. The three descriptor types are XFER, SYNC and WRI. Refer to the reference manual for further information.

Public Attributes

uint32_t	structType Set to 0 to select XFER descriptor type.
uint32_t	reserved0 Reserved.
uint32_t	structReq DMA transfer trigger during LINKLOAD.
uint32_t	xferCnt Transfer count minus one.
uint32_t	byteSwap Enable byte swapping transfers.
uint32_t	blockSize Number of unit transfers per arbitration cycle.
uint32_t	doneIfs Generate interrupt when done.
uint32_t	reqMode Block or cycle transfer selector.
uint32_t	decLoopCnt Enable looped transfers.
uint32_t	ignoreSrec Ignore single requests.
uint32_t	srcInc Source address increment unit size.
uint32_t	size DMA transfer unit size.
uint32_t	dstInc Destination address increment unit size.
uint32_t	srcAddrMode Source addressing mode.
uint32_t	dstAddrMode Destination addressing mode.
uint32_t	srcAddr DMA source address.
uint32_t	dstAddr DMA destination address.
uint32_t	linkMode Select absolute or relative link address.

uint32_t	link	Enable LINKLOAD when transfer is done.
int32_t	linkAddr	Address of next (linked) descriptor.
struct LDMA_Descriptor_t::@2	xfer	TRANSFER DMA descriptor, this is the only descriptor type which can be used to start a DMA transfer.
uint32_t	syncSet	Set bits in LDMA_CTRL.SYNCTRIG register.
uint32_t	syncClr	Clear bits in LDMA_CTRL.SYNCTRIG register.
uint32_t	reserved1	Reserved.
uint32_t	matchVal	Sync trigger match value.
uint32_t	matchEn	Sync trigger match enable.
uint32_t	reserved2	Reserved.
struct LDMA_Descriptor_t::@3	sync	SYNCHRONIZE DMA descriptor, used for intra channel transfer synchronization.
uint32_t	immVal	Data to be written at dstAddr.
struct LDMA_Descriptor_t::@4	wri	WRITE DMA descriptor, used for write immediate operations.

Public Attribute Documentation

structType

uint32_t LDMA_Descriptor_t::structType

Set to 0 to select XFER descriptor type.

Set to 2 to select WRITE descriptor type.

Set to 1 to select SYNC descriptor type.

reserved0

uint32_t LDMA_Descriptor_t::reserved0

Reserved.

structReq

```
uint32_t LDMA_Descriptor_t::structReq
```

DMA transfer trigger during LINKLOAD.

xferCnt

```
uint32_t LDMA_Descriptor_t::xferCnt
```

Transfer count minus one.

byteSwap

```
uint32_t LDMA_Descriptor_t::byteSwap
```

Enable byte swapping transfers.

blockSize

```
uint32_t LDMA_Descriptor_t::blockSize
```

Number of unit transfers per arbitration cycle.

doneIfs

```
uint32_t LDMA_Descriptor_t::doneIfs
```

Generate interrupt when done.

reqMode

```
uint32_t LDMA_Descriptor_t::reqMode
```

Block or cycle transfer selector.

decLoopCnt

```
uint32_t LDMA_Descriptor_t::decLoopCnt
```

Enable looped transfers.

ignoreSrec

```
uint32_t LDMA_Descriptor_t::ignoreSrec
```

Ignore single requests.

srcInc

```
uint32_t LDMA_Descriptor_t::srcInc
```

Source address increment unit size.

size

```
uint32_t LDMA_Descriptor_t::size
```

DMA transfer unit size.

dstInc

```
uint32_t LDMA_Descriptor_t::dstInc
```

Destination address increment unit size.

srcAddrMode

```
uint32_t LDMA_Descriptor_t::srcAddrMode
```

Source addressing mode.

dstAddrMode

```
uint32_t LDMA_Descriptor_t::dstAddrMode
```

Destination addressing mode.

srcAddr

```
uint32_t LDMA_Descriptor_t::srcAddr
```

DMA source address.

dstAddr

```
uint32_t LDMA_Descriptor_t::dstAddr
```

DMA destination address.

DMA write destination address.

linkMode

```
uint32_t LDMA_Descriptor_t::linkMode
```

Select absolute or relative link address.

link

```
uint32_t LDMA_Descriptor_t::link
```

Enable LINKLOAD when transfer is done.

linkAddr

```
int32_t LDMA_Descriptor_t::linkAddr
```

Address of next (linked) descriptor.

xfer

```
struct LDMA_Descriptor_t::@2 LDMA_Descriptor_t::xfer
```

TRANSFER DMA descriptor, this is the only descriptor type which can be used to start a DMA transfer.

syncSet

```
uint32_t LDMA_Descriptor_t::syncSet
```

Set bits in LDMA_CTRL.SYNCTRIG register.

syncClr

```
uint32_t LDMA_Descriptor_t::syncClr
```

Clear bits in LDMA_CTRL.SYNCTRIG register.

reserved1

```
uint32_t LDMA_Descriptor_t::reserved1
```

Reserved.

matchVal

```
uint32_t LDMA_Descriptor_t::matchVal
```

Sync trigger match value.

matchEn

```
uint32_t LDMA_Descriptor_t::matchEn
```

Sync trigger match enable.

reserved2

```
uint32_t LDMA_Descriptor_t::reserved2
```

Reserved.

sync

```
struct LDMA_Descriptor_t::@3 LDMA_Descriptor_t::sync
```

SYNCHRONIZE DMA descriptor, used for intra channel transfer synchronization.

immVal

```
uint32_t LDMA_Descriptor_t::immVal
```

Data to be written at dstAddr.

wri

```
struct LDMA_Descriptor_t::@4 LDMA_Descriptor_t::wri
```

WRITE DMA descriptor, used for write immediate operations.

LDMA_Init_t

LDMA initialization configuration structure.

Public Attributes

uint8_t	ldmaInitCtrlNumFixed Arbitration mode separator.
uint8_t	ldmaInitCtrlSyncPrsClrEn PRS Synctrig clear enable.
uint8_t	ldmaInitCtrlSyncPrsSetEn PRS Synctrig set enable.
uint8_t	ldmaInitIrqPriority LDMA IRQ priority (0..7).

Public Attribute Documentation

ldmaInitCtrlNumFixed

uint8_t LDMA_Init_t::ldmaInitCtrlNumFixed

Arbitration mode separator.

ldmaInitCtrlSyncPrsClrEn

uint8_t LDMA_Init_t::ldmaInitCtrlSyncPrsClrEn

PRS Synctrig clear enable.

ldmaInitCtrlSyncPrsSetEn

uint8_t LDMA_Init_t::ldmaInitCtrlSyncPrsSetEn

PRS Synctrig set enable.

ldmaInitIrqPriority

uint8_t LDMA_Init_t::ldmaInitIrqPriority

LDMA IRQ priority (0..7).

LDMA_TransferCfg_t

DMA transfer configuration structure.

This structure configures all aspects of a DMA transfer.

Public Attributes

uint32_t	IdmaReqSel	Selects DMA trigger source.
uint8_t	IdmaCtrlSyncPrsClrOff	PRS Synctrig clear enables to clear.
uint8_t	IdmaCtrlSyncPrsClrOn	PRS Synctrig clear enables to set.
uint8_t	IdmaCtrlSyncPrsSetOff	PRS Synctrig set enables to clear.
uint8_t	IdmaCtrlSyncPrsSetOn	PRS Synctrig set enables to set.
bool	IdmaReqDis	Mask the PRS trigger input.
bool	IdmaDbgHalt	Dis.
LDMA_CfgArbSlots_t	IdmaCfgArbSlots	Arbitration slot number.
LDMA_CfgSrcIncSign_t	IdmaCfgSrcIncSign	Source address increment sign.
LDMA_CfgDstIncSign_t	IdmaCfgDstIncSign	Destination address increment sign.
uint8_t	IdmaLoopCnt	Counter for looped transfers.

Public Attribute Documentation

IdmaReqSel

```
uint32_t LDMA_TransferCfg_t::IdmaReqSel
```

Selects DMA trigger source.

IdmaCtrlSyncPrsClrOff

```
uint8_t LDMA_TransferCfg_t::IdmaCtrlSyncPrsClrOff
```

PRS Synctrig clear enables to clear.

IdmaCtrlSyncPrsClrOn

```
uint8_t LDMA_TransferCfg_t::IdmaCtrlSyncPrsClrOn
```

PRS Synctrig clear enables to set.

IdmaCtrlSyncPrsSetOff

```
uint8_t LDMA_TransferCfg_t::IdmaCtrlSyncPrsSetOff
```

PRS Synctrig set enables to clear.

IdmaCtrlSyncPrsSetOn

```
uint8_t LDMA_TransferCfg_t::IdmaCtrlSyncPrsSetOn
```

PRS Synctrig set enables to set.

IdmaReqDis

```
bool LDMA_TransferCfg_t::IdmaReqDis
```

Mask the PRS trigger input.

IdmaDbgHalt

```
bool LDMA_TransferCfg_t::IdmaDbgHalt
```

Dis.

DMA trig when CPU is halted.

IdmaCfgArbSlots

```
LDMA_CfgArbSlots_t LDMA_TransferCfg_t::IdmaCfgArbSlots
```

Arbitration slot number.

IdmaCfgSrcIncSign

```
LDMA_CfgSrcIncSign_t LDMA_TransferCfg_t::ldmaCfgSrcIncSign
```

Source address increment sign.

ldmaCfgDstIncSign

```
LDMA_CfgDstIncSign_t LDMA_TransferCfg_t::ldmaCfgDstIncSign
```

Destination address increment sign.

ldmaLoopCnt

```
uint8_t LDMA_TransferCfg_t::ldmaLoopCnt
```

Counter for looped transfers.

LETIMER - Low Energy Timer

LETIMER - Low Energy Timer

Low Energy Timer (LETIMER) Peripheral API.

This module contains functions to control the LETIMER peripheral of Silicon Labs 32-bit MCUs and SoCs. The LETIMER is a down-counter that can keep track of time and output configurable waveforms.

Modules

[LETIMER_Init_TypeDef](#)

Enumerations

```
enum    LETIMER_RepeatMode_TypeDef {
    letimerRepeatFree = _LETIMER_CTRL_REPMODE_FREE
    letimerRepeatOneshot = _LETIMER_CTRL_REPMODE_ONESHOT
    letimerRepeatBuffered = _LETIMER_CTRL_REPMODE_BUFFERED
    letimerRepeatDouble = _LETIMER_CTRL_REPMODE_DOUBLE
}
Repeat mode.
```

```
enum    LETIMER_UFOA_TypeDef {
    letimerUFOANone = _LETIMER_CTRL_UFOA0_NONE
    letimerUFOAToggle = _LETIMER_CTRL_UFOA0_TOGGLE
    letimerUFOAPulse = _LETIMER_CTRL_UFOA0_PULSE
    letimerUFOAPwm = _LETIMER_CTRL_UFOA0_PWM
}
Underflow action on output.
```

Functions

```
uint32_t LETIMER_CompareGet(LETIMER_TypeDef *letimer, unsigned int comp)
Get the LETIMER compare register value.
```

```
void LETIMER_CompareSet(LETIMER_TypeDef *letimer, unsigned int comp, uint32_t value)
Set the LETIMER compare register value.
```

```
uint32_t LETIMER_CounterGet(LETIMER_TypeDef *letimer)
Get LETIMER counter value.
```

```
void LETIMER_CounterSet(LETIMER_TypeDef *letimer, uint32_t value)
Set LETIMER counter value.
```

```
void LETIMER_Enable(LETIMER_TypeDef *letimer, bool enable)
Start/stop LETIMER.
```

```
void LETIMER_Init(LETIMER_TypeDef *letimer, const LETIMER_Init_TypeDef *init)
Initialize LETIMER.
```

```
void LETIMER_IntClear(LETIMER_TypeDef *letimer, uint32_t flags)
Clear one or more pending LETIMER interrupts.
```

```
void LETIMER_IntDisable(LETIMER_TypeDef *letimer, uint32_t flags)
Disable one or more LETIMER interrupts.
```

void	LETIMER_IntEnable (LETIMER_TypeDef *letimer, uint32_t flags) Enable one or more LETIMER interrupts.
uint32_t	LETIMER_IntGet (LETIMER_TypeDef *letimer) Get pending LETIMER interrupt flags.
uint32_t	LETIMER_IntGetEnabled (LETIMER_TypeDef *letimer) Get enabled and pending LETIMER interrupt flags.
void	LETIMER_IntSet (LETIMER_TypeDef *letimer, uint32_t flags) Set one or more pending LETIMER interrupts from SW.
uint32_t	LETIMER_RepeatGet (LETIMER_TypeDef *letimer, unsigned int rep) Get the LETIMER repeat register value.
void	LETIMER_RepeatSet (LETIMER_TypeDef *letimer, unsigned int rep, uint32_t value) Set the LETIMER repeat counter register value.
void	LETIMER_Reset (LETIMER_TypeDef *letimer) Reset LETIMER to the same state that it was in after a hardware reset.
void	LETIMER_SyncWait (LETIMER_TypeDef *letimer) Wait for the LETIMER to complete all synchronization of register changes and commands.
void	LETIMER_TopSet (LETIMER_TypeDef *letimer, uint32_t value) Set the LETIMER top value.
uint32_t	LETIMER_TopGet (LETIMER_TypeDef *letimer) Get the current LETIMER top value.

Macros

#define	LETIMER_INIT_DEFAULT undefined Default configuration for LETIMER initialization structure.
---------	--

Enumeration Documentation

LETIMER_RepeatMode_TypeDef

LETIMER_RepeatMode_TypeDef	
Repeat mode.	
Enumerator	
letimerRepeatFree	Count until stopped by SW.
letimerRepeatOneshot	Count REPO times.
letimerRepeatBuffered	Count REPO times, if REP1 has been written to, it is loaded into REPO when REPO is about to be decremented to 0.
letimerRepeatDouble	Run as long as both REPO and REP1 are not 0.

LETIMER_UFOA_TypeDef

LETIMER_UFOA_TypeDef	
Underflow action on output.	
Enumerator	
letimerUFOANone	No output action.

letimerUFOAToggle	Toggle output when counter underflows.
letimerUFOAPulse	Hold output one LETIMER clock cycle when counter underflows.
letimerUFOAPwm	Set output idle when counter underflows, and active when matching COMP1.

Function Documentation

LETIMER_CompareGet

```
uint32_t LETIMER_CompareGet (LETIMER_TypeDef * letimer, unsigned int comp)
```

Get the LETIMER compare register value.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.
unsigned int	[in]	comp	A compare register to get, either 0 or 1.

Returns

- A compare register value, 0 if invalid register selected.

LETIMER_CompareSet

```
void LETIMER_CompareSet (LETIMER_TypeDef * letimer, unsigned int comp, uint32_t value)
```

Set the LETIMER compare register value.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.
unsigned int	[in]	comp	A compare register to set, either 0 or 1.
uint32_t	[in]	value	An initialization value (<= 0x0000ffff).

Note

- The setting of a compare register requires synchronization into the low frequency domain. If the same register is modified before a previous update has completed, this function will stall until the previous synchronization has completed. This only applies to the Gecko Family. See comments in the LETIMER_Sync() internal function call.

LETIMER_CounterGet

```
uint32_t LETIMER_CounterGet (LETIMER_TypeDef * letimer)
```

Get LETIMER counter value.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to the LETIMER peripheral register block.

Returns

- Current LETIMER counter value.

LETIMER_CounterSet

```
void LETIMER_CounterSet (LETIMER_TypeDef * letimer, uint32_t value)
```

Set LETIMER counter value.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to the LETIMER peripheral register block.
uint32_t	[in]	value	New counter value.

LETIMER_Enable

```
void LETIMER_Enable (LETIMER_TypeDef * letimer, bool enable)
```

Start/stop LETIMER.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.
bool	[in]	enable	True to enable counting, false to disable.

Note

- The enabling/disabling of the LETIMER modifies the LETIMER CMD register which requires synchronization into the low-frequency domain. If this register is modified before a previous update to the same register has completed, this function will stall until the previous synchronization has completed. This only applies to the Gecko Family. See comments in the `LETIMER_Sync()` internal function call.

LETIMER_Init

```
void LETIMER_Init (LETIMER_TypeDef * letimer, const LETIMER_Init_TypeDef * init)
```

Initialize LETIMER.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.
const LETIMER_Init_TypeDef *	[in]	init	A pointer to the LETIMER initialization structure.

Note that the compare/repeat values must be set separately with [LETIMER_CompareSet\(\)](#) and [LETIMER_RepeatSet\(\)](#). That should probably be done prior using this function if configuring the LETIMER to start when initialization is complete.

Note

- The initialization of the LETIMER modifies the LETIMER CTRL/CMD registers which require synchronization into the low-frequency domain. If any of those registers are modified before a previous update to the same

register has completed, this function will stall until the previous synchronization has completed. This only applies to the Gecko Family. See comments in the `LETIMER_Sync()` internal function call.

LETIMER_IntClear

```
void LETIMER_IntClear (LETIMER_TypeDef * letimer, uint32_t flags)
```

Clear one or more pending LETIMER interrupts.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to LETIMER peripheral register block.
uint32_t	[in]	flags	Pending LETIMER interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for the LETIMER module (LETIMER_IF_nnn).

LETIMER_IntDisable

```
void LETIMER_IntDisable (LETIMER_TypeDef * letimer, uint32_t flags)
```

Disable one or more LETIMER interrupts.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to LETIMER peripheral register block.
uint32_t	[in]	flags	LETIMER interrupt sources to disable. Use a bitwise logic OR combination of valid interrupt flags for the LETIMER module (LETIMER_IF_nnn).

LETIMER_IntEnable

```
void LETIMER_IntEnable (LETIMER_TypeDef * letimer, uint32_t flags)
```

Enable one or more LETIMER interrupts.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to the LETIMER peripheral register block.
uint32_t	[in]	flags	LETIMER interrupt sources to enable. Use a bitwise logic OR combination of valid interrupt flags for the LETIMER module (LETIMER_IF_nnn).

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [LETIMER_IntClear\(\)](#) prior to enabling the interrupt.

LETIMER_IntGet

```
uint32_t LETIMER_IntGet (LETIMER_TypeDef * letimer)
```

Get pending LETIMER interrupt flags.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to LETIMER peripheral register block.

Note

- Event bits are not cleared by the use of this function.

Returns

- LETIMER interrupt sources pending. A bitwise logic OR combination of valid interrupt flags for the LETIMER module (LETIMER_IF_nnn).

LETIMER_IntGetEnabled

```
uint32_t LETIMER_IntGetEnabled (LETIMER_TypeDef * letimer)
```

Get enabled and pending LETIMER interrupt flags.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to LETIMER peripheral register block.

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Event bits are not cleared by the use of this function.

Returns

- Pending and enabled LETIMER interrupt sources. Return value is the bitwise AND combination of
 - the OR combination of enabled interrupt sources in LETIMER_IEN_nnn register (LETIMER_IEN_nnn) and
 - the OR combination of valid interrupt flags of the LETIMER module (LETIMER_IF_nnn).

LETIMER_IntSet

```
void LETIMER_IntSet (LETIMER_TypeDef * letimer, uint32_t flags)
```

Set one or more pending LETIMER interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to LETIMER peripheral register block.
uint32_t	[in]	flags	LETIMER interrupt sources to set to pending. Use a bitwise logic OR combination of valid interrupt flags for the LETIMER module (LETIMER_IF_nnn).

LETIMER_RepeatGet

```
uint32_t LETIMER_RepeatGet (LETIMER_TypeDef * letimer, unsigned int rep)
```

Get the LETIMER repeat register value.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.
unsigned int	[in]	rep	Repeat register to get, either 0 or 1.

Returns

- Repeat register value, 0 if invalid register selected.

LETIMER_RepeatSet

```
void LETIMER_RepeatSet (LETIMER_TypeDef * letimer, unsigned int rep, uint32_t value)
```

Set the LETIMER repeat counter register value.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.
unsigned int	[in]	rep	Repeat counter register to set, either 0 or 1.
uint32_t	[in]	value	An initialization value (<= 0x0000ffff).

Note

- The setting of a repeat counter register requires synchronization into the low-frequency domain. If the same register is modified before a previous update has completed, this function will stall until the previous synchronization has completed. This only applies to the Gecko Family. See comments in the LETIMER_Sync() internal function call.

LETIMER_Reset

```
void LETIMER_Reset (LETIMER_TypeDef * letimer)
```

Reset LETIMER to the same state that it was in after a hardware reset.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.

Note

- The ROUTE register is NOT reset by this function to allow for a centralized setup of this feature.

LETIMER_SyncWait

```
void LETIMER_SyncWait (LETIMER_TypeDef * letimer)
```

Wait for the LETIMER to complete all synchronization of register changes and commands.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.

LETIMER_TopSet

```
void LETIMER_TopSet (LETIMER_TypeDef * letimer, uint32_t value)
```

Set the LETIMER top value.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.
uint32_t	[in]	value	The top value. This can be a 16 bit value on series-0 and series-1 devices and a 24 bit value on series-2 devices.

Note

- The LETIMER is a down-counter, so when the counter reaches 0 then the top value will be loaded into the counter. This function can be used to set the top value.

If the LETIMER is not already configured to use a top value then this function will enable that functionality for the user.

LETIMER_TopGet

```
uint32_t LETIMER_TopGet (LETIMER_TypeDef * letimer)
```

Get the current LETIMER top value.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	A pointer to the LETIMER peripheral register block.

Returns

- The top value. This will be a 16 bit value on series-0 and series-1 devices and a 24 bit value on series-2 devices.

Macro Definition Documentation

LETIMER_INIT_DEFAULT

```
#define LETIMER_INIT_DEFAULT
```

Value:

```
0 {
0 true, /* Enable timer when initialization completes. */ \
0 false, /* Stop counter during debug halt. */ \
0 false, /* Do not load COMP0 into CNT on underflow. */ \
0 false, /* Do not load COMP1 into COMP0 when REP0 reaches 0. */ \
```

```
0,          /* Idle value 0 for output 0. */      \
0,          /* Idle value 0 for output 1. */      \
letimerUFOANone, /* No action on underflow on output 0. */ \
letimerUFOANone, /* No action on underflow on output 1. */ \
letimerRepeatFree, /* Count until stopped by SW. */      \
0            /* Use default top Value. */              \
}
```

Default configuration for LETIMER initialization structure.

LETIMER_Init_TypeDef

LETIMER initialization structure.

Public Attributes

bool	enable	Start counting when initialization completes.
bool	debugRun	Counter shall keep running during debug halt.
bool	comp0Top	Load COMP0 register into CNT when counter underflows.
bool	bufTop	Load COMP1 into COMP0 when REPO reaches 0.
uint8_t	out0Pol	Idle value for output 0.
uint8_t	out1Pol	Idle value for output 1.
LETIMER_UFOA_TypeDef	ufoa0	Underflow output 0 action.
LETIMER_UFOA_TypeDef	ufoa1	Underflow output 1 action.
LETIMER_Repeat_Mode_TypeDef	repMode	Repeat mode.
uint32_t	topValue	Top value.

Public Attribute Documentation

enable

```
bool LETIMER_Init_TypeDef::enable
```

Start counting when initialization completes.

debugRun

```
bool LETIMER_Init_TypeDef::debugRun
```

Counter shall keep running during debug halt.

comp0Top

```
bool LETIMER_Init_TypeDef::comp0Top
```

Load COMP0 register into CNT when counter underflows.

bufTop

```
bool LETIMER_Init_TypeDef::bufTop
```

Load COMP1 into COMP0 when REPO reaches 0.

out0Pol

```
uint8_t LETIMER_Init_TypeDef::out0Pol
```

Idle value for output 0.

out1Pol

```
uint8_t LETIMER_Init_TypeDef::out1Pol
```

Idle value for output 1.

ufoa0

```
LETIMER_UFOA_TypeDef LETIMER_Init_TypeDef::ufoa0
```

Underflow output 0 action.

ufoa1

```
LETIMER_UFOA_TypeDef LETIMER_Init_TypeDef::ufoa1
```

Underflow output 1 action.

repMode

```
LETIMER_RepeatMode_TypeDef LETIMER_Init_TypeDef::repMode
```

Repeat mode.

topValue

```
uint32_t LETIMER_Init_TypeDef::topValue
```

Top value.

Counter wraps when top value matches counter value is reached.

MSC - Memory System Controller

MSC - Memory System Controller

Memory System Controller API.

Contains functions to control the MSC, primarily the Flash. Users can perform Flash memory write and erase operations, as well as optimization of the CPU instruction fetch interface for the application. Available instruction fetch features depends on the MCU or SoC family, but features such as instruction pre-fetch, cache, and configurable branch prediction are typically available.

Note

- Flash wait-state configuration is handled by [CMU - Clock Management Unit](#). When core clock configuration is changed by a call to functions such as [CMU_ClockSelectSet\(\)](#) or [CMU_HFRCOBandSet\(\)](#), then Flash wait-state configuration is also updated.

MSC resets into a safe state. To initialize the instruction interface to recommended settings:

```
MSC_ExecConfig_TypeDef execConfig = MSC_EXECCONFIG_DEFAULT;  
MSC_ExecConfigSet(&execConfig);
```

Note

- The optimal configuration is highly application dependent. Performance benchmarking is supported by most families. See [MSC_StartCacheMeasurement\(\)](#) and [MSC_GetCacheMeasurement\(\)](#) for more details.
- The flash write and erase runs from RAM on the EFM32G devices. On all other devices the flash write and erase functions run from flash.
- Flash erase may add ms of delay to interrupt latency if executing from Flash.

Flash write and erase operations are supported by [MSC_WriteWord\(\)](#), [MSC_ErasePage\(\)](#), and [MSC_MassErase\(\)](#). Mass erase is supported for MCU and SoC families with larger Flash sizes.

Note

- [MSC_Init\(\)](#) must be called prior to any Flash write or erase operation.

The following steps are necessary to perform a page erase and write:

```
uint32_t * userDataPage = (uint32_t *) USERDATA_BASE;  
uint32_t userData[] = {  
    0x01020304,  
    0x05060708  
};  
MSC_ErasePage(userDataPage);  
MSC_WriteWord(userDataPage, userData, sizeof(userData));
```

Note

- The configuration [EM_MSC_RUN_FROM_RAM](#) is used to allocate the flash write functions in RAM. By default, flash write functions are placed in RAM on EFM32G and Series 2 devices unless [SL_RAMFUNC_DISABLE](#) is defined. For other devices, flash write functions are placed in FLASH by default unless [EM_MSC_RUN_FROM_RAM](#) is defined and [SL_RAMFUNC_DISABLE](#) is not defined.

Modules

[MSC_ExecConfig_TypeDef](#)
[MSC_EccConfig_TypeDef](#)

Enumerations

```
enum MSC\_Status\_TypeDef {
    mscReturnOk = 0
    mscReturnInvalidAddr = -1
    mscReturnLocked = -2
    mscReturnTimeOut = -3
    mscReturnUnaligned = -4
}
```

Return codes for writing/erasing Flash.

Functions

```
bool MSC\_LockGetLocked(void)
    Get the status of the MSC register lock.

void MSC\_LockSetLocked(void)
    Set the MSC register lock to a locked state.

void MSC\_LockSetUnlocked(void)
    Set the MSC register lock to an unlocked state.

uint32_t MSC\_ReadCTRLGet(void)
    Get the current value of the read control register (MSC_READCTRL).

void MSC\_ReadCTRLSet(uint32_t value)
    Write a value to the read control register (MSC_READCTRL).

void MSC\_PageLockSetLocked(uint32_t page_number)
    Set the lockbit for a flash page in order to prevent page writes/erases to the corresponding page.

bool MSC\_PageLockGetLocked(uint32_t page_number)
    Get the value of the lockbit for a flash page.

uint32_t MSC\_UserDataGetSize(void)
    Get the size of the user data region in flash.

uint32_t MSC\_MiscLockWordGet(void)
    Get the current value of the mass erase and user data page lock word (MSC_MISCLOCKWORD).

void MSC\_MiscLockWordSet(uint32_t value)
    Write a value to the mass erase and user data page lock word (MSC_MISCLOCKWORD).

void MSC\_IntClear(uint32_t flags)
    Clear one or more pending MSC interrupts.

void MSC\_IntDisable(uint32_t flags)
    Disable one or more MSC interrupts.

void MSC\_IntEnable(uint32_t flags)
    Enable one or more MSC interrupts.

uint32_t MSC\_IntGet(void)
    Get pending MSC interrupt flags.

uint32_t MSC\_IntGetEnabled(void)
    Get enabled and pending MSC interrupt flags.

void MSC\_IntSet(uint32_t flags)
    Set one or more pending MSC interrupts from SW.

void MSC\_ExecConfigSet(MSC_ExecConfig_TypeDef *execConfig)
    Set MSC code execution configuration.
```

void	MSC_EccConfigSet (MSC_EccConfig_TypeDef *eccConfig) Configure Error Correcting Code (ECC).
SL_RAMFUNC_DECLARE MSC_Status_TypeDef	MSC_MassErase (void) Erase the entire Flash in one operation.
MSC_RAMFUNC_DECLARE MSC_Status_TypeDef	MSC_ErasePage (uint32_t *startAddress) Erases a page in flash memory.
MSC_RAMFUNC_DECLARE MSC_Status_TypeDef	MSC_WriteWord (uint32_t *address, void const *data, uint32_t numBytes) Writes data to flash memory.
MSC_Status_TypeDef	MSC_WriteWordDma (int ch, uint32_t *address, const void *data, uint32_t numBytes) Writes data to flash memory using the DMA.
void	MSC_Init (void) Initialize MSC module.
void	MSC_Deinit (void) Turn off MSC flash write enable and lock MSC registers.
void	mscEccReadWriteExistingPio (const MSC_EccBank_TypeDef *eccBank) Read and write existing values in RAM (for ECC initialization).
void	mscEccBankInit (const MSC_EccBank_TypeDef *eccBank, uint32_t dmaChannels[2]) Initialize ECC for a given memory bank.
void	mscEccBankDisable (const MSC_EccBank_TypeDef *eccBank) Disable ECC for a given memory bank.

Macros

#define	MSC_PROGRAM_TIMEOUT 10000000UL Timeout used while waiting for Flash to become ready after a write.
#define	MSC_EXECCONFIG_DEFAULT undefined Default MSC ExecConfig initialization.
#define	MSC_ECC_BANKS (1) Series 2 chips incorporate 1 memory bank including ECC support.
#define	MSC_ECCCONFIG_DEFAULT undefined Default MSC EccConfig initialization.

Enumeration Documentation

MSC_Status_TypeDef

MSC_Status_TypeDef	
Return codes for writing/erasing Flash.	
Enumerator	
mscReturnOk	Flash write/erase successful.
mscReturnInvalidAddr	Invalid address.
mscReturnLocked	Flash address is locked.
mscReturnTimeOut	Timeout while writing to Flash.
mscReturnUnaligned	Unaligned access to Flash.

Function Documentation

MSC_LockGetLocked

```
bool MSC_LockGetLocked (void )
```

Get the status of the MSC register lock.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Boolean true if register lock is applied, false otherwise.

MSC_LockSetLocked

```
void MSC_LockSetLocked (void )
```

Set the MSC register lock to a locked state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

MSC_LockSetUnlocked

```
void MSC_LockSetUnlocked (void )
```

Set the MSC register lock to an unlocked state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

MSC_ReadCTRLGet

```
uint32_t MSC_ReadCTRLGet (void )
```

Get the current value of the read control register (MSC_READCTRL).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The 32-bit value read from the MSC_READCTRL register.

MSC_ReadCTRLSet

```
void MSC_ReadCTRLSet (uint32_t value)
```

Write a value to the read control register (MSC_READCTRL).

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	value	The 32-bit value to write to the MSC_READCTRL register.

MSC_PageLockSetLocked

```
void MSC_PageLockSetLocked (uint32_t page_number)
```

Set the lockbit for a flash page in order to prevent page writes/erases to the corresponding page.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	page_number	The index of the page to apply the pagelock to. Must be in the range [0, (flash_size / page_size) - 1].

MSC_PageLockGetLocked

```
bool MSC_PageLockGetLocked (uint32_t page_number)
```

Get the value of the lockbit for a flash page.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	page_number	The index of the page to get the lockbit value from. Must be in the range [0, (flash_size / page_size) - 1].

Returns

- Boolean true if the page is locked, false otherwise.

MSC_UserDataGetSize

```
uint32_t MSC_UserDataGetSize (void )
```

Get the size of the user data region in flash.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The size of the user data region divided by 256.

MSC_MiscLockWordGet

```
uint32_t MSC_MiscLockWordGet (void )
```

Get the current value of the mass erase and user data page lock word (MSC_MISCLOCKWORD).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The 32-bit value read from the MSC_MISCLOCKWORD register.

MSC_MiscLockWordSet

```
void MSC_MiscLockWordSet (uint32_t value)
```

Write a value to the mass erase and user data page lock word (MSC_MISCLOCKWORD).

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	value	The 32-bit value to write to the MSC_MISCLOCKWORD register.

MSC_IntClear

```
void MSC_IntClear (uint32_t flags)
```

Clear one or more pending MSC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	Pending MSC interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for the MSC module (MSC_IF_nnn).

MSC_IntDisable

```
void MSC_IntDisable (uint32_t flags)
```

Disable one or more MSC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	MSC interrupt sources to disable. Use a bitwise logic OR combination of valid interrupt flags for the MSC module (MSC_IF_nnn).

MSC_IntEnable

```
void MSC_IntEnable (uint32_t flags)
```

Enable one or more MSC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	MSC interrupt sources to enable. Use a bitwise logic OR combination of valid interrupt flags for the MSC module (MSC_IF_nnn).

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [MSC_IntClear\(\)](#) prior to enabling the interrupt.

MSC_IntGet

```
uint32_t MSC_IntGet (void )
```

Get pending MSC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- The event bits are not cleared by the use of this function.

Returns

- MSC interrupt sources pending. A bitwise logic OR combination of valid interrupt flags for the MSC module (MSC_IF_nnn).

MSC_IntGetEnabled

```
uint32_t MSC_IntGetEnabled (void )
```

Get enabled and pending MSC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

- Pending and enabled MSC interrupt sources. The return value is the bitwise AND of
 - the enabled interrupt sources in MSC_IEN and
 - the pending interrupt flags MSC_IF

MSC_IntSet

```
void MSC_IntSet (uint32_t flags)
```

Set one or more pending MSC interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	MSC interrupt sources to set to pending. Use a bitwise logic OR combination of valid interrupt flags for the MSC module (MSC_IF_nnn).

MSC_ExecConfigSet

```
void MSC_ExecConfigSet (MSC_ExecConfig_TypeDef * execConfig)
```

Set MSC code execution configuration.

Parameters

Type	Direction	Argument Name	Description
MSC_ExecConfig_TypeDef *	[in]	execConfig	Code execution configuration

MSC_EccConfigSet

```
void MSC_EccConfigSet (MSC_EccConfig_TypeDef * eccConfig)
```

Configure Error Correcting Code (ECC).

Parameters

Type	Direction	Argument Name	Description
MSC_EccConfig_TypeDef *	[in]	eccConfig	ECC configuration

This function configures ECC support according to the configuration input parameter. If the user requests enabling ECC for a given RAM bank this function will initialize ECC memory (syndromes) for the bank by reading and writing the existing values in memory. I.e. all data is preserved. The initialization process runs in a critical section disallowing interrupts and thread scheduling, and will consume a considerable amount of clock cycles. Therefore the user should carefully assess where to call this function. The user can consider to increase the clock frequency in order to reduce the execution time. This function makes use of 2 DMA channels to move data to/from RAM in an efficient way. The user can select which 2 DMA channels to use in order to avoid conflicts with the application. However the user must make sure that no other DMA operations takes place while this function is executing. If the application has been using the DMA controller prior to calling this function, the application will need to reinitialize DMA registers after this function has completed.

Note

- This function protects the ECC initialization procedure from interrupts and other threads by using a critical section (defined by em_core.h) When running on RTOS the user may need to override CORE_EnterCritical CORE_ExitCritical which are declared as 'SL_WEAK' in em_core.c.

MSC_MassErase

```
MSC_RAMFUNC_DEFINITION_END MSC_RAMFUNC_DEFINITION_BEGIN MSC_Status_TypeDef MSC_MassErase
(void)
```

Erase the entire Flash in one operation.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This command will erase the entire contents of the device. Use with care, both a debug session and all contents of the flash will be lost. The lock bit, MLW will prevent this operation from executing and might prevent a successful mass erase.

Returns

- Returns the status of the operation.

MSC_ErasePage

```
MSC_RAMFUNC_DEFINITION_BEGIN MSC_Status_TypeDef MSC_ErasePage (uint32_t * startAddress)
```

Erases a page in flash memory.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[in]	startAddress	Pointer to the flash page to erase. Must be aligned to beginning of page boundary.

For IAR Embedded Workbench, Simplicity Studio and GCC this will be achieved automatically by using attributes in the function proctype. For Keil uVision you must define a section called "ram_code" and place this manually in your project's scatter file.

Returns

- Returns the status of erase operation, [MSC_Status_TypeDef](#)

```
* mscReturnOk - Operation completed successfully.
* mscReturnInvalidAddr - Operation tried to erase a non-flash area.
* flashReturnLocked - MSC registers are locked or the operation tried to
*                     erase a locked area of the flash.
* flashReturnTimeOut - Operation timed out.
*
```

MSC_WriteWord

```
MSC_RAMFUNC_DEFINITION_END MSC_RAMFUNC_DEFINITION_BEGIN MSC_Status_TypeDef MSC_WriteWord
(uint32_t * address, void const * data, uint32_t numBytes)
```

Writes data to flash memory.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[in]	address	Pointer to the flash word to write to. Must be aligned to words.
void const *	[in]	data	Data to write to flash.
uint32_t	[in]	numBytes	Number of bytes to write to flash. NB: Must be divisible by four.

Write data must be aligned to words and contain a number of bytes that is divisible by four. Note

- It is recommended to erase the flash page before performing a write.

For IAR Embedded Workbench, Simplicity Studio and GCC this will be achieved automatically by using attributes in the function proctype. For Keil uVision you must define a section called "ram_code" and place this manually in your project's scatter file.

The Flash memory is organized into 64-bit wide double-words. Each 64-bit double-word can be written only twice using burst write operation between erasing cycles. The user's application must store data in RAM to sustain burst write operation.

EFR32XG21 RevC is not able to program every word twice before the next erase.

Returns

- Returns the status of the write operation, [MSC_Status_TypeDef](#)

```
* flashReturnOk - Operation completed successfully.
* flashReturnInvalidAddr - Operation tried to write to a non-flash area.
* flashReturnLocked - MSC registers are locked or the operation tried to
*                     program a locked area of the flash.
* flashReturnTimeOut - Operation timed out.
*
```

MSC_WriteWordDma

```
MSC_RAMFUNC_DEFINITION_END MSC_Status_TypeDef MSC_WriteWordDma (int ch, uint32_t * address, const
void * data, uint32_t numBytes)
```

Writes data to flash memory using the DMA.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	DMA channel to use
uint32_t *	[in]	address	A pointer to the flash word to write to. Must be aligned to words.
const void *	[in]	data	Data to write to flash and be aligned to words.
uint32_t	[in]	numBytes	A number of bytes to write from flash. NB: Must be divisible by four.

This function uses the LDMA to write data to the internal flash memory. This is the fastest way to write data to the flash and should be used when the application wants to achieve write speeds like they are reported in the datasheet. Note that copying data from flash to flash will be slower than copying from RAM to flash. So the source data must be in RAM in order to see the write speeds similar to the datasheet numbers.

Note

- This function requires that the LDMA and LDMAxBAR clock is enabled.

Returns

- Returns the status of the write operation.

```

* flashReturnOk - The operation completed successfully.
* flashReturnInvalidAddr - The operation tried to erase a non-flash area.
*

```

MSC_Init

```
void MSC_Init (void )
```

Initialize MSC module.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Puts MSC hw in a known state.

MSC_Deinit

```
void MSC_Deinit (void )
```

Turn off MSC flash write enable and lock MSC registers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

msecEccReadWriteExistingPio

```
static void msecEccReadWriteExistingPio (const MSC_EccBank_Typedef * eccBank)
```

Read and write existing values in RAM (for ECC initialization).

Parameters

Type	Direction	Argument Name	Description
const MSC_EccBank_Typedef *	[in]	eccBank	Pointer to ECC RAM bank (MSC_EccBank_Typedef)

This function uses core to load and store the existing data values in the given RAM bank.

msecEccBankInit

```
static void msecEccBankInit (const MSC_EccBank_Typedef * eccBank, uint32_t dmaChannels)
```

Initialize ECC for a given memory bank.

Parameters

Type	Direction	Argument Name	Description
const MSC_EccBank_Typedef *	[in]	eccBank	ECC memory bank device structure.
uint32_t	[in]	dmaChannels	Array of 2 DMA channels that may be used during ECC initialization.

This function initializes ECC for a given memory bank which is specified with the MSC_EccBank_Typedef structure input parameter.

mscEccBankDisable

```
static void mscEccBankDisable (const MSC_EccBank_Typedef * eccBank)
```

Disable ECC for a given memory bank.

Parameters

Type	Direction	Argument Name	Description
const MSC_EccBank_Typedef *	[in]	eccBank	ECC memory bank device structure.

This function disables ECC for a given memory bank which is specified with the MSC_EccBank_Typedef structure input parameter.

Macro Definition Documentation

MSC_PROGRAM_TIMEOUT

```
#define MSC_PROGRAM_TIMEOUT
```

Value:

```
100000000UL
```

Timeout used while waiting for Flash to become ready after a write.

This number indicates the number of iterations to perform before issuing a timeout.

Note

- Timeout is set very large (in the order of 100x longer than necessary). This is to avoid any corner case.

MSC_EXECONFIG_DEFAULT

```
#define MSC_EXECONFIG_DEFAULT
```

Value:

```
{ false, \
```

Default MSC ExecConfig initialization.

MSC_ECC_BANKS

```
#define MSC_ECC_BANKS
```

Value:

```
(1)
```

Series 2 chips incorporate 1 memory bank including ECC support.

MSC_ECCCONFIG_DEFAULT

```
#define MSC_ECCCONFIG_DEFAULT
```

Value:

```
0 | {  
0 |   { false },  
0 |   { 0, 1 },  
0 | }
```

Default MSC EccConfig initialization.

MSC_ExecConfig_TypeDef

Code execution configuration.

Public Attributes

bool [doutBufEn](#)
Flash dout pipeline buffer enable.

Public Attribute Documentation

doutBufEn

bool MSC_ExecConfig_TypeDef::doutBufEn

Flash dout pipeline buffer enable.

MSC_EccConfig_TypeDef

ECC configuration.

Public Attributes

bool	enableEccBank	Array of bools to enable/disable Error Correcting Code (ECC) for each RAM bank that supports ECC on the device.
uint32_t	dmaChannels	Array of 2 DMA channel numbers to use for ECC initialization.

Public Attribute Documentation

enableEccBank

```
bool MSC_EccConfig_TypeDef::enableEccBank[MSC_ECC_BANKS]
```

Array of bools to enable/disable Error Correcting Code (ECC) for each RAM bank that supports ECC on the device.

dmaChannels

```
uint32_t MSC_EccConfig_TypeDef::dmaChannels[2]
```

Array of 2 DMA channel numbers to use for ECC initialization.

PDM - Pulse Density Modulation

PDM - Pulse Density Modulation

Pulse Density Modulation (PDM) peripheral API.

PDM API functions provide full support for the PDM peripheral. The PDM peripheral accepts PDM bitstreams and produces PCM encoded output.

The following is an example PDM usage when interfacing to two PDM microphones:

Configure clocks and GPIO pins:

```
PDM_Init_TypeDef      pdmInit = PDM_INIT_DEFAULT;
CMU_DPLLInit_TypeDef pllInit = CMU_DPLL_LFX0_TO_40MHZ;

CMU_OscillatorEnable(cmuOsc_LFX0, true, true);
// Lock PLL to 1,411,209 Hz to achieve 44,100 kHz PCM sampling rate
// when using 32x PDM oversampling
pllInit.frequency = 1411209;
pllInit.m = 14;
pllInit.n = 645;
CMU_DPLLLock(&pllInit);

// Setup all GPIO's.
GPIO_PinModeSet(MIC_CLK_PORT, MIC_CLK_PIN, gpioModePushPull, 0);
GPIO_PinModeSet(MIC_DATA_PORT, MIC_DATA_PIN, gpioModeInput, 0);

// Set fast slew rate on PDM mic CLK and DATA pins
GPIO_SlewrateSet(MIC_CLK_PORT, 7U, 7U);

// Enable PDM peripheral clock.
CMU_ClockEnable(cmuClock_PDM, true);
// Select PDM reference clock source and enable it.
CMU_ClockSelectSet(cmuClock_PDMREF, cmuSelect_HFRC0);
CMU_ClockEnable(cmuClock_PDMREF, true);

// Route PDM signals to correct GPIO's.
PDM->ROUTELOC0 = (PDM->ROUTELOC0 & ~_PDM_ROUTELOC0_DAT0LOC_MASK)
    | (MIC_DATA_PDM_LOC << _PDM_ROUTELOC0_DAT0LOC_SHIFT);
PDM->ROUTELOC1 = MIC_CLK_PDM_LOC << _PDM_ROUTELOC1_CLKLOC_SHIFT;
PDM->ROUTEPEN |= PDM_ROUTEPEN_CLKPEN | PDM_ROUTEPEN_DATOPEN;
```

Initialize and start PDM, then read PCM samples from FIFO:

```
PDM_Init_TypeDef init = PDM_INIT_DEFAULT;
PDM_Init(PDM, &init);

while (true) {
    *pBuffer++ = PDM_Rx(PDM);
}
```

Modules

[PDM_Init_TypeDef](#)

Enumerations

```
enum PDM_Ch1ClkPolarity_TypeDef {
    pdmCh1ClkPolarityRisingEdge = _PDM_CFG0_CH1CLKPOL_NORMAL
    pdmCh1ClkPolarityFallingEdge = _PDM_CFG0_CH1CLKPOL_INVERT
}
Configure CH1 CLK Polarity.
```

```
enum PDM_Ch0ClkPolarity_TypeDef {
    pdmCh0ClkPolarityRisingEdge = _PDM_CFG0_CH0CLKPOL_NORMAL
    pdmCh0ClkPolarityFallingEdge = _PDM_CFG0_CH0CLKPOL_INVERT
}
Configure CH0 CLK Polarity.
```

```
enum PDM_FifoValidWatermark_TypeDef {
    pdmFifoValidWatermarkOne = _PDM_CFG0_FIFODVL_ONE
    pdmFifoValidWatermarkTwo = _PDM_CFG0_FIFODVL_TWO
    pdmFifoValidWatermarkThree = _PDM_CFG0_FIFODVL_THREE
    pdmFifoValidWatermarkFour = _PDM_CFG0_FIFODVL_FOUR
}
Configure FIFO Data valid level water-mark.
```

```
enum PDM_DataFormat_TypeDef {
    pdmDataFormatRight16 = _PDM_CFG0_DATAFORMAT_RIGHT16
    pdmDataFormatDouble16 = _PDM_CFG0_DATAFORMAT_DOUBLE16
    pdmDataFormatRight24 = _PDM_CFG0_DATAFORMAT_RIGHT24
    pdmDataFormatFull32bit = _PDM_CFG0_DATAFORMAT_FULL32BIT
    pdmDataFormatLeft16 = _PDM_CFG0_DATAFORMAT_LEFT16
    pdmDataFormatLeft24 = _PDM_CFG0_DATAFORMAT_LEFT24
    pdmDataFormatRaw32bit = _PDM_CFG0_DATAFORMAT_RAW32BIT
}
Configure PDM filter data output format.
```

```
enum PDM_NumberOfChannels_TypeDef {
    pdmNumberOfChannelsOne = _PDM_CFG0_NUMCH_ONE
    pdmNumberOfChannelsTwo = _PDM_CFG0_NUMCH_TWO
}
Configure number of PDM channels.
```

```
enum PDM_FilterOrder_TypeDef {
    pdmFilterOrderSecond = _PDM_CFG0_FORDER_SECOND
    pdmFilterOrderThird = _PDM_CFG0_FORDER_THIRD
    pdmFilterOrderFourth = _PDM_CFG0_FORDER_FOURTH
    pdmFilterOrderFifth = _PDM_CFG0_FORDER_FIFTH
}
Configure order of the PDM filter.
```

Functions

```
void PDM_DeInit(PDM_TypeDef *pdm)
De-initialize the PDM peripheral.
```

```
void PDM_Init(PDM_TypeDef *pdm, const PDM_Init_TypeDef *init)
Initialize the PDM peripheral.
```

```
void PDM_Reset(PDM_TypeDef *pdm)
Initialize PDM registers with reset values.
```

```
void PDM_Clear(PDM_TypeDef *pdm)
Clear the PDM filter.
```

```
void PDM_FifoFlush(PDM_TypeDef *pdm)
Flush the PDM sample FIFO.
```

void	PDM_IntClear (PDM_TypeDef *pdm, uint32_t flags) Clear one or more pending PDM interrupts.
void	PDM_IntDisable (PDM_TypeDef *pdm, uint32_t flags) Disable one or more PDM interrupts.
void	PDM_IntEnable (PDM_TypeDef *pdm, uint32_t flags) Enable one or more PDM interrupts.
uint32_t	PDM_IntGet (PDM_TypeDef *pdm) Get pending PDM interrupt flags.
uint32_t	PDM_IntGetEnabled (PDM_TypeDef *pdm) Get enabled and pending PDM interrupt flags.
void	PDM_IntSet (PDM_TypeDef *pdm, uint32_t flags) Set one or more pending PDM interrupts.
uint32_t	PDM_Rx (PDM_TypeDef *pdm) Read one entry from the PDM FIFO.
void	PDM_Start (PDM_TypeDef *pdm) Start the PDM operation (start the PDM filter).
uint32_t	PDM_StatusGet (PDM_TypeDef *pdm) Get the PDM STATUS register.
void	PDM_Stop (PDM_TypeDef *pdm) Stop the PDM operation (stop the PDM filter).

Macros

#define	PDM_INIT_DEFAULT undefined Default configuration for PDM.
---------	---

Enumeration Documentation

PDM_Ch1ClkPolarity_Typedef

PDM_Ch1ClkPolarity_Typedef	
Configure CH1 CLK Polarity.	
Enumerator	
pdmCh1ClkPolarityRisingEdge	Input data clocked on rising clock edge.
pdmCh1ClkPolarityFallingEdge	Input data clocked on falling clock edge.

PDM_Ch0ClkPolarity_Typedef

PDM_Ch0ClkPolarity_Typedef	
Configure CH0 CLK Polarity.	
Enumerator	
pdmCh0ClkPolarityRisingEdge	Input data clocked on rising clock edge.
pdmCh0ClkPolarityFallingEdge	Input data clocked on falling clock edge.

PDM_FifoValidWatermark_Typedef

PDM_FifoValidWatermark_TypeDef

Configure FIFO Data valid level water-mark.

Enumerator	
pdmFifoValidWatermarkOne	At least one word.
pdmFifoValidWatermarkTwo	Two words.
pdmFifoValidWatermarkThree	Three words.
pdmFifoValidWatermarkFour	Four words.

PDM_DataFormat_TypeDef

PDM_DataFormat_TypeDef

Configure PDM filter data output format.

Enumerator	
pdmDataFormatRight16	Right aligned 16-bit, left bits are sign extended.
pdmDataFormatDouble16	Pack two 16-bit samples into one 32-bit word.
pdmDataFormatRight24	Right aligned 24bit, left bits are sign extended.
pdmDataFormatFull32bit	32 bit data.
pdmDataFormatLeft16	Left aligned 16-bit, right bits are zeros.
pdmDataFormatLeft24	Left aligned 24-bit, right bits are zeros.
pdmDataFormatRaw32bit	RAW 32 bit data from integrator.

PDM_NumberOfChannels_TypeDef

PDM_NumberOfChannels_TypeDef

Configure number of PDM channels.

Enumerator	
pdmNumberOfChannelsOne	Only one Channel.
pdmNumberOfChannelsTwo	Two Channels.

PDM_FilterOrder_TypeDef

PDM_FilterOrder_TypeDef

Configure order of the PDM filter.

Enumerator	
pdmFilterOrderSecond	Second order filter.
pdmFilterOrderThird	Third order filter.
pdmFilterOrderFourth	Fourth order filter.
pdmFilterOrderFifth	Fifth order filter.

Function Documentation

PDM_DeInit

```
void PDM_DeInit (PDM_TypeDef * pdm)
```

De-initialize the PDM peripheral.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

This function will stop the PDM filter and set PDM control registers to their reset values.

PDM_Init

```
void PDM_Init (PDM_TypeDef * pdm, const PDM_Init_TypeDef * init)
```

Initialize the PDM peripheral.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.
const PDM_Init_TypeDef *	[in]	init	A pointer to the initialization structure used to configure the PDM.

This function will configure basic settings in PDM according to values in the initialization data structure.

Notice that enabling of PDM clock, setup of PDM pins and setup of PRS is not covered by this function.

PDM_Reset

```
void PDM_Reset (PDM_TypeDef * pdm)
```

Initialize PDM registers with reset values.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

PDM_Clear

```
void PDM_Clear (PDM_TypeDef * pdm)
```

Clear the PDM filter.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

PDM_FifoFlush

```
void PDM_FifoFlush (PDM_TypeDef * pdm)
```

Flush the PDM sample FIFO.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

PDM_IntClear

```
void PDM_IntClear (PDM_TypeDef * pdm, uint32_t flags)
```

Clear one or more pending PDM interrupts.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.
uint32_t	[in]	flags	Pending PDM interrupt sources to clear. Use one or more valid interrupt flags for the PDM module. The flags are PDM_IFC_DV, PDM_IFC_DVL, PDM_IFC_OF and PDM_IFC_UF.

PDM_IntDisable

```
void PDM_IntDisable (PDM_TypeDef * pdm, uint32_t flags)
```

Disable one or more PDM interrupts.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.
uint32_t	[in]	flags	PDM interrupt sources to disable. Use one or more valid interrupt flags for the PDM module. The flags are PDM_IEN_DV, PDM_IEN_DVL, PDM_IEN_OF and PDM_IEN_UF.

PDM_IntEnable

```
void PDM_IntEnable (PDM_TypeDef * pdm, uint32_t flags)
```

Enable one or more PDM interrupts.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	PDM interrupt sources to enable. Use one or more valid interrupt flags for the PDM module. The flags are PDM_IEN_DV, PDM_IEN_DVL, PDM_IEN_OF and PDM_IEN_UF.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [PDM_IntClear\(\)](#) prior to enabling the interrupt.

PDM_IntGet

```
uint32_t PDM_IntGet (PDM_TypeDef * pdm)
```

Get pending PDM interrupt flags.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

Note

- Event bits are not cleared by the use of this function.

Returns

- PDM interrupt sources pending. Returns one or more valid interrupt flags for PDM module. The flags are PDM_IF_DV, PDM_IF_DVL, PDM_IF_OF and PDM_IF_UF.

PDM_IntGetEnabled

```
uint32_t PDM_IntGetEnabled (PDM_TypeDef * pdm)
```

Get enabled and pending PDM interrupt flags.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

- Pending and enabled PDM interrupt sources Return value is the bitwise AND of
 - the enabled interrupt sources in PDM_IEN and
 - the pending interrupt flags PDM_IF

PDM_IntSet

```
void PDM_IntSet (PDM_TypeDef * pdm, uint32_t flags)
```

Set one or more pending PDM interrupts.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.
uint32_t	[in]	flags	PDM interrupt sources to set to pending. Use one or more valid interrupt flags for the PDM module. The flags are PDM_IFS_DV, PDM_IFS_DVL, PDM_IFS_OF and PDM_IFS_UF.

PDM_Rx

```
uint32_t PDM_Rx (PDM_TypeDef * pdm)
```

Read one entry from the PDM FIFO.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

Note

- This function will wait until a sample is available in the FIFO. Depending on PDM configuration, a FIFO entry can consist of one or two samples.

Returns

- The entry read from the FIFO.

PDM_Start

```
void PDM_Start (PDM_TypeDef * pdm)
```

Start the PDM operation (start the PDM filter).

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

PDM_StatusGet

```
uint32_t PDM_StatusGet (PDM_TypeDef * pdm)
```

Get the PDM STATUS register.

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

Returns

-

STATUS register value.

PDM_Stop

```
void PDM_Stop (PDM_TypeDef * pdm)
```

Stop the PDM operation (stop the PDM filter).

Parameters

Type	Direction	Argument Name	Description
PDM_TypeDef *	[in]	pdm	A pointer to the PDM peripheral register block.

Macro Definition Documentation

PDM_INIT_DEFAULT

```
#define PDM_INIT_DEFAULT
```

Value:

```
0 {
0     true,                /* Start PDM filter after initialization. */ \
0     32U,                 /* PDM down sampling rate. */ \
0     5U,                  /* PDM gain. */ \
0     pdmCh1ClkPolarityFallingEdge, /* Input data clocked on falling clock edge. */ \
0     pdmCh0ClkPolarityRisingEdge, /* Input data clocked on rising clock edge. */ \
0     true,                /* Enable stereo mode for channel pair CH0 and CH1. */ \
0     pdmFifoValidWatermarkOne, /* At least one word water-mark level. */ \
0     pdmDataFormatDouble16, /* Two 16-bit samples per FIFO entry. */ \
0     pdmNumberOfChannelsTwo, /* Two Channels. */ \
0     pdmFilterOrderFifth, /* Fifth order filter. */ \
0     0U,                  /* No clock prescaling. */ \
0 }
```

Default configuration for PDM.

Stereo Ch0/1, 16bit samples, 44,100 Hz sampling rate, 32 times oversampling (requires 1,411,209 Hz PDM clock).

PDM_Init_TypeDef

PDM initialization structure.

Public Attributes

bool	start	Start PDM filter after initialization.
uint32_t	dsr	PDM down sampling rate.
uint32_t	gain	PDM gain.
PDM_Ch1ClkPolarity_TypeDef	ch1ClkPolarity	Ch 1 clock polarity.
PDM_Ch0ClkPolarity_TypeDef	ch0ClkPolarity	Ch 0 clock polarity.
bool	enableCh0Ch1Stereo	Enable stereo mode for channel pair CH0 and CH1.
PDM_FifoValidWatermark_TypeDef	fifoValidWatermark	FIFO Data valid level water-mark.
PDM_DataFormat_TypeDef	dataFormat	PDM filter data output format.
PDM_NumberOfChannels_TypeDef	numChannels	Number of PDM channels.
PDM_FilterOrder_TypeDef	filterOrder	PDM filter order.
uint32_t	prescaler	PDM clock prescaler, resulting PDM clock is input clock / (prescaler + 1).

Public Attribute Documentation

start

```
bool PDM_Init_TypeDef::start
```

Start PDM filter after initialization.

dsr

```
uint32_t PDM_Init_TypeDef::dsr
```

PDM down sampling rate.

gain

```
uint32_t PDM_Init_TypeDef::gain
```

PDM gain.

ch1ClkPolarity

```
PDM_Ch1ClkPolarity_TypeDef PDM_Init_TypeDef::ch1ClkPolarity
```

Ch 1 clock polarity.

ch0ClkPolarity

```
PDM_Ch0ClkPolarity_TypeDef PDM_Init_TypeDef::ch0ClkPolarity
```

Ch 0 clock polarity.

enableCh0Ch1Stereo

```
bool PDM_Init_TypeDef::enableCh0Ch1Stereo
```

Enable stereo mode for channel pair CH0 and CH1.

fifoValidWatermark

```
PDM_FifoValidWatermark_TypeDef PDM_Init_TypeDef::fifoValidWatermark
```

FIFO Data valid level water-mark.

dataFormat

```
PDM_DataFormat_TypeDef PDM_Init_TypeDef::dataFormat
```

PDM filter data output format.

numChannels

```
PDM_NumberOfChannels_TypeDef PDM_Init_TypeDef::numChannels
```

Number of PDM channels.

filterOrder

```
PDM_FilterOrder_TypeDef PDM_Init_TypeDef::filterOrder
```

PDM filter order.

prescaler

```
uint32_t PDM_Init_TypeDef::prescaler
```

PDM clock prescaler, resulting PDM clock is input clock / (prescaler + 1).

PRS - Peripheral Reflex System

PRS - Peripheral Reflex System

Peripheral Reflex System (PRS) Peripheral API.

This module contains functions to control the PRS peripheral of Silicon Labs 32-bit MCUs and SoCs. The PRS allows configurable, fast, and autonomous communication between peripherals on the MCU or SoC.

Enumerations

```
enum PRS_ChType_t {
    prsTypeAsync
    prsTypeSync
}
PRS Channel type.

enum PRS_Edge_TypeDef {
    prsEdgeOff
    prsEdgePos
    prsEdgeNeg
    prsEdgeBoth
}
Edge detection type.

enum PRS_Logic_t {
    prsLogic_Zero = _PRS_ASYNC_CH_CTRL_FNSEL_LOGICAL_ZERO
    prsLogic_A_NOR_B = _PRS_ASYNC_CH_CTRL_FNSEL_A_NOR_B
    prsLogic_NOT_A_AND_B = _PRS_ASYNC_CH_CTRL_FNSEL_NOT_A_AND_B
    prsLogic_NOT_A = _PRS_ASYNC_CH_CTRL_FNSEL_NOT_A
    prsLogic_A_AND_NOT_B = _PRS_ASYNC_CH_CTRL_FNSEL_A_AND_NOT_B
    prsLogic_NOT_B = _PRS_ASYNC_CH_CTRL_FNSEL_NOT_B
    prsLogic_A_XOR_B = _PRS_ASYNC_CH_CTRL_FNSEL_A_XOR_B
    prsLogic_A_NAND_B = _PRS_ASYNC_CH_CTRL_FNSEL_A_NAND_B
    prsLogic_A_AND_B = _PRS_ASYNC_CH_CTRL_FNSEL_A_AND_B
    prsLogic_A_XNOR_B = _PRS_ASYNC_CH_CTRL_FNSEL_A_XNOR_B
    prsLogic_B = _PRS_ASYNC_CH_CTRL_FNSEL_B
    prsLogic_NOT_A_OR_B = _PRS_ASYNC_CH_CTRL_FNSEL_NOT_A_OR_B
    prsLogic_A = _PRS_ASYNC_CH_CTRL_FNSEL_A
    prsLogic_A_OR_NOT_B = _PRS_ASYNC_CH_CTRL_FNSEL_A_OR_NOT_B
    prsLogic_A_OR_B = _PRS_ASYNC_CH_CTRL_FNSEL_A_OR_B
    prsLogic_One = _PRS_ASYNC_CH_CTRL_FNSEL_LOGICAL_ONE
}
Logic functions that can be used when combining two PRS channels.
```

```
enum PRS_Signal_t {
    prsSignalNone = PRS_SYNC_CH_CTRL_SOURCESEL_DEFAULT | (0x0 <<
        _PRS_SYNC_CH_CTRL_SIGSEL_SHIFT)
    prsSignalSW = PRS_SYNC_CH_CTRL_SOURCESEL_DEFAULT | (0x1 <<
        _PRS_SYNC_CH_CTRL_SIGSEL_SHIFT)
    prsSignalTIMER0_UF = PRS_TIMER0_UF
    prsSignalTIMER0_OF = PRS_TIMER0_OF
    prsSignalTIMER0_CC0 = PRS_TIMER0_CC0
    prsSignalTIMER0_CC1 = PRS_TIMER0_CC1
    prsSignalTIMER0_CC2 = PRS_TIMER0_CC2
    prsSignalTIMER1_UF = PRS_TIMER1_UF
    prsSignalTIMER1_OF = PRS_TIMER1_OF
    prsSignalTIMER1_CC0 = PRS_TIMER1_CC0
    prsSignalTIMER1_CC1 = PRS_TIMER1_CC1
    prsSignalTIMER1_CC2 = PRS_TIMER1_CC2
    prsSignalTIMER2_UF = PRS_TIMER2_UF
    prsSignalTIMER2_OF = PRS_TIMER2_OF
    prsSignalTIMER2_CC0 = PRS_TIMER2_CC0
    prsSignalTIMER2_CC1 = PRS_TIMER2_CC1
    prsSignalTIMER2_CC2 = PRS_TIMER2_CC2
    prsSignalTIMER3_UF = PRS_TIMER3_UF
    prsSignalTIMER3_OF = PRS_TIMER3_OF
    prsSignalTIMER3_CC0 = PRS_TIMER3_CC0
    prsSignalTIMER3_CC1 = PRS_TIMER3_CC1
    prsSignalTIMER3_CC2 = PRS_TIMER3_CC2
    prsSignalTIMER4_UF = PRS_TIMER4_UF
    prsSignalTIMER4_OF = PRS_TIMER4_OF
    prsSignalTIMER4_CC0 = PRS_TIMER4_CC0
    prsSignalTIMER4_CC1 = PRS_TIMER4_CC1
    prsSignalTIMER4_CC2 = PRS_TIMER4_CC2
    prsSignalLETIMER0_CH0 = PRS_LETIMER0_CH0
    prsSignalLETIMER0_CH1 = PRS_LETIMER0_CH1
    prsSignalCORE_CTIOUT0 = PRS_CORE_CTIOUT0
    prsSignalCORE_CTIOUT1 = PRS_CORE_CTIOUT1
    prsSignalCORE_CTIOUT2 = PRS_CORE_CTIOUT2
    prsSignalCORE_CTIOUT3 = PRS_CORE_CTIOUT3
    prsSignalCMUL_CLKOUT0 = PRS_CMUL_CLKOUT0
    prsSignalCMUL_CLKOUT1 = PRS_CMUL_CLKOUT1
    prsSignalCMUL_CLKOUT2 = PRS_CMUL_CLKOUT2
    prsSignalPRSL_ASYNCH0 = PRS_PRSL_ASYNCH0
    prsSignalPRSL_ASYNCH1 = PRS_PRSL_ASYNCH1
    prsSignalPRSL_ASYNCH2 = PRS_PRSL_ASYNCH2
    prsSignalPRSL_ASYNCH3 = PRS_PRSL_ASYNCH3
    prsSignalPRSL_ASYNCH4 = PRS_PRSL_ASYNCH4
    prsSignalPRSL_ASYNCH5 = PRS_PRSL_ASYNCH5
    prsSignalPRSL_ASYNCH6 = PRS_PRSL_ASYNCH6
    prsSignalPRSL_ASYNCH7 = PRS_PRSL_ASYNCH7
    prsSignalPRS_ASYNCH8 = PRS_PRS_ASYNCH8
    prsSignalPRS_ASYNCH9 = PRS_PRS_ASYNCH9
    prsSignalPRS_ASYNCH10 = PRS_PRS_ASYNCH10
    prsSignalPRS_ASYNCH11 = PRS_PRS_ASYNCH11
    prsSignalRTCC_CCV0 = PRS_RTCC_CCV0
    prsSignalRTCC_CCV1 = PRS_RTCC_CCV1
    prsSignalRTCC_CCV2 = PRS_RTCC_CCV2
    prsSignalBURTC_COMP = PRS_BURTC_COMP
    prsSignalBURTC_OF = PRS_BURTC_OF
    prsSignalUSART0_TXC = PRS_USART0_TXC
    prsSignalUSART0_RXDATA = PRS_USART0_RXDATA
    prsSignalUSART0_IRTX = PRS_USART0_IRTX
    prsSignalUSART0_RTS = PRS_USART0_RTS
    prsSignalUSART0_TX = PRS_USART0_TX
    prsSignalUSART0_CS = PRS_USART0_CS
    prsSignalUSART1_TXC = PRS_USART1_TXC
    prsSignalUSART1_RXDATA = PRS_USART1_RXDATA
    prsSignalUSART1_IRTX = PRS_USART1_IRTX
    prsSignalUSART1_RTS = PRS_USART1_RTS
    prsSignalUSART1_TX = PRS_USART1_TX
    prsSignalUSART1_CS = PRS_USART1_CS
    prsSignalIADC0_SCANENTRY = PRS_IADC0_SCANENTRYDONE
    prsSignalIADC0_SCANTABLE = PRS_IADC0_SCANTABLEDONE
    prsSignalIADC0_SINGLE = PRS_IADC0_SINGLEDONE
    prsSignalGPIO_PIN0 = PRS_GPIO_PIN0
    prsSignalGPIO_PIN1 = PRS_GPIO_PIN1
    prsSignalGPIO_PIN2 = PRS_GPIO_PIN2
    prsSignalGPIO_PIN3 = PRS_GPIO_PIN3
    prsSignalGPIO_PIN4 = PRS_GPIO_PIN4
    prsSignalGPIO_PIN5 = PRS_GPIO_PIN5
    prsSignalGPIO_PIN6 = PRS_GPIO_PIN6
    prsSignalGPIO_PIN7 = PRS_GPIO_PIN7
    prsSignalLFRCO_CALMEAS = PRS_LFRCO_CALMEAS
    prsSignalLFRCO_SDM = PRS_LFRCO_SDM
    prsSignalLFRCO_TCMEAS = PRS_LFRCO_TCMEAS
}
PRS_Signal.
```

```
enum PRS_Consumer_t {
    prsConsumerNone = 0x000
    prsConsumerCMU_CALDN = offsetof(PRS_TypeDef, CONSUMER_CMU_CALDN)
    prsConsumerCMU_CALUP = offsetof(PRS_TypeDef, CONSUMER_CMU_CALUP)
    prsConsumerIADC0_SCANTRIGGER = offsetof(PRS_TypeDef, CONSUMER_IADC0_SCANTRIGGER)
    prsConsumerIADC0_SINGLETRIGGER = offsetof(PRS_TypeDef,
        CONSUMER_IADC0_SINGLETRIGGER)
    prsConsumerLDMA_REQUEST0 = offsetof(PRS_TypeDef, CONSUMER_LDMAXBAR_DMAREQ0)
    prsConsumerLDMA_REQUEST1 = offsetof(PRS_TypeDef, CONSUMER_LDMAXBAR_DMAREQ1)
    prsConsumerLETIMERO_CLEAR = offsetof(PRS_TypeDef, CONSUMER_LETIMERO_CLEAR)
    prsConsumerLETIMERO_START = offsetof(PRS_TypeDef, CONSUMER_LETIMERO_START)
    prsConsumerLETIMERO_STOP = offsetof(PRS_TypeDef, CONSUMER_LETIMERO_STOP)
    prsConsumerTIMER0_CC0 = offsetof(PRS_TypeDef, CONSUMER_TIMER0_CC0)
    prsConsumerTIMER0_CC1 = offsetof(PRS_TypeDef, CONSUMER_TIMER0_CC1)
    prsConsumerTIMER0_CC2 = offsetof(PRS_TypeDef, CONSUMER_TIMER0_CC2)
    prsConsumerTIMER1_CC0 = offsetof(PRS_TypeDef, CONSUMER_TIMER1_CC0)
    prsConsumerTIMER1_CC1 = offsetof(PRS_TypeDef, CONSUMER_TIMER1_CC1)
    prsConsumerTIMER1_CC2 = offsetof(PRS_TypeDef, CONSUMER_TIMER1_CC2)
    prsConsumerTIMER2_CC0 = offsetof(PRS_TypeDef, CONSUMER_TIMER2_CC0)
    prsConsumerTIMER2_CC1 = offsetof(PRS_TypeDef, CONSUMER_TIMER2_CC1)
    prsConsumerTIMER2_CC2 = offsetof(PRS_TypeDef, CONSUMER_TIMER2_CC2)
    prsConsumerTIMER3_CC0 = offsetof(PRS_TypeDef, CONSUMER_TIMER3_CC0)
    prsConsumerTIMER3_CC1 = offsetof(PRS_TypeDef, CONSUMER_TIMER3_CC1)
    prsConsumerTIMER3_CC2 = offsetof(PRS_TypeDef, CONSUMER_TIMER3_CC2)
    prsConsumerTIMER4_CC0 = offsetof(PRS_TypeDef, CONSUMER_TIMER4_CC0)
    prsConsumerTIMER4_CC1 = offsetof(PRS_TypeDef, CONSUMER_TIMER4_CC1)
    prsConsumerTIMER4_CC2 = offsetof(PRS_TypeDef, CONSUMER_TIMER4_CC2)
    prsConsumerUSART0_CLK = offsetof(PRS_TypeDef, CONSUMER_USART0_CLK)
    prsConsumerUSART0_IR = offsetof(PRS_TypeDef, CONSUMER_USART0_IR)
    prsConsumerUSART0_RX = offsetof(PRS_TypeDef, CONSUMER_USART0_RX)
    prsConsumerUSART0_TRIGGER = offsetof(PRS_TypeDef, CONSUMER_USART0_TRIGGER)
    prsConsumerUSART1_CLK = offsetof(PRS_TypeDef, CONSUMER_USART1_CLK)
    prsConsumerUSART1_IR = offsetof(PRS_TypeDef, CONSUMER_USART1_IR)
    prsConsumerUSART1_RX = offsetof(PRS_TypeDef, CONSUMER_USART1_RX)
    prsConsumerUSART1_TRIGGER = offsetof(PRS_TypeDef, CONSUMER_USART1_TRIGGER)
    prsConsumerEUART0_RX = offsetof(PRS_TypeDef, CONSUMER_EUART0_RX)
    prsConsumerEUART0_TRIGGER = offsetof(PRS_TypeDef, CONSUMER_EUART0_TRIGGER)
    prsConsumerWDOG0_SRC0 = offsetof(PRS_TypeDef, CONSUMER_WDOG0_SRC0)
    prsConsumerWDOG0_SRC1 = offsetof(PRS_TypeDef, CONSUMER_WDOG0_SRC1)
    prsConsumerRTCC_CC0 = offsetof(PRS_TypeDef, CONSUMER_RTCC_CC0)
    prsConsumerRTCC_CC1 = offsetof(PRS_TypeDef, CONSUMER_RTCC_CC1)
    prsConsumerRTCC_CC2 = offsetof(PRS_TypeDef, CONSUMER_RTCC_CC2)
}
PRS Consumers.
```

Functions

uint32_t **PRS_ConvertToSyncSource**(uint32_t asyncSource)
Convert an async PRS source to a sync source.

uint32_t **PRS_ConvertToSyncSignal**(uint32_t asyncSource, uint32_t asyncSignal)
Convert an async PRS signal to a sync signal.

void **PRS_SourceSignalSet**(unsigned int ch, uint32_t source, uint32_t signal, PRS_Edge_TypeDef edge)
Set a source and signal for a channel.

void **PRS_SourceAsyncSignalSet**(unsigned int ch, uint32_t source, uint32_t signal)
Set the source and asynchronous signal for a channel.

int **PRS_GetFreeChannel**(PRS_ChType_t type)
Search for the first free PRS channel.

void **PRS_Reset**(void)
Reset all PRS channels.

void **PRS_ConnectSignal**(unsigned int ch, PRS_ChType_t type, PRS_Signal_t signal)
Connect a PRS signal to a channel.

void **PRS_ConnectConsumer**(unsigned int ch, PRS_ChType_t type, PRS_Consumer_t consumer)
Connect a peripheral consumer to a PRS channel.

void **PRS_PinOutput**(unsigned int ch, PRS_ChType_t type, GPIO_Port_TypeDef port, uint8_t pin)
Send the output of a PRS channel to a GPIO pin.

void	PRS_Combine (unsigned int chA, unsigned int chB, PRS_Logic_t logic) Combine two PRS channels using a logic function.
void	PRS_LevelSet (uint32_t level, uint32_t mask) Set level control bit for one or more channels.
uint32_t	PRS_LevelGet (void) Get level control bit for all channels.
uint32_t	PRS_Values (PRS_ChType_t type) Get the PRS channel values for all channels.
bool	PRS_ChannelValue (unsigned int ch, PRS_ChType_t type) Get the PRS channel value for a single channel.
void	PRS_PulseTrigger (uint32_t channels) Trigger a high pulse (one HFPERCLK) for one or more channels.
void	PRS_ChannelLevelSet (unsigned int ch, bool level) Set the PRS channel level for one asynchronous PRS channel.
void	PRS_ChannelPulse (unsigned int ch) Trigger a pulse on one PRS channel.

Macros

#define	PRS_SYNC_CHAN_COUNT PRS_SYNC_CH_NUM PRS Synchronous channel count.
#define	PRS_ASYNC_CHAN_COUNT PRS_ASYNC_CH_NUM PRS Asynchronous channel count.
#define	PRS_ASYNC_SUPPORTED 1 PRS asynchronous support.

Enumeration Documentation

PRS_ChType_t

PRS_ChType_t	
PRS Channel type.	
Enumerator	
prsTypeAsync	Asynchronous channel type.
prsTypeSync	Synchronous channel type.

PRS_Edge_TypeDef

PRS_Edge_TypeDef	
Edge detection type.	
Enumerator	
prsEdgeOff	Leave signal as is.
prsEdgePos	Generate pulses on positive edge.
prsEdgeNeg	Generate pulses on negative edge.
prsEdgeBoth	Generate pulses on both edges.

PRS_Logic_t

PRS_Logic_t

Logic functions that can be used when combining two PRS channels.

Enumerator	
prsLogic_Zero	Logical 0.
prsLogic_A_NOR_B	A NOR B.
prsLogic_NOT_A_AND_B	(!A) NOR B.
prsLogic_NOT_A	!A.
prsLogic_A_AND_NOT_B	A AND (!B).
prsLogic_NOT_B	!B.
prsLogic_A_XOR_B	A XOR B.
prsLogic_A_NAND_B	A NAND B.
prsLogic_A_AND_B	A AND B.
prsLogic_A_XNOR_B	A XNOR B.
prsLogic_B	B.
prsLogic_NOT_A_OR_B	(!A) OR B.
prsLogic_A	A.
prsLogic_A_OR_NOT_B	A OR (!B).
prsLogic_A_OR_B	A OR B.
prsLogic_One	Logical 1.

PRS_Signal_t

PRS_Signal_t

PRS Signal.

Enumerator	
prsSignalNone	No Signal.
prsSignalSW	Software-reserved Signal.
prsSignalTIMER0_UF	TIMER0 underflow Signal.
prsSignalTIMER0_OF	TIMER0 overflow Signal.
prsSignalTIMER0_CC0	TIMER0 capture/compare channel 0 Signal.
prsSignalTIMER0_CC1	TIMER0 capture/compare channel 1 Signal.
prsSignalTIMER0_CC2	TIMER0 capture/compare channel 2 Signal.
prsSignalTIMER1_UF	TIMER1 underflow Signal.
prsSignalTIMER1_OF	TIMER1 overflow Signal.
prsSignalTIMER1_CC0	TIMER1 capture/compare channel 0 Signal.
prsSignalTIMER1_CC1	TIMER1 capture/compare channel 1 Signal.
prsSignalTIMER1_CC2	TIMER1 capture/compare channel 2 Signal.
prsSignalTIMER2_UF	TIMER2 underflow Signal.
prsSignalTIMER2_OF	TIMER2 overflow Signal.

prSignalTIMER2_CC0	TIMER2 capture/compare channel 0 Signal.
prSignalTIMER2_CC1	TIMER2 capture/compare channel 1 Signal.
prSignalTIMER2_CC2	TIMER2 capture/compare channel 2 Signal.
prSignalTIMER3_UF	TIMER3 underflow Signal.
prSignalTIMER3_OF	TIMER3 overflow Signal.
prSignalTIMER3_CC0	TIMER3 capture/compare channel 0 Signal.
prSignalTIMER3_CC1	TIMER3 capture/compare channel 1 Signal.
prSignalTIMER3_CC2	TIMER3 capture/compare channel 2 Signal.
prSignalTIMER4_UF	TIMER4 underflow Signal.
prSignalTIMER4_OF	TIMER4 overflow Signal.
prSignalTIMER4_CC0	TIMER4 capture/compare channel 0 Signal.
prSignalTIMER4_CC1	TIMER4 capture/compare channel 1 Signal.
prSignalTIMER4_CC2	TIMER4 capture/compare channel 2 Signal.
prSignalLETIMER0_CH0	LETIMER0 channel 0 Signal.
prSignalLETIMER0_CH1	LETIMER0 channel 1 Signal.
prSignalCORE_CTIOOUT0	CORE CTIOOUT0 Signal.
prSignalCORE_CTIOOUT1	CORE CTIOOUT1 Signal.
prSignalCORE_CTIOOUT2	CORE CTIOOUT2 Signal.
prSignalCORE_CTIOOUT3	CORE CTIOOUT3 Signal.
prSignalCMUL_CLKOUT0	CMU CLKOUT0 Signal.
prSignalCMUL_CLKOUT1	CMU CLKOUT1 Signal.
prSignalCMUL_CLKOUT2	CMU CLKOUT2 Signal.
prSignalPRSL_ASYNCH0	PRS channel 0 Signal.
prSignalPRSL_ASYNCH1	PRS channel 1 Signal.
prSignalPRSL_ASYNCH2	PRS channel 2 Signal.
prSignalPRSL_ASYNCH3	PRS channel 3 Signal.
prSignalPRSL_ASYNCH4	PRS channel 4 Signal.
prSignalPRSL_ASYNCH5	PRS channel 5 Signal.
prSignalPRSL_ASYNCH6	PRS channel 6 Signal.
prSignalPRSL_ASYNCH7	PRS channel 7 Signal.
prSignalPRS_ASYNCH8	PRS channel 8 Signal.
prSignalPRS_ASYNCH9	PRS channel 9 Signal.
prSignalPRS_ASYNCH10	PRS channel 10 Signal.
prSignalPRS_ASYNCH11	PRS channel 11 Signal.
prSignalRTCC_CCV0	RTCC capture/compare channel 0 Signal.
prSignalRTCC_CCV1	RTCC capture/compare channel 1 Signal.
prSignalRTCC_CCV2	RTCC capture/compare channel 2 Signal.
prSignalBURTC_COMP	BURTC compare Signal.
prSignalBURTC_OF	BURTC overflow Signal.
prSignalUSART0_TXC	USART0 TX complete Signal.
prSignalUSART0_RXDATA	USART0 RX data available Signal.
prSignalUSART0_IRTX	USART0 IR TX Signal.
prSignalUSART0_RTS	USART0 RTS Signal.

prsSignalUSART0_TX	USART0 TX Signal.
prsSignalUSART0_CS	USART0 chip select Signal.
prsSignalUSART1_TXC	USART1 TX complete Signal.
prsSignalUSART1_RXDATA	USART1 RX data available Signal.
prsSignalUSART1_IRTX	USART1 IR TX Signal.
prsSignalUSART1_RTS	USART1 RTS Signal.
prsSignalUSART1_TX	USART1 TX Signal.
prsSignalUSART1_CS	USART1 chip select Signal.
prsSignalIADC0_SCANENTRY	IADC0 scan entry Signal.
prsSignalIADC0_SCANTABLE	IADC0 scan table Signal.
prsSignalIADC0_SINGLE	IADC0 single Signal.
prsSignalGPIO_PIN0	GPIO Pin 0 Signal.
prsSignalGPIO_PIN1	GPIO Pin 1 Signal.
prsSignalGPIO_PIN2	GPIO Pin 2 Signal.
prsSignalGPIO_PIN3	GPIO Pin 3 Signal.
prsSignalGPIO_PIN4	GPIO Pin 4 Signal.
prsSignalGPIO_PIN5	GPIO Pin 5 Signal.
prsSignalGPIO_PIN6	GPIO Pin 6 Signal.
prsSignalGPIO_PIN7	GPIO Pin 7 Signal.
prsSignalLFRCO_CALMEAS	
prsSignalLFRCO_SDM	LFRCO Calibration Measure Signal.
prsSignalLFRCO_TCMEAS	LFRCO Sigma Delta Modulator output Signal.

PRS_Consumer_t

PRS_Consumer_t

PRS Consumers.

	Enumerator
prsConsumerNone	No PRS consumer.
prsConsumerCMU_CALDN	CMU calibration down consumer.
prsConsumerCMU_CALUP	CMU calibration up consumer.
prsConsumerIADC0_SCANTRIGGER	IADC0 scan trigger consumer.
prsConsumerIADC0_SINGLETRIGGER	IADC0 single trigger consumer.
prsConsumerLDMA_REQUEST0	LDMA Request 0 consumer.
prsConsumerLDMA_REQUEST1	LDMA Request 1 consumer.
prsConsumerLETIMER0_CLEAR	LETIMER0 clear consumer.
prsConsumerLETIMER0_START	LETIMER0 start consumer.
prsConsumerLETIMER0_STOP	LETIMER0 stop consumer.
prsConsumerTIMER0_CC0	TIMER0 capture/compare channel 0 consumer.
prsConsumerTIMER0_CC1	TIMER0 capture/compare channel 1 consumer.
prsConsumerTIMER0_CC2	TIMER0 capture/compare channel 2 consumer.
prsConsumerTIMER1_CC0	TIMER1 capture/compare channel 0 consumer.

prsConsumerTIMER1_CC1	TIMER1 capture/compare channel 1 consumer.
prsConsumerTIMER1_CC2	TIMER1 capture/compare channel 2 consumer.
prsConsumerTIMER2_CC0	TIMER2 capture/compare channel 0 consumer.
prsConsumerTIMER2_CC1	TIMER2 capture/compare channel 1 consumer.
prsConsumerTIMER2_CC2	TIMER2 capture/compare channel 2 consumer.
prsConsumerTIMER3_CC0	TIMER3 capture/compare channel 0 consumer.
prsConsumerTIMER3_CC1	TIMER3 capture/compare channel 1 consumer.
prsConsumerTIMER3_CC2	TIMER3 capture/compare channel 2 consumer.
prsConsumerTIMER4_CC0	TIMER4 capture/compare channel 0 consumer.
prsConsumerTIMER4_CC1	TIMER4 capture/compare channel 1 consumer.
prsConsumerTIMER4_CC2	TIMER4 capture/compare channel 2 consumer.
prsConsumerUSART0_CLK	USART0 clock consumer.
prsConsumerUSART0_IR	USART0 IR consumer.
prsConsumerUSART0_RX	USART0 RX consumer.
prsConsumerUSART0_TRIGGER	USART0 trigger consumer.
prsConsumerUSART1_CLK	USART1 clock consumer.
prsConsumerUSART1_IR	USART1 IR consumer.
prsConsumerUSART1_RX	USART1 TX consumer.
prsConsumerUSART1_TRIGGER	USART1 trigger consumer.
prsConsumerEUSART0_RX	EUSART0 RX consumer.
prsConsumerEUSART0_TRIGGER	EUSART0 TRIGGER Consumer.
prsConsumerWDOG0_SRC0	WDOG0 source 0 consumer.
prsConsumerWDOG0_SRC1	WDOG0 source 1 consumer.
prsConsumerRTCC_CC0	RTCC capture/compare channel 0 consumer.
prsConsumerRTCC_CC1	RTCC capture/compare channel 1 consumer.
prsConsumerRTCC_CC2	RTCC capture/compare channel 2 consumer.

Function Documentation

PRS_ConvertToSyncSource

```
uint32_t PRS_ConvertToSyncSource (uint32_t asyncSource)
```

Convert an async PRS source to a sync source.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	asyncSource	The id of the asynchronous PRS source.

This conversion must be done because the id's of the same peripheral source is different depending on if it's used as an asynchronous PRS source or a synchronous PRS source.

Returns

- The id of the corresponding synchronous PRS source.

PRS_ConvertToSyncSignal

```
uint32_t PRS_ConvertToSyncSignal (uint32_t asyncSource, uint32_t asyncSignal)
```

Convert an async PRS signal to a sync signal.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	asyncSource	The id of the asynchronous PRS source.
uint32_t	[in]	asyncSignal	The id of the asynchronous PRS signal.

PRS values for some peripherals signals differ between asynchronous and synchronous PRS channels. This function must be used to handle the conversion.

Returns

- The id of the corresponding synchronous PRS signal.

PRS_SourceSignalSet

```
void PRS_SourceSignalSet (unsigned int ch, uint32_t source, uint32_t signal, PRS_Edge_TypeDef edge)
```

Set a source and signal for a channel.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	ch	A channel to define the signal and source for.
uint32_t	[in]	source	A source to select for the channel. Use one of PRS_CH_CTRL_SOURCESEL_x defines.
uint32_t	[in]	signal	A signal (for selected <code>source</code>) to use. Use one of PRS_CH_CTRL_SIGSEL_x defines.
PRS_Edge_TypeDef	[in]	edge	An edge (for selected source/signal) to generate the pulse for.

PRS_SourceAsyncSignalSet

```
void PRS_SourceAsyncSignalSet (unsigned int ch, uint32_t source, uint32_t signal)
```

Set the source and asynchronous signal for a channel.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	ch	A channel to define the source and asynchronous signal for.
uint32_t	[in]	source	A source to select for the channel. Use one of PRS_CH_CTRL_SOURCESEL_x defines.
uint32_t	[in]	signal	An asynchronous signal (for selected <code>source</code>) to use. Use one of the PRS_CH_CTRL_SIGSEL_x defines that support asynchronous operation.

Asynchronous reflexes are not clocked on HPERCLK and can be used even in EM2/EM3. There is a limitation to reflexes operating in asynchronous mode in that they can only be used by a subset of the reflex consumers.

See the PRS chapter in the reference manual for the complete list of supported asynchronous signals and consumers.

Note

- This function is not supported on EFM32GxxxFyyy parts. In asynchronous mode, the edge detector only works in EMO and should not be used. The EDSEL parameter in PRS_CHx_CTRL register is set to 0 (OFF) by default.

PRS_GetFreeChannel

```
int PRS_GetFreeChannel (PRS_ChType_t type)
```

Search for the first free PRS channel.

Parameters

Type	Direction	Argument Name	Description
PRS_ChType_t	[in]	type	PRS channel type. This can be either prTypeAsync or prTypeSync .

Returns

- Channel number ≥ 0 if an unused PRS channel was found. If no free PRS channel was found then -1 is returned.

PRS_Reset

```
void PRS_Reset (void )
```

Reset all PRS channels.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will reset all the PRS channel configuration.

PRS_ConnectSignal

```
void PRS_ConnectSignal (unsigned int ch, PRS_ChType_t type, PRS_Signal_t signal)
```

Connect a PRS signal to a channel.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	ch	PRS channel number.
PRS_ChType_t	[in]	type	PRS channel type. This can be either prTypeAsync or prTypeSync .
PRS_Signal_t	[in]	signal	This is the PRS signal that should be placed on the channel.

This function will make the PRS signal available on the specific channel. Only a single PRS signal can be connected to any given channel.

PRS_ConnectConsumer

```
void PRS_ConnectConsumer (unsigned int ch, PRS_ChType_t type, PRS_Consumer_t consumer)
```

Connect a peripheral consumer to a PRS channel.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	ch	PRS channel number.
PRS_ChType_t	[in]	type	PRS channel type. This can be either prstTypeAsync or prstTypeSync .
PRS_Consumer_t	[in]	consumer	This is the PRS consumer.

Different peripherals can use PRS channels as their input. This function can be used to connect a peripheral consumer to a PRS channel. Multiple consumers can be connected to a single PRS channel.

PRS_PinOutput

```
void PRS_PinOutput (unsigned int ch, PRS_ChType_t type, GPIO_Port_TypeDef port, uint8_t pin)
```

Send the output of a PRS channel to a GPIO pin.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	ch	PRS channel number.
PRS_ChType_t	[in]	type	PRS channel type. This can be either prstTypeAsync or prstTypeSync .
GPIO_Port_TypeDef	[in]	port	GPIO port
uint8_t	[in]	pin	GPIO pin

This function is used to send the output of a PRS channel to a GPIO pin. Note that there are certain restrictions to where a PRS channel can be routed. Consult the datasheet of the device to see if a channel can be routed to the requested GPIO pin. Some devices for instance can only route the async channels 0-5 on GPIO pins PAX and PBx while async channels 6-11 can only be routed to GPIO pins PCx and PDx

PRS_Combine

```
void PRS_Combine (unsigned int chA, unsigned int chB, PRS_Logic_t logic)
```

Combine two PRS channels using a logic function.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	chA	PRS Channel for the A input.
unsigned int	[in]	chB	PRS Channel for the B input.
PRS_Logic_t	[in]	logic	The logic function to use when combining the Channel A and Channel B. The output of the logic function is the output of Channel A. Function like AND, OR, XOR, NOT and more are available.

This function allows you to combine the output of one PRS channel with the the signal of another PRS channel using various logic functions. Note that for series 2, config 1 devices, the hardware only allows a PRS channel to be combined with the previous channel. So for instance channel 5 can be combined only with channel 4.

The logic function operates on two PRS channels called A and B. The output of PRS channel B is combined with the PRS source configured for channel A to produce an output. This output is used as the output of channel A.

PRS_LevelSet

```
void PRS_LevelSet (uint32_t level, uint32_t mask)
```

Set level control bit for one or more channels.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	level	Level to use for channels indicated by mask . Use logical OR combination of PRS_SWLEVEL_CHnLEVEL defines for channels to set high level, otherwise 0.
uint32_t	[in]	mask	Mask indicating which channels to set level for. Use logical OR combination of PRS_SWLEVEL_CHnLEVEL defines.

The level value for a channel is XORed with both the pulse possibly issued by [PRS_PulseTrigger\(\)](#) and the PRS input signal selected for the channel(s).

Note

- Note that software level control is only available for asynchronous channels on Series 2 devices.

PRS_LevelGet

```
uint32_t PRS_LevelGet (void )
```

Get level control bit for all channels.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The current software level configuration.

PRS_Values

```
uint32_t PRS_Values (PRS_ChType_t type)
```

Get the PRS channel values for all channels.

Parameters

Type	Direction	Argument Name	Description
PRS_ChType_t	[in]	type	PRS channel type. This can be either prstTypeAsync or prstTypeSync .

Returns

- The current PRS channel output values for all channels as a bitset.

PRS_ChannelValue

```
bool PRS_ChannelValue (unsigned int ch, PRS\_ChType\_t type)
```

Get the PRS channel value for a single channel.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	ch	PRS channel number.
PRS_ChType_t	[in]	type	PRS channel type. This can be either prstTypeAsync or prstTypeSync .

Returns

- The current PRS channel output value. This is either 0 or 1.

PRS_PulseTrigger

```
void PRS_PulseTrigger (uint32_t channels)
```

Trigger a high pulse (one HPERCLK) for one or more channels.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	channels	Logical ORed combination of channels to trigger a pulse for. Use PRS_SWPULSE_CHnPULSE defines.

Setting a bit for a channel causes the bit in the register to remain high for one HPERCLK cycle. Pulse is XORed with both the corresponding bit in PRS SWLEVEL register and the PRS input signal selected for the channel(s).

PRS_ChannelLevelSet

```
void PRS_ChannelLevelSet (unsigned int ch, bool level)
```

Set the PRS channel level for one asynchronous PRS channel.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	ch	PRS channel number.
bool	[in]	level	true to set the level high (1) and false to set the level low (0).

PRS_ChannelPulse

```
void PRS_ChannelPulse (unsigned int ch)
```

Trigger a pulse on one PRS channel.

Parameters

Type	Direction	Argument Name	Description
unsigned int	[in]	ch	PRS channel number.

Macro Definition Documentation

PRS_SYNC_CHAN_COUNT

```
#define PRS_SYNC_CHAN_COUNT
```

Value:

```
PRS_SYNC_CH_NUM
```

PRS Synchronous channel count.

PRS_ASYNC_CHAN_COUNT

```
#define PRS_ASYNC_CHAN_COUNT
```

Value:

```
PRS_ASYNC_CH_NUM
```

PRS Asynchronous channel count.

PRS_ASYNC_SUPPORTED

```
#define PRS_ASYNC_SUPPORTED
```

Value:

```
1
```

PRS asynchronous support.

RAMFUNC - RAM Function Support

RAMFUNC - RAM Function Support

RAM code support.

Provides support for executing code from RAM. Provides a unified method to manage RAM code across all supported tools.

Note

- Other cross-compiler support macros are implemented in [COMMON](#).
- Functions executing from RAM should not be declared as static.

Warnings

- Standard library facilities are available to the tool with GCC in hosted mode (default), regardless of the section attribute. Calls to standard libraries placed in the default section may therefore occur. To disable hosted mode, add '-ffreestanding' to the build command line. This is the only way to guarantee no calls to standard libraries with GCC. Read more at www.gnu.org/onlinedocs/gcc-5.3.0/gcc/Standards.html
- Keil/ARM uVision users must add a section named "ram_code" in their linker scatter file. This section must be in RAM memory. Look in the MCU SDK for example scatter files (ram_code.sct).

Usage

In your .h file:

```
#include "em_ramfunc.h"

SL_RAMFUNC_DECLARATOR
void MyPrint(const char* string);
```

Issues have been observed with ARM GCC when there is no declarator. It is recommended to have a declarator also for internal functions but move the declarator to the .c file.

In your .c file:

```
#include "em_ramfunc.h"

SL_RAMFUNC_DEFINITION_BEGIN
void MyPrint(const char* string)
{
    ...
}
SL_RAMFUNC_DEFINITION_END
```

Macros

- | | | |
|---------|---------------------------------------|---|
| #define | SL_RAMFUNC_DISABLE | This define is not present by default. |
| #define | SL_RAMFUNC_DECLARATOR | Compiler ported function declarator for RAM code. |

#define [SL_RAMFUNC_DEFINITION_BEGIN](#)
Compiler ported function definition begin marker for RAM code.

#define [SL_RAMFUNC_DEFINITION_END](#)
Compiler ported function definition end marker for RAM code.

Macro Definition Documentation

SL_RAMFUNC_DISABLE

```
#define SL_RAMFUNC_DISABLE
```

This define is not present by default.

By compiling with define [SL_RAMFUNC_DISABLE](#), code placed in RAM by SL_RAMFUNC macros will be placed in default code space (Flash) instead.

Note

- This define is not present by default.

SL_RAMFUNC_DECLARATOR

```
#define SL_RAMFUNC_DECLARATOR
```

Compiler ported function declarator for RAM code.

SL_RAMFUNC_DEFINITION_BEGIN

```
#define SL_RAMFUNC_DEFINITION_BEGIN
```

Compiler ported function definition begin marker for RAM code.

SL_RAMFUNC_DEFINITION_END

```
#define SL_RAMFUNC_DEFINITION_END
```

Compiler ported function definition end marker for RAM code.

RMU - Reset Management Unit

RMU - Reset Management Unit

Reset Management Unit (RMU) Peripheral API.

This module contains functions to control the RMU peripheral of Silicon Labs 32-bit MCUs and SoCs. RMU ensures correct reset operation and is responsible for connecting the different reset sources to the reset lines of the MCU or SoC.

Enumerations

```
enum RMU_ResetMode_TypeDef {
    rmuResetModeDisabled = 0
    rmuResetModeEnabled = 1
}
RMU reset modes.

enum RMU_Reset_TypeDef {
    rmuResetWdog0 = _EMU_RSTCTRL_WDOGORMODE_MASK
    rmuResetSys = _EMU_RSTCTRL_SYSRMODE_MASK
    rmuResetCoreLockup = _EMU_RSTCTRL_LOCKUPRMODE_MASK
    rmuResetAVDD = _EMU_RSTCTRL_AVddbBODRMODE_MASK
    rmuResetIOVDD0 = _EMU_RSTCTRL_IOVDD0BODRMODE_MASK
    rmuResetDecouple = _EMU_RSTCTRL_DECBODRMODE_MASK
    rmuResetDCI = _EMU_RSTCTRL_DCIRMODE_MASK
}
RMU controlled peripheral reset control and reset source control.
```

Functions

```
void RMU_ResetControl(RMU_Reset_TypeDef reset, RMU_ResetMode_TypeDef mode)
    Disable/enable reset for various peripherals and signal sources.

void RMU_ResetCauseClear(void)
    Clear the reset cause register.

uint32_t RMU_ResetCauseGet(void)
    Get the cause of the last reset.
```

Macros

```
#define RMU_LockupResetDisable (A)
    RMU_LockupResetDisable kept for backwards compatibility.
```

Enumeration Documentation

RMU_ResetMode_TypeDef

RMU_ResetMode_TypeDef	
RMU reset modes.	
Enumerator	
rmuResetModeDisabled	Reset mode disabled.
rmuResetModeEnabled	Reset mode enabled.

RMU_Reset_TypeDef

RMU_Reset_TypeDef

RMU controlled peripheral reset control and reset source control.

	Enumerator
rmuResetWdog0	WDOG0 reset select.
rmuResetSys	SYSRESET select.
rmuResetCoreLockup	Cortex lockup reset select.
rmuResetAVDD	AVDD monitoring select.
rmuResetIOVDD0	IOVDD0 monitoring select.
rmuResetDecouple	Decouple monitoring select.
rmuResetDCI	DCI reset select.

Function Documentation

RMU_ResetControl

void RMU_ResetControl (RMU_Reset_TypeDef reset, RMU_ResetMode_TypeDef mode)

Disable/enable reset for various peripherals and signal sources.

Parameters

Type	Direction	Argument Name	Description
RMU_Reset_TypeDef	[in]	reset	Reset types to enable/disable.s
RMU_ResetMode_TypeDef	[in]	mode	Reset mode.

RMU_ResetCauseClear

void RMU_ResetCauseClear (void)

Clear the reset cause register.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function clears all the reset cause bits of the RSTCAUSE register. The reset cause bits must be cleared by software before a new reset occurs. Otherwise, reset causes may accumulate. See [RMU_ResetCauseGet\(\)](#).

RMU_ResetCauseGet

uint32_t RMU_ResetCauseGet (void)

Get the cause of the last reset.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

To be useful, the reset cause must be cleared by software before a new reset occurs. Otherwise, reset causes may accumulate. See [RMU_ResetCauseClear\(\)](#). This function call will return the main cause for reset, which can be a bit mask (several causes) and clear away "noise".

Returns

- A reset cause mask. See the reference manual for a description of the reset cause mask.

Macro Definition Documentation

RMU_LockupResetDisable

```
#define RMU_LockupResetDisable
```

Value:

```
(A)
```

RMU_LockupResetDisable kept for backwards compatibility.

RTCC - Real Timer Counter/Calendar

RTCC - Real Timer Counter/Calendar

Real Time Counter and Calendar (RTCC) Peripheral API.

This module contains functions to control the RTCC peripheral of Silicon Labs 32-bit MCUs and SoCs. The RTCC ensures timekeeping in low energy modes. The RTCC also includes a BCD calendar mode for easy time and date keeping.

Modules

[RTCC_Init_TypeDef](#)

[RTCC_CCChConf_TypeDef](#)

Enumerations

```
enum    RTCC_CntPresc_TypeDef {
    rtccCntPresc_1 = _RTCC_CFG_CNTPRESC_DIV1
    rtccCntPresc_2 = _RTCC_CFG_CNTPRESC_DIV2
    rtccCntPresc_4 = _RTCC_CFG_CNTPRESC_DIV4
    rtccCntPresc_8 = _RTCC_CFG_CNTPRESC_DIV8
    rtccCntPresc_16 = _RTCC_CFG_CNTPRESC_DIV16
    rtccCntPresc_32 = _RTCC_CFG_CNTPRESC_DIV32
    rtccCntPresc_64 = _RTCC_CFG_CNTPRESC_DIV64
    rtccCntPresc_128 = _RTCC_CFG_CNTPRESC_DIV128
    rtccCntPresc_256 = _RTCC_CFG_CNTPRESC_DIV256
    rtccCntPresc_512 = _RTCC_CFG_CNTPRESC_DIV512
    rtccCntPresc_1024 = _RTCC_CFG_CNTPRESC_DIV1024
    rtccCntPresc_2048 = _RTCC_CFG_CNTPRESC_DIV2048
    rtccCntPresc_4096 = _RTCC_CFG_CNTPRESC_DIV4096
    rtccCntPresc_8192 = _RTCC_CFG_CNTPRESC_DIV8192
    rtccCntPresc_16384 = _RTCC_CFG_CNTPRESC_DIV16384
    rtccCntPresc_32768 = _RTCC_CFG_CNTPRESC_DIV32768
}
Counter prescaler selection.
```

```
enum    RTCC_PrescMode_TypeDef {
    rtccCntTickPresc = _RTCC_CFG_CNTTICK_PRESC
    rtccCntTickCCV0Match = _RTCC_CFG_CNTTICK_CCV0MATCH
}
Prescaler mode of the RTCC counter.
```

```
enum    RTCC_CapComChMode_TypeDef {
    rtccCapComChModeOff = _RTCC_CC_CTRL_MODE_OFF
    rtccCapComChModeCapture = _RTCC_CC_CTRL_MODE_INPUTCAPTURE
    rtccCapComChModeCompare = _RTCC_CC_CTRL_MODE_OUTPUTCOMPARE
}
Capture/Compare channel mode.
```

```
enum    RTCC_CompMatchOutAction_TypeDef {
    rtccCompMatchOutActionPulse = _RTCC_CC_CTRL_CMOA_PULSE
    rtccCompMatchOutActionToggle = _RTCC_CC_CTRL_CMOA_TOGGLE
    rtccCompMatchOutActionClear = _RTCC_CC_CTRL_CMOA_CLEAR
    rtccCompMatchOutActionSet = _RTCC_CC_CTRL_CMOA_SET
}
Compare match output action mode.
```

```
enum RTCC_InEdgeSel_TypeDef {
    rtccInEdgeRising = _RTCC_CC_CTRL_ICEDGE_RISING
    rtccInEdgeFalling = _RTCC_CC_CTRL_ICEDGE_FALLING
    rtccInEdgeBoth = _RTCC_CC_CTRL_ICEDGE_BOTH
    rtccInEdgeNone = _RTCC_CC_CTRL_ICEDGE_NONE
}
Input edge select.

enum RTCC_CompBase_TypeDef {
    rtccCompBaseCnt = _RTCC_CC_CTRL_COMPBASE_CNT
    rtccCompBasePreCnt = _RTCC_CC_CTRL_COMPBASE_PRECNT
}
Capture/Compare channel compare mode.
```

Typedefs

```
typedef uint8_t RTCC_PRSSel_TypeDef
PRS channel number.
```

Functions

```
void RTCC_ChannelInit(int ch, RTCC_CCChConf_TypeDef const *confPtr)
Configure the selected capture/compare channel of the RTCC.

void RTCC_Enable(bool enable)
Enable/disable RTCC counting.

void RTCC_Init(const RTCC_Init_TypeDef *init)
Initialize RTCC.

void RTCC_Reset(void)
Restore RTCC to its reset state.

void RTCC_StatusClear(void)
Clear the STATUS register.

uint32_t RTCC_ChannelCompareValueGet(int ch)
Get the RTCC compare register value for a selected channel.

void RTCC_ChannelCompareValueSet(int ch, uint32_t value)
Set the RTCC compare register value for a selected channel.

uint32_t RTCC_ChannelCaptureValueGet(int ch)
Get the RTCC input capture register value for a selected channel.

uint32_t RTCC_ChannelCCVGet(int ch)
Get the RTCC capture/compare register value for a selected channel.

void RTCC_ChannelCCVSet(int ch, uint32_t value)
Set RTCC capture/compare register value for a selected channel.

uint32_t RTCC_CombinedCounterGet(void)
Get the combined CNT/PRECN register content.

uint32_t RTCC_CounterGet(void)
Get the RTCC counter value.

void RTCC_CounterSet(uint32_t value)
Set the RTCC CNT counter.

void RTCC_IntClear(uint32_t flags)
Clear one or more pending RTCC interrupts.

void RTCC_IntDisable(uint32_t flags)
Disable one or more RTCC interrupts.
```

void	RTCC_IntEnable (uint32_t flags) Enable one or more RTCC interrupts.
uint32_t	RTCC_IntGet (void) Get pending RTCC interrupt flags.
uint32_t	RTCC_IntGetEnabled (void) Get enabled and pending RTCC interrupt flags.
void	RTCC_IntSet (uint32_t flags) Set one or more pending RTCC interrupts from SW.
void	RTCC_Lock (void) Lock RTCC registers.
uint32_t	RTCC_PreCounterGet (void) Get the RTCC pre-counter value.
void	RTCC_PreCounterSet (uint32_t preCntVal) Set the RTCC pre-counter value.
uint32_t	RTCC_StatusGet (void) Get the STATUS register value.
void	RTCC_SyncWait (void) Wait for the RTCC to complete all synchronization of register changes and commands.
void	RTCC_Start (void) Start the RTCC counter.
void	RTCC_Stop (void) Stop the RTCC counter.
void	RTCC_Unlock (void) Unlock RTCC registers.

Macros

#define	RTCC_INIT_DEFAULT undefined Default RTCC initialization structure.
#define	RTCC_CH_INIT_COMPARE_DEFAULT undefined Default RTCC channel output compare initialization structure.
#define	RTCC_CH_INIT_CAPTURE_DEFAULT undefined Default RTCC channel input capture initialization structure.
#define	RTCC_CH_VALID (ch) Number of RTCC capture/compare channels.

Enumeration Documentation

RTCC_CntPresc_TypeDef

RTCC_CntPresc_TypeDef	
Counter prescaler selection.	
Enumerator	
rtccCntPresc_1	Divide clock by 1.
rtccCntPresc_2	Divide clock by 2.
rtccCntPresc_4	Divide clock by 4.
rtccCntPresc_8	Divide clock by 8.

rtccCntPresc_16	Divide clock by 16.
rtccCntPresc_32	Divide clock by 32.
rtccCntPresc_64	Divide clock by 64.
rtccCntPresc_128	Divide clock by 128.
rtccCntPresc_256	Divide clock by 256.
rtccCntPresc_512	Divide clock by 512.
rtccCntPresc_1024	Divide clock by 1024.
rtccCntPresc_2048	Divide clock by 2048.
rtccCntPresc_4096	Divide clock by 4096.
rtccCntPresc_8192	Divide clock by 8192.
rtccCntPresc_16384	Divide clock by 16384.
rtccCntPresc_32768	Divide clock by 32768.

RTCC_PrescMode_TypeDef

RTCC_PrescMode_TypeDef

Prescaler mode of the RTCC counter.

Enumerator

rtccCntTickPresc	CNT register ticks according to the prescaler value.
rtccCntTickCCV0Match	CNT register ticks when PRECNT matches the 15 least significant bits of ch.

RTCC_CapComChMode_TypeDef

RTCC_CapComChMode_TypeDef

Capture/Compare channel mode.

Enumerator

rtccCapComChModeOff	Capture/Compare channel turned off.
rtccCapComChModeCapture	Capture mode.
rtccCapComChModeCompare	Compare mode.

RTCC_CompMatchOutAction_TypeDef

RTCC_CompMatchOutAction_TypeDef

Compare match output action mode.

Enumerator

rtccCompMatchOutActionPulse	Generate a pulse.
rtccCompMatchOutActionToggle	Toggle output.
rtccCompMatchOutActionClear	Clear output.
rtccCompMatchOutActionSet	Set output.

RTCC_InEdgeSel_TypeDef

RTCC_InEdgeSel_TypeDef

Input edge select.

Enumerator	
rtccInEdgeRising	Rising edges detected.
rtccInEdgeFalling	Falling edges detected.
rtccInEdgeBoth	Both edges detected.
rtccInEdgeNone	No edge detection, signal is left as is.

RTCC_CompBase_TypeDef

RTCC_CompBase_TypeDef

Capture/Compare channel compare mode.

Enumerator	
rtccCompBaseCnt	CCVx is compared with the CNT register.
rtccCompBasePreCnt	CCVx is compared with a CNT[16:0] and PRECNT[14:0].

Typedef Documentation

RTCC_PRSSel_TypeDef

```
typedef uint8_t RTCC_PRSSel_TypeDef
```

PRS channel number.

This type is used when configuring input capture mode on a RTCC channel.

Function Documentation

RTCC_ChannelInit

```
void RTCC_ChannelInit (int ch, RTCC_CCChConf_TypeDef const * confPtr)
```

Configure the selected capture/compare channel of the RTCC.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	A channel selector.
RTCC_CCChConf_TypeDef const *	[in]	confPtr	A pointer to the configuration structure.

Use this function to configure an RTCC channel. Select capture/compare mode, match output action, overflow output action, and PRS input configuration. See the configuration structure [RTCC_CCChConf_TypeDef](#) for more details.

RTCC_Enable

```
void RTCC_Enable (bool enable)
```

Enable/disable RTCC counting.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	True to enable RTCC counting, false to disable counting.

RTCC_Init

```
void RTCC_Init (const RTCC_Init_TypeDef * init)
```

Initialize RTCC.

Parameters

Type	Direction	Argument Name	Description
const RTCC_Init_TypeDef *	[in]	init	A pointer to the RTCC initialization structure.

Note that the compare values must be set separately with RTCC_CompareSet(), which should probably be done prior to the use of this function if configuring the RTCC to start when initialization is completed.

RTCC_Reset

```
void RTCC_Reset (void )
```

Restore RTCC to its reset state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

RTCC_StatusClear

```
void RTCC_StatusClear (void )
```

Clear the STATUS register.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

RTCC_ChannelCompareValueGet

```
uint32_t RTCC_ChannelCompareValueGet (int ch)
```

Get the RTCC compare register value for a selected channel.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	Channel selector.

Returns

- Compare register value.

RTCC_ChannelCompareValueSet

```
void RTCC_ChannelCompareValueSet (int ch, uint32_t value)
```

Set the RTCC compare register value for a selected channel.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	Channel selector.
uint32_t	[in]	value	Compare register value

RTCC_ChannelCaptureValueGet

```
uint32_t RTCC_ChannelCaptureValueGet (int ch)
```

Get the RTCC input capture register value for a selected channel.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	Channel selector.

Returns

- Capture register value.

RTCC_ChannelCCVGet

```
uint32_t RTCC_ChannelCCVGet (int ch)
```

Get the RTCC capture/compare register value for a selected channel.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	Channel selector.

For parts with separate capture compare value registers, this function returns the compare value.

Returns

- Capture/compare register value.

RTCC_ChannelCCVSet

```
void RTCC_ChannelCCVSet (int ch, uint32_t value)
```

Set RTCC capture/compare register value for a selected channel.

Parameters

Type	Direction	Argument Name	Description
int	[in]	ch	Channel selector.
uint32_t	[in]	value	Capture/compare register value

For parts with separate capture compare value registers, this function sets the compare value.

RTCC_CombinedCounterGet

```
uint32_t RTCC_CombinedCounterGet (void )
```

Get the combined CNT/PRECNT register content.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- CNT/PRECNT register value.

RTCC_CounterGet

```
uint32_t RTCC_CounterGet (void )
```

Get the RTCC counter value.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Current RTCC counter value.

RTCC_CounterSet

```
void RTCC_CounterSet (uint32_t value)
```

Set the RTCC CNT counter.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	value	CNT value.

RTCC_IntClear

```
void RTCC_IntClear (uint32_t flags)
```

Clear one or more pending RTCC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	RTCC interrupt sources to clear. Use a set of interrupt flags OR-ed together to clear multiple interrupt sources.

RTCC_IntDisable

```
void RTCC_IntDisable (uint32_t flags)
```

Disable one or more RTCC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	RTCC interrupt sources to disable. Use a set of interrupt flags OR-ed together to disable multiple interrupt.

RTCC_IntEnable

```
void RTCC_IntEnable (uint32_t flags)
```

Enable one or more RTCC interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	RTCC interrupt sources to enable. Use a set of interrupt flags OR-ed together to set multiple interrupt.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [RTCC_IntClear\(\)](#) prior to enabling the interrupt.

RTCC_IntGet

```
uint32_t RTCC_IntGet (void )
```

Get pending RTCC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Event bits are not cleared by using this function.

Returns

Pending RTCC interrupt sources. Returns a set of interrupt flags OR-ed together for the interrupt sources set.

RTCC_IntGetEnabled

```
uint32_t RTCC_IntGetEnabled (void )
```

Get enabled and pending RTCC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Useful for handling more interrupt sources in the same interrupt handler.

Returns

- Pending and enabled RTCC interrupt sources. Returns a set of interrupt flags OR-ed together for the interrupt sources set.

RTCC_IntSet

```
void RTCC_IntSet (uint32_t flags)
```

Set one or more pending RTCC interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	RTCC interrupt sources to set to pending. Use a set of interrupt flags (RTCC_IFS_nnn).

RTCC_Lock

```
void RTCC_Lock (void )
```

Lock RTCC registers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- When RTCC registers are locked, RTCC_CTRL, RTCC_PRECNT, RTCC_CNT, RTCC_TIME, RTCC_DATE, RTCC_IEN, RTCC_POWERDOWN and RTCC_CCx_XXX registers cannot be written to.

RTCC_PreCounterGet

```
uint32_t RTCC_PreCounterGet (void )
```

Get the RTCC pre-counter value.

Parameters

Type	Direction	Argument Name	Description
Type	Direction	Argument Name	Description
void	N/A		

Returns

- Current RTCC pre-counter value.

RTCC_PreCounterSet

```
void RTCC_PreCounterSet (uint32_t preCntVal)
```

Set the RTCC pre-counter value.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	preCntVal	RTCC pre-counter value to be set.

RTCC_StatusGet

```
uint32_t RTCC_StatusGet (void )
```

Get the STATUS register value.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Current STATUS register value.

RTCC_SyncWait

```
void RTCC_SyncWait (void )
```

Wait for the RTCC to complete all synchronization of register changes and commands.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

RTCC_Start

```
void RTCC_Start (void )
```

Start the RTCC counter.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will send a start command to the RTCC peripheral. The RTCC peripheral will use some LF clock ticks before the command is executed. The [RTCC_SyncWait\(\)](#) function can be used to wait for the start command to be executed.

Note

- This function requires the RTCC to be enabled.

RTCC_Stop

```
void RTCC_Stop (void )
```

Stop the RTCC counter.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function will send a stop command to the RTCC peripheral. The RTCC peripheral will use some LF clock ticks before the command is executed. The [RTCC_SyncWait\(\)](#) function can be used to wait for the stop command to be executed.

Note

- This function requires the RTCC to be enabled.

RTCC_Unlock

```
void RTCC_Unlock (void )
```

Unlock RTCC registers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- When RTCC registers are locked, RTCC_CTRL, RTCC_PRECNT, RTCC_CNT, RTCC_TIME, RTCC_DATE, RTCC_IEN, RTCC_POWERDOWN and RTCC_CCx_XXX registers cannot be written to.

Macro Definition Documentation

RTCC_INIT_DEFAULT

```
#define RTCC_INIT_DEFAULT
```

Value:

```
{
    true,  /* Start counting when init done.          */
    false, /* Disable RTCC during debug halt.          */
    false, /* Disable precounter wrap on ch. 0 CCV value. */
    false, /* Disable counter wrap on ch. 1 CCV value.   */
}
```

```

0 |     rtccCntPresc_32, /* 977 us per tick.          */ \
0 |     rtccCntTickPresc, /* Counter increments according to prescaler value.*/ \
0 | }

```

Default RTCC initialization structure.

RTCC_CH_INIT_COMPARE_DEFAULT

```
#define RTCC_CH_INIT_COMPARE_DEFAULT
```

Value:

```

0 | {
0 |     rtccCapComChModeCompare, /* Select output compare mode. */ \
0 |     rtccCompMatchOutActionPulse, /* Create pulse on compare match. */ \
0 |     0, /* Don't care. */ \
0 |     rtccInEdgeNone, /* Don't care. */ \
0 |     rtccCompBaseCnt, /* Use CNT for comparison. */ \
0 | }

```

Default RTCC channel output compare initialization structure.

RTCC_CH_INIT_CAPTURE_DEFAULT

```
#define RTCC_CH_INIT_CAPTURE_DEFAULT
```

Value:

```

0 | {
0 |     rtccCapComChModeCapture, /* Select input capture mode. */ \
0 |     rtccCompMatchOutActionPulse, /* Don't care. */ \
0 |     0, /* Use PRS channel 0 as trigger. */ \
0 |     rtccInEdgeRising, /* Capture on rising edge. */ \
0 |     rtccCompBaseCnt, /* Don't care. */ \
0 | }

```

Default RTCC channel input capture initialization structure.

RTCC_CH_VALID

```
#define RTCC_CH_VALID
```

Value:

```
(ch)
```

Number of RTCC capture/compare channels.

Validation of valid RTCC channel for assert statements.

RTCC_Init_TypeDef

RTCC initialization structure.

Public Attributes

- `bool` [enable](#)
Enable/disable counting when initialization is completed.
- `bool` [debugRun](#)
Enable/disable timer counting during debug halt.
- `bool` [precntWrapOnCCV0](#)
Enable/disable pre-counter wrap on ch.
- `bool` [cntWrapOnCCV1](#)
Enable/disable counter wrap on ch.

[RTCC_CntPresc_TypeDef](#) [presc](#)
Counter prescaler.

[RTCC_PrescMode_TypeDef](#) [prescMode](#)
Prescaler mode.

Public Attribute Documentation

enable

```
bool RTCC_Init_TypeDef::enable
```

Enable/disable counting when initialization is completed.

debugRun

```
bool RTCC_Init_TypeDef::debugRun
```

Enable/disable timer counting during debug halt.

precntWrapOnCCV0

```
bool RTCC_Init_TypeDef::precntWrapOnCCV0
```

Enable/disable pre-counter wrap on ch.

0 CCV value.

cntWrapOnCCV1

```
bool RTCC_Init_TypeDef::cntWrapOnCCV1
```

Enable/disable counter wrap on ch.

1 CCV value.

presc

```
RTCC_CntPresc_TypeDef RTCC_Init_TypeDef::presc
```

Counter prescaler.

prescMode

```
RTCC_PrescMode_TypeDef RTCC_Init_TypeDef::prescMode
```

Prescaler mode.

RTCC_CCChConf_TypeDef

RTCC capture/compare channel configuration structure.

Public Attributes

RTCC_CapComChMode_TypeDef	chMode Select mode of Capture/Compare channel.
RTCC_CompMatchOutAction_TypeDef	compMatchOutAction Compare mode channel match output action.
RTCC_PRSSel_TypeDef	prsSel Capture mode channel PRS input channel selection.
RTCC_InEdgeSel_TypeDef	inputEdgeSel Capture mode channel input edge selection.
RTCC_CompBase_TypeDef	compBase Comparison base of channel in compare mode.

Public Attribute Documentation

chMode

RTCC_CapComChMode_TypeDef RTCC_CCChConf_TypeDef::chMode

Select mode of Capture/Compare channel.

compMatchOutAction

RTCC_CompMatchOutAction_TypeDef RTCC_CCChConf_TypeDef::compMatchOutAction

Compare mode channel match output action.

prsSel

RTCC_PRSSel_TypeDef RTCC_CCChConf_TypeDef::prsSel

Capture mode channel PRS input channel selection.

inputEdgeSel

RTCC_InEdgeSel_TypeDef RTCC_CCChConf_TypeDef::inputEdgeSel

Capture mode channel input edge selection.

compBase

RTCC_CompBase_TypeDef RTCC_CCChConf_TypeDef::compBase

Comparison base of channel in compare mode.

SE - Secure Element

SE - Secure Element

Secure Element peripheral API.

Abstraction of the Secure Element's mailbox interface.

For series 2 devices with a part number that is xG23 or higher, the following step is necessary for basic operation:

Clock enable:

```
CMU_ClockEnable(cmuClock_SEMAILBOX, true);
```

Note

- The high-level SE API has been moved to the SE manager, and the implementation in `em_se` should not be used.
- Using the SE's mailbox is not thread-safe in EMLIB, and accessing the SE's mailbox both in regular and IRQ context is not safe. SE operations should be performed using the SE manager if possible.

Modules

[SE_DataTransfer_t](#)

[SE_Command_t](#)

[Deprecated Functions](#)

Typedefs

`typedef uint32_t` [SE_Response_t](#)
Possible responses to a command.

Functions

`void` [SE_addDataInput](#)(SE_Command_t *command, SE_DataTransfer_t *data)
Add input data to a command.

`void` [SE_addDataOutput](#)(SE_Command_t *command, SE_DataTransfer_t *data)
Add output data to a command.

`void` [SE_addParameter](#)(SE_Command_t *command, uint32_t parameter)
Add a parameter to a command.

`void` [SE_executeCommand](#)(SE_Command_t *command)
Execute the passed command.

[SE_Response_t](#) [SE_getVersion](#)(uint32_t *version)
Get current SE version.

[SE_Response_t](#) [SE_getConfigStatusBits](#)(uint32_t *cfgStatus)
Get VSE configuration and status bits.

[SE_Response_t](#) [SE_getOTPVersion](#)(uint32_t *otpVersion)
Get the version number of the OTP from the status field of the output mailbox.

<code>SE_Response_t</code>	<code>SE_ackCommand(SE_Command_t *command)</code> Acknowledge and get status and output data of a completed command.
<code>bool</code>	<code>SE_isCommandCompleted(void)</code> Check whether the running command has completed.
<code>uint32_t</code>	<code>SE_readExecutedCommand(void)</code> Read the previously executed command.
<code>SE_Response_t</code>	<code>SE_readCommandResponse(void)</code> Read the status of the previously executed command.
<code>void</code>	<code>SE_waitCommandCompletion(void)</code> Wait for completion of the current command.
<code>void</code>	<code>SE_disableInterrupt(uint32_t flags)</code> Disable one or more SE interrupts.
<code>void</code>	<code>SE_enableInterrupt(uint32_t flags)</code> Enable one or more SE interrupts.
<code>bool</code>	<code>rootIsOutputMailboxValid(void)</code> Check whether the VSE Output Mailbox is valid.

Macros

<code>#define</code>	<code>SE_RESPONSE_MAILBOX_INVALID</code> 0x00FE0000UL Root Code Mailbox is invalid.
<code>#define</code>	<code>SE_RESPONSE_MAILBOX_VALID</code> 0xE5ECC0DEUL Root Code Mailbox magic word.
<code>#define</code>	<code>SE_RESPONSE_MASK</code> 0x000F0000UL Response status codes for the Secure Element.
<code>#define</code>	<code>SE_RESPONSE_OK</code> 0x00000000UL Command executed successfully or signature was successfully validated.
<code>#define</code>	<code>SE_FIFO_MAX_PARAMETERS</code> 13U Maximum amount of parameters supported by the hardware FIFO.
<code>#define</code>	<code>SE_DATATRANSFER_STOP</code> 0x00000001UL Stop datatransfer.
<code>#define</code>	<code>SE_DATATRANSFER_DISCARD</code> 0x40000000UL Discard datatransfer.
<code>#define</code>	<code>SE_DATATRANSFER_REALIGN</code> 0x20000000UL Realign datatransfer.
<code>#define</code>	<code>SE_DATATRANSFER_CONSTADDRESS</code> 0x10000000UL Datatransfer Const Address.
<code>#define</code>	<code>SE_DATATRANSFER_LENGTH_MASK</code> 0xFFFFFFFUL Stop Length Mask.
<code>#define</code>	<code>SE_MAX_PARAMETERS</code> 4U Maximum amount of parameters for largest command in defined command set.
<code>#define</code>	<code>SE_DATATRANSFER_DEFAULT</code> (address, length) Default initialization of data transfer struct.
<code>#define</code>	<code>SE_COMMAND_DEFAULT</code> (command) Default initialization of command struct.

Typedef Documentation

SE_Response_t

```
typedef uint32_t SE_Response_t
```

Possible responses to a command.

Function Documentation

SE_addDataInput

```
void SE_addDataInput (SE_Command_t * command, SE_DataTransfer_t * data)
```

Add input data to a command.

Parameters

Type	Direction	Argument Name	Description
SE_Command_t *	[in]	command	Pointer to an SE command structure.
SE_DataTransfer_t *	[in]	data	Pointer to a data transfer structure.

This function adds a buffer of input data to the given SE command structure. The buffer gets appended by reference at the end of the list of already added buffers.

Note

- Note that this function does not copy either the data buffer or the buffer structure, so make sure to keep the data object in scope until the command has been executed by the secure element.

SE_addDataOutput

```
void SE_addDataOutput (SE_Command_t * command, SE_DataTransfer_t * data)
```

Add output data to a command.

Parameters

Type	Direction	Argument Name	Description
SE_Command_t *	[in]	command	Pointer to an SE command structure.
SE_DataTransfer_t *	[in]	data	Pointer to a data transfer structure.

This function adds a buffer of output data to the given command structure. The buffer gets appended by reference at the end of the list of already added buffers.

Note

- Note that this function does not copy either the data buffer or the buffer structure, so make sure to keep the data object in scope until the command has been executed by the secure element.

SE_addParameter

```
void SE_addParameter (SE_Command_t * command, uint32_t parameter)
```

Add a parameter to a command.

Parameters

Type	Direction	Argument Name	Description
SE_Command_t *	[in]	command	Pointer to a filled-out SE command structure.
uint32_t	[in]	parameter	Parameter to add.

This function adds a parameter word to the passed command.

Note

- Make sure to not exceed [SE_MAX_PARAMETERS](#).

SE_executeCommand

```
void SE_executeCommand (SE\_Command\_t * command)
```

Execute the passed command.

Parameters

Type	Direction	Argument Name	Description
SE_Command_t *	[in]	command	Pointer to a filled-out SE command structure.

This function starts the execution of the passed command by the secure element. When started, wait for the RXINT interrupt flag, or call [SE_waitCommandCompletion](#) to busy-wait. After completion, you have to call [SE_readCommandResponse](#) to get the command's execution status.

SE_getVersion

```
SE_Response_t SE_getVersion (uint32_t * version)
```

Get current SE version.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[in]	version	Pointer to location where to copy the version of VSE to.

This function returns the current VSE version

Returns

- One of the SE_RESPONSE return codes: SE_RESPONSE_OK when the command was executed successfully
SE_RESPONSE_INVALID_PARAMETER when an invalid parameter was passed SE_RESPONSE_MAILBOX_INVALID when the mailbox content is invalid

SE_getConfigStatusBits

```
SE_Response_t SE_getConfigStatusBits (uint32_t * cfgStatus)
```

Get VSE configuration and status bits.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[out]	cfgStatus	Pointer to location to copy Configuration Status bits into.

This function returns the current VSE configuration and status bits. The following list explains what the different bits in cfgStatus indicate. A bit value of 1 means enabled, while 0 means disabled:

- [0]: Secure boot
- [1]: Verify secure boot certificate
- [2]: Anti-rollback
- [3]: Narrow page lock
- [4]: Full page lock The following status bits can be read with VSE versions higher than 1.2.2.
- [10]: Debug port lock
- [11]: Device erase enabled
- [12]: Secure debug enabled
- [15]: Debug port register state, 1 if the debug port is locked.

Note

- This function will check that the mailbox content is valid before reading the status bits. If the command response has already been read with a call to [SE_ackCommand\(\)](#), the validity check will fail, and the config status bits cannot be read before a reset has occurred.

Returns

- One of the SE_RESPONSE return codes: SE_RESPONSE_OK when the command was executed successfully
SE_RESPONSE_INVALID_PARAMETER when an invalid parameter was passed
SE_RESPONSE_MAILBOX_INVALID when the mailbox content is invalid

SE_getOTPVersion

```
SE_Response_t SE_getOTPVersion (uint32_t * otpVersion)
```

Get the version number of the OTP from the status field of the output mailbox.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[out]	otpVersion	Pointer to location to copy OTP version number into.

This function checks if the OTP version number flag is set in the output mailbox. If it is, the version number is written to otpVersion pointer location. If not, it returns error response.

Returns

- One of the SE_RESPONSE return codes.

Return values

- SE_RESPONSE_OK: when the command was executed successfully

SE_ackCommand

```
SE_Response_t SE_ackCommand (SE_Command_t * command)
```

Acknowledge and get status and output data of a completed command.

Parameters

Type	Direction	Argument Name	Description
<code>SE_Command_t *</code>	[in]	command	Pointer to an SE command structure.

This function acknowledges and gets the status and output data of a completed mailbox command. The mailbox command is acknowledged by inverting all bits in the checksum (XOR with 0xFFFFFFFF). The output data is copied into the linked list of output buffers pointed to in the given command data structure.

Returns

- One of the SE_RESPONSE return codes.

Return values

- SE_RESPONSE_OK: when the command was executed successfully or a signature was successfully verified,
- SE_RESPONSE_INVALID_COMMAND: when the command ID was not recognized,
- SE_RESPONSE_AUTHORIZATION_ERROR: when the command is not authorized,
- SE_RESPONSE_INVALID_SIGNATURE: when signature verification failed,
- SE_RESPONSE_BUS_ERROR: when a bus error was thrown during the command, e.g. because of conflicting Secure/Non-Secure memory accesses,
- SE_RESPONSE_CRYPTO_ERROR: on an internal SE failure, or
- SE_RESPONSE_INVALID_PARAMETER: when an invalid parameter was passed
- SE_RESPONSE_MAILBOX_INVALID: when mailbox command not done or invalid

SE_isCommandCompleted

```
bool SE_isCommandCompleted (void )
```

Check whether the running command has completed.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function polls the SE-to-host mailbox interrupt flag.

Returns

- True if a command has completed and the result is available

SE_readExecutedCommand

```
uint32_t SE_readExecutedCommand (void )
```

Read the previously executed command.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function reads the previously executed command.

Returns

- One of the SE command words. SE_RESPONSE_MAILBOX_INVALID when the mailbox content is invalid.

SE_readCommandResponse

```
SE_Response_t SE_readCommandResponse (void )
```

Read the status of the previously executed command.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function reads the status of the previously executed command.

Returns

- One of the SE_RESPONSE return codes: SE_RESPONSE_OK when the command was executed successfully or a signature was successfully verified, SE_RESPONSE_INVALID_COMMAND when the command ID was not recognized, SE_RESPONSE_AUTHORIZATION_ERROR when the command is not authorized, SE_RESPONSE_INVALID_SIGNATURE when signature verification failed, SE_RESPONSE_BUS_ERROR when a bus error was thrown during the command, e.g. because of conflicting Secure/Non-Secure memory accesses, SE_RESPONSE_CRYPTO_ERROR on an internal SE failure, or SE_RESPONSE_INVALID_PARAMETER when an invalid parameter was passed SE_RESPONSE_MAILBOX_INVALID when the mailbox content is invalid

SE_waitCommandCompletion

```
void SE_waitCommandCompletion (void )
```

Wait for completion of the current command.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function "busy"-waits until the execution of the ongoing instruction has completed.

SE_disableInterrupt

```
void SE_disableInterrupt (uint32_t flags)
```

Disable one or more SE interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	SE interrupt sources to disable. Use a bitwise logic OR combination of valid interrupt flags for the Secure Element module (SE_CONFIGURATION_(TX/RX)INTEN).

SE_enableInterrupt

```
void SE_enableInterrupt (uint32_t flags)
```

Enable one or more SE interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	SE interrupt sources to enable. Use a bitwise logic OR combination of valid interrupt flags for the Secure Element module (SEMAILBOX_CONFIGURATION_TXINTEN or SEMAILBOX_CONFIGURATION_RXINTEN).

rootIsOutputMailboxValid

```
bool rootIsOutputMailboxValid (void )
```

Check whether the VSE Output Mailbox is valid.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if the VSE Output Mailbox is valid (magic and checksum OK)

Macro Definition Documentation

SE_RESPONSE_MAILBOX_INVALID

```
#define SE_RESPONSE_MAILBOX_INVALID
```

Value:

```
0x00FE0000UL
```

Root Code Mailbox is invalid.

SE_RESPONSE_MAILBOX_VALID

```
#define SE_RESPONSE_MAILBOX_VALID
```

Value:

```
0xE5ECC0DEUL
```

Root Code Mailbox magic word.

SE_RESPONSE_MASK

```
#define SE_RESPONSE_MASK
```

Value:

```
0x000F0000UL
```

Response status codes for the Secure Element.

SE_RESPONSE_OK

```
#define SE_RESPONSE_OK
```

Value:

```
0x00000000UL
```

Command executed successfully or signature was successfully validated.

SE_FIFO_MAX_PARAMETERS

```
#define SE_FIFO_MAX_PARAMETERS
```

Value:

```
13U
```

Maximum amount of parameters supported by the hardware FIFO.

SE_DATATRANSFER_STOP

```
#define SE_DATATRANSFER_STOP
```

Value:

```
0x00000001UL
```

Stop datatransfer.

SE_DATATRANSFER_DISCARD

```
#define SE_DATATRANSFER_DISCARD
```

Value:

```
0x40000000UL
```

Discard datatransfer.

SE_DATATRANSFER_REALIGN

```
#define SE_DATATRANSFER_REALIGN
```

Value:

```
0x20000000UL
```

Realign datatransfer.

SE_DATATRANSFER_CONSTADDRESS

```
#define SE_DATATRANSFER_CONSTADDRESS
```

Value:

```
0x10000000UL
```

Datatransfer Const Address.

SE_DATATRANSFER_LENGTH_MASK

```
#define SE_DATATRANSFER_LENGTH_MASK
```

Value:

```
0x0FFFFFFFUL
```

Stop Length Mask.

SE_MAX_PARAMETERS

```
#define SE_MAX_PARAMETERS
```

Value:

```
4U
```

Maximum amount of parameters for largest command in defined command set.

SE_DATATRANSFER_DEFAULT

```
#define SE_DATATRANSFER_DEFAULT
```

Value:

```
0 | {
0 |     (void*)(address),           /* Pointer to data block */ \
0 |     (void*)SE_DATATRANSFER_STOP, /* This is the last block by default */ \
0 |     (length) | SE_DATATRANSFER_REALIGN /* Add size, use realign by default */ \
0 | }
```

Default initialization of data transfer struct.

SE_COMMAND_DEFAULT

```
#define SE_COMMAND_DEFAULT
```

Value:

```
0 | {
0 |     (command),           /* Given command */ \
0 |     NULL,               /* No data in */ \
0 |     NULL,               /* No data out */ \
0 |     { 0, 0, 0, 0 },     /* No parameters */ \
0 |     0,                 /* No parameters */ \
0 | }
```

Default initialization of command struct.

SE_DataTransfer_t

SE DMA transfer descriptor.

Can be linked to each other to provide scatter-gather behavior.

Public Attributes

volatile void *volatile	data Data pointer.
void *volatile	next Next descriptor.
volatile uint32_t	length Length.

Public Attribute Documentation

data

```
volatile void* volatile SE_DataTransfer_t::data
```

Data pointer.

next

```
void* volatile SE_DataTransfer_t::next
```

Next descriptor.

length

```
volatile uint32_t SE_DataTransfer_t::length
```

Length.

SE_Command_t

SE Command structure to which all commands to the SE must adhere.

Public Attributes

uint32_t	command SE Command.
SE_DataTransfer_t *	data_in Input data.
SE_DataTransfer_t *	data_out Output data.
uint32_t	parameters Parameters.
size_t	num_parameters Number of parameters.

Public Attribute Documentation

command

```
uint32_t SE_Command_t::command
```

SE Command.

data_in

```
SE_DataTransfer_t* SE_Command_t::data_in
```

Input data.

data_out

```
SE_DataTransfer_t* SE_Command_t::data_out
```

Output data.

parameters

```
uint32_t SE_Command_t::parameters[SE_MAX_PARAMETERS]
```

Parameters.

num_parameters

size_t SE_Command_t::num_parameters

Number of parameters.

Deprecated Functions

Deprecated Functions

Deprecated Functions.

Modules

- [SE_OTPInit_t](#)
- [SE_DebugStatus_t](#)
- [SE_Status_t](#)

Functions

- [SE_Response_t](#) [SE_initOTP](#)(SE_OTPInit_t *otp_init) SL_DEPRECATED_API_SDK_3_0
- [SE_Response_t](#) [SE_initPubkey](#)(uint32_t key_type, void *pubkey, uint32_t numBytes, bool signature) SL_DEPRECATED_API_SDK_3_0

Function Documentation

SE_initOTP

```
SE_Response_t SE_initOTP (SE_OTPInit_t * otp_init)
```

Parameters

Type	Direction	Argument Name	Description
SE_OTPInit_t *	[in]	otp_init	SE_OTPInit_t structure containing the SE configuration.

Initialize SE one-time-programmable (OTP) configuration.

Configuration is performed by setting the desired options in the [SE_OTPInit_t](#) structure.

This function can be used to enable secure boot, to configure flash page locking, and to enable anti-rollback protection when using the SE to perform an application upgrade, typically a Gecko bootloader upgrade.

Before secure boot can be enabled, the public key used for secure boot verification must be uploaded using [SE_initPubkey\(\)](#).

Warnings

- This command can only be executed once per device! When the configuration has been programmed it is not possible to update any of the fields.

Returns

- One of the SE_RESPONSE return codes.

Return values

- SE_RESPONSE_OK: when the command was executed successfully

SE_initPubkey

SE_Response_t SE_initPubkey (uint32_t key_type, void * pubkey, uint32_t numBytes, bool signature)

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	key_type	ID of key type to initialize.
void *	[in]	pubkey	Pointer to a buffer that contains the public key or signature. Must be word aligned and have a length of 64 bytes.
uint32_t	[in]	numBytes	Length of pubkey buffer (64 bytes).
bool	[in]	signature	If true, initialize signature for the specified key type instead of the public key itself.

Init pubkey or pubkey signature.

Initialize public key stored in the SE, or its corresponding signature. The command can be used to write:

- SE_KEY_TYPE_BOOT – public key used to perform secure boot
- SE_KEY_TYPE_AUTH – public key used to perform secure debug

Note

- These keys can not be overwritten, so this command can only be issued once per key per part.

Returns

- One of the SE_RESPONSE return codes.

Return values

- SE_RESPONSE_OK: when the command was executed successfully
- SE_RESPONSE_TEST_FAILED: when the pubkey is not set
- SE_RESPONSE_INVALID_PARAMETER: when an invalid type is passed

SE_OTPInit_t

SE OTP initialization struct.

Public Attributes

bool	enableSecureBoot	Enable secure boot for the host.
bool	verifySecureBootCertificate	Require certificate based secure boot signing.
bool	enableAntiRollback	Enable anti-rollback for host application upgrades.
bool	secureBootPageLockNarrow	Set flag to enable locking down all flash pages that cover the secure-booted image, except the last page if end of signature is not page-aligned.
bool	secureBootPageLockFull	Set flag to enable locking down all flash pages that cover the secure-booted image, including the last page if end of signature is not page-aligned.

Public Attribute Documentation

enableSecureBoot

```
bool SE_OTPInit_t::enableSecureBoot
```

Enable secure boot for the host.

verifySecureBootCertificate

```
bool SE_OTPInit_t::verifySecureBootCertificate
```

Require certificate based secure boot signing.

enableAntiRollback

```
bool SE_OTPInit_t::enableAntiRollback
```

Enable anti-rollback for host application upgrades.

secureBootPageLockNarrow

```
bool SE_OTPInit_t::secureBootPageLockNarrow
```

Set flag to enable locking down all flash pages that cover the secure-booted image, except the last page if end of signature is not page-aligned.

```
bool SE_OTPInit_t::secureBootPageLockFull
```

Set flag to enable locking down all flash pages that cover the secure-booted image, including the last page if end of signature is not page-aligned.

SE_DebugStatus_t

SE debug status.

Public Attributes

- bool [debugLockEnabled](#)
Whether debug lock is enabled.
- bool [deviceEraseEnabled](#)
Whether device erase is enabled.
- bool [secureDebugEnabled](#)
Whether secure debug is enabled.

Public Attribute Documentation

debugLockEnabled

```
bool SE_DebugStatus_t::debugLockEnabled
```

Whether debug lock is enabled.

deviceEraseEnabled

```
bool SE_DebugStatus_t::deviceEraseEnabled
```

Whether device erase is enabled.

secureDebugEnabled

```
bool SE_DebugStatus_t::secureDebugEnabled
```

Whether secure debug is enabled.

SE_Status_t

SE status.

Public Attributes

uint32_t	bootStatus	Boot status code / error code (Bits [7:0]).
uint32_t	seFwVersion	SE firmware version.
uint32_t	hostFwVersion	Host firmware version (if available).
SE_DebugStatus_t	debugStatus	Debug lock status.
bool	secureBootEnabled	Secure boot enabled.

Public Attribute Documentation

bootStatus

uint32_t SE_Status_t::bootStatus

Boot status code / error code (Bits [7:0]).

seFwVersion

uint32_t SE_Status_t::seFwVersion

SE firmware version.

hostFwVersion

uint32_t SE_Status_t::hostFwVersion

Host firmware version (if available).

debugStatus

SE_DebugStatus_t SE_Status_t::debugStatus

Debug lock status.

secureBootEnabled

```
bool SE_Status_t::secureBootEnabled
```

Secure boot enabled.

SMU - Security Management Unit

SMU - Security Management Unit

Security Management Unit (SMU) Peripheral API.

SMU forms the control and status/reporting component of bus-level security in EFM32/EFR32 devices.

Peripheral-level protection is provided via the Peripheral Protection Unit (PPU). PPU provides hardware access barrier to any peripheral that is configured to be protected. When an attempt is made to access a peripheral without the required privilege/security level, PPU detects the fault and intercepts the access. No write or read of the peripheral register space occurs, and an all-zero value is returned if the access is a read.

Usage example

```
// SMU is always clocked, so no call to CMU_ClockEnable() is necessary

// Initialize SMU to prevent access to CMU, EMU, SMU and GPIO
SMU_Init_TypeDef init = SMU_INIT_DEFAULT;
init.ppu.access.privilegedCMU = true;
init.ppu.access.privilegedEMU = true;
init.ppu.access.privilegedSMU = true;
init.ppu.access.privilegedGPIO = true;
SMU_Init(&init);
```

Modules

[SMU_PrivilegedAccess_TypeDef](#)

[SMU_Init_TypeDef](#)

Enumerations

```
enum SMU_Peripheral_TypeDef {
    smuPeripheralEMU = _SMU_PPUPATDO_EMU_SHIFT
    smuPeripheralCMU = _SMU_PPUPATDO_CMU_SHIFT
    smuPeripheralHFXO = _SMU_PPUPATDO_HFXO0_SHIFT
    smuPeripheralHFRCOO = _SMU_PPUPATDO_HFRCOO_SHIFT
    smuPeripheralFSRCO = _SMU_PPUPATDO_FSRCO_SHIFT
    smuPeripheralDPLL0 = _SMU_PPUPATDO_DPLL0_SHIFT
    smuPeripheralLFXO = _SMU_PPUPATDO_LFXO_SHIFT
    smuPeripheralLFRCO = _SMU_PPUPATDO_LFRCO_SHIFT
    smuPeripheralULFRCO = _SMU_PPUPATDO_ULFRCO_SHIFT
    smuPeripheralMSC = _SMU_PPUPATDO_MSC_SHIFT
    smuPeripheralICACHE0 = _SMU_PPUPATDO_ICACHE0_SHIFT
    smuPeripheralPRS = _SMU_PPUPATDO_PRS_SHIFT
    smuPeripheralGPIO = _SMU_PPUPATDO_GPIO_SHIFT
    smuPeripheralLDMA = _SMU_PPUPATDO_LDMA_SHIFT
    smuPeripheralLDMAXBAR = _SMU_PPUPATDO_LDMAXBAR_SHIFT
    smuPeripheralTIMER0 = _SMU_PPUPATDO_TIMER0_SHIFT
    smuPeripheralTIMER1 = _SMU_PPUPATDO_TIMER1_SHIFT
    smuPeripheralTIMER2 = _SMU_PPUPATDO_TIMER2_SHIFT
    smuPeripheralTIMER3 = _SMU_PPUPATDO_TIMER3_SHIFT
    smuPeripheralTIMER4 = _SMU_PPUPATDO_TIMER4_SHIFT
    smuPeripheralUSART0 = _SMU_PPUPATDO_USART0_SHIFT
    smuPeripheralUSART1 = _SMU_PPUPATDO_USART1_SHIFT
    smuPeripheralBURTC = _SMU_PPUPATDO_BURTC_SHIFT
    smuPeripheralI2C1 = _SMU_PPUPATDO_I2C1_SHIFT
    smuPeripheralCHIPTTESTCTRL = _SMU_PPUPATDO_CHIPTTESTCTRL_SHIFT
    smuPeripheralSYSCFGCFGNS = _SMU_PPUPATDO_SYSCFGCFGNS_SHIFT
    smuPeripheralSYSCFG = _SMU_PPUPATDO_SYSCFG_SHIFT
    smuPeripheralBURAM = _SMU_PPUPATDO_BURAM_SHIFT
    smuPeripheralGPCRC = _SMU_PPUPATDO_GPCRC_SHIFT
    smuPeripheralRTCC = 32 + _SMU_PPUPATD1_RTCC_SHIFT
    smuPeripheralDCDC = 32 + _SMU_PPUPATD1_DCDC_SHIFT
    smuPeripheralPDM = 32 + _SMU_PPUPATD1_PDM_SHIFT
    smuPeripheralLETIMER0 = 32 + _SMU_PPUPATD1_LETIMER0_SHIFT
    smuPeripheralIADC0 = 32 + _SMU_PPUPATD1_IADC0_SHIFT
    smuPeripheralI2C0 = 32 + _SMU_PPUPATD1_I2C0_SHIFT
    smuPeripheralWDOG0 = 32 + _SMU_PPUPATD1_WDOG0_SHIFT
    smuPeripheralAMUXCPO = 32 + _SMU_PPUPATD1_AMUXCPO_SHIFT
    smuPeripheralEUART0 = 32 + _SMU_PPUPATD1_EUART0_SHIFT
    smuPeripheralSMU = 32 + _SMU_PPUPATD1_SMU_SHIFT
    smuPeripheralSMUCFGNS = 32 + _SMU_PPUPATD1_SMUCFGNS_SHIFT
    smuPeripheralCRYPTOACC = 32 + _SMU_PPUPATD1_CRYPTOACC_SHIFT
    smuPeripheralEnd
}
```

SMU peripheral identifiers.

Functions

void **SMU_EnablePPU**(bool enable)
Enable or disable PPU of SMU.

void **SMU_Init**(const SMU_Init_TypeDef *init)
Initialize PPU of SMU.

void **SMU_SetPrivilegedAccess**(SMU_Peripheral_TypeDef peripheral, bool privileged)
Change access settings for a peripheral.

SMU_Peripheral_TypeDef **SMU_GetFaultingPeripheral**(void)
Get the ID of the peripheral that caused an access fault.

void **SMU_IntClear**(uint32_t flags)
Clear one or more pending SMU interrupts.

void **SMU_IntDisable**(uint32_t flags)
Disable one or more SMU interrupts.

void **SMU_IntEnable**(uint32_t flags)
Enable one or more SMU interrupts.

uint32_t **SMU_IntGet**(void)
Get pending SMU interrupts.

uint32_t **SMU_IntGetEnabled**(void)
Get enabled and pending SMU interrupt flags.

void

SMU_IntSet(uint32_t flags)

Set one or more pending SMU interrupts from SW.

void

SMU_SECURE_IRQHandler(void)

SMU secure IRQ Handler.

Macros

#define

SMU_INIT_DEFAULT undefined

Default SMU initialization structure settings.

Enumeration Documentation

SMU_Peripheral_TypeDef

SMU_Peripheral_TypeDef

SMU peripheral identifiers.

	Enumerator
smuPeripheralEMU	SMU peripheral identifier for EMU
smuPeripheralCMU	SMU peripheral identifier for CMU
smuPeripheralHFXO	SMU peripheral identifier for HFXO0
smuPeripheralHFRCO0	SMU peripheral identifier for HFRCO0
smuPeripheralFSRCO	SMU peripheral identifier for FSRCO
smuPeripheralDPLL0	SMU peripheral identifier for DPLL0
smuPeripheralLFXO	SMU peripheral identifier for LFXO
smuPeripheralLFRCO	SMU peripheral identifier for LFRCO
smuPeripheralULFRCO	SMU peripheral identifier for ULFRCO
smuPeripheralMSC	SMU peripheral identifier for MSC
smuPeripheralICACHE0	SMU peripheral identifier for ICACHE0
smuPeripheralPRS	SMU peripheral identifier for PRS
smuPeripheralGPIO	SMU peripheral identifier for GPIO
smuPeripheralLDMA	SMU peripheral identifier for LDMA
smuPeripheralLDMAXBAR	SMU peripheral identifier for LDMAXBAR
smuPeripheralTIMER0	SMU peripheral identifier for TIMER0
smuPeripheralTIMER1	SMU peripheral identifier for TIMER1
smuPeripheralTIMER2	SMU peripheral identifier for TIMER2
smuPeripheralTIMER3	SMU peripheral identifier for TIMER3
smuPeripheralTIMER4	SMU peripheral identifier for TIMER4
smuPeripheralUSART0	SMU peripheral identifier for USART0
smuPeripheralUSART1	SMU peripheral identifier for USART1
smuPeripheralBURTC	SMU peripheral identifier for BURTC
smuPeripheralI2C1	SMU peripheral identifier for I2C1
smuPeripheralCHIPTSTCTRL	SMU peripheral identifier for CHIPTSTCTRL.
smuPeripheralSYSCFGCFGNS	SMU peripheral identifier for SYSCFGCFGNS.
smuPeripheralSYSCFG	SMU peripheral identifier for SYSCFG

smuPeripheralBURAM	SMU peripheral identifier for BURAM
smuPeripheralGPCRC	SMU peripheral identifier for GPCRC
smuPeripheralRTCC	SMU peripheral identifier for RTCC
smuPeripheralDCDC	SMU peripheral identifier for DCDC
smuPeripheralPDM	SMU peripheral identifier for PDM
smuPeripheralLETIMER0	SMU peripheral identifier for LETIMER
smuPeripheralIADC0	SMU peripheral identifier for IADC0
smuPeripheralI2C0	SMU peripheral identifier for I2C0
smuPeripheralWDOG0	SMU peripheral identifier for WDOG0
smuPeripheralAMUXCP0	SMU peripheral identifier for AMUXCP0
smuPeripheralEUART0	SMU peripheral identifier for EUART0
smuPeripheralSMU	SMU peripheral identifier for SMU
smuPeripheralSMUCFGNS	SMU peripheral identifier for SMUCFGNS
smuPeripheralCRYPTOACC	SMU peripheral identifier for CRYPTOACC.
smuPeripheralEnd	SMU peripheral end.

Function Documentation

SMU_EnablePPU

```
void SMU_EnablePPU (bool enable)
```

Enable or disable PPU of SMU.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	Set to true to enable PPU; set to false otherwise.

SMU_Init

```
void SMU_Init (const SMU\_Init\_TypeDef * init)
```

Initialize PPU of SMU.

Parameters

Type	Direction	Argument Name	Description
const SMU_Init_TypeDef *	[in]	init	Pointer to initialization structure that defines which peripherals should only be accessed from privileged mode, and if PPU should be enabled.

SMU_SetPrivilegedAccess

```
void SMU_SetPrivilegedAccess (SMU\_Peripheral\_TypeDef peripheral, bool privileged)
```

Change access settings for a peripheral.

Parameters

Type	Direction	Argument Name	Description
SMU_Peripheral_TypeDef	[in]	peripheral	ID of the peripheral to change access settings for.
bool	[in]	privileged	Set to true if the peripheral should only be accessed from privileged mode; set to false otherwise.

Set to limit access of a peripheral from privileged mode.

SMU_GetFaultingPeripheral

```
SMU_Peripheral_TypeDef SMU_GetFaultingPeripheral (void )
```

Get the ID of the peripheral that caused an access fault.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- The return value is only valid if SMU_IF_PPUPRIV interrupt flag is set.

Returns

- ID of the peripheral that caused an access fault.

SMU_IntClear

```
void SMU_IntClear (uint32_t flags)
```

Clear one or more pending SMU interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	Bitwise logic OR of SMU interrupt sources to clear.

SMU_IntDisable

```
void SMU_IntDisable (uint32_t flags)
```

Disable one or more SMU interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	SMU interrupt sources to disable.

SMU_IntEnable

```
void SMU_IntEnable (uint32_t flags)
```

Enable one or more SMU interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	SMU interrupt sources to enable.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [SMU_IntClear\(\)](#) prior to enabling the interrupt.

SMU_IntGet

```
uint32_t SMU_IntGet (void )
```

Get pending SMU interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- SMU interrupt sources pending.

SMU_IntGetEnabled

```
uint32_t SMU_IntGetEnabled (void )
```

Get enabled and pending SMU interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by this function.

Returns

- Pending and enabled SMU interrupt sources. The return value is the bitwise AND combination of
 - the OR combination of enabled interrupt sources in SMU_IEN register and
 - the OR combination of valid interrupt flags in SMU_IF register.

SMU_IntSet

```
void SMU_IntSet (uint32_t flags)
```

Set one or more pending SMU interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	SMU interrupt sources to set to pending.

SMU_SECURE_IRQHandler

```
void SMU_SECURE_IRQHandler (void )
```

SMU secure IRQ Handler.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

When a PPU detects an access to a secure peripheral at its non-secure address or an access to a non-secure peripheral at its secure address, PPUSECIF in SMU_IF is set and the ID of the peripheral being accessed is written to SMU_PPUFS. If PPUSECIE is set and the SMU's Secure IRQ enabled, the CPU will be interrupted and SMU_SECURE_IRQHandler Will handle the interrupt.

Macro Definition Documentation

SMU_INIT_DEFAULT

```
#define SMU_INIT_DEFAULT
```

Value:

```
{
  { { 0 } },          /* No peripherals access protected. */
  true                /* Enable SMU.*/
}
```

Default SMU initialization structure settings.

SMU_PrivilegedAccess_TypeDef

SMU peripheral privileged access enablers.

Public Attributes

bool	privilegedSCRATCHPAD	Privileged access enabler for SCRATCHPAD
bool	privilegedEMU	Privileged access enabler for EMU
bool	privilegedCMU	Privileged access enabler for CMU
bool	privilegedHFXOO	Privileged access enabler for HFXOO
bool	privilegedHFRCOO	Privileged access enabler for HFRCOO
bool	privilegedFSRCO	Privileged access enabler for FSRCO
bool	privilegedDPLL0	Privileged access enabler for DPLL0
bool	privilegedLFXO	Privileged access enabler for LFXO
bool	privilegedLFRCO	Privileged access enabler for LFRCO
bool	privilegedULFRCO	Privileged access enabler for ULFRCO
bool	privilegedMSC	Privileged access enabler for MSC
bool	privilegedICACHE0	Privileged access enabler for ICACHE0
bool	privilegedPRS	Privileged access enabler for PRS0
bool	privilegedGPIO	Privileged access enabler for GPIO
bool	privilegedLDMA	Privileged access enabler for LDMA
bool	privilegedLDMAXBAR	Privileged access enabler for LDMAXBAR
bool	privilegedTIMERO	Privileged access enabler for TIMERO
bool	privilegedTIMER1	Privileged access enabler for TIMER1
bool	privilegedTIMER2	Privileged access enabler for TIMER2
bool	privilegedTIMER3	Privileged access enabler for TIMER3

bool	privilegedTIMER4 Privileged access enabler for TIMER4
bool	privilegedUSART0 Privileged access enabler for USART0
bool	privilegedUSART1 Privileged access enabler for USART1
bool	privilegedBURTC Privileged access enabler for BURTC
bool	privilegedI2C1 Privileged access enabler for I2C1
bool	privilegedCHIPTSTCTRL Privileged access enabler for CHIPTSTCTRL.
bool	privilegedSYSCFGCFGNS Privileged access enabler for SYSCFGCFGNS
bool	privilegedSYSCFG Privileged access enabler for SYSCFG
bool	privilegedBURAM Privileged access enabler for BURAM
bool	privilegedIFADCDEBUG Privileged access enabler for IFADCDEBUG
bool	privilegedGPCRC Privileged access enabler for GPCRC
bool	privilegedDCI Privileged access enabler for DCI
bool	privilegedROOTCFG Privileged access enabler for ROOTCFG
bool	privilegedDCDC Privileged access enabler for DCDC
bool	privilegedPDM Privileged access enabler for PDM
bool	privilegedRFSense Privileged access enabler for RFSense
bool	privilegedRADIOAES Privileged access enabler for RADIOAES
bool	privilegedSMU Privileged access enabler for SMU
bool	privilegedSMUCFGNS Privileged access enabler for SMUCFGNS
bool	privilegedRTCC Privileged access enabler for RTCC
bool	privilegedLETIMER0 Privileged access enabler for LETIMER0
bool	privilegedIADC0 Privileged access enabler for IADC0

bool	privilegedI2C0	Privileged access enabler for I2C0
bool	privilegedWDOG0	Privileged access enabler for WDOG0
bool	privilegedAMUXCP0	Privileged access enabler for AMUXCP0
bool	privilegedEUART0	Privileged access enabler for EUART0
bool	privilegedCRYPTOACC	Privileged access enabler for CRYPTOACC
bool	privilegedAHBRADIO	Privileged access enabler for AHBRADIO

Public Attribute Documentation

privilegedSCRATCHPAD

`bool SMU_PrivilegedAccess_TypeDef::privilegedSCRATCHPAD`

Privileged access enabler for SCRATCHPAD

privilegedEMU

`bool SMU_PrivilegedAccess_TypeDef::privilegedEMU`

Privileged access enabler for EMU

privilegedCMU

`bool SMU_PrivilegedAccess_TypeDef::privilegedCMU`

Privileged access enabler for CMU

privilegedHFXOO

`bool SMU_PrivilegedAccess_TypeDef::privilegedHFXOO`

Privileged access enabler for HFXOO

privilegedHFRCOO

`bool SMU_PrivilegedAccess_TypeDef::privilegedHFRCOO`

Privileged access enabler for HFRCOO

privilegedFSRCO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedFSRCO
```

Privileged access enabler for FSRCO

privilegedDPLLO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedDPLLO
```

Privileged access enabler for DPLLO

privilegedLFXO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLFXO
```

Privileged access enabler for LFXO

privilegedLFRCO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLFRCO
```

Privileged access enabler for LFRCO

privilegedULFRCO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedULFRCO
```

Privileged access enabler for ULFRCO

privilegedMSC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedMSC
```

Privileged access enabler for MSC

privilegedICACHE0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedICACHE0
```

Privileged access enabler for ICACHE0

privilegedPRS

```
bool SMU_PrivilegedAccess_TypeDef::privilegedPRS
```

Privileged access enabler for PRSO

privilegedGPIO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedGPIO
```

Privileged access enabler for GPIO

privilegedLDMA

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLDMA
```

Privileged access enabler for LDMA

privilegedLDMAXBAR

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLDMAXBAR
```

Privileged access enabler for LDMAXBAR

privilegedTIMER0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedTIMER0
```

Privileged access enabler for TIMER0

privilegedTIMER1

```
bool SMU_PrivilegedAccess_TypeDef::privilegedTIMER1
```

Privileged access enabler for TIMER1

privilegedTIMER2

```
bool SMU_PrivilegedAccess_TypeDef::privilegedTIMER2
```

Privileged access enabler for TIMER2

privilegedTIMER3

```
bool SMU_PrivilegedAccess_TypeDef::privilegedTIMER3
```

Privileged access enabler for TIMER3

privilegedTIMER4

```
bool SMU_PrivilegedAccess_TypeDef::privilegedTIMER4
```

Privileged access enabler for TIMER4

privilegedUSART0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedUSART0
```

Privileged access enabler for USART0

privilegedUSART1

```
bool SMU_PrivilegedAccess_TypeDef::privilegedUSART1
```

Privileged access enabler for USART1

privilegedBURTC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedBURTC
```

Privileged access enabler for BURTC

privilegedI2C1

```
bool SMU_PrivilegedAccess_TypeDef::privilegedI2C1
```

Privileged access enabler for I2C1

privilegedCHIPTTESTCTRL

```
bool SMU_PrivilegedAccess_TypeDef::privilegedCHIPTTESTCTRL
```

Privileged access enabler for CHIPTTESTCTRL.

privilegedSYSCFGCFGNS

```
bool SMU_PrivilegedAccess_TypeDef::privilegedSYSCFGCFGNS
```

Privileged access enabler for SYSCFGCFGNS

privilegedSYSCFG

```
bool SMU_PrivilegedAccess_TypeDef::privilegedSYSCFG
```

Privileged access enabler for SYSCFG

privilegedBURAM

```
bool SMU_PrivilegedAccess_TypeDef::privilegedBURAM
```

Privileged access enabler for BURAM

privilegedIFADCDEBUG

```
bool SMU_PrivilegedAccess_TypeDef::privilegedIFADCDEBUG
```

Privileged access enabler for IFADCDEBUG

privilegedGPCRC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedGPCRC
```

Privileged access enabler for GPCRC

privilegedDCI

```
bool SMU_PrivilegedAccess_TypeDef::privilegedDCI
```

Privileged access enabler for DCI

privilegedROOTCFG

```
bool SMU_PrivilegedAccess_TypeDef::privilegedROOTCFG
```

Privileged access enabler for ROOTCFG

privilegedDCDC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedDCDC
```

Privileged access enabler for DCDC

privilegedPDM

```
bool SMU_PrivilegedAccess_TypeDef::privilegedPDM
```

Privileged access enabler for PDM

privilegedRFSense

```
bool SMU_PrivilegedAccess_TypeDef::privilegedRFSense
```

Privileged access enabler for RFSense

privilegedRadioAES

```
bool SMU_PrivilegedAccess_TypeDef::privilegedRadioAES
```

Privileged access enabler for RadioAES

privilegedSMU

```
bool SMU_PrivilegedAccess_TypeDef::privilegedSMU
```

Privileged access enabler for SMU

privilegedSMUCFGNS

```
bool SMU_PrivilegedAccess_TypeDef::privilegedSMUCFGNS
```

Privileged access enabler for SMUCFGNS

privilegedRTCC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedRTCC
```

Privileged access enabler for RTCC

privilegedLETIMERO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLETIMERO
```

Privileged access enabler for LETIMERO

privilegedIADC0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedIADC0
```

Privileged access enabler for IADC0

privilegedI2C0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedI2C0
```

Privileged access enabler for I2C0

privilegedWDOG0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedWDOG0
```

Privileged access enabler for WDOG0

privilegedAMUXCP0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedAMUXCP0
```

Privileged access enabler for AMUXCP0

privilegedEUART0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedEUART0
```

Privileged access enabler for EUART0

privilegedCRYPTOACC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedCRYPTOACC
```

Privileged access enabler for CRYPTOACC

privilegedAHBRADIO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedAHBRADIO
```

Privileged access enabler for AHBRADIO

SMU_Init_TypeDef

SMU initialization structure.

Public Attributes

uint32_t	reg	Peripheral access control array.
SMU_PrivilegedAccess_TypeDef	access	Peripheral access control array.
union SMU_Init_TypeDef::@1	ppu	PPU init array.
bool	enable	SMU enable flag.

Public Attribute Documentation

reg

```
uint32_t SMU_Init_TypeDef::reg[2]
```

Peripheral access control array.

access

```
SMU_PrivilegedAccess_TypeDef SMU_Init_TypeDef::access
```

Peripheral access control array.

ppu

```
union SMU_Init_TypeDef::@1 SMU_Init_TypeDef::ppu
```

PPU init array.

enable

```
bool SMU_Init_TypeDef::enable
```

SMU enable flag.

When set, [SMU_Init\(\)](#) will enable SMU.

SYSTEM - System Utils

SYSTEM - System Utils

System API.

This module contains functions to read information such as RAM and Flash size, device unique ID, chip revision, family, and part number from DEVINFO and SCB blocks. Functions to configure and read status from FPU are available for compatible devices.

Modules

[SYSTEM_ChipRevision_TypeDef](#)

[SYSTEM_CalAddrVal_TypeDef](#)

Enumerations

```
enum    SYSTEM\_PartFamily\_TypeDef {
    systemPartFamilyMighty22 = DEVINFO_PART_FAMILY_MG | (22 <<
        _DEVINFO_PART_FAMILYNUM_SHIFT)
    systemPartFamilyFlex22 = DEVINFO_PART_FAMILY_FG | (22 << _DEVINFO_PART_FAMILYNUM_SHIFT)
    systemPartFamilyBlue22 = DEVINFO_PART_FAMILY_BG | (22 <<
        _DEVINFO_PART_FAMILYNUM_SHIFT)
    systemPartFamilyEfm32Pearl22 = DEVINFO_PART_FAMILY_PG | (22 <<
        _DEVINFO_PART_FAMILYNUM_SHIFT)
    systemPartFamilyUnknown = 0xFF
}
Family identifiers.
```

```
enum    SYSTEM\_FpuAccess\_TypeDef {
    fpuAccessDenied = (0x0 << 20)
    fpuAccessPrivilegedOnly = (0x5 << 20)
    fpuAccessReserved = (0xA << 20)
    fpuAccessFull = (0xF << 20)
}
Floating point co-processor access modes.
```

```
enum    SYSTEM\_SecurityCapability\_TypeDef {
    securityCapabilityUnknown
    securityCapabilityNA
    securityCapabilityBasic
    securityCapabilityRoT
    securityCapabilitySE
    securityCapabilityVault
}
Family security capability.
```

Functions

```
void    SYSTEM\_ChipRevisionGet(SYSTEM_ChipRevision_TypeDef *rev)
Get a chip major/minor revision.
```

```
SYSTEM\_PartFamily\_TypeDef SYSTEM\_GetFamily(void)
Get the MCU family identifier.
```

```
uint8_t SYSTEM\_GetDevinfoRev(void)
Get DEVINFO revision.
```

void	SYSTEM_FpuAccessModeSet (SYSTEM_FpuAccess_TypeDef accessMode) Set floating point co-processor (FPU) access mode.
bool	SYSTEM_GetCalibrationValue (volatile uint32_t *regAddress) Get a factory calibration value for a given peripheral register.
SYSTEM_SecurityCapability_TypeDef	SYSTEM_GetSecurityCapability (void) Get family security capability.
uint64_t	SYSTEM_GetUnique (void) Get the unique number for this device.
uint8_t	SYSTEM_GetProdRev (void) Get the production revision for this part.
uint32_t	SYSTEM_GetSRAMBaseAddress (void) Get the SRAM Base Address.
uint16_t	SYSTEM_GetSRAMSize (void) Get the SRAM size (in KB).
uint16_t	SYSTEM_GetFlashSize (void) Get the flash size (in KB).
uint32_t	SYSTEM_GetFlashPageSize (void) Get the flash page size in bytes.
uint16_t	SYSTEM_GetPartNumber (void) Get the MCU part number.
uint8_t	SYSTEM_GetCalibrationTemperature (void) Get the calibration temperature (in degrees Celsius).

Enumeration Documentation

SYSTEM_PartFamily_TypeDef

SYSTEM_PartFamily_TypeDef

Family identifiers.

Enumerator	
systemPartFamilyMighty22	EFR32 Mighty Gecko Series 2 Config 2 Value Device Family.
systemPartFamilyFlex22	EFR32 Flex Gecko Series 2 Config 2 Value Device Family.
systemPartFamilyBlue22	EFR32 Blue Gecko Series 2 Config 2 Value Device Family.
systemPartFamilyEfm32Pearl22	EFM32 Pearl Gecko Series 2 Config 2 Value Device Family.
systemPartFamilyUnknown	Unknown Device Family.

SYSTEM_FpuAccess_TypeDef

SYSTEM_FpuAccess_TypeDef

Floating point co-processor access modes.

Enumerator	
fpuAccessDenied	Access denied, any attempted access generates a NOCP UsageFault.
fpuAccessPrivilegedOnly	Privileged access only, an unprivileged access generates a NOCP UsageFault.
fpuAccessReserved	Reserved.

fpuAccessFull	Full access.
---------------	--------------

SYSTEM_SecurityCapability_TypeDef

SYSTEM_SecurityCapability_TypeDef

Family security capability.

	Enumerator
securityCapabilityUnknown	Unknown security capability.
securityCapabilityNA	Security capability not applicable.
securityCapabilityBasic	Basic security capability.
securityCapabilityRoT	Root of Trust security capability.
securityCapabilitySE	Secure Element security capability.
securityCapabilityVault	Secure Vault security capability.

Function Documentation

SYSTEM_ChipRevisionGet

void SYSTEM_ChipRevisionGet (SYSTEM_ChipRevision_TypeDef * rev)

Get a chip major/minor revision.

Parameters

Type	Direction	Argument Name	Description
SYSTEM_ChipRevision_TypeDef *	[out]	rev	A location to place the chip revision information.

SYSTEM_GetFamily

SYSTEM_PartFamily_TypeDef SYSTEM_GetFamily (void)

Get the MCU family identifier.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function retrieves family ID by reading the chip's device info structure in flash memory. Users can retrieve family ID directly by reading DEVINFO->PART item and decode with mask and shift #defines defined in <part_family>_devinfo.h (refer to code below for details).

Returns

- Family identifier of MCU.

SYSTEM_GetDevinfoRev

```
uint8_t SYSTEM_GetDevinfoRev (void )
```

Get DEVINFO revision.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Revision of the DEVINFO contents.

SYSTEM_FpuAccessModeSet

```
void SYSTEM_FpuAccessModeSet (SYSTEM_FpuAccess_TypeDef accessMode)
```

Set floating point co-processor (FPU) access mode.

Parameters

Type	Direction	Argument Name	Description
SYSTEM_FpuAccess_TypeDef	[in]	accessMode	Floating point co-processor access mode. See SYSTEM_FpuAccess_TypeDef for details.

SYSTEM_GetCalibrationValue

```
bool SYSTEM_GetCalibrationValue (volatile uint32_t * regAddress)
```

Get a factory calibration value for a given peripheral register.

Parameters

Type	Direction	Argument Name	Description
volatile uint32_t *	[in]	regAddress	The peripheral calibration register address to get a calibration value for. If the calibration value is found, this register is updated with the calibration value.

Returns

- True if a calibration value exists, false otherwise.

SYSTEM_GetSecurityCapability

```
SYSTEM_SecurityCapability_TypeDef SYSTEM_GetSecurityCapability (void )
```

Get family security capability.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function retrieves the family security capability based on the device number. The device number is one letter and 3 digits: DEVICENUMBER = (alpha-'A')*1000 + numeric. i.e. 0d = "A000"; 1123d = "B123". The security capabilities are represented by [SYSTEM_SecurityCapability_TypeDef](#).

Returns

- Security capability of MCU.

SYSTEM_GetUnique

```
uint64_t SYSTEM_GetUnique (void )
```

Get the unique number for this device.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Unique number for this device.

SYSTEM_GetProdRev

```
uint8_t SYSTEM_GetProdRev (void )
```

Get the production revision for this part.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Production revision for this part.

SYSTEM_GetSRAMBaseAddress

```
uint32_t SYSTEM_GetSRAMBaseAddress (void )
```

Get the SRAM Base Address.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function is used to retrieve the base address of the SRAM.

Returns

- Base address SRAM (32-bit unsigned integer).

SYSTEM_GetSRAMSize

```
uint16_t SYSTEM_GetSRAMSize (void )
```

Get the SRAM size (in KB).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function retrieves SRAM size by reading the chip device info structure. If your binary is made for one specific device only, use SRAM_SIZE instead.

Returns

- Size of internal SRAM (in KB).

SYSTEM_GetFlashSize

```
uint16_t SYSTEM_GetFlashSize (void )
```

Get the flash size (in KB).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function retrieves flash size by reading the chip device info structure. If your binary is made for one specific device only, use FLASH_SIZE instead.

Returns

- Size of internal flash (in KB).

SYSTEM_GetFlashPageSize

```
uint32_t SYSTEM_GetFlashPageSize (void )
```

Get the flash page size in bytes.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function retrieves flash page size by reading the chip device info structure. If your binary is made for one specific device only, use FLASH_PAGE_SIZE instead.

Returns

- Page size of internal flash in bytes.

SYSTEM_GetPartNumber

```
uint16_t SYSTEM_GetPartNumber (void )
```

Get the MCU part number.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The part number of MCU.

SYSTEM_GetCalibrationTemperature

```
uint8_t SYSTEM_GetCalibrationTemperature (void )
```

Get the calibration temperature (in degrees Celsius).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Calibration temperature in Celsius.

SYSTEM_ChipRevision_TypeDef

Chip revision details.

Public Attributes

uint8_t	minor	Minor revision number.
uint8_t	major	Major revision number.
uint8_t	family	Device family number.

Public Attribute Documentation

minor

```
uint8_t SYSTEM_ChipRevision_TypeDef::minor
```

Minor revision number.

major

```
uint8_t SYSTEM_ChipRevision_TypeDef::major
```

Major revision number.

family

```
uint8_t SYSTEM_ChipRevision_TypeDef::family
```

Device family number.

SYSTEM_CalAddrVal_TypeDef

DEVINFO calibration address/value pair.

Public Attributes

uint32_t	address	Peripheral calibration register address.
uint32_t	calValue	Calibration value for register at address.

Public Attribute Documentation

address

```
uint32_t SYSTEM_CalAddrVal_TypeDef::address
```

Peripheral calibration register address.

calValue

```
uint32_t SYSTEM_CalAddrVal_TypeDef::calValue
```

Calibration value for register at address.

TIMER - Timer/Counter

TIMER - Timer/Counter

Timer/Counter (TIMER) Peripheral API.

The timer module consists of three main parts:

- General timer configuration and enable control.
- Compare/capture control.
- Dead time insertion control (may not be available for all timers).

Modules

[TIMER_Init_TypeDef](#)

[TIMER_InitCC_TypeDef](#)

[TIMER_InitDTI_TypeDef](#)

Enumerations

```
enum    TIMER\_CCMODE\_TypeDef {
    timerCCModeOff = _TIMER_CC_CFG_MODE_OFF
    timerCCModeCapture = _TIMER_CC_CFG_MODE_INPUTCAPTURE
    timerCCModeCompare = _TIMER_CC_CFG_MODE_OUTPUTCOMPARE
    timerCCModePWM = _TIMER_CC_CFG_MODE_PWM
}
Timer compare/capture mode.

enum    TIMER\_CLKSEL\_TypeDef {
    timerClkSelHFPerClk = _TIMER_CFG_CLKSEL_PRESCEM01GRPACLK
    timerClkSelCC1 = _TIMER_CFG_CLKSEL_CC1
    timerClkSelCascade = _TIMER_CFG_CLKSEL_TIMEROUF
}
Clock select.

enum    TIMER\_EDGE\_TypeDef {
    timerEdgeRising = _TIMER_CC_CTRL_ICEDGE_RISING
    timerEdgeFalling = _TIMER_CC_CTRL_ICEDGE_FALLING
    timerEdgeBoth = _TIMER_CC_CTRL_ICEDGE_BOTH
    timerEdgeNone = _TIMER_CC_CTRL_ICEDGE_NONE
}
Input capture edge select.

enum    TIMER\_EVENT\_TypeDef {
    timerEventEveryEdge = _TIMER_CC_CTRL_ICEVCTRL_EVERYEDGE
    timerEventEvery2ndEdge = _TIMER_CC_CTRL_ICEVCTRL_EVERYSECONDEGE
    timerEventRising = _TIMER_CC_CTRL_ICEVCTRL_RISING
    timerEventFalling = _TIMER_CC_CTRL_ICEVCTRL_FALLING
}
Input capture event control.
```

```
enum    TIMER_InputAction_TypeDef {
    timerInputActionNone = _TIMER_CTRL_FALLA_NONE
    timerInputActionStart = _TIMER_CTRL_FALLA_START
    timerInputActionStop = _TIMER_CTRL_FALLA_STOP
    timerInputActionReloadStart = _TIMER_CTRL_FALLA_RELOADSTART
}
Input edge action.

enum    TIMER_Mode_TypeDef {
    timerModeUp = _TIMER_CFG_MODE_UP
    timerModeDown = _TIMER_CFG_MODE_DOWN
    timerModeUpDown = _TIMER_CFG_MODE_UPDOWN
    timerModeQDec = _TIMER_CFG_MODE_QDEC
}
Timer mode.

enum    TIMER_OutputAction_TypeDef {
    timerOutputActionNone = _TIMER_CC_CTRL_CUFOA_NONE
    timerOutputActionToggle = _TIMER_CC_CTRL_CUFOA_TOGGLE
    timerOutputActionClear = _TIMER_CC_CTRL_CUFOA_CLEAR
    timerOutputActionSet = _TIMER_CC_CTRL_CUFOA_SET
}
Compare/capture output action.

enum    TIMER_Prescale_TypeDef {
    timerPrescale1 = _TIMER_CFG_PRESC_DIV1
    timerPrescale2 = _TIMER_CFG_PRESC_DIV2
    timerPrescale4 = _TIMER_CFG_PRESC_DIV4
    timerPrescale8 = _TIMER_CFG_PRESC_DIV8
    timerPrescale16 = _TIMER_CFG_PRESC_DIV16
    timerPrescale32 = _TIMER_CFG_PRESC_DIV32
    timerPrescale64 = _TIMER_CFG_PRESC_DIV64
    timerPrescale128 = _TIMER_CFG_PRESC_DIV128
    timerPrescale256 = _TIMER_CFG_PRESC_DIV256
    timerPrescale512 = _TIMER_CFG_PRESC_DIV512
    timerPrescale1024 = _TIMER_CFG_PRESC_DIV1024
}
Prescaler.

enum    TIMER_PrsInput_TypeDef {
    timerPrsInputNone = 0x0
    timerPrsInputSync = _TIMER_CC_CFG_INSEL_PRSSYNC
    timerPrsInputAsyncLevel = _TIMER_CC_CFG_INSEL_PRASYNCLEVEL
    timerPrsInputAsyncPulse = _TIMER_CC_CFG_INSEL_PRASYNCPULSE
}
PRS input type.

enum    TIMER_DtiFaultAction_TypeDef {
    timerDtiFaultActionNone = _TIMER_DTFCFG_DTFA_NONE
    timerDtiFaultActionInactive = _TIMER_DTFCFG_DTFA_INACTIVE
    timerDtiFaultActionClear = _TIMER_DTFCFG_DTFA_CLEAR
    timerDtiFaultActionTristate = _TIMER_DTFCFG_DTFA_TRISTATE
}
DT (Dead Time) Fault Actions.

enum    TIMER_PrsOutput_t {
    timerPrsOutputPulse = 0
    timerPrsOutputLevel = 1
    timerPrsOutputDefault = timerPrsOutputPulse
}
PRS Output configuration.
```

Typedefs

```
typedef uint8_t    TIMER_PRSSEL_TypeDef
Peripheral Reflex System signal.
```

Functions

void	TIMER_Init (TIMER_TypeDef *timer, const TIMER_Init_TypeDef *init) Initialize TIMER.
void	TIMER_InitCC (TIMER_TypeDef *timer, unsigned int ch, const TIMER_InitCC_TypeDef *init) Initialize the TIMER compare/capture channel.
void	TIMER_InitDTI (TIMER_TypeDef *timer, const TIMER_InitDTI_TypeDef *init) Initialize the TIMER DTI unit.
void	TIMER_Reset (TIMER_TypeDef *timer) Reset the TIMER to the same state that it was in after a hardware reset.
void	TIMER_SyncWait (TIMER_TypeDef *timer) Wait for pending synchronization to finish.
bool	TIMER_Valid (const TIMER_TypeDef *ref) Validate the TIMER register block pointer.
bool	TIMER_SupportsDTI (const TIMER_TypeDef *ref) Check whether the TIMER is valid and supports Dead Timer Insertion (DTI).
uint32_t	TIMER_MaxCount (const TIMER_TypeDef *ref) Get the Max count of the timer.
uint32_t	TIMER_CaptureGet (TIMER_TypeDef *timer, unsigned int ch) Get compare/capture value for the compare/capture channel.
uint32_t	TIMER_CaptureBufGet (TIMER_TypeDef *timer, unsigned int ch) Get the buffered compare/capture value for compare/capture channel.
void	TIMER_CompareBufSet (TIMER_TypeDef *timer, unsigned int ch, uint32_t val) Set the compare value buffer for the compare/capture channel when operating in compare or PWM mode.
void	TIMER_CompareSet (TIMER_TypeDef *timer, unsigned int ch, uint32_t val) Set the compare value for compare/capture channel when operating in compare or PWM mode.
uint32_t	TIMER_CounterGet (TIMER_TypeDef *timer) Get the TIMER counter value.
void	TIMER_CounterSet (TIMER_TypeDef *timer, uint32_t val) Set the TIMER counter value.
void	TIMER_Enable (TIMER_TypeDef *timer, bool enable) Start/stop TIMER.
void	TIMER_EnabledDTI (TIMER_TypeDef *timer, bool enable) Enable or disable DTI unit.
uint32_t	TIMER_GetDTIFault (TIMER_TypeDef *timer) Get DTI fault source flags status.
void	TIMER_ClearDTIFault (TIMER_TypeDef *timer, uint32_t flags) Clear DTI fault source flags.
void	TIMER_IntClear (TIMER_TypeDef *timer, uint32_t flags) Clear one or more pending TIMER interrupts.
void	TIMER_IntDisable (TIMER_TypeDef *timer, uint32_t flags) Disable one or more TIMER interrupts.
void	TIMER_IntEnable (TIMER_TypeDef *timer, uint32_t flags) Enable one or more TIMER interrupts.

uint32_t	TIMER_IntGet (TIMER_TypeDef *timer) Get pending TIMER interrupt flags.
uint32_t	TIMER_IntGetEnabled (TIMER_TypeDef *timer) Get enabled and pending TIMER interrupt flags.
void	TIMER_IntSet (TIMER_TypeDef *timer, uint32_t flags) Set one or more pending TIMER interrupts from SW.
void	TIMER_TopBufSet (TIMER_TypeDef *timer, uint32_t val) Set the top value buffer for the timer.
uint32_t	TIMER_TopGet (TIMER_TypeDef *timer) Get the top value setting for the timer.
void	TIMER_TopSet (TIMER_TypeDef *timer, uint32_t val) Set the top value for timer.

Macros

#define	TIMER_INIT_DEFAULT undefined Default configuration for TIMER initialization structure.
#define	TIMER_INITCC_DEFAULT undefined Default configuration for TIMER compare/capture initialization structure.
#define	TIMER_INITDTI_DEFAULT undefined Default configuration for TIMER DTI initialization structure.

Enumeration Documentation

TIMER_CCMODE_TypeDef

TIMER_CCMODE_TypeDef

Timer compare/capture mode.

Enumerator	
timerCCModeOff	Channel turned off.
timerCCModeCapture	Input capture.
timerCCModeCompare	Output compare.
timerCCModePWM	Pulse-Width modulation.

TIMER_CLKSEL_TypeDef

TIMER_CLKSEL_TypeDef

Clock select.

Enumerator	
timerClkSelHFPerClk	Prescaled EM01GRPA clock.
timerClkSelCC1	Compare/Capture Channel 1 Input.
timerClkSelCascade	Cascaded clocked by underflow or overflow by lower numbered timer.

TIMER_EDGE_TypeDef

TIMER_Edge_TypeDef

Input capture edge select.

Enumerator	
timerEdgeRising	Rising edges detected.
timerEdgeFalling	Falling edges detected.
timerEdgeBoth	Both edges detected.
timerEdgeNone	No edge detection, leave signal as is.

TIMER_Event_TypeDef

TIMER_Event_TypeDef

Input capture event control.

Enumerator	
timerEventEveryEdge	PRS output pulse, interrupt flag, and DMA request set on every capture.
timerEventEvery2ndEdge	PRS output pulse, interrupt flag, and DMA request set on every second capture.
timerEventRising	PRS output pulse, interrupt flag, and DMA request set on rising edge (if input capture edge = BOTH).
timerEventFalling	PRS output pulse, interrupt flag, and DMA request set on falling edge (if input capture edge = BOTH).

TIMER_InputAction_TypeDef

TIMER_InputAction_TypeDef

Input edge action.

Enumerator	
timerInputActionNone	No action taken.
timerInputActionStart	Start counter without reload.
timerInputActionStop	Stop counter without reload.
timerInputActionReloadStart	Reload and start counter.

TIMER_Mode_TypeDef

TIMER_Mode_TypeDef

Timer mode.

Enumerator	
timerModeUp	Up-counting.
timerModeDown	Down-counting.
timerModeUpDown	Up/down-counting.
timerModeQDec	Quadrature decoder.

TIMER_OutputAction_TypeDef

TIMER_OutputAction_TypeDef

Compare/capture output action.

Enumerator

timerOutputActionNone	No action.
timerOutputActionToggle	Toggle on event.
timerOutputActionClear	Clear on event.
timerOutputActionSet	Set on event.

TIMER_Prescale_TypeDef

TIMER_Prescale_TypeDef

Prescaler.

Enumerator

timerPrescale1	Divide by 1.
timerPrescale2	Divide by 2.
timerPrescale4	Divide by 4.
timerPrescale8	Divide by 8.
timerPrescale16	Divide by 16.
timerPrescale32	Divide by 32.
timerPrescale64	Divide by 64.
timerPrescale128	Divide by 128.
timerPrescale256	Divide by 256.
timerPrescale512	Divide by 512.
timerPrescale1024	Divide by 1024.

TIMER_PrsInput_TypeDef

TIMER_PrsInput_TypeDef

PRS input type.

Enumerator

timerPrsInputNone	No PRS input.
timerPrsInputSync	Synchronous PRS selected.
timerPrsInputAsyncLevel	Asynchronous level PRS selected.
timerPrsInputAsyncPulse	Asynchronous pulse PRS selected.

TIMER_DtiFaultAction_TypeDef

TIMER_DtiFaultAction_TypeDef

DT (Dead Time) Fault Actions.

Enumerator	
timerDtiFaultActionNone	No action on fault.
timerDtiFaultActionInactive	Set outputs inactive.
timerDtiFaultActionClear	Clear outputs.
timerDtiFaultActionTristate	Tristate outputs.

TIMER_PrsOutput_t

TIMER_PrsOutput_t

PRS Output configuration.

Enumerator	
timerPrsOutputPulse	Pulse PRS output from a channel.
timerPrsOutputLevel	PRS output follows CC out level.
timerPrsOutputDefault	Default PRS output behavior.

Typedef Documentation

TIMER_PRSSEL_TypeDef

typedef uint8_t TIMER_PRSSEL_TypeDef

Peripheral Reflex System signal.

Function Documentation

TIMER_Init

void TIMER_Init (TIMER_TypeDef * timer, const [TIMER_Init_TypeDef](#) * init)

Initialize TIMER.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	A pointer to the TIMER peripheral register block.
const TIMER_Init_TypeDef *	[in]	init	A pointer to the TIMER initialization structure.

Notice that the counter top must be configured separately with, for instance [TIMER_TopSet\(\)](#). In addition, compare/capture and dead-time insertion initialization must be initialized separately if used, which should probably be done prior to using this function if configuring the TIMER to start when initialization is completed.

TIMER_InitCC

void TIMER_InitCC (TIMER_TypeDef * timer, unsigned int ch, const [TIMER_InitCC_TypeDef](#) * init)

Initialize the TIMER compare/capture channel.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	A pointer to the TIMER peripheral register block.
unsigned int	[in]	ch	A compare/capture channel to initialize for.
const TIMER_InitCC_TypeDef *	[in]	init	A pointer to the TIMER initialization structure.

Notice that if operating the channel in compare mode, the CCV and CCVB register must be set separately, as required.

TIMER_InitDTI

```
void TIMER_InitDTI (TIMER_TypeDef * timer, const TIMER\_InitDTI\_TypeDef * init)
```

Initialize the TIMER DTI unit.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	A pointer to the TIMER peripheral register block.
const TIMER_InitDTI_TypeDef *	[in]	init	A pointer to the TIMER DTI initialization structure.

TIMER_Reset

```
void TIMER_Reset (TIMER_TypeDef * timer)
```

Reset the TIMER to the same state that it was in after a hardware reset.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	A pointer to the TIMER peripheral register block.

Note

- The ROUTE register is NOT reset by this function to allow for a centralized setup of this feature.

TIMER_SyncWait

```
void TIMER_SyncWait (TIMER_TypeDef * timer)
```

Wait for pending synchronization to finish.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	

TIMER_Valid

```
bool TIMER_Valid (const TIMER_TypeDef * ref)
```

Validate the TIMER register block pointer.

Parameters

Type	Direction	Argument Name	Description
const TIMER_TypeDef *	[in]	ref	Pointer to the TIMER peripheral register block.

Returns

- True if ref points to a valid timer, false otherwise.

TIMER_SupportsDTI

```
bool TIMER_SupportsDTI (const TIMER_TypeDef * ref)
```

Check whether the TIMER is valid and supports Dead Timer Insertion (DTI).

Parameters

Type	Direction	Argument Name	Description
const TIMER_TypeDef *	[in]	ref	Pointer to the TIMER peripheral register block.

Returns

- True if ref points to a valid timer that supports DTI, false otherwise.

TIMER_MaxCount

```
uint32_t TIMER_MaxCount (const TIMER_TypeDef * ref)
```

Get the Max count of the timer.

Parameters

Type	Direction	Argument Name	Description
const TIMER_TypeDef *	[in]	ref	Pointer to the TIMER peripheral register block.

Returns

- The max count value of the timer. This is 0xFFFF for 16 bit timers and 0xFFFFFFFF for 32 bit timers.

TIMER_CaptureGet

```
uint32_t TIMER_CaptureGet (TIMER_TypeDef * timer, unsigned int ch)
```

Get compare/capture value for the compare/capture channel.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
unsigned int	[in]	ch	Compare/capture channel to access.

Returns

- Current capture value.

TIMER_CaptureBufGet

```
uint32_t TIMER_CaptureBufGet (TIMER_TypeDef * timer, unsigned int ch)
```

Get the buffered compare/capture value for compare/capture channel.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
unsigned int	[in]	ch	Compare/capture channel to access.

Returns

- Current buffered capture value.

TIMER_CompareBufSet

```
void TIMER_CompareBufSet (TIMER_TypeDef * timer, unsigned int ch, uint32_t val)
```

Set the compare value buffer for the compare/capture channel when operating in compare or PWM mode.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
unsigned int	[in]	ch	Compare/capture channel to access.
uint32_t	[in]	val	Value to set in compare value buffer register.

The compare value buffer holds the value which will be written to TIMERN_CCx_CCV on an update event if the buffer has been updated since the last event.

TIMER_CompareSet

```
void TIMER_CompareSet (TIMER_TypeDef * timer, unsigned int ch, uint32_t val)
```

Set the compare value for compare/capture channel when operating in compare or PWM mode.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
unsigned int	[in]	ch	Compare/capture channel to access.
uint32_t	[in]	val	Value to set in compare value register.

TIMER_CounterGet

```
uint32_t TIMER_CounterGet (TIMER_TypeDef * timer)
```

Get the TIMER counter value.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to TIMER peripheral register block.

Returns

- Current TIMER counter value.

TIMER_CounterSet

```
void TIMER_CounterSet (TIMER_TypeDef * timer, uint32_t val)
```

Set the TIMER counter value.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
uint32_t	[in]	val	Value to set counter to.

TIMER_Enable

```
void TIMER_Enable (TIMER_TypeDef * timer, bool enable)
```

Start/stop TIMER.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
bool	[in]	enable	Set to true to enable counting; set to false otherwise.

TIMER_EnabledDTI

```
void TIMER_EnabledDTI (TIMER_TypeDef * timer, bool enable)
```

Enable or disable DTI unit.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
bool	[in]	enable	Set to true to enable DTI unit; set to false otherwise.

TIMER_GetDTIFault

```
uint32_t TIMER_GetDTIFault (TIMER_TypeDef * timer)
```

Get DTI fault source flags status.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.

Note

- Event bits are not cleared by this function.

Returns

- Status of the DTI fault source flags. Returns one or more valid DTI fault source flags (TIMER_DTFAULT_nnn) OR'ed together.

TIMER_ClearDTIFault

```
void TIMER_ClearDTIFault (TIMER_TypeDef * timer, uint32_t flags)
```

Clear DTI fault source flags.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
uint32_t	[in]	flags	DTI fault source(s) to clear. Use one or more valid DTI fault source flags (TIMER_DTFAULT_nnn) OR'ed together.

TIMER_IntClear

```
void TIMER_IntClear (TIMER_TypeDef * timer, uint32_t flags)
```

Clear one or more pending TIMER interrupts.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
uint32_t	[in]	flags	Pending TIMER interrupt source(s) to clear. Use one or more valid interrupt flags for the TIMER module (TIMER_IF_nnn) OR'ed together.

TIMER_IntDisable

```
void TIMER_IntDisable (TIMER_TypeDef * timer, uint32_t flags)
```

Disable one or more TIMER interrupts.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
uint32_t	[in]	flags	TIMER interrupt source(s) to disable. Use one or more valid interrupt flags for the TIMER module (TIMER_IF_nnn) OR'ed together.

TIMER_IntEnable

```
void TIMER_IntEnable (TIMER_TypeDef * timer, uint32_t flags)
```

Enable one or more TIMER interrupts.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
uint32_t	[in]	flags	TIMER interrupt source(s) to enable. Use one or more valid interrupt flags for the TIMER module (TIMER_IF_nnn) OR'ed together.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [TIMER_IntClear\(\)](#) prior to enabling the interrupt.

TIMER_IntGet

```
uint32_t TIMER_IntGet (TIMER_TypeDef * timer)
```

Get pending TIMER interrupt flags.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.

Note

- Event bits are not cleared by this function.

Returns

- TIMER interrupt source(s) pending. Returns one or more valid interrupt flags for the TIMER module (TIMER_IF_nnn) OR'ed together.

TIMER_IntGetEnabled

```
uint32_t TIMER_IntGetEnabled (TIMER_TypeDef * timer)
```

Get enabled and pending TIMER interrupt flags.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by this function.

Returns

- Pending and enabled TIMER interrupt sources. The return value is the bitwise AND combination of
- the OR combination of enabled interrupt sources in `TIMERx_IEN_nnn` register (`TIMERx_IEN_nnn`) and
 - the OR combination of valid interrupt flags of the TIMER module (`TIMERx_IF_nnn`).

TIMER_IntSet

```
void TIMER_IntSet (TIMER_TypeDef * timer, uint32_t flags)
```

Set one or more pending TIMER interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
<code>TIMER_TypeDef *</code>	[in]	timer	Pointer to the TIMER peripheral register block.
<code>uint32_t</code>	[in]	flags	TIMER interrupt source(s) to set to pending. Use one or more valid interrupt flags for the TIMER module (<code>TIMER_IF_nnn</code>) OR'ed together.

TIMER_TopBufSet

```
void TIMER_TopBufSet (TIMER_TypeDef * timer, uint32_t val)
```

Set the top value buffer for the timer.

Parameters

Type	Direction	Argument Name	Description
<code>TIMER_TypeDef *</code>	[in]	timer	Pointer to the TIMER peripheral register block.
<code>uint32_t</code>	[in]	val	Value to set in top value buffer register.

When top value buffer register is updated, value is loaded into top value register at the next wrap around. This feature is useful in order to update top value safely when timer is running.

TIMER_TopGet

```
uint32_t TIMER_TopGet (TIMER_TypeDef * timer)
```

Get the top value setting for the timer.

Parameters

Type	Direction	Argument Name	Description
<code>TIMER_TypeDef *</code>	[in]	timer	Pointer to the TIMER peripheral register block.

Returns

- Current top value.

TIMER_TopSet

```
void TIMER_TopSet (TIMER_TypeDef * timer, uint32_t val)
```

Set the top value for timer.

Parameters

Type	Direction	Argument Name	Description
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
uint32_t	[in]	val	Value to set in top value register.

Macro Definition Documentation

TIMER_INIT_DEFAULT

```
#define TIMER_INIT_DEFAULT
```

Value:

```
0 {
0     true,                /* Enable timer when initialization completes. */ \
0     false,               /* Stop counter during debug halt. */           \
0     timerPrescale1,      /* No prescaling. */                             \
0     timerClkSelHFPerClk, /* Select HFPER / HFPERB clock. */             \
0     false,               /* Not 2x count mode. */                       \
0     false,               /* No ATI. */                                       \
0     false,               /* No RSSCOIST. */                               \
0     timerInputActionNone, /* No action on falling input edge. */         \
0     timerInputActionNone, /* No action on rising input edge. */       \
0     timerModeUp,         /* Up-counting. */                                       \
0     false,               /* Do not clear DMA requests when DMA channel is active. */ \
0     false,               /* Select X2 quadrature decode mode (if used). */       \
0     false,               /* Disable one shot. */                                   \
0     false,               /* Not started/stopped/reloaded by other timers. */   \
0     false,               /* Disable ability to start/stop/reload other timers. */ \
0 }
```

Default configuration for TIMER initialization structure.

TIMER_INITCC_DEFAULT

```
#define TIMER_INITCC_DEFAULT
```

Value:

```
0 {
0     timerEventEveryEdge, /* Event on every capture. */ \
0     timerEdgeRising,    /* Input capture edge on rising edge. */ \
0     0,                  /* Not used by default, select PRS channel 0. */ \
0     timerOutputActionNone, /* No action on underflow. */ \
0     timerOutputActionNone, /* No action on overflow. */ \
0     timerOutputActionNone, /* No action on match. */ \
0     timerCCModeOff,      /* Disable compare/capture channel. */ \
0     false,               /* Disable filter. */ \
0     false,               /* No PRS input. */ \
0     false,               /* Clear output when counter disabled. */ \
0     false,               /* Do not invert output. */ \
0     timerPrsOutputDefault, /* Use default PRS output configuration. */ \
0     timerPrsInputNone    /* No PRS input, so input type is none. */ \
0 }
```

Default configuration for TIMER compare/capture initialization structure.

TIMER_INITDTI_DEFAULT

```
#define TIMER_INITDTI_DEFAULT
```

Value:

```

0  {
0  true,          /* Enable the DTI. */
0  false,        /* CC[0|1|2] outputs are active high. */
0  false,        /* CDTI[0|1|2] outputs are not inverted. */
0  false,        /* No auto restart when debugger exits. */
0  false,        /* No PRS source selected. */
0  0,            /* Not used by default, select PRS channel 0. */
0  timerPrescale1, /* No prescaling. */
0  0,            /* No rise time. */
0  0,            /* No fall time. */
0  TIMER_DTOGEN_DTOGCC0EN | TIMER_DTOGEN_DTOGCCDTI0EN, /* Enable CC0 and CDTI0. */
0  true,         /* Enable core lockup as fault source. */
0  true,         /* Enable debugger as fault source. */
0  false,        /* Disable PRS fault source 0. */
0  0,            /* Not used by default, select PRS channel 0. */
0  false,        /* Disable PRS fault source 1. */
0  0,            /* Not used by default, select PRS channel 0. */
0  timerDtiFaultActionInactive, /* No fault action. */
0  }

```

Default configuration for TIMER DTI initialization structure.

TIMER_Init_TypeDef

TIMER initialization structure.

Public Attributes

	bool	enable	Start counting when initialization completed.
	bool	debugRun	Counter shall keep running during debug halt.
TIMER_Prescale_TypeDef		prescale	Prescaling factor, if HFPER / HFPERB clock used.
TIMER_ClkSel_TypeDef		clkSel	Clock selection.
	bool	count2x	2x Count mode, counter increments/decrements by 2, meant for PWM mode.
	bool	ati	ATI (Always Track Inputs) makes CCPOL always track the polarity of the inputs.
	bool	rssCoist	Reload-Start Sets COIST When enabled, compare output is set to COIST value on a Reload-Start event.
TIMER_InputAction_TypeDef		fallAction	Action on falling input edge.
TIMER_InputAction_TypeDef		riseAction	Action on rising input edge.
TIMER_Mode_TypeDef		mode	Counting mode.
	bool	dmaClrAct	DMA request clear on active.
	bool	quadModeX4	Select X2 or X4 quadrature decode mode (if used).
	bool	oneShot	Determines if only counting up or down once.
	bool	sync	Timer can be start/stop/reload by other timers.
	bool	disSyncOut	Disable ability of timer to start/stop/reload other timers that have their SYNC bit set.

Public Attribute Documentation

enable

```
bool TIMER_Init_TypeDef::enable
```

Start counting when initialization completed.

debugRun

```
bool TIMER_Init_TypeDef::debugRun
```

Counter shall keep running during debug halt.

prescale

```
TIMER_Prescale_TypeDef TIMER_Init_TypeDef::prescale
```

Prescaling factor, if HPPER / HPERB clock used.

clkSel

```
TIMER_ClkSel_TypeDef TIMER_Init_TypeDef::clkSel
```

Clock selection.

count2x

```
bool TIMER_Init_TypeDef::count2x
```

2x Count mode, counter increments/decrements by 2, meant for PWM mode.

ati

```
bool TIMER_Init_TypeDef::ati
```

ATI (Always Track Inputs) makes CCPOL always track the polarity of the inputs.

rssCoist

```
bool TIMER_Init_TypeDef::rssCoist
```

Reload-Start Sets COIST When enabled, compare output is set to COIST value on a Reload-Start event.

fallAction

```
TIMER_InputAction_TypeDef TIMER_Init_TypeDef::fallAction
```

Action on falling input edge.

riseAction

```
TIMER_InputAction_TypeDef TIMER_Init_TypeDef::riseAction
```

Action on rising input edge.

mode

```
TIMER_Mode_TypeDef TIMER_Init_TypeDef::mode
```

Counting mode.

dmaClrAct

```
bool TIMER_Init_TypeDef::dmaClrAct
```

DMA request clear on active.

quadModeX4

```
bool TIMER_Init_TypeDef::quadModeX4
```

Select X2 or X4 quadrature decode mode (if used).

oneShot

```
bool TIMER_Init_TypeDef::oneShot
```

Determines if only counting up or down once.

sync

```
bool TIMER_Init_TypeDef::sync
```

Timer can be start/stop/reload by other timers.

disSyncOut

```
bool TIMER_Init_TypeDef::disSyncOut
```

Disable ability of timer to start/stop/reload other timers that have their SYNC bit set.

TIMER_InitCC_TypeDef

TIMER compare/capture initialization structure.

Public Attributes

<code>TIMER_Event_TypeDef</code>	<code>eventCtrl</code> Input capture event control.
<code>TIMER_Edge_TypeDef</code>	<code>edge</code> Input capture edge select.
<code>TIMER_PRSEL_TypeDef</code>	<code>prsSel</code> Peripheral reflex system trigger selection.
<code>TIMER_OutputAction_TypeDef</code>	<code>cufoa</code> Counter underflow output action.
<code>TIMER_OutputAction_TypeDef</code>	<code>cofoa</code> Counter overflow output action.
<code>TIMER_OutputAction_TypeDef</code>	<code>cmoa</code> Counter match output action.
<code>TIMER_CCMODE_TypeDef</code>	<code>mode</code> Compare/capture channel mode.
<code>bool</code>	<code>filter</code> Enable digital filter.
<code>bool</code>	<code>prsInput</code> Select TIMERNCCx (false) or PRS input (true).
<code>bool</code>	<code>coist</code> Compare output initial state.
<code>bool</code>	<code>outInvert</code> Invert output from compare/capture channel.
<code>TIMER_PrsOutput_t</code>	<code>prsOutput</code> PRS output configuration.
<code>TIMER_PrsInput_TypeDef</code>	<code>prsInputType</code> When PRS input is used this field is used to configure the type of PRS input.

Public Attribute Documentation

eventCtrl

`TIMER_Event_TypeDef TIMER_InitCC_TypeDef::eventCtrl`

Input capture event control.

edge

`TIMER_Edge_TypeDef TIMER_InitCC_TypeDef::edge`

Input capture edge select.

prsSel

```
TIMER_PRSEL_TypeDef TIMER_InitCC_TypeDef::prsSel
```

Peripheral reflex system trigger selection.

Only applicable if `prsInput` is enabled.

cufoa

```
TIMER_OutputAction_TypeDef TIMER_InitCC_TypeDef::cufoa
```

Counter underflow output action.

cofoa

```
TIMER_OutputAction_TypeDef TIMER_InitCC_TypeDef::cofoa
```

Counter overflow output action.

cmoa

```
TIMER_OutputAction_TypeDef TIMER_InitCC_TypeDef::cmoa
```

Counter match output action.

mode

```
TIMER_CCMODE_TypeDef TIMER_InitCC_TypeDef::mode
```

Compare/capture channel mode.

filter

```
bool TIMER_InitCC_TypeDef::filter
```

Enable digital filter.

prsInput

```
bool TIMER_InitCC_TypeDef::prsInput
```

Select TIMERNCCx (false) or PRS input (true).

coist

```
bool TIMER_InitCC_TypeDef::coist
```

Compare output initial state.

Only used in Output Compare and PWM mode. When true, the compare/PWM output is set high when the counter is disabled. When counting resumes, this value will represent the initial value for the compare/PWM output. If the bit is cleared, the output will be cleared when the counter is disabled.

outInvert

```
bool TIMER_InitCC_TypeDef::outInvert
```

Invert output from compare/capture channel.

prsOutput

```
TIMER_PrsOutput_t TIMER_InitCC_TypeDef::prsOutput
```

PRS output configuration.

PRS output from a timer can either be a pulse output or a level output that follows the CC out value.

prsInputType

```
TIMER_PrsInput_TypeDef TIMER_InitCC_TypeDef::prsInputType
```

When PRS input is used this field is used to configure the type of PRS input.

TIMER_InitDTI_TypeDef

TIMER Dead Time Insertion (DTI) initialization structure.

Public Attributes

	bool	enable Enable DTI or leave it disabled until TIMER_EnableDTI() is called.
	bool	activeLowOut DTI Output Polarity.
	bool	invertComplementaryOut DTI Complementary Output Invert.
	bool	autoRestart Enable Automatic Start-up functionality (when debugger exits).
	bool	enablePrsSource Enable/disable PRS as DTI input.
TIMER_PRSEL_TypeDef	prsSel	Select which PRS channel as DTI input.
TIMER_Prescale_TypeDef	prescale	DTI prescaling factor, if HPER / HPERB clock used.
	unsigned int	riseTime DTI Rise Time.
	unsigned int	fallTime DTI Fall Time.
	uint32_t	outputsEnableMask DTI outputs enable bit mask, consisting of one bit per DTI output signal, i.e., CC0, CC1, CC2, CDTI0, CDTI1, and CDTI2.
	bool	enableFaultSourceCoreLockup Enable core lockup as a fault source.
	bool	enableFaultSourceDebugger Enable debugger as a fault source.
	bool	enableFaultSourcePrsSel0 Enable PRS fault source 0 (<code>faultSourcePrsSel0</code>).
TIMER_PRSEL_TypeDef	faultSourcePrsSel0	Select which PRS signal to be PRS fault source 0.
	bool	enableFaultSourcePrsSel1 Enable PRS fault source 1 (<code>faultSourcePrsSel1</code>).
TIMER_PRSEL_TypeDef	faultSourcePrsSel1	Select which PRS signal to be PRS fault source 1.
TIMER_DtiFaultAction_TypeDef	faultAction	Fault Action.

Public Attribute Documentation

enable

```
bool TIMER_InitDTI_TypeDef::enable
```

Enable DTI or leave it disabled until [TIMER_EnableDTI\(\)](#) is called.

activeLowOut

```
bool TIMER_InitDTI_TypeDef::activeLowOut
```

DTI Output Polarity.

invertComplementaryOut

```
bool TIMER_InitDTI_TypeDef::invertComplementaryOut
```

DTI Complementary Output Invert.

autoRestart

```
bool TIMER_InitDTI_TypeDef::autoRestart
```

Enable Automatic Start-up functionality (when debugger exits).

enablePrsSource

```
bool TIMER_InitDTI_TypeDef::enablePrsSource
```

Enable/disable PRS as DTI input.

prsSel

```
TIMER_PRSEL_TypeDef TIMER_InitDTI_TypeDef::prsSel
```

Select which PRS channel as DTI input.

Only valid if `enablePrsSource` is enabled.

prescale

```
TIMER_Prescale_TypeDef TIMER_InitDTI_TypeDef::prescale
```

DTI prescaling factor, if HFPER / HFPERB clock used.

riseTime

```
unsigned int TIMER_InitDTI_TypeDef::riseTime
```

DTI Rise Time.

fallTime

```
unsigned int TIMER_InitDTI_TypeDef::fallTime
```

DTI Fall Time.

outputsEnableMask

```
uint32_t TIMER_InitDTI_TypeDef::outputsEnableMask
```

DTI outputs enable bit mask, consisting of one bit per DTI output signal, i.e., CC0, CC1, CC2, CDTI0, CDTI1, and CDTI2.

This value should consist of one or more `TIMER_DTOGEN_DTOGnnnEN` flags (defined in `<part_name>_timer.h`) OR'ed together.

enableFaultSourceCoreLockup

```
bool TIMER_InitDTI_TypeDef::enableFaultSourceCoreLockup
```

Enable core lockup as a fault source.

enableFaultSourceDebugger

```
bool TIMER_InitDTI_TypeDef::enableFaultSourceDebugger
```

Enable debugger as a fault source.

enableFaultSourcePrsSel0

```
bool TIMER_InitDTI_TypeDef::enableFaultSourcePrsSel0
```

Enable PRS fault source 0 (`faultSourcePrsSel0`).

faultSourcePrsSel0

```
TIMER_PRSEL_TypeDef TIMER_InitDTI_TypeDef::faultSourcePrsSel0
```

Select which PRS signal to be PRS fault source 0.

enableFaultSourcePrsSel1

```
bool TIMER_InitDTI_TypeDef::enableFaultSourcePrsSel1
```

Enable PRS fault source 1 (`faultSourcePrsSel1`).

faultSourcePrsSel1

```
TIMER_PRSSEL_TypeDef TIMER_InitDTI_TypeDef::faultSourcePrsSel1
```

Select which PRS signal to be PRS fault source 1.

faultAction

```
TIMER_DtiFaultAction_TypeDef TIMER_InitDTI_TypeDef::faultAction
```

Fault Action.

USART - Synchronous/Asynchronous Serial

USART - Synchronous/Asynchronous Serial

Universal Synchronous/Asynchronous Receiver/Transmitter Peripheral API.

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is a very flexible serial I/O module. It supports full duplex asynchronous UART communication as well as RS-485, SPI, MicroWire, and 3-wire. It can also interface with ISO7816 Smart-Cards, and IrDA devices.

The USART has a wide selection of operating modes, frame formats, and baud rates. All features are supported through the API of this module.

Triple buffering and DMA support makes high data-rates possible with minimal CPU intervention. It is possible to transmit and receive large frames while the MCU remains in EM1 Sleep.

This module does not support DMA configuration. The UARTDRV and SPIDRV drivers provide full support for DMA and more.

The following steps are necessary for basic operation:

Clock enable:

```
#if !defined(_SILICON_LABS_32B_SERIES_2)
/* USART is a HPERCLK peripheral. Enable HPERCLK domain and USART0.
 * We also need to enable the clock for GPIO to configure pins. */
CMU_ClockEnable(cmuClock_HPPER, true);
CMU_ClockEnable(cmuClock_USART0, true);
CMU_ClockEnable(cmuClock_GPIO, true);
#endif
```

To initialize the USART for asynchronous operation (e.g., UART):

```
/* Initialize with default settings and then update fields according to application requirements. */
USART_InitAsync_TypeDef initAsync = USART_INITASYNC_DEFAULT;
initAsync.baudrate = 38400;
USART_InitAsync(USART0, &initAsync);
```

To initialize the USART for synchronous operation (e.g., SPI):

```
/* Initialize with default settings and then update fields according to application requirements. */
USART_InitSync_TypeDef initSync = USART_INITSYNC_DEFAULT;
/* Operate as SPI master */
initSync.master = true;
/* Clock idle low, sample on falling edge. */
initSync.clockMode = usartClockMode1;
USART_InitSync(USART0, &initSync);
```

After pins are assigned for the application/board, enable pins at the desired location. Available locations can be obtained from the Pin Definitions section in the data sheet. Note

- UARTDRV supports all types of UART flow control. Software assisted hardware flow control is available for parts without true UART hardware flow control.

Modules

[USART_InitAsync_TypeDef](#)
[USART_PrTrgInit_TypeDef](#)
[USART_InitSync_TypeDef](#)
[USART_InitIrDA_TypeDef](#)
[USART_InitI2s_TypeDef](#)

Enumerations

```
enum    USART_Databits_TypeDef {
        usartDatabits4 = USART_FRAME_DATABITS_FOUR
        usartDatabits5 = USART_FRAME_DATABITS_FIVE
        usartDatabits6 = USART_FRAME_DATABITS_SIX
        usartDatabits7 = USART_FRAME_DATABITS_SEVEN
        usartDatabits8 = USART_FRAME_DATABITS_EIGHT
        usartDatabits9 = USART_FRAME_DATABITS_NINE
        usartDatabits10 = USART_FRAME_DATABITS_TEN
        usartDatabits11 = USART_FRAME_DATABITS_ELEVEN
        usartDatabits12 = USART_FRAME_DATABITS_TWELVE
        usartDatabits13 = USART_FRAME_DATABITS_THIRTEEN
        usartDatabits14 = USART_FRAME_DATABITS_FOURTEEN
        usartDatabits15 = USART_FRAME_DATABITS_FIFTEEN
        usartDatabits16 = USART_FRAME_DATABITS_SIXTEEN
    }
    Databit selection.

enum    USART_Enable_TypeDef {
        usartDisable = 0x0
        usartEnableRx = USART_CMD_RXEN
        usartEnableTx = USART_CMD_TXEN
        usartEnable = (USART_CMD_RXEN | USART_CMD_TXEN)
    }
    Enable selection.

enum    USART_OVS_TypeDef {
        usartOVS16 = USART_CTRL_OVS_X16
        usartOVS8 = USART_CTRL_OVS_X8
        usartOVS6 = USART_CTRL_OVS_X6
        usartOVS4 = USART_CTRL_OVS_X4
    }
    Oversampling selection, used for asynchronous operation.

enum    USART_Parity_TypeDef {
        usartNoParity = USART_FRAME_PARITY_NONE
        usartEvenParity = USART_FRAME_PARITY_EVEN
        usartOddParity = USART_FRAME_PARITY_ODD
    }
    Parity selection, mainly used for asynchronous operation.

enum    USART_Stopbits_TypeDef {
        usartStopbits0p5 = USART_FRAME_STOPBITS_HALF
        usartStopbits1 = USART_FRAME_STOPBITS_ONE
        usartStopbits1p5 = USART_FRAME_STOPBITS_ONEANDAHALF
        usartStopbits2 = USART_FRAME_STOPBITS_TWO
    }
    Stop bits selection, used for asynchronous operation.

enum    USART_HwFlowControl_TypeDef {
        usartHwFlowControlNone = 0
        usartHwFlowControlCts
        usartHwFlowControlRts
        usartHwFlowControlCtsAndRts
    }
    Hardware Flow Control Selection.

enum    USART_ClockMode_TypeDef {
```

```

usartClockMode0 =
USART_CTRL_CLKPOL_IDLELOW |
USART_CTRL_CLKPHA_SAMPLELEADING
usartClockMode1 =
USART_CTRL_CLKPOL_IDLELOW |
USART_CTRL_CLKPHA_SAMPLETRAILING
usartClockMode2 =
USART_CTRL_CLKPOL_IDLEHIGH |
USART_CTRL_CLKPHA_SAMPLELEADING
usartClockMode3 =
USART_CTRL_CLKPOL_IDLEHIGH |
USART_CTRL_CLKPHA_SAMPLETRAILING
}

```

Clock polarity/phase mode.

```

enum    USART_IrDAPw_TypeDef {
    usartIrDAPwONE = USART_IRCTRL_IRPW_ONE
    usartIrDAPwTWO = USART_IRCTRL_IRPW_TWO
    usartIrDAPwTHREE = USART_IRCTRL_IRPW_THREE
    usartIrDAPwFOUR = USART_IRCTRL_IRPW_FOUR
}
Pulse width selection for IrDA mode.

enum    USART_I2sFormat_TypeDef {
    usartI2sFormatW32D32 = USART_I2SCTRL_FORMAT_W32D32
    usartI2sFormatW32D24M = USART_I2SCTRL_FORMAT_W32D24M
    usartI2sFormatW32D24 = USART_I2SCTRL_FORMAT_W32D24
    usartI2sFormatW32D16 = USART_I2SCTRL_FORMAT_W32D16
    usartI2sFormatW32D8 = USART_I2SCTRL_FORMAT_W32D8
    usartI2sFormatW16D16 = USART_I2SCTRL_FORMAT_W16D16
    usartI2sFormatW16D8 = USART_I2SCTRL_FORMAT_W16D8
    usartI2sFormatW8D8 = USART_I2SCTRL_FORMAT_W8D8
}
I2S format selection.

enum    USART_I2sJustify_TypeDef {
    usartI2sJustifyLeft = USART_I2SCTRL_JUSTIFY_LEFT
    usartI2sJustifyRight = USART_I2SCTRL_JUSTIFY_RIGHT
}
I2S frame data justify.

```

Typedefs

```

typedef uint8_t    USART_PRS_Channel_t
PRs Channel type.

```

Functions

```

void    USART_BaudrateAsyncSet(USART_TypeDef *usart, uint32_t refFreq, uint32_t baudrate,
USART_OVS_TypeDef ovs)
Configure USART/UART operating in asynchronous mode to use a given baudrate (or as close as possible to a
specified baudrate).

uint32_t    USART_BaudrateCalc(uint32_t refFreq, uint32_t clkdiv, bool syncmode, USART_OVS_TypeDef
ovs)
Calculate baudrate for USART/UART given reference frequency, clock division, and oversampling rate (if async
mode).

uint32_t    USART_BaudrateGet(USART_TypeDef *usart)
Get the current baudrate for USART/UART.

void    USART_BaudrateSyncSet(USART_TypeDef *usart, uint32_t refFreq, uint32_t baudrate)
Configure the USART operating in synchronous mode to use a given baudrate (or as close as possible to a
specified baudrate).

void    USART_Enable(USART_TypeDef *usart, USART_Enable_TypeDef enable)
Enable/disable USART/UART receiver and/or transmitter.

void    USART_InitAsync(USART_TypeDef *usart, const USART_InitAsync_TypeDef *init)
Initialize USART/UART for normal asynchronous mode.

```

void	USART_InitSync (USART_TypeDef *usart, const USART_InitSync_TypeDef *init) Initialize USART for synchronous mode.
void	USARTn_InitIrDA (USART_TypeDef *usart, const USART_InitIrDA_TypeDef *init) Initialize USART for asynchronous IrDA mode.
void	USART_InitI2S (USART_TypeDef *usart, USART_InitI2S_TypeDef *init) Initialize USART for I2S mode.
void	USART_InitPrsTrigger (USART_TypeDef *usart, const USART_PrsTriggerInit_TypeDef *init) Initialize the automatic transmissions using PRS channel as a trigger.
void	USART_Reset (USART_TypeDef *usart) Reset USART/UART to the same state that it was in after a hardware reset.
uint8_t	USART_Rx (USART_TypeDef *usart) Receive one 4-8 bit frame, (or part of 10-16 bit frame).
uint16_t	USART_RxDouble (USART_TypeDef *usart) Receive two 4-8 bit frames or one 10-16 bit frame.
uint32_t	USART_RxDoubleExt (USART_TypeDef *usart) Receive two 4-9 bit frames, or one 10-16 bit frame with extended information.
uint16_t	USART_RxExt (USART_TypeDef *usart) Receive one 4-9 bit frame (or part of 10-16 bit frame) with extended information.
uint8_t	USART_SpiTransfer (USART_TypeDef *usart, uint8_t data) Perform one 8 bit frame SPI transfer.
void	USART_Tx (USART_TypeDef *usart, uint8_t data) Transmit one 4-9 bit frame.
void	USART_TxDouble (USART_TypeDef *usart, uint16_t data) Transmit two 4-9 bit frames or one 10-16 bit frame.
void	USART_TxDoubleExt (USART_TypeDef *usart, uint32_t data) Transmit two 4-9 bit frames or one 10-16 bit frame with extended control.
void	USART_TxExt (USART_TypeDef *usart, uint16_t data) Transmit one 4-9 bit frame with extended control.
void	USART_IntClear (USART_TypeDef *usart, uint32_t flags) Clear one or more pending USART interrupts.
void	USART_IntDisable (USART_TypeDef *usart, uint32_t flags) Disable one or more USART interrupts.
void	USART_IntEnable (USART_TypeDef *usart, uint32_t flags) Enable one or more USART interrupts.
uint32_t	USART_IntGet (USART_TypeDef *usart) Get pending USART interrupt flags.
uint32_t	USART_IntGetEnabled (USART_TypeDef *usart) Get enabled and pending USART interrupt flags.
void	USART_IntSet (USART_TypeDef *usart, uint32_t flags) Set one or more pending USART interrupts from SW.
uint32_t	USART_StatusGet (USART_TypeDef *usart) Get USART STATUS register.
uint8_t	USART_RxDataGet (USART_TypeDef *usart) Receive one 4-8 bit frame, (or part of 10-16 bit frame).

uint16_t	USART_RxDoubleGet (USART_TypeDef *usart) Receive two 4-8 bit frames, or one 10-16 bit frame.
uint32_t	USART_RxDoubleXGet (USART_TypeDef *usart) Receive two 4-9 bit frames, or one 10-16 bit frame with extended information.
uint16_t	USART_RxDataXGet (USART_TypeDef *usart) Receive one 4-9 bit frame, (or part of 10-16 bit frame) with extended information.

Macros

#define	USART_INITASYNC_DEFAULT undefined Default configuration for USART asynchronous initialization structure.
#define	USART_INITPRSTRIGGER_DEFAULT undefined Default configuration for USART PRS triggering structure.
#define	USART_INITSYNC_DEFAULT undefined Default configuration for USART sync initialization structure.
#define	USART_INITIRDA_DEFAULT undefined Default configuration for IrDA mode initialization structure.
#define	USART_INITI2S_DEFAULT undefined Default configuration for I2S mode initialization structure.

Enumeration Documentation

USART_Databits_TypeDef

USART_Databits_TypeDef

Databit selection.

Enumerator	
usartDatabits4	4 data bits (not available for UART).
usartDatabits5	5 data bits (not available for UART).
usartDatabits6	6 data bits (not available for UART).
usartDatabits7	7 data bits (not available for UART).
usartDatabits8	8 data bits.
usartDatabits9	9 data bits.
usartDatabits10	10 data bits (not available for UART).
usartDatabits11	11 data bits (not available for UART).
usartDatabits12	12 data bits (not available for UART).
usartDatabits13	13 data bits (not available for UART).
usartDatabits14	14 data bits (not available for UART).
usartDatabits15	15 data bits (not available for UART).
usartDatabits16	16 data bits (not available for UART).

USART_Enable_TypeDef

USART_Enable_TypeDef

Enable selection.

Enumerator

usartDisable	Disable both receiver and transmitter.
usartEnableRx	Enable receiver only, transmitter disabled.
usartEnableTx	Enable transmitter only, receiver disabled.
usartEnable	Enable both receiver and transmitter.

USART_OVS_TypeDef

USART_OVS_TypeDef

Oversampling selection, used for asynchronous operation.

Enumerator

usartOVS16	16x oversampling (normal).
usartOVS8	8x oversampling.
usartOVS6	6x oversampling.
usartOVS4	4x oversampling.

USART_Parity_TypeDef

USART_Parity_TypeDef

Parity selection, mainly used for asynchronous operation.

Enumerator

usartNoParity	No parity.
usartEvenParity	Even parity.
usartOddParity	Odd parity.

USART_Stopbits_TypeDef

USART_Stopbits_TypeDef

Stop bits selection, used for asynchronous operation.

Enumerator

usartStopbits0p5	0.5 stop bits.
usartStopbits1	1 stop bits.
usartStopbits1p5	1.5 stop bits.
usartStopbits2	2 stop bits.

USART_HwFlowControl_TypeDef

USART_HwFlowControl_TypeDef

Hardware Flow Control Selection.

Enumerator

usartHwFlowControlNone	No hardware flow control.
------------------------	---------------------------

usartHwFlowControlCts	CTS signal is enabled for TX flow control.
usartHwFlowControlRts	RTS signal is enabled for RX flow control.
usartHwFlowControlCtsAndRts	CTS and RTS signals are enabled for TX and RX flow control.

USART_ClockMode_TypeDef

USART_ClockMode_TypeDef

Clock polarity/phase mode.

Enumerator	
usartClockMode0	Clock idle low, sample on rising edge.
usartClockMode1	Clock idle low, sample on falling edge.
usartClockMode2	Clock idle high, sample on falling edge.
usartClockMode3	Clock idle high, sample on rising edge.

USART_IrDAPw_Typedef

USART_IrDAPw_Typedef

Pulse width selection for IrDA mode.

Enumerator	
usartIrDAPwONE	IrDA pulse width is 1/16 for OVS=0 and 1/8 for OVS=1.
usartIrDAPwTWO	IrDA pulse width is 2/16 for OVS=0 and 2/8 for OVS=1.
usartIrDAPwTHREE	IrDA pulse width is 3/16 for OVS=0 and 3/8 for OVS=1.
usartIrDAPwFOUR	IrDA pulse width is 4/16 for OVS=0 and 4/8 for OVS=1.

USART_I2sFormat_TypeDef

USART_I2sFormat_TypeDef

I2S format selection.

Enumerator	
usartI2sFormatW32D32	32-bit word, 32-bit data.
usartI2sFormatW32D24M	32-bit word, 32-bit data with 8 lsb masked.
usartI2sFormatW32D24	32-bit word, 24-bit data.
usartI2sFormatW32D16	32-bit word, 16-bit data.
usartI2sFormatW32D8	32-bit word, 8-bit data.
usartI2sFormatW16D16	16-bit word, 16-bit data.
usartI2sFormatW16D8	16-bit word, 8-bit data.
usartI2sFormatW8D8	8-bit word, 8-bit data.

USART_I2sJustify_TypeDef

USART_I2sJustify_TypeDef

I2S frame data justify.

Enumerator	
usartI2sJustifyLeft	Data is left-justified within the frame.
usartI2sJustifyRight	Data is right-justified within the frame.

Typedef Documentation

USART_PRS_Channel_t

```
typedef uint8_t USART_PRS_Channel_t
```

PRS Channel type.

Function Documentation

USART_BaudrateAsyncSet

```
void USART_BaudrateAsyncSet (USART_TypeDef * usart, uint32_t refFreq, uint32_t baudrate, USART_OVS_TypeDef ovs)
```

Configure USART/UART operating in asynchronous mode to use a given baudrate (or as close as possible to a specified baudrate).

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.
uint32_t	[in]	refFreq	USART/UART reference clock frequency in Hz. If set to 0, the currently configured reference clock is assumed.
uint32_t	[in]	baudrate	Baudrate to try to achieve for USART/UART.
USART_OVS_TypeDef	[in]	ovs	Oversampling to be used. Normal is 16x oversampling but lower oversampling may be used to achieve higher rates or better baudrate accuracy in some cases. Notice that lower oversampling frequency makes the channel more vulnerable to bit faults during reception due to clock inaccuracies compared to the link partner.

USART_BaudrateCalc

```
uint32_t USART_BaudrateCalc (uint32_t refFreq, uint32_t clkdiv, bool syncmode, USART_OVS_TypeDef ovs)
```

Calculate baudrate for USART/UART given reference frequency, clock division, and oversampling rate (if async mode).

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	refFreq	USART/UART HF peripheral frequency used.
uint32_t	[in]	clkdiv	A clock division factor to be used.

Type	Direction	Argument Name	Description
bool	[in]	syncmode	• True - synchronous mode operation. • False - asynchronous mode operation.
USART_OVS_TypeDef	[in]	ovs	Oversampling used if in asynchronous mode. Not used if syncmode is true.

This function returns the baudrate that a USART/UART module will use if configured with the given frequency, clock divisor, and mode. Notice that this function will not use the hardware configuration. It can be used to determine if a given configuration is sufficiently accurate for the application.

Returns

- Baudrate with given settings.

USART_BaudrateGet

```
uint32_t USART_BaudrateGet (USART_TypeDef * usart)
```

Get the current baudrate for USART/UART.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.

This function returns the actual baudrate (not considering oscillator inaccuracies) used by a USART/UART peripheral.

Returns

- The current baudrate.

USART_BaudrateSyncSet

```
void USART_BaudrateSyncSet (USART_TypeDef * usart, uint32_t refFreq, uint32_t baudrate)
```

Configure the USART operating in synchronous mode to use a given baudrate (or as close as possible to a specified baudrate).

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART peripheral register block. (Cannot be used on UART modules.)
uint32_t	[in]	refFreq	A USART reference clock frequency in Hz that will be used. If set to 0, the currently-configured reference clock is assumed.
uint32_t	[in]	baudrate	Baudrate to try to achieve for USART.

The configuration will be set to use a baudrate $\leq$ the specified baudrate to ensure that the baudrate does not exceed the specified value.

The fractional clock division is suppressed, although the hardware design allows it. It could cause half clock cycles to exceed a specified limit and thus potentially violate specifications for the slave device. In some

special situations, a fractional clock division may be useful even in synchronous mode, but in those cases it must be directly adjusted, possibly assisted by `USART_BaudrateCalc()`:

Warnings

- The consequence of the aforementioned suppression of the fractional part of the clock divider is that some frequencies won't be achievable. The divider will only be able to be an integer value so the reference clock will only be dividable by N (where N is a positive integer).

USART_Enable

```
void USART_Enable (USART_TypeDef * usart, USART_Enable_TypeDef enable)
```

Enable/disable USART/UART receiver and/or transmitter.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.
USART_Enable_TypeDef	[in]	enable	Select the status for the receiver/transmitter.

Notice that this function does not do any configuration. Enabling should normally be done after initialization (if not enabled as part of initialization).

USART_InitAsync

```
void USART_InitAsync (USART_TypeDef * usart, const USART_InitAsync_TypeDef * init)
```

Initialize USART/UART for normal asynchronous mode.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.
const USART_InitAsync_TypeDef *	[in]	init	A pointer to the initialization structure used to configure the basic async setup.

This function will configure basic settings to operate in normal asynchronous mode.

A special control setup not covered by this function must be done after using this function by direct modification of the CTRL register.

Notice that pins used by the USART/UART module must be properly configured by the user explicitly for the USART/UART to work as intended. (When configuring pins, remember to consider the sequence of configuration to avoid unintended pulses/glitches on output pins.)

USART_InitSync

```
void USART_InitSync (USART_TypeDef * usart, const USART_InitSync_TypeDef * init)
```

Initialize USART for synchronous mode.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART peripheral register block. (UART does not support this mode.)
const USART_InitSync_TypeDef *	[in]	init	A pointer to the initialization structure used to configure basic async setup.

This function will configure basic settings to operate in synchronous mode.

A special control setup not covered by this function must be done after using this function by direct modification of the CTRL register.

Notice that pins used by the USART module must be properly configured by the user explicitly for the USART to work as intended. (When configuring pins remember to consider the sequence of configuration to avoid unintended pulses/glitches on output pins.)

USARTn_InitIrDA

```
void USARTn_InitIrDA (USART_TypeDef * usart, const USART_InitIrDA_TypeDef * init)
```

Initialize USART for asynchronous IrDA mode.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART peripheral register block.
const USART_InitIrDA_TypeDef *	[in]	init	A pointer to the initialization structure used to configure async IrDA setup.

This function will configure basic settings to operate in asynchronous IrDA mode.

A special control setup not covered by this function must be done after using this function by direct modification of the CTRL and IRCTRL registers.

Notice that pins used by the USART/UART module must be properly configured by the user explicitly for the USART/UART to work as intended. (When configuring pins, remember to consider the sequence of configuration to avoid unintended pulses/glitches on output pins.)

Note

- Not all USART instances support IrDA. See the data sheet for your device.

USART_InitI2s

```
void USART_InitI2s (USART_TypeDef * usart, USART_InitI2s_TypeDef * init)
```

Initialize USART for I2S mode.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART peripheral register block. (UART does not support this mode.)
USART_InitI2s_TypeDef *	[in]	init	A pointer to the initialization structure used to configure the basic I2S setup.

This function will configure basic settings to operate in I2S mode.

A special control setup not covered by this function must be done after using this function by direct modification of the CTRL and I2SCTRL registers.

Notice that pins used by the USART module must be properly configured by the user explicitly for the USART to work as intended. (When configuring pins, remember to consider the sequence of configuration to avoid unintended pulses/glitches on output pins.)

Note

- This function does not apply to all USART's. See the chip Reference Manual.

USART_InitPrsTrigger

```
void USART_InitPrsTrigger (USART_TypeDef * usart, const USART_PrsTriggerInit_TypeDef * init)
```

Initialize the automatic transmissions using PRS channel as a trigger.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to USART to configure.
const USART_PrsTriggerInit_TypeDef *	[in]	init	A pointer to the initialization structure.

Note

- Initialize USART with USART_Init() before setting up the PRS configuration.

USART_Reset

```
void USART_Reset (USART_TypeDef * usart)
```

Reset USART/UART to the same state that it was in after a hardware reset.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to USART/UART peripheral register block.

USART_Rx

```
uint8_t USART_Rx (USART_TypeDef * usart)
```

Receive one 4-8 bit frame, (or part of 10-16 bit frame).

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.

This function is normally used to receive one frame when operating with frame length 4-8 bits. See [USART_RxExt\(\)](#) for reception of 9 bit frames.

Notice that possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.

Note

- This function will stall if the buffer is empty until data is received. Alternatively, the user can explicitly check whether data is available. If data is available, call [USART_RxDataGet\(\)](#) to read the RXDATA register directly.

Returns

- Data received.

USART_RxDouble

```
uint16_t USART_RxDouble (USART_TypeDef * usart)
```

Receive two 4-8 bit frames or one 10-16 bit frame.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.

This function is normally used to receive one frame when operating with frame length 10-16 bits. See [USART_RxDoubleExt\(\)](#) for reception of two 9 bit frames.

Notice that possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.

Note

- This function will stall if the buffer is empty until data is received. Alternatively, the user can explicitly check whether data is available. If data is available, call [USART_RxDoubleGet\(\)](#) to read the RXDOUBLE register directly.

Returns

- Data received.

USART_RxDoubleExt

```
uint32_t USART_RxDoubleExt (USART_TypeDef * usart)
```

Receive two 4-9 bit frames, or one 10-16 bit frame with extended information.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.

This function is normally used to receive one frame when operating with frame length 10-16 bits and additional RX status information is required.

Notice that possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.

Note

- This function will stall if buffer is empty until data is received. Alternatively, the user can explicitly check whether data is available. If data is available, call [USART_RxDoubleXGet\(\)](#) to read the RXDOUBLEX register directly.

Returns

- Data received.

USART_RxExt

```
uint16_t USART_RxExt (USART_TypeDef * usart)
```

Receive one 4-9 bit frame (or part of 10-16 bit frame) with extended information.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.

This function is normally used to receive one frame when operating with frame length 4-9 bits and additional RX status information is required.

Notice that possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.

Note

- This function will stall if the buffer is empty until data is received. Alternatively, the user can explicitly check whether data is available. If data is available, call [USART_RxDataXGet\(\)](#) to read the RXDATA register directly.

Returns

- Data received.

USART_SpiTransfer

```
uint8_t USART_SpiTransfer (USART_TypeDef * usart, uint8_t data)
```

Perform one 8 bit frame SPI transfer.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART peripheral register block.
uint8_t	[in]	data	Data to transmit.

Note

- This function will stall if the transmit buffer is full. When a transmit buffer becomes available, data is written and the function will wait until data is fully transmitted. The SPI return value is then read out and returned.

Returns

- Data received.

USART_Tx

```
void USART_Tx (USART_TypeDef * usart, uint8_t data)
```

Transmit one 4-9 bit frame.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.
uint8_t	[in]	data	Data to transmit. See details above for more information.

Depending on the frame length configuration, 4-8 (least significant) bits from `data` are transmitted. If the frame length is 9, 8 bits are transmitted from `data` and one bit as specified by CTRL register, BIT8DV field. See [USART_TxExt\(\)](#) for transmitting 9 bit frame with full control of all 9 bits.

Notice that possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.

Note

- This function will stall if the buffer is full until the buffer becomes available.

USART_TxDouble

```
void USART_TxDouble (USART_TypeDef * usart, uint16_t data)
```

Transmit two 4-9 bit frames or one 10-16 bit frame.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.
uint16_t	[in]	data	Data to transmit, the least significant byte holds the frame transmitted first. See details above for more info.

Depending on the frame length configuration, 4-8 (least significant) bits from each byte in `data` are transmitted. If frame length is 9, 8 bits are transmitted from each byte in `data` adding one bit as specified by the CTRL register, BIT8DV field, to each byte. See [USART_TxDoubleExt\(\)](#) for transmitting two 9 bit frames with full control of all 9 bits.

If the frame length is 10-16, 10-16 (least significant) bits from `data` are transmitted.

Notice that possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.

Note

- This function will stall if the buffer is full until the buffer becomes available.

USART_TxDoubleExt

```
void USART_TxDoubleExt (USART_TypeDef * usart, uint32_t data)
```

Transmit two 4-9 bit frames or one 10-16 bit frame with extended control.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.

Type	Direction	Argument Name	Description
uint32_t	[in]	data	Data to transmit with extended control. Contains two 16 bit words concatenated. Least significant word holds the frame transmitted first. If the frame length is 4-9, two frames with 4-9 least significant bits from each 16 bit word are transmitted.

Notice that possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.

Note

- This function will stall if the buffer is full until the buffer becomes available.
- If the frame length is 10-16 bits, 8 data bits are taken from the least significant 16 bit word and the remaining bits from the other 16 bit word.
- Additional control bits are available as documented in the reference manual (set to 0 if not used). For 10-16 bit frame length, these control bits are taken from the most significant 16 bit word.

USART_TxExt

```
void USART_TxExt (USART_TypeDef * usart, uint16_t data)
```

Transmit one 4-9 bit frame with extended control.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	A pointer to the USART/UART peripheral register block.
uint16_t	[in]	data	Data to transmit with extended control. Least significant bit contains frame bits. Additional control bits are available as documented in the reference manual (set to 0 if not used).

Notice that possible parity/stop bits in asynchronous mode are not considered part of a specified frame bit length.

Note

- This function will stall if the buffer is full until the buffer becomes available.

USART_IntClear

```
void USART_IntClear (USART_TypeDef * usart, uint32_t flags)
```

Clear one or more pending USART interrupts.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to the USART/UART peripheral register block.
uint32_t	[in]	flags	Pending USART/UART interrupt source(s) to clear. Use one or more valid interrupt flags for the USART module (USART_IF_nnn) OR'ed together.

USART_IntDisable

```
void USART_IntDisable (USART_TypeDef * usart, uint32_t flags)
```

Disable one or more USART interrupts.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to the USART/UART peripheral register block.
uint32_t	[in]	flags	USART/UART interrupt source(s) to disable. Use one or more valid interrupt flags for the USART module (USART_IF_nnn) OR'ed together.

USART_IntEnable

```
void USART_IntEnable (USART_TypeDef * usart, uint32_t flags)
```

Enable one or more USART interrupts.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to the USART/UART peripheral register block.
uint32_t	[in]	flags	USART/UART interrupt source(s) to enable. Use one or more valid interrupt flags for the USART module (USART_IF_nnn) OR'ed together.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using [USART_IntClear\(\)](#) prior to enabling the interrupt.

USART_IntGet

```
uint32_t USART_IntGet (USART_TypeDef * usart)
```

Get pending USART interrupt flags.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to the USART/UART peripheral register block.

Note

- The event bits are not cleared by the use of this function.

Returns

- USART/UART interrupt source(s) pending. Returns one or more valid interrupt flags for the USART module (USART_IF_nnn) OR'ed together.

USART_IntGetEnabled

```
uint32_t USART_IntGetEnabled (USART_TypeDef * usart)
```

Get enabled and pending USART interrupt flags.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to the USART/UART peripheral register block.

Useful for handling more interrupt sources in the same interrupt handler.

Note

- Interrupt flags are not cleared by the use of this function.

Returns

- Pending and enabled USART interrupt sources. The return value is the bitwise AND combination of
 - the OR combination of enabled interrupt sources in USARTx_IEN_nnn register (USARTx_IEN_nnn) and
 - the OR combination of valid interrupt flags of the USART module (USARTx_IF_nnn).

USART_IntSet

```
void USART_IntSet (USART_TypeDef * usart, uint32_t flags)
```

Set one or more pending USART interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to the USART/UART peripheral register block.
uint32_t	[in]	flags	USART/UART interrupt source(s) to set to pending. Use one or more valid interrupt flags for the USART module (USART_IF_nnn) OR'ed together.

USART_StatusGet

```
uint32_t USART_StatusGet (USART_TypeDef * usart)
```

Get USART STATUS register.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to the USART/UART peripheral register block.

Returns

- STATUS register value.

USART_RxDataGet

```
uint8_t USART_RxDataGet (USART_TypeDef * usart)
```

Receive one 4-8 bit frame, (or part of 10-16 bit frame).

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to USART/UART peripheral register block.

This function is used to quickly receive one 4-8 bits frame by reading the RXDATA register directly, without checking the STATUS register for the RXDATAV flag. This can be useful from the RXDATAV interrupt handler, i.e., waiting is superfluous, in order to quickly read the received data. Please refer to [USART_RxDataXGet\(\)](#) for reception of 9 bit frames.

Note

- Because this function does not check whether the RXDATA register actually holds valid data, it should only be used in situations when it is certain that there is valid data, ensured by some external program routine, e.g., when handling an RXDATAV interrupt. The [USART_Rx\(\)](#) is normally a better choice if the validity of the RXDATA register is not certain.
- Notice that possible parity/stop bits in asynchronous mode are not considered part of specified frame bit length.

Returns

- Data received.

USART_RxDoubleGet

```
uint16_t USART_RxDoubleGet (USART_TypeDef * usart)
```

Receive two 4-8 bit frames, or one 10-16 bit frame.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to USART/UART peripheral register block.

This function is used to quickly receive one 10-16 bits frame or two 4-8 bit frames by reading the RXDOUBLE register directly, without checking the STATUS register for the RXDATAV flag. This can be useful from the RXDATAV interrupt handler, i.e., waiting is superfluous, in order to quickly read the received data. This function is normally used to receive one frame when operating with frame length 10-16 bits. Please refer to [USART_RxDoubleXGet\(\)](#) for reception of two 9 bit frames.

Note

- Because this function does not check whether the RXDOUBLE register actually holds valid data, it should only be used in situations when it is certain that there is valid data, ensured by some external program routine, e.g., when handling an RXDATAV interrupt. The [USART_RxDouble\(\)](#) is normally a better choice if the validity of the RXDOUBLE register is not certain.
- Notice that possible parity/stop bits in asynchronous mode are not considered part of specified frame bit length.

Returns

- Data received.

USART_RxDoubleXGet

```
uint32_t USART_RxDoubleXGet (USART_TypeDef * usart)
```

Receive two 4-9 bit frames, or one 10-16 bit frame with extended information.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to USART/UART peripheral register block.

This function is used to quickly receive one 10-16 bits frame or two 4-9 bit frames by reading the RXDOUBLEX register directly, without checking the STATUS register for the RXDATAV flag. This can be useful from the RXDATAV interrupt handler, i.e., waiting is superfluous, in order to quickly read the received data.

Note

- Because this function does not check whether the RXDOUBLEX register actually holds valid data, it should only be used in situations when it is certain that there is valid data, ensured by some external program routine, e.g., when handling an RXDATAV interrupt. The [USART_RxDoubleExt\(\)](#) is normally a better choice if the validity of the RXDOUBLEX register is not certain.
- Notice that possible parity/stop bits in asynchronous mode are not considered part of specified frame bit length.

Returns

- Data received.

USART_RxDataXGet

```
uint16_t USART_RxDataXGet (USART_TypeDef * usart)
```

Receive one 4-9 bit frame, (or part of 10-16 bit frame) with extended information.

Parameters

Type	Direction	Argument Name	Description
USART_TypeDef *	[in]	usart	Pointer to USART/UART peripheral register block.

This function is used to quickly receive one 4-9 bit frame, (or part of 10-16 bit frame) with extended information by reading the RXDATAX register directly, without checking the STATUS register for the RXDATAV flag. This can be useful from the RXDATAV interrupt handler, i.e., waiting is superfluous, in order to quickly read the received data.

Note

- Because this function does not check whether the RXDATAX register actually holds valid data, it should only be used in situations when it is certain that there is valid data, ensured by some external program routine, e.g., when handling an RXDATAV interrupt. The [USART_RxExt\(\)](#) is normally a better choice if the validity of the RXDATAX register is not certain.
- Notice that possible parity/stop bits in asynchronous mode are not considered part of specified frame bit length.

Returns

- Data received.

Macro Definition Documentation

USART_INITASYNC_DEFAULT

```
#define USART_INITASYNC_DEFAULT
```

Value:

```

0  {
0      usartEnable,      /* Enable RX/TX when initialization is complete. */ \
0      0,                /* Use current configured reference clock for configuring baud rate. */ \
0      115200,           /* 115200 bits/s. */ \
0      usartOVS16,       /* 16x oversampling. */ \
0      usartDatabits8,   /* 8 data bits. */ \
0      usartNoParity,    /* No parity. */ \
0      usartStopbits1,  /* 1 stop bit. */ \
0      false,           /* Do not disable majority vote. */ \
0      false,           /* Not USART PRS input mode. */ \
0      0,               /* PRS channel 0. */ \
0      false,           /* Auto CS functionality enable/disable switch */ \
0      false,           /* No CS invert. */ \
0      0,               /* Auto CS Hold cycles. */ \
0      0,               /* Auto CS Setup cycles. */ \
0      usartHwFlowControlNone /* No HW flow control. */ \
0  }

```

Default configuration for USART asynchronous initialization structure.

USART_INITPRSTRIGGER_DEFAULT

```
#define USART_INITPRSTRIGGER_DEFAULT
```

Value:

```

0  {
0      false,           /* Do not enable autoTX triggering. */ \
0      false,           /* Do not enable receive triggering. */ \
0      false,           /* Do not enable transmit triggering. */ \
0      0,              /* Set default channel to zero. */ \
0  }

```

Default configuration for USART PRS triggering structure.

USART_INITSYNC_DEFAULT

```
#define USART_INITSYNC_DEFAULT
```

Value:

```

0  {
0      usartEnable,      /* Enable RX/TX when initialization is complete. */ \
0      0,                /* Use current configured reference clock for configuring baud rate. */ \
0      1000000,          /* 1 Mbits/s. */ \
0      usartDatabits8,   /* 8 databits. */ \
0      true,             /* Master mode. */ \
0      false,           /* Send least significant bit first. */ \
0      usartClockMode0,  /* Clock idle low, sample on rising edge. */ \
0      false,           /* Not USART PRS input mode. */ \
0      0,               /* PRS channel 0. */ \
0      false,           /* No AUTOTX mode. */ \
0      false,           /* No AUTOCS mode. */ \
0      false,           /* No CS invert. */ \
0      0,               /* Auto CS Hold cycles. */ \
0      0,               /* Auto CS Setup cycles. */ \
0  }

```

Default configuration for USART sync initialization structure.

USART_INITIRDA_DEFAULT

```
#define USART_INITIRDA_DEFAULT
```

Value:

```

0  {
0  {
0      usartEnable,      /* Enable RX/TX when initialization is complete. */ \
0      0,                /* Use current configured reference clock for configuring baud rate. */ \
0      115200,           /* 115200 bits/s. */ \
0      usartOVS16,       /* 16x oversampling. */ \
0      usartDataBits8,   /* 8 data bits. */ \
0      usartEvenParity,  /* Even parity. */ \
0      usartStopBits1,   /* 1 stop bit. */ \
0      false,            /* Do not disable majority vote. */ \
0      false,            /* Not USART PRS input mode. */ \
0      0,                /* PRS channel 0. */ \
0      false,            /* Auto CS functionality enable/disable switch */ \
0      false,            /* No CS invert. */ \
0      0,                /* Auto CS Hold cycles */ \
0      0,                /* Auto CS Setup cycles */ \
0      usartHwFlowControlNone /* No HW flow control */ \
0  },
0  false,                /* Rx invert disabled. */ \
0  false,                /* Filtering disabled. */ \
0  usartIrDAPwTHREE /* Pulse width is set to ONE. */ \
0  }

```

Default configuration for IrDA mode initialization structure.

USART_INITI2S_DEFAULT

```
#define USART_INITI2S_DEFAULT
```

Value:

```

0  {
0  {
0      usartEnableTx,    /* Enable TX when init completed. */ \
0      0,                /* Use current configured reference clock for configuring baudrate. */ \
0      1000000,          /* Baudrate 1M bits/s. */ \
0      usartDataBits16,  /* 16 databits. */ \
0      true,             /* Operate as I2S master. */ \
0      true,             /* Most significant bit first. */ \
0      usartClockMode0,  /* Clock idle low, sample on rising edge. */ \
0      false,            /* Don't enable USARTRx via PRS. */ \
0      usartPrsRxCh0,    /* PRS channel selection (dummy). */ \
0      false,            /* Disable AUTOTX mode. */ \
0      false,            /* No AUTOCS mode */ \
0      false,            /* No CS invert. */ \
0      0,                /* Auto CS Hold cycles */ \
0      0,                /* Auto CS Setup cycles */ \
0  },
0  usartI2sFormatW16D16, /* 16-bit word, 16-bit data */ \
0  true,                 /* Delay on I2S data. */ \
0  false,                /* No DMA split. */ \
0  usartI2sJustifyLeft, /* Data is left-justified within the frame */ \
0  false                 /* Stereo mode. */ \
0  }

```

Default configuration for I2S mode initialization structure.

USART_InitAsync_TypeDef

Asynchronous mode initialization structure.

Public Attributes

USART_Enable_TypeDef	enable	Specifies whether TX and/or RX is enabled when initialization is completed.
uint32_t	refFreq	USART/UART reference clock assumed when configuring baud rate setup.
uint32_t	baudrate	Desired baud rate.
USART_OVS_TypeDef	oversampling	Oversampling used.
USART_Databits_TypeDef	databits	Number of data bits in frame.
USART_Parity_TypeDef	parity	Parity mode to use.
USART_Stopbits_TypeDef	stopbits	Number of stop bits to use.
bool	mvdiss	Majority Vote Disable for 16x, 8x and 6x oversampling modes.
bool	prsrxenable	Enable USART Rx via PRS.
USART_PRS_Channel_t	prsrxch	Select PRS channel for USART Rx.
bool	autoCsenable	Auto CS enabling.
bool	csInv	Enable CS invert.
uint8_t	autoCsHold	Auto CS hold time in baud cycles.
uint8_t	autoCsSetup	Auto CS setup time in baud cycles.
USART_HwFlowControl_TypeDef	hwFlowControl	Hardware flow control mode.

Public Attribute Documentation

enable

USART_Enable_TypeDef USART_InitAsync_TypeDef::enable

Specifies whether TX and/or RX is enabled when initialization is completed.

refFreq

```
uint32_t USART_InitAsync_TypeDef::refFreq
```

USART/UART reference clock assumed when configuring baud rate setup.

Set to 0 to use the currently configured reference clock.

baudrate

```
uint32_t USART_InitAsync_TypeDef::baudrate
```

Desired baud rate.

oversampling

```
USART_OVS_TypeDef USART_InitAsync_TypeDef::oversampling
```

Oversampling used.

databits

```
USART_Databits_TypeDef USART_InitAsync_TypeDef::databits
```

Number of data bits in frame.

Notice that UART modules only support 8 or 9 data bits.

parity

```
USART_Parity_TypeDef USART_InitAsync_TypeDef::parity
```

Parity mode to use.

stopbits

```
USART_Stopbits_TypeDef USART_InitAsync_TypeDef::stopbits
```

Number of stop bits to use.

mvdiss

```
bool USART_InitAsync_TypeDef::mvdiss
```

Majority Vote Disable for 16x, 8x and 6x oversampling modes.

prsrRxEnable

```
bool USART_InitAsync_TypeDef::prsRxEnable
```

Enable USART Rx via PRS.

prsRxCh

```
USART_PRS_Channel_t USART_InitAsync_TypeDef::prsRxCh
```

Select PRS channel for USART Rx.

(Only valid if prsRxEnable is true).

autoCsEnable

```
bool USART_InitAsync_TypeDef::autoCsEnable
```

Auto CS enabling.

csInv

```
bool USART_InitAsync_TypeDef::csInv
```

Enable CS invert.

By default, chip select is active low. Set to true to make chip select active high.

autoCsHold

```
uint8_t USART_InitAsync_TypeDef::autoCsHold
```

Auto CS hold time in baud cycles.

autoCsSetup

```
uint8_t USART_InitAsync_TypeDef::autoCsSetup
```

Auto CS setup time in baud cycles.

hwFlowControl

```
USART_HwFlowControl_TypeDef USART_InitAsync_TypeDef::hwFlowControl
```

Hardware flow control mode.

USART_PrsTriggerInit_TypeDef

USART PRS trigger enable.

Public Attributes

bool	autoTxTriggerEnable Enable AUTOTX.
bool	rxTriggerEnable Trigger receive via PRS channel.
bool	txTriggerEnable Trigger transmit via PRS channel.
USART_PRS_Channel_t	prsTriggerChannel PRS channel to be used to trigger auto transmission.

Public Attribute Documentation

autoTxTriggerEnable

```
bool USART_PrsTriggerInit_TypeDef::autoTxTriggerEnable
```

Enable AUTOTX.

rxTriggerEnable

```
bool USART_PrsTriggerInit_TypeDef::rxTriggerEnable
```

Trigger receive via PRS channel.

txTriggerEnable

```
bool USART_PrsTriggerInit_TypeDef::txTriggerEnable
```

Trigger transmit via PRS channel.

prsTriggerChannel

```
USART_PRS_Channel_t USART_PrsTriggerInit_TypeDef::prsTriggerChannel
```

PRS channel to be used to trigger auto transmission.

USART_InitSync_TypeDef

Synchronous mode initialization structure.

Public Attributes

USART_Enable_TypeDef	enable	Specifies whether TX and/or RX shall be enabled when initialization is completed.
uint32_t	refFreq	USART/UART reference clock assumed when configuring baud rate setup.
uint32_t	baudrate	Desired baud rate.
USART_Databits_TypeDef	databits	Number of data bits in frame.
bool	master	Select if to operate in master or slave mode.
bool	msbf	Select if to send most or least significant bit first.
USART_ClockMode_TypeDef	clockMode	Clock polarity/phase mode.
bool	prsRxEnable	Enable USART Rx via PRS.
USART_PRS_Channel_t	prsRxCh	Select PRS channel for USART Rx.
bool	autoTx	Enable AUTOTX mode.
bool	autoCsEnable	Auto CS enabling.
bool	csInv	Enable CS invert.
uint8_t	autoCsHold	Auto CS hold time in baud cycles.
uint8_t	autoCsSetup	Auto CS setup time in baud cycles.

Public Attribute Documentation

enable

```
USART_Enable_TypeDef USART_InitSync_TypeDef::enable
```

Specifies whether TX and/or RX shall be enabled when initialization is completed.

refFreq

```
uint32_t USART_InitSync_TypeDef::refFreq
```

USART/UART reference clock assumed when configuring baud rate setup.

Set to 0 to use the currently configured reference clock.

baudrate

```
uint32_t USART_InitSync_TypeDef::baudrate
```

Desired baud rate.

databits

```
USART_Databits_TypeDef USART_InitSync_TypeDef::databits
```

Number of data bits in frame.

master

```
bool USART_InitSync_TypeDef::master
```

Select if to operate in master or slave mode.

msbf

```
bool USART_InitSync_TypeDef::msbf
```

Select if to send most or least significant bit first.

clockMode

```
USART_ClockMode_TypeDef USART_InitSync_TypeDef::clockMode
```

Clock polarity/phase mode.

prsRxEnable

```
bool USART_InitSync_TypeDef::prsRxEnable
```

Enable USART Rx via PRS.

prsRxCh

```
USART_PRS_Channel_t USART_InitSync_TypeDef::prsRxCh
```

Select PRS channel for USART Rx.

(Only valid if prsRxEnable is true).

autoTx

```
bool USART_InitSync_TypeDef::autoTx
```

Enable AUTOTX mode.

Transmits as long as RX is not full. Generates underflows if TX is empty.

autoCsEnable

```
bool USART_InitSync_TypeDef::autoCsEnable
```

Auto CS enabling.

csInv

```
bool USART_InitSync_TypeDef::csInv
```

Enable CS invert.

By default, chip select is active low. Set to true to make chip select active high.

autoCsHold

```
uint8_t USART_InitSync_TypeDef::autoCsHold
```

Auto CS hold time in baud cycles.

autoCsSetup

```
uint8_t USART_InitSync_TypeDef::autoCsSetup
```

Auto CS setup time in baud cycles.

USART_InitIrDA_TypeDef

IrDA mode initialization structure.

Inherited from asynchronous mode initialization structure.

Public Attributes

USART_InitAsyn c_TypeDef	async General Asynchronous initialization structure.
bool	irRxInv Set to invert Rx signal before IrDA demodulator.
bool	irFilt Set to enable filter on IrDA demodulator.
USART_IrDAPw_ Typedef	irPw Configure the pulse width generated by the IrDA modulator as a fraction of the configured USART bit period.

Public Attribute Documentation

async

```
USART_InitAsync_TypeDef USART_InitIrDA_TypeDef::async
```

General Asynchronous initialization structure.

irRxInv

```
bool USART_InitIrDA_TypeDef::irRxInv
```

Set to invert Rx signal before IrDA demodulator.

irFilt

```
bool USART_InitIrDA_TypeDef::irFilt
```

Set to enable filter on IrDA demodulator.

irPw

```
USART_IrDAPw_Typedef USART_InitIrDA_TypeDef::irPw
```

Configure the pulse width generated by the IrDA modulator as a fraction of the configured USART bit period.

USART_InitI2s_TypeDef

I2S mode initialization structure.

Inherited from synchronous mode initialization structure.

Public Attributes

USART_InitSync_TypeDef	sync General Synchronous initialization structure.
USART_I2sFormat_TypeDef	format I2S mode.
bool	delay Delay on I2S data.
bool	dmaSplit Separate DMA Request For Left/Right Data.
USART_I2sJustify_TypeDef	justify Justification of I2S data within the frame.
bool	mono Stereo or Mono, set to true for mono.

Public Attribute Documentation

sync

```
USART_InitSync_TypeDef USART_InitI2s_TypeDef::sync
```

General Synchronous initialization structure.

format

```
USART_I2sFormat_TypeDef USART_InitI2s_TypeDef::format
```

I2S mode.

delay

```
bool USART_InitI2s_TypeDef::delay
```

Delay on I2S data.

Set to add a one-cycle delay between a transition on the word-clock and the start of the I2S word. Should be set for standard I2S format.

dmaSplit

```
bool USART_InitI2s_TypeDef::dmaSplit
```

Separate DMA Request For Left/Right Data.

justify

```
USART_I2sJustify_TypeDef USART_InitI2s_TypeDef::justify
```

Justification of I2S data within the frame.

mono

```
bool USART_InitI2s_TypeDef::mono
```

Stereo or Mono, set to true for mono.

VERSION - Version Defines

VERSION - Version Defines

Version API.

Macros specifying the EMLIB and CMSIS version.

Macros

```
#define  _CMSIS_VERSION 5.8.0
    Version number of targeted CMSIS package.

#define  _CMSIS_VERSION_MAJOR 5
    Major version of CMSIS.

#define  _CMSIS_VERSION_MINOR 8
    Minor version of CMSIS.

#define  _CMSIS_VERSION_PATCH 0
    Patch revision of CMSIS.
```

Macro Definition Documentation

_CMSIS_VERSION

#define _CMSIS_VERSION

Value:

5.8.0

Version number of targeted CMSIS package.

_CMSIS_VERSION_MAJOR

#define _CMSIS_VERSION_MAJOR

Value:

5

Major version of CMSIS.

_CMSIS_VERSION_MINOR

#define _CMSIS_VERSION_MINOR

Value:

8

Minor version of CMSIS.

_CMSIS_VERSION_PATCH

```
#define _CMSIS_VERSION_PATCH
```

Value:

```
0
```

Patch revision of CMSIS.

WDOG - Watchdog

WDOG - Watchdog

Watchdog (WDOG) Peripheral API.

This module contains functions to control the WDOG peripheral of Silicon Labs 32-bit MCUs and SoCs. The WDOG resets the system in case of a fault condition.

Modules

[WDOG_Init_TypeDef](#)

Enumerations

```
enum WDOG\_PeriodSel\_TypeDef {
    wdogPeriod_9 = 0x0
    wdogPeriod_17 = 0x1
    wdogPeriod_33 = 0x2
    wdogPeriod_65 = 0x3
    wdogPeriod_129 = 0x4
    wdogPeriod_257 = 0x5
    wdogPeriod_513 = 0x6
    wdogPeriod_1k = 0x7
    wdogPeriod_2k = 0x8
    wdogPeriod_4k = 0x9
    wdogPeriod_8k = 0xA
    wdogPeriod_16k = 0xB
    wdogPeriod_32k = 0xC
    wdogPeriod_64k = 0xD
    wdogPeriod_128k = 0xE
    wdogPeriod_256k = 0xF
}
Watchdog clock selection.
```

```
enum WDOG\_WarnSel\_TypeDef {
    wdogWarnDisable = 0
    wdogWarnTime25pct = 1
    wdogWarnTime50pct = 2
    wdogWarnTime75pct = 3
}
Select Watchdog warning timeout period as percentage of timeout.
```

```
enum WDOG\_WinSel\_TypeDef {
    wdogIllegalWindowDisable = 0
    wdogIllegalWindowTime12_5pct = 1
    wdogIllegalWindowTime25_0pct = 2
    wdogIllegalWindowTime37_5pct = 3
    wdogIllegalWindowTime50_0pct = 4
    wdogIllegalWindowTime62_5pct = 5
    wdogIllegalWindowTime75_0pct = 6
    wdogIllegalWindowTime87_5pct = 7
}
Select Watchdog illegal window limit.
```

Functions

```
void WDOGN\_Enable(WDOG_TypeDef *wdog, bool enable)
    Enable/disable the watchdog timer.
```

```
void WDOGN\_Feed(WDOG_TypeDef *wdog)
    Feed WDOG.
```

void	WDOGn_Init (WDOG_TypeDef *wdog, const WDOG_Init_TypeDef *init) Initialize WDOG (assuming the WDOG configuration has not been locked).
void	WDOGn_Lock (WDOG_TypeDef *wdog) Lock the WDOG configuration.
void	WDOGn_SyncWait (WDOG_TypeDef *wdog) Wait for the WDOG to complete all synchronization of register changes and commands.
void	WDOGn_Unlock (WDOG_TypeDef *wdog) Unlock the WDOG configuration.
void	WDOGn_IntClear (WDOG_TypeDef *wdog, uint32_t flags) Clear one or more pending WDOG interrupts.
void	WDOGn_IntDisable (WDOG_TypeDef *wdog, uint32_t flags) Disable one or more WDOG interrupts.
void	WDOGn_IntEnable (WDOG_TypeDef *wdog, uint32_t flags) Enable one or more WDOG interrupts.
uint32_t	WDOGn_IntGet (WDOG_TypeDef *wdog) Get pending WDOG interrupt flags.
uint32_t	WDOGn_IntGetEnabled (WDOG_TypeDef *wdog) Get enabled and pending WDOG interrupt flags.
void	WDOGn_IntSet (WDOG_TypeDef *wdog, uint32_t flags) Set one or more pending WDOG interrupts from SW.
bool	WDOGn_IsEnabled (WDOG_TypeDef *wdog) Get enabled status of the Watchdog.
bool	WDOGn_IsLocked (WDOG_TypeDef *wdog) Get locked status of the Watchdog.

Macros

#define	DEFAULT_WDOG WDOG0 Default WDOG instance for deprecated functions.
#define	WDOG_INIT_DEFAULT undefined Suggested default configuration for WDOG initialization structure.
#define	WDOG_SYNC_TIMEOUT 30000 In some scenarios when the watchdog is disabled the synchronization register might be set and not be cleared until the watchdog is enabled again.

Enumeration Documentation

WDOG_PeriodSel_TypeDef

WDOG_PeriodSel_TypeDef	
Watchdog clock selection.	
Watchdog period selection.	
Enumerator	
wdogPeriod_9	9 clock periods
wdogPeriod_17	17 clock periods
wdogPeriod_33	33 clock periods

wdogPeriod_65	65 clock periods
wdogPeriod_129	129 clock periods
wdogPeriod_257	257 clock periods
wdogPeriod_513	513 clock periods
wdogPeriod_1k	1025 clock periods
wdogPeriod_2k	2049 clock periods
wdogPeriod_4k	4097 clock periods
wdogPeriod_8k	8193 clock periods
wdogPeriod_16k	16385 clock periods
wdogPeriod_32k	32769 clock periods
wdogPeriod_64k	65537 clock periods
wdogPeriod_128k	131073 clock periods
wdogPeriod_256k	262145 clock periods

WDOG_WarnSel_TypeDef

WDOG_WarnSel_TypeDef

Select Watchdog warning timeout period as percentage of timeout.

Enumerator	
wdogWarnDisable	Watchdog warning period is disabled.
wdogWarnTime25pct	Watchdog warning period is 25% of the timeout.
wdogWarnTime50pct	Watchdog warning period is 50% of the timeout.
wdogWarnTime75pct	Watchdog warning period is 75% of the timeout.

WDOG_WinSel_TypeDef

WDOG_WinSel_TypeDef

Select Watchdog illegal window limit.

Enumerator	
wdogIllegalWindowDisable	Watchdog illegal window disabled.
wdogIllegalWindowTime12_5pct	Window timeout is 12.5% of the timeout.
wdogIllegalWindowTime25_0pct	Window timeout is 25% of the timeout.
wdogIllegalWindowTime37_5pct	Window timeout is 37.5% of the timeout.
wdogIllegalWindowTime50_0pct	Window timeout is 50% of the timeout.
wdogIllegalWindowTime62_5pct	Window timeout is 62.5% of the timeout.
wdogIllegalWindowTime75_0pct	Window timeout is 75% of the timeout.
wdogIllegalWindowTime87_5pct	Window timeout is 87.5% of the timeout.

Function Documentation

WDOGN_Enable

```
void WDOGn_Enable (WDOG_TypeDef * wdog, bool enable)
```

Enable/disable the watchdog timer.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	A pointer to the WDOG peripheral register block.
bool	[in]	enable	True to enable Watchdog, false to disable. Watchdog cannot be disabled if it's been locked.

Note

- This function modifies the WDOG CTRL register which requires synchronization into the low-frequency domain. If this register is modified before a previous update to the same register has completed, this function will stall until the previous synchronization has completed.

WDOGn_Feed

```
void WDOGn_Feed (WDOG_TypeDef * wdog)
```

Feed WDOG.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	A pointer to the WDOG peripheral register block.

When WDOG is activated, it must be fed (i.e., clearing the counter) before it reaches the defined timeout period. Otherwise, WDOG will generate a reset.

Note

- Note that WDOG is an asynchronous peripheral and when calling the [WDOGn_Feed\(\)](#) function the hardware starts the process of clearing the counter. This process takes some time before it completes depending on the selected oscillator (up to 4 peripheral clock cycles). When using the ULFRCO for instance as the oscillator the watchdog runs on a 1 kHz clock and a watchdog clear operation might take up to 4 ms.

If the device enters EM2 or EM3 while a command is in progress then that command will be aborted. An application can use [WDOGn_SyncWait\(\)](#) to wait for a command to complete.

WDOGn_Init

```
void WDOGn_Init (WDOG_TypeDef * wdog, const WDOG_Init_TypeDef * init)
```

Initialize WDOG (assuming the WDOG configuration has not been locked).

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.
const WDOG_Init_TypeDef *	[in]	init	The structure holding the WDOG configuration. A default setting WDOG_INIT_DEFAULT is available for initialization.

Note

- This function modifies the WDOG CTRL register which requires synchronization into the low-frequency domain. If this register is modified before a previous update to the same register has completed, this function will stall until the previous synchronization has completed.

WDOGn_Lock

```
void WDOGn_Lock (WDOG_TypeDef * wdog)
```

Lock the WDOG configuration.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	A pointer to WDOG peripheral register block.

This prevents errors from overwriting the WDOG configuration, possibly disabling it. Only a reset can unlock the WDOG configuration once locked.

If the LFRCO or LFXO clocks are used to clock WDOG, consider using the option of inhibiting those clocks to be disabled. See the WDOG_Enable() initialization structure.

Note

- This function modifies the WDOG CTRL register which requires synchronization into the low-frequency domain. If this register is modified before a previous update to the same register has completed, this function will stall until the previous synchronization has completed.

WDOGn_SyncWait

```
void WDOGn_SyncWait (WDOG_TypeDef * wdog)
```

Wait for the WDOG to complete all synchronization of register changes and commands.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	A pointer to WDOG peripheral register block.

WDOGn_Unlock

```
void WDOGn_Unlock (WDOG_TypeDef * wdog)
```

Unlock the WDOG configuration.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	A pointer to WDOG peripheral register block.

Note that this function will have no effect on devices where a reset is the only way to unlock the watchdog.

WDOGn_IntClear

```
void WDOGn_IntClear (WDOG_TypeDef * wdog, uint32_t flags)
```

Clear one or more pending WDOG interrupts.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.
uint32_t	[in]	flags	WDOG interrupt sources to clear. Use a set of interrupt flags OR-ed together to clear multiple interrupt sources.

WDOGn_IntDisable

```
void WDOGn_IntDisable (WDOG_TypeDef * wdog, uint32_t flags)
```

Disable one or more WDOG interrupts.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.
uint32_t	[in]	flags	WDOG interrupt sources to disable. Use a set of interrupt flags OR-ed together to disable multiple interrupt.

WDOGn_IntEnable

```
void WDOGn_IntEnable (WDOG_TypeDef * wdog, uint32_t flags)
```

Enable one or more WDOG interrupts.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.
uint32_t	[in]	flags	WDOG interrupt sources to enable. Use a set of interrupt flags OR-ed together to set multiple interrupt.

Note

- Depending on the use, a pending interrupt may already be set prior to enabling the interrupt. To ignore a pending interrupt, consider using WDOG_IntClear() prior to enabling the interrupt.

WDOGn_IntGet

```
uint32_t WDOGn_IntGet (WDOG_TypeDef * wdog)
```

Get pending WDOG interrupt flags.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.

Note

- The event bits are not cleared by the use of this function.

Returns

- Pending WDOG interrupt sources. Returns a set of interrupt flags OR-ed together for the interrupt sources set.

WDOGN_IntGetEnabled

```
uint32_t WDOGN_IntGetEnabled (WDOG_TypeDef * wdog)
```

Get enabled and pending WDOG interrupt flags.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.

Useful for handling more interrupt sources in the same interrupt handler.

Returns

- Pending and enabled WDOG interrupt sources. Returns a set of interrupt flags OR-ed together for the interrupt sources set.

WDOGN_IntSet

```
void WDOGN_IntSet (WDOG_TypeDef * wdog, uint32_t flags)
```

Set one or more pending WDOG interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.
uint32_t	[in]	flags	WDOG interrupt sources to set to pending. Use a set of interrupt flags (WDOG_IFS_nnn).

WDOGN_IsEnabled

```
bool WDOGN_IsEnabled (WDOG_TypeDef * wdog)
```

Get enabled status of the Watchdog.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.

Returns

- True if Watchdog is enabled.

WDOGn_IsLocked

```
bool WDOGn_IsLocked (WDOG_TypeDef * wdog)
```

Get locked status of the Watchdog.

Parameters

Type	Direction	Argument Name	Description
WDOG_TypeDef *	[in]	wdog	Pointer to the WDOG peripheral register block.

Returns

- True if Watchdog is locked.

Macro Definition Documentation

DEFAULT_WDOG

```
#define DEFAULT_WDOG
```

Value:

```
WDOG0
```

Default WDOG instance for deprecated functions.

WDOG_INIT_DEFAULT

```
#define WDOG_INIT_DEFAULT
```

Value:

```
0      {
0      true,           /* Start Watchdog when initialization is done. */
0      false,          /* WDOG is not counting during debug halt. */
0      false,          /* The clear bit will clear the WDOG counter. */
0      false,          /* WDOG is not counting when in EM2. */
0      false,          /* WDOG is not counting when in EM3. */
0      false,          /* EM4 can be entered. */
0      false,          /* PRS Source 0 missing event will not trigger a WDOG reset. */
0      false,          /* PRS Source 1 missing event will not trigger a WDOG reset. */
0      false,          /* Do not lock WDOG configuration. */
0      wdogPeriod_256k, /* Set longest possible timeout period. */
0      wdogWarnDisable, /* Disable warning interrupt. */
0      wdogIllegalWindowDisable, /* Disable illegal window interrupt. */
0      false           /* Do not disable reset. */
0  }
```

Suggested default configuration for WDOG initialization structure.

WDOG_SYNC_TIMEOUT

```
#define WDOG_SYNC_TIMEOUT
```

Value:

30000

In some scenarios when the watchdog is disabled the synchronization register might be set and not be cleared until the watchdog is enabled again.

This will happen when for instance some watchdog register is modified while the watchdog clock is disabled. In these scenarios we need to make sure that the software does not wait forever.

WDOG_Init_TypeDef

Watchdog initialization structure.

Public Attributes

bool	enable	Enable Watchdog when initialization completed.
bool	debugRun	Counter keeps running during debug halt.
bool	clrSrc	Select WDOG clear source: False: Write to the clear bit will clear the WDOG counter True: Rising edge on the PRS Source 0 will clear the WDOG counter.
bool	em2Run	Counter keeps running when in EM2.
bool	em3Run	Counter keeps running when in EM3.
bool	em4Block	Block EMU from entering EM4.
bool	prs0MissRstEn	When set, a PRS Source 0 missing event will trigger a WDOG reset.
bool	prs1MissRstEn	When set, a PRS Source 1 missing event will trigger a WDOG reset.
bool	lock	Block SW from disabling LFRCO/LFXO oscillators.
WDOG_PeriodSel_TypeDef	perSel	Clock source to use for Watchdog.
WDOG_WarnSel_TypeDef	warnSel	Select warning time as % of the Watchdog timeout.
WDOG_WinSel_TypeDef	winSel	Select illegal window time as % of the Watchdog timeout.
bool	resetDisable	Disable Watchdog reset output if true.

Public Attribute Documentation

enable

```
bool WDOG_Init_TypeDef::enable
```

Enable Watchdog when initialization completed.

debugRun

```
bool WDOG_Init_TypeDef::debugRun
```

Counter keeps running during debug halt.

clrSrc

```
bool WDOG_Init_TypeDef::clrSrc
```

Select WDOG clear source: False: Write to the clear bit will clear the WDOG counter True: Rising edge on the PRS Source 0 will clear the WDOG counter.

em2Run

```
bool WDOG_Init_TypeDef::em2Run
```

Counter keeps running when in EM2.

em3Run

```
bool WDOG_Init_TypeDef::em3Run
```

Counter keeps running when in EM3.

em4Block

```
bool WDOG_Init_TypeDef::em4Block
```

Block EMU from entering EM4.

prs0MissRstEn

```
bool WDOG_Init_TypeDef::prs0MissRstEn
```

When set, a PRS Source 0 missing event will trigger a WDOG reset.

prs1MissRstEn

```
bool WDOG_Init_TypeDef::prs1MissRstEn
```

When set, a PRS Source 1 missing event will trigger a WDOG reset.

lock

```
bool WDOG_Init_TypeDef::lock
```

Block SW from disabling LFRCO/LFXO oscillators.

Block SW from modifying the configuration (a reset is needed to reconfigure).

```
WDOG_PeriodSel_TypeDef WDOG_Init_TypeDef::perSel
```

Clock source to use for Watchdog.

Watchdog timeout period.

warnSel

```
WDOG_WarnSel_TypeDef WDOG_Init_TypeDef::warnSel
```

Select warning time as % of the Watchdog timeout.

winSel

```
WDOG_WinSel_TypeDef WDOG_Init_TypeDef::winSel
```

Select illegal window time as % of the Watchdog timeout.

resetDisable

```
bool WDOG_Init_TypeDef::resetDisable
```

Disable Watchdog reset output if true.

EFM32PG22

API Documentation

List of modules	Description
BURTC - Backup RTC	Backup Real Time Counter (BURTC) Peripheral API.
BUS - Bitfield Read/Write	BUS register and RAM bit/field read/write API.
CHIP - Chip Errata Workarounds	Chip errata workaround APIs.
CMU - Clock Management Unit	Clock management unit (CMU) Peripheral API.
CORE - Core Interrupt	Core interrupt handling API.
DBG - Debug	Debug (DBG) Peripheral API.
EMU - Energy Management Unit	Energy Management Unit (EMU) Peripheral API.
EUSART - Extended USART	Extended Universal Synchronous/Asynchronous Receiver/Transmitter.
GPCRC - General Purpose CRC	General Purpose Cyclic Redundancy Check (GPCRC) API.
GPIO - General Purpose Input/Output	General Purpose Input/Output (GPIO) API.
I2C - Inter-Integrated Circuit	Inter-integrated Circuit (I2C) Peripheral API.
IADC - Current DAC	Incremental Analog to Digital Converter (IADC) Peripheral API.
LDMA - Linked DMA	Linked Direct Memory Access (LDMA) Peripheral API.
LETIMER - Low Energy Timer	Low Energy Timer (LETIMER) Peripheral API.
LEUART - Low Energy UART	Low Energy Universal Asynchronous Receiver/Transmitter (LEUART) Peripheral API.
MSC - Memory System Controller	Memory System Controller API.
PDM - Pulse Density Modulation	Pulse Density Modulation (PDM) peripheral API.
PRS - Peripheral Reflex System	Peripheral Reflex System (PRS) Peripheral API.
RAMFUNC - RAM Function Support	RAM code support.
RMU - Reset Management Unit	Reset Management Unit (RMU) Peripheral API.
RTCC - Real Timer Counter/Calendar	Real Time Counter and Calendar (RTCC) Peripheral API.
SE - Secure Element	Secure Element peripheral API.
[Deprecated Functions]	Deprecated Functions
SMU - Security Management Unit	Security Management Unit (SMU) Peripheral API.
SYSTEM - System Utils	System API.
TIMER - Timer/Counter	Timer/Counter (TIMER) Peripheral API.
USART - Synchronous/Asynchronous Serial	Universal Synchronous/Asynchronous Receiver/Transmitter Peripheral API.
VERSION - Version Defines	Version API.
WDOG - Watchdog	Watchdog (WDOG) Peripheral API.