EFR32XG28

- ACMP - Analog Comparator
  - ACMP_CapsenseInit_TypeDef
  - ACMP_Init_TypeDef
- 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_BUFOUTLeaderInit_TypeDef
  - CMU_CrystalSharingFollowerInit_TypeDef
  - CMU_DPLLInit_TypeDef
- 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_SpiAdvancedInit_TypeDef
  - EUSART_SpiInit_TypeDef
  - EUSART_DaliInit_TypeDef
- GPCRC - General Purpose CRC
  - GPCRC_Init_TypeDef
- GPIO - General Purpose Input/Output
- I2C - Inter-Integrated Circuit
  - I2C_Init_TypeDef
  - I2C_TransferSeq_TypeDef
- IADC - Incremental ADC
  - IADC_Init_t

Copyright © 2026 Silicon Laboratories. All rights reserved.

IADC_Config_t
IADC_AllConfigs_t
IADC_InitScan_t
IADC_InitSingle_t
IADC_SingleInput_t
IADC_ScanTableEntry_t
IADC_ScanTable_t
IADC_Result_t
LCD - Liquid Crystal Display
LCD_AnimInit_TypeDef
LCD_FrameCountInit_TypeDef
LCD_Init_TypeDef
LDMA - Linked DMA
LDMA_Descriptor_t
LDMA_Init_t
LDMA_TransferCfg_t
LESENSE - Low Energy Sensor
LESENSE_CoreCtrlDesc_TypeDef
LESENSE_TimeCtrlDesc_TypeDef
LESENSE_PerCtrlDesc_TypeDef
LESENSE_DecCtrlDesc_TypeDef
LESENSE_Init_TypeDef
LESENSE_ChDesc_TypeDef
LESENSE_ChAll_TypeDef
LESENSE_AltExDesc_TypeDef
LESENSE_ConfAltEx_TypeDef
LESENSE_DecStCond_TypeDef
LESENSE_DecStAll_TypeDef
LETIMER - Low Energy Timer
LETIMER_Init_TypeDef
MSC - Memory System Controller
MSC_ExecConfig_TypeDef
MSC_EccConfig_TypeDef
PCNT - Pulse Counter
PCNT_Init_TypeDef
PCNT_Filter_TypeDef
PRS - Peripheral Reflex System
RAMFUNC - RAM Function Support
RMU - Reset Management Unit
SE - Secure Element

Copyright © 2026 Silicon Laboratories. All rights reserved.

Deprecated Functions
SMU - Security Management Unit
SMU_PrivilegedAccess_TypeDef
SMU_Init_TypeDef
SYSTEM - System Utils
SYSTEM_CalAddrVal_TypeDef
SYSTEM_ChipRevision_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
VDAC - Voltage DAC
VDAC_Init_TypeDef
VDAC_InitChannel_TypeDef
VERSION - Version Defines
WDOG - Watchdog
WDOG_Init_TypeDef
API Documentation

Copyright © 2026 Silicon Laboratories. All rights reserved.

ACMP - Analog Comparator

ACMP - Analog Comparator

Analog comparator (ACMP) Peripheral API.

The Analog Comparator is used to compare voltage of two analog inputs with a digital output indicating which input voltage is higher. Inputs can either be one of the selectable internal references or from external pins. Response time and current consumption can be configured by altering the current supply to the comparator.

ACMP is available down to EM3 and is able to wake up the system when input signals pass a certain threshold. Use ACMP_IntEnable() to enable an edge interrupt to use this functionality.

This example shows how to use the em_acmp.h API for comparing an input pin to an internal 2.5 V reference voltage.

```
/* Initialize with default settings. */
ACMP_Init_TypeDef init = ACMP_INIT_DEFAULT;
ACMP_Init(ACMP0, &init);

/* In this example we want to compare an analog input to the 2.5 V
 * internal reference. The default settings resets the divider for
 * acmpInputVREFDIV2V5, which we can use as a 2.5 V reference. */

/* Now we select the two inputs to compare. Here we compare the acmpInputPD2
 * input to the internal 2.5V reference. When acmpInputPD2 is lower than
 * 2.5 V then the ACMP output is 0 and when acmpInputPD2 is is higher than
 * 2.5 V then the ACMP output is 1. */
ACMP_ChannelSet(ACMP0, acmpInputVREFDIV2V5, acmpInputPD2);

/* Allocate CDEVEN0 to ACMP0 to be able to use the input. */
GPIO->CDBUSALLOC = GPIO_CDBUSALLOC_CDEVEN0_ACMP0;

/* To be able to probe the output we can send the ACMP output to a pin.
 * The second argument to this function is the pin location which is
 * device dependent. */
ACMP_GPIOSetup(ACMP0, gpioPortD, 1, true, false);
```

Note

- ACMP can also be used to compare two separate input pins.

ACMP also contains specialized hardware for capacitive sensing. This module contains the ACMP_CapsenseInit() function to initialize ACMP for capacitive sensing and the ACMP_CapsenseChannelSet() function to select the current capsense channel.

For applications that require capacitive sensing it is recommended to use a library, such as cslib, which is provided by Silicon Labs.

Modules

ACMP_CapsenseInit_TypeDef

ACMP_Init_TypeDef

Copyright © 2026 Silicon Laboratories. All rights reserved.

Enumerations

enum ACMP_CapsenseResistor_TypeDef {

acmpResistor0 = _ACMP_INPUTCTRL_CSRESSEL_RES0
acmpResistor1 = _ACMP_INPUTCTRL_CSRESSEL_RES1
acmpResistor2 = _ACMP_INPUTCTRL_CSRESSEL_RES2
acmpResistor3 = _ACMP_INPUTCTRL_CSRESSEL_RES3
acmpResistor4 = _ACMP_INPUTCTRL_CSRESSEL_RES4
acmpResistor5 = _ACMP_INPUTCTRL_CSRESSEL_RES5
acmpResistor6 = _ACMP_INPUTCTRL_CSRESSEL_RES6

}

Resistor values used for the internal capacitive sense resistor.

enum ACMP_HysteresisLevel_TypeDef {

acmpHysteresisDisabled = _ACMP_CFG_HYST_DISABLED
acmpHysteresis10Sym = _ACMP_CFG_HYST_SYM10MV
acmpHysteresis20Sym = _ACMP_CFG_HYST_SYM20MV
acmpHysteresis30Sym = _ACMP_CFG_HYST_SYM30MV
acmpHysteresis10Pos = _ACMP_CFG_HYST_POS10MV
acmpHysteresis20Pos = _ACMP_CFG_HYST_POS20MV
acmpHysteresis30Pos = _ACMP_CFG_HYST_POS30MV
acmpHysteresis10Neg = _ACMP_CFG_HYST_NEG10MV
acmpHysteresis20Neg = _ACMP_CFG_HYST_NEG20MV
acmpHysteresis30Neg = _ACMP_CFG_HYST_NEG30MV

}

Hysteresis level.

enum ACMP_InputRange_TypeDef {

acmpInputRangeFull = _ACMP_CFG_INPUTRANGE_FULL
acmpInputRangeReduced = _ACMP_CFG_INPUTRANGE_REDUCED

}

Adjust ACMP performance for a given input voltage range.

enum ACMP_Accuracy_TypeDef {

acmpAccuracyLow = _ACMP_CFG_ACCURACY_LOW
acmpAccuracyHigh = _ACMP_CFG_ACCURACY_HIGH

}

ACMP accuracy mode.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum ACMP_Channel_TypeDef {

acmpInputVSS = _ACMP_INPUTCTRL_POSSEL_VSS
acmpInputVREFDIVAVDD = _ACMP_INPUTCTRL_POSSEL_VREFDIVAVDD
acmpInputVREFDIVAVDDLP = _ACMP_INPUTCTRL_POSSEL_VREFDIVAVDDLP
acmpInputVREFDIV1V25 = _ACMP_INPUTCTRL_POSSEL_VREFDIV1V25
acmpInputVREFDIV1V25LP = _ACMP_INPUTCTRL_POSSEL_VREFDIV1V25LP
acmpInputVREFDIV2V5 = _ACMP_INPUTCTRL_POSSEL_VREFDIV2V5
acmpInputVREFDIV2V5LP = _ACMP_INPUTCTRL_POSSEL_VREFDIV2V5LP
acmpInputVSENSE01DIV4 = _ACMP_INPUTCTRL_POSSEL_VSENSE01DIV4
acmpInputVSENSE01DIV4LP = _ACMP_INPUTCTRL_POSSEL_VSENSE01DIV4LP
acmpInputVSENSE11DIV4 = _ACMP_INPUTCTRL_POSSEL_VSENSE11DIV4
acmpInputVSENSE11DIV4LP = _ACMP_INPUTCTRL_POSSEL_VSENSE11DIV4LP
acmpInputCAPSENSE = _ACMP_INPUTCTRL_NEGSEL_CAPSENSE
acmpInputVDACOUT0 = _ACMP_INPUTCTRL_POSSEL_VDACOUT0
acmpInputVDACOUT1 = _ACMP_INPUTCTRL_POSSEL_VDACOUT1
acmpInputEXTPA = _ACMP_INPUTCTRL_POSSEL_EXTPA
acmpInputEXTPB = _ACMP_INPUTCTRL_POSSEL_EXTPB
acmpInputEXTPC = _ACMP_INPUTCTRL_POSSEL_EXTPC
acmpInputEXTPD = _ACMP_INPUTCTRL_POSSEL_EXTPD
acmpInputPA0 = _ACMP_INPUTCTRL_POSSEL_PA0
acmpInputPA1 = _ACMP_INPUTCTRL_POSSEL_PA1
acmpInputPA2 = _ACMP_INPUTCTRL_POSSEL_PA2
acmpInputPA3 = _ACMP_INPUTCTRL_POSSEL_PA3
acmpInputPA4 = _ACMP_INPUTCTRL_POSSEL_PA4
acmpInputPA5 = _ACMP_INPUTCTRL_POSSEL_PA5
acmpInputPA6 = _ACMP_INPUTCTRL_POSSEL_PA6
acmpInputPA7 = _ACMP_INPUTCTRL_POSSEL_PA7
acmpInputPA8 = _ACMP_INPUTCTRL_POSSEL_PA8
acmpInputPA9 = _ACMP_INPUTCTRL_POSSEL_PA9
acmpInputPA10 = _ACMP_INPUTCTRL_POSSEL_PA10
acmpInputPA11 = _ACMP_INPUTCTRL_POSSEL_PA11
acmpInputPA12 = _ACMP_INPUTCTRL_POSSEL_PA12
acmpInputPA13 = _ACMP_INPUTCTRL_POSSEL_PA13
acmpInputPA14 = _ACMP_INPUTCTRL_POSSEL_PA14
acmpInputPA15 = _ACMP_INPUTCTRL_POSSEL_PA15
acmpInputPB0 = _ACMP_INPUTCTRL_POSSEL_PB0
acmpInputPB1 = _ACMP_INPUTCTRL_POSSEL_PB1
acmpInputPB2 = _ACMP_INPUTCTRL_POSSEL_PB2
acmpInputPB3 = _ACMP_INPUTCTRL_POSSEL_PB3
acmpInputPB4 = _ACMP_INPUTCTRL_POSSEL_PB4
acmpInputPB5 = _ACMP_INPUTCTRL_POSSEL_PB5
acmpInputPB6 = _ACMP_INPUTCTRL_POSSEL_PB6
acmpInputPB7 = _ACMP_INPUTCTRL_POSSEL_PB7
acmpInputPB8 = _ACMP_INPUTCTRL_POSSEL_PB8
acmpInputPB9 = _ACMP_INPUTCTRL_POSSEL_PB9
acmpInputPB10 = _ACMP_INPUTCTRL_POSSEL_PB10
acmpInputPB11 = _ACMP_INPUTCTRL_POSSEL_PB11
acmpInputPB12 = _ACMP_INPUTCTRL_POSSEL_PB12
acmpInputPB13 = _ACMP_INPUTCTRL_POSSEL_PB13
acmpInputPB14 = _ACMP_INPUTCTRL_POSSEL_PB14
acmpInputPB15 = _ACMP_INPUTCTRL_POSSEL_PB15
acmpInputPC0 = _ACMP_INPUTCTRL_POSSEL_PC0
acmpInputPC1 = _ACMP_INPUTCTRL_POSSEL_PC1
acmpInputPC2 = _ACMP_INPUTCTRL_POSSEL_PC2
acmpInputPC3 = _ACMP_INPUTCTRL_POSSEL_PC3
acmpInputPC4 = _ACMP_INPUTCTRL_POSSEL_PC4
acmpInputPC5 = _ACMP_INPUTCTRL_POSSEL_PC5
acmpInputPC6 = _ACMP_INPUTCTRL_POSSEL_PC6
acmpInputPC7 = _ACMP_INPUTCTRL_POSSEL_PC7
acmpInputPC8 = _ACMP_INPUTCTRL_POSSEL_PC8
acmpInputPC9 = _ACMP_INPUTCTRL_POSSEL_PC9
acmpInputPC10 = _ACMP_INPUTCTRL_POSSEL_PC10
acmpInputPC11 = _ACMP_INPUTCTRL_POSSEL_PC11
acmpInputPC12 = _ACMP_INPUTCTRL_POSSEL_PC12
acmpInputPC13 = _ACMP_INPUTCTRL_POSSEL_PC13
acmpInputPC14 = _ACMP_INPUTCTRL_POSSEL_PC14
acmpInputPC15 = _ACMP_INPUTCTRL_POSSEL_PC15

Copyright © 2026 Silicon Laboratories. All rights reserved.

acmpInputPD0 =
_ACMP_INPUTCTRL_POSSEL_PD0
acmpInputPD1 =
_ACMP_INPUTCTRL_POSSEL_PD1
acmpInputPD2 =
_ACMP_INPUTCTRL_POSSEL_PD2
acmpInputPD3 =
_ACMP_INPUTCTRL_POSSEL_PD3
acmpInputPD4 =
_ACMP_INPUTCTRL_POSSEL_PD4
acmpInputPD5 =
_ACMP_INPUTCTRL_POSSEL_PD5
acmpInputPD6 =
_ACMP_INPUTCTRL_POSSEL_PD6
acmpInputPD7 =
_ACMP_INPUTCTRL_POSSEL_PD7
acmpInputPD8 =
_ACMP_INPUTCTRL_POSSEL_PD8
acmpInputPD9 =
_ACMP_INPUTCTRL_POSSEL_PD9
acmpInputPD10 =
_ACMP_INPUTCTRL_POSSEL_PD10
acmpInputPD11 =
_ACMP_INPUTCTRL_POSSEL_PD11
acmpInputPD12 =
_ACMP_INPUTCTRL_POSSEL_PD12
acmpInputPD13 =
_ACMP_INPUTCTRL_POSSEL_PD13
acmpInputPD14 =
_ACMP_INPUTCTRL_POSSEL_PD14
acmpInputPD15 =
_ACMP_INPUTCTRL_POSSEL_PD15

}

ACMP Input Selection.

Functions

void ACMP_CapsenseInit(ACMP_TypeDef *acmp, const ACMP_CapsenseInit_TypeDef *init)

Set up ACMP for use in capacitive sense applications.

void ACMP_CapsenseChannelSet(ACMP_TypeDef *acmp, ACMP_Channel_TypeDef channel)

Set the ACMP channel used for capacitive sensing.

void ACMP_ChannelSet(ACMP_TypeDef *acmp, ACMP_Channel_TypeDef negSel, ACMP_Channel_TypeDef posSel)

Set which channels should be used in ACMP comparisons.

void ACMP_Disable(ACMP_TypeDef *acmp)

Disable ACMP.

void ACMP_Enable(ACMP_TypeDef *acmp)

Enable ACMP.

void ACMP_GPIOSetup(ACMP_TypeDef *acmp, GPIO_Port_TypeDef port, unsigned int pin, bool enable, bool invert)

Sets up GPIO output from the ACMP.

void ACMP_Init(ACMP_TypeDef *acmp, const ACMP_Init_TypeDef *init)

Initialize ACMP.

void ACMP_Reset(ACMP_TypeDef *acmp)

Reset ACMP to the same state that it was in after a hardware reset.

void ACMP_IntClear(ACMP_TypeDef *acmp, uint32_t flags)

Copyright © 2026 Silicon Laboratories. All rights reserved.

Clear one or more pending ACMP interrupts.

void ACMP_IntDisable(ACMP_TypeDef *acmp, uint32_t flags)
Disable one or more ACMP interrupts.

void ACMP_IntEnable(ACMP_TypeDef *acmp, uint32_t flags)
Enable one or more ACMP interrupts.

uint32_t ACMP_IntGet(ACMP_TypeDef *acmp)
Get pending ACMP interrupt flags.

uint32_t ACMP_IntGetEnabled(ACMP_TypeDef *acmp)
Get enabled and pending ACMP interrupt flags.

void ACMP_IntSet(ACMP_TypeDef *acmp, uint32_t flags)
Set one or more pending ACMP interrupts from software.

ACMP_Channel_TypeDef ACMP_PortPinToInput(GPIO_Port_TypeDef port, uint8_t pin)
Convert GPIO port/pin to ACMP input selection.

bool ACMP_OutputGet(ACMP_TypeDef *acmp)
Get state of ACMP output value.

Macros

#define PM5507_ACMP_CFG_BIAS_DEFAULT 0x00000004UL
A default configuration for capacitive sense mode initialization.

#define PM5507_ACMP_CFG_RESETVALUE 0x00000004UL
Analog comparator reset value.

#define ACMP_CAPSENSE_INIT_DEFAULT undefined
Capacitive sense mode configuration default values.

#define ACMP_INIT_DEFAULT undefined
Default configuration for ACMP regular initialization.

Enumeration Documentation

ACMP_CapsenseResistor_TypeDef

> ACMP_CapsenseResistor_TypeDef

Resistor values used for the internal capacitive sense resistor.

See data sheet for your device for details on each resistor value.

Enumerator	
acmpResistor0	Resistor value 0.
acmpResistor1	Resistor value 1.
acmpResistor2	Resistor value 2.
acmpResistor3	Resistor value 3.
acmpResistor4	Resistor value 4.
acmpResistor5	Resistor value 5.
acmpResistor6	Resistor value 6.

Copyright © 2026 Silicon Laboratories. All rights reserved.

ACMP_HysteresisLevel_TypeDef

```
ACMP_HysteresisLevel_TypeDef
```

Hysteresis level.

See data sheet for your device for details on each level.

Enumerator	
acmpHysteresisDisabled	Mode DISABLED for ACMP_CFG.
acmpHysteresis10Sym	Mode HYST10SYM for ACMP_CFG.
acmpHysteresis20Sym	Mode HYST20SYM for ACMP_CFG.
acmpHysteresis30Sym	Mode HYST30SYM for ACMP_CFG.
acmpHysteresis10Pos	Mode HYST10POS for ACMP_CFG.
acmpHysteresis20Pos	Mode HYST20POS for ACMP_CFG.
acmpHysteresis30Pos	Mode HYST30POS for ACMP_CFG.
acmpHysteresis10Neg	Mode HYST10NEG for ACMP_CFG.
acmpHysteresis20Neg	Mode HYST20NEG for ACMP_CFG.
acmpHysteresis30Neg	Mode HYST30NEG for ACMP_CFG.

ACMP_InputRange_TypeDef

```
ACMP_InputRange_TypeDef
```

Adjust ACMP performance for a given input voltage range.

Enumerator	
acmpInputRangeFull	Input can be from 0 to VDD.
acmpInputRangeReduced	Input can be from 0 to VDD-0.7 V.

ACMP_Accuracy_TypeDef

```
ACMP_Accuracy_TypeDef
```

ACMP accuracy mode.

Enumerator	
acmpAccuracyLow	Low-accuracy mode which consumes less current.
acmpAccuracyHigh	High-accuracy mode which consumes more current.

ACMP_Channel_TypeDef

```
ACMP_Channel_TypeDef
```

ACMP Input Selection.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Enumerator	
acmpInputVSS	Select VSS.
acmpInputVREFDIVAVDD	Select Divided AVDD.
acmpInputVREFDIVAVDDLP	Select Low-Power Divided AVDD.
acmpInputVREFDIV1V25	Select Divided 1V25 reference.
acmpInputVREFDIV1V25LP	Select Low-power Divided 1V25 reference.
acmpInputVREFDIV2V5	Select Divided 2V5 reference.
acmpInputVREFDIV2V5LP	Select Low-power Divided 2V5 reference.
acmpInputVSENSE01DIV4	Select VSENSE0 divided by 4.
acmpInputVSENSE01DIV4LP	Select Low-power VSENSE0 divided by 4.
acmpInputVSENSE11DIV4	VSENSE1 divided by 4.
acmpInputVSENSE11DIV4LP	Low-power VSENSE1 divided by 4.
acmpInputCAPSENSE	Select Low-Power Divided AVDD.
acmpInputVDACOUT0	Select VDAC0 channel 0 output.
acmpInputVDACOUT1	Select VDAC0 channel 1 output.
acmpInputEXTPA	Select external interface, base is PA0.
acmpInputEXTPB	Select external interface, base is PB0.
acmpInputEXTPC	Select external interface, base is PC0.
acmpInputEXTPD	Select external interface, base is PD0.
acmpInputPA0	Select Port A Pin0.
acmpInputPA1	Select Port A Pin1.
acmpInputPA2	Select Port A Pin2.
acmpInputPA3	Select Port A Pin3.
acmpInputPA4	Select Port A Pin4.
acmpInputPA5	Select Port A Pin5.
acmpInputPA6	Select Port A Pin6.
acmpInputPA7	Select Port A Pin7.
acmpInputPA8	Select Port A Pin8.
acmpInputPA9	Select Port A Pin9.
acmpInputPA10	Select Port A Pin10.
acmpInputPA11	Select Port A Pin11.
acmpInputPA12	Select Port A Pin12.
acmpInputPA13	Select Port A Pin13.
acmpInputPA14	Select Port A Pin14.
acmpInputPA15	Select Port A Pin15.
acmpInputPB0	Select Port B Pin0.
acmpInputPB1	Select Port B Pin1.
acmpInputPB2	Select Port B Pin2.
acmpInputPB3	Select Port B Pin3.
acmpInputPB4	Select Port B Pin4.
acmpInputPB5	Select Port B Pin5.
acmpInputPB6	Select Port B Pin6.
acmpInputPB7	Select Port B Pin7.

Copyright © 2026 Silicon Laboratories. All rights reserved.

acmpInputPB8	Select Port B Pin8.
acmpInputPB9	Select Port B Pin9.
acmpInputPB10	Select Port B Pin10.
acmpInputPB11	Select Port B Pin11.
acmpInputPB12	Select Port B Pin12.
acmpInputPB13	Select Port B Pin13.
acmpInputPB14	Select Port B Pin14.
acmpInputPB15	Select Port B Pin15.
acmpInputPC0	Select Port C Pin0.
acmpInputPC1	Select Port C Pin1.
acmpInputPC2	Select Port C Pin2.
acmpInputPC3	Select Port C Pin3.
acmpInputPC4	Select Port C Pin4.
acmpInputPC5	Select Port C Pin5.
acmpInputPC6	Select Port C Pin6.
acmpInputPC7	Select Port C Pin7.
acmpInputPC8	Select Port C Pin8.
acmpInputPC9	Select Port C Pin9.
acmpInputPC10	Select Port C Pin10.
acmpInputPC11	Select Port C Pin11.
acmpInputPC12	Select Port C Pin12.
acmpInputPC13	Select Port C Pin13.
acmpInputPC14	Select Port C Pin14.
acmpInputPC15	Select Port C Pin15.
acmpInputPD0	Select Port D Pin0.
acmpInputPD1	Select Port D Pin1.
acmpInputPD2	Select Port D Pin2.
acmpInputPD3	Select Port D Pin3.
acmpInputPD4	Select Port D Pin4.
acmpInputPD5	Select Port D Pin5.
acmpInputPD6	Select Port D Pin6.
acmpInputPD7	Select Port D Pin7.
acmpInputPD8	Select Port D Pin8.
acmpInputPD9	Select Port D Pin9.
acmpInputPD10	Select Port D Pin10.
acmpInputPD11	Select Port D Pin11.
acmpInputPD12	Select Port D Pin12.
acmpInputPD13	Select Port D Pin13.
acmpInputPD14	Select Port D Pin14.
acmpInputPD15	Select Port D Pin15.

Function Documentation

Copyright © 2026 Silicon Laboratories. All rights reserved.

ACMP_CapsenseInit

```
void ACMP_CapsenseInit (ACMP_TypeDef * acmp, const ACMP_CapsenseInit_TypeDef * init)
```

Set up ACMP for use in capacitive sense applications.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.
const ACMP_CapsenseInit_TypeDef *	[in]	init	A pointer to the initialization structure used to configure ACMP for capacitive sensing operation.

This function sets up ACMP for use in capacitive sense applications. To use the capacitive sense functionality in the ACMP, use the PRS output of the ACMP module to count the number of oscillations in the capacitive sense circuit (possibly using a TIMER).

Note

- A basic example of capacitive sensing can be found in the STK BSP (capsense demo).
- A call to ACMP_CapsenseInit will enable and disable the ACMP peripheral, which can cause side effects if it was previously set up.

ACMP_CapsenseChannelSet

```
void ACMP_CapsenseChannelSet (ACMP_TypeDef * acmp, ACMP_Channel_TypeDef channel)
```

Set the ACMP channel used for capacitive sensing.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.
ACMP_Channel_TypeDef	[in]	channel	The ACMP channel to use for capacitive sensing (Possel).

Note

- A basic example of capacitive sensing can be found in the STK BSP (capsense demo).
- Can only be called when the peripheral is enabled.

ACMP_ChannelSet

```
void ACMP_ChannelSet (ACMP_TypeDef * acmp, ACMP_Channel_TypeDef negSel, ACMP_Channel_TypeDef posSel)
```

Set which channels should be used in ACMP comparisons.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.
ACMP_Channel_TypeDef	N/A	negSel	A channel to use on the negative input to the ACMP.
ACMP_Channel_TypeDef	N/A	posSel	A channel to use on the positive input to the ACMP.

Note

- Can only be called when the peripheral is enabled.
- If GPIO is used for both posSel and negSel, they cannot both use even or odd pins.

ACMP_Disable

> void ACMP_Disable (ACMP_TypeDef * acmp)

Disable ACMP.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.

ACMP_Enable

> void ACMP_Enable (ACMP_TypeDef * acmp)

Enable ACMP.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.

ACMP_GPIOSetup

> void ACMP_GPIOSetup (ACMP_TypeDef * acmp, GPIO_Port_TypeDef port, unsigned int pin, bool enable, bool invert)

Sets up GPIO output from the ACMP.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	Pointer to the ACMP peripheral register block.
GPIO_Port_TypeDef	N/A	port	The GPIO port to use.
unsigned int	N/A	pin	The GPIO pin to use.
bool	N/A	enable	Enable or disable pin output.
bool	N/A	invert	Invert output.

Note

Copyright © 2026 Silicon Laboratories. All rights reserved.

- GPIO must be enabled in the CMU before this function call, i.e.

```
CMU_ClockEnable(cmuClock_GPIO, true);
```

ACMP_Init

```
void ACMP_Init (ACMP_TypeDef * acmp, const ACMP_Init_TypeDef * init)
```

Initialize ACMP.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.
const ACMP_Init_TypeDef *	[in]	init	A pointer to the initialization structure used to configure ACMP.

Note

- A call to ACMP_Init can cause side effects since it can enable/disable the peripheral.

ACMP_Reset

```
void ACMP_Reset (ACMP_TypeDef * acmp)
```

Reset ACMP to the same state that it was in after a hardware reset.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.

Note

- The GPIO ACMP ROUTE register is NOT reset by this function to allow for centralized setup of this feature.
- The peripheral may be enabled and disabled during reset.

ACMP_IntClear

```
void ACMP_IntClear (ACMP_TypeDef * acmp, uint32_t flags)
```

Clear one or more pending ACMP interrupts.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	Pending ACMP interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for the ACMP module. The flags can be, for instance, ACMP_IFC_EDGE or ACMP_IFC_WARMUP.

ACMP_IntDisable

```
void ACMP_IntDisable (ACMP_TypeDef * acmp, uint32_t flags)
```

Disable one or more ACMP interrupts.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.
uint32_t	[in]	flags	ACMP interrupt sources to disable. Use a bitwise logic OR combination of valid interrupt flags for the ACMP module. The flags can be, for instance, ACMP_IEN_EDGE or ACMP_IEN_WARMUP.

ACMP_IntEnable

```
void ACMP_IntEnable (ACMP_TypeDef * acmp, uint32_t flags)
```

Enable one or more ACMP interrupts.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.
uint32_t	[in]	flags	ACMP interrupt sources to enable. Use a bitwise logic OR combination of valid interrupt flags for the ACMP module. The flags can be, for instance, ACMP_IEN_EDGE or ACMP_IEN_WARMUP.

Note

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

ACMP_IntGet

```
uint32_t ACMP_IntGet (ACMP_TypeDef * acmp)
```

Get pending ACMP interrupt flags.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.

Note

- This function does not clear event bits.

Returns

- Pending ACMP interrupt sources. A bitwise logic OR combination of valid interrupt flags for the ACMP module. The pending interrupt sources can be, for instance, ACMP_IF_EDGE or ACMP_IF_WARMUP.

ACMP_IntGetEnabled

```
uint32_t ACMP_IntGetEnabled (ACMP_TypeDef * acmp)
```

Get enabled and pending ACMP interrupt flags.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.

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

Note

- This function does not clear interrupt flags.

Returns

- Pending and enabled ACMP interrupt sources. The return value is the bitwise AND combination of
  - the OR combination of enabled interrupt sources in ACMPx_IEN_nnn register (ACMPx_IEN_nnn) and
  - the OR combination of valid interrupt flags of the ACMP module (ACMPx_IF_nnn).

ACMP_IntSet

```
void ACMP_IntSet (ACMP_TypeDef * acmp, uint32_t flags)
```

Set one or more pending ACMP interrupts from software.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.
uint32_t	[in]	flags	ACMP interrupt sources to set as pending. Use a bitwise logic OR combination of valid interrupt flags for the ACMP module. The flags can be, for instance, ACMP_IFS_EDGE or ACMP_IFS_WARMUP.

ACMP_PortPinToInput

Copyright © 2026 Silicon Laboratories. All rights reserved.

> ACMP_Channel_TypeDef ACMP_PortPinToInput (GPIO_Port_TypeDef port, uint8_t pin)

Convert GPIO port/pin to ACMP input selection.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	GPIO port
uint8_t	[in]	pin	GPIO pin

Returns

- ACMP input selection

ACMP_OutputGet

> bool ACMP_OutputGet (ACMP_TypeDef * acmp)

Get state of ACMP output value.

Parameters

Type	Direction	Argument Name	Description
ACMP_TypeDef *	[in]	acmp	A pointer to the ACMP peripheral register block.

Returns

- State of ACMP output value

Copyright © 2026 Silicon Laboratories. All rights reserved.

ACMP_CapsenseInit_TypeDef

Capsense initialization structure.

Public Attributes

uint32_t	biasProg Bias current.
ACMP_HysteresisLevel_TypeDef	hysteresisLevel Hysteresis level.
ACMP_CapsenseResistor_TypeDef	resistor A resistor used in the capacitive sensing circuit.
uint32_t	vrefDiv VDD division factor.
bool	enable If true, ACMP is enabled after configuration.

Public Attribute Documentation

biasProg

```
uint32_t ACMP_CapsenseInit_TypeDef::biasProg
```

Bias current.

See the ACMP chapter about bias and response time in the reference manual for details.

hysteresisLevel

```
ACMP_HysteresisLevel_TypeDef ACMP_CapsenseInit_TypeDef::hysteresisLevel
```

Hysteresis level.

resistor

```
ACMP_CapsenseResistor_TypeDef ACMP_CapsenseInit_TypeDef::resistor
```

A resistor used in the capacitive sensing circuit.

For values see the device data sheet.

vrefDiv

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
uint32_t ACMP_CapsenseInit_TypeDef::vrefDiv
```

VDD division factor.

VREFOUT = VREFIN * (VREFDIV / 63). Valid values are in the 0-63 range.

enable

```
bool ACMP_CapsenseInit_TypeDef::enable
```

If true, ACMP is enabled after configuration.

Copyright © 2026 Silicon Laboratories. All rights reserved.

ACMP_Init_TypeDef

ACMP initialization structure.

Public Attributes

uint32_t	biasProg Bias current.
ACMP_InputRange_TypeDef	inputRange Input range.
ACMP_Accuracy_TypeDef	accuracy ACMP accuracy mode.
ACMP_HysteresisLevel_TypeDef	hysteresisLevel Hysteresis level.
bool	inactiveValue Inactive value emitted by ACMP during warmup.
uint32_t	vrefDiv VDD division factor.
bool	enable If true, ACMP is enabled after configuration.

Public Attribute Documentation

biasProg

```
uint32_t ACMP_Init_TypeDef::biasProg
```

Bias current.

See the ACMP chapter about bias and response time in the reference manual for details. Valid values are in the range 0-7.

inputRange

```
ACMP_InputRange_TypeDef ACMP_Init_TypeDef::inputRange
```

Input range.

Adjust this setting to optimize the performance for a given input voltage range.

accuracy

```
ACMP_Accuracy_TypeDef ACMP_Init_TypeDef::accuracy
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

ACMP accuracy mode.

Select the accuracy mode that matches the required current usage and accuracy requirement. Low accuracy consumes less current while high accuracy consumes more current.

hysteresisLevel

```
ACMP_HysteresisLevel_TypeDef ACMP_Init_TypeDef::hysteresisLevel
```

Hysteresis level.

inactiveValue

```
bool ACMP_Init_TypeDef::inactiveValue
```

Inactive value emitted by ACMP during warmup.

vrefDiv

```
uint32_t ACMP_Init_TypeDef::vrefDiv
```

VDD division factor.

VREFOUT = VREFIN * (VREFDIV / 63). Valid values are in the 0-63 range.

enable

```
bool ACMP_Init_TypeDef::enable
```

If true, ACMP is enabled after configuration.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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		

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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		

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Returns

- An unshifted and masked register value.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_BUFOUTLeaderInit_TypeDef

CMU_CrystalSharingFollowerInit_TypeDef

CMU_DPLLInit_TypeDef

Enumerations

enum CMU_HFRCODPLLFreq_TypeDef {

cmuHFRCODPLLFreq_1M0Hz = 1000000U
cmuHFRCODPLLFreq_2M0Hz = 2000000U
cmuHFRCODPLLFreq_4M0Hz = 4000000U
cmuHFRCODPLLFreq_7M0Hz = 7000000U
cmuHFRCODPLLFreq_13M0Hz = 13000000U
cmuHFRCODPLLFreq_16M0Hz = 16000000U
cmuHFRCODPLLFreq_19M0Hz = 19000000U
cmuHFRCODPLLFreq_26M0Hz = 26000000U
cmuHFRCODPLLFreq_32M0Hz = 32000000U
cmuHFRCODPLLFreq_38M0Hz = 38000000U
cmuHFRCODPLLFreq_48M0Hz = 48000000U
cmuHFRCODPLLFreq_56M0Hz = 56000000U
cmuHFRCODPLLFreq_64M0Hz = 64000000U
cmuHFRCODPLLFreq_80M0Hz = 80000000U
cmuHFRCODPLLFreq_UserDefined = 0

}

HFRCODPLL frequency bands.

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
enum    CMU_HFRCOEM23Freq_TypeDef {

  cmuHFRCOEM23Freq_1M0Hz = 1000000U
  cmuHFRCOEM23Freq_2M0Hz = 2000000U
  cmuHFRCOEM23Freq_4M0Hz = 4000000U
  cmuHFRCOEM23Freq_13M0Hz = 13000000U
  cmuHFRCOEM23Freq_16M0Hz = 16000000U
  cmuHFRCOEM23Freq_19M0Hz = 19000000U
  cmuHFRCOEM23Freq_26M0Hz = 26000000U
  cmuHFRCOEM23Freq_32M0Hz = 32000000U
  cmuHFRCOEM23Freq_40M0Hz = 40000000U
  cmuHFRCOEM23Freq_5M0Hz = 5000000U
  cmuHFRCOEM23Freq_10M0Hz = 10000000U
  cmuHFRCOEM23Freq_20M0Hz = 20000000U
  cmuHFRCOEM23Freq_UserDefined = 0

}
```

HFRCOEM23 frequency bands.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_EM01GRPCCLK = (CMU_EM01GRPCCLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_EUSART0CLK = (CMU_EUART0CLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_IADCCLK = (CMU_IADCCLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_EM23GRPACLK = (CMU_EM23GRPACLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_WDOG0CLK = (CMU_WDOG0CLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_WDOG1CLK = (CMU_WDOG1CLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_SYSRTCCLK = (CMU_SYSRTCCLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_EM4GRPACLK = (CMU_EM4GRPACLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_DPLLREFCLK = (CMU_DPLLREFCLK_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_LCDCLK = (CMU_LCD_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_VDAC0CLK = (CMU_VDAC0_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_PCNT0CLK = (CMU_PCNT_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_LESENSEHFCLK = (CMU_LESENSEHF_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_LESENSECLK = (CMU_LESENSE_BRANCH << CMU_CLK_BRANCH_POS)
cmuClock_CORE = (CMU_CORE_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_RADIOAES = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_RADIOAES_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_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_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_IADC0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_IADC0_SHIFT << CMU_EN_BIT_POS)
cmuClock_AMUXCP0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_AMUXCP0_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_WDOG1 = (CMU_CLKEN1_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN1_WDOG1_SHIFT << CMU_EN_BIT_POS)
cmuClock_I2C0 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_I2C0_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_HFRCOEM23 = (CMU_CLKEN0_EN_REG << CMU_EN_REG_POS) | (_CMU_CLKEN0_HFRCOEM23_SHIFT << CMU_EN_BIT_POS)

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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_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_DCDC =
(CMU_CLKEN0_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN0_DCDC_SHIFT
<< CMU_EN_BIT_POS)
cmuClock_SYSRTC =
(CMU_CLKEN0_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN0_SYSRTC0_SHIFT
<< CMU_EN_BIT_POS)
cmuClock_EUSART0 =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_EUSART0_SHIFT
<< CMU_EN_BIT_POS)
cmuClock_EUSART1 =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_EUSART1_SHIFT
<< CMU_EN_BIT_POS)
cmuClock_EUSART2 =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
(_CMU_CLKEN1_EUSART2_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_SEMAILBOX =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_SEMAILBOXHOST_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_LESENSE =
(CMU_CLKEN0_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN0_LESENSE_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_ACMP0 =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_ACMP0_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_ACMP1 =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_ACMP1_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_VDAC0 =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_VDAC0_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_PCNT0 =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_PCNT0_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_DMEM =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_DMEM_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_KEYSCAN =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_KEYSCAN_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_LCD =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_LCD_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_MVP =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_MVP_SHIFT <<
CMU_EN_BIT_POS)
cmuClock_MSC =
(CMU_CLKEN1_EN_REG <<
CMU_EN_REG_POS) |
(_CMU_CLKEN1_MSC_SHIFT <<
CMU_EN_BIT_POS)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

}
Clock points in CMU clock-tree.

enum CMU_Osc_TypeDef {

cmuOsc_LFXO
cmuOsc_LFRCO
cmuOsc_FSRCO
cmuOsc_HFXO
cmuOsc_HFRCODPLL
cmuOsc_HFRCOEM23
cmuOsc_ULFRCO

}
Oscillator types.

enum CMU_Select_TypeDef {

cmuSelect_Error
cmuSelect_Disabled
cmuSelect_FSRCO
cmuSelect_HFXO
cmuSelect_HFXORT
cmuSelect_HFRCODPLL
cmuSelect_HFRCODPLLRT
cmuSelect_HFRCOEM23
cmuSelect_CLKIN0
cmuSelect_LFXO
cmuSelect_LFRCO
cmuSelect_ULFRCO
cmuSelect_HCLK
cmuSelect_SYSCLK
cmuSelect_HCLKDIV1024
cmuSelect_EM01GRPACLK
cmuSelect_EM23GRPACLK
cmuSelect_EM01GRPCCLK
cmuSelect_EXPCLK
cmuSelect_PRS
cmuSelect_PCNTEXTCLK
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
cmuHfxoOscMode_ExternalSinePkDet = _HFXO_CFG_MODE_EXTCLKPKDET

}

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

}

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum CMU_BufoutTimeoutStartup_TypeDef {

startupTimeout42Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T42US
startupTimeout83Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T83US
startupTimeout108Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T108US
startupTimeout133Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T133US
startupTimeout158Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T158US
startupTimeout183Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T183US
startupTimeout208Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T208US
startupTimeout233Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T233US
startupTimeout258Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T258US
startupTimeout283Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T283US
startupTimeout333Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T333US
startupTimeout375Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T375US
startupTimeout417Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T417US
startupTimeout458Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T458US
startupTimeout500Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T500US
startupTimeout667Us = _HFXO_BUFOUTCTRL_TIMEOUTSTARTUP_T667US

}
Crystal sharing timeout start up timeout.

enum CMU_PRS_Status_Output_Select_TypeDef {

PRS_Status_select_0
PRS_Status_select_1

}
PRS status select output signal.

Typedefs

typedef uint32_t CMU_ClkDiv_TypeDef
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
void	CMU_HFXOStartCrystalSharingLeader(const CMU_BUFOUTLeaderInit_TypeDef *bufoutInit, GPIO_Port_TypeDef port, unsigned int pin) Initialize HFXO Bufout (Crystal sharing) leader control registers.
void	CMU_HFXOCrystalSharingFollowerInit(CMU_PRS_Status_Output_Select_TypeDef prsStatusSelectOutput, unsigned int prsAsyncCh, GPIO_Port_TypeDef port, unsigned int pin) Initialize HFXO Bufout (Crystal sharing) follower 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.
int32_t	CMU_HFXOCTuneCurrentDeltaGet(void) Get the current delta between HFXO XI and XO.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
CMU_HFRCOEM23Freq_TypeDef	CMU_HFRCOEM23BandGet(void) Get HFRCOEM23 band in use.
void	CMU_HFRCOEM23BandSet(CMU_HFRCOEM23Freq_TypeDef freq) Set HFRCOEM23 band and the tuning value based on the value in the calibration table made during production.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_EUSART0CLKCTRL_CLKSEL_DISABLED 0x00000000UL
Mode DISABLED for CMU_EUSART0CLKCTRL

#define CMU_EUSART0CLKCTRL_CLKSEL_DISABLED (_CMU_EUSART0CLKCTRL_CLKSEL_DISABLED << 0)
Shifted mode DISABLED for CMU_EUSART0CLKCTRL.

#define _CMU_SYSRTC0CLKCTRL_CLKSEL_DISABLED 0x00000000UL
Mode DISABLED for CMU_SYSRTC0CLKCTRL

Copyright © 2026 Silicon Laboratories. All rights reserved.

#define CMU_SYSRTC0CLKCTRL_CLKSEL_DISABLED (_CMU_SYSRTC0CLKCTRL_CLKSEL_DISABLED << 0)
Shifted mode DISABLED for CMU_SYSRTC0CLKCTRL.

#define CMU_HFRCODPLL_MIN cmuHFRCODPLLFreq_1M0Hz
HFRCODPLL maximum frequency.

#define CMU_HFRCODPLL_MAX cmuHFRCODPLLFreq_80M0Hz
HFRCODPLL minimum frequency.

#define CMU_HFRCOEM23_MIN cmuHFRCOEM23Freq_1M0Hz
HFRCOEM23 maximum frequency.

#define CMU_HFRCOEM23_MAX cmuHFRCOEM23Freq_40M0Hz
HFRCOEM23 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_CTUNEFIXANA_DEFAULT cmuHfxoCtuneFixCap_Xo
Default configuration of fixed tuning capacitance on XI or XO for EFR32XG23 and EFR32XG28.

#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_HFXO_CRYSTAL_INIT_LEADER_DEFAULT undefined
Default crystal sharing master initialization values.

#define CMU_HFXO_CRYSTAL_INIT_Follower_DEFAULT undefined
Default crystal sharing follower initialization values.

#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

Copyright © 2026 Silicon Laboratories. All rights reserved.

HFRCODPLL frequency bands.

Enumerator	
cmuHFRCODPLLFreq_1M0Hz	1MHz RC band.
cmuHFRCODPLLFreq_2M0Hz	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_HFRCOEM23Freq_TypeDef

> CMU_HFRCOEM23Freq_TypeDef

HFRCOEM23 frequency bands.

Enumerator	
cmuHFRCOEM23Freq_1M0Hz	1MHz RC band.
cmuHFRCOEM23Freq_2M0Hz	2MHz RC band.
cmuHFRCOEM23Freq_4M0Hz	4MHz RC band.
cmuHFRCOEM23Freq_13M0Hz	13MHz RC band.
cmuHFRCOEM23Freq_16M0Hz	16MHz RC band.
cmuHFRCOEM23Freq_19M0Hz	19MHz RC band.
cmuHFRCOEM23Freq_26M0Hz	26MHz RC band.
cmuHFRCOEM23Freq_32M0Hz	32MHz RC band.
cmuHFRCOEM23Freq_40M0Hz	40MHz RC band.
cmuHFRCOEM23Freq_5M0Hz	5MHz RC band.
cmuHFRCOEM23Freq_10M0Hz	10MHz RC band.
cmuHFRCOEM23Freq_20M0Hz	20MHz RC band.
cmuHFRCOEM23Freq_UserDefined	

CMU_Clock_TypeDef

> CMU_Clock_TypeDef

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_EM01GRPCCLK	EM01GRPC clock.
cmuClock_EUSART0CLK	EUSART0 clock.
cmuClock_IADCCLK	IADC clock.
cmuClock_EM23GRPACLK	EM23GRPA clock.
cmuClock_WDOG0CLK	WDOG0 clock.
cmuClock_WDOG1CLK	WDOG1 clock.
cmuClock_SYSRTCCLK	SYSRTC clock.
cmuClock_EM4GRPACLK	EM4GRPA clock.
cmuClock_DPLLREFCLK	DPLLREF clock.
cmuClock_LCDCLK	LCD clock.
cmuClock_VDAC0CLK	VDAC0 clock.
cmuClock_PCNT0CLK	PCNT0 clock.
cmuClock_LESENSEHFCLK	LESENSE high frequency clock.
cmuClock_LESENSECLK	LESENSE low frequency clock.
cmuClock_CORE	Cortex-M33 core clock.
cmuClock_LDMA	LDMA clock.
cmuClock_LDMAXBAR	LDMAXBAR clock.
cmuClock_RADIOAES	RADIOAES 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_IADC0	IADC0 clock.
cmuClock_AMUXCP0	AMUXCP0 clock.
cmuClock_LETIMER0	LETIMER0 clock.
cmuClock_WDOG0	WDOG0 clock.
cmuClock_WDOG1	WDOG1 clock.
cmuClock_I2C0	I2C0 clock.
cmuClock_I2C1	I2C1 clock.

Copyright © 2026 Silicon Laboratories. All rights reserved.

cmuClock_SYSCFG	SYSCFG clock.
cmuClock_DPLL0	DPLL0 clock.
cmuClock_HFRCO0	HFRCO0 clock.
cmuClock_HFRCOEM23	HFRCOEM23 clock.
cmuClock_HFXO	HFXO clock.
cmuClock_FSRCO	FSRCO clock.
cmuClock_LFRCO	LFRCO clock.
cmuClock_LFXO	LFXO clock.
cmuClock_ULFRCO	ULFRCO clock.
cmuClock_GPIO	GPIO clock.
cmuClock_PRS	PRS clock.
cmuClock_BURAM	BURAM clock.
cmuClock_BURTC	BURTC clock.
cmuClock_DCDC	DCDC clock.
cmuClock_SYSRTC	SYSRTC clock.
cmuClock_EUSART0	EUSART0 clock.
cmuClock_EUSART1	EUSART1 clock.
cmuClock_EUSART2	EUSART2 clock.
cmuClock_SEMAILBOX	SEMAILBOX clock.
cmuClock_SMU	SMU clock.
cmuClock_ICACHE	ICACHE clock.
cmuClock_LESENSE	LESENSE clock.
cmuClock_ACMP0	ACMP0 clock.
cmuClock_ACMP1	ACMP1 clock.
cmuClock_VDAC0	VDAC0 clock.
cmuClock_PCNT0	PCNT0 clock.
cmuClock_DMEM	DMEM clock.
cmuClock_KEYSCAN	KEYSCAN clock.
cmuClock_LCD	LCD clock.
cmuClock_MVP	MVP 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

cmuOsc_HFRCODPLL	High frequency RC and DPLL oscillator.
cmuOsc_HFRCOEM23	High frequency deep sleep RC 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_HFRCOEM23	High frequency deep sleep RC oscillator.
cmuSelect_CLKIN0	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_EM01GRPCCLK	EM01GRPC clock.
cmuSelect_EXPCLK	Pin export clock.
cmuSelect_PRS	PRS input as clock.
cmuSelect_PCNTEXTCLK	Pulse counter external source or PRS 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

HFXO oscillator modes.

Enumerator	
cmuHfxoOscMode_Crystal	Crystal oscillator.
cmuHfxoOscMode_ExternalSine	External digital clock.
cmuHfxoOscMode_ExternalSinePkDet	External digital clock with peak detector used.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

CMU_BufoutTimeoutStartup_TypeDef

```
CMU_BufoutTimeoutStartup_TypeDef
```

Crystal sharing timeout start up timeout.

Enumerator	
startupTimeout42Us	Timeout set to 42 us.
startupTimeout83Us	Timeout set to 83 us.
startupTimeout108Us	Timeout set to 108 us.
startupTimeout133Us	Timeout set to 133 us.
startupTimeout158Us	Timeout set to 158 us.
startupTimeout183Us	Timeout set to 183 us.
startupTimeout208Us	Timeout set to 208 us.
startupTimeout233Us	Timeout set to 233 us.
startupTimeout258Us	Timeout set to 258 us.
startupTimeout283Us	Timeout set to 283 us.
startupTimeout333Us	Timeout set to 333 us.
startupTimeout375Us	Timeout set to 375 us.
startupTimeout417Us	Timeout set to 417 us.
startupTimeout458Us	Timeout set to 458 us.
startupTimeout500Us	Timeout set to 500 us.
startupTimeout667Us	Timeout set to 667 us.

CMU_PRS_Status_Output_Select_TypeDef

```
CMU_PRS_Status_Output_Select_TypeDef
```

PRS status select output signal.

Enumerator	
PRS_Status_select_0	PRS status 0 output signal.
PRS_Status_select_1	PRS status 1 output signal.

Typedef Documentation

CMU_ClkDiv_TypeDef

```
typedef uint32_t CMU_ClkDiv_TypeDef
```

Clock divider configuration.

Function Documentation

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 againt 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 sre 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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: Fout = Fref * (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 <= 4095
  0 <= M <= 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_HFXOStartCrystalSharingLeader

```
void CMU_HFXOStartCrystalSharingLeader (const CMU_BUFOUTLeaderInit_TypeDef * bufoutInit,
GPIO_Port_TypeDef port, unsigned int pin)
```

Initialize HFXO Bufout (Crystal sharing) leader control registers.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
Type	Direction	Argument Name	Description
const CMU_BUFOUTLeaderInit_TypeDef *	[in]	bufoutInit	Bufout setup parameters.
GPIO_Port_TypeDef	[in]	port	Bufout request GPIO port.
unsigned int	[in]	pin	Bufout request GPIO pin.

Configure the bufout request input GPIO as a clock request signal to add the crystal sharing follower chip as a source of clock request.

Warnings

- If EM2 capabilities are needed, a GPIO that fully retains its capabilities while in EM2 must be selected.

CMU_HFXOCrystalSharingFollowerInit

> void CMU_HFXOCrystalSharingFollowerInit (CMU_PRS_Status_Output_Select_TypeDef prsStatusSelectOutput, unsigned int prsAsyncCh, GPIO_Port_TypeDef port, unsigned int pin)

Initialize HFXO Bufout (Crystal sharing) follower control registers.

Parameters

Type	Direction	Argument Name	Description
CMU_PRS_Status_Output_Select_TypeDef	[in]	prsStatusSelectOutput	Selected HFXO PRS signal output.
unsigned int	[in]	prsAsyncCh	PRS producer asynchronous signal channel.
GPIO_Port_TypeDef	[in]	port	Bufout request GPIO port.
unsigned int	[in]	pin	Bufout request GPIO pin.

Configure the clock request signal to a specified GPIO to automatically request the high frequency crystal oscillator sine wave clock. This function must be used in conjunction with CMU_HFXOInit() configured with EXTERNAL_SINE or EXTERNAL_SINEPKDET mode.

Warnings

- If EM2 capabilities are needed, a GPIO that fully retains its capabilities while in EM2 must be selected.

Note

- This function can be emulated on XG21/XG22 chips by controlling the clock request GPIO to ask the crystal sharing leader clock when needed.

CMU_HFXOCTuneSet

> sl_status_t CMU_HFXOCTuneSet (uint32_t ctune)

Set the HFXO crystal tuning capacitance.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_HFXOCTuneCurrentDeltaGet 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. Neither XI nor XO is actually modified by this function, CMU_HFXOCTuneSet() needs to be called for the change to be propagated.

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
int32_t CMU_HFXOCTuneDeltaGet (void )
```

Get the HFXO crystal tuning delta.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

It can be default recommended delta value or value set by CMU_HFXOCTuneDeltaSet.

Returns

- Chip dependent crystal capacitor bank tuning delta.

CMU_HFXOCTuneCurrentDeltaGet

```
int32_t CMU_HFXOCTuneCurrentDeltaGet (void )
```

Get the current delta between HFXO XI and XO.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- the current delta between HFXO XI and XO.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_HFRCOEM23BandGet

```
CMU_HFRCOEM23Freq_TypeDef CMU_HFRCOEM23BandGet (void )
```

Get HFRCOEM23 band in use.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- HFRCOEM23 band in use.

CMU_HFRCOEM23BandSet

```
void CMU_HFRCOEM23BandSet (CMU_HFRCOEM23Freq_TypeDef freq)
```

Set HFRCOEM23 band and the tuning value based on the value in the calibration table made during production.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
CMU_HFRCOEM23Freq_TypeDef	[in]	freq	HFRCOEM23 frequency band to activate.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 `enable` 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)

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_LfxoStartupDelay_TypeDef	timeout Startup delay.
CMU_LfxoOscMode_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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

bool disOnDemand
Disable on-demand requests.

bool forceEn
Force oscillator enable.

bool em23OnDemand
Enable deep sleep.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

em23OnDemand

```
bool CMU_HFXOInit_TypeDef::em23OnDemand
```

Enable deep sleep.

Copyright © 2026 Silicon Laboratories. All rights reserved.

regLock

```
bool CMU_HFXOInit_TypeDef::regLock
```

Lock register access.

Copyright © 2026 Silicon Laboratories. All rights reserved.

CMU_BUFOUTLeaderInit_TypeDef

Crystal sharing leader initialization structure.

Public Attributes

bool	minimalStartupDelay If enabled, bufout won't start until timeout expires.
CMU_BufoutTimeoutStartup_TypeDef	timeoutStartup Wait duration of the oscillator startup sequence to prevent bufout starting too early.

Public Attribute Documentation

minimalStartupDelay

```
bool CMU_BUFOUTLeaderInit_TypeDef::minimalStartupDelay
```

If enabled, bufout won't start until timeout expires.

timeoutStartup

```
CMU_BufoutTimeoutStartup_TypeDef CMU_BUFOUTLeaderInit_TypeDef::timeoutStartup
```

Wait duration of the oscillator startup sequence to prevent bufout starting too early.

Copyright © 2026 Silicon Laboratories. All rights reserved.

CMU_CrystalSharingFollowerInit_TypeDef

Crystal sharing follower initialization structure.

Public Attributes

CMU_PRS_Status_Output_Select_TypeDef	prsStatusSelectOutput PRS status output select.
bool	em23OnDemand Enable em23 on demand.
bool	regLock Lock registers.

Public Attribute Documentation

prsStatusSelectOutput

```
CMU_PRS_Status_Output_Select_TypeDef CMU_CrystalSharingFollowerInit_TypeDef::prsStatusSelectOutput
```

PRS status output select.

em23OnDemand

```
bool CMU_CrystalSharingFollowerInit_TypeDef::em23OnDemand
```

Enable em23 on demand.

regLock

```
bool CMU_CrystalSharingFollowerInit_TypeDef::regLock
```

Lock registers.

Copyright © 2026 Silicon Laboratories. All rights reserved.

CMU_DPLLInit_TypeDef

DPLL initialization structure.

Frequency will be Fref*(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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_EM01Init_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 {

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
  emuVreginCmpThreshold_2v0
  = 0
  emuVreginCmpThreshold_2v1
  = 1
  emuVreginCmpThreshold_2v2
  = 2
  emuVreginCmpThreshold_2v3
  = 3
}
```

VREGIN comparator threshold.

```
enum    EMU_DcdcTonMaxTimeout_TypeDef {
  emuDcdcTonMaxTimeout_Off = 0
  emuDcdcTonMaxTimeout_0P14us = 1
  emuDcdcTonMaxTimeout_0P21us = 2
  emuDcdcTonMaxTimeout_0P28us = 3
  emuDcdcTonMaxTimeout_0P35us = 4
  emuDcdcTonMaxTimeout_0P42us = 5
  emuDcdcTonMaxTimeout_0P49us = 6
  emuDcdcTonMaxTimeout_0P56us = 7
  emuDcdcTonMaxTimeout_0P63us = 8
  emuDcdcTonMaxTimeout_0P70us = 9
  emuDcdcTonMaxTimeout_0P77us = 10
  emuDcdcTonMaxTimeout_0P84us = 11
  emuDcdcTonMaxTimeout_0P91us = 12
  emuDcdcTonMaxTimeout_0P98us = 13
  emuDcdcTonMaxTimeout_1P05us = 14
  emuDcdcTonMaxTimeout_1P12us = 15
  emuDcdcTonMaxTimeout_1P19us = 16
  emuDcdcTonMaxTimeout_1P26us = 17
  emuDcdcTonMaxTimeout_1P33us = 18
  emuDcdcTonMaxTimeout_1P40us = 19
  emuDcdcTonMaxTimeout_1P47us = 20
  emuDcdcTonMaxTimeout_1P54us = 21
  emuDcdcTonMaxTimeout_1P61us = 22
  emuDcdcTonMaxTimeout_1P68us = 23
  emuDcdcTonMaxTimeout_1P75us = 24
  emuDcdcTonMaxTimeout_1P82us = 25
  emuDcdcTonMaxTimeout_1P89us = 26
  emuDcdcTonMaxTimeout_1P96us = 27
  emuDcdcTonMaxTimeout_2P03us = 28
  emuDcdcTonMaxTimeout_2P10us = 29
  emuDcdcTonMaxTimeout_2P17us = 30
  emuDcdcTonMaxTimeout_2P24us = 31
}
```

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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).

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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_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.

void EMU_DCDCSetPFMXModePeakCurrent(uint32_t value)
Set PFMX mode Peak Current setting.

Copyright © 2026 Silicon Laboratories. All rights reserved.

void	EMU_DCDCSetPFMXTimeoutMaxCtrl(EMU_DcdcTonMaxTimeout_TypeDef value) Set Ton_max timeout control.
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_DCDCSync(uint32_t mask) Wait for the DCDC to complete all synchronization of register changes.
void	EMU_CallWFI(void) Executes WFI with required memory barriers.
void	EMU_EnterEM1(void) Enter energy mode 1 (EM1).
void	EMU_VScaleWait(void) Wait for voltage scaling to complete.
EMU_VScaleEM01_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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_0P14us	Ton max is 0.14us.
emuDcdcTonMaxTimeout_0P21us	Ton max is 0.21us.
emuDcdcTonMaxTimeout_0P28us	Ton max is 0.28us.
emuDcdcTonMaxTimeout_0P35us	Ton max is 0.35us.
emuDcdcTonMaxTimeout_0P42us	Ton max is 0.42us.
emuDcdcTonMaxTimeout_0P49us	Ton max is 0.49us.
emuDcdcTonMaxTimeout_0P56us	Ton max is 0.56us.
emuDcdcTonMaxTimeout_0P63us	Ton max is 0.63us.
emuDcdcTonMaxTimeout_0P70us	Ton max is 0.70us.
emuDcdcTonMaxTimeout_0P77us	Ton max is 0.77us.
emuDcdcTonMaxTimeout_0P84us	Ton max is 0.84us.
emuDcdcTonMaxTimeout_0P91us	Ton max is 0.91us.
emuDcdcTonMaxTimeout_0P98us	Ton max is 0.98us.
emuDcdcTonMaxTimeout_1P05us	Ton max is 1.05us.
emuDcdcTonMaxTimeout_1P12us	Ton max is 1.12us.
emuDcdcTonMaxTimeout_1P19us	Ton max is 1.19us.
emuDcdcTonMaxTimeout_1P26us	Ton max is 1.26us.
emuDcdcTonMaxTimeout_1P33us	Ton max is 1.33us.
emuDcdcTonMaxTimeout_1P40us	Ton max is 1.40us.

Copyright © 2026 Silicon Laboratories. All rights reserved.

emuDcdcTonMaxTimeout_1P47us	Ton max is 1.47us.
emuDcdcTonMaxTimeout_1P54us	Ton max is 1.54us.
emuDcdcTonMaxTimeout_1P61us	Ton max is 1.61us.
emuDcdcTonMaxTimeout_1P68us	Ton max is 1.68us.
emuDcdcTonMaxTimeout_1P75us	Ton max is 1.75us.
emuDcdcTonMaxTimeout_1P82us	Ton max is 1.82us.
emuDcdcTonMaxTimeout_1P89us	Ton max is 1.89us.
emuDcdcTonMaxTimeout_1P96us	Ton max is 1.96us.
emuDcdcTonMaxTimeout_2P03us	Ton max is 2.03us.
emuDcdcTonMaxTimeout_2P10us	Ton max is 2.10us.
emuDcdcTonMaxTimeout_2P17us	Ton max is 2.17us.
emuDcdcTonMaxTimeout_2P24us	Ton max is 2.24us.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 )
```

EFP'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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
-

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
-

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Note

- Only a power on reset or external reset pin can wake the device from EM4. Device which is configured in Boost DC-DC mode can not enter 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		

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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. Except for xg27, It can only have two values: 0 or more than RAM0_END. Any other valid RAM address will just do nothing without any error or indication that nothing happened. 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 power down blocks from start to the end of RAM. For xg27, it will shut off 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

- The specified memory block(s) will stay off until a call to EMU_RamPowerUp() is done.

EMU_RamPowerUp

```
void EMU_RamPowerUp (void )
```

Power up all available RAM memory blocks.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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

-

Copyright © 2026 Silicon Laboratories. All rights reserved.

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. At least 200uS delay is required before voltage scaling for Lion only.
- 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 EMO1 mode Peak Current setting.

Parameters

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

EMU_DCDCSetPFMXModePeakCurrent

void EMU_DCDCSetPFMXModePeakCurrent (uint32_t value)

Set PFMX mode Peak Current setting.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	value	Peak load current coefficient in PFMX mode.

EMU_DCDCSetPFMXTimeoutMaxCtrl

void EMU_DCDCSetPFMXTimeoutMaxCtrl (EMU_DcdcTonMaxTimeout_TypeDef value)

Set Ton_max timeout control.

Parameters

Type	Direction	Argument Name	Description
EMU_DcdcTonMaxTimeout_TypeDef	[in]	value	Maximum time for peak current detection.

EMU_TemperatureGet

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_DCDCSync

```
void EMU_DCDCSync (uint32_t mask)
```

Wait for the DCDC to complete all synchronization of register changes.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	mask	A bitmask corresponding to SYNCBUSY register defined bits indicating registers that must complete any ongoing synchronization.

EMU_CallWFI

```
void EMU_CallWFI (void )
```

Executes WFI with required memory barriers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Copyright © 2026 Silicon Laboratories. All rights reserved.

Note

- Calls __DSB() and __ISB() before __WFI() to ensure instruction and data synchronization before entering sleep mode.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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).

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

EMU_EM4Init_TypeDef

EM4 initialization structure.

Public Attributes

bool	retainLfxo 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

retainLfxo

```
bool EMU_EM4Init_TypeDef::retainLfxo
```

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Hibernate or shutoff EM4 state.

pinRetentionMode

```
EMU_EM4PinRetention_TypeDef EMU_EM4Init_TypeDef::pinRetentionMode
```

EM4 pin retention mode.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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:

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
{
 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 theses 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

EUSART_UartInit_TypeDef

EUSART_IrDAInit_TypeDef

EUSART_PrsTriggerInit_TypeDef

EUSART_SpiAdvancedInit_TypeDef

EUSART_SpiInit_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
eusartDataBits10 = EUSART_FRAMECFG_DATABITS_TEN
eusartDataBits11 = EUSART_FRAMECFG_DATABITS_ELEVEN
eusartDataBits12 = EUSART_FRAMECFG_DATABITS_TWELVE
eusartDataBits13 = EUSART_FRAMECFG_DATABITS_THIRTEEN
eusartDataBits14 = EUSART_FRAMECFG_DATABITS_FOURTEEN
eusartDataBits15 = EUSART_FRAMECFG_DATABITS_FIFTEEN
eusartDataBits16 = EUSART_FRAMECFG_DATABITS_SIXTEEN
```

}

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 {

```
eusartStopbits0p5 = EUSART_FRAMECFG_STOPBITS_HALF
eusartStopbits1p5 = EUSART_FRAMECFG_STOPBITS_ONEANDAHALF
eusartStopbits1 = EUSART_FRAMECFG_STOPBITS_ONE
eusartStopbits2 = EUSART_FRAMECFG_STOPBITS_TWO
```

}

Stop bits selection.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum EUSART_PrsTriggerEnable_TypeDef {

eusartPrsTriggerDisable = 0x0
eusartPrsTriggerEnableRx = EUSART_TRIGCTRL_RXTEN
eusartPrsTriggerEnableTx = EUSART_TRIGCTRL_TXTEN
eusartPrsTriggerEnableRxTx = (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
eusartRxFiFoWatermark5Frame = EUSART_CFG1_RXFIW_FIVEFRAMES
eusartRxFiFoWatermark6Frame = EUSART_CFG1_RXFIW_SIXFRAMES
eusartRxFiFoWatermark7Frame = EUSART_CFG1_RXFIW_SEVENFRAMES
eusartRxFiFoWatermark8Frame = EUSART_CFG1_RXFIW_EIGHTFRAMES
eusartRxFiFoWatermark9Frame = EUSART_CFG1_RXFIW_NINEFRAMES
eusartRxFiFoWatermark10Frame = EUSART_CFG1_RXFIW_TENFRAMES
eusartRxFiFoWatermark11Frame = EUSART_CFG1_RXFIW_ELEVENFRAMES
eusartRxFiFoWatermark12Frame = EUSART_CFG1_RXFIW_TWELVEFRAMES
eusartRxFiFoWatermark13Frame = EUSART_CFG1_RXFIW_THIRTEENFRAMES
eusartRxFiFoWatermark14Frame = EUSART_CFG1_RXFIW_FOURTEENFRAMES
eusartRxFiFoWatermark15Frame = EUSART_CFG1_RXFIW_FIFTEENFRAMES
eusartRxFiFoWatermark16Frame = EUSART_CFG1_RXFIW_SIXTEENFRAMES

}
RX FIFO Interrupt ans Status Watermark.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum EUSART_TxFifoWatermark_TypeDef {

eusartTxFiFoWatermark1Frame = EUSART_CFG1_TXFIW_ONEFRAME
eusartTxFiFoWatermark2Frame = EUSART_CFG1_TXFIW_TWOFRAMES
eusartTxFiFoWatermark3Frame = EUSART_CFG1_TXFIW_THREEFRAMES
eusartTxFiFoWatermark4Frame = EUSART_CFG1_TXFIW_FOURFRAMES
eusartTxFiFoWatermark5Frame = EUSART_CFG1_TXFIW_FIVEFRAMES
eusartTxFiFoWatermark6Frame = EUSART_CFG1_TXFIW_SIXFRAMES
eusartTxFiFoWatermark7Frame = EUSART_CFG1_TXFIW_SEVENFRAMES
eusartTxFiFoWatermark8Frame = EUSART_CFG1_TXFIW_EIGHTFRAMES
eusartTxFiFoWatermark9Frame = EUSART_CFG1_TXFIW_NINEFRAMES
eusartTxFiFoWatermark10Frame = EUSART_CFG1_TXFIW_TENFRAMES
eusartTxFiFoWatermark11Frame = EUSART_CFG1_TXFIW_ELEVENFRAMES
eusartTxFiFoWatermark12Frame = EUSART_CFG1_TXFIW_TWELVEFRAMES
eusartTxFiFoWatermark13Frame = EUSART_CFG1_TXFIW_THIRTEENFRAMES
eusartTxFiFoWatermark14Frame = EUSART_CFG1_TXFIW_FOURTEENFRAMES
eusartTxFiFoWatermark15Frame = EUSART_CFG1_TXFIW_FIFTEENFRAMES
eusartTxFiFoWatermark16Frame = EUSART_CFG1_TXFIW_SIXTEENFRAMES

}
TX FIFO Interrupt and Status Watermark.

enum EUSART_ClockMode_TypeDef {

eusartClockMode0 = EUSART_CFG2_CLKPOL_IDLELOW | EUSART_CFG2_CLKPHA_SAMPLELEADING
eusartClockMode1 = EUSART_CFG2_CLKPOL_IDLELOW | EUSART_CFG2_CLKPHA_SAMPLETRAILING
eusartClockMode2 = EUSART_CFG2_CLKPOL_IDLEHIGH | EUSART_CFG2_CLKPHA_SAMPLELEADING
eusartClockMode3 = EUSART_CFG2_CLKPOL_IDLEHIGH | EUSART_CFG2_CLKPHA_SAMPLETRAILING

}
Clock polarity/phase mode.

enum EUSART_CsPolarity_TypeDef {

eusartCsActiveLow = EUSART_CFG2_CSINV_AL
eusartCsActiveHigh = EUSART_CFG2_CSINV_AH

}
Chip select polarity.

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_SpiInit(EUSART_TypeDef *eusart, const EUSART_SpiInit_TypeDef *init)
Initialize EUSART when used in SPI mode.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
uint16_t	EUSART_Spi_TxRx(EUSART_TypeDef *eusart, uint16_t data) Transmit one 8-16 bit frame and return received data.
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 16 Define EUSART FIFO Depth information.
#define	EUSART1_FIFO_DEPTH EUSART0_FIFO_DEPTH

Copyright © 2026 Silicon Laboratories. All rights reserved.

#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.

#define EUSART_SPI_ADVANCED_INIT_DEFAULT undefined
Default advanced configuration for EUSART initialization structure in SPI mode with high-frequency clock.

#define EUSART_SPI_MASTER_INIT_DEFAULT_HF undefined
Default configuration for EUSART initialization structure in SPI master mode with high-frequency clock.

#define EUSART_SPI_SLAVE_INIT_DEFAULT_HF undefined
Default configuration for EUSART initialization structure in SPI slave mode with high-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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Enumerator	
eusartDataBits7	7 data bits.
eusartDataBits8	8 data bits.
eusartDataBits9	9 data bits.
eusartDataBits10	10 data bits, SPI mode only.
eusartDataBits11	11 data bits, SPI mode only.
eusartDataBits12	12 data bits, SPI mode only.
eusartDataBits13	13 data bits, SPI mode only.
eusartDataBits14	14 data bits, SPI mode only.
eusartDataBits15	15 data bits, SPI mode only.
eusartDataBits16	16 data bits, SPI mode only.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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	
eusartRxFiFoWatermark5Frame	
eusartRxFiFoWatermark6Frame	
eusartRxFiFoWatermark7Frame	
eusartRxFiFoWatermark8Frame	
eusartRxFiFoWatermark9Frame	
eusartRxFiFoWatermark10Frame	
eusartRxFiFoWatermark11Frame	
eusartRxFiFoWatermark12Frame	
eusartRxFiFoWatermark13Frame	
eusartRxFiFoWatermark14Frame	
eusartRxFiFoWatermark15Frame	
eusartRxFiFoWatermark16Frame	

Copyright © 2026 Silicon Laboratories. All rights reserved.

EUSART_TxFifoWatermark_TypeDef

> EUSART_TxFifoWatermark_TypeDef

TX FIFO Interrupt and Status Watermark.

Enumerator	
eusartTxFiFoWatermark1Frame	
eusartTxFiFoWatermark2Frame	
eusartTxFiFoWatermark3Frame	
eusartTxFiFoWatermark4Frame	
eusartTxFiFoWatermark5Frame	
eusartTxFiFoWatermark6Frame	
eusartTxFiFoWatermark7Frame	
eusartTxFiFoWatermark8Frame	
eusartTxFiFoWatermark9Frame	
eusartTxFiFoWatermark10Frame	
eusartTxFiFoWatermark11Frame	
eusartTxFiFoWatermark12Frame	
eusartTxFiFoWatermark13Frame	
eusartTxFiFoWatermark14Frame	
eusartTxFiFoWatermark15Frame	
eusartTxFiFoWatermark16Frame	

EUSART_ClockMode_TypeDef

> EUSART_ClockMode_TypeDef

Clock polarity/phase mode.

Enumerator	
eusartClockMode0	Clock idle low, sample on rising edge.
eusartClockMode1	Clock idle low, sample on falling edge.
eusartClockMode2	Clock idle high, sample on falling edge.
eusartClockMode3	Clock idle high, sample on rising edge.

EUSART_CsPolarity_TypeDef

> EUSART_CsPolarity_TypeDef

Chip select polarity.

Enumerator	
eusartCsActiveLow	Chip select active low.
eusartCsActiveHigh	Chip select active high.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 ~ 9600 baudrate.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_SpiInit

```
void EUSART_SpiInit (EUSART_TypeDef * eusart, const EUSART_SpiInit_TypeDef * init)
```

Initialize EUSART when used in SPI mode.

Parameters

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

Initialize EUSART when used in SPI mode.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_Spi_TxRx

```
uint16_t EUSART_Spi_TxRx (EUSART_TypeDef * eusart, uint16_t data)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Transmit one 8-16 bit frame and return received data.

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.

Returns

- Data received and receive status.

Note

- SPI master mode only.
- This function will stall if the TX buffer is full until the buffer becomes available.

Transmit one 8-16 bit frame and return received data.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_PrsTriggerEnable

```
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.

Parameters

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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).

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 transmistter 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Enable Loop Back configuration.

advancedSettings

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

Advanced initialization structure pointer. It can be NULL.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

EUSART_PrsTriggerInit_TypeDef

PRS Trigger initialization structure.

Public Attributes

EUSART_PrsTriggerEnable_TypeDef	prs_trigger_enable PRS to EUSART trigger mode.
EUSART_PrsChannel_TypeDef	prs_trigger_channel PRS channel to be used to trigger auto transmission.

Public Attribute Documentation

prs_trigger_enable

```
EUSART_PrsTriggerEnable_TypeDef EUSART_PrsTriggerInit_TypeDef::prs_trigger_enable
```

PRS to EUSART trigger mode.

prs_trigger_channel

```
EUSART_PrsChannel_TypeDef EUSART_PrsTriggerInit_TypeDef::prs_trigger_channel
```

PRS channel to be used to trigger auto transmission.

Copyright © 2026 Silicon Laboratories. All rights reserved.

EUSART_SpiAdvancedInit_TypeDef

SPI Advanced initialization structure.

Public Attributes

EUSART_CsPolarity_TypeDef	csPolarity Chip select polarity.
EUSART_InvertIO_TypeDef	invertIO Enable inversion of RX and/or TX signals.
bool	autoCsEnable Enable automatic chip select. CS is managed by the peripheral.
bool	msbFirst If true, data will be send with most significant bit first.
uint8_t	autoCsSetupTime Auto CS setup time (before transmission) in baud cycles. Acceptable value (0 to 7 baud cycle).
uint8_t	autoCsHoldTime Auto CS hold time (after transmission) in baud cycles. Acceptable value (0 to 7 baud cycle).
uint8_t	autoInterFrameTime Inter-frame time in baud cycles. Acceptable value (0 to 7 baud cycle).
bool	autoTxEnable Enable AUTOTX mode.
uint16_t	defaultTxData Default transmitted data when the TXFIFO is empty.
bool	dmaWakeUpOnRx Enable the automatic wake up from EM2 to EM1 for DMA RX operation.
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	prsClockEnable Enable EUSART capability to use a PRS channel as an input SPI Clock.
EUSART_PrsChannel_TypeDef	prsClockChannel PRS Channel used to transmit SCLK from PRS to the EUSART.
EUSART_RxFifoWatermark_TypeDef	RxFifoWatermark Interrupt and status level of the Receive FIFO.
EUSART_TxFifoWatermark_TypeDef	TxFifoWatermark Interrupt and status level of the Receive FIFO.
bool	forceLoad Force load the first FIFO value.

Copyright © 2026 Silicon Laboratories. All rights reserved.

uint8_t	setupWindow	Setup window in bus clock cycles before the sampling edge of SCLK at word-boundary to avoid force load error.

Public Attribute Documentation

csPolarity

```
EUSART_CsPolarity_TypeDef EUSART_SpiAdvancedInit_TypeDef::csPolarity
```

Chip select polarity.

invertIO

```
EUSART_InvertIO_TypeDef EUSART_SpiAdvancedInit_TypeDef::invertIO
```

Enable inversion of RX and/or TX signals.

autoCsEnable

```
bool EUSART_SpiAdvancedInit_TypeDef::autoCsEnable
```

Enable automatic chip select. CS is managed by the peripheral.

msbFirst

```
bool EUSART_SpiAdvancedInit_TypeDef::msbFirst
```

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

autoCsSetupTime

```
uint8_t EUSART_SpiAdvancedInit_TypeDef::autoCsSetupTime
```

Auto CS setup time (before transmission) in baud cycles. Acceptable value (0 to 7 baud cycle).

autoCsHoldTime

```
uint8_t EUSART_SpiAdvancedInit_TypeDef::autoCsHoldTime
```

Auto CS hold time (after transmission) in baud cycles. Acceptable value (0 to 7 baud cycle).

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
uint8_t EUSART_SpiAdvancedInit_TypeDef::autoInterFrameTime
```

Inter-frame time in baud cycles. Acceptable value (0 to 7 baud cycle).

autoTxEnable

```
bool EUSART_SpiAdvancedInit_TypeDef::autoTxEnable
```

Enable AUTOTX mode.

Transmits as long as the RX FIFO is not full. Generates underflow interrupt if the TX FIFO is empty.

defaultTxData

```
uint16_t EUSART_SpiAdvancedInit_TypeDef::defaultTxData
```

Default transmitted data when the TXFIFO is empty.

dmaWakeUpOnRx

```
bool EUSART_SpiAdvancedInit_TypeDef::dmaWakeUpOnRx
```

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

Only applicable to EM2 (low frequency) capable EUSART instances.

prsRxEnable

```
bool EUSART_SpiAdvancedInit_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_SpiAdvancedInit_TypeDef::prsRxChannel
```

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

prsClockEnable

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
bool EUSART_SpiAdvancedInit_TypeDef::prsClockEnable
```

Enable EUSART capability to use a PRS channel as an input SPI Clock.

Slave mode only.

prsClockChannel

```
EUSART_PrsChannel_TypeDef EUSART_SpiAdvancedInit_TypeDef::prsClockChannel
```

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

RxFifoWatermark

```
EUSART_RxFifoWatermark_TypeDef EUSART_SpiAdvancedInit_TypeDef::RxFifoWatermark
```

Interrupt and status level of the Receive FIFO.

TxFifoWatermark

```
EUSART_TxFifoWatermark_TypeDef EUSART_SpiAdvancedInit_TypeDef::TxFifoWatermark
```

Interrupt and status level of the Receive FIFO.

forceLoad

```
bool EUSART_SpiAdvancedInit_TypeDef::forceLoad
```

Force load the first FIFO value.

setupWindow

```
uint8_t EUSART_SpiAdvancedInit_TypeDef::setupWindow
```

Setup window in bus clock cycles before the sampling edge of SCLK at word-boundary to avoid force load error.

Copyright © 2026 Silicon Laboratories. All rights reserved.

EUSART_SpiInit_TypeDef

SPI 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	**bitRate** Desired bit rate in Hz.
EUSART_Databits_TypeDef	**databits** Number of data bits in frame.
bool	**master** Select to operate in master or slave mode.
EUSART_ClockMode_TypeDef	**clockMode** Clock polarity/phase mode.
EUSART_LoopbackEnable_TypeDef	**loopbackEnable** Enable Loop Back configuration.
EUSART_SpiAdvancedInit_TypeDef *	**advancedSettings** Advanced initialization structure pointer. It can be NULL.

Public Attribute Documentation

enable

```
EUSART_Enable_TypeDef EUSART_SpiInit_TypeDef::enable
```

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

refFreq

```
uint32_t EUSART_SpiInit_TypeDef::refFreq
```

EUSART reference clock assumed when configuring baud rate setup.

Set to 0 if using currently configured reference clock.

bitRate

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
uint32_t EUSART_SpiInit_TypeDef::bitRate
```

Desired bit rate in Hz.

Depending on EUSART instance clock, not all bitrates are achievable as the divider is limited to 255.

databits

```
EUSART_Databits_TypeDef EUSART_SpiInit_TypeDef::databits
```

Number of data bits in frame.

master

```
bool EUSART_SpiInit_TypeDef::master
```

Select to operate in master or slave mode.

clockMode

```
EUSART_ClockMode_TypeDef EUSART_SpiInit_TypeDef::clockMode
```

Clock polarity/phase mode.

loopbackEnable

```
EUSART_LoopbackEnable_TypeDef EUSART_SpiInit_TypeDef::loopbackEnable
```

Enable Loop Back configuration.

advancedSettings

```
EUSART_SpiAdvancedInit_TypeDef* EUSART_SpiInit_TypeDef::advancedSettings
```

Advanced initialization structure pointer. It can be NULL.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
 * HFPER clock in addition to the GPCRC clock. */
CMU_ClockEnable(cmuClock_HFPER, 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:

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
#if !defined(_SILICON_LABS_32B_SERIES_2)
/* The GPCRC is a high frequency peripheral so we need to enable the
 * HFPER clock in addition to the GPCRC clock. */
CMU_ClockEnable(cmuClock_HFPER, 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
 * HFPER clock in addition to the GPCRC clock. */
CMU_ClockEnable(cmuClock_HFPER, true);
CMU_ClockEnable(cmuClock_GPCRC, true);
#endif

/* Initialize GPCRC with the 16-bit CRC-CCIT polynomial 0x1021 and use
 * the inital 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)

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

void	GPIO_PinModeSet(GPIO_Port_TypeDef port, unsigned int pin, GPIO_Mode_TypeDef mode, unsigned int out) Set the mode for a GPIO pin.
GPIO_Mode_TypeDef	GPIO_PinModeGet(GPIO_Port_TypeDef port, unsigned int pin) Get the mode for a GPIO pin.
void	GPIO_EM4EnablePinWakeup(uint32_t pinmask, uint32_t polaritymask) Enable GPIO pin wake-up from EM4.
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.
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_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.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Macros

#define SL_GPIO_PORT_A gpioPortA
Mapping between SL_GPIO_PORT_ enums and gpioPort values.

#define SL_GPIO_PORT_B gpioPortB

#define SL_GPIO_PORT_C gpioPortC

#define SL_GPIO_PORT_D gpioPortD

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
GPIO_Port_TypeDef	[in]	port	The port to associate with the `pin`.
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 `pin`.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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_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_EM4EnablePinWakeup

> void GPIO_EM4EnablePinWakeup (uint32_t pinmask, uint32_t polaritymask)

Enable GPIO pin wake-up from EM4.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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_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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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_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 `val` 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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		

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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)

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_IntClear(I2C_TypeDef *i2c, uint32_t flags)
Clear one or more pending I2C interrupts.

void I2C_IntDisable(I2C_TypeDef *i2c, uint32_t flags)

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
void	I2C_Reset(I2C_TypeDef *i2c) Reset I2C to the same state that it was in after a hardware reset.
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).
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).

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

#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.
i2cClockHLRAsymetric	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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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_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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

(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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 `len` long.
uint16_t	len Number of bytes in `data` to send or receive.
struct I2C_TransferSeq_TypeDef::@1	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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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::@1 I2C_TransferSeq_TypeDef::buf[2]

Buffers used to hold data to send from or receive into, depending on sequence type.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_KEEPINSTANDBY
iadcWarmupKeepWarm = _IADC_CTRL_WARMUPMODE_KEEPWARM

}
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))
iadcNegInputDac1 = (_IADC_SCAN_PORTNEG_DAC1 << (_IADC_SCAN_PORTNEG_SHIFT - _IADC_SCAN_PINNEG_SHIFT))
iadcNegInputPadAna1 = (_IADC_SCAN_PORTNEG_PADANA1 << (_IADC_SCAN_PORTNEG_SHIFT - _IADC_SCAN_PINNEG_SHIFT))

Copyright © 2026 Silicon Laboratories. All rights reserved.

iadcNegInputPadAna3 =
(_IADC_SCAN_PORTNEG_PADANA3
<< (_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

Copyright © 2026 Silicon Laboratories. All rights reserved.

iadcNegInputPortDPin13
iadcNegInputPortDPin14
iadcNegInputPortDPin15

}

IADC negative input selection.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
iadcPosInputDac0 = (_IADC_SCAN_PORTPOS_DAC0 << (_IADC_SCAN_PORTPOS_SHIFT - _IADC_SCAN_PINPOS_SHIFT))
iadcPosInputPadAna0 = (_IADC_SCAN_PORTPOS_PADANA0 << (_IADC_SCAN_PORTPOS_SHIFT - _IADC_SCAN_PINPOS_SHIFT))
iadcPosInputPadAna2 = (_IADC_SCAN_PORTPOS_PADANA2 << (_IADC_SCAN_PORTPOS_SHIFT - _IADC_SCAN_PINPOS_SHIFT))
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
  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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
iadcTriggerSelLesense = _IADC_TRIGGER_SCANTRIGSEL_LESENSE

}
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
iadcFifoCfgDvl5 = _IADC_SCANFIFOCFG_DVL_VALID5
iadcFifoCfgDvl6 = _IADC_SCANFIFOCFG_DVL_VALID6
iadcFifoCfgDvl7 = _IADC_SCANFIFOCFG_DVL_VALID7
iadcFifoCfgDvl8 = _IADC_SCANFIFOCFG_DVL_VALID8

}
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 sructure 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

#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.
iadcNegInputDac1	Direct connection to DAC_1 input pin.
iadcNegInputPadAna1	Direct connection to Pad_ana_1 input pin.
iadcNegInputPadAna3	Direct connection to Pad_ana_3 input pin.
iadcNegInputPortAPin0	GPIO port A pin 0.
iadcNegInputPortAPin1	GPIO port A pin 1.
iadcNegInputPortAPin2	GPIO port A pin 2.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
iadcPosInputDac0	Direct connection to DAC_0 input pin.
iadcPosInputPadAna0	Direct connection to Pad_ana_0 input pin.
iadcPosInputPadAna2	Direct connection to Pad_ana_2 input pin.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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
iadcTriggerSelLesense	LESENSE starts 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.
iadcFifoCfgDvl5	Data valid level is 5 before requesting DMA transfer.
iadcFifoCfgDvl6	Data valid level is 6 before requesting DMA transfer.
iadcFifoCfgDvl7	Data valid level is 7 before requesting DMA transfer.
iadcFifoCfgDvl8	Data valid level is 8 before requesting DMA transfer.

IADC_DigitalAveraging_t

> IADC_DigitalAveraging_t

IADC digital averaging function.

Enumerator

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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: CLK_SRC_ADC / (srcClkPrescaler + 1).

Returns

- Divider value to use for IADC in order to achieve a high speed clock value <= `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: CLK_SRC_ADC / (adcClkprescale + 1).

Copyright © 2026 Silicon Laboratories. All rights reserved.

Returns

- Divider value to use for IADC in order to achieve a ADC_CLK frequency <= `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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

IADC_Config_t

IADC config structure.

Public Attributes

IADC_CfgAdcMode_t	**adcMode** IADC mode; Normal, High speed or High Accuracy.
IADC_CfgOsrHighSpeed_t	**osrHighSpeed** Over sampling ratio for High Speed and Normal modes.
IADC_CfgAnalogGain_t	**analogGain** Analog gain.
IADC_CfgReference_t	**reference** Reference selection.
IADC_CfgTwosComp_t	**twosComplement** Two's complement reporting.
uint32_t	**adcClkPrescale** ADC_CLK divider (prescale+1).
uint32_t	**vRef** Vref magnitude expressed in millivolts.
IADC_DigitalAveraging_t	**digAvg** 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

IADC_InitScan_t

IADC scan init structure.

Public Attributes

IADC_Alignment_t	**alignment** Alignment of data in FIFO.
bool	**showId** Tag FIFO entry with scan table entry id.
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	**start** 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

IADC_SingleInput_t

IADC single input selection structure.

Public Attributes

IADC_NegInput_t	negInput Port/pin input for the negative side of the ADC.
IADC_PosInput_t	posInput Port/pin input for the positive side of the ADC.
uint8_t	configId Configuration id.
bool	compare 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

IADC_ScanTableEntry_t

IADC scan table entry structure.

Public Attributes

IADC_NegInput_t	negInput Port/pin input for the negative side of the ADC.
IADC_PosInput_t	posInput Port/pin input for the positive side of the ADC.
uint8_t	configId Configuration id.
bool	compare Perform digital window comparison on the result from this entry.
bool	includeInScan 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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[IADC0_ENTRIES]
```

IADC scan table entries.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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).

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD - Liquid Crystal Display

LCD - Liquid Crystal Display

Liquid Crystal Display (LCD) Peripheral API.

This module contains functions to control the LDC peripheral of Silicon Labs 32-bit MCUs and SoCs. The LCD driver can drive up to 8x36 segmented LCD directly. The animation feature makes it possible to have active animations without the CPU intervention.

Modules

LCD_AnimInit_TypeDef

LCD_FrameCountInit_TypeDef

LCD_Init_TypeDef

Enumerations

enum LCD_Mux_TypeDef {

lcdMuxStatic = LCD_DISPCTRL_MUX_STATIC
lcdMuxDuplex = LCD_DISPCTRL_MUX_DUPLEX
lcdMuxTriplex = LCD_DISPCTRL_MUX_TRIPLEX
lcdMuxQuadruplex = LCD_DISPCTRL_MUX_QUADRUPLEX
lcdMuxSextaplex = LCD_DISPCTRL_MUX_SEXTAPLEX
lcdMuxOctaplex = LCD_DISPCTRL_MUX_OCTAPLEX

}
MUX setting.

enum LCD_Wave_TypeDef {

lcdWaveLowPower = LCD_DISPCTRL_WAVE_TYPEB
lcdWaveNormal = LCD_DISPCTRL_WAVE_TYPEA

}
Wave type.

enum LCD_Bias_TypeDef {

lcdBiasStatic = LCD_DISPCTRL_BIAS_STATIC
lcdBiasOneHalf = LCD_DISPCTRL_BIAS_ONEHALF
lcdBiasOneThird = LCD_DISPCTRL_BIAS_ONETHIRD
lcdBiasOneFourth = LCD_DISPCTRL_BIAS_ONEFOURTH

}
Bias setting.

enum LCD_Mode_Typedef {

lcdModeStepDown = LCD_BIASCTRL_MODE_STEPDOWN
lcdModeChargePump = LCD_BIASCTRL_MODE_CHARGEPUMP

}
Contrast Configuration.

enum LCD_FCPreScale_TypeDef {

lcdFCPrescDiv1 = LCD_BACFG_FCPRESC_DIV1
lcdFCPrescDiv2 = LCD_BACFG_FCPRESC_DIV2

Copyright © 2026 Silicon Laboratories. All rights reserved.

lcdFCPrescDiv4 =
LCD_BACFG_FCPRESC_DIV4
lcdFCPrescDiv8 =
LCD_BACFG_FCPRESC_DIV8

}
Frame Counter Clock Prescaler,
FC-CLK = FrameRate (Hz) / this
factor.

enum LCD_UpdateCtrl_TypeDef {

lcdUpdateCtrlRegular = LCD_CTRL_UDCTRL_REGULAR
lcdUpdateCtrlFCEvent = LCD_CTRL_UDCTRL_FCEVENT
lcdUpdateCtrlFrameStart = LCD_CTRL_UDCTRL_FRAMESTART
lcdUpdateCtrlDisplayEvent = LCD_CTRL_UDCTRL_DISPLAYEVENT

}
Update Data Control.

enum LCD_LoadAddr_TypeDef {

lcdLoadAddrNone = 0
lcdLoadAddrBactrl = LCD_UPDATECTRL_LOADADDR_BACTRLWR
lcdLoadAddrAregA = LCD_UPDATECTRL_LOADADDR_AREGAWR
lcdLoadAddrAregB = LCD_UPDATECTRL_LOADADDR_AREGBWR
lcdLoadAddrSegd0 = LCD_UPDATECTRL_LOADADDR_SEGD0WR
lcdLoadAddrSegd1 = LCD_UPDATECTRL_LOADADDR_SEGD1WR
lcdLoadAddrSegd2 = LCD_UPDATECTRL_LOADADDR_SEGD2WR
lcdLoadAddrSegd3 = LCD_UPDATECTRL_LOADADDR_SEGD3WR
lcdLoadAddrSegd4 = LCD_UPDATECTRL_LOADADDR_SEGD4WR
lcdLoadAddrSegd5 = LCD_UPDATECTRL_LOADADDR_SEGD5WR
lcdLoadAddrSegd6 = LCD_UPDATECTRL_LOADADDR_SEGD6WR
lcdLoadAddrSegd7 = LCD_UPDATECTRL_LOADADDR_SEGD7WR

}
Auto Load address which will start the synchronization from CLK_BUS to CLK_PER.

enum LCD_AnimShift_TypeDef {

lcdAnimShiftNone = _LCD_BACTRL_AREGASC_NOSHIFT
lcdAnimShiftLeft = _LCD_BACTRL_AREGASC_SHIFTLEFT
lcdAnimShiftRight = _LCD_BACTRL_AREGASC_SHIFTRIGHT

}
Animation Shift operation; none, left or right.

enum LCD_AnimLogic_TypeDef {

lcdAnimLogicAnd = LCD_BACTRL_ALOGSEL_AND
lcdAnimLogicOr = LCD_BACTRL_ALOGSEL_OR

}
Animation Logic Control, how AReg and BReg should be combined.

enum LCD_AnimLoc_TypeDef {

lcdAnimLocSeg0To7 = LCD_BACTRL_ALOC_SEG0TO7
lcdAnimLocSeg8To15 = LCD_BACTRL_ALOC_SEG8TO15

}
Animation Location, set the LCD segments which animation applies to.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum LCD_ChargeRedistribution_TypeDef {

lcdChargeRedistributionDisable = LCD_DISPCTRL_CHGRDST_DISABLE
lcdChargeRedistributionEnable = LCD_DISPCTRL_CHGRDST_ONE
lcdChargeRedistributionTwoCycle = LCD_DISPCTRL_CHGRDST_TWO
lcdChargeRedistributionThreeCycle = LCD_DISPCTRL_CHGRDST_THREE
lcdChargeRedistributionFourCycle = LCD_DISPCTRL_CHGRDST_FOUR

}
Charge redistribution control.

enum LCD_DmaMode_Typedef {

lcdDmaModeDisable = LCD_BIASCTRL_DMAMODE_DMADISABLE
lcdDmaModeFrameCounterEvent = LCD_BIASCTRL_DMAMODE_DMAFC
lcdDmaModeDisplayEvent = LCD_BIASCTRL_DMAMODE_DMADISPLAY

}
DMA mode of operation.

Functions

void LCD_Init(const LCD_Init_TypeDef *lcdInit)
Initialize the Liquid Crystal Display (LCD) controller.

void LCD_UpdateCtrl(LCD_UpdateCtrl_TypeDef ud)
Configure Update Control.

void LCD_FrameCountInit(const LCD_FrameCountInit_TypeDef *fcInit)
Initialize the LCD Frame Counter.

void LCD_AnimInit(const LCD_AnimInit_TypeDef *animInit)
Configure the LCD controller Animation feature.

void LCD_SegmentEnable(uint32_t seg_nbr, bool enable)
Enables a given LCD segment line.

void LCD_ComEnable(uint8_t com, bool enable)
Enables a given LCD COM line.

void LCD_DmaModeSet(LCD_DmaMode_Typedef mode)
Set a given DMA mode operation.

void LCD_SegmentSet(int com, int bit, bool enable)
Turn on or clear a segment.

void LCD_SegmentSetLow(int com, uint32_t mask, uint32_t bits)
Update 0-31 lowest segments on a given COM-line in one operation according to the bit mask.

void LCD_ContrastSet(int level)
Configure the contrast level on the LCD panel.

void LCD_BiasSet(LCD_Bias_TypeDef bias)
Configure the bias level on the LCD panel.

void LCD_BiasSegmentSet(int segmentLine, int biasLevel)
Configure the bias level for a specific segment line for Direct Segment Control.

void LCD_BiasComSet(int comLine, int biasLevel)
Configure the bias level for a specific segment line.

void LCD_ModeSet(LCD_Mode_Typedef mode)
Configure the mode for the LCD panel.

Copyright © 2026 Silicon Laboratories. All rights reserved.

void	LCD_ChargeRedistributionCyclesSet(uint8_t cycles) Configure the charge redistribution cycles for the LCD panel.
void	LCD_LoadBusyWait(void) Wait for load synchronization completion.
void	LCD_ReadyWait(void) Wait for the LCD to complete resetting or disabling procedure.
void	LCD_Enable(bool enable) Enable or disable LCD controller.
void	LCD_Reset(void) Reset the LCD.
void	LCD_AnimEnable(bool enable) Enable or disable LCD Animation feature.
void	LCD_BlinkEnable(bool enable) Enable or disable the LCD blink.
void	LCD_BlankEnable(bool enable) Disable all segments while keeping segment state.
void	LCD_FrameCountEnable(bool enable) Enable or disable LCD Frame counter.
void	LCD_DisplayCountEnable(bool enable) Enable or disable the LCD Display counter.
int	LCD_AnimState(void) Return the current animation state.
int	LCD_BlinkState(void) Return the current blink state.
void	LCD_SyncStart(bool autoload, LCD_LoadAddr_TypeDef load_addr) Start the synchronization process.
void	LCD_SyncStop(bool autoload) Stop the synchronization process.
uint32_t	LCD_IntGet(void) Get pending LCD interrupt flags.
uint32_t	LCD_IntGetEnabled(void) Get enabled and pending LCD interrupt flags.
void	LCD_IntSet(uint32_t flags) Set one or more pending LCD interrupts from SW.
void	LCD_IntEnable(uint32_t flags) Enable LCD interrupts.
void	LCD_IntDisable(uint32_t flags) Disable LCD interrupts.
void	LCD_IntClear(uint32_t flags) Clear one or more interrupt flags.
void	LCD_DSCEnable(bool enable) Enable or disable LCD Direct Segment Control.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Macros

#define LCD_DEFAULT_CLOCK_PRESCALER 64
Default Clock Prescaler.

#define LCD_DEFAULT_FRAME_RATE_DIV 4
Default LCD Frame Rate Divisor.

#define LCD_DEFAULT_CONTRAST 15
Default LCD Contrast.

#define LCD_COM_LINES_MAX (LCD_COM_NUM + LCD_SEGASCOM_NUM)
Maximum common lines of LCD.

#define LCD_SEGMENT_LINES_MAX LCD_SEG_NUM
Maximum segment lines of LCD.

#define LCD_INIT_DEFAULT undefined
Default configuration for LCD initialization structure, enables 160 segments.

#define LCD_FRAME_COUNTER_VAL_MAX 64
Frame counter uses a maximum of 5 bits (FCTOP[5:0]).

Enumeration Documentation

LCD_Mux_TypeDef

> LCD_Mux_TypeDef

MUX setting.

Enumerator	
lcdMuxStatic	Static (segments can be multiplexed with LCD_COM[0]).
lcdMuxDuplex	Duplex / 1/2 Duty cycle (segments can be multiplexed with LCD_COM[0:1]).
lcdMuxTriplex	Triplex / 1/3 Duty cycle (segments can be multiplexed with LCD_COM[0:2]).
lcdMuxQuadruplex	Quadruplex / 1/4 Duty cycle (segments can be multiplexed with LCD_COM[0:3]).
lcdMuxSextaplex	Sextaplex / 1/6 Duty cycle (segments can be multiplexed with LCD_COM[0:5]).
lcdMuxOctaplex	Octaplex / 1/8 Duty cycle (segments can be multiplexed with LCD_COM[0:7]).

LCD_Wave_TypeDef

> LCD_Wave_TypeDef

Wave type.

Enumerator	
lcdWaveLowPower	Low power optimized waveform output.
lcdWaveNormal	Regular waveform output.

LCD_Bias_TypeDef

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_Bias_TypeDef

Bias setting.

Enumerator	
lcdBiasStatic	Static (2 levels).
lcdBiasOneHalf	1/2 Bias (3 levels).
lcdBiasOneThird	1/3 Bias (4 levels).
lcdBiasOneFourth	1/4 Bias (5 levels).

LCD_Mode_Typedef

LCD_Mode_Typedef

Contrast Configuration.

Mode of operation.

Enumerator	
lcdModeStepDown	External cap with resistor string.
lcdModeChargePump	External cap and internal oscillator.

LCD_FCPreScale_TypeDef

LCD_FCPreScale_TypeDef

Frame Counter Clock Prescaler, FC-CLK = FrameRate (Hz) / this factor.

Enumerator	
lcdFCPrescDiv1	Prescale Div 1.
lcdFCPrescDiv2	Prescale Div 2.
lcdFCPrescDiv4	Prescale Div 4.
lcdFCPrescDiv8	Prescale Div 8.

LCD_UpdateCtrl_TypeDef

LCD_UpdateCtrl_TypeDef

Update Data Control.

Enumerator	
lcdUpdateCtrlRegular	Regular update, data transfer done immediately.
lcdUpdateCtrlFCEvent	Data transfer done at Frame Counter event.
lcdUpdateCtrlFrameStart	Data transfer done at Frame Start.
lcdUpdateCtrlDisplayEvent	Data transfer done at Display Counter event.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_LoadAddr_TypeDef

LCD_LoadAddr_TypeDef

Auto Load address which will start the synchronization from CLK_BUS to CLK_PER.

Enumerator	
lcdLoadAddrNone	Starts synchronizing registers after a write to BACTRL.
lcdLoadAddrBactrl	Starts synchronizing registers after a write to BACTRL.
lcdLoadAddrAregA	Starts synchronizing registers after a write to AREGA.
lcdLoadAddrAregB	Starts synchronizing registers after a write to AREGB.
lcdLoadAddrSegd0	Starts synchronizing registers after a write to SEGD0.
lcdLoadAddrSegd1	Starts synchronizing registers after a write to SEGD1.
lcdLoadAddrSegd2	Starts synchronizing registers after a write to SEGD2.
lcdLoadAddrSegd3	Starts synchronizing registers after a write to SEGD3.
lcdLoadAddrSegd4	Starts synchronizing registers after a write to SEGD4.
lcdLoadAddrSegd5	Starts synchronizing registers after a write to SEGD5.
lcdLoadAddrSegd6	Starts synchronizing registers after a write to SEGD6.
lcdLoadAddrSegd7	Starts synchronizing registers after a write to SEGD7.

LCD_AnimShift_TypeDef

LCD_AnimShift_TypeDef

Animation Shift operation; none, left or right.

Enumerator	
lcdAnimShiftNone	No shift.
lcdAnimShiftLeft	Shift segment bits left.
lcdAnimShiftRight	Shift segment bits right.

LCD_AnimLogic_TypeDef

LCD_AnimLogic_TypeDef

Animation Logic Control, how AReg and BReg should be combined.

Enumerator	
lcdAnimLogicAnd	Use bitwise logic AND to mix animation register A (AREGA) and B (AREGB).
lcdAnimLogicOr	Use bitwise logic OR to mix animation register A (AREGA) and B (AREGB).

LCD_AnimLoc_TypeDef

LCD_AnimLoc_TypeDef

Copyright © 2026 Silicon Laboratories. All rights reserved.

Animation Location, set the LCD segments which animation applies to.

Enumerator	
lcdAnimLocSeg0To7	Animation appears on segments 0 to 7.
lcdAnimLocSeg8To15	Animation appears on segments 8 to 15.

LCD_ChargeRedistribution_TypeDef

```
LCD_ChargeRedistribution_TypeDef
```

Charge redistribution control.

Enumerator	
lcdChargeRedistributionDisable	Disable charge redistribution.
lcdChargeRedistributionEnable	Use 1 prescaled low frequency clock cycle for charge redistribution.
lcdChargeRedistributionTwoCycle	Use 2 prescaled low frequency clock cycle for charge redistribution.
lcdChargeRedistributionThreeCycle	Use 3 prescaled low frequency clock cycle for charge redistribution.
lcdChargeRedistributionFourCycle	Use 4 prescaled low frequency clock cycle for charge redistribution.

LCD_DmaMode_Typedef

```
LCD_DmaMode_Typedef
```

DMA mode of operation.

Enumerator	
lcdDmaModeDisable	No DMA requests are generated.
lcdDmaModeFrameCounterEvent	DMA request on frame counter event.
lcdDmaModeDisplayEvent	DMA request on display counter event.

Function Documentation

LCD_Init

```
void LCD_Init (const LCD_Init_TypeDef * lcdInit)
```

Initialize the Liquid Crystal Display (LCD) controller.

Parameters

Type	Direction	Argument Name	Description
const LCD_Init_TypeDef *	[in]	lcdInit	A pointer to the initialization structure which configures the LCD controller.

Configures the LCD controller. You must enable it afterwards, potentially configuring Frame Control and interrupts first according to requirements.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_UpdateCtrl

```
void LCD_UpdateCtrl (LCD_UpdateCtrl_TypeDef ud)
```

Configure Update Control.

Parameters

Type	Direction	Argument Name	Description
LCD_UpdateCtrl_TypeDef	[in]	ud	Configures the LCD update method.

LCD_FrameCountInit

```
void LCD_FrameCountInit (const LCD_FrameCountInit_TypeDef * fcInit)
```

Initialize the LCD Frame Counter.

Parameters

Type	Direction	Argument Name	Description
const LCD_FrameCountInit_TypeDef *	[in]	fcInit	A pointer to the Frame Counter initialization structure.

LCD_AnimInit

```
void LCD_AnimInit (const LCD_AnimInit_TypeDef * animInit)
```

Configure the LCD controller Animation feature.

Parameters

Type	Direction	Argument Name	Description
const LCD_AnimInit_TypeDef *	[in]	animInit	A pointer to the LCD Animation initialization structure.

LCD_SegmentEnable

```
void LCD_SegmentEnable (uint32_t seg_nbr, bool enable)
```

Enables a given LCD segment line.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	seg_nbr	Segment line number.
bool	[in]	enable	Boolean true to enable a segment, false to disable.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_ComEnable

```
void LCD_ComEnable (uint8_t com, bool enable)
```

Enables a given LCD COM line.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	com	COM line number.
bool	[in]	enable	Boolean true to enable a COM , false to disable.

LCD_DmaModeSet

```
void LCD_DmaModeSet (LCD_DmaMode_Typedef mode)
```

Set a given DMA mode operation.

Parameters

Type	Direction	Argument Name	Description
LCD_DmaMode_Typedef	[in]	mode	DMA mode.

LCD_SegmentSet

```
void LCD_SegmentSet (int com, int bit, bool enable)
```

Turn on or clear a segment.

Parameters

Type	Direction	Argument Name	Description
int	[in]	com	A COM line to change.
int	[in]	bit	A bit index indicating which field to change.
bool	[in]	enable	True will set segment, false will clear segment.

Note

- For the Gecko Family, the maximum configuration is (COM-lines x Segment-Lines) 4x40. For the Tiny Gecko Family, the maximum configuration is 8x20 or 4x24. For the Giant Gecko Family, the maximum configuration is 8x36 or 4x40. For the Series 2 Family, the maximum configuration is 4x20. For the Series 2 xG28, the maximum configuration is 8x24 or 4x28.

LCD_SegmentSetLow

```
void LCD_SegmentSetLow (int com, uint32_t mask, uint32_t bits)
```

Update 0-31 lowest segments on a given COM-line in one operation according to the bit mask.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
int	[in]	com	Indicates a COM line to update.
uint32_t	[in]	mask	A bit mask for segments 0-31.
uint32_t	[in]	bits	A bit pattern for segments 0-31.

LCD_ContrastSet

> void LCD_ContrastSet (int level)

Configure the contrast level on the LCD panel.

Parameters

Type	Direction	Argument Name	Description
int	[in]	level	The contrast level in range 0-63.

LCD_BiasSet

> void LCD_BiasSet (LCD_Bias_TypeDef bias)

Configure the bias level on the LCD panel.

Parameters

Type	Direction	Argument Name	Description
LCD_Bias_TypeDef	[in]	bias	The bias level.

LCD_BiasSegmentSet

> void LCD_BiasSegmentSet (int segmentLine, int biasLevel)

Configure the bias level for a specific segment line for Direct Segment Control.

Parameters

Type	Direction	Argument Name	Description
int	[in]	segmentLine	A segment line number.
int	[in]	biasLevel	The bias configuration level. This value must be within the constraints defined by the LCD_DISPCTRL bias settings. For more information, see the applicable Reference Manual and data sheet.

Note

- When DSC is active, each configuration takes up 4 bits in the corresponding Segment Registers (SEGD0L/SEGD1H for Series 0 and 1, SEGDx/SEGDxH for Series 2) which defines the bias level. For optimal use of this feature, the entire SEGD-registers should be set at once in an optimized routine. Therefore, this function shows how to correctly configure the bias levels and should be used with care.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_BiasComSet

```
void LCD_BiasComSet (int comLine, int biasLevel)
```

Configure the bias level for a specific segment line.

Parameters

Type	Direction	Argument Name	Description
int	[in]	comLine	A COM line number, between 0 and 7 for Series 0 and 1. For Series 2, max is LCD_COM_NUM, defined in device-specific headers.
int	[in]	biasLevel	The bias configuration level. This value must be within the constraints defined by the LCD_DISPCTRL bias settings. For more information, see the appropriate Reference Manual and data sheet.

Note

- When DSC is active, each configuration takes up 4 bits in the corresponding Segment Registers (SEGD4L/SEGD4H for Series 0 and 1, AREGA/AREGB for Series 2) which defines bias level. For optimal use of this feature, the entire register set should be set at once in a optimized routine. Therefore, this function shows how to correctly configure the bias levels and should be used with care.

LCD_ModeSet

```
void LCD_ModeSet (LCD_Mode_Typedef mode)
```

Configure the mode for the LCD panel.

Parameters

Type	Direction	Argument Name	Description
LCD_Mode_Typedef	[in]	mode	A mode.

LCD_ChargeRedistributionCyclesSet

```
void LCD_ChargeRedistributionCyclesSet (uint8_t cycles)
```

Configure the charge redistribution cycles for the LCD panel.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	cycles	Charge redistribution cycles, range 0-4.

LCD_LoadBusyWait

```
void LCD_LoadBusyWait (void )
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Wait for load synchronization completion.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Doing any writes to HV registers will not go through and will cause a bus fault.

LCD_ReadyWait

> void LCD_ReadyWait (void)

Wait for the LCD to complete resetting or disabling procedure.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

LCD_Enable

> void LCD_Enable (bool enable)

Enable or disable LCD controller.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	If true, enables LCD controller with current configuration. If false, disables LCD controller. Enable CMU clock for LCD for correct operation.

LCD_Reset

> void LCD_Reset (void)

Reset the LCD.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

LCD_AnimEnable

> void LCD_AnimEnable (bool enable)

Copyright © 2026 Silicon Laboratories. All rights reserved.

Enable or disable LCD Animation feature.

Parameters

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

LCD_BlinkEnable

```
void LCD_BlinkEnable (bool enable)
```

Enable or disable the LCD blink.

Parameters

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

LCD_BlankEnable

```
void LCD_BlankEnable (bool enable)
```

Disable all segments while keeping segment state.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	Boolean true clears all segments, boolean false restores all segment lines.

LCD_FrameCountEnable

```
void LCD_FrameCountEnable (bool enable)
```

Enable or disable LCD Frame counter.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	Boolean true enables frame counter, false disables frame counter.

LCD_DisplayCountEnable

```
void LCD_DisplayCountEnable (bool enable)
```

Enable or disable the LCD Display counter.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	Boolean true enables display counter, false disables display counter.

LCD_AnimState

```
int LCD_AnimState (void )
```

Return the current animation state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Animation state, in range 0-15.

LCD_BlinkState

```
int LCD_BlinkState (void )
```

Return the current blink state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Return value is 1 if segments are enabled, 0 if disabled.

LCD_SyncStart

```
void LCD_SyncStart (bool autoload, LCD_LoadAddr_TypeDef load_addr)
```

Start the synchronization process.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	autoload	Flag indicating if the synchronization is started manually with CMD.LOAD (false) or if the synchronization is managed automatically by Auto Load (true).
LCD_LoadAddr_TypeDef	[in]	load_addr	Address which will start the synchronization from CLK_BUS to CLK_PER when Auto Load is selected. This argument has no effect if 'autoload' is false.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_SyncStop

```
void LCD_SyncStop (bool autoload)
```

Stop the synchronization process.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	autoload	Flag indicating if the synchronization is stopped manually with CMD.CLEAR (false) or if the synchronization managed by Auto Load is disabled (true).

LCD_IntGet

```
uint32_t LCD_IntGet (void )
```

Get pending LCD interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

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

LCD_IntGetEnabled

```
uint32_t LCD_IntGetEnabled (void )
```

Get enabled and pending LCD interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

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 LCD interrupt sources. Return value is the bitwise AND combination of
  - the OR combination of enabled interrupt sources in LCD_IEN_nnn register (LCD_IEN_nnn) and
  - the bitwise OR combination of valid interrupt flags of LCD module (LCD_IF_nnn).

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_IntSet

```
void LCD_IntSet (uint32_t flags)
```

Set one or more pending LCD interrupts from SW.

Parameters

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

LCD_IntEnable

```
void LCD_IntEnable (uint32_t flags)
```

Enable LCD interrupts.

Parameters

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

LCD_IntDisable

```
void LCD_IntDisable (uint32_t flags)
```

Disable LCD interrupts.

Parameters

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

LCD_IntClear

```
void LCD_IntClear (uint32_t flags)
```

Clear one or more interrupt flags.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

LCD_DSCEnable

```
void LCD_DSCEnable (bool enable)
```

Enable or disable LCD Direct Segment Control.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	If true, enables LCD controller Direct Segment Control Segment and COM line bias levels need to be set explicitly with LCD_BiasSegmentSet() and LCD_BiasComSet() function calls respectively.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_AnimInit_TypeDef

LCD Animation Configuration.

Public Attributes

bool	enable Enable Animation at end of initialization.
uint32_t	AReg Initial Animation Register A Value.
LCD_AnimShift_TypeDef	AShift Shift operation of Animation Register A.
uint32_t	BReg Initial Animation Register B Value.
LCD_AnimShift_TypeDef	BShift Shift operation of Animation Register B.
LCD_AnimLogic_TypeDef	animLogic A and B Logical Operation to use for mixing and outputting resulting segments.
LCD_AnimLoc_TypeDef	startSeg Number of first segment to animate.

Public Attribute Documentation

enable

```
bool LCD_AnimInit_TypeDef::enable
```

Enable Animation at end of initialization.

AReg

```
uint32_t LCD_AnimInit_TypeDef::AReg
```

Initial Animation Register A Value.

AShift

```
LCD_AnimShift_TypeDef LCD_AnimInit_TypeDef::AShift
```

Shift operation of Animation Register A.

BReg

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
uint32_t LCD_AnimInit_TypeDef::BReg
```

Initial Animation Register B Value.

BShift

```
LCD_AnimShift_TypeDef LCD_AnimInit_TypeDef::BShift
```

Shift operation of Animation Register B.

animLogic

```
LCD_AnimLogic_TypeDef LCD_AnimInit_TypeDef::animLogic
```

A and B Logical Operation to use for mixing and outputting resulting segments.

startSeg

```
LCD_AnimLoc_TypeDef LCD_AnimInit_TypeDef::startSeg
```

Number of first segment to animate.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_FrameCountInit_TypeDef

LCD Frame Control Initialization.

Public Attributes

bool	enable Enable at end.
uint32_t	top Frame Counter top value.
LCD_FCPreScale_TypeDef	prescale Frame Counter clock prescaler.

Public Attribute Documentation

enable

```
bool LCD_FrameCountInit_TypeDef::enable
```

Enable at end.

top

```
uint32_t LCD_FrameCountInit_TypeDef::top
```

Frame Counter top value.

prescale

```
LCD_FCPreScale_TypeDef LCD_FrameCountInit_TypeDef::prescale
```

Frame Counter clock prescaler.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LCD_Init_TypeDef

LCD Controller Initialization structure.

Public Attributes

bool	enable Enable controller at end of initialization.
LCD_Mux_TypeDef	mux Mux configuration.
LCD_Bias_TypeDef	bias Bias configuration.
LCD_Wave_TypeDef	wave Wave configuration.
LCD_Mode_Typedef	mode Mode of operation.
LCD_ChargeRedistribution_TypeDef	chargeRedistribution Charge redistribution cycles.
uint8_t	frameRateDivider Frame rate divider.
int	contrastLevel Contrast level.
uint32_t	clockPrescaler Clock Prescaler.

Public Attribute Documentation

enable

```
bool LCD_Init_TypeDef::enable
```

Enable controller at end of initialization.

mux

```
LCD_Mux_TypeDef LCD_Init_TypeDef::mux
```

Mux configuration.

bias

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
LCD_Bias_TypeDef LCD_Init_TypeDef::bias
```

Bias configuration.

wave

```
LCD_Wave_TypeDef LCD_Init_TypeDef::wave
```

Wave configuration.

mode

```
LCD_Mode_Typedef LCD_Init_TypeDef::mode
```

Mode of operation.

chargeRedistribution

```
LCD_ChargeRedistribution_TypeDef LCD_Init_TypeDef::chargeRedistribution
```

Charge redistribution cycles.

frameRateDivider

```
uint8_t LCD_Init_TypeDef::frameRateDivider
```

Frame rate divider.

contrastLevel

```
int LCD_Init_TypeDef::contrastLevel
```

Contrast level.

clockPrescaler

```
uint32_t LCD_Init_TypeDef::clockPrescaler
```

Clock Prescaler.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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:

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
/* 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:

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
#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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Ping-pong DMA from serial port peripheral to memory:

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
#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);
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
/* 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. */

#if defined (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

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum LDMA_CtrlBlockSize_t {

ldmaCtrlBlockSizeUnit1 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT1
ldmaCtrlBlockSizeUnit2 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT2
ldmaCtrlBlockSizeUnit3 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT3
ldmaCtrlBlockSizeUnit4 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT4
ldmaCtrlBlockSizeUnit6 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT6
ldmaCtrlBlockSizeUnit8 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT8
ldmaCtrlBlockSizeUnit16 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT16
ldmaCtrlBlockSizeUnit32 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT32
ldmaCtrlBlockSizeUnit64 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT64
ldmaCtrlBlockSizeUnit128 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT128
ldmaCtrlBlockSizeUnit256 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT256
ldmaCtrlBlockSizeUnit512 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT512
ldmaCtrlBlockSizeUnit1024 = _LDMA_CH_CTRL_BLOCKSIZE_UNIT1024
ldmaCtrlBlockSizeAll = _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 {

ldmaCtrlStructTypeXfer = _LDMA_CH_CTRL_STRUCTTYPE_TRANSFER
ldmaCtrlStructTypeSync = _LDMA_CH_CTRL_STRUCTTYPE_SYNCHRONIZE
ldmaCtrlStructTypeWrite = _LDMA_CH_CTRL_STRUCTTYPE_WRITE

}

DMA structure type.

enum LDMA_CtrlReqMode_t {

ldmaCtrlReqModeBlock = _LDMA_CH_CTRL_REQMODE_BLOCK
ldmaCtrlReqModeAll = _LDMA_CH_CTRL_REQMODE_ALL

}

DMA transfer block or cycle selector.

enum LDMA_CtrlSrcInc_t {

ldmaCtrlSrcIncOne = _LDMA_CH_CTRL_SRCINC_ONE
ldmaCtrlSrcIncTwo = _LDMA_CH_CTRL_SRCINC_TWO
ldmaCtrlSrcIncFour = _LDMA_CH_CTRL_SRCINC_FOUR
ldmaCtrlSrcIncNone = _LDMA_CH_CTRL_SRCINC_NONE

}

Source address increment unit size.

enum LDMA_CtrlSize_t {

ldmaCtrlSizeByte = _LDMA_CH_CTRL_SIZE_BYTE
ldmaCtrlSizeHalf = _LDMA_CH_CTRL_SIZE_HALFWORD
ldmaCtrlSizeWord = _LDMA_CH_CTRL_SIZE_WORD

}

DMA transfer unit size.

enum LDMA_CtrlDstInc_t {

ldmaCtrlDstIncOne = _LDMA_CH_CTRL_DSTINC_ONE
ldmaCtrlDstIncTwo = _LDMA_CH_CTRL_DSTINC_TWO
ldmaCtrlDstIncFour = _LDMA_CH_CTRL_DSTINC_FOUR
ldmaCtrlDstIncNone = _LDMA_CH_CTRL_DSTINC_NONE

}

Destination address increment unit size.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum LDMA_CtrlSrcAddrMode_t {

ldmaCtrlSrcAddrModeAbs = _LDMA_CH_CTRL_SRCMODE_ABSOLUTE
ldmaCtrlSrcAddrModeRel = _LDMA_CH_CTRL_SRCMODE_RELATIVE

}

Source addressing mode.

enum LDMA_CtrlDstAddrMode_t {

ldmaCtrlDstAddrModeAbs = _LDMA_CH_CTRL_DSTMODE_ABSOLUTE
ldmaCtrlDstAddrModeRel = _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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum LDMA_PeripheralSignal_t {

ldmaPeripheralSignal_NONE = LDMAXBAR_CH_REQSEL_SOURCESEL_NONE
ldmaPeripheralSignal_LDMAXBAR_PRSREQ0 = LDMAXBAR_CH_REQSEL_SIGSEL_LDMAXBARPRSREQ0 | LDMAXBAR_CH_REQSEL_SOURCESEL_LDMAXBAR
ldmaPeripheralSignal_LDMAXBAR_PRSREQ1 = LDMAXBAR_CH_REQSEL_SIGSEL_LDMAXBARPRSREQ1 | LDMAXBAR_CH_REQSEL_SOURCESEL_LDMAXBAR
ldmaPeripheralSignal_TIMER0_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER0CC0 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER0
ldmaPeripheralSignal_TIMER0_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER0CC1 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER0
ldmaPeripheralSignal_TIMER0_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER0CC2 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER0
ldmaPeripheralSignal_TIMER0_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER0UFOF | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER0
ldmaPeripheralSignal_TIMER1_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER1CC0 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER1
ldmaPeripheralSignal_TIMER1_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER1CC1 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER1
ldmaPeripheralSignal_TIMER1_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER1CC2 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER1
ldmaPeripheralSignal_TIMER1_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER1UFOF | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER1
ldmaPeripheralSignal_USART0_RXDATAV = LDMAXBAR_CH_REQSEL_SIGSEL_USART0RXDATAV | LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
ldmaPeripheralSignal_USART0_RXDATAVRIGHT = LDMAXBAR_CH_REQSEL_SIGSEL_USART0RXDATAVRIGHT | LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
ldmaPeripheralSignal_USART0_TXBL = LDMAXBAR_CH_REQSEL_SIGSEL_USART0TXBL | LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
ldmaPeripheralSignal_USART0_TXBLRIGHT = LDMAXBAR_CH_REQSEL_SIGSEL_USART0TXBLRIGHT | LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
ldmaPeripheralSignal_USART0_TXEMPTY = LDMAXBAR_CH_REQSEL_SIGSEL_USART0TXEMPTY | LDMAXBAR_CH_REQSEL_SOURCESEL_USART0
ldmaPeripheralSignal_I2C0_RXDATAV = LDMAXBAR_CH_REQSEL_SIGSEL_I2C0RXDATAV | LDMAXBAR_CH_REQSEL_SOURCESEL_I2C0
ldmaPeripheralSignal_I2C0_TXBL = LDMAXBAR_CH_REQSEL_SIGSEL_I2C0TXBL | LDMAXBAR_CH_REQSEL_SOURCESEL_I2C0
ldmaPeripheralSignal_I2C1_RXDATAV = LDMAXBAR_CH_REQSEL_SIGSEL_I2C1RXDATAV | LDMAXBAR_CH_REQSEL_SOURCESEL_I2C1
ldmaPeripheralSignal_I2C1_TXBL = LDMAXBAR_CH_REQSEL_SIGSEL_I2C1TXBL | LDMAXBAR_CH_REQSEL_SOURCESEL_I2C1
ldmaPeripheralSignal_IADC0_IADC_SCAN = LDMAXBAR_CH_REQSEL_SIGSEL_IADC0IADC_SCAN | LDMAXBAR_CH_REQSEL_SOURCESEL_IADC0
ldmaPeripheralSignal_IADC0_IADC_SINGLE = LDMAXBAR_CH_REQSEL_SIGSEL_IADC0IADC_SINGLE | LDMAXBAR_CH_REQSEL_SOURCESEL_IADC0
ldmaPeripheralSignal_MSC_WDATA = LDMAXBAR_CH_REQSEL_SIGSEL_MSCWDATA | LDMAXBAR_CH_REQSEL_SOURCESEL_MSC
ldmaPeripheralSignal_TIMER2_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER2CC0 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER2
ldmaPeripheralSignal_TIMER2_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER2CC1 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER2
ldmaPeripheralSignal_TIMER2_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER2CC2 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER2
ldmaPeripheralSignal_TIMER2_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER2UFOF | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER2
ldmaPeripheralSignal_TIMER3_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER3CC0 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER3
ldmaPeripheralSignal_TIMER3_CC1 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER3CC1 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER3
ldmaPeripheralSignal_TIMER3_CC2 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER3CC2 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER3
ldmaPeripheralSignal_TIMER3_UFOF = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER3UFOF | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER3
ldmaPeripheralSignal_LCD = LDMAXBAR_CH_REQSEL_SIGSEL_LCD | LDMAXBAR_CH_REQSEL_SOURCESEL_LCD
ldmaPeripheralSignal_TIMER4_CC0 = LDMAXBAR_CH_REQSEL_SIGSEL_TIMER4CC0 | LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER4

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
  ldmaPeripheralSignal_TIMER4_CC1 =
  LDMAXBAR_CH_REQSEL_SIGSEL_TIMER4CC1 |
  LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER4
  ldmaPeripheralSignal_TIMER4_CC2 =
  LDMAXBAR_CH_REQSEL_SIGSEL_TIMER4CC2 |
  LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER4
  ldmaPeripheralSignal_TIMER4_UFOF =
  LDMAXBAR_CH_REQSEL_SIGSEL_TIMER4UFOF |
  LDMAXBAR_CH_REQSEL_SOURCESEL_TIMER4
  ldmaPeripheralSignal_VDAC0CH0REQ =
  LDMAXBAR_CH_REQSEL_SIGSEL_VDAC0CH0_REQ
  | LDMAXBAR_CH_REQSEL_SOURCESEL_VDAC0
  ldmaPeripheralSignal_VDAC0CH1REQ =
  LDMAXBAR_CH_REQSEL_SIGSEL_VDAC0CH1_REQ
  | LDMAXBAR_CH_REQSEL_SOURCESEL_VDAC0
  ldmaPeripheralSignal_EUSART0_RXFL =
  LDMAXBAR_CH_REQSEL_SIGSEL_EUSART0RXFL |
  LDMAXBAR_CH_REQSEL_SOURCESEL_EUSART0
  ldmaPeripheralSignal_EUSART0_TXFL =
  LDMAXBAR_CH_REQSEL_SIGSEL_EUSART0TXFL |
  LDMAXBAR_CH_REQSEL_SOURCESEL_EUSART0
  ldmaPeripheralSignal_EUSART1_RXFL =
  LDMAXBAR_CH_REQSEL_SIGSEL_EUSART1RXFL |
  LDMAXBAR_CH_REQSEL_SOURCESEL_EUSART1
  ldmaPeripheralSignal_EUSART1_TXFL =
  LDMAXBAR_CH_REQSEL_SIGSEL_EUSART1TXFL |
  LDMAXBAR_CH_REQSEL_SOURCESEL_EUSART1
  ldmaPeripheralSignal_EUSART2_RXFL =
  LDMAXBAR_CH_REQSEL_SIGSEL_EUSART2RXFL |
  LDMAXBAR_CH_REQSEL_SOURCESEL_EUSART2
  ldmaPeripheralSignal_EUSART2_TXFL =
  LDMAXBAR_CH_REQSEL_SIGSEL_EUSART2TXFL |
  LDMAXBAR_CH_REQSEL_SOURCESEL_EUSART2
  ldmaPeripheralSignal_LESENSE_BUFDATAV =
  LDMAXBAR_CH_REQSEL_SIGSEL_LESENSEFIFO |
  LDMAXBAR_CH_REQSEL_SOURCESEL_LESENSE
}
```

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

#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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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_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_LCD	

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_VDAC0CH0REQ	Trigger on VDAC0_CH0REQ.
ldmaPeripheralSignal_VDAC0CH1REQ	Trigger on VDAC0_CH1REQ.
ldmaPeripheralSignal_EUSART0_RXFL	Trigger on EUSART0_RXFL.
ldmaPeripheralSignal_EUSART0_TXFL	Trigger on EUSART0_TXFL.
ldmaPeripheralSignal_EUSART1_RXFL	Trigger on EUSART1_RXFL.
ldmaPeripheralSignal_EUSART1_TXFL	Trigger on EUSART1_TXFL.
ldmaPeripheralSignal_EUSART2_RXFL	Trigger on EUSART2_RXFL.
ldmaPeripheralSignal_EUSART2_TXFL	Trigger on EUSART2_TXFL.
ldmaPeripheralSignal_LESENSE_BUFDATAV	Trigger on LESENSEFIFO.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 )
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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	reserved1 Reserved.
uint32_t	syncSet Set bits in LDMA_CTRL.SYNCTRIG register.
uint32_t	syncClr Clear bits in LDMA_CTRL.SYNCTRIG register.
uint32_t	reserved2 Reserved.
uint32_t	matchVal Sync trigger match value.
uint32_t	matchEn Sync trigger match enable.
uint32_t	reserved3 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

reserved1

```
uint32_t LDMA_Descriptor_t::reserved1
```

Reserved.

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.

reserved2

Copyright © 2026 Silicon Laboratories. All rights reserved.

uint32_t LDMA_Descriptor_t::reserved2

Reserved.

matchVal

uint32_t LDMA_Descriptor_t::matchVal

Sync trigger match value.

matchEn

uint32_t LDMA_Descriptor_t::matchEn

Sync trigger match enable.

reserved3

uint32_t LDMA_Descriptor_t::reserved3

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

struct LDMA_Descriptor_t::@4 LDMA_Descriptor_t::wri

WRITE DMA descriptor, used for write immediate operations.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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).

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LDMA_TransferCfg_t

DMA transfer configuration structure.

This structure configures all aspects of a DMA transfer.

Public Attributes

uint32_t	ldmaReqSel Selects DMA trigger source.
uint8_t	ldmaCtrlSyncPrsClrOff PRS Synctrig clear enables to clear.
uint8_t	ldmaCtrlSyncPrsClrOn PRS Synctrig clear enables to set.
uint8_t	ldmaCtrlSyncPrsSetOff PRS Synctrig set enables to clear.
uint8_t	ldmaCtrlSyncPrsSetOn PRS Synctrig set enables to set.
bool	ldmaReqDis Mask the PRS trigger input.
bool	ldmaDbgHalt Dis.
LDMA_CfgArbSlots_t	ldmaCfgArbSlots Arbitration slot number.
LDMA_CfgSrcIncSign_t	ldmaCfgSrcIncSign Source address increment sign.
LDMA_CfgDstIncSign_t	ldmaCfgDstIncSign Destination address increment sign.
uint8_t	ldmaLoopCnt Counter for looped transfers.

Public Attribute Documentation

ldmaReqSel

```
uint32_t LDMA_TransferCfg_t::ldmaReqSel
```

Selects DMA trigger source.

ldmaCtrlSyncPrsClrOff

```
uint8_t LDMA_TransferCfg_t::ldmaCtrlSyncPrsClrOff
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

PRS Synctrig clear enables to clear.

ldmaCtrlSyncPrsClrOn

```
uint8_t LDMA_TransferCfg_t::ldmaCtrlSyncPrsClrOn
```

PRS Synctrig clear enables to set.

ldmaCtrlSyncPrsSetOff

```
uint8_t LDMA_TransferCfg_t::ldmaCtrlSyncPrsSetOff
```

PRS Synctrig set enables to clear.

ldmaCtrlSyncPrsSetOn

```
uint8_t LDMA_TransferCfg_t::ldmaCtrlSyncPrsSetOn
```

PRS Synctrig set enables to set.

ldmaReqDis

```
bool LDMA_TransferCfg_t::ldmaReqDis
```

Mask the PRS trigger input.

ldmaDbgHalt

```
bool LDMA_TransferCfg_t::ldmaDbgHalt
```

Dis.

DMA trig when CPU is halted.

ldmaCfgArbSlots

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
LDMA_CfgArbSlots_t LDMA_TransferCfg_t::ldmaCfgArbSlots
```

Arbitration slot number.

ldmaCfgSrcIncSign

```
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE - Low Energy Sensor

LESENSE - Low Energy Sensor

Low Energy Sensor (LESENSE) Peripheral API.

This module contains functions to control the LESENSE peripheral of Silicon Labs 32-bit MCUs and SoCs. LESENSE is a low-energy sensor interface capable of autonomously collecting and processing data from multiple sensors even when in EM2.

Modules

LESENSE_CoreCtrlDesc_TypeDef

LESENSE_TimeCtrlDesc_TypeDef

LESENSE_PerCtrlDesc_TypeDef

LESENSE_DecCtrlDesc_TypeDef

LESENSE_Init_TypeDef

LESENSE_ChDesc_TypeDef

LESENSE_ChAll_TypeDef

LESENSE_AltExDesc_TypeDef

LESENSE_ConfAltEx_TypeDef

LESENSE_DecStCond_TypeDef

LESENSE_DecStAll_TypeDef

Enumerations

enum LESENSE_ClkPresc_TypeDef {

lesenseClkDiv_1 = 0
lesenseClkDiv_2 = 1
lesenseClkDiv_4 = 2
lesenseClkDiv_8 = 3
lesenseClkDiv_16 = 4
lesenseClkDiv_32 = 5
lesenseClkDiv_64 = 6
lesenseClkDiv_128 = 7

}

Clock divisors for controlling the prescaling factor of the period counter.

enum LESENSE_ScanMode_TypeDef {

lesenseScanStartPeriodic = LESENSE_CFG_SCANMODE_PERIODIC
lesenseScanStartOneShot = LESENSE_CFG_SCANMODE_ONESHOT
lesenseScanStartPRS = LESENSE_CFG_SCANMODE_PRS

}

Scan modes.

enum LESENSE_PRSSel_TypeDef {

lesensePRSCh0 = 0

Copyright © 2026 Silicon Laboratories. All rights reserved.

lesensePRSCh1 =
1
lesensePRSCh2
= 2
lesensePRSCh3
= 3
lesensePRSCh4
= 4
lesensePRSCh5
= 5
lesensePRSCh6
= 6
lesensePRSCh7
= 7
lesensePRSCh8
= 8
lesensePRSCh9
= 9
lesensePRSCh10
= 10
lesensePRSCh11
= 11

}
PRS sources.

enum LESENSE_DMAWakeUp_TypeDef {

lesenseDMAWakeUpDisable = LESENSE_CFG_DMAWU_DISABLE
lesenseDMAWakeUpEnable = LESENSE_CFG_DMAWU_ENABLE

}
Modes of operation for DMA wakeup from EM2.

enum LESENSE_ScanConfSel_TypeDef {

lesenseScanConfDirMap = LESENSE_CFG_SCANCONF_DIRMAP
lesenseScanConfInvMap = LESENSE_CFG_SCANCONF_INVMAP
lesenseScanConfToggle = LESENSE_CFG_SCANCONF_TOGGLE
lesenseScanConfDecDef = LESENSE_CFG_SCANCONF_DECDEF

}
Scan configuration.

enum LESENSE_ControlDACData_TypeDef {

lesenseDACIfData = _LESENSE_PERCTRL_DACCH0DATA_DACDATA
lesenseThres = _LESENSE_PERCTRL_DACCH0DATA_THRES

}
DAC CHx data control configuration.

enum LESENSE_ControlACMP_TypeDef {

lesenseACMPModeMux = _LESENSE_PERCTRL_ACMP0MODE_MUX
lesenseACMPModeMuxThres = _LESENSE_PERCTRL_ACMP0MODE_MUXTHRES

}
ACMPx control configuration.

enum LESENSE_ChSampleMode_TypeDef {

lesenseSampleModeCounter = LESENSE_CH_INTERACT_SAMPLE_ACMPCOUNT
lesenseSampleModeACMP = LESENSE_CH_INTERACT_SAMPLE_ACMP
lesenseSampleModeADC = LESENSE_CH_INTERACT_SAMPLE_ADC
lesenseSampleModeADCDiff = LESENSE_CH_INTERACT_SAMPLE_ADCDIFF

}
Decoder input source configuration.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum LESENSE_ChIntMode_TypeDef {

lesenseSetIntNone = LESENSE_CH_INTERACT_SETIF_NONE
lesenseSetIntLevel = LESENSE_CH_INTERACT_SETIF_LEVEL
lesenseSetIntPosEdge = LESENSE_CH_INTERACT_SETIF_POSEDGE
lesenseSetIntNegEdge = LESENSE_CH_INTERACT_SETIF_NEGEDGE
lesenseSetIntBothEdges = LESENSE_CH_INTERACT_SETIF_BOTHEDGES

}
Interrupt generation setup for CHx interrupt flag.

enum LESENSE_ChPinExMode_TypeDef {

lesenseChPinExDis = LESENSE_CH_INTERACT_EXMODE_DISABLE
lesenseChPinExHigh = LESENSE_CH_INTERACT_EXMODE_HIGH
lesenseChPinExLow = LESENSE_CH_INTERACT_EXMODE_LOW
lesenseChPinExDACOut = LESENSE_CH_INTERACT_EXMODE_DACOUT

}
Channel pin mode for the excitation phase of the scan sequence.

enum LESENSE_ChPinIdleMode_TypeDef {

lesenseChPinIdleDis = _LESENSE_IDLECONF_CHIDLE0_DISABLE
lesenseChPinIdleHigh = _LESENSE_IDLECONF_CHIDLE0_HIGH
lesenseChPinIdleLow = _LESENSE_IDLECONF_CHIDLE0_LOW
lesenseChPinIdleDACC = _LESENSE_IDLECONF_CHIDLE0_DAC

}
Channel pin mode for the idle phase of scan sequence.

enum LESENSE_ChClk_TypeDef {

lesenseClkLF = _LESENSE_CH_INTERACT_EXCLK_LFACLK
lesenseClkHF = _LESENSE_CH_INTERACT_EXCLK_AUXHFRCO

}
Clock used for excitation and sample delay timing.

enum LESENSE_ChCompMode_TypeDef {

lesenseCompModeLess = LESENSE_CH_EVALCFG_COMP_LESS
lesenseCompModeGreaterOrEq = LESENSE_CH_EVALCFG_COMP_GE

}
Compare modes for counter comparison.

enum LESENSE_StoreSample_TypeDef {

lesenseStoreSampleDisable = _LESENSE_CH_EVALCFG_STRSAMPLE_DISABLE
lesenseStoreSampleData = _LESENSE_CH_EVALCFG_STRSAMPLE_DATA
lesenseStoreSampleDataSrc = _LESENSE_CH_EVALCFG_STRSAMPLE_DATASRC

}
Mode of Storing of Sensor Sample in Result Buffer.

enum LESENSE_ChEvalMode_TypeDef {

lesenseEvalModeThreshold = _LESENSE_CH_EVALCFG_MODE_THRES
lesenseEvalModeSlidingWindow = _LESENSE_CH_EVALCFG_MODE_SLIDINGWIN
lesenseEvalModeStepDetection = _LESENSE_CH_EVALCFG_MODE_STEPDET

}
Sensor evaluation modes.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum LESENSE_StTransAct_TypeDef {

lesenseTransActNone = LESENSE_ST_ARC_PRSACT_NONE
lesenseTransActPRS0 = LESENSE_ST_ARC_PRSACT_PRS0
lesenseTransActPRS1 = LESENSE_ST_ARC_PRSACT_PRS1
lesenseTransActPRS01 = LESENSE_ST_ARC_PRSACT_PRS01
lesenseTransActPRS2 = LESENSE_ST_ARC_PRSACT_PRS2
lesenseTransActPRS02 = LESENSE_ST_ARC_PRSACT_PRS02
lesenseTransActPRS12 = LESENSE_ST_ARC_PRSACT_PRS12
lesenseTransActPRS012 = LESENSE_ST_ARC_PRSACT_PRS012
lesenseTransActUp = LESENSE_ST_ARC_PRSACT_UP
lesenseTransActDown = LESENSE_ST_ARC_PRSACT_DOWN
lesenseTransActUpAndPRS2 = LESENSE_ST_ARC_PRSACT_UPANDPRS2
lesenseTransActDownAndPRS2 = LESENSE_ST_ARC_PRSACT_DOWNANDPRS2

}

Transition action modes.

Typedefs

typedef LESENSE_DecStCond_TypeDef — LESENSE_DecStDesc_TypeDef
Decoder state x configuration structure.

Functions

void LESENSE_Init(const LESENSE_Init_TypeDef *init, bool reqReset)
Initialize the LESENSE module.

uint32_t LESENSE_ScanFreqSet(uint32_t refFreq, uint32_t scanFreq)
Set the scan frequency for periodic scanning.

void LESENSE_ScanModeSet(LESENSE_ScanMode_TypeDef scanMode, bool start)
Set scan mode of the LESENSE channels.

void LESENSE_StartDelaySet(uint8_t startDelay)
Set the start delay of the sensor interaction on each channel.

void LESENSE_ClkDivSet(LESENSE_ChClk_TypeDef clk, LESENSE_ClkPresc_TypeDef clkDiv)
Set the clock division for LESENSE timers.

void LESENSE_ChannelAllConfig(const LESENSE_ChAll_TypeDef *confChAll)
Configure all (16) LESENSE sensor channels.

void LESENSE_ChannelConfig(const LESENSE_ChDesc_TypeDef *confCh, uint32_t chIdx)
Configure a single LESENSE sensor channel.

void LESENSE_AltExConfig(const LESENSE_ConfAltEx_TypeDef *confAltEx)
Configure the LESENSE alternate excitation modes.

void LESENSE_ChannelEnable(uint8_t chIdx, bool enaScanCh, bool enaPin)
Enable/disable LESENSE scan channel and the pin assigned to it.

void LESENSE_ChannelEnableMask(uint16_t chMask, uint16_t pinMask)
Enable/disable LESENSE scan channel and the pin assigned to it.

void LESENSE_ChannelTimingSet(uint8_t chIdx, uint8_t exTime, uint8_t sampleDelay, uint16_t measDelay)
Set LESENSE channel timing parameters.

Copyright © 2026 Silicon Laboratories. All rights reserved.

void LESENSE_ChannelThresSet(uint8_t chIdx, uint16_t acmpThres, uint16_t cntThres)
Set LESENSE channel threshold parameters.

void LESENSE_ChannelSlidingWindow(uint8_t chIdx, uint32_t windowSize, uint32_t initValue)
Configure a Sliding Window evaluation mode for a specific channel.

void LESENSE_ChannelStepDetection(uint8_t chIdx, uint32_t stepSize, uint32_t initValue)
Configure the step detection evaluation mode for a specific channel.

void LESENSE_WindowSizeSet(uint32_t windowSize)
Set the window size for all LESENSE channels.

void LESENSE_StepSizeSet(uint32_t stepSize)
Set the step size for all LESENSE channels.

void LESENSE_DecoderStateAllConfig(const LESENSE_DecStAll_TypeDef *confDecStAll)
Configure all LESENSE decoder states.

void LESENSE_DecoderStateConfig(const LESENSE_DecStDesc_TypeDef *confDecSt, uint32_t decSt)
Configure a single LESENSE decoder state.

uint32_t LESENSE_DecoderStateGet(void)
Get the current state of the LESENSE decoder.

void LESENSE_DecoderPrsOut(bool enable, uint32_t decMask, uint32_t decVal)
Enable or disable the PRS output from the LESENSE decoder.

void LESENSE_ScanStart(void)
Start scanning sensors.

void LESENSE_ScanStop(void)
Stop scanning sensors.

void LESENSE_DecoderStart(void)
Start the LESENSE decoder.

void LESENSE_ResultBufferClear(void)
Clear the result buffer.

void LESENSE_Reset(void)
Reset the LESENSE module.

void LESENSE_DecoderStop(void)
Stop LESENSE decoder.

uint32_t LESENSE_StatusGet(void)
Get the current status of LESENSE.

void LESENSE_StatusWait(uint32_t flag)
Wait until status of LESENSE is equal to what was requested.

uint32_t LESENSE_ChannelActiveGet(void)
Get the currently active channel index.

uint32_t LESENSE_ScanResultGet(void)
Get the latest scan comparison result (1 bit / channel).

uint32_t LESENSE_ScanResultDataGet(void)
Get the oldest unread data from the result buffer.

uint32_t LESENSE_SensorStateGet(void)
Get the current state of the LESENSE sensor.

Copyright © 2026 Silicon Laboratories. All rights reserved.

void LESENSE_IntClear(uint32_t flags)
Clear one or more pending LESENSE interrupts.

void LESENSE_IntEnable(uint32_t flags)
Enable one or more LESENSE interrupts.

void LESENSE_IntDisable(uint32_t flags)
Disable one or more LESENSE interrupts.

void LESENSE_IntSet(uint32_t flags)
Set one or more pending LESENSE interrupts from SW.

uint32_t LESENSE_IntGet(void)
Get pending LESENSE interrupt flags.

uint32_t LESENSE_IntGetEnabled(void)
Get enabled and pending LESENSE interrupt flags.

Macros

#define LESENSE_NUM_DECODER_STATES (_LESENSE_DECSTATE_DECSTATE_MASK + 1)
Number of decoder states supported by current device.

#define LESENSE_NUM_ARCS (LESENSE_NUM_DECODER_STATES << 1)
Number of ARCs supported by current device.

#define LESENSE_NUM_CHANNELS 16
Number of LESENSE channels.

#define LESENSE_CORECTRL_DESC_DEFAULT undefined
Default configuration for LESENSE_CtrlDesc_TypeDef structure.

#define LESENSE_TIMECTRL_DESC_DEFAULT undefined
Default configuration for LESENSE_TimeCtrlDesc_TypeDef structure.

#define LESENSE_PERCTRL_DESC_DEFAULT undefined
Default configuration for LESENSE_PerCtrl_TypeDef structure.

#define LESENSE_DECCTRL_DESC_DEFAULT undefined
Default configuration for LESENSE_PerCtrl_TypeDef structure.

#define LESENSE_INIT_DEFAULT undefined
Default configuration for LESENSE_Init_TypeDef structure.

#define LESENSE_CH_CONF_DEFAULT undefined
Default configuration for the scan channel.

#define LESENSE_SCAN_CONF_DEFAULT undefined
Default configuration for all the sensor channels.

#define LESENSE_ALTEX_CH_CONF_DEFAULT undefined
Default configuration for the alternate excitation channel.

#define LESENSE_ALTEX_CONF_DEFAULT undefined
Default configuration for all the alternate excitation channels.

#define LESENSE_ST_CONF_DEFAULT undefined
Default configuration for the decoder state condition.

Copyright © 2026 Silicon Laboratories. All rights reserved.

#define LESENSE_DECODER_CONF_DEFAULT
Default configuration for all decoder states.

Enumeration Documentation

LESENSE_ClkPresc_TypeDef

LESENSE_ClkPresc_TypeDef

Clock divisors for controlling the prescaling factor of the period counter.

Note: These enumeration values are used for different clock division related configuration parameters (hfPresc, lfPresc, pcPresc).

Enumerator	
lesenseClkDiv_1	Divide clock by 1.
lesenseClkDiv_2	Divide clock by 2.
lesenseClkDiv_4	Divide clock by 4.
lesenseClkDiv_8	Divide clock by 8.
lesenseClkDiv_16	Divide clock by 16.
lesenseClkDiv_32	Divide clock by 32.
lesenseClkDiv_64	Divide clock by 64.
lesenseClkDiv_128	Divide clock by 128.

LESENSE_ScanMode_TypeDef

LESENSE_ScanMode_TypeDef

Scan modes.

Enumerator	
lesenseScanStartPeriodic	New scan is started each time the period counter overflows.
lesenseScanStartOneShot	Single scan is performed when LESENSE_ScanStart() is called.
lesenseScanStartPRS	New scan is triggered by pulse on PRS channel.

LESENSE_PRSSel_TypeDef

LESENSE_PRSSel_TypeDef

PRS sources.

Note: These enumeration values are being used for different PRS related configuration parameters.

Enumerator	
lesensePRSCh0	PRS channel 0.
lesensePRSCh1	PRS channel 1.
lesensePRSCh2	PRS channel 2.

Copyright © 2026 Silicon Laboratories. All rights reserved.

lesensePRSCh3	PRS channel 3.
lesensePRSCh4	PRS channel 4.
lesensePRSCh5	PRS channel 5.
lesensePRSCh6	PRS channel 6.
lesensePRSCh7	PRS channel 7.
lesensePRSCh8	PRS channel 8.
lesensePRSCh9	PRS channel 9.
lesensePRSCh10	PRS channel 10.
lesensePRSCh11	PRS channel 11.

LESENSE_DMAWakeUp_TypeDef

> LESENSE_DMAWakeUp_TypeDef

Modes of operation for DMA wakeup from EM2.

Enumerator	
lesenseDMAWakeUpDisable	No DMA wakeup from EM2.
lesenseDMAWakeUpEnable	DMA wakeup from EM2.

LESENSE_ScanConfSel_TypeDef

> LESENSE_ScanConfSel_TypeDef

Scan configuration.

Enumerator	
lesenseScanConfDirMap	Channel configuration registers (CHx_CONF) used are directly mapped to the channel number.
lesenseScanConfInvMap	Channel configuration registers used are CHx+8_CONF for channels 0-7 and CHx-8_CONF for channels 8-15.
lesenseScanConfToggle	Channel configuration registers used toggles between CHX_SCANCONF and CHX+8_SCANCONF when channel x triggers.
lesenseScanConfDecDef	Decoder state defines the channel configuration register (CHx_CONF) to be used.

LESENSE_ControlDACData_TypeDef

> LESENSE_ControlDACData_TypeDef

DAC CHx data control configuration.

Enumerator	
lesenseDACIfData	DAC channel x data is defined by the DAC_CHxDATA register.
lesenseThres	DAC channel x data is defined by the THRES in LESENSE_CHx_INTERACT.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_ControlACMP_TypeDef

LESENSE_ControlACMP_TypeDef

ACMPx control configuration.

Enumerator	
lesenseACMPModeMux	LESENSE does not control ACMPx.
lesenseACMPModeMuxThres	LESENSE controls input mux of and threshold value of ACMPx.

LESENSE_ChSampleMode_TypeDef

LESENSE_ChSampleMode_TypeDef

Decoder input source configuration.

Ocelot only provides SENSORSTATE as input for the decoder. Compare source selection for sensor sampling.

Enumerator	
lesenseSampleModeCounter	Counter output will be used in comparison.
lesenseSampleModeACMP	ACMP output will be used in comparison.
lesenseSampleModeADC	ADC output will be used in comparison.
lesenseSampleModeADCDiff	Differential ADC output will be used in comparison.

LESENSE_ChIntMode_TypeDef

LESENSE_ChIntMode_TypeDef

Interrupt generation setup for CHx interrupt flag.

Enumerator	
lesenseSetIntNone	No interrupt is generated.
lesenseSetIntLevel	Set interrupt flag if the sensor triggers.
lesenseSetIntPosEdge	Set interrupt flag on positive edge of the sensor state.
lesenseSetIntNegEdge	Set interrupt flag on negative edge of the sensor state.
lesenseSetIntBothEdges	Set interrupt flag on both edges of the sensor state.

LESENSE_ChPinExMode_TypeDef

LESENSE_ChPinExMode_TypeDef

Channel pin mode for the excitation phase of the scan sequence.

Enumerator	
lesenseChPinExDis	Channel pin is disabled.

Copyright © 2026 Silicon Laboratories. All rights reserved.

lesenseChPinExHigh	Channel pin is configured as push-pull, driven HIGH.
lesenseChPinExLow	Channel pin is configured as push-pull, driven LOW.
lesenseChPinExDACOut	DAC output (only available on channel 0, 1, 2, 3, 12, 13, 14 and 15)

LESENSE_ChPinIdleMode_TypeDef

```
LESENSE_ChPinIdleMode_TypeDef
```

Channel pin mode for the idle phase of scan sequence.

Enumerator

lesenseChPinIdleDis	Channel pin is disabled in idle phase.
lesenseChPinIdleHigh	Channel pin is configured as push-pull, driven HIGH in idle phase.
lesenseChPinIdleLow	Channel pin is configured as push-pull, driven LOW in idle phase.
lesenseChPinIdleDACC	Channel pin is connected to DAC output in idle phase.

LESENSE_ChClk_TypeDef

```
LESENSE_ChClk_TypeDef
```

Clock used for excitation and sample delay timing.

Enumerator

lesenseClkLF	LFACLK (LF clock) is used.
lesenseClkHF	AUXHFRCO (HF clock) is used.

LESENSE_ChCompMode_TypeDef

```
LESENSE_ChCompMode_TypeDef
```

Compare modes for counter comparison.

Enumerator

lesenseCompModeLess	Comparison evaluates to 1 if sensor data is less than the counter threshold, or if ACMP output is 0.
lesenseCompModeGreaterOrEq	Comparison evaluates to 1 if sensor data is greater than, or equal to the counter threshold, or if the ACMP output is 1.

LESENSE_StoreSample_TypeDef

```
LESENSE_StoreSample_TypeDef
```

Mode of Storing of Sensor Sample in Result Buffer.

Enumerator

Copyright © 2026 Silicon Laboratories. All rights reserved.

lesenseStoreSampleDisable	Nothing will be stored in the result buffer.
lesenseStoreSampleData	The sensor sample data will be stored in the result buffer.
lesenseStoreSampleDataSrc	The data source (i.e.

LESENSE_ChEvalMode_TypeDef

```
LESENSE_ChEvalMode_TypeDef
```

Sensor evaluation modes.

Enumerator	
lesenseEvalModeThreshold	Threshold comparison evaluation mode.
lesenseEvalModeSlidingWindow	Sliding window evaluation mode.
lesenseEvalModeStepDetection	Step detection evaluation mode.

LESENSE_StTransAct_TypeDef

```
LESENSE_StTransAct_TypeDef
```

Transition action modes.

Enumerator	
lesenseTransActNone	No PRS pulses generated (if PRSCOUNT == 0).
lesenseTransActPRS0	Generate pulse on LESPRS0 (if PRSCOUNT == 0).
lesenseTransActPRS1	Generate pulse on LESPRS1 (if PRSCOUNT == 0).
lesenseTransActPRS01	Generate pulse on LESPRS0 and LESPRS1 (if PRSCOUNT == 0).
lesenseTransActPRS2	Generate pulse on LESPRS2 (for both PRSCOUNT == 0 and PRSCOUNT == 1).
lesenseTransActPRS02	Generate pulse on LESPRS0 and LESPRS2 (if PRSCOUNT == 0).
lesenseTransActPRS12	Generate pulse on LESPRS1 and LESPRS2 (if PRSCOUNT == 0).
lesenseTransActPRS012	Generate pulse on LESPRS0, LESPRS1 and LESPRS2 (if PRSCOUNT == 0).
lesenseTransActUp	Count up (if PRSCOUNT == 1).
lesenseTransActDown	Count down (if PRSCOUNT == 1).
lesenseTransActUpAndPRS2	Count up and generate pulse on LESPRS2 (if PRSCOUNT == 1).
lesenseTransActDownAndPRS2	Count down and generate pulse on LESPRS2 (if PRSCOUNT == 1).

Typedef Documentation

LESENSE_DecStDesc_TypeDef

```
typedef LESENSE_DecStCond_TypeDef LESENSE_DecStDesc_TypeDef
```

Decoder state x configuration structure.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Function Documentation

LESENSE_Init

```
void LESENSE_Init (const LESENSE_Init_TypeDef * init, bool reqReset)
```

Initialize the LESENSE module.

Parameters

Type	Direction	Argument Name	Description
const LESENSE_Init_TypeDef *	[in]	init	The LESENSE initialization structure.
bool	[in]	reqReset	Request to call LESENSE_Reset() first to initialize all LESENSE registers with default values.

This function configures the main parameters of the LESENSE interface. See the initialization parameter type definition (LESENSE_Init_TypeDef) for more details.

Note

- LESENSE_Init() is designed to initialize LESENSE once in an operation cycle. Be aware of the effects of reconfiguration if using this function from multiple sources in your code. This function has not been designed to be re-entrant. Requesting reset by setting `reqReset` to true is required in each reset or power-on cycle to configure the default values of the RAM mapped LESENSE registers. Notice that GPIO pins used by the LESENSE module must be properly configured by the user explicitly for the LESENSE to work as intended. (When configuring pins, one should remember to consider the sequence of configuration to avoid unintended pulses/glitches on output pins.)

LESENSE_ScanFreqSet

```
uint32_t LESENSE_ScanFreqSet (uint32_t refFreq, uint32_t scanFreq)
```

Set the scan frequency for periodic scanning.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	refFreq	Select reference LFACLK clock frequency in Hz. If set to 0, the current clock frequency is being used as a reference.
uint32_t	[in]	scanFreq	Set the desired scan frequency in Hz.

This function only applies to LESENSE if a period counter is used as a trigger for scan start. The calculation is based on the following formula: Fscan = LFACLKles / ((1+PCTOP)*2^PCPRESC)

Note

- Note that the calculation does not necessarily result in the requested scan frequency due to integer division. Check the return value for the resulted scan frequency.

Returns

-

Copyright © 2026 Silicon Laboratories. All rights reserved.

Frequency in Hz calculated and set by this function. Users can use this to compare the requested and set values.

LESENSE_ScanModeSet

```
void LESENSE_ScanModeSet (LESENSE_ScanMode_TypeDef scanMode, bool start)
```

Set scan mode of the LESENSE channels.

Parameters

Type	Direction	Argument Name	Description
LESENSE_ScanMode_TypeDef	[in]	scanMode	Select the location to map LESENSE alternate excitation channels. • lesenseScanStartPeriodic - A new scan is started each time the period counter overflows. • lesenseScanStartOneShot - A single scan is performed when LESENSE_ScanStart() is called. • lesenseScanStartPRS - A new scan is triggered by pulse on the PRS channel.
bool	[in]	start	If true, LESENSE_ScanStart() is immediately issued after configuration.

This function configures how the scan start is triggered. It can be used for re-configuring the scan mode while running the application but it is also used by LESENSE_Init() for initialization.

Note

- Users can configure the scan mode by LESENSE_Init() function, but only with a significant overhead. This simple function serves the purpose of controlling this parameter after the channel has been configured. Be aware of the effects of the non-atomic Read-Modify-Write cycle.

LESENSE_StartDelaySet

```
void LESENSE_StartDelaySet (uint8_t startDelay)
```

Set the start delay of the sensor interaction on each channel.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	startDelay	A number of LFACLK cycles to delay. A valid range: 0-3 (2 bit).

This function sets the start delay of the sensor interaction on each channel. It can be used for adjusting the start delay while running the application but it is also used by LESENSE_Init() for initialization.

Note

- Users can configure the start delay by LESENSE_Init() function, but only with a significant overhead. This simple function serves the purpose of controlling this parameter after the channel has been configured. Be aware of the effects of the non-atomic Read-Modify-Write cycle.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_ClkDivSet

> void LESENSE_ClkDivSet (LESENSE_ChClk_TypeDef clk, LESENSE_ClkPresc_TypeDef clkDiv)

Set the clock division for LESENSE timers.

Parameters

Type	Direction	Argument Name	Description
LESENSE_ChClk_TypeDef	[in]	clk	Select the clock to prescale. • lesenseClkHF - set AUXHFRCO clock divisor for HF timer. • lesenseClkLF - set LFACLKles clock divisor for LF timer.
LESENSE_ClkPresc_TypeDef	[in]	clkDiv	The clock divisor value. A valid range depends on the `clk` value.

Use this function to configure the clock division for the LESENSE timers used for excitation timing. The division setting is global but the clock source can be selected for each channel using LESENSE_ChannelConfig() function. See documentation for more details.

Note

- If AUXHFRCO is used for excitation timing, LFACLK can't exceed 500 kHz. LFACLK can't exceed 50 kHz if the ACMP threshold level (ACMPTHRES) is not equal for all channels.

LESENSE_ChannelAllConfig

> void LESENSE_ChannelAllConfig (const LESENSE_ChAll_TypeDef * confChAll)

Configure all (16) LESENSE sensor channels.

Parameters

Type	Direction	Argument Name	Description
const LESENSE_ChAll_TypeDef *	[in]	confChAll	A configuration structure for all (16) LESENSE sensor channels.

This function configures all sensor channels of the LESENSE interface. See the configuration parameter type definition (LESENSE_ChAll_TypeDef) for more details.

Note

- Channels can be configured individually using LESENSE_ChannelConfig() function. Notice that pins used by the LESENSE module must be properly configured by the user explicitly for LESENSE to work as intended. (When configuring pins, consider the sequence of the configuration to avoid unintended pulses/glitches on output pins.)

LESENSE_ChannelConfig

Copyright © 2026 Silicon Laboratories. All rights reserved.

> void LESENSE_ChannelConfig (const LESENSE_ChDesc_TypeDef * confCh, uint32_t chIdx)

Configure a single LESENSE sensor channel.

Parameters

Type	Direction	Argument Name	Description
const LESENSE_ChDesc_TypeDef *	[in]	confCh	A configuration structure for a single LESENSE sensor channel.
uint32_t	[in]	chIdx	A channel index to configure (0-15).

This function configures a single sensor channel of the LESENSE interface. See the configuration parameter type definition (LESENSE_ChDesc_TypeDef) for more details.

Note

- This function has been designed to minimize the effects of sensor channel reconfiguration while LESENSE is in operation. However, be aware of these effects and the right timing to call this function. Parameter `useAltEx` must be true in the channel configuration to use alternate excitation pins.

LESENSE_AltExConfig

> void LESENSE_AltExConfig (const LESENSE_ConfAltEx_TypeDef * confAltEx)

Configure the LESENSE alternate excitation modes.

Parameters

Type	Direction	Argument Name	Description
const LESENSE_ConfAltEx_TypeDef *	[in]	confAltEx	A configuration structure for LESENSE alternate excitation pins.

This function configures the alternate excitation channels of the LESENSE interface. See the configuration parameter type definition (LESENSE_ConfAltEx_TypeDef) for more details.

Note

- The `useAltEx` parameter must be true in the channel configuration structure (LESENSE_ChDesc_TypeDef) to use alternate excitation pins on the channel.

LESENSE_ChannelEnable

> void LESENSE_ChannelEnable (uint8_t chIdx, bool enaScanCh, bool enaPin)

Enable/disable LESENSE scan channel and the pin assigned to it.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	chIdx	An identifier of the scan channel. A valid range: 0-15.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
bool	[in]	enaScanCh	Enable/disable the selected scan channel by setting this parameter to true/false respectively.
bool	[in]	enaPin	Enable/disable the pin assigned to the channel selected by `chIdx`.

Use this function to enable/disable a selected LESENSE scan channel and the pin assigned to it.

Note

- Users can enable/disable scan channels and the channel pin with the LESENSE_ChannelConfig() function, but only with a significant overhead. This simple function controls these parameters after the channel has been configured.

LESENSE_ChannelEnableMask

```
void LESENSE_ChannelEnableMask (uint16_t chMask, uint16_t pinMask)
```

Enable/disable LESENSE scan channel and the pin assigned to it.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	chMask	Set the corresponding bit to 1 to enable, 0 to disable the selected scan channel.
uint16_t	[in]	pinMask	Set the corresponding bit to 1 to enable, 0 to disable the pin on selected channel.

Use this function to enable/disable LESENSE scan channels and the pins assigned to them using a mask.

Note

- Users can enable/disable scan channels and channel pins by using the LESENSE_ChannelAllConfig() function, but only with a significant overhead. This simple function controls these parameters after the channel has been configured.

LESENSE_ChannelTimingSet

```
void LESENSE_ChannelTimingSet (uint8_t chIdx, uint8_t exTime, uint8_t sampleDelay, uint16_t measDelay)
```

Set LESENSE channel timing parameters.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	chIdx	An identifier of the scan channel. A valid range is 0-15.
uint8_t	[in]	exTime	An excitation time on chIdx. The excitation will last exTime+1 excitation clock cycles. A valid range is 0-63 (6 bits).
uint8_t	[in]	sampleDelay	Sample delay on chIdx. Sampling will occur after sampleDelay+1 sample clock cycles. A valid range is 0-127 (7 bits).

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
uint16_t	[in]	measDelay	A measure delay on chIdx. Sensor measuring is delayed for measDelay+1 excitation clock cycles. A valid range is 0-127 (7 bits).

Use this function to set timing parameters on a selected LESENSE channel.

Note

- Users can configure the channel timing parameters with the LESENSE_ChannelConfig() function, but only with a significant overhead. This simple function controls these parameters after the channel has been configured.

LESENSE_ChannelThresSet

```
void LESENSE_ChannelThresSet (uint8_t chIdx, uint16_t acmpThres, uint16_t cntThres)
```

Set LESENSE channel threshold parameters.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	chIdx	An identifier of the scan channel. A valid range is 0-15.
uint16_t	[in]	acmpThres	ACMP threshold. • If perCtrl.dacCh0Data or perCtrl.dacCh1Data is set to lesenseDACIfData, acmpThres defines the 12-bit DAC data in the corresponding data register of the DAC interface (DACn_CH0DATA and DACn_CH1DATA). In this case, the valid range is 0-4095 (12 bits). • If perCtrl.dacCh0Data or perCtrl.dacCh1Data is set to lesenseACMPThres, acmpThres defines the 6-bit Vdd scaling factor of ACMP negative input (VDDLEVEL in ACMP_INPUTSEL register). In this case, the valid range is 0-63 (6 bits).
uint16_t	[in]	cntThres	A decision threshold for counter comparison. A valid range is 0-65535 (16 bits).

Use this function to set threshold parameters on a selected LESENSE channel.

Note

- Users can configure the channel threshold parameters with the LESENSE_ChannelConfig() function, but only with a significant overhead. This simple function serves controls these parameters after the channel has been configured.

LESENSE_ChannelSlidingWindow

```
void LESENSE_ChannelSlidingWindow (uint8_t chIdx, uint32_t windowSize, uint32_t initValue)
```

Configure a Sliding Window evaluation mode for a specific channel.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
uint8_t	[in]	chIdx	An identifier of the scan channel. A valid range is 0-15.
uint32_t	[in]	windowSize	A window size to be used on all channels.
uint32_t	[in]	initValue	The initial sensor value for the channel.

This function will configure the evaluation mode, the initial sensor measurement (COMPTHRES), and the window size. For other channel-related configuration, see the LESENSE_ChannelConfig() function.

Warnings

- Note that the step size and window size configuration are global to all LESENSE channels and use the same register field in the hardware. This means that any windowSize configuration passed to this function will apply for all channels and override all other stepSize/windowSize configurations.

LESENSE_ChannelStepDetection

```
void LESENSE_ChannelStepDetection (uint8_t chIdx, uint32_t stepSize, uint32_t initValue)
```

Configure the step detection evaluation mode for a specific channel.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	chIdx	An identifier of the scan channel. A valid range is 0-15.
uint32_t	[in]	stepSize	A step size to be used on all channels.
uint32_t	[in]	initValue	The initial sensor value for the channel.

This function will configure the evaluation mode, the initial sensor measurement (COMPTHRES) and the window size. For other channel-related configuration, see the LESENSE_ChannelConfig() function.

Warnings

- Note that the step size and window size configuration are global to all LESENSE channels and use the same register field in the hardware. This means that any stepSize configuration passed to this function will apply for all channels and override all other stepSize/windowSize configurations.

LESENSE_WindowSizeSet

```
void LESENSE_WindowSizeSet (uint32_t windowSize)
```

Set the window size for all LESENSE channels.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	windowSize	The window size to use for all channels.

The window size is used by all channels that are configured as lesenseEvalModeSlidingWindow.

Warnings

- The window size configuration is using the same register field as the step detection size. As a result, the window size configuration will have an effect on channels configured with the lesenseEvalModeStepDetection evaluation mode as well.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_StepSizeSet

```
void LESENSE_StepSizeSet (uint32_t stepSize)
```

Set the step size for all LESENSE channels.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	stepSize	The step size to use for all channels.

The step size is configured using the same register field as used to configure window size. Therefore, calling this function will overwrite any previously configured window size as done by the LESENSE_WindowSizeSet() function.

LESENSE_DecoderStateAllConfig

```
void LESENSE_DecoderStateAllConfig (const LESENSE_DecStAll_TypeDef * confDecStAll)
```

Configure all LESENSE decoder states.

Parameters

Type	Direction	Argument Name	Description
const LESENSE_DecStAll_TypeDef *	[in]	confDecStAll	A configuration structure for all (16 or 32) LESENSE decoder states.

This function configures all the decoder states of the LESENSE interface. See the configuration parameter type definition (LESENSE_DecStAll_TypeDef) for more details.

Note

- Decoder states can be configured individually using LESENSE_DecoderStateConfig() function. Starting from Series 2 Config 3 (xG23 and higher), this function configures a transition ARC instead of a decoder state.

LESENSE_DecoderStateConfig

```
void LESENSE_DecoderStateConfig (const LESENSE_DecStDesc_TypeDef * confDecSt, uint32_t decSt)
```

Configure a single LESENSE decoder state.

Parameters

Type	Direction	Argument Name	Description
const LESENSE_DecStDesc_TypeDef *	[in]	confDecSt	A configuration structure for a single LESENSE decoder state.
uint32_t	[in]	decSt	A decoder state index to configure (0-15) or (0-31) depending on the device.

Copyright © 2026 Silicon Laboratories. All rights reserved.

This function configures a single decoder state of the LESENSE interface. See the configuration parameter type definition (LESENSE_DecStDesc_TypeDef) for more details.

Note

- Starting from Series 2 Config 3 (xG23 and higher), this function configures a transition ARC instead of a decoder state.

LESENSE_DecoderStateGet

```
uint32_t LESENSE_DecoderStateGet (void )
```

Get the current state of the LESENSE decoder.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- This function returns the value of the LESENSE_DECSTATE register that represents the current state of the LESENSE decoder.

LESENSE_DecoderPrsOut

```
void LESENSE_DecoderPrsOut (bool enable, uint32_t decMask, uint32_t decVal)
```

Enable or disable the PRS output from the LESENSE decoder.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	Enable/disable the PRS output from the LESENSE decoder. True to enable and false to disable.
uint32_t	[in]	decMask	A decoder state compare value mask.
uint32_t	[in]	decVal	A decoder state comparison value.

LESENSE_ScanStart

```
void LESENSE_ScanStart (void )
```

Start scanning sensors.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

Copyright © 2026 Silicon Laboratories. All rights reserved.

This function will wait for any pending previous write operation to the CMD register to complete before accessing the CMD register. It will also wait for the write operation to the CMD register to complete before returning. Each write operation to the CMD register may take up to 3 LF clock cycles, so expect some delay. The user may implement a separate function to write multiple command bits in the CMD register in one single operation to optimize an application.

LESENSE_ScanStop

```
void LESENSE_ScanStop (void )
```

Stop scanning sensors.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function will wait for any pending previous write operation to the CMD register to complete before accessing the CMD register. It will also wait for the write operation to the CMD register to complete before returning. Each write operation to the CMD register may take up to 3 LF clock cycles, so the user should expect some delay. The user may implement a separate function to write multiple command bits in the CMD register in one single operation in order to optimize an application.
- If issued during a scan, the command takes effect after scan completion.

LESENSE_DecoderStart

```
void LESENSE_DecoderStart (void )
```

Start the LESENSE decoder.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function will wait for any pending previous write operation to the CMD register to complete before accessing the CMD register. It will also wait for the write operation to the CMD register to complete before returning. Each write operation to the CMD register may take up to 3 LF clock cycles, so expect some delay. The user may implement a separate function to write multiple command bits in the CMD register in one single operation to optimize an application.

LESENSE_ResultBufferClear

```
void LESENSE_ResultBufferClear (void )
```

Clear the result buffer.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function will wait for any pending previous write operation to the CMD register to complete before accessing the CMD register. It will also wait for the write operation to the CMD register to complete before returning. Each write operation to the CMD register may take up to 3 LF clock cycles, so expect some delay. The user may implement a separate function to write multiple command bits in the CMD register in one single operation to optimize an application.

LESENSE_Reset

```
void LESENSE_Reset (void )
```

Reset the LESENSE module.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Use this function to reset LESENSE registers.

Note

- Resetting LESENSE registers is required in each reset or power-on cycle to configure the default values of the RAM mapped LESENSE registers. LESENSE_Reset() can be called on initialization by setting the `reqReset` parameter to true in LESENSE_Init(). Starting from Series 2 Config 3 (xG23 and higher), this function leaves LESENSE in the disabled state.

LESENSE_DecoderStop

```
void LESENSE_DecoderStop (void )
```

Stop LESENSE decoder.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Disables LESENSE decoder by setting the command to LESENSE_DECCTRL register.

LESENSE_StatusGet

```
uint32_t LESENSE_StatusGet (void )
```

Get the current status of LESENSE.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Returns the value of the LESENSE_STATUS register that contains the OR combination of the following status bits for EFR series 0/1:
  - LESENSE_STATUS_BUFDATAV - Result data valid. Set when data is available in result buffer. Cleared when buffer is empty.
  - LESENSE_STATUS_BUFFULL - Result buffer full. Set when result buffer is full.
  - LESENSE_STATUS_BUFHALFFULL - Result buffer half full. Set when result buffer is half full.
  - LESENSE_STATUS_RUNNING - LESENSE is active.
  - LESENSE_STATUS_SCANACTIVE - LESENSE is currently interfacing sensors.

  The OR combination of the following status bits for EFR series 2:
  - LESENSE_STATUS_RESFIFOV - Result Fifo valid. Set when data is available in result Fifo. Cleared when Fifo is empty.
  - LESENSE_STATUS_RESFIFOFULL - Result Fifo full. Set when result Fifo is full.
  - LESENSE_STATUS_RUNNING - LESENSE is active.
  - LESENSE_STATUS_SCANACTIVE - LESENSE is currently interfacing sensors.
  - LESENSE_STATUS_FLUSHING - Fifo flushing
  - LESENSE_STATUS_READBUSY - Fifo Read busy

LESENSE_StatusWait

```
void LESENSE_StatusWait (uint32_t flag)
```

Wait until status of LESENSE is equal to what was requested.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flag	The OR combination of the following status bits for EFR series 0/1: • LESENSE_STATUS_BUFDATAV - Result data valid. Set when data is available in result buffer. Cleared when buffer is empty. • LESENSE_STATUS_BUFHALFFULL - Result buffer half full. Set when result buffer is half full. • LESENSE_STATUS_BUFFULL - Result buffer full. Set when result buffer is full. • LESENSE_STATUS_RUNNING - LESENSE is active. • LESENSE_STATUS_SCANACTIVE - LESENSE is currently interfacing sensors. • LESENSE_STATUS_DACACTIVE - The DAC interface is currently active. The OR combination of the following status bits for EFR series 2: • LESENSE_STATUS_RESFIFOV - Result FIFO valid. Set when data is available in result FIFO. Cleared when FIFO is empty. • LESENSE_STATUS_RESFIFOFULL - Result FIFO full. Set when result FIFO is full. • LESENSE_STATUS_RUNNING - LESENSE is active. • LESENSE_STATUS_SCANACTIVE - LESENSE is currently interfacing sensors. • LESENSE_STATUS_FLUSHING - FIFO flushing • LESENSE_STATUS_READBUSY - FIFO Read busy

Copyright © 2026 Silicon Laboratories. All rights reserved.

Polls LESENSE_STATUS register and waits until requested combination of flags are set.

LESENSE_ChannelActiveGet

```
uint32_t LESENSE_ChannelActiveGet (void )
```

Get the currently active channel index.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Returns the value of the LESENSE_CHINDEX register that contains the index of currently active channel (0-15).

LESENSE_ScanResultGet

```
uint32_t LESENSE_ScanResultGet (void )
```

Get the latest scan comparison result (1 bit / channel).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Returns the value of the LESENSE_SCANRES register that contains the comparison result of last scan on all channels. Bit x is set if a comparison triggered on channel x, which means that LESENSE counter met the comparison criteria set in LESENSE_CHx_EVAL by COMPMODE and CNTTHRES.

LESENSE_ScanResultDataGet

```
uint32_t LESENSE_ScanResultDataGet (void )
```

Get the oldest unread data from the result buffer.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Make sure that the STORERES bit is set in LESENSE_CHx_EVAL, or the STRSCANRES bit is set in LESENSE_CTRL; otherwise, returns the undefined value.

Returns

-

Copyright © 2026 Silicon Laboratories. All rights reserved.

Returns the value of LESENSE_RESDATA register that contains the oldest unread counter result from result buffer.

LESENSE_SensorStateGet

```
uint32_t LESENSE_SensorStateGet (void )
```

Get the current state of the LESENSE sensor.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Returns the value of LESENSE_SENSORSTATE register that represents the current state of the LESENSE sensor.

LESENSE_IntClear

```
void LESENSE_IntClear (uint32_t flags)
```

Clear one or more pending LESENSE interrupts.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	flags	Pending LESENSE interrupt sources to clear. Use a set of interrupt flags OR-ed together to clear multiple interrupt sources of LESENSE module (LESENSE_IF_nnn).

LESENSE_IntEnable

```
void LESENSE_IntEnable (uint32_t flags)
```

Enable one or more LESENSE interrupts.

Parameters

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

LESENSE_IntDisable

```
void LESENSE_IntDisable (uint32_t flags)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Disable one or more LESENSE interrupts.

Parameters

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

LESENSE_IntSet

```
void LESENSE_IntSet (uint32_t flags)
```

Set one or more pending LESENSE interrupts from SW.

Parameters

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

LESENSE_IntGet

```
uint32_t LESENSE_IntGet (void )
```

Get pending LESENSE interrupt flags.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

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

Returns

- Pending LESENSE interrupt sources. The OR combination of valid interrupt flags of the LESENSE module (LESENSE_IF_nnn).

LESENSE_IntGetEnabled

```
uint32_t LESENSE_IntGetEnabled (void )
```

Get enabled and pending LESENSE interrupt flags.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
void	N/A		

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 LESENSE interrupt sources. Return value is the bitwise AND combination of
  - the OR combination of enabled interrupt sources in LESENSE_IEN_nnn register (LESENSE_IEN_nnn) and
  - the OR combination of valid interrupt flags of LESENSE module (LESENSE_IF_nnn).

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_CoreCtrlDesc_TypeDef

Core control (LESENSE_CTRL/CFG) descriptor structure.

Public Attributes

LESENSE_ScanMode_TypeDef	scanStart Select scan start mode to control how the scan start is being triggered.
LESENSE_PRSSel_TypeDef	prsSel Select PRS source for scan start if scanMode is set to lesensePrsPulse.
LESENSE_ScanConfSel_TypeDef	scanConfSel Select scan configuration register usage strategy.
bool	invACMP0 Set to true to invert ACMP0 output.
bool	invACMP1 Set to true to invert ACMP1 output.
bool	dualSample Set to true to sample both ACMPs simultaneously.
bool	storeScanRes Set to true in order to store SCANRES in the RAM (accessible via RESDATA) after each scan.
uint8_t	fifoTrigLevel Select result FIFO interrupt and DMA trigger level.
LESENSE_DMAWakeUp_TypeDef	wakeupOnDMA Configure trigger condition for the DMA wakeup from EM2.
bool	debugRun Set to true to keep LESENSE running in the debug mode.

Public Attribute Documentation

scanStart

```
LESENSE_ScanMode_TypeDef LESENSE_CoreCtrlDesc_TypeDef::scanStart
```

Select scan start mode to control how the scan start is being triggered.

prsSel

```
LESENSE_PRSSel_TypeDef LESENSE_CoreCtrlDesc_TypeDef::prsSel
```

Select PRS source for scan start if scanMode is set to lesensePrsPulse.

scanConfSel

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
LESENSE_ScanConfSel_TypeDef LESENSE_CoreCtrlDesc_TypeDef::scanConfSel
```

Select scan configuration register usage strategy.

invACMP0

```
bool LESENSE_CoreCtrlDesc_TypeDef::invACMP0
```

Set to true to invert ACMP0 output.

invACMP1

```
bool LESENSE_CoreCtrlDesc_TypeDef::invACMP1
```

Set to true to invert ACMP1 output.

dualSample

```
bool LESENSE_CoreCtrlDesc_TypeDef::dualSample
```

Set to true to sample both ACMPs simultaneously.

storeScanRes

```
bool LESENSE_CoreCtrlDesc_TypeDef::storeScanRes
```

Set to true in order to store SCANRES in the RAM (accessible via RESDATA) after each scan.

fifoTrigLevel

```
uint8_t LESENSE_CoreCtrlDesc_TypeDef::fifoTrigLevel
```

Select result FIFO interrupt and DMA trigger level.

wakeupOnDMA

```
LESENSE_DMAWakeUp_TypeDef LESENSE_CoreCtrlDesc_TypeDef::wakeupOnDMA
```

Configure trigger condition for the DMA wakeup from EM2.

Copyright © 2026 Silicon Laboratories. All rights reserved.

debugRun

```
bool LESENSE_CoreCtrlDesc_TypeDef::debugRun
```

Set to true to keep LESENSE running in the debug mode.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_TimeCtrlDesc_TypeDef

LESENSE timing control descriptor structure.

Public Attributes

uint8_t	startDelay Set number of LFACLK cycles to delay sensor interaction on each channel.
bool	delayAuxStartup Set to true do delay startup of AUXHFRCO until the system enters excite phase.

Public Attribute Documentation

startDelay

```
uint8_t LESENSE_TimeCtrlDesc_TypeDef::startDelay
```

Set number of LFACLK cycles to delay sensor interaction on each channel.

Valid range: 0-3 (2 bit).

delayAuxStartup

```
bool LESENSE_TimeCtrlDesc_TypeDef::delayAuxStartup
```

Set to true do delay startup of AUXHFRCO until the system enters excite phase.

This will reduce the time AUXHFRCO is enabled and reduce power usage.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_PerCtrlDesc_TypeDef

LESENSE peripheral control descriptor structure.

Public Attributes

LESENSE_ControlDACData_TypeDef	dacCh0Data Configure DAC channel 0 data control.
LESENSE_ControlACMP_TypeDef	acmp0Mode Configure how LESENSE controls ACMP 0.
LESENSE_ControlACMP_TypeDef	acmp1Mode Configure how LESENSE controls ACMP 1.
bool	dacScan When set to true the DAC is only enabled once for each scan.
bool	dacStartupHalf When set to true the DAC is started a half clock cycle before sensor interaction starts.

Public Attribute Documentation

dacCh0Data

```
LESENSE_ControlDACData_TypeDef LESENSE_PerCtrlDesc_TypeDef::dacCh0Data
```

Configure DAC channel 0 data control.

acmp0Mode

```
LESENSE_ControlACMP_TypeDef LESENSE_PerCtrlDesc_TypeDef::acmp0Mode
```

Configure how LESENSE controls ACMP 0.

acmp1Mode

```
LESENSE_ControlACMP_TypeDef LESENSE_PerCtrlDesc_TypeDef::acmp1Mode
```

Configure how LESENSE controls ACMP 1.

dacScan

```
bool LESENSE_PerCtrlDesc_TypeDef::dacScan
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

When set to true the DAC is only enabled once for each scan.

When set to false the DAC is enabled before every channel measurement.

dacStartupHalf

```
bool LESENSE_PerCtrlDesc_TypeDef::dacStartupHalf
```

When set to true the DAC is started a half clock cycle before sensor interaction starts.

When set to false, a full clock cycle is used.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_DecCtrlDesc_TypeDef

LESENSE decoder control descriptor structure.

Public Attributes

bool	chkState Set to enable decoder to check the present state in addition to the states defined in TCONF.
bool	intMap When set, a transition from state x in decoder will set the interrupt flag CHx.
bool	hystPRS0 Set to enable hysteresis in decoder for suppressing the changes on PRS channel 0.
bool	hystPRS1 Set to enable hysteresis in decoder for suppressing the changes on PRS channel 1.
bool	hystPRS2 Set to enable hysteresis in decoder for suppressing the changes on PRS channel 2.
bool	hystIRQ Set to enable hysteresis in decoder for suppressing the interrupt requests.
bool	prsCount Set to enable count mode on decoder PRS channels 0 and 1 to produce outputs which can be used by a PCNT to count up or down.

Public Attribute Documentation

chkState

```
bool LESENSE_DecCtrlDesc_TypeDef::chkState
```

Set to enable decoder to check the present state in addition to the states defined in TCONF.

intMap

```
bool LESENSE_DecCtrlDesc_TypeDef::intMap
```

When set, a transition from state x in decoder will set the interrupt flag CHx.

hystPRS0

```
bool LESENSE_DecCtrlDesc_TypeDef::hystPRS0
```

Set to enable hysteresis in decoder for suppressing the changes on PRS channel 0.

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
bool LESENSE_DecCtrlDesc_TypeDef::hystPRS1
```

Set to enable hysteresis in decoder for suppressing the changes on PRS channel 1.

hystPRS2

```
bool LESENSE_DecCtrlDesc_TypeDef::hystPRS2
```

Set to enable hysteresis in decoder for suppressing the changes on PRS channel 2.

hystIRQ

```
bool LESENSE_DecCtrlDesc_TypeDef::hystIRQ
```

Set to enable hysteresis in decoder for suppressing the interrupt requests.

prsCount

```
bool LESENSE_DecCtrlDesc_TypeDef::prsCount
```

Set to enable count mode on decoder PRS channels 0 and 1 to produce outputs which can be used by a PCNT to count up or down.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_Init_TypeDef

LESENSE module initialization structure.

Public Attributes

LESENSE_CoreCtrlDesc_TypeDef	coreCtrl LESENSE core configuration parameters.
LESENSE_TimeCtrlDesc_TypeDef	timeCtrl LESENSE timing configuration parameters.
LESENSE_PerCtrlDesc_TypeDef	perCtrl LESENSE peripheral configuration parameters.
LESENSE_DecCtrlDesc_TypeDef	decCtrl LESENSE decoder configuration parameters.

Public Attribute Documentation

coreCtrl

```
LESENSE_CoreCtrlDesc_TypeDef LESENSE_Init_TypeDef::coreCtrl
```

LESENSE core configuration parameters.

timeCtrl

```
LESENSE_TimeCtrlDesc_TypeDef LESENSE_Init_TypeDef::timeCtrl
```

LESENSE timing configuration parameters.

perCtrl

```
LESENSE_PerCtrlDesc_TypeDef LESENSE_Init_TypeDef::perCtrl
```

LESENSE peripheral configuration parameters.

decCtrl

```
LESENSE_DecCtrlDesc_TypeDef LESENSE_Init_TypeDef::decCtrl
```

LESENSE decoder configuration parameters.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_ChDesc_TypeDef

Channel descriptor structure.

Public Attributes

bool	enaScanCh Set to enable scan channel CHx.
bool	enaPin Set to enable CHx pin.
bool	enaInt Enable/disable channel interrupts after configuring all the sensor channel parameters.
LESENSE_ChPinExMode_TypeDef	chPinExMode Configure channel pin mode for the excitation phase of the scan sequence.
LESENSE_ChPinIdleMode_TypeDef	chPinIdleMode Configure channel pin idle setup in LESENSE idle phase.
bool	useAltEx Set to use alternate excite pin for excitation.
bool	shiftRes Set to enable result from this channel being shifted into the decoder register.
bool	invRes Set to invert result bit stored in the SCANRES register.
LESENSE_StoreSample_TypeDef	storeCntRes Set to store counter value in the RAM (accessible via RESDATA) and make the comparison result available in the SCANRES register.
LESENSE_ChClk_TypeDef	exClk Select clock used for the excitation timing.
LESENSE_ChClk_TypeDef	sampleClk Select clock used for the sample delay timing.
uint8_t	exTime Configure the excitation time.
uint8_t	sampleDelay Configure the sample delay.
uint16_t	measDelay Configure the measure delay.
uint16_t	acmpThres Configure the ACMP threshold or the DAC data.
LESENSE_ChSampleMode_TypeDef	sampleMode Select if the ACMP output, the ADC output or the counter output should be used in comparison.
LESENSE_ChIntMode_TypeDef	intMode Configure the interrupt generation mode for the CHx interrupt flag.

Copyright © 2026 Silicon Laboratories. All rights reserved.

uint16_t	cntThres Configure the decision threshold for the sensor data comparison.
LESENSE_ChCompMode_TypeDef	compMode Select the mode for counter comparison.
LESENSE_ChEvalMode_TypeDef	evalMode Select the sensor evaluation mode.
uint8_t	offset Offset for IADC/ACMP interaction.

Public Attribute Documentation

enaScanCh

```
bool LESENSE_ChDesc_TypeDef::enaScanCh
```

Set to enable scan channel CHx.

enaPin

```
bool LESENSE_ChDesc_TypeDef::enaPin
```

Set to enable CHx pin.

enaInt

```
bool LESENSE_ChDesc_TypeDef::enaInt
```

Enable/disable channel interrupts after configuring all the sensor channel parameters.

chPinExMode

```
LESENSE_ChPinExMode_TypeDef LESENSE_ChDesc_TypeDef::chPinExMode
```

Configure channel pin mode for the excitation phase of the scan sequence.

Note: OPAOUT is only available on channels 2, 3, 4, and 5.

chPinIdleMode

```
LESENSE_ChPinIdleMode_TypeDef LESENSE_ChDesc_TypeDef::chPinIdleMode
```

Configure channel pin idle setup in LESENSE idle phase.

Copyright © 2026 Silicon Laboratories. All rights reserved.

useAltEx

```
bool LESENSE_ChDesc_TypeDef::useAltEx
```

Set to use alternate excite pin for excitation.

shiftRes

```
bool LESENSE_ChDesc_TypeDef::shiftRes
```

Set to enable result from this channel being shifted into the decoder register.

invRes

```
bool LESENSE_ChDesc_TypeDef::invRes
```

Set to invert result bit stored in the SCANRES register.

storeCntRes

```
LESENSE_StoreSample_TypeDef LESENSE_ChDesc_TypeDef::storeCntRes
```

Set to store counter value in the RAM (accessible via RESDATA) and make the comparison result available in the SCANRES register.

exClk

```
LESENSE_ChClk_TypeDef LESENSE_ChDesc_TypeDef::exClk
```

Select clock used for the excitation timing.

sampleClk

```
LESENSE_ChClk_TypeDef LESENSE_ChDesc_TypeDef::sampleClk
```

Select clock used for the sample delay timing.

exTime

```
uint8_t LESENSE_ChDesc_TypeDef::exTime
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Configure the excitation time.

Excitation will last exTime+1 excitation clock cycles. Valid range: 0-63 (6 bits).

sampleDelay

```
uint8_t LESENSE_ChDesc_TypeDef::sampleDelay
```

Configure the sample delay.

Sampling will occur after sampleDelay+1 sample clock cycles. Valid range: 0-127 (7 bits) or 0-255 (8 bits) depending on device.

measDelay

```
uint16_t LESENSE_ChDesc_TypeDef::measDelay
```

Configure the measure delay.

Sensor measuring is delayed for measDelay excitation clock cycles. Valid range: 0-127 (7 bits) or 0-1023 (10 bits) depending on device.

acmpThres

```
uint16_t LESENSE_ChDesc_TypeDef::acmpThres
```

Configure the ACMP threshold or the DAC data.

If perCtrl.dacCh0Data or perCtrl.dacCh1Data is set to lesenseDACIfData, acmpThres defines the 12-bit DAC data in the corresponding data register of DAC interface (DACn_CH0DATA and DACn_CH1DATA). In this case, the valid range is: 0-4095 (12 bits). If perCtrl.dacCh0Data or perCtrl.dacCh1Data is set to lesenseACMPThres, acmpThres defines the 6-bit Vdd scaling factor of ACMP negative input (VDDLEVEL in ACMP_INPUTSEL register). In this case, the valid range is: 0-63 (6 bits).

sampleMode

```
LESENSE_ChSampleMode_TypeDef LESENSE_ChDesc_TypeDef::sampleMode
```

Select if the ACMP output, the ADC output or the counter output should be used in comparison.

intMode

```
LESENSE_ChIntMode_TypeDef LESENSE_ChDesc_TypeDef::intMode
```

Configure the interrupt generation mode for the CHx interrupt flag.

Copyright © 2026 Silicon Laboratories. All rights reserved.

cntThres

```
uint16_t LESENSE_ChDesc_TypeDef::cntThres
```

Configure the decision threshold for the sensor data comparison.

Valid range: 0-65535 (16 bits).

compMode

```
LESENSE_ChCompMode_TypeDef LESENSE_ChDesc_TypeDef::compMode
```

Select the mode for counter comparison.

evalMode

```
LESENSE_ChEvalMode_TypeDef LESENSE_ChDesc_TypeDef::evalMode
```

Select the sensor evaluation mode.

offset

```
uint8_t LESENSE_ChDesc_TypeDef::offset
```

Offset for IADC/ACMP interaction.

ACMP: offset determines which of the port I/O pins on the external override interface to access. IADC: offset determines which of the IADC scanner channels is sampled.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_ChAll_TypeDef

Configuration structure for all the scan channels.

Public Attributes

LESENSE_ChDesc_TypeDef	Ch Channel descriptor for all the LESENSE channels.

Public Attribute Documentation

Ch

```
LESENSE_ChDesc_TypeDef LESENSE_ChAll_TypeDef::Ch[LESENSE_NUM_CHANNELS]
```

Channel descriptor for all the LESENSE channels.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_AltExDesc_TypeDef

Alternate excitation descriptor structure.

Public Attributes

bool	enablePin Configure alternate excitation pins.

Public Attribute Documentation

enablePin

```
bool LESENSE_AltExDesc_TypeDef::enablePin
```

Configure alternate excitation pins.

If set, the corresponding alternate excitation pin/signal is enabled.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_ConfAltEx_TypeDef

Configuration structure for the alternate excitation.

Public Attributes

LESENSE_AltExDesc_TypeDef	AltEx Alternate excitation channel descriptors.

Public Attribute Documentation

AltEx

```
LESENSE_AltExDesc_TypeDef LESENSE_ConfAltEx_TypeDef::AltEx[16]
```

Alternate excitation channel descriptors.

When altExMap==lesenseAltExMapALTEX, only the 8 first descriptors are used. In this mode they describe the configuration of LES_ALTEX0-7 pins. When altExMap==lesenseAltExMapACMP, all 16 descriptors are used. In this mode they describe the configuration of the 16 possible ACMP0-1 excitation channels. Please refer to the user manual for a complete mapping of routing. NOTE: Some parameters in the descriptors are not valid when altExMap==lesenseAltExMapACMP. Refer to the definition of the LESENSE_AltExDesc_TypeDef structure for details regarding which parameters are valid.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_DecStCond_TypeDef

Decoder state condition descriptor structure.

Public Attributes

uint8_t	compVal Configure compare value.
uint8_t	compMask Configure compare mask.
uint8_t	curState Configure index of the current state when evaluation is done.
uint8_t	nextState Configure index of state to be entered if the sensor state equals to compVal.
LESENSE_StTransAct_TypeDef	prsAct Configure which PRS action to perform when the sensor state equals to compVal.
bool	setInt If enabled, interrupt flag is set when sensor state equals to compVal.

Public Attribute Documentation

compVal

```
uint8_t LESENSE_DecStCond_TypeDef::compVal
```

Configure compare value.

State transition is triggered when the sensor state equals to this value. Valid range: 0-15 (4 bits).

compMask

```
uint8_t LESENSE_DecStCond_TypeDef::compMask
```

Configure compare mask.

Set bit X to exclude sensor X from evaluation. Note: decoder can handle sensor inputs from up to 4 sensors; therefore, this mask is 4 bit long.

curState

```
uint8_t LESENSE_DecStCond_TypeDef::curState
```

Configure index of the current state when evaluation is done.

Valid range: 0-15 (4 bits).

Copyright © 2026 Silicon Laboratories. All rights reserved.

nextState

> uint8_t LESENSE_DecStCond_TypeDef::nextState

Configure index of state to be entered if the sensor state equals to compVal.

Valid range: 0-15 (4 bits).

prsAct

> LESENSE_StTransAct_TypeDef LESENSE_DecStCond_TypeDef::prsAct

Configure which PRS action to perform when the sensor state equals to compVal.

setInt

> bool LESENSE_DecStCond_TypeDef::setInt

If enabled, interrupt flag is set when sensor state equals to compVal.

Copyright © 2026 Silicon Laboratories. All rights reserved.

LESENSE_DecStAll_TypeDef

Configuration structure for decoder.

Public Attributes

LESENSE_DecStDesc_TypeDef	St Descriptor of the 64 Arcs on series 2 devices.

Public Attribute Documentation

St

```
LESENSE_DecStDesc_TypeDef LESENSE_DecStAll_TypeDef::St[LESENSE_NUM_ARCS]
```

Descriptor of the 64 Arcs on series 2 devices.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

void LETIMER_Lock(LETIMER_TypeDef *letimer)
Lock LETIMER registers.

void LETIMER_Unlock(LETIMER_TypeDef *letimer)
Unlock LETIMER registers.

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 REP0 times.

Copyright © 2026 Silicon Laboratories. All rights reserved.

letimerRepeatBuffered	Count REP0 times, if REP1 has been written to, it is loaded into REP0 when REP0 is about to be decremented to 0.
letimerRepeatDouble	Run as long as both REP0 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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_Lock

```
void LETIMER_Lock (LETIMER_TypeDef * letimer)
```

Lock LETIMER registers.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to LETIMER peripheral register block.

Note

- When LETIMER registers are locked LETIMER_EN, LETIMER_SWRST, LETIMER_CTRL, LETIMER_CMD, LETIMER_CNT, LETIMER_COMPx, LETIMER_TOP, LETIMER_TOPBUFF, LETIMER_REPx, and PRSMODE registers cannot be written to.

LETIMER_Unlock

```
void LETIMER_Unlock (LETIMER_TypeDef * letimer)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Unlock LETIMER registers.

Parameters

Type	Direction	Argument Name	Description
LETIMER_TypeDef *	[in]	letimer	Pointer to LETIMER peripheral register block.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 REP0 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_RepeatMode_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

Copyright © 2026 Silicon Laboratories. All rights reserved.

bool LETIMER_Init_TypeDef::comp0Top

Load COMP0 register into CNT when counter underflows.

bufTop

bool LETIMER_Init_TypeDef::bufTop

Load COMP1 into COMP0 when REP0 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

topValue

```
uint32_t LETIMER_Init_TypeDef::topValue
```

Top value.

Counter wraps when top value matches counter value is reached.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

enum MSC_DmemMaster_TypeDef {

  mscDmemMasterLDMA = _SYSCFG_DMEM0PORTMAPSEL_LDMAPORTSEL_SHIFT
  mscDmemMasterSRWAES = _SYSCFG_DMEM0PORTMAPSEL_SRWAESPORTSEL_SHIFT
  mscDmemMasterAHBSRW = _SYSCFG_DMEM0PORTMAPSEL_AHBSRWPORTSEL_SHIFT
  mscDmemMasterSRWECA0 = _SYSCFG_DMEM0PORTMAPSEL_SRWECA0PORTSEL_SHIFT
  mscDmemMasterSRWECA1 = _SYSCFG_DMEM0PORTMAPSEL_SRWECA1PORTSEL_SHIFT
  mscDmemMasterMVPAHBDATA0 = _SYSCFG_DMEM0PORTMAPSEL_MVPAHBDATA0PORTSEL_SHIFT
  mscDmemMasterMVPAHBDATA1 = _SYSCFG_DMEM0PORTMAPSEL_MVPAHBDATA1PORTSEL_SHIFT
  mscDmemMasterMVPAHBDATA2 = _SYSCFG_DMEM0PORTMAPSEL_MVPAHBDATA2PORTSEL_SHIFT

}

AHBHOST masters that can use alternate MPAHBRAM ports.

enum MSC_PortPriority_TypeDef {

  mscPortPriorityNone = _MPAHBRAM_CTRL_AHBPORTPRIORITY_NONE
  mscPortPriorityPort0 = _MPAHBRAM_CTRL_AHBPORTPRIORITY_PORT0
  mscPortPriorityPort1 = _MPAHBRAM_CTRL_AHBPORTPRIORITY_PORT1
  mscPortPriorityPort2 = _MPAHBRAM_CTRL_AHBPORTPRIORITY_PORT2
  mscPortPriorityPort3 = _MPAHBRAM_CTRL_AHBPORTPRIORITY_PORT3

}

AHB port given priority.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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).
void	MSC_DmemPortMapSet(MSC_DmemMaster_TypeDef master, uint8_t port) Set MPAHBRAM port to use to access DMEM.
void	MSC_PortSetPriority(MSC_PortPriority_TypeDef portPriority) Set MPAHBRAM port priority for arbitration when multiple concurrent transactions to DMEM.
MSC_PortPriority_TypeDef	MSC_PortGetCurrentPriority(void) Get MPAHBRAM port arbitration priority selection.
SL_RAMFUNC_DECLARATORMSC_Status_TypeDef	MSC_MassErase(void) Erase the entire Flash in one operation.
MSC_RAMFUNC_DECLARATOR MSC_Status_TypeDef	MSC_ErasePage(uint32_t *startAddress) Erases a page in flash memory.
MSC_RAMFUNC_DECLARATOR 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)

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

MSC_DmemMaster_TypeDef

> MSC_DmemMaster_TypeDef

AHBHOST masters that can use alternate MPAHBRAM ports.

Enumerator	
mscDmemMasterLDMA	
mscDmemMasterSRWAES	
mscDmemMasterAHBSRW	
mscDmemMasterSRWECA0	
mscDmemMasterSRWECA1	

Copyright © 2026 Silicon Laboratories. All rights reserved.

mscDmemMasterMVPAHBDATA0	
mscDmemMasterMVPAHBDATA1	
mscDmemMasterMVPAHBDATA2	

MSC_PortPriority_TypeDef

```
MSC_PortPriority_TypeDef
```

AHB port given priority.

Enumerator	
mscPortPriorityNone	
mscPortPriorityPort0	
mscPortPriorityPort1	
mscPortPriorityPort2	
mscPortPriorityPort3	

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 )
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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).

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_DmemPortMapSet

```
void MSC_DmemPortMapSet (MSC_DmemMaster_TypeDef master, uint8_t port)
```

Set MPAHBRAM port to use to access DMEM.

Parameters

Type	Direction	Argument Name	Description
MSC_DmemMaster_TypeDef	[in]	master	AHBHOST master to be configured.
uint8_t	[in]	port	AHBHOST slave port to use.

Copyright © 2026 Silicon Laboratories. All rights reserved.

This function configures which MPAHBRAM slave port is used to access DMEM. Depending on the use case, it might improve performance by spreading the load over the N ports (N is usually 2 or 4), instead of starving because a port is used by another master.

MSC_PortSetPriority

```
void MSC_PortSetPriority (MSC_PortPriority_TypeDef portPriority)
```

Set MPAHBRAM port priority for arbitration when multiple concurrent transactions to DMEM.

Parameters

Type	Direction	Argument Name	Description
MSC_PortPriority_TypeDef	[in]	portPriority	AHBHOST slave port having elevated priority.

This function configures which MPAHBRAM slave port will have priority. The AHB port arbitration default scheme, round-robin arbitration, is selected when portPriority == mscPortPriorityNone.

Note

- Doing this can potentially starve the others AHB port(s).

MSC_PortGetCurrentPriority

```
MSC_PortPriority_TypeDef MSC_PortGetCurrentPriority (void )
```

Get MPAHBRAM port arbitration priority selection.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function returns the AHBHOST slave with raised priority.

Returns

- Returns the AHBHOST slave port given priority or none.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 divisable 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 )
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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		

mscEccReadWriteExistingPio

```
static void mscEccReadWriteExistingPio (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.

mscEccBankInit

```
static void mscEccBankInit (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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

PCNT - Pulse Counter

PCNT - Pulse Counter

Pulse Counter (PCNT) Peripheral API.

This module contains functions to control the PCNT peripheral of Silicon Labs 32-bit MCUs and SoCs. The PCNT decodes incoming pulses. The module has a quadrature mode which may be used to decode the speed and direction of a mechanical shaft.

Modules

PCNT_Init_TypeDef

PCNT_Filter_TypeDef

Enumerations

enum PCNT_Mode_TypeDef {

pcntModeDisable = PCNT_MODE_DISABLE
pcntModeOvsSingle = _PCNT_CFG_MODE_OVSSINGLE
pcntModeExtSingle = _PCNT_CFG_MODE_EXTCLKSINGLE
pcntModeExtQuad = _PCNT_CFG_MODE_EXTCLKQUAD
pcntModeOvsQuad1 = _PCNT_CFG_MODE_OVSQUAD1X
pcntModeOvsQuad2 = _PCNT_CFG_MODE_OVSQUAD2X
pcntModeOvsQuad4 = _PCNT_CFG_MODE_OVSQUAD4X

}

Mode selection.

enum PCNT_CntEvent_TypeDef {

pcntCntEventBoth = _PCNT_CTRL_CNTEV_BOTH
pcntCntEventUp = _PCNT_CTRL_CNTEV_UP
pcntCntEventDown = _PCNT_CTRL_CNTEV_DOWN
pcntCntEventNone = PCNT_CNT_EVENT_NONE

}

Counter event selection.

enum PCNT_PRSInput_TypeDef {

pcntPRSInputS0 = 0
pcntPRSInputS1 = 1

}

PRS inputs of PCNT.

Typedefs

typedef unsigned int PCNT_PRSSel_TypeDef

PRS sources for `s0PRS` and `s1PRS`.

Variables

Copyright © 2026 Silicon Laboratories. All rights reserved.

PCNT_CntEvent_TypeDef	initCntEvent
PCNT_CntEvent_TypeDef	initAuxCntEvent

Functions

void	PCNT_CounterReset(PCNT_TypeDef *pcnt) Reset PCNT counters and TOP register.
void	PCNT_Enable(PCNT_TypeDef *pcnt, PCNT_Mode_TypeDef mode) Set PCNT operational mode.
bool	PCNT_IsEnabled(PCNT_TypeDef *pcnt) Returns if the PCNT module is enabled or not.
void	PCNT_CounterTopSet(PCNT_TypeDef *pcnt, uint32_t count, uint32_t top) Set the counter and top values.
void	PCNT_PRSInputEnable(PCNT_TypeDef *pcnt, PCNT_PRSInput_TypeDef prsInput, bool enable) Enable/disable the selected PRS input of PCNT.
void	PCNT_Init(PCNT_TypeDef *pcnt, const PCNT_Init_TypeDef *init) Initialize the pulse counter.
void	PCNT_Reset(PCNT_TypeDef *pcnt) Reset PCNT to the same state that it was in after a hardware reset.
void	PCNT_FilterConfiguration(PCNT_TypeDef *pcnt, const PCNT_Filter_TypeDef *config, bool enable) Set the filter configuration.
void	PCNT_TopBufferSet(PCNT_TypeDef *pcnt, uint32_t val) Set top buffer value.
void	PCNT_TopSet(PCNT_TypeDef *pcnt, uint32_t val) Set the top value.
uint32_t	PCNT_CounterGet(PCNT_TypeDef *pcnt) Get the pulse counter value.
uint32_t	PCNT_AuxCounterGet(PCNT_TypeDef *pcnt) Get the auxiliary counter value.
void	PCNT_CounterSet(PCNT_TypeDef *pcnt, uint32_t count) Set a counter value.
void	PCNT_IntClear(PCNT_TypeDef *pcnt, uint32_t flags) Clear one or more pending PCNT interrupts.
void	PCNT_IntDisable(PCNT_TypeDef *pcnt, uint32_t flags) Disable one or more PCNT interrupts.
void	PCNT_IntEnable(PCNT_TypeDef *pcnt, uint32_t flags) Enable one or more PCNT interrupts.
uint32_t	PCNT_IntGet(PCNT_TypeDef *pcnt) Get pending PCNT interrupt flags.

Copyright © 2026 Silicon Laboratories. All rights reserved.

uint32_t PCNT_IntGetEnabled(PCNT_TypeDef *pcnt)
Get enabled and pending PCNT interrupt flags.

void PCNT_IntSet(PCNT_TypeDef *pcnt, uint32_t flags)
Set one or more pending PCNT interrupts from SW.

void PCNT_Lock(PCNT_TypeDef *pcnt)
Lock PCNT registers.

void PCNT_Unlock(PCNT_TypeDef *pcnt)
Unlock PCNT registers.

uint32_t PCNT_TopBufferGet(PCNT_TypeDef *pcnt)
Get the pulse counter top buffer value.

uint32_t PCNT_TopGet(PCNT_TypeDef *pcnt)
Get the pulse counter top value.

void PCNT_Sync(PCNT_TypeDef *pcnt, uint32_t mask)
Wait for an ongoing sync of register(s) to low-frequency domain to complete.

void PCNT_StartMainCnt(PCNT_TypeDef *pcnt)
Start the main PCNT counter.

void PCNT_StopMainCnt(PCNT_TypeDef *pcnt)
Stop the main PCNT counter.

void PCNT_StartAuxCnt(PCNT_TypeDef *pcnt)
Start the auxiliary PCNT counter.

void PCNT_StopAuxCnt(PCNT_TypeDef *pcnt)
Stop the auxiliary PCNT counter.

Macros

#define PCNT0_CNT_SIZE (16)
PCNT0 Counter register size.

#define PCNT_MODE_DISABLE 0xFF
PCNT mode disable.

#define PCNT_CNT_EVENT_NONE 0xFF
PCNT count event is none.

#define DEFAULT_DEBUG_HALT true,
Default Debug.

#define DEFAULT_MODE pcntModeDisable,
Default Mode.

#define DEFAULT_HYST false,
Default Hysteresis.

#define DEFAULT_CDIR true,
Default counter direction.

#define DEFAULT_CNTEV pcntCntEventUp,
Default count event.

Copyright © 2026 Silicon Laboratories. All rights reserved.

#define DEFAULT_AUXCNTEV pcntCntEventNone,
Default auxiliary count event.

#define DEFAULT_PRS_CH 0u,
Default selected PRS channel as S0IN and S1IN.

#define PCNT_INIT_DEFAULT undefined
Default configuration for PCNT initialization structure.

#define PCNT_FILTER_DEFAULT undefined
Default configuration for PCNT initialization structure.

Enumeration Documentation

PCNT_Mode_TypeDef

> PCNT_Mode_TypeDef

Mode selection.

Enumerator	
pcntModeDisable	Disable pulse counter.
pcntModeOvsSingle	Single input LFACLK oversampling mode (available in EM0-EM2).
pcntModeExtSingle	Externally clocked single input counter mode (available in EM0-EM3).
pcntModeExtQuad	Externally clocked quadrature decoder mode (available in EM0-EM3).
pcntModeOvsQuad1	LFACLK oversampling quadrature decoder 1X mode (available in EM0-EM2).
pcntModeOvsQuad2	LFACLK oversampling quadrature decoder 2X mode (available in EM0-EM2).
pcntModeOvsQuad4	LFACLK oversampling quadrature decoder 4X mode (available in EM0-EM2).

PCNT_CntEvent_TypeDef

> PCNT_CntEvent_TypeDef

Counter event selection.

Note: unshifted values are being used for enumeration because multiple configuration structure members use this type definition.

Enumerator	
pcntCntEventBoth	Counts up on up-count and down on down-count events.
pcntCntEventUp	Only counts up on up-count events.
pcntCntEventDown	Only counts down on down-count events.
pcntCntEventNone	Never counts.

PCNT_PRSInput_TypeDef

> PCNT_PRSInput_TypeDef

PRS inputs of PCNT.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Enumerator	
pcntPRSInputS0	
pcntPRSInputS1	PRS input 0.

Typedef Documentation

PCNT_PRSSel_TypeDef

```
typedef unsigned int PCNT_PRSSel_TypeDef
```

PRS sources for `s0PRS` and `s1PRS`.

Variable Documentation

initCntEvent

```
PCNT_CntEvent_TypeDef initCntEvent
```

initAuxCntEvent

```
PCNT_CntEvent_TypeDef initAuxCntEvent
```

Function Documentation

PCNT_CounterReset

```
void PCNT_CounterReset (PCNT_TypeDef * pcnt)
```

Reset PCNT counters and TOP register.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.

Note

- Notice that special SYNCBUSY handling is not applicable for the RSTEN bit of the control register, so we don't need to wait for it when only modifying RSTEN. (It would mean undefined wait time if clocked by an external clock.) The SYNCBUSY bit will however be set, leading to a synchronization in the LF domain, with, in reality, no changes.

PCNT_Enable

Copyright © 2026 Silicon Laboratories. All rights reserved.

Set PCNT operational mode.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.
PCNT_Mode_TypeDef	[in]	mode	An operational mode to use for PCNT.

Notice that this function does not do any configuration. Setting operational mode is normally only required after initialization is done, and if not done as part of initialization or if requiring to disable/reenable pulse counter.

Note

- This function may stall until synchronization to low-frequency domain is completed. For that reason, it should normally not be used when an external clock is used for the PCNT module, since stall time may be undefined.

PCNT_IsEnabled

```
bool PCNT_IsEnabled (PCNT_TypeDef * pcnt)
```

Returns if the PCNT module is enabled or not.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.

Notice that this function does not do any configuration.

Returns

- Returns TRUE if the module is enabled.

PCNT_CounterTopSet

```
void PCNT_CounterTopSet (PCNT_TypeDef * pcnt, uint32_t count, uint32_t top)
```

Set the counter and top values.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.
uint32_t	[in]	count	A value to set in the counter register.
uint32_t	[in]	top	A value to set in the top register.

The pulse counter is disabled while changing these values and reenabled (if originally enabled) when values have been set.

Note

- This function will stall until synchronization to low-frequency domain is completed. For that reason, it should normally not be used when an external clock is used for the PCNT module, since stall time may be undefined. The counter should normally only be set when operating in (or about to enable) pcntModeOvsSingle mode.

Copyright © 2026 Silicon Laboratories. All rights reserved.

PCNT_PRSInputEnable

```
void PCNT_PRSInputEnable (PCNT_TypeDef * pcnt, PCNT_PRSInput_TypeDef prsInput, bool enable)
```

Enable/disable the selected PRS input of PCNT.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.
PCNT_PRSInput_TypeDef	[in]	prsInput	PRS input (S0 or S1) of the selected PCNT module.
bool	[in]	enable	Set to true to enable, false to disable the selected PRS input.

Notice that this function does not do any configuration.

PCNT_Init

```
void PCNT_Init (PCNT_TypeDef * pcnt, const PCNT_Init_TypeDef * init)
```

Initialize the pulse counter.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.
const PCNT_Init_TypeDef *	[in]	init	A pointer to the initialization structure.

This function will configure the pulse counter. The clock selection is configured as follows, depending on operational mode:

- pcntModeOvsSingle - Use LFACLK.
- pcntModeExtSingle - Use external PCNTn_S0 pin.
- pcntModeExtQuad - Use external PCNTn_S0 pin.

Notice that the LFACLK must be enabled in all modes, since some basic setup is done with this clock even if the external pin clock usage mode is chosen. The pulse counter clock for the selected instance must also be enabled prior to initialization.

Notice that pins used by the PCNT module must be properly configured by the user explicitly through setting the ROUTE register for the PCNT to work as intended.

Writing to CNT will not occur in external clock modes (EXTCLKQUAD and EXTCLKSINGLE) because the external clock rate is unknown. The user should handle it manually depending on the application.

TOPB is written for all modes but in external clock mode it will take 3 external clock cycles to sync to TOP.

Note

- Initializing requires synchronization into the low-frequency domain. This may cause a delay.

PCNT_Reset

Copyright © 2026 Silicon Laboratories. All rights reserved.

void PCNT_Reset (PCNT_TypeDef * pcnt)

Reset PCNT to the same state that it was in after a hardware reset.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.

Notice the LFACLK must be enabled, since some basic reset is done with this clock. The pulse counter clock for the selected instance must also be enabled prior to initialization.

Note

- The ROUTE register is NOT reset by this function to allow for centralized setup of this feature.

PCNT_FilterConfiguration

void PCNT_FilterConfiguration (PCNT_TypeDef * pcnt, const PCNT_Filter_TypeDef * config, bool enable)

Set the filter configuration.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.
const PCNT_Filter_TypeDef *	[in]	config	A pointer to the configuration structure to be applied.
bool	[in]	enable	Indicates whether to enable or disable filtering.

This function will configure the PCNT input filter when the PCNT mode is configured to take an LFA-derived clock as an input clock.

PCNT_TopBufferSet

void PCNT_TopBufferSet (PCNT_TypeDef * pcnt, uint32_t val)

Set top buffer value.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.
uint32_t	[in]	val	A value to set in the top buffer register.

Note

- This function may stall until synchronization to low-frequency domain is completed. For that reason, it should normally not be used when an external clock is used for the PCNT module since stall time may be undefined.

Copyright © 2026 Silicon Laboratories. All rights reserved.

PCNT_TopSet

```
void PCNT_TopSet (PCNT_TypeDef * pcnt, uint32_t val)
```

Set the top value.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.
uint32_t	[in]	val	A value to set in the top register.

Note

- This function will stall until synchronization to low-frequency domain is completed. For that reason, it should normally not be used when an external clock is used for the PCNT module since stall time may be undefined.

PCNT_CounterGet

```
uint32_t PCNT_CounterGet (PCNT_TypeDef * pcnt)
```

Get the pulse counter value.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.

Returns

- Current pulse counter value.

PCNT_AuxCounterGet

```
uint32_t PCNT_AuxCounterGet (PCNT_TypeDef * pcnt)
```

Get the auxiliary counter value.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.

Returns

- Current auxiliary counter value.

PCNT_CounterSet

```
void PCNT_CounterSet (PCNT_TypeDef * pcnt, uint32_t count)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Set a counter value.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.
uint32_t	[in]	count	Value to set in counter register.

Pulse counter is disabled while changing counter value and re-enabled (if originally enabled) when counter value has been set.

Note

- This function will stall until synchronization to low-frequency domain is completed. For that reason, it should normally not be used when using an external clock to clock the PCNT module since stall time may be undefined in that case. The counter should normally only be set when operating in (or about to enable) pcntModeOvsSingle mode.

PCNT_IntClear

```
void PCNT_IntClear (PCNT_TypeDef * pcnt, uint32_t flags)
```

Clear one or more pending PCNT interrupts.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.
uint32_t	[in]	flags	Pending PCNT interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for the PCNT module (PCNT_IF_nnn).

PCNT_IntDisable

```
void PCNT_IntDisable (PCNT_TypeDef * pcnt, uint32_t flags)
```

Disable one or more PCNT interrupts.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.
uint32_t	[in]	flags	PCNT interrupt sources to disable. Use a bitwise logic OR combination of valid interrupt flags for PCNT module (PCNT_IF_nnn).

PCNT_IntEnable

Copyright © 2026 Silicon Laboratories. All rights reserved.

> void PCNT_IntEnable (PCNT_TypeDef * pcnt, uint32_t flags)

Enable one or more PCNT interrupts.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.
uint32_t	[in]	flags	PCNT interrupt sources to enable. Use a bitwise logic OR combination of valid interrupt flags for PCNT module (PCNT_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 PCNT_IntClear() prior to enabling the interrupt.

PCNT_IntGet

> uint32_t PCNT_IntGet (PCNT_TypeDef * pcnt)

Get pending PCNT interrupt flags.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.

Note

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

Returns

- PCNT interrupt sources pending. A bitwise logic OR combination of valid interrupt flags for PCNT module (PCNT_IF_nnn).

PCNT_IntGetEnabled

> uint32_t PCNT_IntGetEnabled (PCNT_TypeDef * pcnt)

Get enabled and pending PCNT interrupt flags.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to thePCNT peripheral register block.

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

Note

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

Returns

- Pending and enabled PCNT interrupt sources. The return value is the bitwise AND combination of
  - the OR combination of enabled interrupt sources in PCNT_IEN_nnn register (PCNT_IEN_nnn) and
  - the OR combination of valid interrupt flags of the PCNT module (PCNT_IF_nnn).

PCNT_IntSet

```
void PCNT_IntSet (PCNT_TypeDef * pcnt, uint32_t flags)
```

Set one or more pending PCNT interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.
uint32_t	[in]	flags	PCNT interrupt sources to set to pending. Use a bitwise logic OR combination of valid interrupt flags for PCNT module (PCNT_IF_nnn).

PCNT_Lock

```
void PCNT_Lock (PCNT_TypeDef * pcnt)
```

Lock PCNT registers.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.

Note

- When PCNT registers are locked PCNT_CFG, PCNT_EN, PCNT_SWRST, PCNT_CMD, PCNT_CTRL, PCNT_OVSCTRL, PCNT_CNT, PCNT_TOP, and PCNT_TOPB registers cannot be written to.

PCNT_Unlock

```
void PCNT_Unlock (PCNT_TypeDef * pcnt)
```

Unlock PCNT registers.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to thePCNT peripheral register block.

PCNT_TopBufferGet

Copyright © 2026 Silicon Laboratories. All rights reserved.

> uint32_t PCNT_TopBufferGet (PCNT_TypeDef * pcnt)

Get the pulse counter top buffer value.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.

Returns

- Current pulse counter top buffer value.

PCNT_TopGet

> uint32_t PCNT_TopGet (PCNT_TypeDef * pcnt)

Get the pulse counter top value.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	Pointer to the PCNT peripheral register block.

Returns

- Current pulse counter top value.

PCNT_Sync

> void PCNT_Sync (PCNT_TypeDef * pcnt, uint32_t mask)

Wait for an ongoing sync of register(s) to low-frequency domain to complete.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.
uint32_t	[in]	mask	A bitmask corresponding to SYNCBUSY register defined bits indicating registers that must complete any ongoing synchronization.

PCNT_StartMainCnt

> void PCNT_StartMainCnt (PCNT_TypeDef * pcnt)

Start the main PCNT counter.

Parameters

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.

This function will send a start command to the PCNT peripheral. The PCNT peripheral will use some LF clock ticks before the command is executed. The PCNT_Sync() function can be used to wait for the start command to be executed.

Note

- This function requires the PCNT to be enabled.

PCNT_StopMainCnt

```
void PCNT_StopMainCnt (PCNT_TypeDef * pcnt)
```

Stop the main PCNT counter.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.

This function will send a stop command to the PCNT peripheral. The PCNT peripheral will use some LF clock ticks before the command is executed. The PCNT_Sync() function can be used to wait for the stop command to be executed.

Note

- This function requires the PCNT to be enabled.

PCNT_StartAuxCnt

```
void PCNT_StartAuxCnt (PCNT_TypeDef * pcnt)
```

Start the auxiliary PCNT counter.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.

This function will send a start command to the PCNT peripheral. The PCNT peripheral will use some LF clock ticks before the command is executed. The PCNT_Sync() function can be used to wait for the start command to be executed.

Note

- This function requires the PCNT to be enabled.

PCNT_StopAuxCnt

```
void PCNT_StopAuxCnt (PCNT_TypeDef * pcnt)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Stop the auxiliary PCNT counter.

Parameters

Type	Direction	Argument Name	Description
PCNT_TypeDef *	[in]	pcnt	A pointer to the PCNT peripheral register block.

This function will send a stop command to the PCNT peripheral. The PCNT peripheral will use some LF clock ticks before the command is executed. The PCNT_Sync() function can be used to wait for the stop command to be executed.

Note

- This function requires the PCNT to be enabled.

Copyright © 2026 Silicon Laboratories. All rights reserved.

PCNT_Init_TypeDef

Initialization structure.

Public Attributes

PCNT_Mode_TypeDef	**mode** Mode to operate in.
uint32_t	**counter** Initial counter value (refer to reference manual for max value allowed).
uint32_t	**top** Initial top value (refer to reference manual for max value allowed).
bool	**negEdge** Polarity of incoming edge.
bool	**countDown** Counting direction, only applicable for pcntModeOvsSingle and pcntModeExtSingle modes.
bool	**filter** Enable filter, only available in pcntModeOvsSingle* mode.
bool	**debugHalt** Enable/disable PCNT counting during debug halt.
bool	**hyst** Set to true to enable hysteresis.
bool	**s1CntDir** Set to true to enable S1 to determine the direction of counting in OVSSINGLE or EXTCLKSINGLE modes.
PCNT_CntEvent_TypeDef	**cntEvent** Selects whether the regular counter responds to up-count events, down-count events, both, or none.
PCNT_CntEvent_TypeDef	**auxCntEvent** Selects whether the auxiliary counter responds to up-count events, down-count events, both, or none.
PCNT_PRSSel_TypeDef	**s0PRS** Select PRS channel as input to S0IN in PCNTx_INPUT register.
PCNT_PRSSel_TypeDef	**s1PRS** Select PRS channel as input to S1IN in PCNTx_INPUT register.

Public Attribute Documentation

mode

```
PCNT_Mode_TypeDef PCNT_Init_TypeDef::mode
```

Mode to operate in.

counter

Copyright © 2026 Silicon Laboratories. All rights reserved.

uint32_t PCNT_Init_TypeDef::counter

Initial counter value (refer to reference manual for max value allowed).

Only used for pcntModeOvsSingle (and possibly pcntModeDisable) modes. If using pcntModeExtSingle or pcntModeExtQuad modes, counter value is reset to HW reset value.

top

uint32_t PCNT_Init_TypeDef::top

Initial top value (refer to reference manual for max value allowed).

Only used for pcntModeOvsSingle (and possibly pcntModeDisable) modes. If using pcntModeExtSingle or pcntModeExtQuad modes, top value is reset to HW reset value.

negEdge

bool PCNT_Init_TypeDef::negEdge

Polarity of incoming edge.

- pcntModeExtSingle mode - if false, positive edges are counted, otherwise negative edges.
- pcntModeExtQuad mode - if true, counting direction is inverted.

countDown

bool PCNT_Init_TypeDef::countDown

Counting direction, only applicable for pcntModeOvsSingle and pcntModeExtSingle modes.

filter

bool PCNT_Init_TypeDef::filter

Enable filter, only available in pcntModeOvsSingle* mode.

debugHalt

bool PCNT_Init_TypeDef::debugHalt

Enable/disable PCNT counting during debug halt.

Only in OVSSINGLE and OVSQUAD modes.

Copyright © 2026 Silicon Laboratories. All rights reserved.

hyst

```
bool PCNT_Init_TypeDef::hyst
```

Set to true to enable hysteresis.

When enabled, PCNT will always overflow and underflow to TOP/2.

s1CntDir

```
bool PCNT_Init_TypeDef::s1CntDir
```

Set to true to enable S1 to determine the direction of counting in OVSSINGLE or EXTCLKSINGLE modes.

When S1 is high, the count direction is given by CNTDIR, and when S1 is low, the count direction is the opposite.

cntEvent

```
PCNT_CntEvent_TypeDef PCNT_Init_TypeDef::cntEvent
```

Selects whether the regular counter responds to up-count events, down-count events, both, or none.

auxCntEvent

```
PCNT_CntEvent_TypeDef PCNT_Init_TypeDef::auxCntEvent
```

Selects whether the auxiliary counter responds to up-count events, down-count events, both, or none.

s0PRS

```
PCNT_PRSSel_TypeDef PCNT_Init_TypeDef::s0PRS
```

Select PRS channel as input to S0IN in PCNTx_INPUT register.

s1PRS

```
PCNT_PRSSel_TypeDef PCNT_Init_TypeDef::s1PRS
```

Select PRS channel as input to S1IN in PCNTx_INPUT register.

Copyright © 2026 Silicon Laboratories. All rights reserved.

PCNT_Filter_TypeDef

Filter initialization structure.

Public Attributes

uint8_t	filtLen Used only in OVSINGLE and OVSQUAD1X-4X modes.
bool	flutterrm When set, removes flutter from Quaddecoder inputs S0IN and S1IN.

Public Attribute Documentation

filtLen

```
uint8_t PCNT_Filter_TypeDef::filtLen
```

Used only in OVSINGLE and OVSQUAD1X-4X modes.

To use this, enable filter by setting filter to true during PCNT_Init(). Filter length = (filtLen + 5) LFACLK cycles.

flutterrm

```
bool PCNT_Filter_TypeDef::flutterrm
```

When set, removes flutter from Quaddecoder inputs S0IN and S1IN.

Available only in OVSQUAD1X-4X modes.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
prsSignalPCNT0_UFOF = PRS_PCNT0_UFOF
prsSignalPCNT0_DIR = PRS_PCNT0_DIR
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
prsSignalBURTC_COMP = PRS_BURTC_COMP
prsSignalBURTC_OF = PRS_BURTC_OF
prsSignalSYSRTC0_GRP0OUT0 = PRS_SYSRTC0_GRP0OUT0
prsSignalSYSRTC0_GRP0OUT1 = PRS_SYSRTC0_GRP0OUT1
prsSignalSYSRTC0_GRP1OUT0 = PRS_SYSRTC0_GRP1OUT0
prsSignalSYSRTC0_GRP1OUT1 = PRS_SYSRTC0_GRP1OUT1
prsSignalHFXO0L_STATUS = PRS_HFXO0L_STATUS
prsSignalHFXO0L_STATUS1 = PRS_HFXO0L_STATUS1
prsSignalACMP0_OUT = PRS_ACMP0_OUT
prsSignalACMP1_OUT = PRS_ACMP1_OUT
prsSignalVDAC0_CH0WARM = PRS_VDAC0L_CH0WARM
prsSignalVDAC0_CH1WARM = PRS_VDAC0L_CH1WARM
prsSignalVDAC0_CH0DONE = PRS_VDAC0L_CH0DONEASYNC
prsSignalVDAC0_CH1DONE = PRS_VDAC0L_CH1DONEASYNC

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsSignalVDAC0_INTERNALTIMEROF
= PRS_VDAC0L_INTERNALTIMEROF
prsSignalVDAC0_REFRESHTIMEROF
= PRS_VDAC0L_REFRESHTIMEROF
prsSignalLESENSE_DECOUT0 =
PRS_LESENSE_DECOUT0
prsSignalLESENSE_DECOUT1 =
PRS_LESENSE_DECOUT1
prsSignalLESENSE_DECOUT2 =
PRS_LESENSE_DECOUT2
prsSignalLESENSE_DECCMP =
PRS_LESENSE_DECCMP
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
prsSignalEUSART0_CS =
PRS_EUSART0L_CS
prsSignalEUSART0_IRTX =
PRS_EUSART0L_IRDATX
prsSignalEUSART0_RTS =
PRS_EUSART0L_RTS
prsSignalEUSART0_RXDATA =
PRS_EUSART0L_RXDATAV
prsSignalEUSART0_TX =
PRS_EUSART0L_TX
prsSignalEUSART0_TXC =
PRS_EUSART0L_TXC
prsSignalEUSART0_RXFL =
PRS_EUSART0L_RXFL
prsSignalEUSART0_TXFL =
PRS_EUSART0L_TXFL
prsSignalEUSART1_CS =
PRS_EUSART1L_CS
prsSignalEUSART1_IRTX =
PRS_EUSART1L_IRDATX
prsSignalEUSART1_RTS =
PRS_EUSART1L_RTS
prsSignalEUSART1_RXDATA =
PRS_EUSART1L_RXDATAV
prsSignalEUSART1_TX =
PRS_EUSART1L_TX
prsSignalEUSART1_TXC =
PRS_EUSART1L_TXC
prsSignalEUSART1_RXFL =
PRS_EUSART1L_RXFL
prsSignalEUSART1_TXFL =
PRS_EUSART1L_TXFL
prsSignalEUSART2_CS =
PRS_EUSART2L_CS
prsSignalEUSART2_IRTX =
PRS_EUSART2L_IRDATX
prsSignalEUSART2_RTS =
PRS_EUSART2L_RTS
prsSignalEUSART2_RXDATA =
PRS_EUSART2L_RXDATAV
prsSignalEUSART2_TX =
PRS_EUSART2L_TX
prsSignalEUSART2_TXC =
PRS_EUSART2L_TXC

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsSignalEUSART2_RXFL =
PRS_EUSART2L_RXFL
prsSignalEUSART2_TXFL =
PRS_EUSART2L_TXFL
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
prsSignalAGCL_CCA =
PRS_AGCL_CCA
prsSignalAGCL_CCAREQ =
PRS_AGCL_CCAREQ
prsSignalAGCL_GAINADJUST
= PRS_AGCL_GAINADJUST
prsSignalAGCL_GAINOK =
PRS_AGCL_GAINOK
prsSignalAGCL_GAINREDUCED
= PRS_AGCL_GAINREDUCED
prsSignalAGCL_IFPKI1 =
PRS_AGCL_IFPKI1
prsSignalAGCL_IFPKQ2 =
PRS_AGCL_IFPKQ2
prsSignalAGCL_IFPKRST =
PRS_AGCL_IFPKRST
prsSignalAGC_PEAKDET =
PRS_AGC_PEAKDET
prsSignalAGC_PROPAGATED =
PRS_AGC_PROPAGATED
prsSignalAGC_RSSIDONE =
PRS_AGC_RSSIDONE
prsSignalBUFC_THR0 =
PRS_BUFC_THR0
prsSignalBUFC_THR1 =
PRS_BUFC_THR1
prsSignalBUFC_THR2 =
PRS_BUFC_THR2
prsSignalBUFC_THR3 =
PRS_BUFC_THR3
prsSignalBUFC_CNT0 =
PRS_BUFC_CNT0
prsSignalBUFC_CNT1 =
PRS_BUFC_CNT1
prsSignalBUFC_FULL =
PRS_BUFC_FULL
prsSignalMODEML_ADVANCE =
PRS_MODEML_ADVANCE
prsSignalMODEML_ANT0 =
PRS_MODEML_ANT0

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsSignalMODEML_COHDSADET = PRS_MODEML_COHDSADET
prsSignalMODEML_COHDSALIVE = PRS_MODEML_COHDSALIVE
prsSignalMODEML_DCLK = PRS_MODEML_DCLK
prsSignalMODEML_DOUT = PRS_MODEML_DOUT
prsSignalMODEML_FRAMEDET = PRS_MODEML_FRAMEDET
prsSignalMODEM_FRAMESENT = PRS_MODEM_FRAMESENT
prsSignalMODEM_PREDET = PRS_MODEM_PREDET
prsSignalMODEM_LRDSADET = PRS_MODEM_LRDSADET
prsSignalMODEM_LRDSALIVE = PRS_MODEM_LRDSALIVE
prsSignalMODEM_LOWCORR = PRS_MODEM_LOWCORR
prsSignalMODEM_NEWSYMBOL = PRS_MODEM_NEWSYMBOL
prsSignalMODEM_NEWWND = PRS_MODEM_NEWWND
prsSignalMODEM_POSTPONE = PRS_MODEM_POSTPONE
prsSignalMODEMH_PRESENT = PRS_MODEMH_PRESENT
prsSignalMODEMH_RSSIJUMP = PRS_MODEMH_RSSIJUMP
prsSignalMODEMH_SYNCSENT = PRS_MODEMH_SYNCSENT
prsSignalMODEMH_TIMDET = PRS_MODEMH_TIMDET
prsSignalMODEMH_WEAK = PRS_MODEMH_WEAK
prsSignalMODEMH_EOF = PRS_MODEMH_EOF
prsSignalFRC_DCLK = PRS_FRC_DCLK
prsSignalFRC_DOUT = PRS_FRC_DOUT
prsSignalPROTIMERL_BOF = PRS_PROTIMERL_BOF
prsSignalPROTIMERL_CC0 = PRS_PROTIMERL_CC0
prsSignalPROTIMERL_CC1 = PRS_PROTIMERL_CC1
prsSignalPROTIMERL_CC2 = PRS_PROTIMERL_CC2
prsSignalPROTIMERL_CC3 = PRS_PROTIMERL_CC3
prsSignalPROTIMERL_CC4 = PRS_PROTIMERL_CC4
prsSignalPROTIMERL_LBTF = PRS_PROTIMERL_LBTF
prsSignalPROTIMERL_LBTR = PRS_PROTIMERL_LBTR
prsSignalPROTIMER_LBTS = PRS_PROTIMER_LBTS
prsSignalPROTIMER_POF = PRS_PROTIMER_POF
prsSignalPROTIMER_T0MATCH = PRS_PROTIMER_T0MATCH
prsSignalPROTIMER_T0UF = PRS_PROTIMER_T0UF
prsSignalPROTIMER_T1MATCH = PRS_PROTIMER_T1MATCH

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsSignalPROTIMER_T1UF = PRS_PROTIMER_T1UF
prsSignalPROTIMER_WOF = PRS_PROTIMER_WOF
prsSignalRACL_ACTIVE = PRS_RACL_ACTIVE
prsSignalRACL_LNAEN = PRS_RACL_LNAEN
prsSignalRACL_PAEN = PRS_RACL_PAEN
prsSignalRACL_RX = PRS_RACL_RX
prsSignalRACL_TX = PRS_RACL_TX
prsSignalRACL_CTIOUT0 = PRS_RACL_CTIOUT0
prsSignalRACL_CTIOUT1 = PRS_RACL_CTIOUT1
prsSignalRACL_CTIOUT2 = PRS_RACL_CTIOUT2
prsSignalRAC_CTIOUT3 = PRS_RAC_CTIOUT3
prsSignalRAC_AUXADCDATA = PRS_RAC_AUXADCDATA
prsSignalRAC_AUXADCDATAVALID = PRS_RAC_AUXADCDATAVALID
prsSignalSYNTH_MUX0 = PRS_SYNTH_MUX0
prsSignalSYNTH_MUX1 = PRS_SYNTH_MUX1

}

PRS Signal.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
prsConsumerLETIMER0_CLEAR = offsetof(PRS_TypeDef, CONSUMER_LETIMER0_CLEAR)
prsConsumerLETIMER0_START = offsetof(PRS_TypeDef, CONSUMER_LETIMER0_START)
prsConsumerLETIMER0_STOP = offsetof(PRS_TypeDef, CONSUMER_LETIMER0_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)
prsConsumerEUSART0_CLK = offsetof(PRS_TypeDef, CONSUMER_EUSART0_CLK)
prsConsumerEUSART0_RX = offsetof(PRS_TypeDef, CONSUMER_EUSART0_RX)
prsConsumerEUSART0_TRIGGER = offsetof(PRS_TypeDef, CONSUMER_EUSART0_TRIGGER)
prsConsumerEUSART1_CLK = offsetof(PRS_TypeDef, CONSUMER_EUSART1_CLK)
prsConsumerEUSART1_RX = offsetof(PRS_TypeDef, CONSUMER_EUSART1_RX)
prsConsumerEUSART1_TRIGGER = offsetof(PRS_TypeDef, CONSUMER_EUSART1_TRIGGER)
prsConsumerEUSART2_CLK = offsetof(PRS_TypeDef, CONSUMER_EUSART2_CLK)
prsConsumerEUSART2_RX = offsetof(PRS_TypeDef, CONSUMER_EUSART2_RX)
prsConsumerEUSART2_TRIGGER = offsetof(PRS_TypeDef, CONSUMER_EUSART2_TRIGGER)
prsConsumerWDOG0_SRC0 = offsetof(PRS_TypeDef, CONSUMER_WDOG0_SRC0)
prsConsumerWDOG0_SRC1 = offsetof(PRS_TypeDef, CONSUMER_WDOG0_SRC1)
prsConsumerWDOG1_SRC0 = offsetof(PRS_TypeDef, CONSUMER_WDOG1_SRC0)
prsConsumerWDOG1_SRC1 = offsetof(PRS_TypeDef, CONSUMER_WDOG1_SRC1)
prsConsumerPCNT0_IN0 = offsetof(PRS_TypeDef, CONSUMER_PCNT0_S0IN)
prsConsumerPCNT0_IN1 = offsetof(PRS_TypeDef, CONSUMER_PCNT0_S1IN)
prsConsumerSYSRTC0_SRC0 = offsetof(PRS_TypeDef, CONSUMER_SYSRTC0_IN0)
prsConsumerSYSRTC0_SRC1 = offsetof(PRS_TypeDef, CONSUMER_SYSRTC0_IN1)
prsConsumerHFXO0_OSCREQ = offsetof(PRS_TypeDef, CONSUMER_HFXO0_OSCREQ)
prsConsumerHFXO0_TIMEOUT = offsetof(PRS_TypeDef, CONSUMER_HFXO0_TIMEOUT)
prsConsumerLESENSE_START = offsetof(PRS_TypeDef, CONSUMER_LESENSE_START)
prsConsumerVDAC0_ASYNCTRIGCH0 = offsetof(PRS_TypeDef, CONSUMER_VDAC0_ASYNCTRIGCH0)
prsConsumerVDAC0_ASYNCTRIGCH1 = offsetof(PRS_TypeDef, CONSUMER_VDAC0_ASYNCTRIGCH1)
prsConsumerVDAC0_SYNCTRIGCH0 = offsetof(PRS_TypeDef, CONSUMER_VDAC0_SYNCTRIGCH0)
prsConsumerVDAC0_SYNCTRIGCH1 = offsetof(PRS_TypeDef, CONSUMER_VDAC0_SYNCTRIGCH1)
prsConsumerCORE_M33RXEV = offsetof(PRS_TypeDef, CONSUMER_CORE_M33RXEV)

}

PRS Consumers.

Functions

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

#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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
prsSignalTIMER2_CC0	TIMER2 capture/compare channel 0 Signal.
prsSignalTIMER2_CC1	TIMER2 capture/compare channel 1 Signal.
prsSignalTIMER2_CC2	TIMER2 capture/compare channel 2 Signal.
prsSignalTIMER3_UF	TIMER3 underflow Signal.
prsSignalTIMER3_OF	TIMER3 overflow Signal.
prsSignalTIMER3_CC0	TIMER3 capture/compare channel 0 Signal.
prsSignalTIMER3_CC1	TIMER3 capture/compare channel 1 Signal.
prsSignalTIMER3_CC2	TIMER3 capture/compare channel 2 Signal.
prsSignalTIMER4_UF	TIMER4 underflow Signal.
prsSignalTIMER4_OF	TIMER4 overflow Signal.
prsSignalTIMER4_CC0	TIMER4 capture/compare channel 0 Signal.
prsSignalTIMER4_CC1	TIMER4 capture/compare channel 1 Signal.
prsSignalTIMER4_CC2	TIMER4 capture/compare channel 2 Signal.
prsSignalLETIMER0_CH0	LETIMER0 channel 0 Signal.
prsSignalLETIMER0_CH1	LETIMER0 channel 1 Signal.
prsSignalPCNT0_UFOF	PCNT0_TCC Signal.
prsSignalPCNT0_DIR	PCNT0_TCC Signal.

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsSignalCORE_CTIOUT0	CORE CTIOUT0 Signal.
prsSignalCORE_CTIOUT1	CORE CTIOUT1 Signal.
prsSignalCORE_CTIOUT2	CORE CTIOUT2 Signal.
prsSignalCORE_CTIOUT3	CORE CTIOUT3 Signal.
prsSignalCMUL_CLKOUT0	CMU CLKOUT0 Signal.
prsSignalCMUL_CLKOUT1	CMU CLKOUT1 Signal.
prsSignalCMUL_CLKOUT2	CMU CLKOUT2 Signal.
prsSignalPRSL_ASYNCH0	PRS channel 0 Signal.
prsSignalPRSL_ASYNCH1	PRS channel 1 Signal.
prsSignalPRSL_ASYNCH2	PRS channel 2 Signal.
prsSignalPRSL_ASYNCH3	PRS channel 3 Signal.
prsSignalPRSL_ASYNCH4	PRS channel 4 Signal.
prsSignalPRSL_ASYNCH5	PRS channel 5 Signal.
prsSignalPRSL_ASYNCH6	PRS channel 6 Signal.
prsSignalPRSL_ASYNCH7	PRS channel 7 Signal.
prsSignalPRS_ASYNCH8	PRS channel 8 Signal.
prsSignalPRS_ASYNCH9	PRS channel 9 Signal.
prsSignalPRS_ASYNCH10	PRS channel 10 Signal.
prsSignalPRS_ASYNCH11	PRS channel 11 Signal.
prsSignalBURTC_COMP	BURTC compare Signal.
prsSignalBURTC_OF	BURTC overflow Signal.
prsSignalSYSRTC0_GRP0OUT0	SYSRTC GRP0OUT0 Signal.
prsSignalSYSRTC0_GRP0OUT1	SYSRTC GRP0OUT1 Signal.
prsSignalSYSRTC0_GRP1OUT0	SYSRTC GRP1OUT0 Signal.
prsSignalSYSRTC0_GRP1OUT1	SYSRTC GRP1OUT1 Signal.
prsSignalHFXO0L_STATUS	HFXO0L_STATUS Signal.
prsSignalHFXO0L_STATUS1	HFXO0L_STATUS1 Signal.
prsSignalACMP0_OUT	ACMP0 Signal.
prsSignalACMP1_OUT	ACMP1 output Signal.
prsSignalVDAC0_CH0WARM	VDAC0 channel 0 warmed Signal.
prsSignalVDAC0_CH1WARM	VDAC0 channel 1 warmed Signal.
prsSignalVDAC0_CH0DONE	VDAC0 channel 0 conversion done Signal.
prsSignalVDAC0_CH1DONE	VDAC0 channel 1 conversion done Signal.
prsSignalVDAC0_INTERNALTIMEROF	VDAC0 internal timer overflow Signal.
prsSignalVDAC0_REFRESHTIMEROF	VDAC0 internal timer overflow Signal.
prsSignalLESENSE_DECOUT0	LESENSE_DECOUT0 Signal.
prsSignalLESENSE_DECOUT1	LESENSE_DECOUT1 Signal.
prsSignalLESENSE_DECOUT2	LESENSE_DECOUT2 Signal.
prsSignalLESENSE_DECCMP	LESENSE_DECCMP Signal.
prsSignalUSART0_TXC	USART0 TX complete Signal.
prsSignalUSART0_RXDATA	USART0 RX data available Signal.
prsSignalUSART0_IRTX	USART0 IR TX Signal.
prsSignalUSART0_RTS	USART0 RTS Signal.

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsSignalUSART0_TX	USART0 TX Signal.
prsSignalUSART0_CS	USART0 chip select Signal.
prsSignalEUSART0_CS	EUSART0 chip select Signal.
prsSignalEUSART0_IRTX	EUSART0 IR RX Signal.
prsSignalEUSART0_RTS	EUSART0 RTS Signal.
prsSignalEUSART0_RXDATA	EUSART0 RX data available Signal.
prsSignalEUSART0_TX	EUSART0 TX Signal.
prsSignalEUSART0_TXC	EUSART0 TX complete Signal.
prsSignalEUSART0_RXFL	EUSART0 rxfl Signal.
prsSignalEUSART0_TXFL	EUSART0 txfl Signal.
prsSignalEUSART1_CS	EUSART1 chip select Signal.
prsSignalEUSART1_IRTX	EUSART1 IR TX Signal.
prsSignalEUSART1_RTS	EUSART1 RTS Signal.
prsSignalEUSART1_RXDATA	EUSART1 RX data available Signal.
prsSignalEUSART1_TX	EUSART1 TX Signal.
prsSignalEUSART1_TXC	EUSART1 TX complete Signal.
prsSignalEUSART1_RXFL	EUSART1 rxfl Signal.
prsSignalEUSART1_TXFL	EUSART1 txfl Signal.
prsSignalEUSART2_CS	EUSART2 chip select Signal.
prsSignalEUSART2_IRTX	EUSART2 IR TX Signal.
prsSignalEUSART2_RTS	EUSART2 RTS Signal.
prsSignalEUSART2_RXDATA	EUSART2 RX data available Signal.
prsSignalEUSART2_TX	EUSART2 TX Signal.
prsSignalEUSART2_TXC	EUSART2 TX complete Signal.
prsSignalEUSART2_RXFL	EUSART2 rxfl Signal.
prsSignalEUSART2_TXFL	EUSART2 txfl 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.
prsSignalAGCL_CCA	AGCL_CCA Signal.
prsSignalAGCL_CCAREQ	AGCL_CCAREQ Signal.
prsSignalAGCL_GAINADJUST	AGCL_GAINADJUST Signal.
prsSignalAGCL_GAINOK	AGCL_GAINOK Signal.
prsSignalAGCL_GAINREDUCED	AGCL_GAINREDUCED Signal.
prsSignalAGCL_IFPKI1	AGCL_IFPKI1 Signal.

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsSignalAGCL_IFPKQ2	AGCL_IFPKQ2 Signal.
prsSignalAGCL_IFPKRST	AGCL_IFPKRST Signal.
prsSignalAGC_PEAKDET	AGC_PEAKDET Signal.
prsSignalAGC_PROPAGATED	AGC_PROPAGATED Signal.
prsSignalAGC_RSSIDONE	AGC_RSSIDONE Signal.
prsSignalBUFC_THR0	BUFC_THR0 Signal.
prsSignalBUFC_THR1	BUFC_THR1 Signal.
prsSignalBUFC_THR2	BUFC_THR2 Signal.
prsSignalBUFC_THR3	BUFC_THR3 Signal.
prsSignalBUFC_CNT0	BUFC_CNT0 Signal.
prsSignalBUFC_CNT1	BUFC_CNT1 Signal.
prsSignalBUFC_FULL	BUFC_FULL Signal.
prsSignalMODEML_ADVANCE	MODEML_ADVANCE Signal.
prsSignalMODEML_ANT0	MODEML_ANT0 Signal.
prsSignalMODEML_ANT1	MODEML_ANT1 Signal.
prsSignalMODEML_COHDSADET	MODEML_COHDSADET Signal.
prsSignalMODEML_COHDSALIVE	MODEML_COHDSALIVE Signal.
prsSignalMODEML_DCLK	MODEML_DCLK Signal.
prsSignalMODEML_DOUT	MODEML_DOUT Signal.
prsSignalMODEML_FRAMEDET	MODEML_FRAMEDET Signal.
prsSignalMODEM_FRAMESENT	MODEM_FRAMESENT Signal.
prsSignalMODEM_PREDET	MODEM_PREDET Signal.
prsSignalMODEM_LRDSADET	MODEM_LRDSADET Signal.
prsSignalMODEM_LRDSALIVE	MODEM_LRDSALIVE Signal.
prsSignalMODEM_LOWCORR	MODEM_LOWCORR Signal.
prsSignalMODEM_NEWSYMBOL	MODEM_NEWSYMBOL Signal.
prsSignalMODEM_NEWWND	MODEM_NEWWND Signal.
prsSignalMODEM_POSTPONE	MODEM_POSTPONE Signal.
prsSignalMODEMH_PRESENT	MODEMH_PRESENT Signal.
prsSignalMODEMH_RSSIJUMP	MODEMH_RSSIJUMP Signal.
prsSignalMODEMH_SYNCSENT	MODEMH_SYNCSENT Signal.
prsSignalMODEMH_TIMDET	MODEMH_TIMDET Signal.
prsSignalMODEMH_WEAK	MODEMH_WEAK Signal.
prsSignalMODEMH_EOF	MODEMH_EOF Signal.
prsSignalFRC_DCLK	FRC_DCLK Signal.
prsSignalFRC_DOUT	FRC_DOUT Signal.
prsSignalPROTIMERL_BOF	PROTIMERL_BOF Signal.
prsSignalPROTIMERL_CC0	PROTIMERL_CC0 Signal.
prsSignalPROTIMERL_CC1	PROTIMERL_CC1 Signal.
prsSignalPROTIMERL_CC2	PROTIMERL_CC2 Signal.
prsSignalPROTIMERL_CC3	PROTIMERL_CC3 Signal.
prsSignalPROTIMERL_CC4	PROTIMERL_CC4 Signal.
prsSignalPROTIMERL_LBTF	PROTIMERL_LBTF Signal.

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsSignalPROTIMERL_LBTR	PROTIMERL_LBTR Signal.
prsSignalPROTIMER_LBTS	PROTIMER_LBTS Signal.
prsSignalPROTIMER_POF	PROTIMER_POF Signal.
prsSignalPROTIMER_T0MATCH	PROTIMER_T0MATCH Signal.
prsSignalPROTIMER_T0UF	PROTIMER_T0UF Signal.
prsSignalPROTIMER_T1MATCH	PROTIMER_T1MATCH Signal.
prsSignalPROTIMER_T1UF	PROTIMER_T1UF Signal.
prsSignalPROTIMER_WOF	PROTIMER_WOF Signal.
prsSignalRACL_ACTIVE	RACL_ACTIVE Signal.
prsSignalRACL_LNAEN	RACL_LNAEN Signal.
prsSignalRACL_PAEN	RACL_PAEN Signal.
prsSignalRACL_RX	RACL_RX Signal.
prsSignalRACL_TX	RACL_TX Signal.
prsSignalRACL_CTIOUT0	RACL_CTIOUT0 Signal.
prsSignalRACL_CTIOUT1	RACL_CTIOUT1 Signal.
prsSignalRACL_CTIOUT2	RACL_CTIOUT2 Signal.
prsSignalRAC_CTIOUT3	RAC_CTIOUT3 Signal.
prsSignalRAC_AUXADCDATA	RAC_AUXADCDATA Signal.
prsSignalRAC_AUXADCDATAVALID	RAC_AUXADCDATAVALID Signal.
prsSignalSYNTH_MUX0	SYNTH_MUX0 Signal.
prsSignalSYNTH_MUX1	SYNTH_MUX1 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
prsConsumerEUSART0_CLK	EUSART0 clk consumer.
prsConsumerEUSART0_RX	EUSART0 RX consumer.
prsConsumerEUSART0_TRIGGER	EUSART0 trigger consumer.
prsConsumerEUSART1_CLK	EUSART1 clk consumer.
prsConsumerEUSART1_RX	EUSART1 RX consumer.
prsConsumerEUSART1_TRIGGER	EUSART1 trigger consumer.
prsConsumerEUSART2_CLK	EUSART1 clk consumer.
prsConsumerEUSART2_RX	EUSART2 RX consumer.
prsConsumerEUSART2_TRIGGER	EUSART2 trigger consumer.
prsConsumerWDOG0_SRC0	WDOG0 source 0 consumer.
prsConsumerWDOG0_SRC1	WDOG0 source 1 consumer.
prsConsumerWDOG1_SRC0	WDOG1 source 0 consumer.
prsConsumerWDOG1_SRC1	WDOG1 source 1 consumer.
prsConsumerPCNT0_IN0	PCNT0 input 0 consumer.
prsConsumerPCNT0_IN1	PCNT0 input 1 consumer.
prsConsumerSYSRTC0_SRC0	SYSRTC0 input 0 consumer.
prsConsumerSYSRTC0_SRC1	SYSRTC0 input 1 consumer.
prsConsumerHFXO0_OSCREQ	OSCREQ consumer.
prsConsumerHFXO0_TIMEOUT	HFXO0_TIMEOUT consumer.
prsConsumerLESENSE_START	LESENSE_START consumer.
prsConsumerVDAC0_ASYNCTRIGCH0	VDAC0 ASYNC TRIGER CH0 consumer.
prsConsumerVDAC0_ASYNCTRIGCH1	VDAC0 ASYNC TRIGER CH1 consumer.
prsConsumerVDAC0_SYNCTRIGCH0	VDAC0 SYNC TRIGER CH0 consumer.
prsConsumerVDAC0_SYNCTRIGCH1	VDAC0 SYNC TRIGER CH1 consumer.
prsConsumerCORE_M33RXEV	M33 Consumer Selection.

Function Documentation

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
uint32_t	[in]	signal	A signal (for selected `source`) 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 `source`) to use. Use one of the PRS_CH_CTRL_SIGSEL_x defines that support asynchronous operation.

Asynchronous reflexes are not clocked on HFPERCLK 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 EM0 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 prsTypeAsync or prsTypeSync.

Returns

- Channel number >= 0 if an unused PRS channel was found. If no free PRS channel was found then -1 is returned.

PRS_Reset

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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 prsTypeAsync or prsTypeSync.
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 prsTypeAsync or prsTypeSync.
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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 prsTypeAsync or prsTypeSync.
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 prsTypeAsync or prsTypeSync.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 prsTypeAsync or prsTypeSync.

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 HFPERCLK) 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 HFPERCLK 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.gcc.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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

#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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_WDOG0RMODE_MASK
  rmuResetSys = _EMU_RSTCTRL_SYSRMODE_MASK
  rmuResetCoreLockup = _EMU_RSTCTRL_LOCKUPRMODE_MASK
  rmuResetAVDD = _EMU_RSTCTRL_AVDDBODRMODE_MASK
  rmuResetIOVDD0 = _EMU_RSTCTRL_IOVDD0BODRMODE_MASK
  rmuResetDecouple = _EMU_RSTCTRL_DECBODRMODE_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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

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().

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Deprecated Functions

Typedefs

typedef sli_se_datatransfer_t	SE_DataTransfer_t SE DMA transfer descriptor.
typedef sli_se_mailbox_command_t	SE_Command_t SE Command structure.
typedef sli_se_mailbox_response_t	SE_Response_t Possible responses to a command.

Functions

void	writeToFifo(uint32_t value) Write to FIFO.
void	SE_addDataInput(SE_Command_t *command, volatile SE_DataTransfer_t *data) Add input data to a command.
void	SE_addDataOutput(SE_Command_t *command, volatile 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

bool SE_isCommandCompleted(void)
Check whether the running command has completed.

SE_Response_t SE_readCommandResponse(void)
Read the status of the previously executed command.

void SE_waitCommandCompletion(void)
Wait for completion of the current command.

void SE_disableInterrupt(uint32_t flags)
Disable one or more SE interrupts.

void SE_enableInterrupt(uint32_t flags)
Enable one or more SE interrupts.

Macros

#define SE_RESPONSE_MASK 0x000F0000UL
Response status codes for the Secure Element.

#define SE_RESPONSE_OK 0x00000000UL
Command executed successfully or signature was successfully validated.

#define SE_FIFO_MAX_PARAMETERS 13U
Maximum amount of parameters supported by the hardware FIFO.

#define SE_DATATRANSFER_STOP 0x00000001UL
Stop datatransfer.

#define SE_DATATRANSFER_DISCARD 0x40000000UL
Discard datatransfer.

#define SE_DATATRANSFER_REALIGN 0x20000000UL
Realign datatransfer.

#define SE_DATATRANSFER_CONSTADDRESS 0x10000000UL
Datatransfer Const Address.

#define SE_DATATRANSFER_LENGTH_MASK 0x0FFFFFFFUL
Stop Length Mask.

#define SE_MAX_PARAMETERS 4U
Maximum amount of parameters for largest command in defined command set.

#define SE_DATATRANSFER_DEFAULT (address, length)
Default initialization of data transfer struct.

#define SE_COMMAND_DEFAULT (command)
Default initialization of command struct.

Typedef Documentation

SE_DataTransfer_t

```
typedef sli_se_datatransfer_t SE_DataTransfer_t
```

SE DMA transfer descriptor.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Can be linked to each other to provide scatter-gather behavior.

SE_Command_t

```
typedef sli_se_mailbox_command_t SE_Command_t
```

SE Command structure.

See

SE_Response_t

```
typedef sli_se_mailbox_response_t SE_Response_t
```

Possible responses to a command.

Function Documentation

writeToFifo

```
void writeToFifo (uint32_t value)
```

Write to FIFO.

Parameters

Type	Direction	Argument Name	Description
uint32_t	N/A	value	Value to write to FIFO

SE_addDataInput

```
void SE_addDataInput (SE_Command_t * command, volatile 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.
volatile 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

SE_addDataOutput

void SE_addDataOutput (SE_Command_t * command, volatile 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.
volatile 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_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.

Note

- The command response needs to be read for every executed command, and can only be read once per executed command (FIFO behavior).

Returns

- One of the SE_RESPONSE return codes: SE_RESPONSE_OK when the command was executed successfully or a signature was successfully verified.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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).

Copyright © 2026 Silicon Laboratories. All rights reserved.

Deprecated Functions

Deprecated Functions

Deprecated Functions.

Functions

SE_Response_t SE_readPubkey(uint32_t key_type, void *pubkey, uint32_t numBytes, bool signature)

Function Documentation

SE_readPubkey

> SE_Response_t SE_readPubkey (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 read.
void *	[out]	pubkey	Pointer to a buffer to contain the returned public key. 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, the function will return the signature programmed for the specified public key instead of the public key itself.

Read pubkey or pubkey signature.

Read out a public key stored in the SE, or its signature. The command can be used to read:

- SE_KEY_TYPE_BOOT
- SE_KEY_TYPE_AUTH

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum SMU_Peripheral_TypeDef {

smuPeripheralEMU = _SMU_PPUPATD0_EMU_SHIFT
smuPeripheralCMU = _SMU_PPUPATD0_CMU_SHIFT
smuPeripheralHFXO = 32 + _SMU_PPUPATD1_HFXO0_SHIFT
smuPeripheralHFRCO0 = _SMU_PPUPATD0_HFRCO0_SHIFT
smuPeripheralFSRCO = _SMU_PPUPATD0_FSRCO_SHIFT
smuPeripheralDPLL0 = _SMU_PPUPATD0_DPLL0_SHIFT
smuPeripheralLFXO = _SMU_PPUPATD0_LFXO_SHIFT
smuPeripheralLFRCO = _SMU_PPUPATD0_LFRCO_SHIFT
smuPeripheralULFRCO = _SMU_PPUPATD0_ULFRCO_SHIFT
smuPeripheralMSC = _SMU_PPUPATD0_MSC_SHIFT
smuPeripheralICACHE0 = _SMU_PPUPATD0_ICACHE0_SHIFT
smuPeripheralPRS = _SMU_PPUPATD0_PRS_SHIFT
smuPeripheralGPIO = _SMU_PPUPATD0_GPIO_SHIFT
smuPeripheralLDMA = _SMU_PPUPATD0_LDMA_SHIFT
smuPeripheralLDMAXBAR = _SMU_PPUPATD0_LDMAXBAR_SHIFT
smuPeripheralTIMER0 = _SMU_PPUPATD0_TIMER0_SHIFT
smuPeripheralTIMER1 = _SMU_PPUPATD0_TIMER1_SHIFT
smuPeripheralTIMER2 = _SMU_PPUPATD0_TIMER2_SHIFT
smuPeripheralTIMER3 = _SMU_PPUPATD0_TIMER3_SHIFT
smuPeripheralTIMER4 = _SMU_PPUPATD0_TIMER4_SHIFT
smuPeripheralUSART0 = _SMU_PPUPATD0_USART0_SHIFT
smuPeripheralBURTC = _SMU_PPUPATD0_BURTC_SHIFT
smuPeripheralI2C1 = _SMU_PPUPATD0_I2C1_SHIFT
smuPeripheralSYSCFGCFGNS = _SMU_PPUPATD0_SYSCFGCFGNS_SHIFT
smuPeripheralSYSCFG = _SMU_PPUPATD0_SYSCFG_SHIFT
smuPeripheralBURAM = _SMU_PPUPATD0_BURAM_SHIFT
smuPeripheralGPCRC = _SMU_PPUPATD0_GPCRC_SHIFT
smuPeripheralDCDC = _SMU_PPUPATD0_DCDC_SHIFT
smuPeripheralHOSTMAILBOX = _SMU_PPUPATD0_HOSTMAILBOX_SHIFT
smuPeripheralEUSART0 = 32 + _SMU_PPUPATD1_EUSART0_SHIFT
smuPeripheralEUSART1 = _SMU_PPUPATD0_EUSART1_SHIFT
smuPeripheralEUSART2 = _SMU_PPUPATD0_EUSART2_SHIFT
smuPeripheralSYSRTC = 32 + _SMU_PPUPATD1_SYSRTC_SHIFT
smuPeripheralLCD = 32 + _SMU_PPUPATD1_LCD_SHIFT
smuPeripheralKEYSCAN = 32 + _SMU_PPUPATD1_KEYSCAN_SHIFT
smuPeripheralDMEM = 32 + _SMU_PPUPATD1_DMEM_SHIFT
smuPeripheralLCDRF = 32 + _SMU_PPUPATD1_LCDRF_SHIFT
smuPeripheralPFMXPPRF = 32 + _SMU_PPUPATD1_PFMXPPRF_SHIFT
smuPeripheralVDAC0 = 32 + _SMU_PPUPATD1_VDAC0_SHIFT
smuPeripheralPCNT = 32 + _SMU_PPUPATD1_PCNT_SHIFT
smuPeripheralLESENSE = 32 + _SMU_PPUPATD1_LESENSE_SHIFT
smuPeripheralHFRCO1 = 32 + _SMU_PPUPATD1_HFRCO1_SHIFT
smuPeripheralHFXO0 = 32 + _SMU_PPUPATD1_HFXO0_SHIFT
smuPeripheralLETIMER0 = 32 + _SMU_PPUPATD1_LETIMER0_SHIFT
smuPeripheralIADC0 = 32 + _SMU_PPUPATD1_IADC0_SHIFT
smuPeripheralACMP0 = 32 + _SMU_PPUPATD1_ACMP0_SHIFT
smuPeripheralACMP1 = 32 + _SMU_PPUPATD1_ACMP1_SHIFT
smuPeripheralI2C0 = 32 + _SMU_PPUPATD1_I2C0_SHIFT
smuPeripheralWDOG0 = 32 + _SMU_PPUPATD1_WDOG0_SHIFT
smuPeripheralWDOG1 = 32 + _SMU_PPUPATD1_WDOG1_SHIFT
smuPeripheralAMUXCP0 = 32 + _SMU_PPUPATD1_AMUXCP0_SHIFT
smuPeripheralRADIOAES = 32 + _SMU_PPUPATD1_RADIOAES_SHIFT
smuPeripheralSMU = 32 + _SMU_PPUPATD1_SMU_SHIFT
smuPeripheralSMUCFGNS = 32 + _SMU_PPUPATD1_SMUCFGNS_SHIFT
smuPeripheralAHBRADIO = 32 + _SMU_PPUPATD1_AHBRADIO_SHIFT
smuPeripheralSEMAILBOX = 32 + _SMU_PPUPATD1_SEMAILBOX_SHIFT
smuPeripheralMVP = 32 + _SMU_PPUPATD1_MVP_SHIFT
smuPeripheralEnd

}

SMU peripheral identifiers.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
smuPeripheralBURTC	SMU peripheral identifier for BURTC
smuPeripheralI2C1	SMU peripheral identifier for I2C1
smuPeripheralSYSCFGCFGNS	SMU peripheral identifier for SYSCFGCFGNS.
smuPeripheralSYSCFG	SMU peripheral identifier for SYSCFG
smuPeripheralBURAM	SMU peripheral identifier for BURAM
smuPeripheralGPCRC	SMU peripheral identifier for GPCRC
smuPeripheralDCDC	SMU peripheral identifier for DCDC
smuPeripheralHOSTMAILBOX	SMU peripheral identifier for HOSTMAILBOX.
smuPeripheralEUSART0	SMU peripheral identifier for EUSART0
smuPeripheralEUSART1	SMU peripheral identifier for EUSART1
smuPeripheralEUSART2	SMU peripheral identifier for EUSART2
smuPeripheralSYSRTC	SMU peripheral identifier for SYSRTC
smuPeripheralLCD	SMU peripheral identifier for LCD
smuPeripheralKEYSCAN	SMU peripheral identifier for KEYSCAN
smuPeripheralDMEM	SMU peripheral identifier for DMEM
smuPeripheralLCDRF	SMU peripheral identifier for LCDRF
smuPeripheralPFMXPPRF	SMU peripheral identifier for PFMXPPRF
smuPeripheralVDAC0	SMU peripheral identifier for VDAC0
smuPeripheralPCNT	SMU peripheral identifier for PCNT
smuPeripheralLESENSE	SMU peripheral identifier for LESENSE
smuPeripheralHFRCO1	SMU peripheral identifier for HFRCO1
smuPeripheralHFXO0	SMU peripheral identifier for HFXO0
smuPeripheralLETIMER0	SMU peripheral identifier for LETIMER
smuPeripheralIADC0	SMU peripheral identifier for IADC0
smuPeripheralACMP0	SMU peripheral identifier for ACMP0
smuPeripheralACMP1	SMU peripheral identifier for ACMP1
smuPeripheralI2C0	SMU peripheral identifier for I2C0

Copyright © 2026 Silicon Laboratories. All rights reserved.

smuPeripheralWDOG0	SMU peripheral identifier for WDOG0
smuPeripheralWDOG1	SMU peripheral identifier for WDOG1
smuPeripheralAMUXCP0	SMU peripheral identifier for AMUXCP0
smuPeripheralRADIOAES	SMU peripheral identifier for RADIOAES
smuPeripheralSMU	SMU peripheral identifier for SMU
smuPeripheralSMUCFGNS	SMU peripheral identifier for SMUCFGNS
smuPeripheralAHBRADIO	SMU peripheral identifier for AHBRADIO
smuPeripheralSEMAILBOX	SMU peripheral identifier for SEMAILBOX.
smuPeripheralMVP	SMU peripheral identifier for MVP
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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 PPUSECIEN is set and the SMU's Secure IRQ enabled, the CPU will be interrupted and SMU_SECURE_IRQHandler Will handle the interrupt.

Copyright © 2026 Silicon Laboratories. All rights reserved.

SMU_PrivilegedAccess_TypeDef

SMU peripheral privileged access enablers.

Public Attributes

bool **privilegedReserved0**
Reserved privileged access enabler

bool **privilegedEMU**
Privileged access enabler for EMU

bool **privilegedCMU**
Privileged access enabler for CMU

bool **privilegedHFRCO0**
Privileged access enabler for HFRCO0

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 **privilegedTIMER0**
Privileged access enabler for TIMER0

bool **privilegedTIMER1**
Privileged access enabler for TIMER1

bool **privilegedTIMER2**
Privileged access enabler for TIMER2

Copyright © 2026 Silicon Laboratories. All rights reserved.

bool **privilegedTIMER3**
Privileged access enabler for TIMER3

bool **privilegedTIMER4**
Privileged access enabler for TIMER4

bool **privilegedUSART0**
Privileged access enabler for USART0

bool **privilegedBURTC**
Privileged access enabler for BURTC

bool **privilegedI2C1**
Privileged access enabler for I2C1

bool **privilegedCHIPTESTCTRL**
Privileged access enabler for CHIPTESTCTRL.

bool **privilegedSYSCFGCFGNS**
Privileged access enabler for SYSCFGCFGNS

bool **privilegedSYSCFG**
Privileged access enabler for SYSCFG

bool **privilegedBURAM**
Privileged access enabler for BURAM

bool **privilegedGPCRC**
Privileged access enabler for GPCRC

bool **privilegedDCDC**
Privileged access enabler for DCDC

bool **privilegedHOSTMAILBOX**
Privileged access enabler for HOSTMAILBOX

bool **privilegedEUSART1**
Privileged access enabler for EUSART1

bool **privilegedEUSART2**
Privileged access enabler for EUSART2

bool **privilegedSYSRTC**
Privileged access enabler for SYSRTC

bool **privilegedLCD**
Privileged access enabler for LCD

bool **privilegedKEYSCAN**
Privileged access enabler for KEYSCAN

bool **privilegedDMEM**
Privileged access enabler for DMEM

bool **privilegedLCDRF**
Privileged access enabler for LCDRF

bool **privilegedPFMXPPRF**
Privileged access enabler for PFMXPPRF

bool **privilegedRADIOAES**
Privileged access enabler for RADIOAES

Copyright © 2026 Silicon Laboratories. All rights reserved.

bool **privilegedSMU**
Privileged access enabler for SMU

bool **privilegedSMUCFGNS**
Privileged access enabler for SMUCFGNS

bool **privilegedLETIMER0**
Privileged access enabler for LETIMER0

bool **privilegedIADC0**
Privileged access enabler for IADC0

bool **privilegedACMP0**
Privileged access enabler for ACMP0

bool **privilegedACMP1**
Privileged access enabler for ACMP1

bool **privilegedAMUXCP0**
Privileged access enabler for AMUXCP0

bool **privilegedVDAC0**
Privileged access enabler for VDAC0

bool **privilegedPCNT**
Privileged access enabler for PCNT

bool **privilegedLESENSE**
Privileged access enabler for LESENSE

bool **privilegedHFRCO1**
Privileged access enabler for HFRCO1

bool **privilegedHFXO0**
Privileged access enabler for HFXO0

bool **privilegedI2C0**
Privileged access enabler for I2C0

bool **privilegedWDOG0**
Privileged access enabler for WDOG0

bool **privilegedWDOG1**
Privileged access enabler for WDOG1

bool **privilegedEUSART0**
Privileged access enabler for EUSART0

bool **privilegedSEMAILBOX**
Privileged access enabler for SEMAILBOX

bool **privilegedMVP**
Privileged access enabler for MVP

bool **privilegedAHBRADIO**
Privileged access enabler for AHBRADIO

Public Attribute Documentation

privilegedReserved0

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
bool SMU_PrivilegedAccess_TypeDef::privilegedReserved0
```

Reserved privileged access enabler

privilegedEMU

```
bool SMU_PrivilegedAccess_TypeDef::privilegedEMU
```

Privileged access enabler for EMU

privilegedCMU

```
bool SMU_PrivilegedAccess_TypeDef::privilegedCMU
```

Privileged access enabler for CMU

privilegedHFRCO0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedHFRCO0
```

Privileged access enabler for HFRCO0

privilegedFSRCO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedFSRCO
```

Privileged access enabler for FSRCO

privilegedDPLL0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedDPLL0
```

Privileged access enabler for DPLL0

privilegedLFXO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLFXO
```

Privileged access enabler for LFXO

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 PRS0

privilegedGPIO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedGPIO
```

Privileged access enabler for GPIO

privilegedLDMA

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLDMA
```

Privileged access enabler for LDMA

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

privilegedBURTC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedBURTC
```

Privileged access enabler for BURTC

privilegedI2C1

```
bool SMU_PrivilegedAccess_TypeDef::privilegedI2C1
```

Privileged access enabler for I2C1

privilegedCHIPTESTCTRL

```
bool SMU_PrivilegedAccess_TypeDef::privilegedCHIPTESTCTRL
```

Privileged access enabler for CHIPTESTCTRL.

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

privilegedGPCRC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedGPCRC
```

Privileged access enabler for GPCRC

Copyright © 2026 Silicon Laboratories. All rights reserved.

privilegedDCDC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedDCDC
```

Privileged access enabler for DCDC

privilegedHOSTMAILBOX

```
bool SMU_PrivilegedAccess_TypeDef::privilegedHOSTMAILBOX
```

Privileged access enabler for HOSTMAILBOX

privilegedEUSART1

```
bool SMU_PrivilegedAccess_TypeDef::privilegedEUSART1
```

Privileged access enabler for EUSART1

privilegedEUSART2

```
bool SMU_PrivilegedAccess_TypeDef::privilegedEUSART2
```

Privileged access enabler for EUSART2

privilegedSYSRTC

```
bool SMU_PrivilegedAccess_TypeDef::privilegedSYSRTC
```

Privileged access enabler for SYSRTC

privilegedLCD

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLCD
```

Privileged access enabler for LCD

privilegedKEYSCAN

```
bool SMU_PrivilegedAccess_TypeDef::privilegedKEYSCAN
```

Privileged access enabler for KEYSCAN

Copyright © 2026 Silicon Laboratories. All rights reserved.

privilegedDMEM

```
bool SMU_PrivilegedAccess_TypeDef::privilegedDMEM
```

Privileged access enabler for DMEM

privilegedLCDRF

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLCDRF
```

Privileged access enabler for LCDRF

privilegedPFMXPPRF

```
bool SMU_PrivilegedAccess_TypeDef::privilegedPFMXPPRF
```

Privileged access enabler for PFMXPPRF

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

privilegedLETIMER0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLETIMER0
```

Privileged access enabler for LETIMER0

Copyright © 2026 Silicon Laboratories. All rights reserved.

privilegedIADC0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedIADC0
```

Privileged access enabler for IADC0

privilegedACMP0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedACMP0
```

Privileged access enabler for ACMP0

privilegedACMP1

```
bool SMU_PrivilegedAccess_TypeDef::privilegedACMP1
```

Privileged access enabler for ACMP1

privilegedAMUXCP0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedAMUXCP0
```

Privileged access enabler for AMUXCP0

privilegedVDAC0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedVDAC0
```

Privileged access enabler for VDAC0

privilegedPCNT

```
bool SMU_PrivilegedAccess_TypeDef::privilegedPCNT
```

Privileged access enabler for PCNT

privilegedLESENSE

```
bool SMU_PrivilegedAccess_TypeDef::privilegedLESENSE
```

Privileged access enabler for LESENSE

Copyright © 2026 Silicon Laboratories. All rights reserved.

privilegedHFRCO1

```
bool SMU_PrivilegedAccess_TypeDef::privilegedHFRCO1
```

Privileged access enabler for HFRCO1

privilegedHFXO0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedHFXO0
```

Privileged access enabler for HFXO0

privilegedI2C0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedI2C0
```

Privileged access enabler for I2C0

privilegedWDOG0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedWDOG0
```

Privileged access enabler for WDOG0

privilegedWDOG1

```
bool SMU_PrivilegedAccess_TypeDef::privilegedWDOG1
```

Privileged access enabler for WDOG1

privilegedEUSART0

```
bool SMU_PrivilegedAccess_TypeDef::privilegedEUSART0
```

Privileged access enabler for EUSART0

privilegedSEMAILBOX

```
bool SMU_PrivilegedAccess_TypeDef::privilegedSEMAILBOX
```

Privileged access enabler for SEMAILBOX

Copyright © 2026 Silicon Laboratories. All rights reserved.

privilegedMVP

```
bool SMU_PrivilegedAccess_TypeDef::privilegedMVP
```

Privileged access enabler for MVP

privilegedAHBRADIO

```
bool SMU_PrivilegedAccess_TypeDef::privilegedAHBRADIO
```

Privileged access enabler for AHBRADIO

Copyright © 2026 Silicon Laboratories. All rights reserved.

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::@0	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::@0 SMU_Init_TypeDef::ppu
```

PPU init array.

enable

```
bool SMU_Init_TypeDef::enable
```

SMU enable flag.

When set, SMU_Init() will enable SMU.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_CalAddrVal_TypeDef

SYSTEM_ChipRevision_TypeDef

Enumerations

enum SYSTEM_SecurityCapability_TypeDef {

securityCapabilityUnknown
securityCapabilityNA
securityCapabilityBasic
securityCapabilityRoT
securityCapabilitySE
securityCapabilityVault

}

Family security capability.

enum SYSTEM_PartFamily_TypeDef {

systemPartFamilyFlex28 = DEVINFO_PART_FAMILY_FG | (28 << _DEVINFO_PART_FAMILYNUM_SHIFT)
systemPartFamilyZen28 = DEVINFO_PART_FAMILY_ZG | (28 << _DEVINFO_PART_FAMILYNUM_SHIFT)
systemPartFamilySideWalk28 = DEVINFO_PART_FAMILY_SG | (28 << _DEVINFO_PART_FAMILYNUM_SHIFT)
systemPartFamilyEfm32Pearl28 = DEVINFO_PART_FAMILY_PG | (28 << _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.

Functions

bool SYSTEM_GetCalibrationValue(volatile uint32_t *regAddress)

Get a factory calibration value for a given peripheral register.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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).
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.

Enumeration Documentation

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.

SYSTEM_PartFamily_TypeDef

Copyright © 2026 Silicon Laboratories. All rights reserved.

SYSTEM_PartFamily_TypeDef

Family identifiers.

Enumerator	
systemPartFamilyFlex28	EFR32 Flex Gecko Series 2 Config 8 Value Device Family.
systemPartFamilyZen28	EFR32 Zen Gecko Series 2 Config 8 Value Device Family.
systemPartFamilySideWalk28	EFR32 Side Walk Gecko Series 2 Config 8 Value Device Family.
systemPartFamilyEfm32Pearl28	EFM32 Pearl Gecko Series 2 Config 8 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.

Function Documentation

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)

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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_ChipRevisionGet

```
void SYSTEM_ChipRevisionGet (SYSTEM_ChipRevision_TypeDef * rev)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
SYSTEM_FpuAccess_TypeDef	[in]	accessMode	Floating point co-processor access mode. See SYSTEM_FpuAccess_TypeDef for details.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

SYSTEM_ChipRevision_TypeDef

Chip revision details.

Public Attributes

uint8_t	minor Minor revision number.
uint8_t	major Major revision number.
uint16_t	partNumber Device part number.

Public Attribute Documentation

minor

```
uint8_t SYSTEM_ChipRevision_TypeDef::minor
```

Minor revision number.

major

```
uint8_t SYSTEM_ChipRevision_TypeDef::major
```

Major revision number.

partNumber

```
uint16_t SYSTEM_ChipRevision_TypeDef::partNumber
```

Device part number.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_EVERYSECONDEDGE
  timerEventRising = _TIMER_CC_CTRL_ICEVCTRL_RISING
  timerEventFalling = _TIMER_CC_CTRL_ICEVCTRL_FALLING

}

Input capture event control.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_PRSASYNCLEVEL
  timerPrsInputAsyncPulse = _TIMER_CC_CFG_INSEL_PRSASYNCPULSE

}

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

void TIMER_EnableDTI(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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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_EnableDTI

> void TIMER_EnableDTI (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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

- 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
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
uint32_t	[in]	flags	TIMER interrupt source(s) to set to pending. Use one or more valid interrupt flags for the TIMER module (TIMER_IF_nnn) 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
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.
uint32_t	[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
TIMER_TypeDef *	[in]	timer	Pointer to the TIMER peripheral register block.

Returns

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 HFPER / HFPERB 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

> 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

disSyncOut

```
bool TIMER_Init_TypeDef::disSyncOut
```

Disable ability of timer to start/stop/reload other timers that have their SYNC bit set.

Copyright © 2026 Silicon Laboratories. All rights reserved.

TIMER_InitCC_TypeDef

TIMER compare/capture initialization structure.

Public Attributes

TIMER_Event_TypeDef	**eventCtrl** Input capture event control.
TIMER_Edge_TypeDef	**edge** Input capture edge select.
TIMER_PRSSEL_TypeDef	**prsSel** Peripheral reflex system trigger selection.
TIMER_OutputAction_TypeDef	**cufoa** Counter underflow output action.
TIMER_OutputAction_TypeDef	**cofoa** Counter overflow output action.
TIMER_OutputAction_TypeDef	**cmoa** Counter match output action.
TIMER_CCMode_TypeDef	**mode** Compare/capture channel mode.
bool	**filter** Enable digital filter.
bool	**prsInput** Select TIMERnCCx (false) or PRS input (true).
bool	**coist** Compare output initial state.
bool	**outInvert** Invert output from compare/capture channel.
TIMER_PrsOutput_t	**prsOutput** PRS output configuration.
TIMER_PrsInput_TypeDef	**prsInputType** 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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
TIMER_Edge_TypeDef TIMER_InitCC_TypeDef::edge
```

Input capture edge select.

prsSel

```
TIMER_PRSSEL_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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_PRSSEL_TypeDef	prsSel Select which PRS channel as DTI input.
TIMER_Prescale_TypeDef	prescale DTI prescaling factor, if HFPER / HFPERB 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 (faultSourcePrsSel0).
TIMER_PRSSEL_TypeDef	faultSourcePrsSel0 Select which PRS signal to be PRS fault source 0.
bool	enableFaultSourcePrsSel1 Enable PRS fault source 1 (faultSourcePrsSel1).
TIMER_PRSSEL_TypeDef	faultSourcePrsSel1 Select which PRS signal to be PRS fault source 1.
TIMER_DtiFaultAction_TypeDef	faultAction Fault Action.

Public Attribute Documentation

Copyright © 2026 Silicon Laboratories. All rights reserved.

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_PRSSEL_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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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`).

Copyright © 2026 Silicon Laboratories. All rights reserved.

faultSourcePrsSel0

```
TIMER_PRSSEL_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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 HFPERCLK peripheral. Enable HFPERCLK domain and USART0.
 * We also need to enable the clock for GPIO to configure pins. */
CMU_ClockEnable(cmuClock_HFPER, 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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Modules

USART_InitAsync_TypeDef

USART_PrsTriggerInit_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

Copyright © 2026 Silicon Laboratories. All rights reserved.

}
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,

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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.

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 bitrate less than or equal to the desired bitrate to ensure that the bitrate does not exceed the requested value.

For USART peripherals, fractional clock division is suppressed in synchronous mode by this function, even though the hardware allows it. This restriction prevents timing violations causing framing errors and data corruption at high frequencies. The fractional clock division modifies the clock cycles to average out around the desired bitrate, adjusting some half periods clock cycle count. Fractional divider affects the duty cycle of the SCLK, impact is greater at high bitrates where there are few clock cycle count per half period, and can cause timing issues like MISO setup time violations. The clock divider will be set to an integer value, so the reference clock will only be divided by 2N (where N is a positive integer).

For example, if the reference clock is 39 MHz, the maximum baudrate would be 19.5 MHz (RefClock/2). If the baudrate is set to 19.5 MHz, the clock divider will be set to 0 and the baudrate will be 19.5 MHz. If the baudrate is [9.75 - 19.5[MHz, the clock divider will be set to 1, and the actual baudrate will be (RefClock/4) = 9.75 MHz. If the baudrate is [6.5 - 9.75[MHz, the clock divider will be set to 2, and the actual baudrate will be (RefClock/6) = 6.5 MHz. If the baudrate is [4.875 - 6.5[MHz, the clock divider will be set to 3, and the actual baudrate will be (RefClock/8) = 4.875 MHz.

USART_BaudrateGet will return the actual baudrate according to the USART peripheral configuration.

Warnings

- If you require fractional clock division, you may manually modify the CLKDIV register. Users should ensure compliance with USART synchronous mode timing requirements and consider using integer division ratios of the reference clock for reliable operation at high bitrates.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.)

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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.)

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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 RXDATAX 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

-

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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)
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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	**mvdis** 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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

mvdis

```
bool USART_InitAsync_TypeDef::mvdis
```

Majority Vote Disable for 16x, 8x and 6x oversampling modes.

prsRxEnable

```
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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

USART_InitIrDA_TypeDef

IrDA mode initialization structure.

Inherited from asynchronous mode initialization structure.

Public Attributes

USART_InitAsync_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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

VDAC - Voltage DAC

VDAC - Voltage DAC

Digital to Analog Voltage Converter (VDAC) Peripheral API.

This module contains functions to control the VDAC peripheral of Silicon Labs' 32-bit MCUs and SoCs. VDAC converts digital values to analog signals at up to 500 ksps with 12-bit accuracy. VDAC is designed for low energy consumption, but can also provide very good performance.

The following steps are necessary for basic operation:

Clock enable:

```
CMU_ClockEnable(cmuClock_VDAC0, true);
```

Initialize the VDAC with default settings and modify selected fields:

```
VDAC_Init_TypeDef vdacInit        = VDAC_INIT_DEFAULT;
VDAC_InitChannel_TypeDef vdacChInit = VDAC_INITCHANNEL_DEFAULT;

// Set prescaler to get 1 MHz VDAC clock frequency.
vdacInit.prescaler = VDAC_PrescaleCalc(1000000, true, 0); // function call for series 0/1
VDAC_Init(VDAC0, &vdacInit);

vdacChInit.enable = true;
VDAC_InitChannel(VDAC0, &vdacChInit, 0);
```

Perform a conversion:

```
VDAC_ChannelOutputSet(VDAC0, 0, 250);
```

Note

- The output stage of a VDAC channel consists of an on-chip operational amplifier (OPAMP) in the OPAMP module. This OPAMP is highly configurable; and to exploit the VDAC functionality fully, configure the OPAMP using the OPAMP API. Using the OPAMP API also loads OPAMP calibration values. The default (reset) settings of OPAMP is sufficient for many applications.

Modules

VDAC_Init_TypeDef

VDAC_InitChannel_TypeDef

Enumerations

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum VDAC_Refresh_TypeDef {

vdacRefresh2 = _VDAC_CFG_REFRESHPERIOD_CYCLES2
vdacRefresh4 = _VDAC_CFG_REFRESHPERIOD_CYCLES4
vdacRefresh8 = _VDAC_CFG_REFRESHPERIOD_CYCLES8
vdacRefresh16 = _VDAC_CFG_REFRESHPERIOD_CYCLES16
vdacRefresh32 = _VDAC_CFG_REFRESHPERIOD_CYCLES32
vdacRefresh64 = _VDAC_CFG_REFRESHPERIOD_CYCLES64
vdacRefresh128 = _VDAC_CFG_REFRESHPERIOD_CYCLES128
vdacRefresh256 = _VDAC_CFG_REFRESHPERIOD_CYCLES256

}
Channel refresh period.

enum VDAC_TimerOverflow_TypeDef {

vdacCycles2 = _VDAC_CFG_TIMEROVRFLOWPERIOD_CYCLES2
vdacCycles4 = _VDAC_CFG_TIMEROVRFLOWPERIOD_CYCLES4
vdacCycles8 = _VDAC_CFG_TIMEROVRFLOWPERIOD_CYCLES8
vdacCycles16 = _VDAC_CFG_TIMEROVRFLOWPERIOD_CYCLES16
vdacCycles32 = _VDAC_CFG_TIMEROVRFLOWPERIOD_CYCLES32
vdacCycles64 = _VDAC_CFG_TIMEROVRFLOWPERIOD_CYCLES64

}
Timer overflow period.

enum VDAC_Ref_TypeDef {

vdacRef1V25 = _VDAC_CFG_REFRSEL_V125
vdacRef2V5 = _VDAC_CFG_REFRSEL_V25
vdacRefAvdd = _VDAC_CFG_REFRSEL_VDD
vdacRefExtPin = _VDAC_CFG_REFRSEL_EXT

}
Reference voltage for VDAC.

enum VDAC_RefreshSource_TypeDef {

vdacRefreshSrcNone = _VDAC_CH0CFG_REFRESHSOURCE_NONE
vdacRefreshSrcRefreshTimer = _VDAC_CH0CFG_REFRESHSOURCE_REFRESHTIMER
vdacRefreshSrcSyncPrs = _VDAC_CH0CFG_REFRESHSOURCE_SYNCPRS
vdacRefreshSrcAsyncPrs = _VDAC_CH0CFG_REFRESHSOURCE_ASYNCPRS

}
Refresh source for VDAC.

enum VDAC_TrigMode_TypeDef {

vdacTrigModeNone = _VDAC_CH0CFG_TRIGMODE_NONE
vdacTrigModeSw = _VDAC_CH0CFG_TRIGMODE_SW
vdacTrigModeSyncPrs = _VDAC_CH0CFG_TRIGMODE_SYNCPRS
vdacTrigModeLesense = _VDAC_CH0CFG_TRIGMODE_LESENSE
vdacTrigModeInternalTimer = _VDAC_CH0CFG_TRIGMODE_INTERNALTIMER
vdacTrigModeAsyncPrs = _VDAC_CH0CFG_TRIGMODE_ASYNCPRS

}
Channel conversion trigger mode.

enum VDAC_PowerMode_TypeDef {

vdacPowerModeHighPower = _VDAC_CH0CFG_POWERMODE_HIGHPOWER
vdacPowerModeLowPower = _VDAC_CH0CFG_POWERMODE_LOWPOWER

}
Channel power mode.

Copyright © 2026 Silicon Laboratories. All rights reserved.

enum VDAC_ChPortSel_t {

vdacChPortNone = _VDAC_OUTCTRL_ABUSPORTSELCH0_NONE
vdacChPortA = _VDAC_OUTCTRL_ABUSPORTSELCH0_PORTA
vdacChPortB = _VDAC_OUTCTRL_ABUSPORTSELCH0_PORTB
vdacChPortC = _VDAC_OUTCTRL_ABUSPORTSELCH0_PORTC
vdacChPortD = _VDAC_OUTCTRL_ABUSPORTSELCH0_PORTD

}

VDAC channel Abus port selection.

Functions

void VDAC_Enable(VDAC_TypeDef *vdac, unsigned int ch, bool enable)
Enable/disable the VDAC channel.

void VDAC_Init(VDAC_TypeDef *vdac, const VDAC_Init_TypeDef *init)
Initialize VDAC.

void VDAC_InitChannel(VDAC_TypeDef *vdac, const VDAC_InitChannel_TypeDef *init, unsigned int ch)
Initialize a VDAC channel.

void VDAC_ChannelOutputSet(VDAC_TypeDef *vdac, unsigned int channel, uint32_t value)
Set the output signal of a VDAC channel to a given value.

uint32_t VDAC_PrescaleCalc(VDAC_TypeDef *vdac, uint32_t vdacFreq)
Calculate the prescaler value used to determine VDAC clock.

void VDAC_Reset(VDAC_TypeDef *vdac)
Reset VDAC to same state that it was in after a hardwares reset.

void VDAC_SineModeStart(VDAC_TypeDef *vdac, bool start)
Start/stop Sinemode.

void VDAC_Channel0OutputSet(VDAC_TypeDef *vdac, uint32_t value)
Set the output signal of VDAC channel 0 to a given value.

void VDAC_Channel1OutputSet(VDAC_TypeDef *vdac, uint32_t value)
Set the output signal of VDAC channel 1 to a given value.

void VDAC_IntClear(VDAC_TypeDef *vdac, uint32_t flags)
Clear one or more pending VDAC interrupts.

void VDAC_IntDisable(VDAC_TypeDef *vdac, uint32_t flags)
Disable one or more VDAC interrupts.

void VDAC_IntEnable(VDAC_TypeDef *vdac, uint32_t flags)
Enable one or more VDAC interrupts.

uint32_t VDAC_IntGet(VDAC_TypeDef *vdac)
Get pending VDAC interrupt flags.

uint32_t VDAC_IntGetEnabled(VDAC_TypeDef *vdac)
Get enabled and pending VDAC interrupt flags.

void VDAC_IntSet(VDAC_TypeDef *vdac, uint32_t flags)
Set one or more pending VDAC interrupts from SW.

uint32_t VDAC_GetStatus(VDAC_TypeDef *vdac)
Get the VDAC Status register.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Macros

#define VDAC_INIT_DEFAULT undefined
Default configuration for VDAC initialization structure.

#define VDAC_INIT_SINE_GENERATION_MODE undefined
Sine mode configuration for VDAC initialization structure.

#define VDAC_INITCHANNEL_DEFAULT undefined
Default configuration for VDAC channel initialization structure.

Enumeration Documentation

VDAC_Refresh_TypeDef

VDAC_Refresh_TypeDef

Channel refresh period.

Enumerator	
vdacRefresh2	Refresh every 2 clock cycles.
vdacRefresh4	Refresh every 4 clock cycles.
vdacRefresh8	Refresh every 8 clock cycles.
vdacRefresh16	Refresh every 16 clock cycles.
vdacRefresh32	Refresh every 32 clock cycles.
vdacRefresh64	Refresh every 64 clock cycles.
vdacRefresh128	Refresh every 128 clock cycles.
vdacRefresh256	Refresh every 256 clock cycles.

VDAC_TimerOverflow_TypeDef

VDAC_TimerOverflow_TypeDef

Timer overflow period.

Enumerator	
vdacCycles2	Overflows every 2 clock cycles.
vdacCycles4	Overflows every 4 clock cycles.
vdacCycles8	Overflows every 8 clock cycles.
vdacCycles16	Overflows every 16 clock cycles.
vdacCycles32	Overflows every 32 clock cycles.
vdacCycles64	Overflows every 64 clock cycles.

VDAC_Ref_TypeDef

VDAC_Ref_TypeDef

Copyright © 2026 Silicon Laboratories. All rights reserved.

Reference voltage for VDAC.

Enumerator	
vdacRef1V25	Internal 1.25 V band gap reference.
vdacRef2V5	Internal 2.5 V band gap reference.
vdacRefAvdd	AVDD reference.
vdacRefExtPin	External pin reference.

VDAC_RefreshSource_TypeDef

> VDAC_RefreshSource_TypeDef

Refresh source for VDAC.

Enumerator	
vdacRefreshSrcNone	No refresh source.
vdacRefreshSrcRefreshTimer	Refresh triggered by refresh timer overflow.
vdacRefreshSrcSyncPrs	Refresh triggered by sync PRS.
vdacRefreshSrcAsyncPrs	Refresh triggered by async PRS.

VDAC_TrigMode_TypeDef

> VDAC_TrigMode_TypeDef

Channel conversion trigger mode.

Enumerator	
vdacTrigModeNone	No conversion trigger source selected.
vdacTrigModeSw	Channel is triggered by CHnDATA or COMBDATA write.
vdacTrigModeSyncPrs	Channel is triggered by Sync PRS input.
vdacTrigModeLesense	Channel is triggered by LESENSE.
vdacTrigModeInternalTimer	Channel is triggered by Internal Timer.
vdacTrigModeAsyncPrs	Channel is triggered by Async PRS input.

VDAC_PowerMode_TypeDef

> VDAC_PowerMode_TypeDef

Channel power mode.

Enumerator	
vdacPowerModeHighPower	High power buffer mode.
vdacPowerModeLowPower	Low power buffer mode.

VDAC_ChPortSel_t

Copyright © 2026 Silicon Laboratories. All rights reserved.

VDAC_ChPortSel_t

VDAC channel Abus port selection.

Enumerator	
vdacChPortNone	No GPIO selected.
vdacChPortA	Port A selected.
vdacChPortB	Port B selected.
vdacChPortC	Port C selected.
vdacChPortD	Port D selected.

Function Documentation

VDAC_Enable

void VDAC_Enable (VDAC_TypeDef * vdac, unsigned int ch, bool enable)

Enable/disable the VDAC channel.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	A pointer to the VDAC peripheral register block.
unsigned int	[in]	ch	A channel to enable/disable.
bool	[in]	enable	True to enable VDAC channel, false to disable.

VDAC_Init

void VDAC_Init (VDAC_TypeDef * vdac, const VDAC_Init_TypeDef * init)

Initialize VDAC.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	A pointer to the VDAC peripheral register block.
const VDAC_Init_TypeDef *	[in]	init	A pointer to the VDAC initialization structure.

Initializes the common parts for both channels. This function will also load calibration values from the Device Information (DI) page into the VDAC calibration register. To complete a VDAC setup, channel control configuration must also be done. See VDAC_InitChannel().

Note

- This function will disable both channels prior to configuration.

VDAC_InitChannel

Copyright © 2026 Silicon Laboratories. All rights reserved.

> void VDAC_InitChannel (VDAC_TypeDef * vdac, const VDAC_InitChannel_TypeDef * init, unsigned int ch)

Initialize a VDAC channel.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	A pointer to the VDAC peripheral register block.
const VDAC_InitChannel_TypeDef *	[in]	init	A pointer to the VDAC channel initialization structure.
unsigned int	[in]	ch	A channel number to initialize.

VDAC_ChannelOutputSet

> void VDAC_ChannelOutputSet (VDAC_TypeDef * vdac, unsigned int channel, uint32_t value)

Set the output signal of a VDAC channel to a given value.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	A pointer to the VDAC peripheral register block.
unsigned int	[in]	channel	A channel number to set the output of.
uint32_t	[in]	value	A value to write to the channel output register CHnDATA.

This function sets the output signal of a VDAC channel by writing `value` to the corresponding CHnDATA register.

VDAC_PrescaleCalc

> uint32_t VDAC_PrescaleCalc (VDAC_TypeDef * vdac, uint32_t vdacFreq)

Calculate the prescaler value used to determine VDAC clock.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.
uint32_t	[in]	vdacFreq	VDAC frequency target. The frequency will automatically be adjusted to be below maximum allowed VDAC clock.

The VDAC clock is given by the input clock divided by the prescaler+1.

VDAC_CLK = IN_CLK / (prescale + 1)

The maximum VDAC clock is 1 MHz.

Note

Copyright © 2026 Silicon Laboratories. All rights reserved.

If the requested VDAC frequency is low and the maximum prescaler value can't adjust the actual VDAC frequency lower than requested, the maximum prescaler value is returned resulting in a higher VDAC frequency than requested.

Returns

- A prescaler value to use for VDAC to achieve a clock value less than or equal to `vdacFreq`.

VDAC_Reset

```
void VDAC_Reset (VDAC_TypeDef * vdac)
```

Reset VDAC to same state that it was in after a hardwares reset.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	A pointer to the VDAC peripheral register block.

VDAC_SineModeStart

```
void VDAC_SineModeStart (VDAC_TypeDef * vdac, bool start)
```

Start/stop Sinemode.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.
bool	[in]	start	True to start the Sine mode, false to stop it.

This function sends the sine mode start/stop signal to the DAC.

VDAC_Channel0OutputSet

```
void VDAC_Channel0OutputSet (VDAC_TypeDef * vdac, uint32_t value)
```

Set the output signal of VDAC channel 0 to a given value.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.
uint32_t	[in]	value	Value to write to channel 0 output register CH0DATA.

This function sets the output signal of VDAC channel 0 by writing `value` to the CH0DATA register.

VDAC_Channel1OutputSet

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
void VDAC_Channel1OutputSet (VDAC_TypeDef * vdac, uint32_t value)
```

Set the output signal of VDAC channel 1 to a given value.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.
uint32_t	[in]	value	Value to write to channel 1 output register CH1DATA.

This function sets the output signal of VDAC channel 1 by writing `value` to the CH1DATA register.

VDAC_IntClear

```
void VDAC_IntClear (VDAC_TypeDef * vdac, uint32_t flags)
```

Clear one or more pending VDAC interrupts.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.
uint32_t	[in]	flags	Pending VDAC interrupt source to clear. Use a bitwise logic OR combination of valid interrupt flags for the VDAC module (VDAC_IF_nnn).

VDAC_IntDisable

```
void VDAC_IntDisable (VDAC_TypeDef * vdac, uint32_t flags)
```

Disable one or more VDAC interrupts.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.
uint32_t	[in]	flags	VDAC interrupt sources to disable. Use a bitwise logic OR combination of valid interrupt flags for the VDAC module (VDAC_IF_nnn).

VDAC_IntEnable

```
void VDAC_IntEnable (VDAC_TypeDef * vdac, uint32_t flags)
```

Enable one or more VDAC interrupts.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.
uint32_t	[in]	flags	VDAC interrupt sources to enable. Use a bitwise logic OR combination of valid interrupt flags for the VDAC module (VDAC_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 VDAC_IntClear() prior to enabling the interrupt.

VDAC_IntGet

```
uint32_t VDAC_IntGet (VDAC_TypeDef * vdac)
```

Get pending VDAC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.

Note

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

Returns

- VDAC interrupt sources pending. Use a bitwise logic OR combination of valid interrupt flags for the VDAC module (VDAC_IF_nnn).

VDAC_IntGetEnabled

```
uint32_t VDAC_IntGetEnabled (VDAC_TypeDef * vdac)
```

Get enabled and pending VDAC interrupt flags.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC 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 VDAC interrupt sources. The return value is the bitwise AND combination of
  - the OR combination of enabled interrupt sources in VDACx_IEN_nnn register (VDACx_IEN_nnn) and
  - the OR combination of valid interrupt flags of the VDAC module (VDACx_IF_nnn).

Copyright © 2026 Silicon Laboratories. All rights reserved.

VDAC_IntSet

```
void VDAC_IntSet (VDAC_TypeDef * vdac, uint32_t flags)
```

Set one or more pending VDAC interrupts from SW.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.
uint32_t	[in]	flags	VDAC interrupt sources to set to pending. Use a bitwise logic OR combination of valid interrupt flags for the VDAC module (VDAC_IF_nnn).

VDAC_GetStatus

```
uint32_t VDAC_GetStatus (VDAC_TypeDef * vdac)
```

Get the VDAC Status register.

Parameters

Type	Direction	Argument Name	Description
VDAC_TypeDef *	[in]	vdac	Pointer to VDAC peripheral register block.

Returns

- Current STATUS register value.

Copyright © 2026 Silicon Laboratories. All rights reserved.

VDAC_Init_TypeDef

VDAC initialization structure, common for both channels.

Public Attributes

uint32_t	warmupTime Number of prescaled CLK_DAC + 1 for the vdac to warmup.
bool	dbgHalt Halt during debug.
bool	onDemandClk Always allow clk_dac.
bool	dmaWakeUp DMA Wakeup.
bool	biasKeepWarm Bias keep warm enable.
VDAC_Refresh_TypeDef	refresh Channel refresh period.
VDAC_TimerOverflow_TypeDef	timerOverflow Internal timer overflow period.
uint32_t	prescaler Prescaler for VDAC clock.
VDAC_Ref_TypeDef	reference Reference voltage to use.
bool	ch0ResetPre Enable/disable reset of prescaler on CH 0 start.
bool	sineReset Sine reset mode.
bool	sineEnable Enable/disable sine mode.
bool	diff Select if single ended or differential output mode.
bool	sineModePrsEnable PRS controlled sinemode enable.
bool	prsOutEnable PRS controlled channel output enable.

Public Attribute Documentation

warmupTime

```
uint32_t VDAC_Init_TypeDef::warmupTime
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Number of prescaled CLK_DAC + 1 for the vdac to warmup.

dbgHalt

```
bool VDAC_Init_TypeDef::dbgHalt
```

Halt during debug.

onDemandClk

```
bool VDAC_Init_TypeDef::onDemandClk
```

Always allow clk_dac.

dmaWakeUp

```
bool VDAC_Init_TypeDef::dmaWakeUp
```

DMA Wakeup.

biasKeepWarm

```
bool VDAC_Init_TypeDef::biasKeepWarm
```

Bias keep warm enable.

refresh

```
VDAC_Refresh_TypeDef VDAC_Init_TypeDef::refresh
```

Channel refresh period.

timerOverflow

```
VDAC_TimerOverflow_TypeDef VDAC_Init_TypeDef::timerOverflow
```

Internal timer overflow period.

prescaler

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
uint32_t VDAC_Init_TypeDef::prescaler
```

Prescaler for VDAC clock.

Clock is source clock divided by prescaler+1.

reference

```
VDAC_Ref_TypeDef VDAC_Init_TypeDef::reference
```

Reference voltage to use.

ch0ResetPre

```
bool VDAC_Init_TypeDef::ch0ResetPre
```

Enable/disable reset of prescaler on CH 0 start.

sineReset

```
bool VDAC_Init_TypeDef::sineReset
```

Sine reset mode.

sineEnable

```
bool VDAC_Init_TypeDef::sineEnable
```

Enable/disable sine mode.

diff

```
bool VDAC_Init_TypeDef::diff
```

Select if single ended or differential output mode.

sineModePrsEnable

```
bool VDAC_Init_TypeDef::sineModePrsEnable
```

PRS controlled sinemode enable.

Copyright © 2026 Silicon Laboratories. All rights reserved.

prsOutEnable

```
bool VDAC_Init_TypeDef::prsOutEnable
```

PRS controlled channel output enable.

Copyright © 2026 Silicon Laboratories. All rights reserved.

VDAC_InitChannel_TypeDef

VDAC channel initialization structure.

Public Attributes

bool	enable Enable channel.
bool	warmupKeepOn Warm-up mode, keep VDAC on (in idle) - or shutdown between conversions.
bool	highCapLoadEnable Select high capacitance load mode in conjunction with high power.
uint32_t	fifoLowDataThreshold Channel x FIFO Low threshold data valid level.
VDAC_RefreshSource_TypeDef	chRefreshSource Channel refresh source.
VDAC_TrigMode_TypeDef	trigMode Channel conversion trigger mode.
VDAC_PowerMode_TypeDef	powerMode Channel power mode.
bool	sampleOffMode Set channel conversion mode to sample/shut-off mode.
uint32_t	pin Vdac channel output pin.
VDAC_ChPortSel_t	port Vdac channel output port.
bool	shortOutput Short High power and low power output.
bool	auxOutEnable Alternative output enable.
bool	mainOutEnable Main output enable.
uint32_t	holdOutTime Channel output hold time.

Public Attribute Documentation

enable

```
bool VDAC_InitChannel_TypeDef::enable
```

Enable channel.

Copyright © 2026 Silicon Laboratories. All rights reserved.

warmupKeepOn

```
bool VDAC_InitChannel_TypeDef::warmupKeepOn
```

Warm-up mode, keep VDAC on (in idle) - or shutdown between conversions.

highCapLoadEnable

```
bool VDAC_InitChannel_TypeDef::highCapLoadEnable
```

Select high capacitance load mode in conjunction with high power.

fifoLowDataThreshold

```
uint32_t VDAC_InitChannel_TypeDef::fifoLowDataThreshold
```

Channel x FIFO Low threshold data valid level.

chRefreshSource

```
VDAC_RefreshSource_TypeDef VDAC_InitChannel_TypeDef::chRefreshSource
```

Channel refresh source.

trigMode

```
VDAC_TrigMode_TypeDef VDAC_InitChannel_TypeDef::trigMode
```

Channel conversion trigger mode.

powerMode

```
VDAC_PowerMode_TypeDef VDAC_InitChannel_TypeDef::powerMode
```

Channel power mode.

sampleOffMode

```
bool VDAC_InitChannel_TypeDef::sampleOffMode
```

Set channel conversion mode to sample/shut-off mode.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Default is continuous.

pin

```
uint32_t VDAC_InitChannel_TypeDef::pin
```

Vdac channel output pin.

port

```
VDAC_ChPortSel_t VDAC_InitChannel_TypeDef::port
```

Vdac channel output port.

shortOutput

```
bool VDAC_InitChannel_TypeDef::shortOutput
```

Short High power and low power output.

auxOutEnable

```
bool VDAC_InitChannel_TypeDef::auxOutEnable
```

Alternative output enable.

mainOutEnable

```
bool VDAC_InitChannel_TypeDef::mainOutEnable
```

Main output enable.

holdOutTime

```
uint32_t VDAC_InitChannel_TypeDef::holdOutTime
```

Channel output hold time.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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	WDOG_SYNC_TIMEOUT 30000 In some scenarioes when the watchdog is disabled the synchronization register might be set and not be cleared until the watchdog is enabled again.
#define	DEFAULT_WDOG WDOG0 Default WDOG instance for deprecated functions.
#define	WDOG_INIT_DEFAULT undefined Suggested default configuration for WDOG initialization structure.

Enumeration Documentation

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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	em1Run Counter keeps running when in EM1.
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.

Copyright © 2026 Silicon Laboratories. All rights reserved.

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.

em1Run

```
bool WDOG_Init_TypeDef::em1Run
```

Counter keeps running when in EM1.

Available for series2.

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

Copyright © 2026 Silicon Laboratories. All rights reserved.

```
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).

perSel

```
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
```

Copyright © 2026 Silicon Laboratories. All rights reserved.

Disable Watchdog reset output if true.

Copyright © 2026 Silicon Laboratories. All rights reserved.

EFR32XG28

API Documentation

List of modules	Description
ACMP - Analog Comparator	Analog comparator (ACMP) Peripheral API.
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.
IADC - Incremental ADC	Incremental Analog to Digital Converter (IADC) Peripheral API.
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.
KEYSCAN - Keyboard Scan	Keyscan (KEYSCAN) Peripheral API.
LCD - Liquid Crystal Display	Liquid Crystal Display (LCD) Peripheral API.
LDMA - Linked DMA	Linked Direct Memory Access (LDMA) Peripheral API.
LESENSE - Low Energy Sensor	Low Energy Sensor (LESENSE) Peripheral API.
LETIMER - Low Energy Timer	Low Energy Timer (LETIMER) Peripheral API.
MSC - Memory System Controller	Memory System Controller API.
PCNT - Pulse Counter	Pulse Counter (PCNT) 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.
SE - Secure Element	Secure Element peripheral API.
SMU - Security Management Unit	Security Management Unit (SMU) Peripheral API.
SYSRTC - System Real Time Counter	System Real Time Counter (SYSRTC) 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.
VDAC - Voltage DAC	Digital to Analog Voltage Converter (VDAC) Peripheral API.
VERSION - Version Defines	Version API.
WDOG - Watchdog	Watchdog (WDOG) Peripheral API.

Copyright © 2026 Silicon Laboratories. All rights reserved.

Copyright © 2026 Silicon Laboratories. All rights reserved.