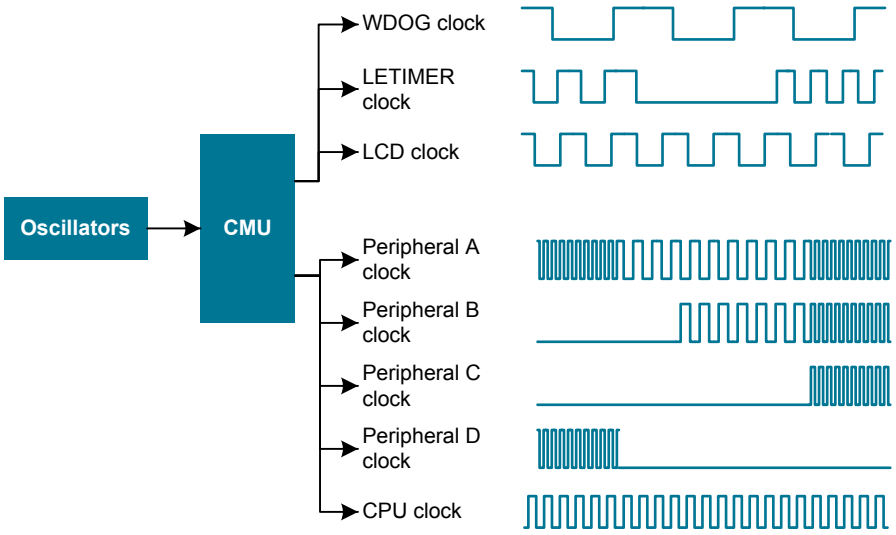
AN0004.1: EFM32 Series 1 and EFR32 Wireless MCU Clock Management Unit (CMU)

This application note provides an overview of the CMU module for EFM32 and EFR32 Wireless Gecko Series 1 devices with explanations on how to choose clock sources, prescaling, and clock calibration.

It also contains information about how to handle oscillators on wake up, external clock sources, and RC oscillator calibration.

KEY POINTS

- The CMU has several internal clock sources available.
- The CMU can also use external high frequency and low frequency clock sources.
- Selecting the right clock source is key for creating low energy applications.
- This application note includes:
  - This PDF document
  - Source files
    - Example C-code
    - Multiple IDE projects



1. Device Compatibility

This application note supports multiple device families, and some functionality is different depending on the device.

EFM32 Series 1 consists of:

- EFM32 Jade Gecko (EFM32JG1/EFM32JG12/EFM32JG13)
- EFM32 Pearl Gecko (EFM32PG1/EFM32PG12/EFM32PG13)
- EFM32 Giant Gecko (EFM32GG11)

EFR32 Wireless Gecko Series 1 consists of:

- EFR32 Blue Gecko (EFR32BG1/EFR32BG12/EFR32BG13)
- EFR32 Flex Gecko (EFR32FG1/EFR32FG12/EFR32FG13)
- EFR32 Mighty Gecko (EFR32MG1/EFR32MG12/EFR32MG13)

2. Functional Description

The Clock Management Unit (CMU) controls the oscillators and clocks. It can enable or disable the clock to the different peripherals individually, as well as enable, disable, or configure the available oscillators. This allows for minimizing energy consumption by disabling the clock for unused peripherals or having them run at lower frequencies.

2.1 Clock Branches

The CMU main and sub clock branches are described in the tables below. Some peripherals have dedicated prescalers, such as the Low Energy peripherals. Other peripheral clocks must be prescaled at the source such that the same prescaled clock is driven to all peripherals using that same source.

A detailed clock tree diagram can be found in the CMU chapter at the beginning of the Functional Description section of a given device's reference manual.

Table 2.1. Blue, Flex, Jade, Mighty, and Pearl Gecko Clock Branches

Main Clock Branch[1]	Clock Source[2]	Sub-clock Branch 1[1]	Sub-clock Branch 2[1]
HFCLK	HFSRCCLK • HFRCO • HFXO • LFRCO • LFXO • HFRCODIV2 • CLKIN0	HFPERCLK	• ACMP0, ACMP1 • ADC0 • CRYOTIMER • CSEN • IDAC0 • I2C0, I2C1 • TIMER0, TIMER1 • TRNG0 • USART0, USART1, USART2, USART3 • WTIMER0, WTIMER1 • VDAC0
		HFCORECLK	CORTEX (Core)
		HFEXPCLK	—
		HFBUSCLK	• CRYPTO0, CRYPTO1 • GPCRC • GPIO • LDMA • LE • PRS
		DBGCLK	—
		HFRADIOCLK	• AGC • BUFC • CRC • FRC • MODEM • PROTIMER • RAC • RFSENSE • SYNTH
Radio Transceiver	HFXO	—	—
F_{ref} (DPLL)	• HFXO • CLKIN0 • LFXO	—	—

Main Clock Branch[1]	Clock Source[2]	Sub-clock Branch 1[1]	Sub-clock Branch 2[1]
AUXCLK	AUXHFRCO	ADC_CLK	—
		DBGCLK	
		LESENSE	—
		MSC	—
LFACLK	• LFRCO • LFXO • ULFRCO	• LESENSE • LETIMER0 • PCNT	—
LFBCLK	• HFCLKLE ($HFBUSCLK_{LE}$/2 or /4) • LFRCO • LFXO • ULFRCO	• CSEN • LEUART0 • SYSTICK	—
LFECLK	• LFRCO • LFXO • ULFRCO	RTCC	—
WDOGCLK	• $HFRCORECLK_{CORETEX}$ • LFRCO • LFXO • ULFRCO	WDOG	—
CRYOCLK	• LFRCO • LFXO • ULFRCO	CRYOTIMER	—
RFSENSECLK	• RF Detector Clock • LFRCO • LFXO • ULFRCO	RFSENSE	—

Note:

1. Not all main and sub clock branches are available on a given device. Refer to the device reference manual and data sheet for details
2. Not all clock sources for main clock branches are available on a given device. Refer to the device reference manual and data sheet for details.

Table 2.2. Giant Gecko Series 1 Clock Branches

Main Clock Branch[1]	Clock Source[2]	Sub-clock Branch 1[1]	Sub-clock Branch 2[1]
HFCLK	HFSRCCLK • HFRCO • HFXO • LFRCO • LFXO • HFRCODIV2 • USHFRCO • CLKIN0	HFPERCLK	• ADC0 • CAN0, CAN1 • TIMER1, TIMER2, TIMER3 TIMER4, TIMER5, TIMER6 • TRNG0 • UART0, UART1 • USART0, USART1, USART3, USART4, USART5 • WTIMER0, WTIMER1, WTIMER2, WTIMER3
		HFPERBCLK	• TIMER0 • USART2
		HFPERCCLK	• ACMP0, ACMP1, ACMP2, ACMP3 • ADC0, ADC1 • CRYOTIMER • CSEN • IDAC0 • I2C0, I2C1, I2C2 • VDAC0
		HFCORECLK	CORTEX (Core)
		HFEXPCLK	—
		HFBUSCLK	• CRYPTO0 • EBI • ETH • GPCRC • GPIO • LDMA • LE • PRS • QSPI0 • SDIO • USB
		DBGCLK	—
F_{ref} (DPLL)	• HFXO • LFXO • USHFRCO • CLKIN0	—	—

Main Clock Branch[1]	Clock Source[2]	Sub-clock Branch 1[1]	Sub-clock Branch 2[1]
AUXCLK	AUXHFRCO	ADC0_CLK	—
		ADC1_CLK	—
		DBGCLK	
		LESENSE	—
		MSC	—
		QSPI0_CLK	—
		SDIO_CLK	—
LFACLK	• LFRCO • LFXO • ULFRCO	• LCD • LESENSE • LETIMER0, LETIMER1 • PCNT • RTC	—
LFBCLK	• HFCLKLE ($HFBUSCLK_{LE}$/2 or /4) • LFRCO • LFXO • ULFRCO	• CSEN • LEUART0, LEUART1 • SYSTICK	—
LFCCLK	• LFRCO • LFXO • ULFRCO	USB	—
LFECLK	• LFRCO • LFXO • ULFRCO	RTCC	—
WDOGnCLK	• $HFRCORECLK_{CORETEX}$ • LFRCO • LFXO • ULFRCO	WDOGn	—
CRYOCLK	• LFRCO • LFXO • ULFRCO	CRYOTIMER	—

Note:

1. Not all main and sub clock branches are available on a given device. Refer to the device reference manual and data sheet for details
2. Not all clock sources for main clock branches are available on a given device. Refer to the device reference manual and data sheet for details.

2.2 Clock Sources

There are a maximum of eight oscillators that can be used as clock sources for different purposes. The HFCLK is usually clocked by the HFXO or HFRCO, whereas low energy peripherals are usually clocked by the LFXO, LFRCO, or ULFRCO. The AUXHFRCO is typically used for the LESENSE, ADC asynchronous mode, flash programming, and the SWO debug output. The USHFRCO is intended primarily for the USB controller on Giant Gecko Series 1 but can also drive the HFCLK, if needed.

Table 2.3. Clock Sources

Oscillator	Frequency Range
HFXO	38 – 40 MHz[1] or 4 to 50 MHz[2]
HFRCO	1 – 38 MHz[1] or 72 MHz[2]
AUXHFRCO	1 – 38[1] MHz or 1 – 50 MHz[2]
USHFRCO	1 – 50 MHz[2]
LFXO	32768 Hz
LFRCO	32768 Hz
PLFRCO[3]	32768 Hz
ULFRCO	1000 Hz

Note:
1. EFM32xG1 and EFR32xG1/xG12/xG13
2. Giant Gecko Series 1 only
3. EFM32xG13 and EFR32xG13

To select the clock source for a branch (e.g. HFCLK, LFA, or LFB), the chosen oscillator must be enabled before it is selected as the clock source. If this is not done, the modules that are running from that clock branch will stop. In the case of selecting a disabled oscillator for the HFCLK branch, the CPU will stop and can only be recovered after a reset.

After a reset, the HFCLK branch is clocked by the HFRCO at the default 19 MHz frequency band, and all low frequency branches are disabled.

Emlib has functions to enable or disable an oscillator and select it as a clock source.

Table 2.4. emlib Functions for Oscillator Enable, Disable and Selection

emlib Function	Usage	Example

`CMU_OscillatorEnable(CMU_Osc_TypeDef osc, bool enable, bool wait)`	Select which oscillator to enable or disable and if it should wait for the oscillator to stabilize before returning.	Enable HFXO and wait for it to stabilize:
		`CMU_OscillatorEnable (cmuOsc_HFXO, true, true);`
		Disable HFRCO:
		`CMU_OscillatorEnable (cmuOsc_HFRCO, false, false);`

`CMU_ClockSelectSet(CMU_Clock_TypeDef clock, CMU_Select_TypeDef ref)`	• Enables the chosen clock source in case it has not been enabled yet.
	• The `clock` parameter is one of the main clock branches, and the `ref` parameter is one of the clock sources for the selected clock branch.
		Select HFXO as the source of HFCLK:
		`CMU_ClockSelectSet (cmuClock_HF, cmuSelect_HFXO);`
		Select LFXO as the source of LFACLK:
		`CMU_ClockSelectSet(cmuClock_LFA, cmuSelect_LFXO);`

2.2.1 Clock Input from a Pin

It is possible to configure the CMU to use an external clock input on the CLKIN0 pin. This clock can be selected as the HFSRCCLK and as the DPLL reference using the CMU_HFCLKSEL and CMU_DPLLCTRL registers, respectively. The input pin must be enabled in the CMU_ROUTEPEN register, and the pin location is selected by the CLKIN0LOC bit field in the CMU_ROUTELOC1 register.

Note: This feature is not available on EFM32xG1 and EFR32xG1 devices.

2.3 Oscillator Configuration

2.3.1 HFXO

The High Frequency Crystal Oscillator (HFXO) is configured to ensure safe startup and operation for most common crystals by default. In order to optimize startup time and power consumption for a given crystal, it is possible to adjust certain oscillator parameters. For more information, refer to application note, *AN0016: Oscillator Design Considerations*.

The HFXO (38 MHz - 40 MHz) needs to be configured to ensure safe startup for the given crystal. The HFXO includes on-chip tunable capacitance, which can replace external load capacitors.

Upon enabling the HFXO, a hardware state machine sequentially applies the configurable startup state and steady state control settings from the CMU_HFXOSTARTUPCTRL and CMU_HFXOSTEADYSTATECTRL registers. Please refer to the device reference manual for the detailed CMU HFXO control state machine flow.

Both the startup state and the steady state of the HFXO require configuration. After reaching the steady operating state, the HFXO configuration can optionally be further tuned to minimize noise and current consumption.

Optimization for noise can be performed using an automatic Peak Detection Algorithm (PDA). Optimization for current can be performed after running the PDA by an automatic Shunt Current Optimization algorithm (SCO). HFXO operation is possible without running PDA and SCO at the cost of higher noise and current consumption than is necessary.

Fixed frequency clock sources (e.g. the HFXO or CLKIN) must be disabled if voltage scaling is used (not availble on EFM32xG1 and EFR32xG1 devices) and the frequency of such sources exceeds the maximum supported system frequency.

Table 2.5. HFXO Configuration

Configuration and Optimization	Bit Field and Register
Configurable startup state	Bit fields in CMU_HFXOSTARTUPCTRL
Configurable Steady State	Bit fields in CMU_HFXOSTEADYSTATECTRL
Optimization for Noise (PDA)[1]	PEAKDETSHUNTOPTMODE bit field in CMU_HFXOCTRL
Optimization for Current (SCO)[1]	PEAKDETSHUNTOPTMODE bit field in CMU_HFXOCTRL

Note:
1. The manual PDA and SCO modes are not recommended for general use. PDA should not be used with an external clock source.

2.3.1.1 Using emlib for HFXO Configuration

Emlib has structures and functions that simplify configuration of the HFXO for efficient operation. Use of emlib is strongly recommended for this reason and also in order to avoid or workaround errata related to the HFXO.

Initialization of the HFXO depends on the structure of type `CMU_HFXOInit_TypeDef`:

```
typedef struct
{
#if defined( _CMU_HFXOCTRL_MASK )
  bool lowPowerMode;                    /**< Enable low-power mode */
  bool autoStartEm01;                   /**< @deprecated Use @ref CMU_HFXOAutostartEnable instead. */
  bool autoSelEm01;                     /**< @deprecated Use @ref CMU_HFXOAutostartEnable instead. */
  bool autoStartSelOnRacWakeup;         /**< @deprecated Use @ref CMU_HFXOAutostartEnable instead. */
  uint16_t ctuneStartup;                /**< Startup phase CTUNE (load capacitance) value */
  uint16_t ctuneSteadyState;            /**< Steady-state phase CTUNE (load capacitance) value */
  uint8_t regIshSteadyState;            /**< Shunt steady-state current */
  uint8_t xoCoreBiasTrimStartup;        /**< Startup XO core bias current trim */
  uint8_t xoCoreBiasTrimSteadyState;    /**< Steady-state XO core bias current trim */
  uint8_t thresholdPeakDetect;          /**< Peak detection threshold */
  uint8_t timeoutShuntOptimization;     /**< Timeout - shunt optimization */
  uint8_t timeoutPeakDetect;            /**< Timeout - peak detection */
  uint8_t timeoutSteady;                /**< Timeout - steady-state */
  uint8_t timeoutStartup;               /**< Timeout - startup */
#else
  uint8_t boost;                        /**< HFXO Boost, 0=50% 1=70%, 2=80%, 3=100% */
  uint8_t timeout;                      /**< Startup delay */
  bool glitchDetector;                  /**< Enable/disable glitch detector */
#endif
  CMU_OscMode_TypeDef mode;             /**< Oscillator mode */
} CMU_HFXOInit_TypeDef;
```

Structure members can be set by the user, otherwise the default structures `CMU_HFXOINIT_DEFAULT` and `CMU_HFXOINIT_EXTERNAL_CLOCK` can be used as templates for HFXO initialization.

```
#if defined( _EFR_DEVICE )
#define CMU_HFXOINIT_DEFAULT                                                \
{                                                                           \
  false,        /* Low-noise mode for EFR32 */                            \
  false,        /* @deprecated no longer in use */                        \
  false,        /* @deprecated no longer in use */                        \
  false,        /* @deprecated no longer in use */                        \
  _CMU_HFXOSTARTUPCTRL_CTUNE_DEFAULT,                                       \
  _CMU_HFXOSTEADYSTATECTRL_CTUNE_DEFAULT,                                   \
  _CMU_HFXOSTEADYSTATECTRL_REGISH_DEFAULT,                                  \
  0x20,         /* Matching errata fix in CHIP_Init() */                    \
  0x7,          /* Recommended steady-state XO core bias current */         \
  0x6,          /* Recommended peak detection threshold */                  \
  _CMU_HFXOTIMEOUTCTRL_SHUNTOPTTIMEOUT_DEFAULT,                             \
  0xA,          /* Recommended peak detection timeout  */                   \
  0x4,          /* Recommended steady timeout */                            \
  _CMU_HFXOTIMEOUTCTRL_STARTUPTIMEOUT_DEFAULT,                              \
  cmuOscMode_Crystal,                                                       \
}
#else /* EFM32 device */
#define CMU_HFXOINIT_DEFAULT                                                \
{                                                                           \
  true,         /* Low-power mode for EFM32 */                            \
  false,        /* @deprecated no longer in use */                        \
  false,        /* @deprecated no longer in use */                        \
  false,        /* @deprecated no longer in use */                        \
  _CMU_HFXOSTARTUPCTRL_CTUNE_DEFAULT,                                       \
  _CMU_HFXOSTEADYSTATECTRL_CTUNE_DEFAULT,                                   \
  _CMU_HFXOSTEADYSTATECTRL_REGISH_DEFAULT,                                  \
  0x20,         /* Matching errata fix in CHIP_Init() */                    \
  0x7,          /* Recommended steady-state osc core bias current */        \
  0x6,          /* Recommended peak detection threshold */                  \
  _CMU_HFXOTIMEOUTCTRL_SHUNTOPTTIMEOUT_DEFAULT,                             \
  0xA,          /* Recommended peak detection timeout  */                   \
  0x4,          /* Recommended steady timeout */                            \
  _CMU_HFXOTIMEOUTCTRL_STARTUPTIMEOUT_DEFAULT,                              \
```

```
  cmuOscMode_Crystal,                                                   \
}
#endif /* _EFR_DEVICE */
#define CMU_HFXOINIT_EXTERNAL_CLOCK                                     \
{                                                                       \
  true,         /* Low-power mode */                                    \
  false,        /* @deprecated no longer in use */                      \
  false,        /* @deprecated no longer in use */                      \
  false,        /* @deprecated no longer in use */                      \
  0,            /* Startup CTUNE=0 recommended for external clock */    \
  0,            /* Steady  CTUNE=0 recommended for external clock */    \
  _CMU_HFXOSTEADYSTATECTRL_REGISH_DEFAULT,                              \
  0,            /* Startup IBTRIMXOCORE=0 recommended for external clock */  \
  0,            /* Steady  IBTRIMXOCORE=0 recommended for external clock */  \
  0x6,          /* Recommended peak detection threshold */              \
  _CMU_HFXOTIMEOUTCTRL_SHUNTOPTTIMEOUT_DEFAULT,                         \
  0x0,          /* Peak-detect not recommended for external clock usage */  \
  _CMU_HFXOTIMEOUTCTRL_STEADYTIMEOUT_2CYCLES,  /* Minimal steady timeout */  \
  _CMU_HFXOTIMEOUTCTRL_STARTUPTIMEOUT_2CYCLES, /* Minimal startup timeout */  \
  cmuOscMode_External,                                                  \
}
```

The HFXO initialization structure is used as an argument when calling the `CMU_HFXOInit(const CMU_HFXOInit_TypeDef *hfxoInit)` function, which writes the HFXO initialization parameters to the relevant CMU registers. After calling this function call, the HFXO can be enabled and selected as the source of HFCLK as shown below.

```
/* Initialize HFXO with specific parameters */
CMU_HFXOInit_TypeDef hfxoInit = CMU_HFXOINIT_DEFAULT;
CMU_HFXOInit(&hfxoInit);

/* Enable and set HFXO for HFCLK */
CMU_ClockSelectSet(cmuClock_HF, cmuSelect_HFXO);
```

2.3.2 LFXO

The Low Frequency Crystal Oscillator (LFXO) is configured to ensure safe startup and operation for most common crystals by default. In order to optimize startup time and power consumption for a given crystal, it is possible to adjust certain oscillator parameters. For more information, refer to application note, *AN0016: Oscillator Design Considerations*.

The LFXO includes on-chip tunable capacitance, which can replace external load capacitors. The LFXO is configured by bit fields in the CMU_LFXOCTRL register. Note that these bit fields should set only during initialization and are not be changed while the LFXO is enabled.

Table 2.6. CMU_LFXOCTRL Register for LFXO Configuration

Bit Field	Usage
GAIN	Adjusts the oscillator startup gain .
TUNING	Tunes the internal load capacitance connected between LFXTAL_P and ground and LFXTAL_N and ground symmetrically.
HIGHAMPL	Setting this bit drives the crystal with a higher amplitude waveform, which in turn provides safer operation, somewhat improves duty cycle, and lowers sensitivity to noise at the cost of increased current consumption.
AGC	Setting this bit enables Automatic Gain Control, which limits the amplitude of the driving waveform in order to reduce current draw. When AGC is disabled, the LFXO runs at the startup current, and the crystal will oscillate rail to rail, providing safer operation, improved duty cycle, and lower sensitivity to noise at the cost of increased current consumption.
BUFCUR	The default value is intended to cover all use cases and reprogramming is not recommended.
CUR	The default value is intended to cover all use cases and reprogramming is not recommended.

2.3.2.1 Using emlib for LFXO Configuration

Emlib has structures and functions that simplify configuration of the LFXO for efficient operation. Use of emlib is strongly recommended for this reason and also in order to avoid or workaround errata related to the LFXO.

Initialization of the LFXO depends on the structure of type `CMU_LFXOInit_TypeDef`:

```
typedef struct
{
#if defined( _CMU_LFXOCTRL_MASK )
  uint8_t ctune;                        /**< CTUNE (load capacitance) value */
  uint8_t gain;                         /**< Gain / max startup margin */
#else
  CMU_LFXOBoost_TypeDef boost;          /**< LFXO boost */
#endif
  uint8_t timeout;                      /**< Startup delay */
  CMU_OscMode_TypeDef mode;             /**< Oscillator mode */
} CMU_LFXOInit_TypeDef;
```

Structure members can be set by the user, otherwise the default structures `CMU_LFXOINIT_DEFAULT` and `CMU_LFXOINIT_EXTERNAL_CLOCK` below can be used as templates for LFXO initialization.

```
#define CMU_LFXOINIT_DEFAULT                                                \
  {                                                                         \
    _CMU_LFXOCTRL_TUNING_DEFAULT,   /* Default CTUNE value, 0 */            \
    _CMU_LFXOCTRL_GAIN_DEFAULT,     /* Default gain, 2 */                   \
    _CMU_LFXOCTRL_TIMEOUT_DEFAULT,  /* Default start-up delay, 32k cycles */ \
    cmuOscMode_Crystal,             /* Crystal oscillator */                \
  }
#define CMU_LFXOINIT_EXTERNAL_CLOCK                                         \
  {                                                                         \
    0,                              /* No CTUNE value needed */             \
    0,                              /* No LFXO startup gain */              \
    _CMU_LFXOCTRL_TIMEOUT_2CYCLES,  /* Minimal lfxo start-up delay, 2 cycles */ \
    cmuOscMode_External,            /* External digital clock */            \
  }
```

The LFXO initialization structure is used as an argument when calling the `CMU_LFXOInit(const CMU_LFXOInit_TypeDef *lfxoInit)` function, which writes the LFXO initialization parameters to the relevant CMU registers. After calling this function, the LFXO can be enabled and selected as the clock source for low energy peripherals.

```
/* Initialize LFXO with specific parameters */
CMU_LFXOInit_TypeDef lfxoInit = CMU_LFXOINIT_DEFAULT;
CMU_LFXOInit(&lfxoInit);

/* Enable and set LFXO for LFACLK */
CMU_ClockSelectSet(cmuClock_LFA, cmuSelect_LFXO);
```

2.3.3 HFRCO, AUXHFRCO, and USHFRCO

The HFRCO, AUXHFRCO, and USHFRCO (available on Giant Gecko Series 1 only) default to operation in the 19 MHz band but can be switched to operate in one of the other preset frequency bands by changing the FREQRANGE bit field in the CMU_HFRCOCTRL, CMU_AUXHFRCOCTRL, or CMU_USHFRCOCTRL registers. The USHFRCO (Universal HFRCO) is functionally identical to the AUXHFRCO and can provide the clock for the QSPI, SDIO, or USB independent of the HFSRCCLK.

Table 2.7. High Frequency RC Oscillator Band Selection

Oscillator	Frequency Band
AUXHFRCO	• 1 MHz (DIV4 from 4 MHz) • 2 MHz (DIV2 from 4 MHz) • 4 MHz • 7 MHz • 13 MHz • 16 MHz • 19 MHz (Default) • 26 MHz • 32 MHz • 38 MHz • 48 MHz (Giant Gecko Series 1 only) • 50 MHz (Giant Gecko Series 1 only)
HFRCO	• 1 MHz (DIV4 from 4 MHz) • 2 MHz (DIV2 from 4 MHz) • 4 MHz • 7 MHz • 13 MHz • 16 MHz • 19 MHz (Default) • 26 MHz • 32 MHz • 38 MHz • 48 MHz (Giant Gecko Series 1 only) • 56 MHz (Giant Gecko Series 1 only) • 64 MHz (Giant Gecko Series 1 only) • 72 MHz (Giant Gecko Series 1 only)
USHFRCO (Giant Gecko Series 1 only)	• 1 MHz (DIV4 from 4 MHz) • 2 MHz (DIV2 from 4 MHz) • 4 MHz • 7 MHz • 13 MHz • 16 MHz • 19 MHz (Default) • 26 MHz • 32 MHz • 38 MHz • 48 MHz • 50 MHz

The specific values that select each tuning band are written to the TUNING bit field in the CMU_HFRCOCTRL and CMU_AUXHFRCOCTRL registers. Each band is calibrated during production, with suitable tuning values written to the Device Information (DI) page.

The HFRCO and AUXHFRCO frequency can be more accurately tuned at the cost of increased current consumption via the FINETUNING and FINETUNINGEN bit fields in the CMU_HFRCOCTRL and CMU_AUXHFRCOCTRL registers.

When Voltage Scaling is used (not available on EFM32xG1 and EFR32xG1 devices), the HFRCO and AUXHFRCO must be tuned to frequencies that do not exceed the maximum system frequency permitted with scaling before voltage scaling is actually applied.

Emlib has specific functions to change the frequency band to which the AUXHFRCO and HFRCO are tuned. Use of emlib is strongly recommended because it specifically handles the need to increase the number of flash wait states or voltage scaling when a higher frequency tuning band is selected. These functions switch to the desired frequency band by loading the correct tuning value from the Device Information (DI) page.

Table 2.8. emlib Functions for Changing the AUXHFRCO and HFRCO Tuning Bands

emlib Function	Usage	Example
`CMU_HFRCOBandSet(CMU_HFRCOFreq_TypeDef setFreq)`	Change HFRCO frequency band.	`CMU_HFRCOBandSet(cmuHFRCOFreq_26M0Hz);`
`CMU_AUXHFRCOBandSet(CMU_AUXHFRCOFreq_TypeDef setFreq)`	Change AUXHFRCO frequency band.	`CMU_ AUXHFRCOBandSet(cmuAUXHFRCOFreq_13M0Hz);`

2.3.4 LFRCO

It is possible to calibrate the LFRCO to achieve higher accuracy. The frequency is adjusted by changing the TUNING bi tfield in the CMU_LFRCOCTRL register. The LFRCO is also calibrated in production, and its default TUNING value is set during reset.

Voltage reference duty cycling can be enabled for the LFRCO by setting the ENVREF bit in the CMU_LFRCOCTRL register before enabling the LFRCO. This reduces current consumption at the expense of slightly worse frequency accuracy, especially at high temperatures. Comparator chopping and dynamic element matching are enabled with the ENCHOP and ENDEM bits in the CMU_LFRCOCTRL register, which improves the average LFRCO frequency accuracy at the cost of reduced cycle-to-cycle accuracy (increased jitter).

2.3.5 PLFRCO

The Precision Low Frequency RC Oscillator (PLFRCO), is a 500 ppm RC oscillator that eliminates the need for a 32.768kHz crystal in Bluetooth Low Energy applications by automatically recalibrating itself against the HFXO. The PLFRCO operates in energy modes EM0/1/2/3. In both hibernate (EM4H) and shutdown modes (EM4S), the PLFRCO is powered down and thus cannot provide a clock to any peripherals that can operate in these modes.

Several times a second, the PLFRCO checks for a relative change in temperature. If the change is large enough, the PLFRCO will start the HFXO and recalibrate to maintain 500 ppm. Temperature checking is also performed in EM2/3, and the PLFRCO can wake from these modes, go to EM1, start the HFXO, recalibrate, and go back to EM2/3 automatically without any software interaction.

After being enabled, the PLFRCO will perform a one time baseline calibration to determine the minimum and maximum frequencies of the oscillator, perform a calibration to adjust its fine trim tuning, and take a relative temperature measurement. The temperature measurement is not correlated to a specific temperature in degrees centigrade. Instead, it is a part-specific measurement that allows the PLFRCO to determine when the temperature has changed by a small amount. Once a large enough change has occured, the PLFRCO will recalibrate against the HFXO.

For the PLFRCO to operate correctly, the HFXO must be connected to a 38.4 MHz crystal. Other frequencies are not supported and, along with failure of the HFXO to start, cause the PLFRCOHFXODNSERR error bit in the CMU Interrupt Flag (CMU_IF) register to be set.

2.3.6 ULFRCO

The ULFRCO is always on in EM0, EM1, EM2, EM3 and EM4H and cannot be disabled via the CMU_OSCENCMD register. As such, it is always available as a clock source for many of the peripherals in the low-frequency clock domains. It is not possible to calibrate the ULFRCO to achieve higher accuracy.

2.4 Oscillator Start-Up Time and Time-Out

The start-up time for each of the previously discussed oscillators differs and, in certain cases, can be further extended by one or more programmable time-out delays. When enabled, the oscillator's output is given time to stabilize by stalling assertion of its ready signal for the specified number of clock cycles. Low start-up times can be selected when the clock is coming from a high quality source, while longer time-out delays are necessary when the clock is coming directly from a crystal.

Programmable time-outs are available for the LFXO, HFXO, and LFRCO. The HFRCO, AUXHFRCO, PLFRCO, ULFRCO, and USHFRCO time-out delays are fixed and cannot be changed.

Table 2.9. Oscillator Time-Out Configuraton

Oscillator	Bit Field	Register
LFXO	TIMEOUT	CMU_LFXOCTRL
HFXO	• STARTUPTIMEOUT • STEADYTIMEOUT • PEAKDETTIMEOUT • SHUNTOPTTIMEOUT	CMU_HFXOTIMEOUTCTRL
LFRCO	TIMEOUT	CMU_LFRCOCTRL
HFRCO	Start-up time is fixed.	
AUXHFRCO	Start-up time is fixed.	
PLFRCO	Start-up time is fixed.	
ULFRCO	Start-up time is fixed.	
USHFRCO	Start-up time is fixed.	

2.4.1 HFXO Deterministic Start-up Time

A second time-out counter allows the HFXO to achieve deterministic start-up timing based on the LFXO, ULFRCO, or LFRCO. It runs off the LFECLK and can even be used during wake-up from EM2 when either the ULFRCO, LFRCO, or LFXO is already running and stable (see 5.2.3 Automatic HFXO Start with Deterministic Start-Up).

When enabled, this counter delays HFXO ready assertion by the number of LFECLK cycles specified by the LFTIMEOUT bit field in the CMU_HFXOCTRL register. The HFXO ready signal is asserted when the time-out counters configured via the CMU_HFXOTIMEOUTCTRL register and the LFTIMEOUT counter have both elapsed. If the LFTIMEOUT countdown completes first, the LFTIMEOUTERR bit in CMU_IF will be set. Note that use of LFTIMEOUT requires the peripheral causing the wake-up from EM2 to reside in the LFECLK domain.

2.5 Prescaling

Each of the clock sub-branches derived from the HFCLK, such as the HFCORECLK, HFBUSCLK, and HFPERCLK, can be individually prescaled by a factor of 1 to 512, while the HFCLK itself can be divided down by a value between 1 and 32. The HFCORECLK drives the CPU and modules tightly coupled to it like the flash (MSC) and SRAM. The HFBUSCLK does the same for modules which can read and write data on their own like the LDMA and USB but also for modules requiring fast response to CPU reads and writes, such as the GPIO and external bus interface (EBI). The HFPERCLK and its branches (HFPERBCLK and HFPERCCLK) drive high-frequency peripherals like the timers and USARTs. Some peripherals permit further clock prescaling (such as the ADC), which is controlled independently in each such peripheral's register space.

Prescaling is controlled independently by dedicated bit fields for most peripherals in the low frequency clock domains (LFA, LFB, and LFE), with the exception of the Pulse Counter (PCNT).

Table 2.10. Prescaler of Clock Branches

Clock Branch	Prescaler Bitfield[1]	Prescaler Register	Prescaler Range
HFCLK	PRESC	CMU_HFPRESC	1 to 32
HFPERCLK	PRESC	CMU_HFPERPRESC	1 to 512
HFPERBCLK	PRESC	CMU_HFPERPRESCB	1 to 512
HFPERCCLK	PRESC	CMU_HFPERPRESCC	1 to 512
HFRADIOCLK	PRESC	CMU_HFRADIOPRESC	1 to 512
HFCORECLK	PRESC	CMU_HFCOREPRESC	1 to 512
HFEXPCLK	PRESC	CMU_HFEXPPRESC	1 to 512
LFACLK	LCD	CMU_LFAPRESC0	2^0 to 2^7
	LESENSE		2^0 to 2^3
	LETIMER0, LETIMER1		2^0 to 2^{15}
	RTC		2^0 to 2^{15}
LFBCLK	CSEN	CMU_LFBPRESC0	2^4 to 2^7
	LEUART0, LEUART1		2^0 to 2^3
	SYSTICK		2^0
LFECLK	RTCC	CMU_LFEPRESC0	2^0 or 2^0 to 2^2

Note:
1. Not all prescaler bitfields are available on a given device. Refer to the device reference manual and data sheet for details.

Emlib has functions to set the clock prescalers, and its use is highly encouraged because synchronization is required for prescalers in the low frequency clock domains.

Table 2.11. emlib Clock Divider and Prescaler Functions

emlib Function	Divider/Prescaler	Example
`CMU_ClockDivSet(CMU_Clock_TypeDef clock, CMU_ClkDiv_TypeDef div)`	One of the enumerated power-of-2 dividers of type CMU_ClkDiv_TypeDef	Divide by 32768 (2^{15}) for LETIMER0: `CMU_ClockDivSet(cmuClock_LETIMER0, cmuClkDiv_32768);`
`CMU_ClockPrescSet(CMU_Clock_TypeDef clock, uint32_t presc)`	An integer prescaler between 1 and 512	HFPERCLK divide by 6: `CMU_ClockDivSet(cmuClock_HFPER, 6);`

When using these functions, careful consideration is required for both parameters. Not all clocks have a prescaler, and the maximum prescaling value is also not the same for the different clocks (HFPERCLK, for instance, has a maximum of 512 while the LETIMER clocks can be divided by 32768).

2.6 Flash Wait States

When increasing the memory subsystem clock (HFCLK) frequency above certain limits, the number of wait states required for flash read accesses must be increased **before** the frequency change is performed. Likewise, the number of wait states can be reduced when the HFCLK frequency falls below certain limits, and this must be perfomred **after** the frequency change has taken place. Wait state and frequency changes are properly sequenced and handled automatically by emlib's CMU functions, and their use is strongly encouraged. Changes to the number of wait states can be made under user control directly by writing to the MODE bit field in the MSC_READCTRL register.

Table 2.12. Flash Wait States for EFM32xG1 and EFR32xG1

Clock Frequency	Minimum Flash Wait States
HFCLK ≤ 32 MHz	0
HFCLK > 32 MHz	1

Table 2.13. Flash Wait States for EFM32xG12/13 and EFR32xG12/13

Clock Frequency	Minimum Flash Wait States
HFCLK ≤ 25 MHz	0
HFCLK > 25 MHz	1

Table 2.14. Flash Wait States for Giant Gecko Series 1

Clock Frequency	Minimum Flash Wait States
HFCLK ≤ 18 MHz	0
18 MHz < HFCLK ≤ 36 MHz	1
36 MHz < HFCLK ≤ 54 MHz	2
54 MHz < HFCLK ≤ 72 MHz	3

Voltage scaling (not supported on EFM32xG1 and EFR32xG1 devices) allows for a tradeoff between power and performance. The flash wait states for voltage scaling level 0 are described in table below. Operation above 21 MHz with voltage scaling is not supported.

Table 2.15. Flash Wait States at Voltage Scaling Level 0

Clock Frequency	Minimum Flash Wait States
HFCLK <= 7 MHz	0
7 MHz < HFCLK<= 14 MHz	1
14 MHz < HFCLK<= 21 MHz	2

The emlib functions below optimize flash access wait-state configuration if the source or frequency of HFCLK is changed when invoking these functions.

- `CMU_ClockDivSet(CMU_Clock_TypeDef clock, CMU_ClkDiv_TypeDef div)`
- `CMU_ClockPrescSet(CMU_Clock_TypeDef clock, CMU_ClkPresc_TypeDef presc)`
- `CMU_ClockSelectSet(CMU_Clock_TypeDef clock, CMU_Select_TypeDef ref)`
- `CMU_HFRCOBandSet(CMU_HFRCOFreq_TypeDef setFreq)`
- `CMU_HFXOAutostartEnable(uint32_t userSel, bool enEM0EM1Start, bool enEM0EM1StartSel)`

2.7 Low Energy (LE) Peripheral Clcok Divider

To use the low energy peripheral modules, the LE interface clock (HFCLKLE) must be enabled in addition to the module clock.

The HFCLKLEPRESC bitfield in the CMU_HFPRESC register is used to divide the $HFBUSCLK_{LE}$ frequency to ensure the HFCLKLE frequency is within the limit.

The WSHFLE bitfield in the CMU_CTRL register is used to ensure that the Low Energy Peripheral Interface is able to operate at the given $HFBUSCLK_{LE}$ frequency by inserting wait states when using this interface.

Table 2.16. Low Energy Peripheral Clock Divider

Device	$HFBUSCLK_{LE}$ Maximum	HFCLKLE Maximum	WSHFLE	HFCLKLEPRESC
• EFM32xG1 • EFM32xG12 • EFM32xG13 • EFR32xG1 • EFR32xG12 • EFR32xG13	40 MHz	16 MHz	• 0 or 1 if $HFBUSCLK_{LE}$ <= 32 MHz • 1 if $HFBUSCLK_{LE}$ > 32 MHz	• DIV2 or DIV4 if $HFBUSCLK_{LE}$ <= 32 MHz • DIV4 if $HFBUSCLK_{LE}$ > 32 MHz

Before going to a high frequency, make sure the registers in the table above have the correct values. When going down in frequency, make sure to keep the registers at the values required by the higher frequency until after the switch completes.

To make this task easier, there is also a function available in the emlib to ensure the registers in the table above have the correct values when enabling the LE clock.

Table 2.17. The emlib Function to Enable a Module Clock

emlib Function	Usage	Example
`CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable)`	To enable a module clock.	Enable LE clock with correct settings on corresponding registers: `CMU_ClockEnable(cmuClock_HFLE, true);`

2.8 External Clock Sources

By default, the LFXO and HFXO are started in crystal mode, but it is possible to connect an active external sine wave or square wave clock source to the LFXTAL_N and HFXTAL_N pins of the LFXO and HFXO as discussed in the following sections.

2.8.1 External Sine Wave

An AC-coupled, externally buffered sine wave can be applied to the HFXTAL_N or LFXTAL_N pin. The amplitude of this signal must be at least 200 mV peak-to-peak, and the frequency is subject to the same limit as that of a crystal connected to the HFXO or LFXO.

Each oscillator has a way to be configured for use with a stimulus other than a crystal. In the case of the HFXO, the MODE bit in the CMU_HFXOCTRL register is set to indicate that an external clock source (EXTCLK) is being used. Sine wave operation for the LFXO is selected by the BUFEXTCLK setting of the CMU_LFXOCTRL register's MODE bit field.

2.8.2 Digital External Clock

A rail-to-rail square wave with 50% duty cycle can be applied to the HFXTAL_N or LFXTAL_N pin, subject to the same frequency limit as that of a crystal connected to the HFXO or LFXO.

Each oscillator has a way to be configured for use with a stimulus other than a crystal. In the case of the HFXO, the MODE bit in the CMU_HFXOCTRL register is set to indicate that an external clock source (EXTCLK) is being used. Operation with a digital external clock for the LFXO is selected by the DIGEXTCLK setting of the CMU_LFXOCTRL register's MODE bit field.

2.8.3 Oscillator Pin Availability

When a clock is supplied via LFXTAL_N, the corresponding LFXTAL_P pin is not needed and can be used for GPIO or peripheral functionality. HFXTAL_P availability when not used to connect a crystal is device-dependent and can be found in the datasheet pin alternate functionality tables.

2.9 Output Clock to Pin

The CMU can drive user-specified clocks on certain pins. Selection is done using the CLKOUTSELn bit fields in the CMU_CTRL register. It is also necessary to (a) select which pins are driven in the CMU_ROUTELOC0 register, (b) configure them as outputs in the appropriate GPIO pin mode registers, and (c) enable each one via its corresponding bit in the CMU_ROUTEPEN register.

Note that a clock sourced from an oscillator (e.g. the HFXO) can be unstable after startup and should not be output on a pin before its corresponding ready flag (e.g. HFXORDY) is set in the CMU_STATUS register.

Table 2.18. Clock Output on a Pin

Pin	Bit Field and Register
CMU_OUT0	CLKOUTSEL0 in CMU_CTRL: • ULFRCO (directly from oscillator) • LFRCO (directly from oscillator) • LFXO (directly from oscillator) • HFXO (directly from oscillator) • HFEXPCLK • ULFRCOQ (qualified) • LFRCOQ (qualified) • LFXOQ (qualified) • HFRCOQ (qualified) • AUXHFRCOQ (qualified) • HFXOQ (qualified) • HFSRCCLK • PLFRCO (directly from oscillator)[1] • PLFRCOQ (qualified)[1] • USHFRCOQ (qualified)[2]
CMU_OUT1	CLKOUTSEL1 in CMU_CTRL: • ULFRCO (directly from oscillator) • LFRCO (directly from oscillator) • LFXO (directly from oscillator) • HFXO (directly from oscillator) • HFEXPCLK • ULFRCOQ (qualified) • LFRCOQ (qualified) • LFXOQ (qualified) • HFRCOQ (qualified) • AUXHFRCOQ (qualified) • HFXOQ (qualified) • HFSRCCLK • USHFRCOQ (qualified)[2]

Pin	Bit Field and Register
CMU_OUT2[2]	CLKOUTSEL2 in CMU_CTRL: • ULFRCO (directly from oscillator) • LFRCO (directly from oscillator) • LFXO (directly from oscillator) • HFXODIV2Q (HFXO ÷ 2 qualified) • HFXO (directly from oscillator) • HFEXPCLK • HFXOX2Q (HFXO × 2 qualified) • ULFRCOQ (qualified) • LFRCOQ (qualified) • LFXOQ (qualified) • HFRCOQ (qualified) • AUXHFRCOQ (qualified) • HFXOQ (qualified) • USHFRCOQ (qualified)
Location	CLKOUT0LOC, CLKOUT1LOC, and CLKOUT2LOC[2]in CMU_ROUTELOC0
Enable	CLKOUT0PEN, CLKOUT1PEN, and CLKOUT2PEN[2] in CMU_ROUTEPEN register.

Note:

1. Blue/Flex/Mighty Gecko 13 families and Jade/Pearl Gecko 13 only
2. Giant Gecko Series 1 only

2.10 Clock Output to PRS

The CMU can be used as a PRS producer with the clocks selected by the CLKOUTSEL bit fields in the CMU_CTRL register directed to any of the PRS consumers.

Other than picking the clock(s) desired via the CLKOUTSEL bit field(s), no additional configuration of the CMU is required. Instead, a given PRS channel routes one of the CMU clock outputs to a consumer via the SOURCESEL and SIGSEL fields in the PRS_CHx_CTRL register.

Apart from its typical uses (e.g. for ADC conversion start timing), the PRS can be used as an enhanced clock output routing mechanism. Depending on the device, each PRS channel can be output to anywhere between 3 and 17 different pins, making it possible to route the different CMU clock outputs to pins not otherwise made available via location bit fields in the CMU_ROUTELOC0 register.

2.11 Interrupts

The CMU can request interrupts from multiple sources that are serviced by a single interrupt vector. If CMU interrupts are enabled, an interrupt is requested when one or more of the flags in CMU_IF register and their corresponding bits in CMU_IEN register are set.

The LFRCORDY and LFXORDY interrupts can be used to wake from EM2. In this way, waiting for either the LFRCO or LFXO to stablilize by polling its associated ready bit can be avoided by going into EM2 after enabling the oscillator and sleeping until the ready interrupt causes a wake-up.

2.12 Automatic HFXO Start

EFM32 and EFR32 Wireless Gecko Series 1 devices can enable the HFXO and select it as the HFSRCCLK automatically under hardware control. This is intended to minimize current draw by allowing the system to remain in a low-energy mode until the HFXO is ready instead of having software poll the HFXORDY bit in the CMU_STATUS or CMU_IF register.

The AUTOSTARTRDYSELRAC bit in the HFXOCTRL register on the EFR32 Wireless Gecko Series 1family permits automatic HFXO start-up and its selection as the HFSRCCLK when ready in response to the wake-up event from the Radio Controller (RAC). The CMU documentation for these devices describes a typical use case whereby the RAC (which must run from the HFXO) periodically wakes from EM2 to perform work and then goes back to sleep all without software intervention courtesy of this feature.

A separate mechanism controlled by the AUTOSTARTSELEM0EM1 and AUTOSTARTEM0EM1 bits in the CMU_HFXOCTRL register performs a similar function depending on the current energy mode.

Table 2.19. Energy Mode-Dependent HFXO Start-up and HFSRCCLK Selection

AUTOSTARTSELEM0EM1	AUTOSTARTEM0EM1	Action
0	0	None. HFXO enabled in software as usual.
0	1	• Automatic start-up of the HFXO in EM0/EM1 (also after entry from EM2/EM3). • HFSRCCLK remains unchanged.
1	X	• Automatic start-up of the HFXO in EM0/EM1 (also after entry from EM2/EM3). • Immediate selection of the HFXO as the HFSRCCLK • HFSRCCLK stalled until the HFXO is ready.

When AUTOSTARTSELEM0EM1 is set to 1 in EM0/EM1 (regardless of the state of the AUTOSTARTRDYSELRAC or AUTOSTARTEM0EM1 bits), selection of the HFXO as the HFSRCCLK will occur immediately, even if HFXO is not yet ready. Upon wake-up into EM0/EM1, this can therefore lead to a relatively long start-up time as the system will not operate from the HFRCO as it would otherwise normally do (see 5.2.2 Automatic HFXO Start with Normal Start-Up and 5.2.3 Automatic HFXO Start with Deterministic Start-Up).

Care should be taken to ensure that the settings of the MSC_READCTRL (flash wait states) and CMU_CTRL (WSHFLE) registers are compatible with the HFXO frequency before enabling the automatic start-up feature. This is properly handled when HFXO auto-start is enabled using the emlib `CMU_HFXOAutostartEnable()` function.

2.13 Digital Phased-Locked Loop (DPLL)

The Digital Phase-Locked Loop (DPLL) uses a digitally controlled oscillator (DCO) and 12-bit N and M dividers as part of a feedback mechanism that allows more accurate tuning of the HFRCO clock output.

Note: EFR32 Wireless Gecko Series 1and EFM32xG1 devices do not include the DPLL.

Table 2.20. Digital Phase-Locked Loop Operation

Item	Operation
Frequency-lock loop mode (FREQLL)	• Keeps the DCO frequency-locked to the reference clock, which means the DCO frequency will be accurate. • Phase error can accumulate over time and cause the average frequency error to be non-zero. • FREQLL mode provides better jitter and transient performance and should be used unless a specific phase requirement exists.
Phase-lock loop mode (PHASELL)	• Keeps the DCO phase-locked to the reference clock,. • Phase error does not accumulate over time. so the average frequency error converges to zero.
Reference clock source (FREF)	Selected by REFSEL bit field in the CMU_DPLLCTRL register: • HFXO • LFXO • CLKIN0
Output frequency	• Output frequency = FREF × (N+1)/(M+1), where N (must > 32) and M are 12-bit values written to the CMU_DPLLCTRL1 register. • The DCO lock time increases and DCO jitter decreases as N increases. • N+1 > 300 recommended as a reasonable compromise between lock time and jitter unless a specific lock time is required. • All configuration must be done before enabling the DPLL and must not be changed while the DPLL is running.
Lock detection	Associated interrupt flag and enable bits in the CMU_IF and CMU_IEN registers: • DPLLRDY is set when the DPLL successfully locks to the reference clock based on the programmed configuration. • DPLLLOCKFAILLOW is set when DPLL fails to lock because the period lower boundary is hit. • DPLLLOCKFAILHIGH is set when DPLL fail to lock because the period upper boundary is hit.
Output spectrum spreading	The DPLL can randomize the generated output period by a configurable amount of spread.
Enabling and disabling	• Software enable and disable bits reside in the CMU_OSCENCMD register. • The DPLL is disabled automatically when entering EM2, EM3, EM4H or EM4S. • Before enabling DPLL, the HFCLK must temporarily be switched to the HFRCODIV2 source until the DPLL is locked to avoid exceeding the maximum operating frequency due to overshoot. • Before enabling DPLL, flash wait states must be increased if warranted by the selected target frequency • It is also necessary to set the FINETUNINGEN bit in the CMU_HFRCOCTRL register to link the DPLL with the HFRCO. • The HFRCO serves as the local oscillator for DPLL; it is also enabled/disabled when DPLL is enabled/disabled.

3. Energy Modes

3.1 Active Oscillators

The energy mode of the device determines which oscillators are active. In EM0 and EM1, all oscillators can be enabled and used as clock sources. Upon entering EM2, the high frequency oscillators (HFXO, HFRCO, AUXHFRCO, and USHFRCO, if present) shut off automatically, such that the high frequency peripherals running from them are also effectively shutdown. In EM3, the low frequency oscillators (LFXO and LFRCO, but not the PLFRCO, if present) also stop, disabling the low frequency peripherals. The ULFRCO is active in all energy modes down to EM4H and is only disabled in EM4S unless explicitly retained via the RETAINULFRCO bit in the EMU_EM4CTRL register.

The following table summarizes oscillator availability in each energy mode:

Table 3.1. Energy Mode Oscillator Availability

Oscillator	Energy Mode					
	EM0	EM1	EM2	EM3	EM4H	EM4S
HFXO	Available	Available	On demand[1]	On demand[1]	—	—
HFRCO	Available	Available	—	—	—	—
AUXHFRCO	Available	Available	On demand[2]	On demand[2]	—	—
USHFRCO	Available	Available	—	—	—	—
DPLL	Available	Available	—	—	—	—
LFXO	Available	Available	Available	—	Available[3]	Available[3]
LFRCO	Available	Available	Available	—	Available[4]	Available[4]
PLFRCO	Available	Available	Available	Available	—	—
ULFRCO	On	On	On	On	On	Available[5]

Note:
1. If needed by the PLFRCO for calibration.
2. In response to an asynchronous ADC trigger from the PRS.
3. If retained by the RETAINLFXO bit in EMU_EM4CTRL.
4. If retained by the RETAINLFRCO bit in EMU_EM4CTRL.
5. If retained by the RETAINULFRCO bit in EMU_EM4CTRL.

3.2 Wake Up Considerations

3.2.1 Waking up from Low-Energy Modes

All the oscillators are able to run in EM1, so the core wakes up instantly from this energy mode. In EM2 and EM3 the high frequency oscillators are disabled, and these need to be re-enabled before the core starts running code.

When waking up from EM2 or EM3, the core will run from the HFRCO by default, regardless of which oscillator it was running from before entering these energy modes. The HFRCO has a very short wake-up time, and it takes only few microseconds before the CPU starts running code. The previously-selected HFRCO band is also restored by hardware on wake-up. The core will run the HFRCO at the default frequency band when waking up from EM4H or EM4S.

Note: Refer to the device-specific data sheet for the wake-up times from each energy mode.

3.2.2 Restoring the Oscillator

To use an oscillator other than the HFRCO after wake-up from a low-energy mode, the user must either enable and select the oscillator manually or use the energy mode entry functions in emlib, like `EMU_EnterEM2()`, which can handle this automatically.

If an oscillator (the HFXO, for instance) is manually selected as a clock source before it is stable, the system may not operate correctly. The `EMU_EnterEM2()` and `EMU_EnterEM3()`functions in emlib have a boolean parameter that permits any previously enabled oscillators to be restored upon wake-up (see figure below) .

When the user chooses not to restore previously enabled oscillators upon low-energy mode wake-up, any oscillator, other than the HFRCO, that is needed must be manually enabled and selected as a clock source. When the `EMU_EnterEM2()` or `EMU_EnterEM3()`boolean restore parameter is true, emlib will re-enable oscillators, wait for the previously selected HFCLK source to stabilize, and re-select it as the HFCLK at the expense of additional processing time upon wake-up. If the HFRCO was disabled prior to low-energy mode entry (e.g. the HFCLK was sourced from the HFXO), emlib will also disable it after restoring the previously selected HFCLK source.

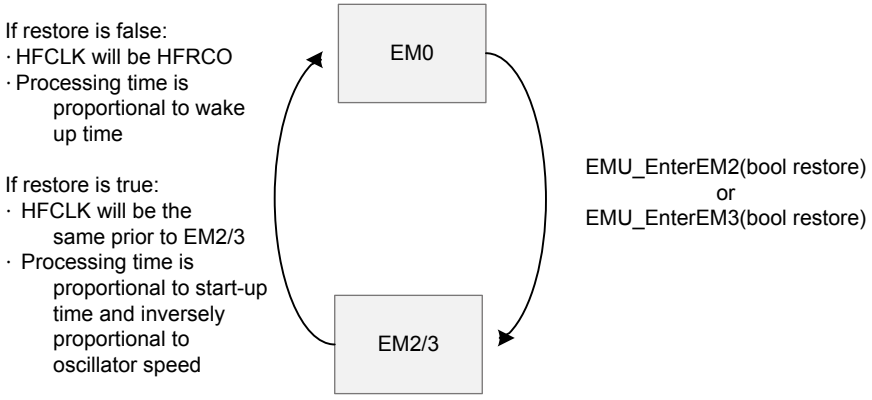


Figure 3.1. Oscillator restoring

The LFXO and HFXO have programmable start-up delays that are set in software by bit fields in each oscillator's respective control registers (see Table 2.9 Oscillator Time-Out Configuraton on page 15). These bit fields specify the number of cycles that must elapse before the oscillator's ready flag is set and can be adjusted to match the characteristics of a particular crystal (faster or slower start-up). A longer time-out (e.g. 16K cycles) guarantees oscillator stability but also extends the period after wake-up spent polling the oscillator's ready flag. For a 32 MHz crystal, a time-out of 16K translates into an approximate delay of 500 µs between the HFXO starting up and its ready flag being set. A lower time-out results in the ready flag setting sooner, but with the risk that the oscillator might not yet be stable.

Each oscillator's ready flag can also be enabled as an interrupt request, the benefit of which is that the processor can continue running code after enabling an oscillator without the need for polling. This is particularly useful when the HFXO provides the HFCLK. Instead of using the `EMU_EnterEM2()` or `EMU_EnterEM3()` restore capability, the HFXO ready interrupt can be enabled prior to entering EM2 or EM3. Upon wake-up, software can manually restart the HFXO and continue running code with the HFCLK still sourced from the HFRCO, only switching to the HFXO when the ready interrupt is eventually requested.

4. RC Oscillator Calibration

The CMU has built-in hardware support to efficiently calibrate the RC oscillators at run-time by comparing the RC oscillator frequency with a selected reference clock. When the calibration circuit is started, one down-counter running from a selectable clock (DOWNSEL bit field in the CMU_CALCTRL register) and one up-counter running from another selectable clock (UPSEL bit field in the CMU_CALCTRL register) are started simultaneously.

When the down-counter has reached 0, the up-counter is sampled and the CALRDY interrupt flag is set. If the CONT bit in the CMU_CALCTRL register is cleared, the counters are stopped after finishing the ongoing calibration. Continuous mode is selected by setting the CONT bit, such that the down-counter reloads its top value and continues counting, and the up-counter restarts from 0. Software can then read the sampled up-counter value from the CMU_CALCNT register.

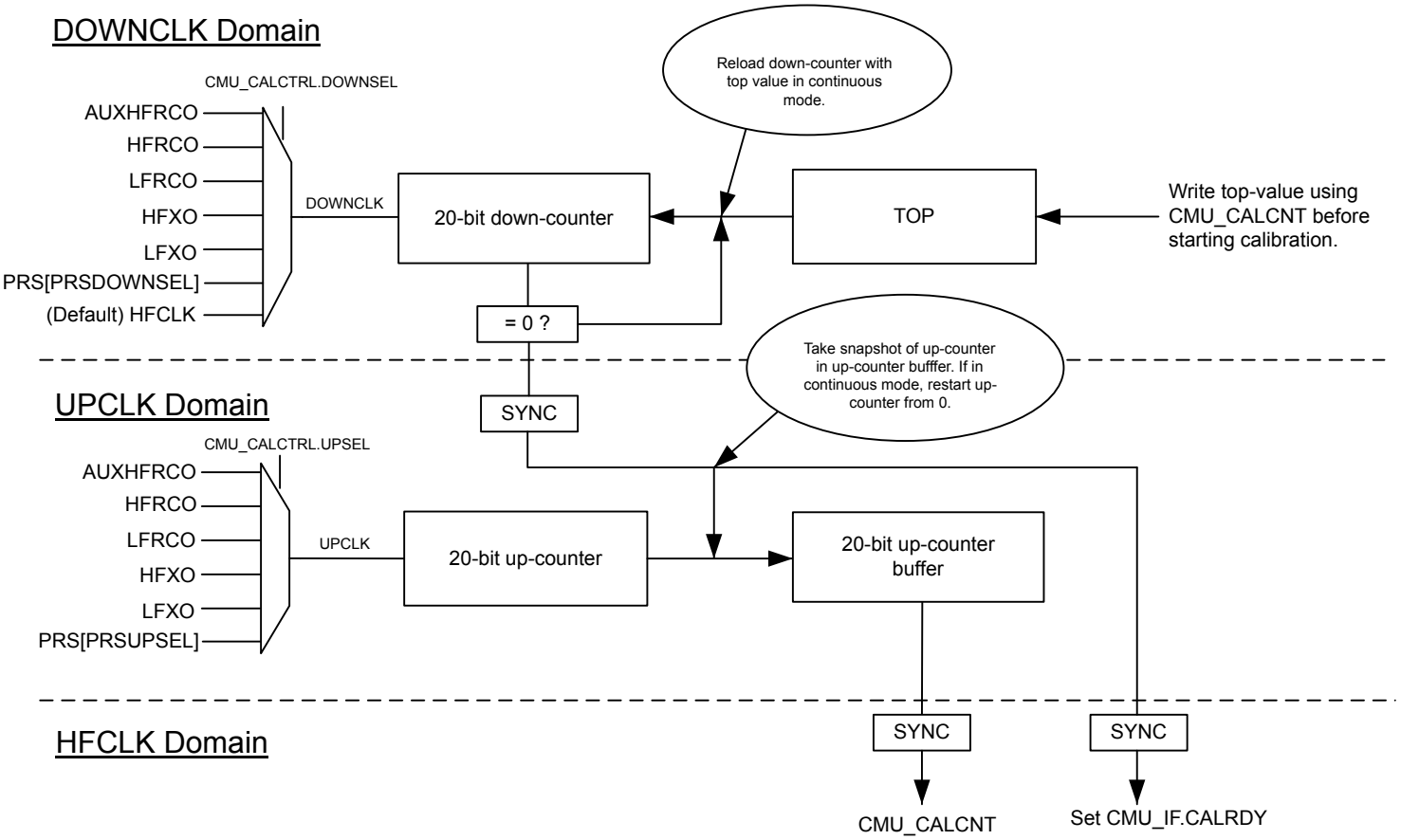


Figure 4.1. Hardware Support for RC Oscillator Calibration

The initial down-counter value (TOP) is written to the CMU_CALCNT register before starting the calibration. When the calibration finishes, the the up-counter is also read from CMU_CALCNT. For the selected up- and down-counter clocks, the following formula is used to calculate the desired up counter value ($UPCOUNT_{DESIRED}$):

$$UPCOUNT_{DESIRED} = \frac{(TOP + 1) \times UPCLK_{FREQ}}{DOWNCLK_{FREQ}}$$

As noted above, TOP is the value from which the down-counter starts; $UPCLK_{FREQ}$ and $DOWNCLK_{FREQ}$ are the frequencies of the oscillators selected to provide the clocks for the up- and down-counters. The maximum value for TOP is `0xFFFFF`, and TOP is directly proportional to calibration accuracy and time such that a higher value for TOP results in better calibration accuracy at the expense of more calibration time.

It is possible to select the RC oscillator for calibration as the source of the clock for the up- or down-counter, but the $UPCLK_{FREQ}$ should be less than $DOWNCLK_{FREQ}$ to make sure $UPCOUNT_{DESIRED}$ is less than the maximum 20-bit up counter value of `0xFFFFF`.

By adjusting the TUNING bit field in the corresponding RC oscillator's control register (e.g. CMU_HFRCOCTRL), the RC oscillator can be tuned to the desired frequency when the up counter value is close to or matches $UPCOUNT_{DESIRED}$.

Table 4.1. The CMU_CALCTRL Register

Bit Field	Usage
DOWNSEL	Down-counter clock source: • HFCLK (Default) • HFXO • LFXO • HFRCO • LFRCO • AUXHFRCO • PRS (PRSDOWNSEL) • USHFRCO[xref1] • PLFRCO[2]
UPSEL	Up-counter clock source: • HFXO • LFXO • HFRCO • LFRCO • AUXHFRCO • PRS (PRSUPSEL) • USHFRCO[xref1] • PLFRCO[2]
CONT	Enables continuous calibration.
PRSDOWNSEL	PRS channel for down-counter clock.
PRSUPSEL	PRS channel for up-counter clock.
Note: 1. Giant Gecko Series 1 only 2. EFM32xG13 and EFR32xG13 only	

An additional FINETUNING bit field is used to tune the HFRCO and AUXHFRCO frequencies in smaller increments than the TUNING bit field. By default, fine tuning is disabled and the FINETUNING value has no effect. Fine tuning can be enabled by writing 1 to FINETUNINGEN bitfield in the CMU_HFRCOCTRL and CMU_AUXHFRCOCTRL registers.

5. Software Examples

5.1 RC Oscillator Calibration

This example calibrates the RC oscillators against the crystal oscillators (see the table below) and displays the TUNING and FINETUNING values before and after calibration on the LCD.

Table 5.1. Clock Selection for RC Oscillator Calibration

RC Oscillator for Calibration	UPCLK	DOWNCLK
LFRCO	LFRCO	HFXO
HFRCO	LFXO	HFRCO
AUXHFRCO	LFXO	AUXHFRCO

The `calibrateRcOsc(CMU_Osc_TypeDef rcOsc, uint32_t rcOscFreq, bool calCont, bool fineEnable)` function in the `cmu_calibrate.c` source file shows how to use the emlib functions for RC oscillator calibration.

Table 5.2. Parameters for calibrateRcOsc( ) Function

Parameter	Use
`rcOsc`	RC oscillator for calibration.
`rcOscFreq`	• Frequency in Hz of RC oscillator to be calibrated. • Calibration may fail if this value is far from the nominal frequency of the RC oscillator.
`calCont`	True to enable continuous mode.
`fineEnable`	True to enable fine tuning.

The current TUNING value of the RC oscillator is loaded from the Device Information (DI) page or the TUNING bit field in the corresponding RC oscillator control register. The default down-counter initial value (TOP) is `0xFFFFF` for maximum calibration accuracy.

The calibration process is interrupt driven (`CMU_IRQHandler()` in `cmu_calibrate.c`), and the TUNING value can either be decremented or incremented in each iteration until the expected value for the up-counter ($UPCOUNT_{DESIRED}$) is reached.

Then the two last counter values are compared to see which one is closer to the one resulting from the formula, and the TUNING value is adjusted accordingly. The final saved TUNING value is displayed on the LCD, and the `endOfTune` flag is set to indicate that calibration is finished.

The HFRCO and AUXHFRCO frequencies can be tuned more accurately via the FINETUNING and FINETUNINGEN bit fields in the CMU_HFRCOCTRL and CMU_AUXHFRCOCTRL registers.

When fine tuning is used, the HFRCO and AUXHFRCO frequencies will drift away from their nominal frequencies when the FINETUNINGEN bits in the CMU_HFRCOCTRL and CMU_AUXHFRCOCTRL registers are set to 1 for the first time. This happens because the FINETUNING bit field values are not calibrated during production testing, and a default value of `0x1F` is used.

Table 5.3. RC Oscillator Tuning Ranges

RC Oscillator	Tuning Range
LFRCO	0 – 255
HFRCO	0 – 127
HFRCO FINETUNING	0 – 63
AUXHFRCO	0 – 127
AUXHFRCO FINETUNING	0 – 63

RC Oscillator	Tuning Range
Note: 1. Larger values written to the TUNING or FINETUNING bit fields translate into lower frequencies.	

If an oscilloscope is available, the RC oscillator output can be observed on one of the Starter Kit (STK) Expansion Header pins, which is configured as a CMU clock output pin during the calibration process. The following table lists the pin to probe on a given STK.

Table 5.4. RC Oscillator Probe Output

Device	Starter Kit	Expansion Header Pin
EFM32PG1	SLSTK3401A	16
EFR32MG1	SLWSTK6000A	
EFM32PG12	SLSTK3402A	
EFR32MG12	SLWSTK6000B	
EFM32GG11	SLSTK3701A	8

This example project uses the naming convention `BOARD_cmu_example`, where `BOARD` is one of the STK part numbers from the table above. A menu screen shows the RC oscillator to be calibrated; BTN0 runs the calibration, and BTN1 cycles through the different RC oscillators.

5.1.1 Calibrate LFRCO with HFXO

When the example code is run to calibrate the LFRCO against the HFXO, the output should look something like the screens shown below.

Press push button BTN1 to select [**Example 1: HFXO to Calibrate LFRCO**].

```
Example 1:
HFXO to Calibrate
LFRCO

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to start the LFRCO calibration process. The current LFRCO TUNING value and target up counter are displayed on the memory LCD as shown below.

```
Example 1:
HFXO to Calibrate
LFRCO

Old tune value: 167
UpCnt Target: 858
```

The final up counter and LFRCO TUNING value are displayed on the memory LCD when LFRCO calibration is complete.

```
Example 1:
HFXO to Calibrate
LFRCO

Old tune value: 167
UpCnt Target: 858

Tune count: 1
UpCnt Actual: 860
New tune value: 165

Press BTN0 to start
Press BTN1 to exit
```

5.1.2 Calibrate HFRCO with LFXO

When the example code is run to calibrate the HFRCO against the LFXO, the output should look something like the screens shown below.

Press push button BTN1 to select the [**Example 2: LFXO to Calibrate HFRCO**].

```
Example 2:
LFXO to Calibrate
HFRCO

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to start the HFRCO calibration process. The current HFRCO TUNING value, FINETUNE value (if enabled), and target up counter are displayed on the memory LCD as shown below.

```
Example 2:
LFXO to Calibrate
HFRCO

Old tune value: 62
Old finetune val: 31
UpCnt Target: 1808
```

The final up counter, HFRCO TUNING value, and FINETUNE value (if enabled) are displayed on the memory LCD when HFRCO calibration is complete.

```
Example 2:
LFXO to Calibrate
HFRCO

Old tune value: 62
Old finetune val: 31
UpCnt Target: 1808

Tune count: 1
UpCnt Actual: 1807
New tune value: 52
New finetune val: 35

Press BTN0 to start
Press BTN1 to exit
```

5.1.3 Calibrate AUXHFRCO with LFXO

When the example code is run to calibrate the AUHFRCO against the LFXO, the output should look something like the screens shown below.

Press push button BTN1 to select the [**Example 3: LFXO to Calibrate AUXHFRCO**].

```
Example 3:
LFXO to Calibrate
AUXHFRCO

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to start the AUXHFRCO calibration process. The current AUXHFRCO TUNING value, FINETUNE value (if enabled), and target up counter are displayed on the memory LCD as shown below.

```
Example 3:
LFXO to Calibrate
AUXHFRCO

Old tune value: 61
Old finetune val: 31
UpCnt Target: 1808
```

The final up counter, AUXHFRCO TUNING value, and FINETUNE value (if enabled) are displayed on the memory LCD when AUXHFRCO calibration is complete.

```
Example 3:
LFXO to Calibrate
AUXHFRCO

Old tune value: 61
Old finetune val: 31
UpCnt Target: 1808

Tune count: 1
UpCnt Actual: 1808
New tune value: 52
New finetune val: 33

Press BTN0 to start
Press BTN1 to exit
```

5.2 HFXO Start-Up Time

This example illustrates, in particular, when the HFXO is used, the time it takes to wake from EM2 and begin code execution depending upon the selected clock source and start-up mechanism. Via the PRS, the RTCC generates pulses at one second intervals that are routed to a pin. This pin, which can be observed on an oscilloscope, must be connected using a jumper wire to another pin, which, in turn, wakes the system via GPIO interrupts.

When configured to do so, enabling of the HFXO and its selection as the HFSRCCLK source is performed automatically by hardware upon wake-up from EM2. HFXO start-up time can be made deterministic and is programmed via the LFTIMEOUT bit field in the CMU_HFXOCTRL register to be 16 LFECLK cycles (~ 488 µs) in this example.

As noted above, GPIOs can be observed to trace the automatic HFXO start sequence. The pins used and their purpose are shown in the following tables and differ depending on the STK. To measure wake-up timing from EM2, exit the debugger and return to the IDE. With the STK still connected via the mini USB connector, switch the power selector on the STK to the BAT position and then back to the AEM position to reset the system and observe the different GPIO strobes and their timing.

Table 5.5. Pin Use for HFXO Start-Up Demo

Device	Starter Kit	RTCC PRS Output		GPIO IRQ Handler Entry		CMU IRQ Handler Entry		GPIO Wake-Up Input		HFSRCCLK Output	
		GPIO	Expansion Header Pin	GPIO	Expansion Header Pin	GPIO	Expansion Header Pin	GPIO	Expansion Header Pin	GPIO	Expansion Header Pin
EFM32PG1	SLSTK3401A	PA0	12	PA1	14	PA2	3	PA3	5	PC10	16
EFM32PG12	SLSTK3402A										
EFR32MG1	WSTK6000A										
EFR32MG12	WSTK6000B										
EGM32GG11	STK3701A	PE10	4	PE11	6	PE12	8	PE13	10	PA12	3

Just as with the previous example, this project uses the naming convention `BOARD_cmu_hfxo_autostart`, where `BOARD` is the STK part number for the device being used.

5.2.1 Normal HFRCO Wake-Up

Set the `USE_AUTO_HFXO` define in `main_cmu_hfxo_autostart.c` to 0 to disable the automatic HFXO start feature (`USE_LFTIMEOUT` is not used here).

In this example, the device will run from the HFRCO when waking up from EM2, and the previously configured HFRCO band (19 MHz in this example) is also restored by hardware on wake-up. The HFRCO has a short wake-up time, and it takes about 10 µs (see the figure below) before the CPU starts running code.

```
Normal HFRCO Start
Example

Connects PA3 to PA0
to wakeup device from
EM2 every second

EM2 wakeup count:  1
```

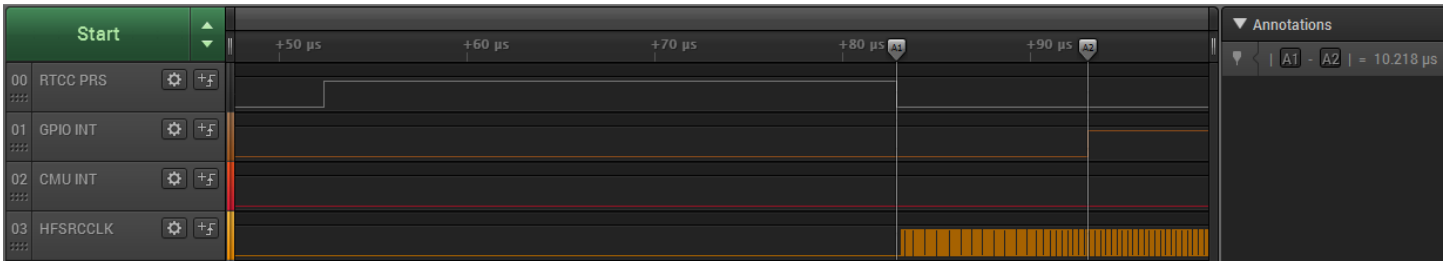


Figure 5.1. Normal HFRCO Start

5.2.2 Automatic HFXO Start with Normal Start-Up

Set the `USE_LFTIMEOUT` define in `main_cmu_hfxo_autostart.c` to 0 to disable the deterministic HFXO start-up feature (`USE_AUTO_HFXO` is set to 1).

In this example, the HFXO is automatically started and immediately selected upon waking from EM2 into EM0. The HFSRCCLK is stalled until HFXO becomes ready (see HFSRCCLK output in figure below), which can lead to a relatively long start-up time as the system will run from the HFRCO as it would otherwise do. The normal start-up time varies from crystal to crystal, so the timing of every HFXO start-up (~ 419 µs in this example) differs even with the CMU_HFXOTIMEOUTCTRL register set to the same value.

```
Automatic HFXO Start
with Normal Start-up
Time Example

Connects PA3 to PA0
to wakeup device from
EM2 every second

EM2 wakeup count:  1
```

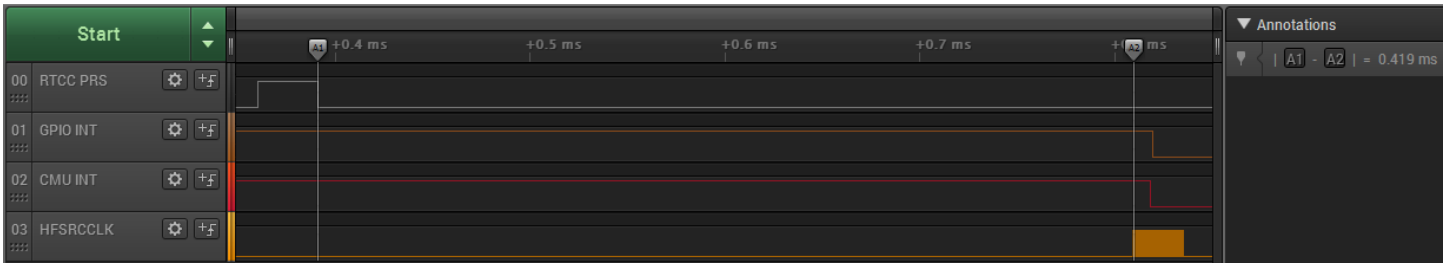


Figure 5.2. Automatic HFXO Start with Normal Start-up Time

5.2.3 Automatic HFXO Start with Deterministic Start-Up

Set the `USE_LFTIMEOUT` define in `main_cmu_hfxo_autostart.c` to 1 to enable the deterministic HFXO start-up feature (`USE_AUTO_HFXO` is set to 1).

In this example, HFXO ready assertion is delayed by 16 LFECLK cycles as programmed into the CMU_HFXO_CTRL LFTIMEOUT to achieve deterministic start-up time (~ 488 µs on every HFXO start-up). The time base typically used for the normal start-up is not as accurate as the time base accuracy that can be achieved for the LFTIMEOUT counter, specifically if the timing is based on the LFXO. For a detailed timing diagram of CMU Automatic startup and selection of HFXO, please refer to the device reference manual.

```
Automatic HFXO Start
with Deterministic
Start-up Time
Example

Connects PA3 to PA0
to wakeup device from
EM2 every second

EM2 wakeup count:  1
```

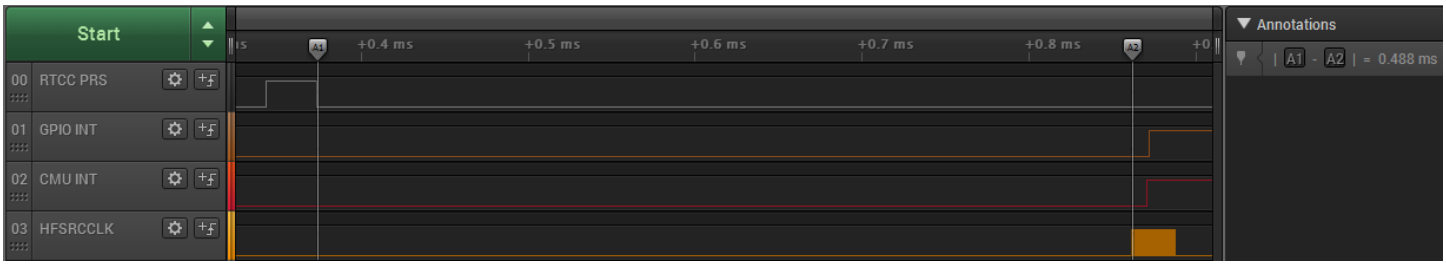


Figure 5.3. Automatic HFXO Start with Deterministic Start-up Time

5.3 Digital Phased-Locked Loop (DPLL)

This software example is run on the EFM32 Pearl Gecko PG12 Starter Kit (project `cmu_lfxo_dpll_pg` or `SLSTK3402A_cmu_lfxo_dpll`) with source file `main_cmu_lfxo_dpll.c`.

The Memory LCD display requires that the pin EXTCOMIN is toggled at a constant frequency to avoid charge buildup. The current Memory LCD display driver uses the RTC/RTCC to trigger a PRS output to toggle the EXTCOMIN pin at the required frequency.

The EFM32 Pearl Gecko PG12 device has a clock output on PRS function, which can achieve this output waveform automatically in EM2 or EM3. The ULFRCO is selected as a PRS producer of CMUCLKOUT0 (`PAL_GpioPinAutoToggle()` function in `main_cmu_lfxo_dpll.c`) to toggle the EXTCOMIN pin (PD13), but the pin toggle frequency is fixed at 1 kHz which is much higher than the nominal 64 Hz.

The `bool setupDpll(CMU_Osc_TypeDef xtalOsc, uint16_t factorN, uint16_t factorM)` function in the `main_cmu_lfxo_dpll.c` source file is used to set up the DPLL.

Table 5.6. Parameters for setupDpll( ) Function

Parameter	Usage
`xtalOsc`	The crystal oscillator for DPLL reference clock (see Table 2.20 Digital Phase-Locked Loop Operation on page 23).
`factorN`	The factor N to setup the locked DCO frequency (see Table 2.20 Digital Phase-Locked Loop Operation on page 23).
`factorM`	The factor M to setup the locked DCO frequency (see Table 2.20 Digital Phase-Locked Loop Operation on page 23).

This example uses the LFXO as the DPLL reference clock to generate the locked DCO frequency of 24.576 MHz (32768 Hz x 750) from the 26 MHz HFRCO frequency band. Based on the formula in Table 2.20 Digital Phase-Locked Loop Operation on page 23, factor M is 1 for a 32768 Hz F_{ref} if factor N is set to 1499.

The 24.576 MHz HFRCO can be used to generate the bit clock at 3.072 MHz (= 24.576 MHz/8) for a stereo 48 kHz 32-bit I2S data stream. This example configures USART3 as the I2S master to send 48 kHz 32-bit dummy data `0x55AA55AA` continuously (see Figure 5.4 USART3 Stereo 48 kHz 32-bit Data I2S Waveform on page 36).

Table 5.7. GPIO Usage on USART3 I2S Master

GPIO	Usage
PD8 (USART3 #29)	I2S word clock (EXP header pin 13)
PB8 (USART3 #10)	I2S bit clock (EXP header pin 11)
PB6 (USART3 #10)	I2S TX (EXP header pin 7)

```
DPLL Example
HFRCO is tuned to
24.576 MHz with LFXO

DPLL is locked

Transmit 48kHz 32-bit
I2S data on USART3

I2S_WS on PD8
I2S_CLK on PB8
I2S_TX on PB6
```

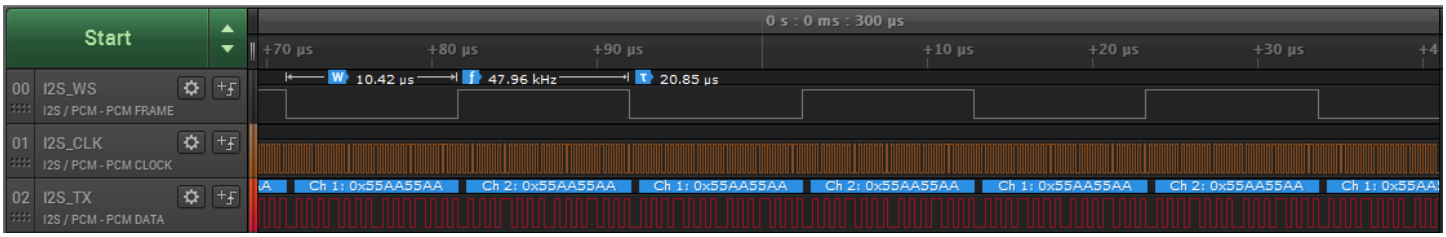


Figure 5.4. USART3 Stereo 48 kHz 32-bit Data I2S Waveform

If a scope is available, the HFSRCCLK (= HFRCO in this example) can be probed on PD10 (EXP header pin 12). Set the `CMU_OUT_EN` define in `main_cmu_lfxo_dpll.c` to 1 (default is 0) to enable this feature.

6. Revision History

6.1 Revision 1.10

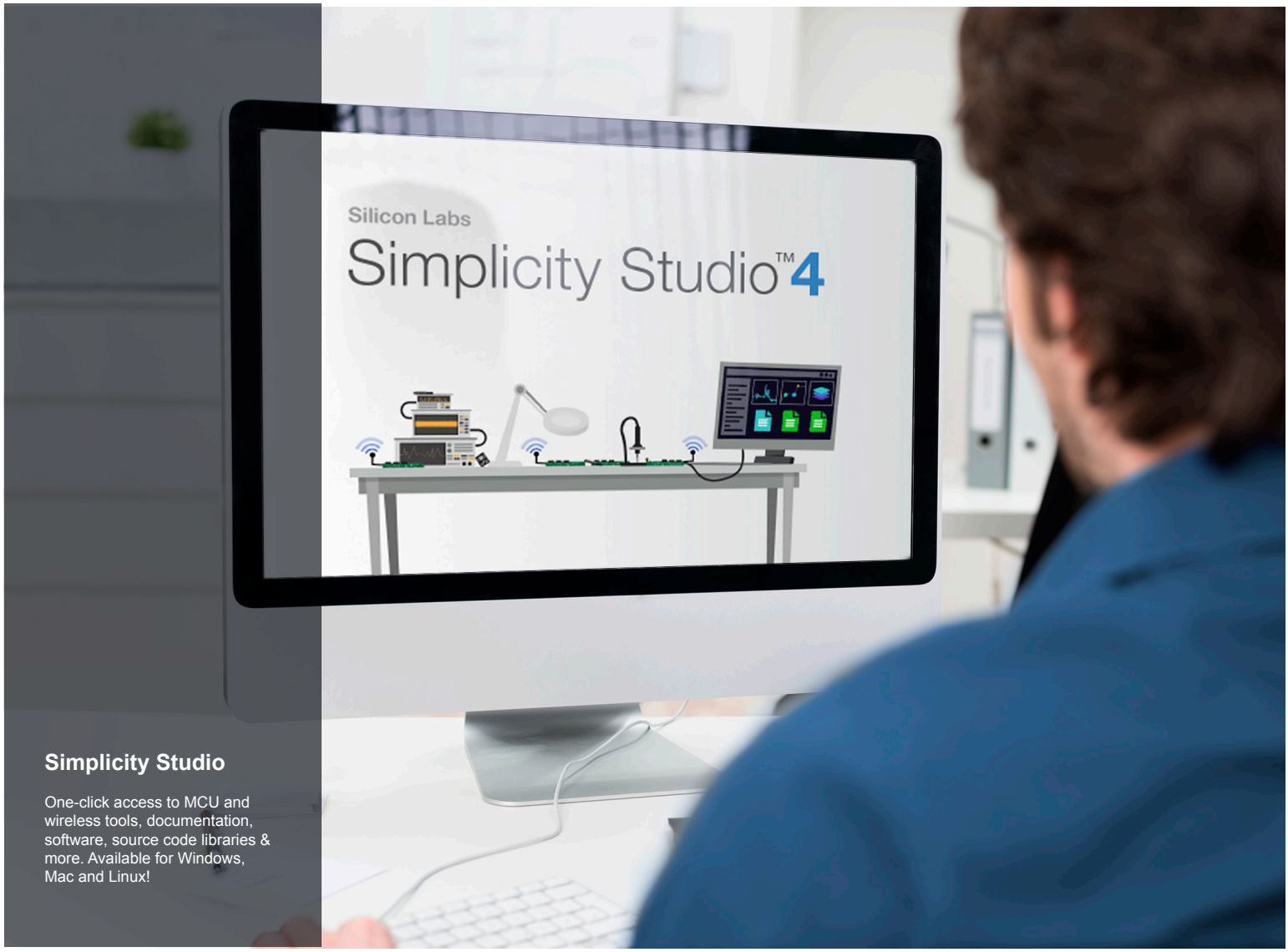
2017-6-29

Rewrite and addition of EFM32xG13/EFR32xG13 and EFM32 Giant Gecko Series 1.

6.2 Revision 1.00

2017-1-26

Initial revision.



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com