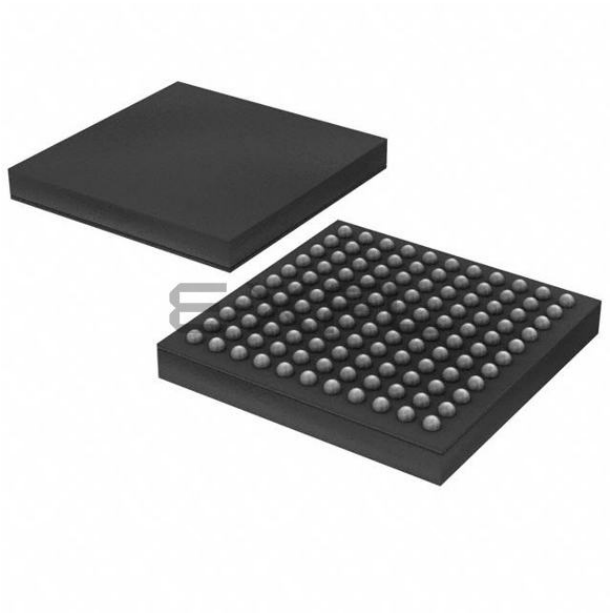
Silicon Labs - EFM32GG995F1024G-E-BGA120 Datasheet



Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	ARM® Cortex®-M3
Core Size	32-Bit Single-Core
Speed	48MHz
Connectivity	EBI/EMI, I²C, IrDA, SmartCard, SPI, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, LCD, POR, PWM, WDT
Number of I/O	93
Program Memory Size	1MB (1M x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	128K x 8
Voltage - Supply (Vcc/Vdd)	1.98V ~ 3.8V
Data Converters	A/D 8x12b; D/A 2x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	120-VFBGA
Supplier Device Package	120-BGA (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/silicon-labs/efm32gg995f1024g-e-bga120

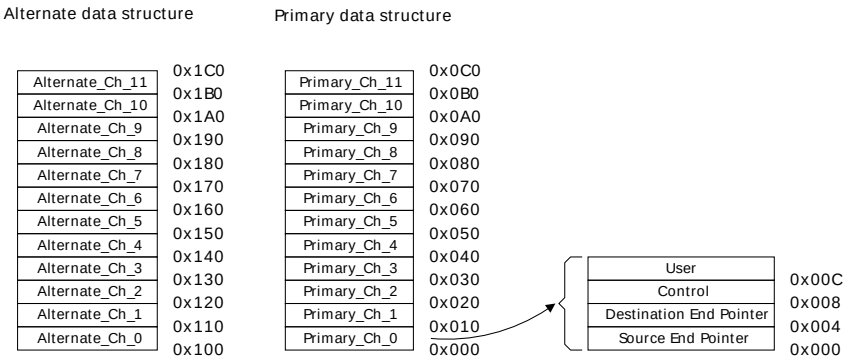
Channel number	Priority level setting	Descending order of channel priority
1	High	-
2	High	-
3	High	-
4	High	-
5	High	-
6	High	-
7	High	-
8	High	-
9	High	-
10	High	-
11	High	-
0	Default	-
1	Default	-
2	Default	-
3	Default	-
4	Default	-
5	Default	-
6	Default	-
7	Default	-
8	Default	-
9	Default	-
10	Default	-
11	Default	Lowest-priority DMA channel

After a DMA transfer completes, the controller polls all the DMA channels that are available. Figure 8.2 (p. 53) shows the process it uses to determine which DMA transfer to perform next.

- have a base address that is an integer multiple of the total size of the channel control data structure.

Figure 8.6 (p. 62) shows the memory that the controller requires for the channel control data structure, when all 12 channels and the optional alternate data structure are in use.

Figure 8.6. Memory map for 12 channels, including the alternate data structure



This structure in Figure 8.6 (p. 62) uses 384 bytes of system memory. The controller uses the lower 8 address bits to enable it to access all of the elements in the structure and therefore the base address must be at `0xXXXXXX00`.

You can configure the base address for the primary data structure by writing the appropriate value in the DMA_CTRLBASE register.

You do not need to set aside the full 384 bytes if all dma channels are not used or if all alternate descriptors are not used. If, for example, only 4 channels are used and they only need the primary descriptors, then only 64 bytes need to be set aside.

Table 8.6 (p. 62) lists the address bits that the controller uses when it accesses the elements of the channel control data structure.

Table 8.6. Address bit settings for the channel control data structure

Address bits					
[8]	[7]	[6]	[5]	[4]	[3:0]
A	C[3]	C[2]	C[1]	C[0]	0x0, 0x4, or 0x8

Where:

A — Selects one of the channel control data structures:
- A = 0 Selects the primary data structure.
- A = 1 Selects the alternate data structure.

C[3:0] — Selects the DMA channel.

Address[3:0] — Selects one of the control elements:
- `0x0` Selects the source data end pointer.
- `0x4` Selects the destination data end pointer.
- `0x8` Selects the control data configuration.
- `0xC` The controller does not access this address location. If required, you can enable the host processor to use this memory location as system memory.

Note

It is not necessary for you to calculate the base address of the alternate data structure because the DMA_ALTCTRLBASE register provides this information.

8.7.9 DMA_CHREQMASKS - Channel Request Mask Set Register

Offset	Bit Position																													
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																					0	0	0	0	0	0	0	0	0	0	0	0
Access																					RW1	RW1	RW1	RW1	RW1	RW1	RW1	RW1	RW1	RW1	RW1	RW1
Name																					CH11REQMASKS	CH10REQMASKS	CH9REQMASKS	CH8REQMASKS	CH7REQMASKS	CH6REQMASKS	CH5REQMASKS	CH4REQMASKS	CH3REQMASKS	CH2REQMASKS	CH1REQMASKS	CH0REQMASKS

Bit	Name	Reset	Access	Description
31:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11	CH11REQMASKS	0	RW1	**Channel 11 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
10	CH10REQMASKS	0	RW1	**Channel 10 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
9	CH9REQMASKS	0	RW1	**Channel 9 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
8	CH8REQMASKS	0	RW1	**Channel 8 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
7	CH7REQMASKS	0	RW1	**Channel 7 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
6	CH6REQMASKS	0	RW1	**Channel 6 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
5	CH5REQMASKS	0	RW1	**Channel 5 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
4	CH4REQMASKS	0	RW1	**Channel 4 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
3	CH3REQMASKS	0	RW1	**Channel 3 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
2	CH2REQMASKS	0	RW1	**Channel 2 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
1	CH1REQMASKS	0	RW1	**Channel 1 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
0	CH0REQMASKS	0	RW1	**Channel 0 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			

12.3.1 Clock Source

Three clock sources are available for use with the watchdog, through the CLKSEL field in WDOG_CTRL. The corresponding clocks must be enabled in the CMU. The SWOSCBLOCK bit in WDOG_CTRL can be written to prevent accidental disabling of the selected clocks. Also, setting this bit will automatically start the selected oscillator source when the watchdog is enabled. The PERSEL field in WDOG_CTRL is used to divide the selected watchdog clock, and the timeout for the watchdog timer can be calculated like this:

WDOG Timeout Equation

$$T_{TIMEOUT} = (2^{3+PERSEL} + 1)/f, \quad (12.1)$$

where f is the frequency of the selected clock.

It is recommended to clear the watchdog first, if PERSEL is changed while the watchdog is enabled.

To use this module, the LE interface clock must be enabled in CMU_HFCORECLKEN0, in addition to the module clock.

Note

Before changing the clock source for WDOG, the EN bit in WDOG_CTRL should be cleared. In addition to this, the WDOG_SYNCBUSY value should be zero.

12.3.2 Debug Functionality

The watchdog timer can either keep running or be frozen when the device is halted by a debugger. This configuration is done through the DEBUGRUN bit in WDOG_CTRL. When code execution is resumed, the watchdog will continue counting where it left off.

12.3.3 Energy Mode Handling

The watchdog timer can be configured to either keep on running or freeze when entering EM2 or EM3. The configuration is done individually for each energy mode in the EM2RUN and EM3RUN bits in WDOG_CTRL. When the watchdog has been frozen and is re-entering an energy mode where it is running, the watchdog timer will continue counting where it left off. For the watchdog there is no difference between EM0 and EM1. The watchdog does not run in EM4, and if EM4BLOCK in WDOG_CTRL is set, the CPU is prevented from entering EM4.

Note

If the WDOG is clocked by the LFXO or LFRCO, writing the SWOSCBLOCK bit will effectively prevent the CPU from entering EM3. When running from the ULFRCO, writing the SWOSCBLOCK bit will prevent the CPU from entering EM4.

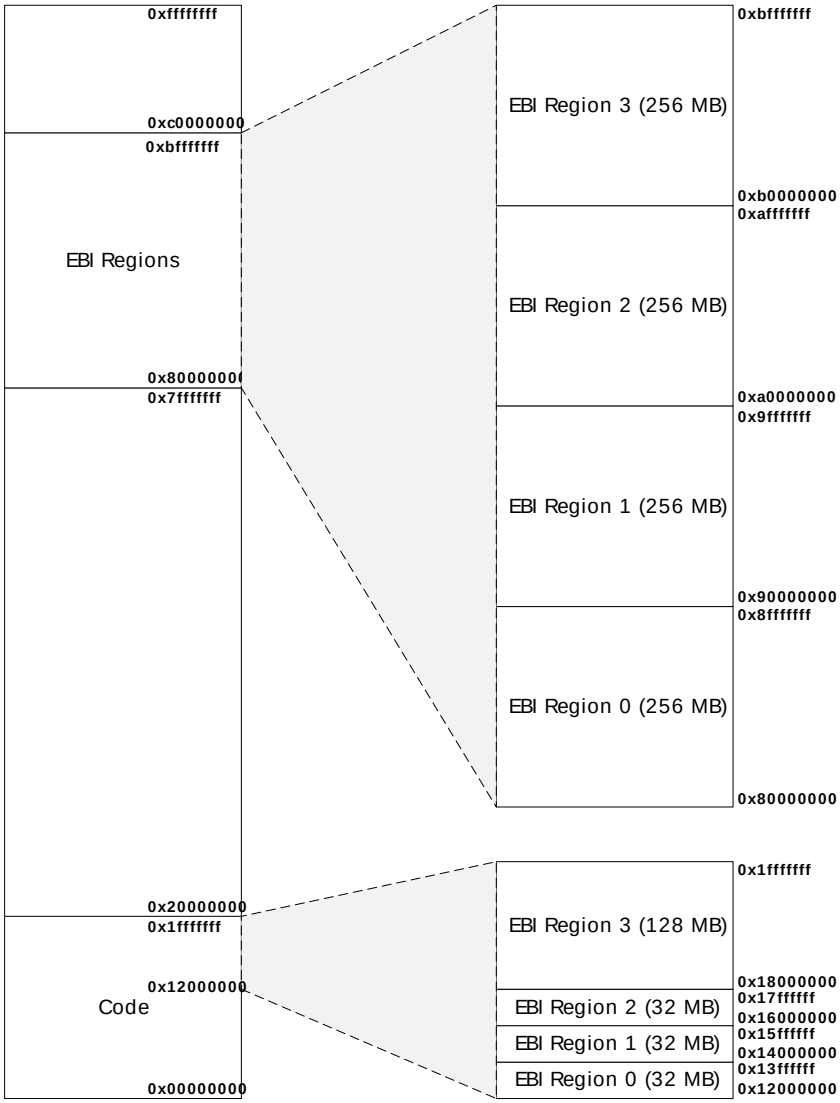
12.3.4 Register access

Since this module is a Low Energy Peripheral, and runs off a clock which is asynchronous to the HFCORECLK, special considerations must be taken when accessing registers. Please refer to Section 5.3 (p. 20) for a description on how to perform register accesses to Low Energy Peripherals. note that clearing the EN bit in WDOG_CTRL will reset the WDOG module, which will halt any ongoing register synchronization.

Note

Never write to the WDOG registers when it is disabled, except to enable it by setting WDOG_CTRL_EN or when changing the clock source using WDOG_CTRL_CLKSEL. Make sure that the enable is registered (i.e. WDOG_SYNCBUSY_CTRL goes low), before writing other registers.

Figure 14.24. EBI Alternative Memory Map (ALTMAP = 1)



14.3.13 WAIT/ARDY.

Some external devices are able to indicate that they are not finished with either write or read operation by asserting the WAIT / ARDY line. This input signal is used to extend the REn/WEn cycles for slow devices. The interpretation of the polarity of this signal can be configured with the ARDYPOL bit in EBI_POLARITY. E.g. if the ARDYPOL is set to ACTIVELOW, then the REn/WEn cycle is extended while the ARDY line is kept low. The ARDY functionality is enabled by setting the ARDYEN bit in the EBI_CTRL register. It is also possible to enable a timeout check, which generates a bus error if the ARDY is not deasserted within the timeout period. This prevents a system lock up condition in the case that the external device does not deassert ARDY. The timeout functionality is disabled by setting ARDYTODIS in the EBI_CTRL register.

When the ITS bitfield in the EBI_CTRL register is set to 0, the wait behavior defined in the ARDYEN and ARDYTODIS bitfields applies to all 4 memory banks. When ITS is set to 1 each memory bank uses an individual wait behavior definition. In this case bitfields ARDYEN and ARDYTODIS only apply to bank 0. Wait behavior for bank n is then defined in the ARDYnEN and ARDYTOnDIS bitfields.

6. If the application sets or clears a STALL for an endpoint due to a SetFeature.Endpoint Halt command or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control endpoint.

Special Case: Stalling the Control IN/OUT Endpoint

The core must stall IN/OUT tokens if, during the Data stage of a control transfer, the host sends more IN/OUT tokens than are specified in the SETUP packet. In this case, the application must to enable USB_DIEPx_INT.INTKNTXFEMP and USB_DOEPx_INT.OUTTKNEPDIS interrupts during the Data stage of the control transfer, after the core has transferred the amount of data specified in the SETUP packet. Then, when the application receives this interrupt, it must set the STALL bit in the corresponding endpoint control register, and clear this interrupt.

15.4.4.2.3.8 Worst-Case Response Time

When the acts as a device, there is a worst case response time for any tokens that follow an isochronous OUT. This worst case response time depends on the AHB clock frequency.

The core registers are in the AHB domain, and the core does not accept another token before updating these register values. The worst case is for any token following an isochronous OUT, because for an isochronous transaction, there is no handshake and the next token could come sooner. This worst case value is 7 PHY clocks in FS mode.

If this worst case condition occurs, the core responds to bulk/interrupt tokens with a NAK and drops isochronous and SETUP tokens. The host interprets this as a timeout condition for SETUP and retries the SETUP packet. For isochronous transfers, the INCOMPISOIN and INCOMPLP interrupts inform the application that isochronous IN/OUT packets were dropped.

15.4.4.2.3.9 Choosing the Value of USB_GUSBCFG.USBTRDTIM

The value in USB_GUSBCFG.USBTRDTIM is the time it takes for the MAC, in terms of PHY clocks after it has received an IN token, to get the FIFO status, and thus the first data from PFC (Packet FIFO Controller) block. This time involves the synchronization delay between the PHY and AHB clocks. This delay is 5 clocks.

Once the MAC receives an IN token, this information (token received) is synchronized to the AHB clock by the PFC (the PFC runs on the AHB clock). The PFC then reads the data from the SPRAM and writes it into the dual clock source buffer. The MAC then reads the data out of the source buffer (4 deep).

If the AHB is running at a higher frequency than the PHY (in Low-speed mode), the application can use a smaller value for USB_GUSBCFG.USBTRDTIM. Figure 15.26 (p. 312) explains the 5-clock delay. This diagram has the following signals:

- tkn_rcvd: Token received information from MAC to PFC
- dynced_tkn_rcvd: Doubled sync tkn_rcvd, from pclk to hclk domain
- spr_read: Read to SPRAM
- spr_addr: Address to SPRAM
- spr_rdata: Read data from SPRAM
- srcbuf_push: Push to the source buffer
- srcbuf_rdata: Read data from the source buffer. Data seen by MAC

The application can use the following formula to calculate the value of USB_GUSBCFG.USBTRDTIM:

4 * AHB Clock + 1 PHY Clock = (2 clock sync + 1 clock memory address + 1 clock memory data from sync RAM) + (1 PHY Clock (next PHY clock MAC can sample the 2-clock FIFO output)

Bit	Name	Reset	Access	Description
1	CHHLTD	0	RW1H	**Channel Halted**
	In DMA mode this bit indicates the transfer completed abnormally either because of any USB transaction error or in response to disable request by the application or because of a completed transfer.			
0	XFERCOMPL	0	RW1H	**Transfer Completed**
	Transfer completed normally without any errors. This bit can be set only by the core and the application should write 1 to clear it.			

15.6.37 USB_HCx_INTMSK - Host Channel x Interrupt Mask Register

This register reflects the mask for each channel status described in the USB_CHx_INT.

Offset	Bit Position																											
0x3C50C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0	0		0	0	0	0	0	0
Access																						RW	RW	RW	RW		RW	RW	RW	RW	RW	RW
Name																						DATATGLERRMSK	FRMOVRUNMSK	BBLERRMSK	XACTERRMSK		ACKMSK	NAKMSK	STALLMSK	AHBERRMSK	CHHLTDMSK	XFERCOMPLMSK

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	DATATGLERRMSK	0	RW	**Data Toggle Error Mask**
	Set to unmask DATATGLERR interrupt.			
9	FRMOVRUNMSK	0	RW	**Frame Overrun Mask**
	Set to unmask FRMOVRUN interrupt.			
8	BBLERRMSK	0	RW	**Babble Error Mask**
	Set to unmask BBLERR interrupt.			
7	XACTERRMSK	0	RW	**Transaction Error Mask**
	Set to unmask XACTERR interrupt.			
6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	ACKMSK	0	RW	**ACK Response Received/Transmitted Interrupt Mask**
	Set to unmask ACK interrupt.			
4	NAKMSK	0	RW	**NAK Response Received Interrupt Mask**
	Set to unmask NAK interrupt.			
3	STALLMSK	0	RW	**STALL Response Received Interrupt Mask**
	Set to unmask STALL interrupt.			
2	AHBERRMSK	0	RW	**AHB Error Mask**
	Set to unmask AHBERR interrupt.			
1	CHHLTDMSK	0	RW	**Channel Halted Mask**
	Set to unmask CHHLTD interrupt.			
0	XFERCOMPLMSK	0	RW	**Transfer Completed Mask**
	Set to unmask XFERCOMPL interrupt.			

I2Cn_STAT	Description	I2Cn_IF	Required interaction	Response
			START	Repeated start condition will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
-	Data transmitted	TXBL interrupt flag (TXC interrupt flag)	None	
0xD7	Data transmitted,ACK received	ACK interrupt flag (BUSHOLD interrupt flag)	TXDATA	DATA will be sent
			STOP	STOP will be sent. Bus will be released
			START	Repeated start condition will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
0xDF	Data transmitted,NACK received	NACK(BUSHOLD interrupt flag)	CONT + TXDATA	DATA will be sent
			STOP	STOP will be sent. Bus will be released
			START	Repeated start condition will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
-	Stop transmitted	MSTOP interrupt flag	None	
			START	START will be sent when bus becomes idle
-	Arbitration lost	ARBLOST interrupt flag	None	
			START	START will be sent when bus becomes idle

16.3.7.5 Master Receiver

To receive data from a slave, the master must operate as a master receiver, see Table 16.6 (p. 428) . This is done by transmitting ADDR+R as the address byte instead of ADDR+W, which is transmitted to become a master transmitter. The address byte loaded into the data register thus has to contain the 7-bit slave address in the 7 most significant bits of the byte, and have the least significant bit set.

When the address has been transmitted, the master receives an ACK or a NACK. If an ACK is received, the ACK interrupt flag in I2Cn_IF is set, and if space is available in the receive shift register, reception of a byte from the slave begins. If the receive buffer and shift register is full however, the bus is held until data is read from the receive buffer or another interaction is made. Note that the STOP and START interactions have a higher priority than the data-available interaction, so if a STOP or START command is pending, the highest priority interaction will be performed, and data will not be received from the slave.

If a NACK was received, the CONT command in I2Cn_CMD has to be issued in order to continue receiving data, even if there is space available in the receive buffer and/or shift register.

After a data byte has been received the master must ACK or NACK the received byte. If an ACK is pending or AUTOACK in I2Cn_CTRL is set, an ACK is sent automatically and reception continues if space is available in the receive buffer.

If a NACK is sent, the CONT command must be used in order to continue transmission. If an ACK or NACK is issued along with a START or STOP or both, then the ACK/NACK is transmitted and the reception is ended. If START in I2Cn_CMD is set alone, a repeated start condition is transmitted after the ACK/NACK. If STOP in I2Cn_CMD is set, a stop condition is sent regardless of whether START is set. If START is set in this case, it is set as pending.

- Tristate transmitter after transmission: If TXTRIAT is set, TXTRI is set after the frame has been fully transmitted, tristating the transmitter output. Tristating of the output can also be performed automatically by setting AUTOTRI. If AUTOTRI is set TXTRI is always read as 0.

Note

When in SmartCard mode with repeat enabled, none of the actions, except generate break, will be performed until the frame is transmitted without failure. Generation of a break in SmartCard mode with repeat enabled will cause the USART to detect a NACK on every frame.

17.3.2.4 Data Reception

Data reception is enabled by setting RXEN in USARTn_CMD. When the receiver is enabled, it actively samples the input looking for a transition from high to low indicating the start baud of a new frame. When a start baud is found, reception of the new frame begins if the receive shift register is empty and ready for new data. When the frame has been received, it is pushed into the receive buffer, making the shift register ready for another frame of data, and the receiver starts looking for another start baud. If the receive buffer is full, the received frame remains in the shift register until more space in the receive buffer is available. If an incoming frame is detected while both the receive buffer and the receive shift register are full, the data in the shift register is overwritten, and the RXOF interrupt flag in USARTn_IF is set to indicate the buffer overflow.

The receiver can be disabled by setting the command bit RXDIS in USARTn_CMD. Any frame currently being received when the receiver is disabled is discarded. Whether or not the receiver is enabled at a given time can be read out from RXENS in USARTn_STATUS.

17.3.2.4.1 Receive Buffer Operation

When data becomes available in the receive buffer, the RXDATAV flag in USARTn_STATUS, and the RXDATAV interrupt flag in USARTn_IF are set, and when the buffer becomes full, RXFULL in USARTn_STATUS and the RXFULL interrupt flag in USARTn_IF are set. The status flags RXDATAV and RXFULL are automatically cleared by hardware when their condition is no longer true. This also goes for the RXDATAV interrupt flag, but the RXFULL interrupt flag must be cleared by software. When the RXFULL flag is set, notifying that the buffer is full, space is still available in the receive shift register for one more frame.

Data can be read from the receive buffer in a number of ways. USARTn_RXDATA gives access to the 8 least significant bits of the received frame, and USARTn_RXDOUBLE makes it possible to read the 8 least significant bits of two frames at once, pulling two frames from the buffer. To get access to the 9th, most significant bit, USARTn_RXDATAX must be used. This register also contains status information regarding the frame. USARTn_RXDOUBLEX can be used to get two frames complete with the 9th bits and status bits.

When a frame is read from the receive buffer using USARTn_RXDATA or USARTn_RXDATAX, the frame is pulled out of the buffer, making room for a new frame. USARTn_RXDOUBLE and USARTn_RXDOUBLEX pull two frames out of the buffer. If an attempt is done to read more frames from the buffer than what is available, the RXUF interrupt flag in USARTn_IF is set to signal the underflow, and the data read from the buffer is undefined.

Frames can be read from the receive buffer without removing the data by using USARTn_RXDATAXP and USARTn_RXDOUBLEXP. USARTn_RXDATAXP gives access the first frame in the buffer with status bits, while USARTn_RXDOUBLEXP gives access to both frames with status bits. The data read from these registers when the receive buffer is empty is undefined. If the receive buffer contains one valid frame, the first frame in USARTn_RXDOUBLEXP will be valid. No underflow interrupt is generated by a read using these registers, i.e. RXUF in USARTn_IF is never set as a result of reading from USARTn_RXDATAXP or USARTn_RXDOUBLEXP.

When AUTOTX in USARTn_CTRL is set, the USART transmits data as long as there is available space in the RX shift register for the chosen frame size. This happens even though there is no data in the TX buffer. The TX underflow interrupt flag TXUF in USARTn_IF is set on the first word that is transmitted which does not contain valid data.

During AUTOTX the USART will always send the previous sent bit, thus reducing the number of transitions on the TX output. So if the last bit sent was a 0, 0's will be sent during AUTOTX and if the last bit sent was a 1, 1's will be sent during AUTOTX.

17.3.3.4 Slave Mode

When the USART is in slave mode, data transmission is not controlled by the USART, but by an external master. The USART is therefore not able to initiate a transmission, and has no control over the number of bytes written to the master.

The output and input to the USART are also swapped when in slave mode, making the receiver take its input from USn_TX (MOSI) and the transmitter drive USn_RX (MISO).

To transmit data when in slave mode, the slave must load data into the transmit buffer and enable the transmitter. The data will remain in the USART until the master starts a transmission by pulling the USn_CS input of the slave low and transmitting data. For every frame the master transmits to the slave, a frame is transferred from the slave to the master. After a transmission, MISO remains in the same state as the last bit transmitted. This also applies if the master transmits to the slave and the slave TX buffer is empty.

If the transmitter is enabled in synchronous slave mode and the master starts transmission of a frame, the underflow interrupt flag TXUF in USARTn_IF will be set if no data is available for transmission to the master.

If the slave needs to control its own chip select signal, this can be achieved by clearing CSPEN in the ROUTE register. The internal chip select signal can then be controlled through CSINV in the CTRL register. The chip select signal will be CSINV inverted, i.e. if CSINV is cleared, the chip select is active and vice versa.

17.3.3.5 Synchronous Half Duplex Communication

Half duplex communication in synchronous mode is very similar to half duplex communication in asynchronous mode as detailed in Section 17.3.2.6 (p. 460). The main difference is that in this mode, the master must generate the bus clock even when it is not transmitting data, i.e. it must provide the slave with a clock to receive data. To generate the bus clock, the master should transmit data with the transmitter tristated, i.e. TXTRI in USARTn_STATUS set, when receiving data. If 2 bytes are expected from the slave, then transmit 2 bytes with the transmitter tristated, and the slave uses the generated bus clock to transmit data to the master. TXTRI can be set by setting the TXTRIEN command bit in USARTn_CMD.

Note

When operating as SPI slave in half duplex mode, TX has to be tristated (not disabled) during data reception if the slave is to transmit data in the current transfer.

17.3.3.6 I2S

I2S is a synchronous format for transmission of audio data. The frame format is 32-bit, but since data is always transmitted with MSB first, an I2S device operating with 16-bit audio may choose to only process the 16 msb of the frame, and only transmit data in the 16 msb of the frame.

In addition to the bit clock used for regular synchronous transfers, I2S mode uses a separate word clock. When operating in mono mode, with only one channel of data, the word clock pulses once at the start of each new word. In stereo mode, the word clock toggles at the start of new words, and also gives away

Bit	Name	Reset	Access	Description

Value	Mode	Description
1	X8	Double speed with 8X oversampling in asynchronous mode
2	X6	6X oversampling in asynchronous mode
3	X4	Quadruple speed with 4X oversampling in asynchronous mode

4 MPAB 0 RW **Multi-Processor Address-Bit**

Defines the value of the multi-processor address bit. An incoming frame with its 9th bit equal to the value of this bit marks the frame as a multi-processor address frame.

3 MPM 0 RW **Multi-Processor Mode**

Multi-processor mode uses the 9th bit of the USART frames to tell whether the frame is an address frame or a data frame.

Value	Description
0	The 9th bit of incoming frames has no special function
1	An incoming frame with the 9th bit equal to MPAB will be loaded into the receive buffer regardless of RXBLOCK and will result in the MPAB interrupt flag being set

2 CCEN 0 RW **Collision Check Enable**

Enables collision checking on data when operating in half duplex modus.

Value	Description
0	Collision check is disabled
1	Collision check is enabled. The receiver must be enabled for the check to be performed

1 LOOPBK 0 RW **Loopback Enable**

Allows the receiver to be connected directly to the USART transmitter for loopback and half duplex communication.

Value	Description
0	The receiver is connected to and receives data from U(S)n_RX
1	The receiver is connected to and receives data from U(S)n_TX

0 SYNC 0 RW **USART Synchronous Mode**

Determines whether the USART is operating in asynchronous or synchronous mode.

Value	Description
0	The USART operates in asynchronous mode
1	The USART operates in synchronous mode

17.5.2 USARTn_FRAME - USART Frame Format Register

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																			0x1				0x0						0x5			
Access																			RW				RW						RW			
Name																			STOPBITS				PARITY						DATABITS			

Bit	Name	Reset	Access	Description
31:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

13:12 STOPBITS 0x1 RW **Stop-Bit Mode**

Determines the number of stop-bits used.

Value	Mode	Description
0	HALF	The transmitter generates a half stop bit. Stop-bits are not verified by receiver
1	ONE	One stop bit is generated and verified
2	ONEANDAHALF	The transmitter generates one and a half stop bit. The receiver verifies the first stop bit

When 8 data-bit frame formats are used, only the 8 least significant bits of LEUARTn_STARTFRAME are compared to incoming frames. The full length of LEUARTn_STARTFRAME is used when operating with frames consisting of 9 data bits.

Note

The receiver must be enabled for start frames to be detected. In addition, a start frame with a parity error or framing error is not detected as a start frame.

19.3.5.7 Programmable Signal Frame

As well as the configurable start frame, a special signal frame can be specified. When a frame matching the frame defined in LEUARTn_SIGFRAME is detected by the receiver, the SIGF interrupt flag in LEUARTn_IF is set. As for start frame detection, the receiver must be enabled for signal frames to be detected.

One use of the programmable signal frame is to signal the end of a multi-frame message transmitted to the LEUART. An interrupt will then be triggered when the packet has been completely received, allowing software to process it. Used in conjunction with the programmable start frame and DMA, this makes it possible for the LEUART to automatically begin the reception of a packet on a specified start frame, load the entire packet into memory, and give an interrupt when reception of a packet has completed. The device can thus wait for data packets in EM2, and only be woken up when a packet has been completely received.

A signal frame with a parity error or framing error is not detected as a signal frame.

19.3.5.8 Multi-Processor Mode

To simplify communication between multiple processors and maintain compatibility with the USART, the LEUART supports a multi-processor mode. In this mode the 9th data bit in each frame is used to indicate whether the content of the remaining 8 bits is data or an address.

When multi-processor mode is enabled, an incoming 9-bit frame with the 9th bit equal to the value of MPAB in LEUARTn_CTRL is identified as an address frame. When an address frame is detected, the MPAF interrupt flag in LEUARTn_IF is set, and the address frame is loaded into the receive register. This happens regardless of the value of RXBLOCK in LEUARTn_STATUS.

Multi-processor mode is enabled by setting MPM in LEUARTn_CTRL. The mode can be used in buses with multiple slaves, allowing the slaves to be addressed using the special address frames. An addressed slave, which was previously blocking reception using RXBLOCK, would then unblock reception, receive a message from the bus master, and then block reception again, waiting for the next message. See the USART for a more detailed example.

Note

The programmable start frame functionality can be used for automatic address matching, enabling reception on a correctly configured incoming frame.

An address frame with a parity error or a framing error is not detected as an address frame.

19.3.6 Loopback

The LEUART receiver samples LEUn_RX by default, and the transmitter drives LEUn_TX by default. This is not the only configuration however. When LOOPBK in LEUARTn_CTRL is set, the receiver is connected to the LEUn_TX pin as shown in Figure 19.5 (p. 506) . This is useful for debugging, as the LEUART can receive the data it transmits, but it is also used to allow the LEUART to read and write to the same pin, which is required for some half duplex communication modes. In this mode, the LEUn_TX pin must be enabled as an output in the GPIO.

Bit	Name	Reset	Access	Description

Value	Description
1	A compare match on RTC compare channel 1 starts the LETIMER if the LETIMER is not already started

10 RTCC0TEN 0 RW **RTC Compare 0 Trigger Enable**

Allows the LETIMER to be started on a compare match on RTC compare channel 0.

Value	Description
0	LETIMER is not affected by RTC compare channel 0
1	A compare match on RTC compare channel 0 starts the LETIMER if the LETIMER is not already started

9 COMP0TOP 0 RW **Compare Value 0 Is Top Value**

When set, the counter is cleared in the clock cycle after a compare match with compare channel 0.

Value	Description
0	The top value of the LETIMER is 65535 (0xFFFF)
1	The top value of the LETIMER is given by COMP0

8 BUFTOP 0 RW **Buffered Top**

Set to load COMP1 into COMP0 when REP0 reaches 0, allowing a buffered top value

Value	Description
0	COMP0 is only written by software
1	COMP0 is set to COMP1 when REP0 reaches 0

7 OPOL1 0 RW **Output 1 Polarity**

Defines the idle value of output 1.

6 OPOL0 0 RW **Output 0 Polarity**

Defines the idle value of output 0.

5:4 UFOA1 0x0 RW **Underflow Output Action 1**

Defines the action on LETn_O1 on a LETIMER underflow.

Value	Mode	Description
0	NONE	LETn_O1 is held at its idle value as defined by OPOL1.
1	TOGGLE	LETn_O1 is toggled on CNT underflow.
2	PULSE	LETn_O1 is held active for one $LFACLK_{LETIMER0}$ clock cycle on CNT underflow. The output then returns to its idle value as defined by OPOL1.
3	PWM	LETn_O1 is set idle on CNT underflow, and active on compare match with COMP1

3:2 UFOA0 0x0 RW **Underflow Output Action 0**

Defines the action on LETn_O0 on a LETIMER underflow.

Value	Mode	Description
0	NONE	LETn_O0 is held at its idle value as defined by OPOL0.
1	TOGGLE	LETn_O0 is toggled on CNT underflow.
2	PULSE	LETn_O0 is held active for one $LFACLK_{LETIMER0}$ clock cycle on CNT underflow. The output then returns to its idle value as defined by OPOL0.
3	PWM	LETn_O0 is set idle on CNT underflow, and active on compare match with COMP1

1:0 REPMODE 0x0 RW **Repeat Mode**

Allows the repeat counter to be enabled and disabled.

Value	Mode	Description
0	FREE	When started, the LETIMER counts down until it is stopped by software.
1	ONESHOT	The counter counts REP0 times. When REP0 reaches zero, the counter stops.
2	BUFFERED	The counter counts REP0 times. If REP1 has been written, it is loaded into REP0 when REP0 reaches zero. Else the counter stops
3	DOUBLE	Both REP0 and REP1 are decremented when the LETIMER wraps around. The LETIMER counts until both REP0 and REP1 are zero

23.5.4 LETIMERn_CNT - Counter Value Register

Offset	Bit Position																															
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RWH															
Name																	CNT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	CNT	0x0000	RWH	**Counter Value**
	Use to read the current value of the LETIMER.			

23.5.5 LETIMERn_COMP0 - Compare Value Register 0 (Async Reg)

For more information about Asynchronous Registers please see Section 5.3 (p. 20) .

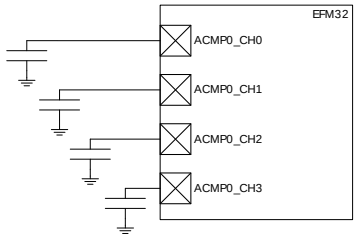
Offset	Bit Position																															
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	COMP0															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	COMP0	0x0000	RW	**Compare Value 0**
	Compare and optionally top value for LETIMER			

23.5.6 LETIMERn_COMP1 - Compare Value Register 1 (Async Reg)

For more information about Asynchronous Registers please see Section 5.3 (p. 20) .

Figure 25.10. Capacitive sense setup



The following steps show how to configure LESENSE to scan through the four buttons 100 times per second, issuing an interrupt if one of them is pressed.

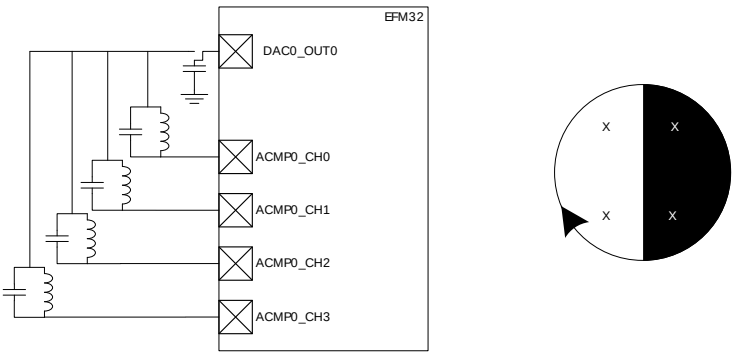
1. Assuming $LFACLK_{LESENSE}$ is 32kHz, set PCPRESC to 3 and PCTOP to 39 in CTRL. This will make the LESENSE scan frequency 100Hz.
2. Enable channels 0 through 3 in CHEN and set IDLECONF for these channels to DISABLED. In capacitive sense mode, the GPIO should always be disabled (analog input).
3. Configure the ACMP to operate in CAPSENSE mode, refer to Section 26.3.5 (p. 672) for details.
4. Configure the following bit fields in CHx_CONF, for channels 0 through 3:
   a. Set EXTIME to 0. No excitation is needed in this mode.
   b. Set SAMPLE to COUNTER and COMP to LESS. This makes LESENSE interpret a sensor as active if the frequency on a channel drops below the threshold, i.e. the button is pressed.
   c. Set SAMPLEDLY to an appropriate value, each sensor will be measured for SAMPLEDLY/ $LFACLK_{LESENSE}$ seconds. MEASUREDLY should be set to 0
5. Set CTRTHRESHOLD to an appropriate value. An interrupt will be issued if the counter value for a sensor is below this threshold after the measurement phase.
6. Enable interrupts on channels 0 through 3.
7. Start scan sequence by writing a 1 to START in CMD.

In a capacitive sense application, it might be required to calibrate the threshold values on a periodic basis, this is done in order to compensate for humidity and other physical variations. LESENSE is able to store up to 16 counter values from a configurable number of channels, making it possible to collect sample data while in EM2. When calibration is to be performed, the CPU only has to be woken up for a short period of time as the data to be processed already lies in the result registers. To enable storing of the count value for a channel, set STRSAMPLE in the CHx_INTERACT register.

25.3.14.2 LC sensor

Figure 25.11 (p. 635) below illustrates how the EFM32GG can be set up to monitor four LC sensors.

Figure 25.11. LC sensor setup



LESENSE can be used to excite and measure the damping factor in LC sensor oscillations. To measure the damping factor, the ACMP can be used to generate a high output each time the sensor voltage

Bit	Name	Reset	Access	Description
15	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
14:12	PRSACT	0xX	RW	**Configure transition action**
	Configure which action to perform when sensor state equals COMP			

DECCTRL_PRSCNT = 0		
Mode	Value	Description
NONE	0	No PRS pulses generated
PRS0	1	Generate pulse on LESPRS0
PRS1	2	Generate pulse on LESPRS1
PRS01	3	Generate pulse on LESPRS0 and LESPRS1
PRS2	4	Generate pulse on LESPRS2
PRS02	5	Generate pulse on LESPRS0 and LESPRS2
PRS12	6	Generate pulse on LESPRS1 and LESPRS2
PRS012	7	Generate pulse on LESPRS0, LESPRS1 and LESPRS2
DECCTRL_PRSCNT = 1		
NONE	0	Do not count
UP	1	Count up
DOWN	2	Count down
PRS2	4	Generate pulse on LESPRS2
UPANDPRS2	5	Count up and generate pulse on LESPRS2.
DOWNANDPRS2	6	Count down and generate pulse on LESPRS2.

11:8	NEXTSTATE	0xX	RW	**Next state index**
	Index of next state to be entered if the sensor state equals COMP			
7:4	MASK	0xX	RW	**Sensor mask**
	Set bit X to exclude sensor X from evaluation.			
3:0	COMP	0xX	RW	**Sensor compare value**
	State transition is triggered when sensor state equals COMP			

25.5.25 LESENSE_STx_TCONFB - State transition configuration B (Async Reg)

For more information about Asynchronous Registers please see Section 5.3 (p. 20) .

Offset	Bit Position																													
0x204	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																X		0xX			0xX				0xX				0xX			
Access																RW		RW			RW				RW				RW			
Name																SETIF		PRSACT			NEXTSTATE				MASK				COMP			

Bit	Name	Reset	Access	Description
31:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
16	SETIF	X	RW	**Set interrupt flag**
	Set interrupt flag when sensor state equals COMP			
15	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
14:12	PRSACT	0xX	RW	**Configure transition action**
	Configure which action to perform when sensor state equals COMP			

Bit	Name	Reset	Access	Description
11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	RESINMUX	0x0	RW	**OPA0 Resistor Ladder Input Mux**

These bits selects the source for the input mux to the resistor ladder

Value	Mode	Description
0	DISABLE	Set for Unity Gain
1	OPA0INP	Set for OPA0 input
2	NEGPAD	NEG pad connected
3	POSPAD	POS pad connected
4	VSS	VSS connected

Bit	Name	Reset	Access	Description
7:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5:4	NEGSEL	0x0	RW	**OPA0 inverting Input Mux**

These bits selects the source for the inverting input on OPA0

Value	Mode	Description
0	DISABLE	Input disabled
1	UG	Unity Gain feedback path
2	OPATAP	OPA0 Resistor ladder as input
3	NEGPAD	Input from NEG PAD

Bit	Name	Reset	Access	Description
3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	POSSEL	0x0	RW	**OPA0 non-inverting Input Mux**

These bits selects the source for the non-inverting input on OPA0

Value	Mode	Description
0	DISABLE	Input disabled
1	DAC	DAC as input
2	POSPAD	POS PAD as input
3	OPA0INP	OPA0 as input
4	OPATAP	OPA0 Resistor ladder as input

29.5.17 DACn_OPA1MUX - Operational Amplifier Mux Configuration Register

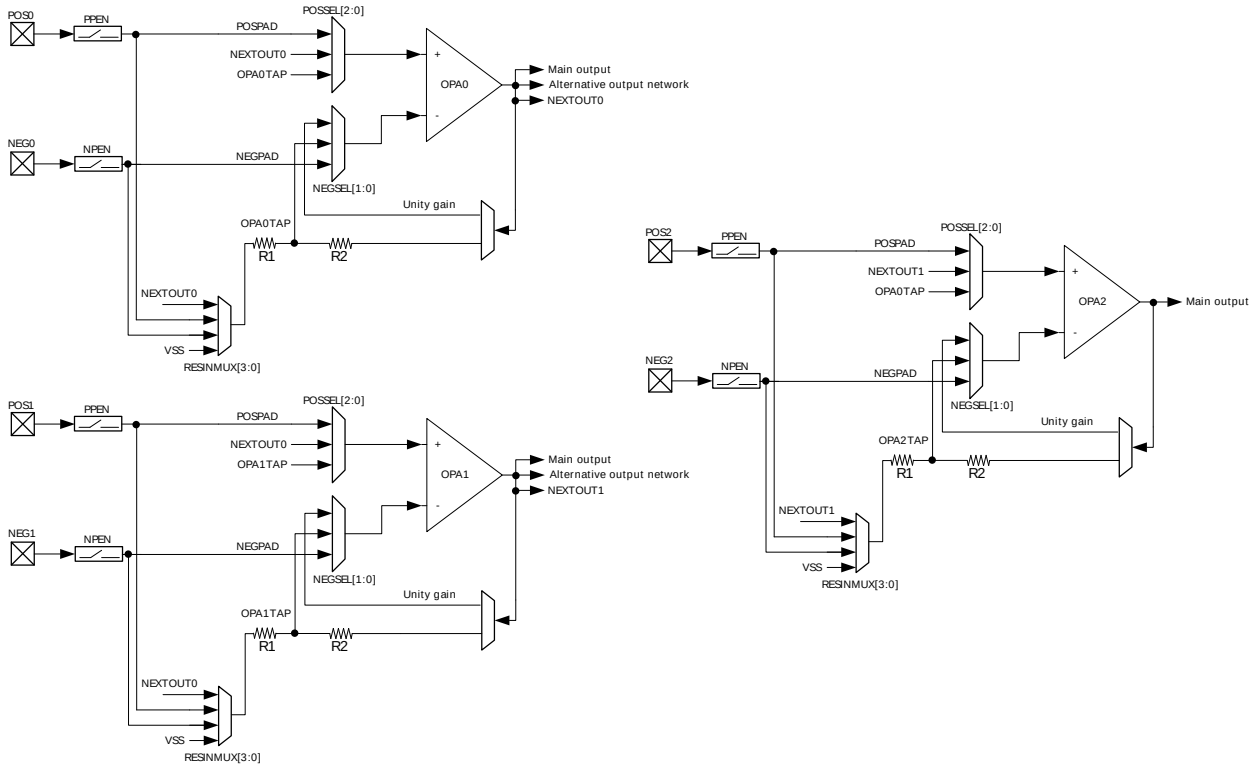
Offset	Bit Position																															
0x060	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x0				0			0x0					0x00					0	0		0x0					0x0			0x0		
Access		RW				RW			RW					RW					RW	RW		RW					RW			RW		
Name		RESSEL				NEXTOUT			OUTMODE					OUTPEN					NPEN	PPEN		RESINMUX					NEGSEL			POSSEL		

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30:28	RESSEL	0x0	RW	**OPA1 Resistor Ladder Select**

Configures the resistor ladder tap for OPA1.

Value	Mode	Resistor Value	Inverting Mode Gain (-R2/R1)	Non-inverting Mode Gain (1+(R2/R1)
0	RES0	R2 = 1/3 x R1	-1/3	1 1/3
1	RES1	R2 = R1	-1	2
2	RES2	R2 = 1 2/3 x R1	-1 2/3	2 2/3
3	RES3	R2 = 2 x R1	-2 1/5	3 1/5

Figure 30.2. OPAMP Overview



30.3.1 Opamp Configuration

Since two of the three opamps (OPA0, OPA1) are part of the DAC, the opamp configuration registers are located in the DAC. The mux registers for OPA0/OPA1 together with OPA2 registers are separate registers, also located under the DAC module. OPA0 and OPA1 can be enabled by setting OPAxEN in DACn_OPACTRL and CHxEN in CHxCTRL. OPA2 can be enabled by only setting OPA2EN in DACn_OPACTRL.

30.3.1.1 Input Configuration

The inputs to the opamps are controlled through a set of input muxes. The mux connected to the positive input is configured by the POSSEL bit-field in the DACn_OPAxMUX register. Similarly, the mux connected to the negative input is configured by setting the NEGSEL bit-field in DACn_OPAxMUX. To connect the pins to the input muxes, the pin switches must also be enabled. Setting the PPEN bit-field enables to POSPADx, while setting the NPEN bit-field enables the NEGPADx, both located in DACn_OPAxMUX. The input into the resistor ladder can be configured by setting the RESINMUX bit-field in DACn_OPAxMUX.

30.3.1.2 Output Configuration

The opamp have two outputs, one main output and one alternative output with lower drive strength. These two outputs can be used to drive the different outputs as shown in Figure 30.3 (p. 736). The main opamp output can be used to drive the main output by setting OUTMODE to MAIN in DACn_OPAxMUX. The alternative opamp output can drive the alternative output network by setting OUTMODE to ALT in DACn_OPAxMUX. In addition, it is also possible to use the main opamp output to drive both the main output and the alternative output network by setting OUTMODE to ALL in DACn_OPAxMUX.

Bit	Name	Reset	Access	Description

Value	Mode	Description
0	PORTA	Port A pin 13 selected for external interrupt 13
1	PORTB	Port B pin 13 selected for external interrupt 13
2	PORTC	Port C pin 13 selected for external interrupt 13
3	PORTD	Port D pin 13 selected for external interrupt 13
4	PORTE	Port E pin 13 selected for external interrupt 13
5	PORTF	Port F pin 13 selected for external interrupt 13

19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18:16	EXTIPSEL12	0x0	RW	**External Interrupt 12 Port Select**

Select input port for external interrupt 12.

Value	Mode	Description
0	PORTA	Port A pin 12 selected for external interrupt 12
1	PORTB	Port B pin 12 selected for external interrupt 12
2	PORTC	Port C pin 12 selected for external interrupt 12
3	PORTD	Port D pin 12 selected for external interrupt 12
4	PORTE	Port E pin 12 selected for external interrupt 12
5	PORTF	Port F pin 12 selected for external interrupt 12

15	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
14:12	EXTIPSEL11	0x0	RW	**External Interrupt 11 Port Select**

Select input port for external interrupt 11.

Value	Mode	Description
0	PORTA	Port A pin 11 selected for external interrupt 11
1	PORTB	Port B pin 11 selected for external interrupt 11
2	PORTC	Port C pin 11 selected for external interrupt 11
3	PORTD	Port D pin 11 selected for external interrupt 11
4	PORTE	Port E pin 11 selected for external interrupt 11
5	PORTF	Port F pin 11 selected for external interrupt 11

11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	EXTIPSEL10	0x0	RW	**External Interrupt 10 Port Select**

Select input port for external interrupt 10.

Value	Mode	Description
0	PORTA	Port A pin 10 selected for external interrupt 10
1	PORTB	Port B pin 10 selected for external interrupt 10
2	PORTC	Port C pin 10 selected for external interrupt 10
3	PORTD	Port D pin 10 selected for external interrupt 10
4	PORTE	Port E pin 10 selected for external interrupt 10
5	PORTF	Port F pin 10 selected for external interrupt 10

7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:4	EXTIPSEL9	0x0	RW	**External Interrupt 9 Port Select**

Select input port for external interrupt 9.

Value	Mode	Description
0	PORTA	Port A pin 9 selected for external interrupt 9
1	PORTB	Port B pin 9 selected for external interrupt 9
2	PORTC	Port C pin 9 selected for external interrupt 9
3	PORTD	Port D pin 9 selected for external interrupt 9
4	PORTE	Port E pin 9 selected for external interrupt 9
5	PORTF	Port F pin 9 selected for external interrupt 9

3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	EXTIPSEL8	0x0	RW	**External Interrupt 8 Port Select**

Example 33.2. LCD Animation Enable Example

- Write data into the animation registers LCD_AREGA, LCD_AREGB
- Enable the correct shift direction (if any)
- Decide which logical function to perform on the registers
  - ALOGSEL = 0: Data_out = LCD_AREGA & LCD_AREGB
  - ALOGSEL = 1:Data_out = LCD_AREGA | LCD_AREGB
- Configure the right animation period (CLK_{EVENT})
- Enable the animation pattern and frame counter (AEN = 1, FCEN = 1)

For updating data in the LCD while it is running an animation, and the new animation data depends on the pattern visible on the LCD, see the following example.

Example 33.3. LCD Animation Dependence Example

- Enable the LCD interrupt (the interrupt will be triggered simultaneously as the Animation State machine changes state)
- In the interrupt handler, read back the current state (ASTATE)
- Knowing the current state of the Animation State Machine makes it possible to calculate what data that is currently output
- Modify data as required (Data will be updated at the next Frame Counter Event). It is important that new data is written before the next Frame Counter Event.

33.3.13 LCD in Low Energy Modes

As long as the LFACLK is running (EM0-EM2), the LCD controller continues to output LCD waveforms according to the data that is currently synchronized to the LCD Driver logic. In addition, the following features are still active if enabled:

- Animation State Machine
- Blink
- LCD Event Interrupt

33.3.14 Register access

Since this module is a Low Energy Peripheral, and runs off a clock which is asynchronous to the HFCORECLK, special considerations must be taken when accessing registers. Please refer to Section 5.3 (p. 20) for a description on how to perform register accesses to Low Energy Peripherals.