EFM32

... the world's most energy friendly microcontrollers

Low Energy Sensor

AN0029 - Application Note

Introduction

This application note covers the basics of inductive sensing (LC) and describes how to use the Low Energy Sensor Interface (LESENSE) to scan a number of LC sensors while remaining in EM2 achieving current consumption below 2µA.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects

ARM ZERO TINY GECKO LEOPARD GIANT WONDER

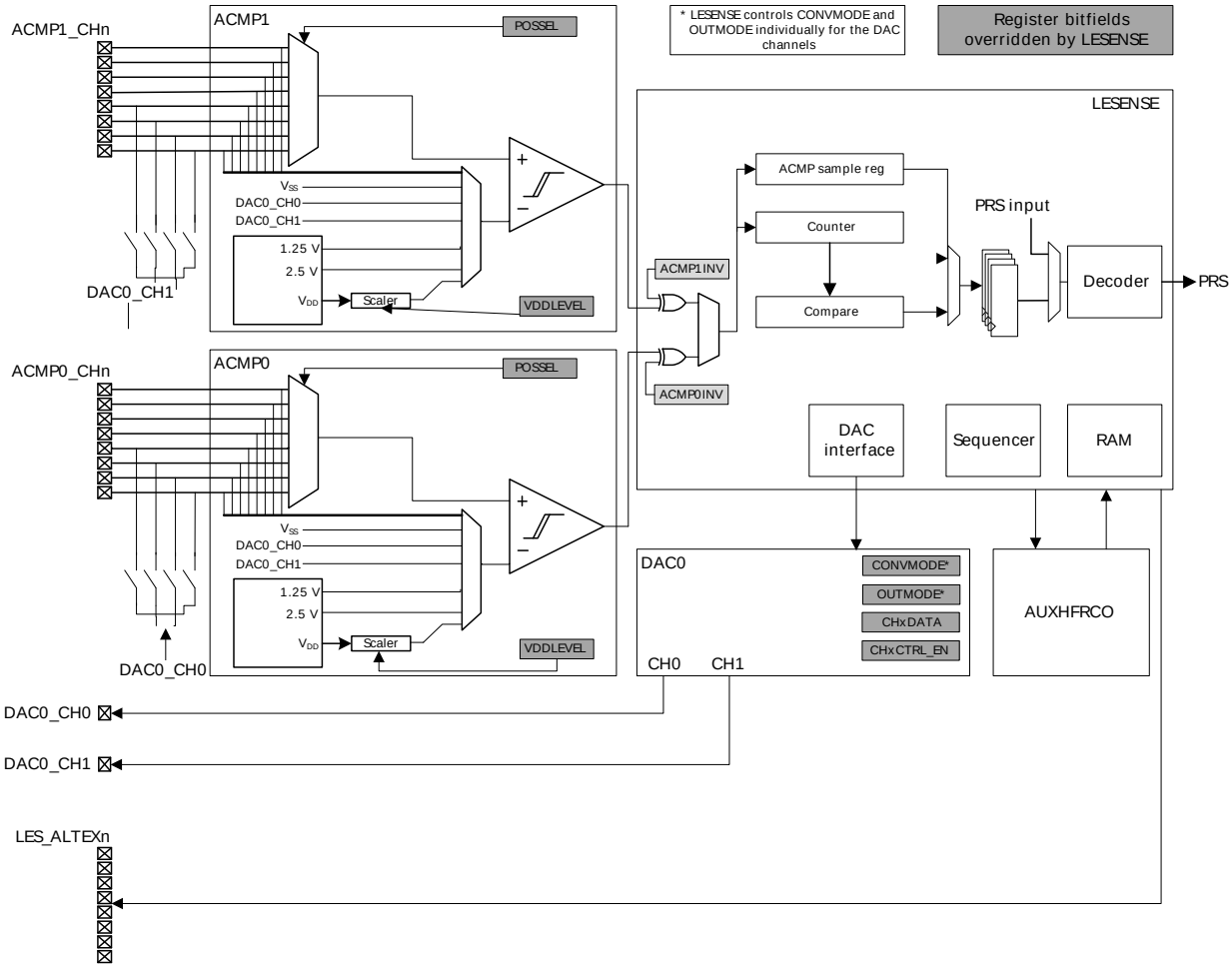
SILICON LABS

1 Introduction

1.1 LESENSE

The Low Energy Sensor Interface (LESENSE) is a peripheral which utilizes other on-chip peripherals to perform measurement of a configurable set of sensors. LESENSE uses the analog comparators (ACMP) for measurement of sensor signals together with the DAC to generate accurate reference voltages or perform sensor excitation. Figure 1.1 (p. 2) gives an overview of the LESENSE peripheral. LESENSE consists of a sequencer, count and compare block, and a RAM block used for configuration and result storage. The sequencer handles interaction with other peripherals as well as timing of sensor measurements. The count and compare block is used to count pulses from ACMP outputs before comparing with a configurable threshold. To autonomously analyze sensor results, the LESENSE decoder provides possibility to define a finite state machine with up to 16 states, and programmable actions upon state transitions. This allows the decoder to implement a wide range of decoding schemes, for instance quadrature decoding. A RAM block is used for storage of configuration and measurement results. This allows LESENSE to have a relatively large result buffer enabling the chip to remain in a low energy mode for long periods of time while collecting sensor data. LESENSE can operate in EM2, in addition to EM1 and EM0 and can wake up the CPU on configurable events.

Figure 1.1. LESENSE Overview



The LESENSE supports multiple sensor types: inductive (LC), capacitive and general analog sensors. This application note will focus on how to configure the LESENSE to read a given number of LC sensors and what kind of actions can be taken based on those readings.

1.2 Inductive Sensing

Inductive sensors are electronic proximity sensors which are able to detect the presence of a conductive target. Some common applications of inductive sensors include metal detectors, traffic lights, car washes and various automated industrial applications. Since there is no need for physical contact inductive sensors are particularly used in harsh environmental conditions (e.g. dirty environments). High performance inductive sensors also known as eddy-current sensors can do high-resolution measurements of the position and/or change of position of any conductive target. Lower cost inductive sensors are used as proximity switches giving a simple ON/OFF output indicating whether a conductive target is present or not.

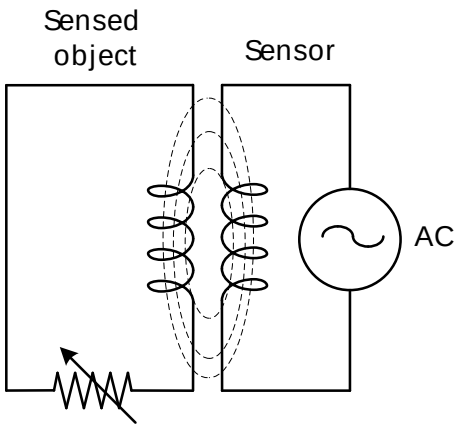
This application note will focus on the second type of sensors used to detect the presence of a conductive target.

2 Inductive Sensing

2.1 Theory

Inductive proximity sensors detect magnetic loss due to induced current generated on a conductive surface or target by an external magnetic field. When an AC current is applied to a coil called the detection coil, it generates an AC magnetic field. If a conductive target approaches the sensor it generates currents also known as eddy currents, on the sensed object due to the alternating magnetic field.

Figure 2.1. Transformer-like Coupling

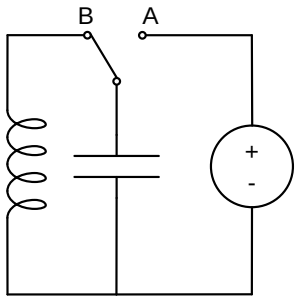


The relation between the detection coil and the sensed object is much like a transformer coupling (Figure 2.1 (p. 4)). When a conductive target approaches the coil the impedance of the coil changes. The change of impedance means that the magnetic flux through the coil changes and the apparent resistance seen by the coil increases because there is energy transfer between the coil and the target to generate the eddy currents. Commonly it is said that the impedance is loaded.

2.1.1 Tank circuit

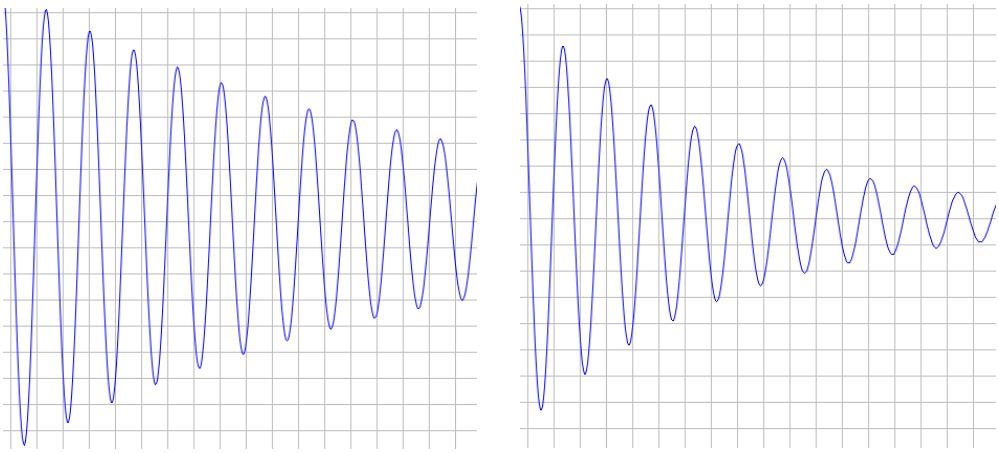
One way of producing oscillation on an inductance is using what is called a tank circuit (Figure 2.2 (p. 5) . The circuit consists of an inductive coil and a capacitor. The capacitor stores energy in the form of an electric field while the coil stores energy in the form of a magnetic field. When the switch is in position A the capacitor is charged up to the DC supply voltage. When the capacitor is fully charged the switch changes to position B placing the capacitor in parallel with the inductor coil and starts to discharge through the coil. The voltage across the capacitor starts falling as the current through the coil begins to rise. This rising current creates a magnetic field around the coil. When the capacitor is fully discharged the energy previously stored in the capacitor is now stored in the inductive coil.

Figure 2.2. Tank Circuit



Since there is no external voltage in the circuit to maintain the current within the coil it starts to fall and flows back to the capacitor which is then charged with the opposite polarity of its original charge. After that the whole cycle is repeated resulting in a periodic energy transfer between the two circuit elements. The polarity of the voltage changes as the energy is passed between the inductor and capacitor producing an AC voltage and current waveform.

Figure 2.3. Oscillation Damping

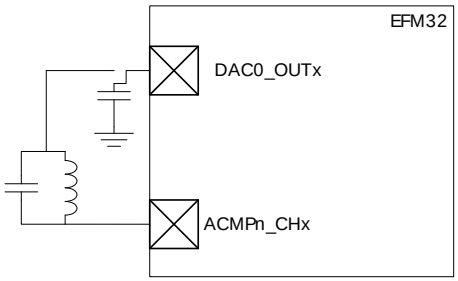


However, every time energy is transferred between the two circuit elements losses occur which will decay the oscillations. This is due to the resistive circuit components which will dissipate energy over time. The amplitude of the oscillation decreases at each half cycle of oscillation until the circuit loses all power. The oscillation is then said to be damped. If a metallic object is near the coil the currents induced in the target damp the oscillations quicker (Figure 2.3 (p. 5) right side) than if no object is present (Figure 2.3 (p. 5) left side).

2.2 LC Inductive Sensing with the EFM32

The LC sensing can be done with the EFM32's LESENSE using the setup depicted in Figure 2.4 (p. 6) .

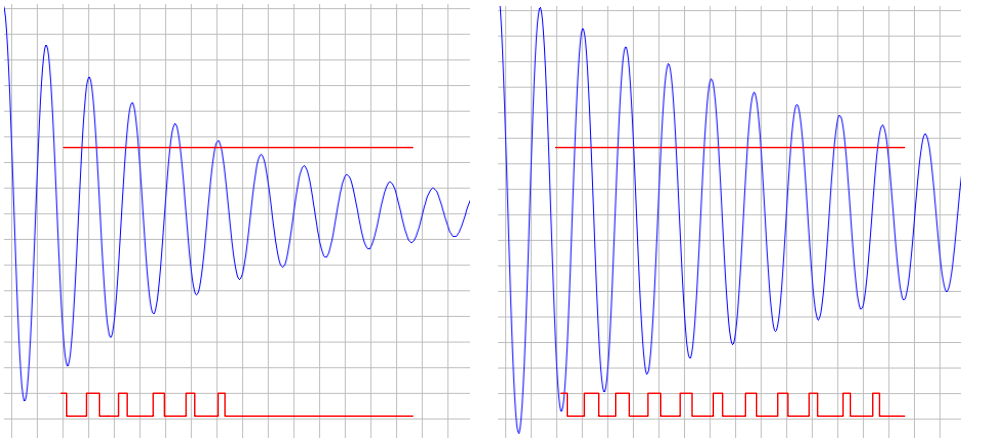
Figure 2.4. LC Setup



The sensor oscillation level should not be higher than Vdd. For that purpose the DAC has to be configured to generate a $V_{dd}/2$ level which will be the oscillations middle point. Initially the LC circuit has to be in a stable state which can be guaranteed if the voltage differential across both elements is zero. Therefore the excitation/measurement pin (ACMP_CH0) can be configured as either DAC output or disabled during the idle period. If configured as DAC output it will kill the oscillations after the measure phase and configuring the pin as disabled will allow the oscillations damp naturally towards $V_{dd}/2$

To excite the sensor the excitation/measurement pin should be pulled low which will charge the capacitor with a $V_{dd}/2$ level from the DAC output. The excitation time should be short so that the inductor does not cause a short circuit between the DAC and ground. To have a short excitation period it should be based on the number of AUXHFRCO cycles and start with 1 cycle until a proper excitation time is achieved. After the excitation period comes the measurement phase and the pin goes to Hi-Z and is routed to the positive input of the ACMP with a scaled V_{dd} in the negative output. It is also possible to use the DAC to generate a more accurate reference. The oscillations in the LC circuit will trigger pulses on the output of the ACMP as depicted in Figure 2.5 (p. 6) .

Figure 2.5. ACMP Pulse Generation



With metallic object near the LC Without metallic object near the LC

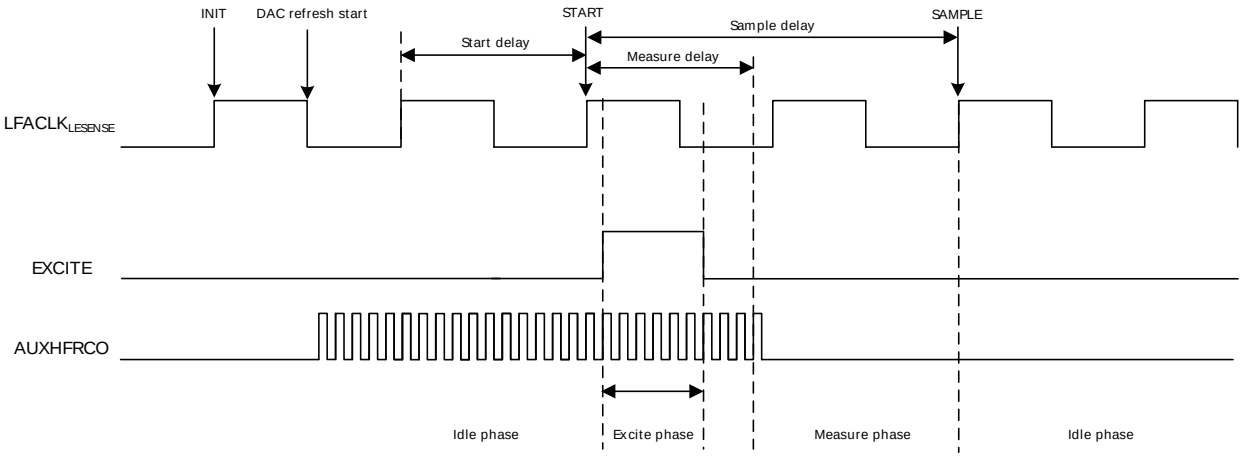
After the measure phase the excitation pin goes back to an idle state. The number of pulses can be counted and compared against a configurable threshold value to determine if a metallic object is present.

3 LESENSE

The LESENSE is an extremely configurable peripheral which allows interaction with a wide range of sensors. LESENSE is able to control the channel pins or DAC for sensor voltage excitation and the ACMP mux for sensor reading. The sensors can be excited and read using the same pin or using different pins which results in different sensor setups that can be implemented with the LESENSE. Each ACMP pin is a LESENSE channel and the number of ACMP pins yields the maximum number of LESENSE channels. By controlling the ACMP mux the LESENSE can scan through the different channels and either store the results in memory or feed them to a decoder as input for a configurable state machine (Section 3.6 (p. 10)).

When the LESENSE interacts with sensors there are 2 main phases: excitation and measure phase. These can use either the low or the high frequency clock as timebase and the duration is adjustable in number of clock cycles. The HF clock is driven by the AUXHFRCO and the LF clock by the LFACLK branch. In addition to these there is also the option of introducing a start delay which will delay both excitation and measure phase and a measure delay which will delay the measure phase only. The relation between the different phases and associated delays is depicted in Figure 3.1 (p. 7) .

Figure 3.1. Timing diagram



The AUXHFRCO is controlled by the LESENSE and enabled only when needed. For short excitation or measure phase it is recommended to use the AUXHFRCO clock as timebase.

The emlib comes with a set of functions to configure the LESENSE (efm32_lesense). Using these functions it is possible to setup the LESENSE in an easier manner. This chapter will show how to use these functions to setup the LESENSE for sensor interaction.

3.1 LESENSE Initialization

For the initialization of the LESENSE the function `void LESENSE_Init(LESENSE_Init_TypeDef const *init)` can be used. This function is intended to initialize the LESENSE once in an operation cycle and configures core, timing, peripheral and decoder parameters

3.1.1 Core configuration

The structure type `LESENSE_CoreCtrlDesc_TypeDef` defines the following parameters for the core control:

- Scan start mode to control how the scan start is triggered
- PRS source for scan start if PRS is selected to trigger a scan

- Scan configuration register usage (e.g. direct, inverse, toggle or decoder mapping)
- Invert ACMP0 output
- Invert ACMP1 output
- Scan ACMPs simultaneously
- Store SCANRES in RAM after each scan
- Always write result buffer even if full
- Trigger condition for interrupt and DMA
- Trigger condition for DMA wakeup from EM2
- Bias mode
- Keep LESENSE running in debug mode

3.1.2 Timing configuration

The structure type `LESENSE_TimeCtrlDesc_TypeDef` defines the following parameter for timing control:

- Number of LFACLK cycles to delay sensor interaction (Start Delay)

3.1.3 Peripheral configuration

The structure type `LESENSE_PerCtrlDesc_TypeDef` defines the following parameters for peripheral control:

- DAC channel 0 data control
- Configure LESENSE conversion control on DAC channel 0
- Configure LESENSE output control on DAC channel 0
- DAC channel 1 data control
- Configure LESENSE conversion control on DAC channel 1
- Configure LESENSE output control on DAC channel 1
- Prescaling factor for the LESENSE-DAC interface
- DAC reference to be used
- LESENSE control over ACMP0
- LESENSE control over ACMP1
- LESENSE control over ACMPs and DAC warm up in idle mode

3.1.4 Decoder configuration

The structure type `LESENSE_DecCtrlDesc_TypeDef` defines the following parameters for peripheral control:

- Input for the LESENSE decoder
- Initial state of the decoder
- Check the present state in addition to the ones defined in DEFCONF
- Set interrupt flag for CHx when a transition from state x occurs
- Enable hysteresis in the decoder for suppressing changes on PRS channel 0
- Enable hysteresis in the decoder for suppressing changes on PRS channel 1
- Enable hysteresis in the decoder for suppressing changes on PRS channel 2
- Enable hysteresis in the decoder for suppressing interrupt requests
- Enable count mode on decoder PRS channels 0 and 1 to produce output which can be used by a PCNT to count up or down
- PRS channel input for bit 0 of the LESENSE decoder
- PRS channel input for bit 1 of the LESENSE decoder
- PRS channel input for bit 2 of the LESENSE decoder

- PRS channel input for bit 3 of the LESENSE decoder

3.2 Clock Prescaling

The function `LESENSE_ClkDivSet(LESENSE_ChClk_TypeDef const clk, LESENSE_ClkPresc_TypeDef const clkDiv)` sets the prescaler value for the high and low frequency clocks of the LESENSE. The maximum prescaling values are 8 and 128 respectively and the resulting frequency is given by Equation 3.1 (p. 9) .

Prescaling equation

$$PRESC\_CLK_{freq} = CLK_{freq} / 2\text{^}PRESC_{value} \quad (3.1)$$

For the AUXHFRCO the $PRESC_{value}$ bitfield is AUXPRESC and for the LFACLK is LFPRESC, both in the LESENSE_TIMCTRL register.

3.3 Setting Scan Frequency

The function `LESENSE_ScanFreqSet(uint32_t refFreq, uint32_t const scanFreq)` allows to set the scan frequency for the LESENSE. The calculation is based on Equation 3.2 (p. 9) and it does not necessarily result in the requested scan frequency due to integer division.

Prescaling equation

$$F_{scan} = LFACLK_{LESENSE} / ((1 + PCTOP) \times 2^{PCPRESC}) \quad (3.2)$$

3.4 Channel Configuration

The LESENSE channels can be configured either by using the function `LESENSE_ChannelConfig(LESENSE_ChDesc_TypeDef const *confCh, uint32_t const chIdx)` which configures a single channel or `LESENSE_ChannelAllConfig(LESENSE_ChAll_TypeDef const *confChAll)` which configures all channels.

The structure `LESENSE_ChDesc_TypeDef` defines the following parameters for channel configuration:

- Enable channel scan
- Enable channel pin
- Enable channel interrupts after configuring all the sensor parameters
- Configure GPIO mode for the excitation phase of the scan sequence
- Configure channel pin setup in idle phase
- Use alternate excitation pin
- Enable channel result shift into the decoder register
- Invert result bit stored in the scan result register (SCANRES)
- Enable result storage in RAM
- Select clock for excitation timing
- Select clock for sample delay timing
- Configure excitation time
- Configure sample delay time
- Configure measure delay time
- Configure ACMP threshold
- Select ACMP output or counter output for comparison
- Configure interrupt generation mode for CHx interrupt flag
- Configure decision threshold for counter
- Select mode for counter comparison

To enable LESENSE to control the GPIO pins they have to be configured as push-pull. Please refer to AN0012 GPIO for more information on pin configuration.

After the LESENSE is fully configured the scan can start by using `LESENSE_ScanStart()` and stopped using `LESENSE_ScanStop()`.

3.5 Alternate Excitation

LESENSE is able to perform sensor excitation on another pin than the one to be measured. When ALTEX in CHx_INTERACT is set, the excitation will occur on the alternative excite pin associated with the given channel. All LESENSE channels mapped to ACMP0 have their alternative channel mapped to the corresponding channel on ACMP1, and vice versa. Alternatively, the alternative excite pins can be routed to the LES_ALTEX pins. Mapping of the alternative excite pins is configured in ALTEXMAP in CTRL. Table 3.1 (p. 10) summarizes the mapping of excitation pins for different configurations.

Table 3.1. LESENSE excitation pin mapping

LESENSE channel	ALTEX = 0	ALTEX = 1	
		ALTEXMAP = ACMP	ALTEXMAP = ALTEX
0	ACMP0_CH0	ACMP1_CH0	LES_ALTEX0
1	ACMP0_CH1	ACMP1_CH1	LES_ALTEX1
2	ACMP0_CH2	ACMP1_CH2	LES_ALTEX2
3	ACMP0_CH3	ACMP1_CH3	LES_ALTEX3
4	ACMP0_CH4	ACMP1_CH4	LES_ALTEX4
5	ACMP0_CH5	ACMP1_CH5	LES_ALTEX5
6	ACMP0_CH6	ACMP1_CH6	LES_ALTEX6
7	ACMP0_CH7	ACMP1_CH7	LES_ALTEX7
8	ACMP1_CH0	ACMP0_CH0	LES_ALTEX0
9	ACMP1_CH1	ACMP0_CH1	LES_ALTEX1
10	ACMP1_CH2	ACMP0_CH2	LES_ALTEX2
11	ACMP1_CH3	ACMP0_CH3	LES_ALTEX3
12	ACMP1_CH4	ACMP0_CH4	LES_ALTEX4
13	ACMP1_CH5	ACMP0_CH5	LES_ALTEX5
14	ACMP1_CH6	ACMP0_CH6	LES_ALTEX6
15	ACMP1_CH7	ACMP0_CH7	LES_ALTEX7

The alternate excitation pins can be configured using the `LESENSE_AltExConfig(LESENSE_ConfAltEx_TypeDef const *confAltEx)` function in the emlib. The `LESENSE_ConfAltEx_TypeDef` parameter structure allows to:

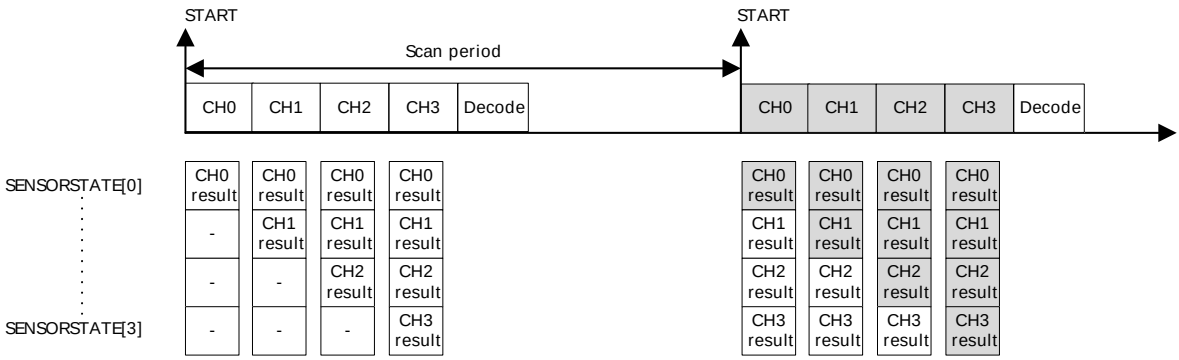
- Select alternate excitation mapping
- Enable alternate excitation pin
- Configure idle phase setup of alternate excitation pins
- Configure if alternate excitation pins should excite for all channels or only the corresponding channel

3.6 State Machine

Many applications require some sort of processing of the sensor readings, for instance in the case of quadrature decoding. In quadrature decoding, the sensors repeatedly pass through a set of states which

corresponds to the position of the sensors. This sequence, and many other decoding schemes, can be described as a finite state machine. To support this type of decoding without CPU intervention, LESENSE includes a highly configurable decoder, capable of decoding input from up to four sensors. The decoder is implemented as a programmable state machine with up to 16 states. When doing a sensor scan, the results from the sensors are placed in the decoder input register, SENSORSTATE, if DECODE in CHx_INTERACT is set. The resulting position after a scan is illustrated in Figure 3.2 (p. 11), where the bottom blocks show how the SENSORSTATE register is filled. When the scan sequence is complete, the decoder evaluates the state of the sensors chosen for decoding, as depicted in Figure 3.2 (p. 11) .

Figure 3.2. Sensor scan and decode sequence



The decoder is a programmable state machine with support for up to 16 states. The behavior of each state can be individually configured

The decoder state can be configured using the function `LESENSE_DecoderStateConfig(LESENSE_DecStDesc_TypeDef const *confDecSt, uint32_t const decSt)`. The structure type `LESENSE_DecStDesc_TypeDef` allows to configure the following parameters:

- Enable chaining the descriptor, meaning that the next descriptor pair will also be evaluated
- State condition descriptor A
  - Comparator value for sensor state
  - Comparator mask to exclude sensors from evaluation
  - Next state to be entered if sensor state equals compare value
  - PRS action to perform if sensor state equals compare value
  - Set interrupt flag if sensor state equals compare value
- State condition descriptor B
  - The same options as descriptor A

After configuring all the needed states it is necessary to initialize the state machine to indicate which is the initial state. This is done by writing to the LESENSE_DECSTATE register and the function `LESENSE_DecoderStateGet()` can be used for that purpose.

The state machine can start by using `LESENSE_DecoderStart()` and stopped using `LESENSE_DecoderStart()`.

4 Software Example

This application notes comes with a software example for the EFM32TG and EFM32GG STK which uses the inductive sensor in the lower right hand corner of the kit.

4.1 Single/Accumulated Sensor Reading

The `lcsense_single` and `lcsense_accumulated` example projects set up the LESENSE to interact with the LC sensor and use the USER LED to acknowledge the presence of metal near the sensor. A sensor trigger will light up the USER LED for 3 seconds. Depending on the metal, the sensor range should be 5-6 mm. In `lcsense_single` the LED goes on every time there is a sensor trigger. In `lcsense_accumulated` this happens after 5 sensor triggers. In this case the number of sensor triggers is counted using the PCNT together with PRS. Bit 6 from SCANRES is used as the PRS signal for the PCNT.

The reference for the ACMP was chosen to be very close to the voltage generated by the DAC. This way we will count more cycles and also increase the sensor range.

The counter threshold is not set in the initial LESENSE configuration. Instead the sensor is calibrated after configuring the LESENSE. The way this is done is assuming that there is no metal near the sensor when the program starts, the result buffer is filled and the results will to be the maximum number of pulses for the sensor, so any metal that comes close to the sensor will have to decrease that number. The last value from the result buffer array is then used as counter threshold.

Both projects use a scanning frequency of 20Hz which results in a current consumption of 1.2µA. The current consumption goes up with scanning frequency and for 50Hz and 100Hz the current consumption is 1.5µA and 1.9µA respectively.

To estimate the LESENSE base current the scanning frequency was set to 1Hz which results in 1.1µA. This allows us to calculate that the current increases roughly 8nA per Hz. This is the difference between the current for 100Hz and 1Hz dividing by 100.

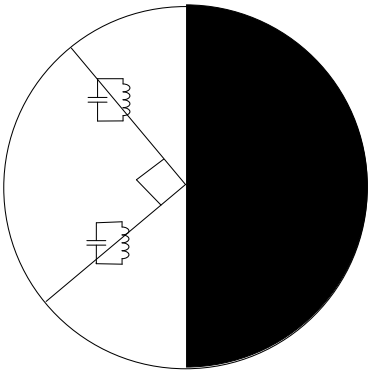
Although these are orientational numbers for scanning one sensor the impact of adding more sensors is the same as increasing the scanning frequency for one sensor. If one sensor is added while maintaining the same scanning frequency the result in current consumption is the same as keeping one sensor but doubling the scanning frequency.

The used LESENSE configuration together with the LC hardware gives a detection range of approximately 5mm. Increasing the capacitor and inductor in the LC loop will also increase de detection range but it also increases the current consumption.

4.2 Two sensors and State Machine Implementation

Example project `lcsense_state_machine` shows how to implement a state machine with two LC sensors. This was done by connecting a second LC sensor to the STK. The Tiny Gecko STK uses PC12 (LESENSE channel 12) and PB12 (DAC output) while the Giant gecko STK uses PC11 (LESENSE channel 11) amd PB12 (DAC output). The setup is used for rotation sensing with 2 sensors in a 90° separation (Figure 4.1 (p. 13)). A disk rotates on top of the sensors where half has a metal coating to trigger the sensors.

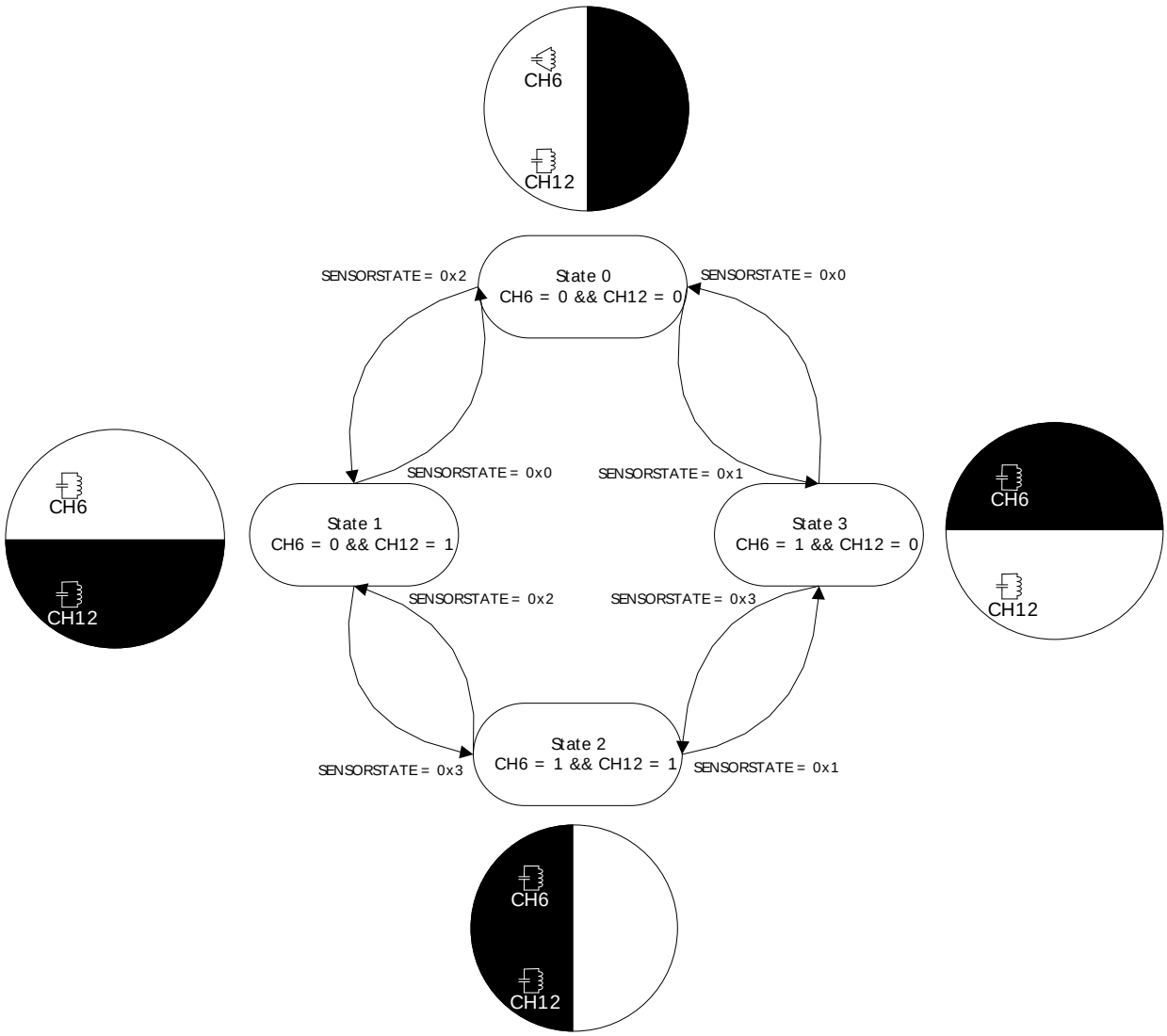
Figure 4.1. Sensor Placement



Using this setup it is possible to detect ¼ rotation of the disk. The sensor that is integrated in the STK is connected to LESENSE channel 6 and the additional sensor to channel 12.

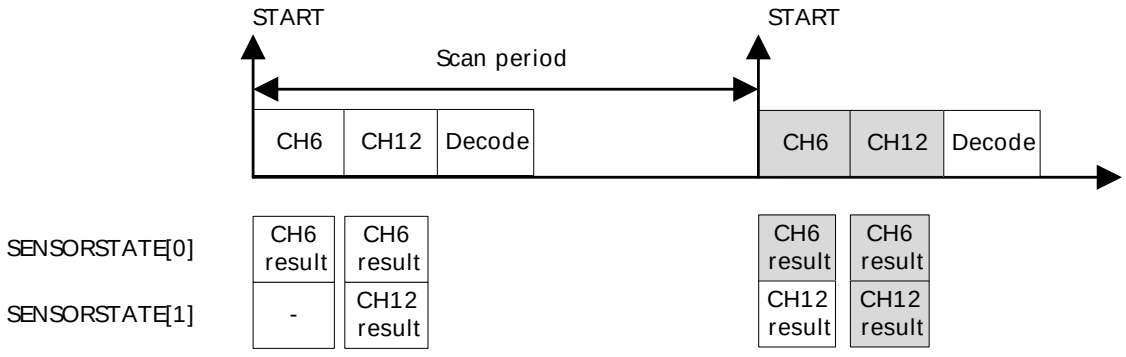
Both sensors are read at a frequency of 100Hz and the scanning result is used to feed the decoder which determines the rotation state based on the state machine depicted in Figure 4.2 (p. 14) .

Figure 4.2. State Machine



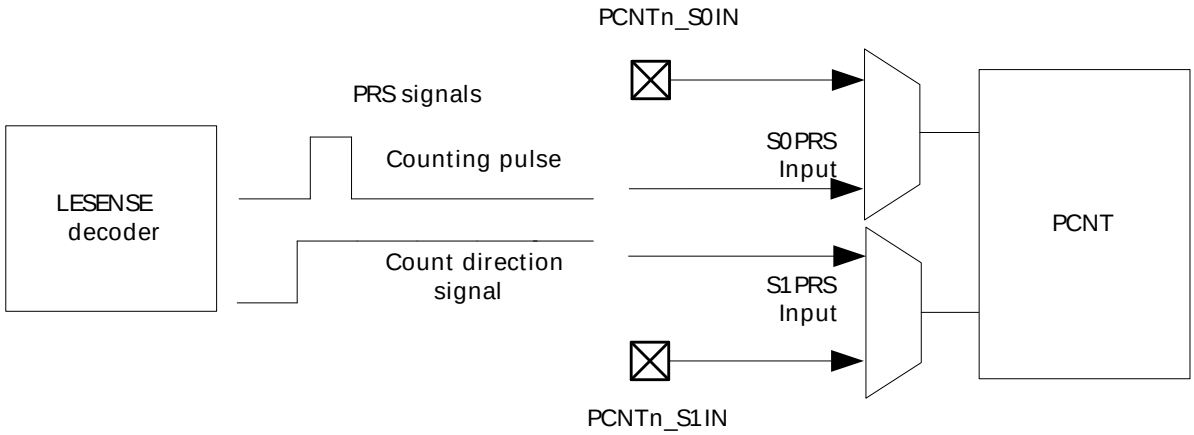
The SENSORSTATE register is filled with the sensor scan results as it is shown in Figure 4.3 (p. 14) .

Figure 4.3. Sensor State



In this example the pulse counter is used to count the revolutions and detect rotation direction changes. This is done by using the quadrature decoding feature in the decoder which outputs a PRS signal for counting and one for count direction to be used by the pulse counter (Figure 4.4 (p. 15)).

Figure 4.4. Sensor State



When the state machine jumps between states 0 and 3 a counting pulse is issued and the count direction signal will be high (up counting). If the state machine jumps between states 3 and 0 the same counting pulse is issued but now the direction signal will be low (down counting). Please note that the direction signal will be driven to the correct level before the pulse is issued to ensure correct operation. However the PCNT has to be correctly configured so that the counting is done in the intended direction. For more information on the pulse counter please refer to the AN0024 Pulse Counter. Also on transition between states 0 and 3 or 3 and 0 the LCD shows the number of revolutions and when there is a change in direction the string "DIRCHNG" is written in the LCD.

5 Revision History

5.1 Revision 1.06

2013-09-03

New cover layout

5.2 Revision 1.05

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

5.3 Revision 1.04

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

Added software support for Giant Gecko STK.

5.4 Revision 1.03

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

5.5 Revision 1.02

2012-03-14

Fixed a warning in the Keil example project.

5.6 Revision 1.01

2011-10-21

Updated IDE project paths with new kits directory.

5.7 Revision 1.00

2011-05-26

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, the Silicon Labs logo, Energy Micro, EFM, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

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

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

1. Introduction 2
1.1. LESENSE 2
1.2. Inductive Sensing 3
2. Inductive Sensing 4
2.1. Theory 4
2.2. LC Inductive Sensing with the EFM32 5
3. LESENSE 7
3.1. LESENSE Initialization 7
3.2. Clock Prescaling 9
3.3. Setting Scan Frequency 9
3.4. Channel Configuration 9
3.5. Alternate Excitation 10
3.6. State Machine 10
4. Software Example 12
4.1. Single/Accumulated Sensor Reading 12
4.2. Two sensors and State Machine Implementation 12
5. Revision History 16
5.1. Revision 1.06 16
5.2. Revision 1.05 16
5.3. Revision 1.04 16
5.4. Revision 1.03 16
5.5. Revision 1.02 16
5.6. Revision 1.01 16
5.7. Revision 1.00 16
A. Disclaimer and Trademarks 17
A.1. Disclaimer 17
A.2. Trademark Information 17
B. Contact Information 18
B.1. 18

List of Figures

1.1. LESENSE Overview 2
2.1. Transformer-like Coupling 4
2.2. Tank Circuit 5
2.3. Oscillation Damping 5
2.4. LC Setup 6
2.5. ACMP Pulse Generation 6
3.1. Timing diagram 7
3.2. Sensor scan and decode sequence 11
4.1. Sensor Placement 13
4.2. State Machine 14
4.3. Sensor State 14
4.4. Sensor State 15

List of Tables

3.1. LESENSE excitation pin mapping 10

List of Equations

3.1. Prescaling equation 9
3.2. Prescaling equation 9

silabs.com


ZERO
ARM Cortex-M0+
TINY
ARM Cortex-M3
GECKO
ARM Cortex-M3
LEOPARD
ARM Cortex-M3
GIANT
ARM Cortex-M3
WONDER
ARM Cortex-M4