EFM32
... the world's most energy friendly microcontrollers

Low Energy Sensor Interface - Resistive sense

AN0036 - Application Note

Introduction

This application note covers the basics of analog and describes how to use the Low Energy Sensor Interface (LESENSE) to scan a number of resistive 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

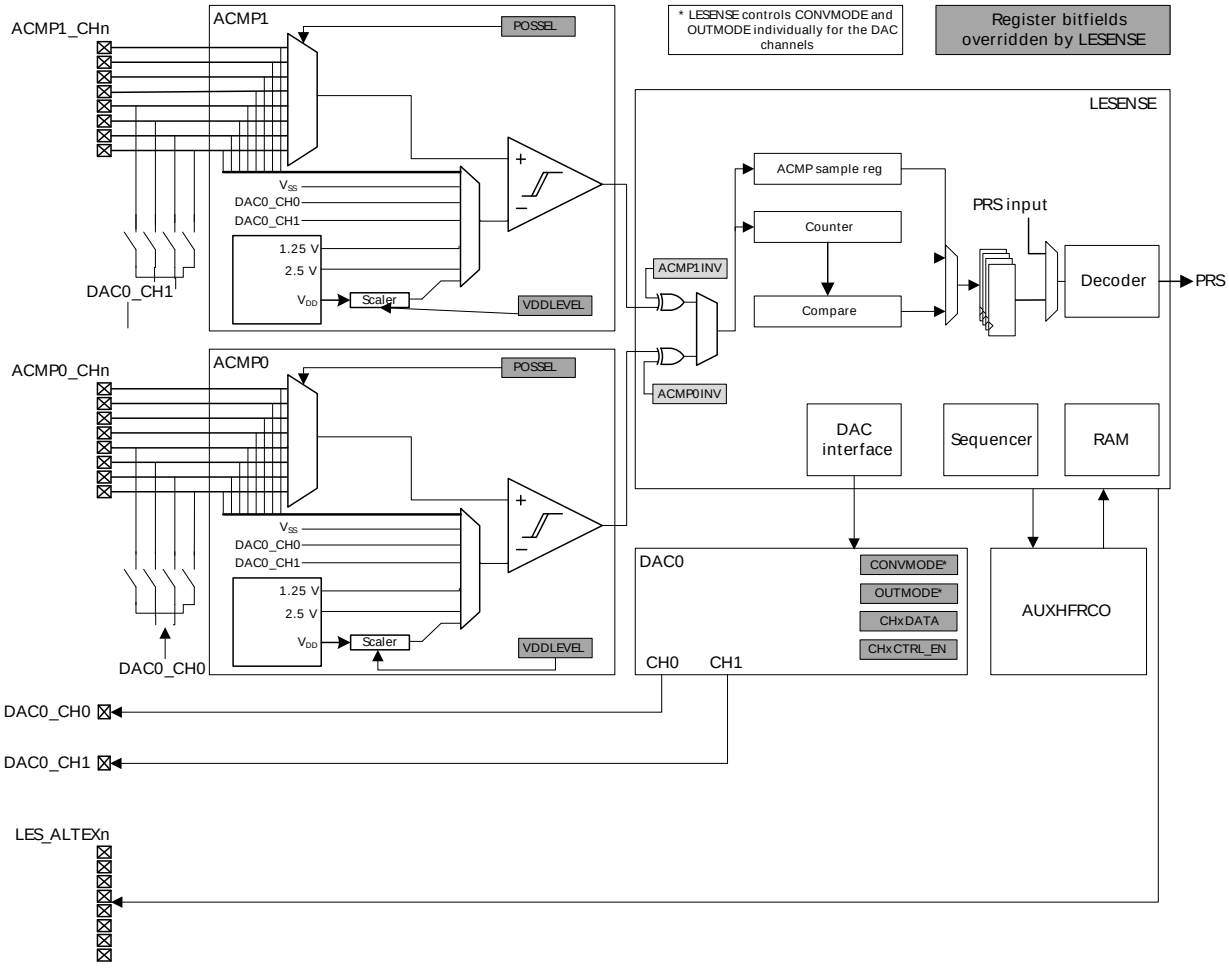
SILICON LABS

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 analog sensors with an example being given for a photo transistor.

1.2 Analog Sensors

Analog sensors are widely used in all kind of applications for measuring continuous information, unlike digital sensors which produce discrete (discontinuous) values to represent information. Some types of analog sensors include:

- Humidity
- Temperature
- Light
- Pressure
- Strain Gauge
- Potentiometers

The normal reading procedure for an analog sensor consists in applying a certain supply voltage (depending on the sensor characteristics) and reading an output voltage/current which has a mathematical relation with the measured physical quantity. Coupled to analog sensors are very commonly signal conditioning circuits which translate the sensor's electric output so it can be used by a reading device (e.g. microcontroller).

2 Resistive Sensing

2.1 Theory

Resistive sensors are the most basic type of analog sensors. These sensors display a change in their electrical resistance and when placed in an electric circuit such as a voltage divider or a wheatstone bridge produce a voltage signal equivalent to the measured physical quantity. Very commonly used resistive sensors include potentiometers, light sensors (photo resistors) or temperature sensors (thermistor).

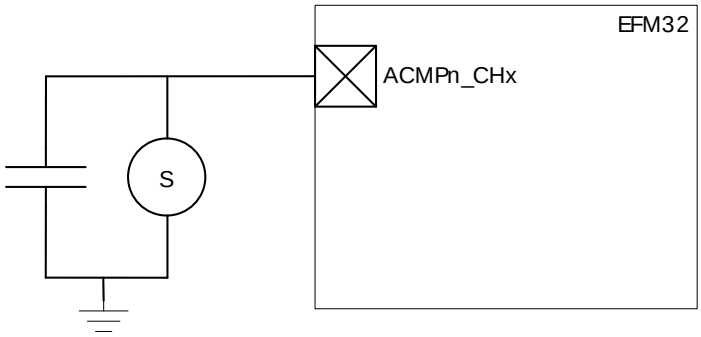
2.2 Resistive Sense in the EFM32

The resistive sensor reading in the EFM32 can be done using either one or two pins.

2.2.1 One-pin Resistive Sensor Reading

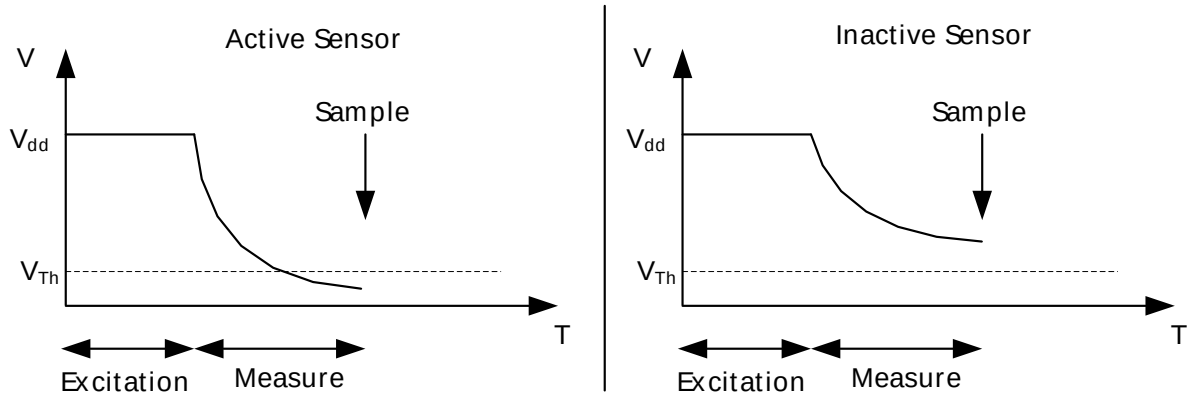
Resistive sensors can be read using the LESENSE in the EFM32 using only one LESENSE pin. This means that the pin will be used for both sensor excitation and sensor reading. Because of this double role a capacitor must be placed in parallel with the sensor to hold the voltage level when the pin goes from the excitation to the measure phase (Figure 2.1 (p. 4)).

Figure 2.1. One-pin Resistive Sensor Reading



Using this setup the capacitor will discharge through the sensor during the measure phase thus the discharge curve depends of the sensor resistance. The LESENSE measurement threshold has to be calibrated according to the sensor threshold that is to be measured. Figure 2.2 (p. 5) shows the hypothetical discharge curve of a sensor on an active state (left side) and inactive state (right side).

Figure 2.2. Sensor Discharge

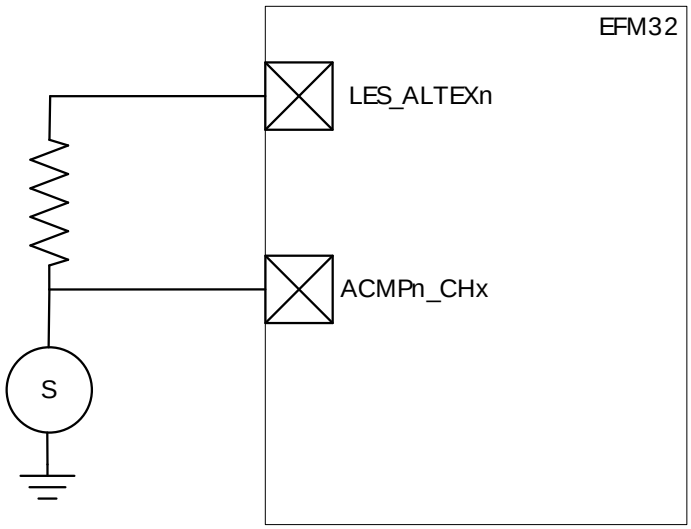


The LESENSE controls the ACMP mux for the positive channel and can also optionally control the mux for the negative input and Vdd scaling (V_{th}) for threshold calibration. The measurement of the discharge curve is then done by the ACMP which has an output high while the curve is above the threshold and output low when the curve drops below the threshold. The output of the ACMP is sampled in the end of the measure phase by the LESENSE indicating if the sensor is active or not.

2.2.2 Two-pin Resistive Sensor Reading

A resistive sensor can also be read using two pins where the alternative excitation pins (LES_ALTEXn) are used to excite the sensor. In this setup the sensor is excited also during the measure phase and a resistor divider is setup (Figure 2.3 (p. 5)) to give a stable voltage level dependent on the sensor resistance. It is important that the excitation phase is equal or longer than the measure phase to make sure that the sensor is being excited when sampled by the LESENSE.

Figure 2.3. Two-pin Resistive Sensor Reading



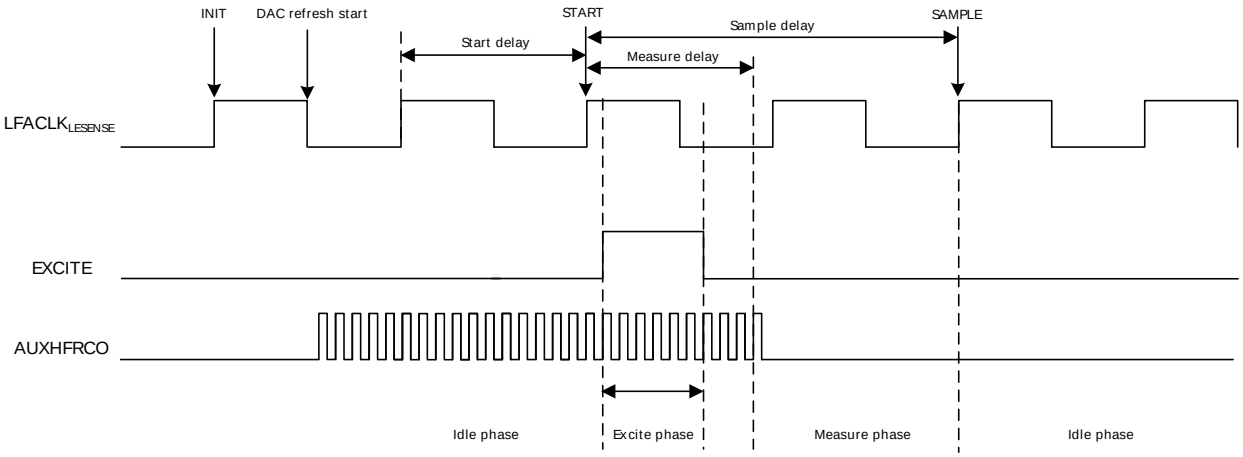
Similarly to Section 2.2.1 (p. 4) the ACMP is used to measure the sensor state and its output is sampled by the LESENSE in the end of the measure phase to determine if the sensor is active or not.

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. 9)).

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. 6) .

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. 8) .

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. 8) 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. 9) 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:

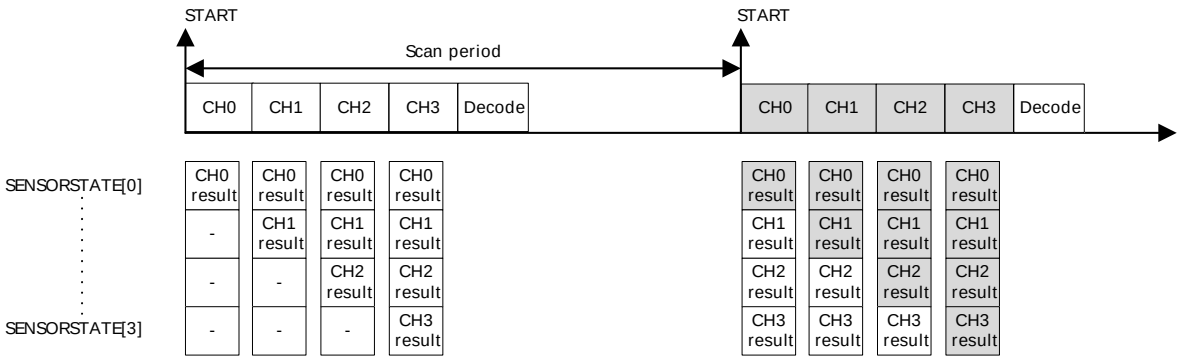
- Select alternate excitation mapping
- Enable alternate excitation pin
- Configure idle phase setup of alternate excitation pins
- Configure if alternate excitation pins should excite for all channels or only the corresponding channel

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. 10), 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. 10) .

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

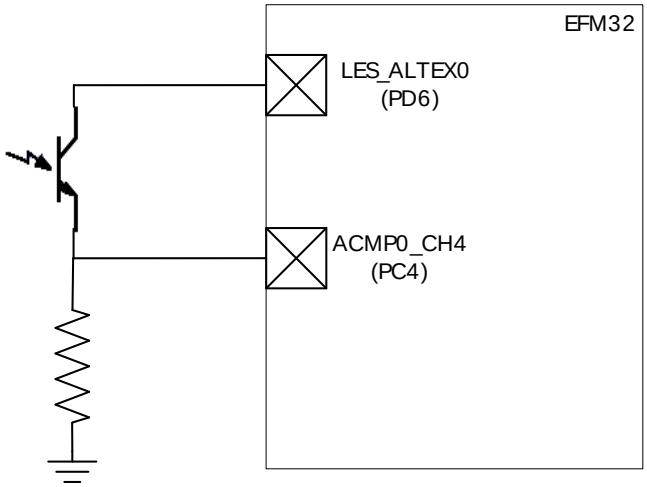
After configuring all the needed states it is necessary to initialize the state machine to indicate which is the initial state. This is done by writing to the LESENSE_DECSTATE register and the function `LESENSE_DecoderStateGet()` can be used for that purpose.

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

4 Software Example

This application note comes with a software example for resistive sensor reading which uses the light sensor in the EFM32TG STK. When using the EFM32GG STK the LESENSE channel 4 is on pin PC6. The light sensor is a photo transistor where the light intensity determines the amount of current flowing through the transistor. A resistor is then connected between the sensor emitter and ground, and the voltage drop across the resistor can be used to determine the sensor state (Figure 4.1 (p. 11)).

Figure 4.1. Light Sensor Setup



The voltage drop depends on the current flowing through the resistor which is controlled by the light on the base of the transistor. Although not being a resistive sensor it behaves in the same way so it can be measured using a two pin measurement setup.

The transistor collector is connected to the alternate excitation pin (LES_ALTEX0) and the emitter and resistor node if connected to LESENSE channel 4 (ACMP0_CH4). The sensor excitation occurs during the whole measure phase.

When the sensor is triggered the user LED is lit up. In the project `lightsense_single` the led it lit each time the sensor triggers and in `lightsense_accumulated` it is lit every 5 times using the PRS and Pulse Counter (PCNT) to count the sensor triggers.

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 1.6µA and 2.3µA respectively.

To estimate the LESENSE base current the scanning frequency was set to 1Hz which results in 1µA. This allows us to calculate that the current increases roughly 13nA 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.

5 Revision History

5.1 Revision 1.07

2014-05-07

Updated example code to CMSIS 3.20.5

Changed to Silicon Labs license on code examples

Added project files for Simplicity IDE

Removed makefiles for Sourcery CodeBench Lite

5.2 Revision 1.06

2013-10-14

New cover layout

5.3 Revision 1.05

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

5.4 Revision 1.04

2012-12-06

Added comments to source files to aid in understanding the code.

5.5 Revision 1.03

2012-11-12

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

Added software support for Giant Gecko STK.

5.6 Revision 1.02

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

5.7 Revision 1.01

2012-03-14

Fixed redefine warning for Rowley projects.

5.8 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

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

List of Figures

List of Tables

List of Equations

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