KU LEUVEN

LABO 2: PROJECT NOISELEVEL

Embedded Systems Design 1

Mathijs Schokkaert en Tuur Baele

Inhoud Verslag

Embedded project opdracht

Het doel van de opdracht is een systeem te maken die bij kan dragen tot een slimme campus. Voor dit project waren we vrij om te kiezen wat het systeem precies doet. In het project moet een draadloos communicatiesysteem zo lowpower mogelijk een sensor kunnen uitlezen. Het doorsturen van de data is nog niet verreist in deze opdracht maar kan gedaan worden via lora.

Keuze project

Voor ons project hebben we gekozen om een geluidsensor te plaatsen in de gangen die vervolgens het geluidsniveau kan doorsturen. Aan de hand van het geluidsniveau in de gang kan de drukte in de gang bepaald worden. Door vervolgens deze data te verzamelen op een centrale plaats kan in een applicatie de gangen van de plattegrond van de campus een kleur geven naarmate de drukte. In het voorbeeld onderaan met vier drukteniveaus met rood het drukst en donkergroen een lege/zeer rustige gang.

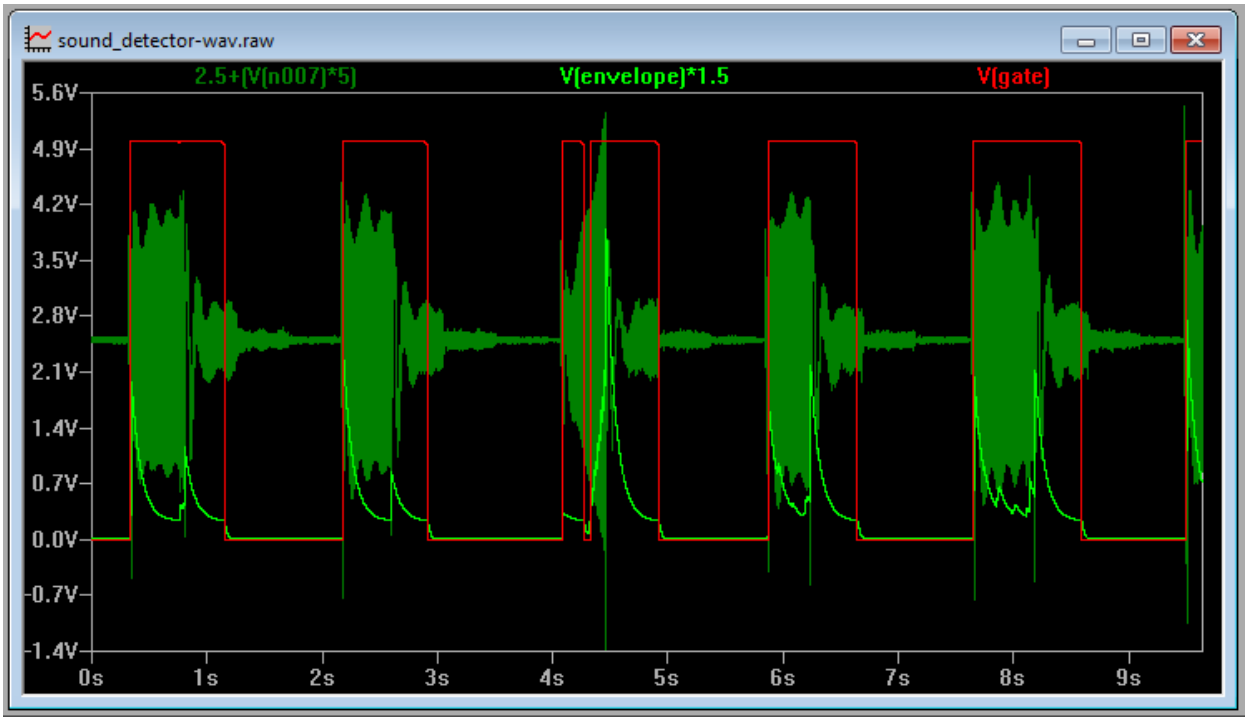
Door bijvoorbeeld in een applicatie te kijken naar de kleuren van de gangen kan een student bijvoorbeeld een andere route naar zijn lokaal kiezen. Een andere mogelijke toepassing van dit systeem is het gebruiken bij de lokalenplanning. Als bepaalde een gang vaak druk is kan men bepaalde lessen misschien beter in een ander lokaal laten doorgaan. Hiermee kan de campus efficiënter werken en de drukte in de gangen zoveel mogelijk beperkt worden.

De reden dat we kozen voor deze toepassing is door ervaringen in het schakeljaar waar we bijvoorbeeld door de drukte in de gangen soms meer dan 5 minuten verloren door de extreme drukte in de gang tijdens lokalenwissels.

Sensor

Beschrijving

In ons project hebben we gebruik gemaakt van de sparkfun sound detector [1]. De reden waarom we voor deze sensor hebben gekozen is door zijn analoge uitgangssignalen die voor onze toepassing handig zijn. Een andere reden is de gate en envelope pinnen die ook handig zijn. De gate pin is een digitale pin die hoog is wanneer het geluidsniveau gedetecteerd door de microfoon hoger is dan een ingestelde threshold. De threshold kan ingesteld worden door de waarden van R17 en R3. De envelope is handig om de amplitude direct te kunnen uitlezen als een positieve spanning van 0 tot de voedingspanning. Op onderstaande figuur zijn alle meetbare signalen van de pinnen weergegeven.



De sensor wordt gevoed met een spanning van 3,3V waardoor de uitgangsignalen beperkt zijn tot 3,3V. Het grootste verbruik van de sensor is de opamp, die deel uitmaakt van de sensor, om de verschillende signalen te kunnen genereren. De tweede grote verbruiker is de led van de sensor. Om het stroomverbruik van de sensor te beperken kan de led uitgeschakeld worden door het pad SJ6 te desolderen. De standaard gain van de sensor is 20db, deze gain kan aangepast worden door een

andere weerstand te plaatsen. Hiermee kan de gevoeligheid van het syteem aangepast worden. Meer informatie hierover is te vinden in de hookup guide op sparkfun [1].

Aansluiten sensor

De sensor wordt gevoed door de sensor aan te sluiten op 3,3V en op GND van een Pearl Gecko. De envelope sluiten we aan op PA0 (ACMP) en op PA1 (ADC).

Code en werking

Onder de code is er een flowchart weergegeven die de werking van elke blok code schematisch verduidelijkt.

Code

```
/**************************************************************************
***
 * @file main_noise.c
 * @brief Noise meter
 * @author Tuur Baele Mathijs Schokkaert
 * @version 1.08

**************************************************************************
***
 * @section License
 * <b>(C) Copyright 2014 Silicon Labs, http://www.silabs.com</b>

**************************************************************************
****
 *
 * Permission is granted to anyone to use this software for any purpose,
 * including commercial applications, and to alter it and redistribute it
 * freely, subject to the following restrictions:
 *
 * 1. The origin of this software must not be misrepresented; you must not
 *    claim that you wrote the original software.
 * 2. Altered source versions must be plainly marked as such, and must not be
 *    misrepresented as being the original software.
 * 3. This notice may not be removed or altered from any source distribution.
 *
 * DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Silicon Labs has no
 * obligation to support this Software. Silicon Labs is providing the
 * Software "AS IS", with no express or implied warranties of any kind,
 * including, but not limited to, any implied warranties of merchantability
 * or fitness for any particular purpose or warranties against infringement
 * of any proprietary rights of a third party.
 *
 * Silicon Labs will not be liable for any consequential, incidental, or
 * special damages, or any other relief, or for any claim by any third party,
 * arising from your use of this Software.
 *

**************************************************************************
***/


#include <stdio.h>
#include "em_device.h"
#include "em_chip.h"
#include "em_emu.h"
```

```
#include "em_cmu.h"
#include "em_acmp.h"
#include "em_adc.h"
#include "em_rtcc.h"


#include "bsp.h" //leds

//Sampling
#define NUMBER_OF_SAMPLES                       10


//Defines for RTC
#define LFRCO_FREQUENCY              32768
#define WAKEUP_INTERVAL_MS           100
#define RTC_COUNT_BETWEEN_WAKEUP     (((LFRCO_FREQUENCY *
WAKEUP_INTERVAL_MS) / 1000)-1)

/* Definess for ADC */
#define ADC_CLOCK            1000000          /* ADC conversion clock */
//#define ADC_INPUT0          adcPosSelAPORT3XCH8    /* PA0 */
#define ADC_INPUT0           adcPosSelAPORT3YCH9    /* PA1 */
#define ADC_SCAN_DVL         4
#define ADC_BUFFER_SIZE      64
#define ADC_SE_VFS           (float)3.3          /* AVDD */
#define ADC_12BIT_MAX        4096               /* 2^12 */
#define ADC_16BIT_MAX        65536              /* 2^16 */
#define ADC_CMP_GT_VALUE     3724               /* ~3.0V for 3.3V AVDD */
#define ADC_CMP_LT_VALUE     620                /* ~0.5V for 3.3V AVDD */

/* Buffer for ADC interrupt flag */
volatile uint32_t adcIntFlag;

/* Buffer for ADC single and scan conversion */
uint32_t adcBuffer[ADC_BUFFER_SIZE];

float noiseLevel=0;
int counterTime=0;

/**************************************************************************
//**
* @brief
*   Init analog comparator.
**************************************************************************
****/
float adcSingleScan(bool ovs);


static void ACMPInit(void)
{
```

```
  const ACMP_Init_TypeDef acmp_init =
  {
    false,                /* fullBias */
    0x7,                  /* biasProg */
    false,                /* Disable interrupt for falling edge */
    true,                 /* Enable interrupt for rising edge */
    acmpInputRangeFull,   /* Input range from 0 to VDD. */
    acmpAccuracyLow,      /* Low accuracy, less current usage. */
    acmpPowerSourceAvdd,  /* Use the AVDD supply. */
    acmpHysteresisLevel5, /* Use hysteresis level 5 when output is 0 */
    acmpHysteresisLevel5, /* Use hysteresis level 5 when output is 1 */
    acmpVLPInputVADIV,    /* Use VADIV as the VLP input source. */
    false,                /* Output 0 when ACMP is inactive. */
    true                  /* Enable after init. */
  };

  /* Init and set ACMP channel */
  ACMP_Init(ACMP0, &acmp_init);

  /* We want to compare an analog input to the 1.5 V
   * internal reference. To achieve this we first have to configure the
   * VB source which is used for internal reference voltages. */
  ACMP_VBConfig_TypeDef vbconfig = ACMP_VBCONFIG_DEFAULT;
  vbconfig.input = acmpVBInput1V25;
  /**
    * A divider for VB voltage input source when ACMP output is 1. This value is
    * used to divide the VB voltage input source by a specific value. The valid
    * range is between 0 and 63.
    *
    *  VB divided = VB input * (div1 + 1) / 64
    */
  vbconfig.div0=31; //instellen op 0,725V
  vbconfig.div1=31; //instellen op 0,725V
  ACMP_VBSetup(ACMP0, &vbconfig);



  ACMP_ChannelSet(ACMP0,acmpInputVBDIV , acmpInputAPORT3XCH8);//
acmpInputAPORT3XCH8     /* PA0 */

  ACMP_IntEnable(ACMP0, ACMP_IEN_EDGE);   /* Enable edge interrupt */


  /* Wait for warmup */
  while (!(ACMP0->STATUS & ACMP_STATUS_ACMPACT)) ;

  /* Enable interrupts */
  NVIC_ClearPendingIRQ(ACMP0_IRQn);
  NVIC_EnableIRQ(ACMP0_IRQn);
}
```

```
void updateNoiseState(float level){
	if(level > 0.1){			//luidruchtige gang
		BSP_LedSet(1);
		BSP_LedSet(0);
	}
	else if(level > 0.055){		//drukke gang
		BSP_LedClear(0);
		BSP_LedSet(1);
	}else if(level > 0.05){		//licht bezette gang
		BSP_LedClear(1);
		BSP_LedSet(0);
	}else{					//stille gang
		BSP_LedClear(0);
		BSP_LedClear(1);
	}
}

void noiseMeasure(void){
	float value=0;
	for(int i=0;i<NUMBER_OF_SAMPLES;i++){
		value+=adcSingleScan(false);
	}
	value/=NUMBER_OF_SAMPLES;
	noiseLevel+=value;
	counterTime++;
	if(counterTime>50){
		noiseLevel/=50;
		counterTime=0;
		updateNoiseState(noiseLevel);
	}

}

void RTCC_IRQHandler(void)
{
 /* Clear interrupt source */

	RTCC_IntClear(RTCC_IFC_CC1);
	noiseMeasure();
}


/**************************************************************************//**
/**
 * @brief ACMP0 Interrupt handler

*****************************************************************************
**/
void ACMP0_IRQHandler(void)
{
```

```
 /* Clear interrupt flag */
 ACMP0->IFC = ACMP_IFC_EDGE;
 float demonstratie= adcSingleScan(false);
 updateNoiseState(demonstratie); //status onmiddelijk veranderen door plots sterk geluid
}


/**************************************************************************/
/**
 * @brief Initialize ADC for single and scan conversion

 ***************************************************************************
 **/
void adcSetup(void)
{
 /* Enable ADC clock */
 CMU_ClockEnable(cmuClock_ADC0, true);

 /* Select AUXHFRCO for ADC ASYNC mode so that ADC can run on EM2 */
 CMU->ADCCTRL = CMU_ADCCTRL_ADC0CLKSEL_AUXHFRCO;

 /* Initialize compare threshold for both single and scan conversion */
 ADC0->CMPTHR = _ADC_CMPTHR_RESETVALUE;
 ADC0->CMPTHR = (ADC_CMP_GT_VALUE << _ADC_CMPTHR_ADGT_SHIFT) +
           (ADC_CMP_LT_VALUE << _ADC_CMPTHR_ADLT_SHIFT);
}

void adcReset(void)
{
 uint32_t i;

 /* Switch the ADCCLKMODE to SYNC */
 NVIC_DisableIRQ(ADC0_IRQn);
 ADC0->CTRL &= ~ADC_CTRL_ADCCLKMODE_ASYNC;

 /* Rest ADC registers */
 ADC_Reset(ADC0);

 /* Fill buffer and clear flag */
 for (i=0; i<ADC_BUFFER_SIZE; i++)
 {
  adcBuffer[i] = ADC_CMP_LT_VALUE;
 }
 adcIntFlag = 0;

 /* Reset AUXHFRCO to default */
 CMU_AUXHFRCOFreqSet(cmuAUXHFRCOFreq_19M0Hz);

// /* Disable clock */
// CMU_ClockEnable(cmuClock_PRS, false);
```

```
// CMU_ClockEnable(cmuClock_LDMA, false);
}

float adcSingleScan(bool ovs)
{
 uint32_t sample, adcMax;

 ADC_Init_TypeDef init = ADC_INIT_DEFAULT;
 ADC_InitSingle_TypeDef singleInit = ADC_INITSINGLE_DEFAULT;
 ADC_InitScan_TypeDef scanInit = ADC_INITSCAN_DEFAULT;

 adcMax = ADC_12BIT_MAX;
 if (ovs)
 {
  adcMax = ADC_16BIT_MAX;
 }

 /* Init common issues for both single conversion and scan mode */
 init.timebase = ADC_TimebaseCalc(0);
 init.prescale = ADC_PrescaleCalc(ADC_CLOCK, 0);
 if (ovs)
 {
  /* Set oversampling rate */
  init.ovsRateSel = adcOvsRateSel256;
 }
 ADC_Init(ADC0, &init);

 /* Initialize for single conversion */
 singleInit.reference = adcRefVDD;
 singleInit.posSel = ADC_INPUT0;
 singleInit.negSel = adcNegSelVSS;
 if (ovs)
 {
  /* Enable oversampling rate */
  singleInit.resolution = adcResOVS;
 }
 ADC_InitSingle(ADC0, &singleInit);

 /* Setup scan channels, define DEBUG_EFM in debug build to identify invalid channel
range */
 ADC_ScanSingleEndedInputAdd(&scanInit, adcScanInputGroup0, ADC_INPUT0);
// ADC_ScanSingleEndedInputAdd(&scanInit, adcScanInputGroup0, ADC_INPUT1);
// ADC_ScanSingleEndedInputAdd(&scanInit, adcScanInputGroup1, ADC_INPUT2);
// ADC_ScanSingleEndedInputAdd(&scanInit, adcScanInputGroup1, ADC_INPUT3);

 /* Initialize for scan conversion */
 scanInit.reference = adcRefVDD;
 if (ovs)
 {
  /* Enable oversampling rate */
```

```
    scanInit.resolution = adcResOVS;
  }
  ADC_InitScan(ADC0, &scanInit);

  /* Set scan data valid level to trigger */
  ADC0->SCANCTRLX |= (ADC_SCAN_DVL - 1) <<
_ADC_SCANCTRLX_DVL_SHIFT;

  /* Start ADC single conversion */
  ADC_Start(ADC0, adcStartSingle);
  while ((ADC0->IF & ADC_IF_SINGLE) == 0)
    ;

  /* Get ADC single result */
  sample = ADC_DataSingleGet(ADC0);
//  printf("Single PA0: %1.4fV\n",((float)sample * ADC_SE_VFS)/adcMax);

  float adcValue= ((float)sample * ADC_SE_VFS)/adcMax;
  adcReset();

  return adcValue;
}

/**************************************************************************//**
/**
 * @brief  Start LFRCO for RTC/RTCC
 * Starts the low frequency RC oscillator (LFRCO) and routes it to the RTC/RTCC

 *****************************************************************************
**/
void startLfrcoForRtc(void)
{
  /* Starting LFRCO and waiting until it is stable */
  CMU_OscillatorEnable(cmuOsc_LFRCO, true, true);

  /* Enabling clock to the interface of the low energy modules */
  CMU_ClockEnable(cmuClock_CORELE, true);

#if defined(RTCC_PRESENT) && (RTCC_COUNT > 0)
  /* Routing the LFRCO clock to the RTCC */
  CMU_ClockSelectSet(cmuClock_LFE, cmuSelect_LFRCO);
  CMU_ClockEnable(cmuClock_RTCC, true);
#else
  /* Routing the LFRCO clock to the RTC */
  CMU_ClockSelectSet(cmuClock_LFA,cmuSelect_LFRCO);
  CMU_ClockEnable(cmuClock_RTC, true);
#endif
}

void setupRtc(void)
```

```
{
  /* Configuring clocks in the Clock Management Unit (CMU) */
  startLfrcoForRtc();

#if defined(RTCC_PRESENT) && (RTCC_COUNT > 0)
  RTCC_Init_TypeDef rtccInit = RTCC_INIT_DEFAULT;
  RTCC_CCChConf_TypeDef rtccInitCompareChannel =
RTCC_CH_INIT_COMPARE_DEFAULT;

  rtccInit.cntWrapOnCCV1 = true;       /* Clear counter on compare match */
  rtccInit.presc = rtccCntPresc_1;

  /* Setting the compare value of the RTCC */
  RTCC_ChannelInit(1, &rtccInitCompareChannel);
  RTCC_ChannelCCVSet(1, RTC_COUNT_BETWEEN_WAKEUP);

  /* Enabling Interrupt from RTCC */
  RTCC_IntEnable(RTCC_IEN_CC1);
  NVIC_ClearPendingIRQ(RTCC_IRQn);
  NVIC_EnableIRQ(RTCC_IRQn);

  /* Initialize the RTCC */
  RTCC_Init(&rtccInit);
#else
  RTC_Init_TypeDef rtcInit = RTC_INIT_DEFAULT;

  /* Setting the compare value of the RTC */
  RTC_CompareSet(0, RTC_COUNT_BETWEEN_WAKEUP);

  /* Enabling Interrupt from RTC */
  RTC_IntEnable(RTC_IEN_COMP0);
  NVIC_ClearPendingIRQ(RTC_IRQn);
  NVIC_EnableIRQ(RTC_IRQn);

  /* Initialize the RTC */
  RTC_Init(&rtcInit);
#endif
}
```

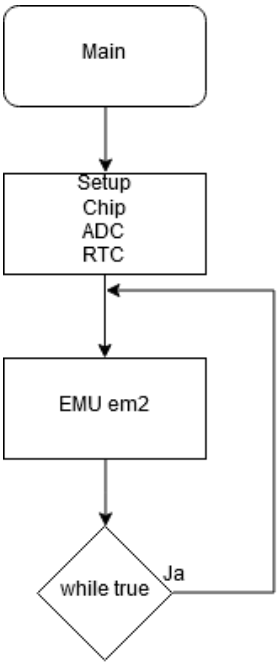
```
/**************************************************************************
***
 * @brief  Main function, enables analog comparator 0 and waits in em2 for
 * comparator interrupt, both rising and falling edges.
 * Main is called from _program_start, see assembly startup file
```

```
**********************************************************************
**/
int main(void)
{
  /* Initialize chip */
  CHIP_Init();

  /* Enable clocks required */
  CMU_ClockEnable(cmuClock_ACMP0, true);

  /* Init DCDC regulator if available */ //?????
#if defined( _EMU_DCDCCTRL_MASK )
  EMU_DCDCInit_TypeDef dcdcInit = EMU_DCDCINIT_DEFAULT;
  EMU_DCDCInit(&dcdcInit);
#endif

  /* Setup MCU clock to 4 MHz */
  CMU_HFRCOFreqSet(cmuHFRCOFreq_4M0Hz); //??

  adcSetup();

  /* Enable atomic read-clear operation on reading IFC register */
  MSC->CTRL |= MSC_CTRL_IFCREADCLEAR;

  ACMPInit();
  GPIO_PinModeSet(BSP_GPIO_PB0_PORT, BSP_GPIO_PB0_PIN,
gpioModeInputPullFilter, 1);
  BSP_LedsInit();
  //BSP_LedSet(0);


  //setting up RTC
  setupRtc();




  /* Stay in this loop forever at end of program */
  while (1)
  {
    /* Enter em2 and wait for acmp or RTC clock interrupt */
    EMU_EnterEM2(true);
  }
}
```

Flowcharts per methode

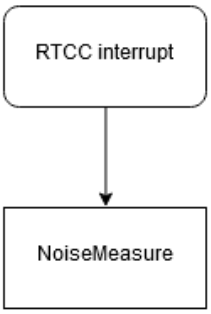
Main

In de main methode worden de verschillende setups doorlopen om alles klaar te zetten voor de werking van alle verschillende code onderdelen. Hierna gaat de code in een while loop die constant de energiemodus op em2 plaatst. Dit doen we zodat als er in een andere stuk code even naar een lagere energiemodus is moeten gaan dat hij vanzelf weer in de laagst mogelijke modus gaat.



RTCC interrupt

In deze methode wordt er om de 100ms een korte meting (100 samples) uitgevoerd van het geluid. De interrupt zorgt voor een laag stroomverbruik over het algemeen want tussen de interrupts kan de chip slapen in ‘deep sleep’ of em2.

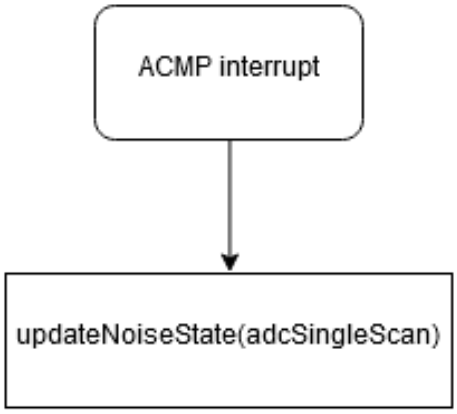


ACMP interrupt

Deze methode zal een interrupt genereren als de voltage op de ACMP een gekozen pin hoger is dan een bepaalde vooraf ingestelde referentiewaarde van 0,725V. De referentiewaarde van ACMP kan worden ingesteld door de vaste spanning U_{input} te delen door een bepaalde waarde div. U_{input} kan 1,25 of 2,5 volt zijn en div kan men instellen van 0 tot 63. De referentie spanning wordt dan gegeven door volgende formule:

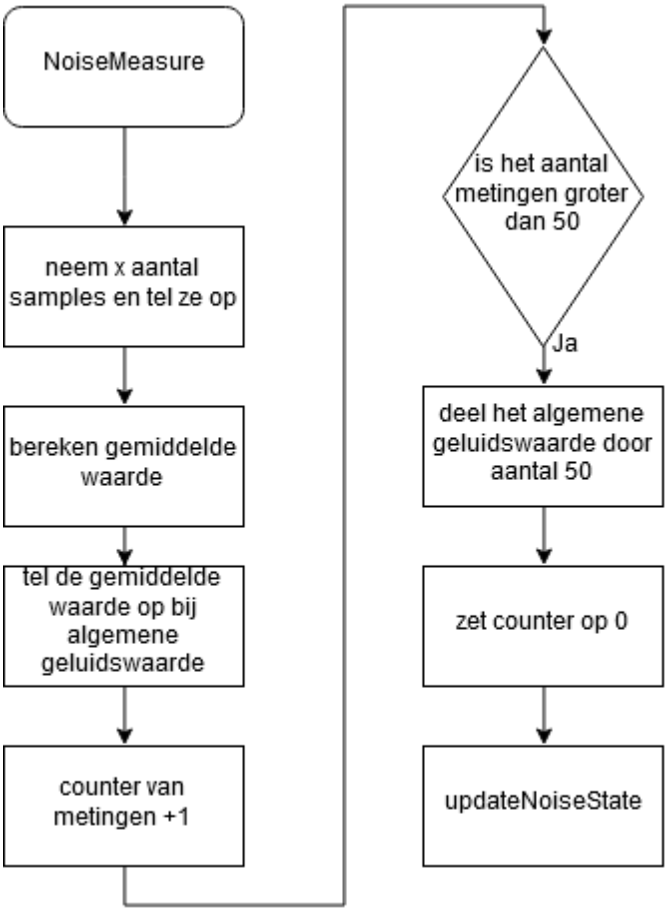
$$U_{referentie} = U_{input}\frac{div+1}{64} = 1{,}25V.\frac{31+1}{64} = 0{,}725V$$

Hierna wordt deze gemeten waarde direct doorgegeven aan de updateNoiseState methode. Deze waarde komt overeen met een zeer luide gang waardoor de status van de gang direct kan updaten naar een drukke gang. De ACMP kan gebruikt worden tot em3.



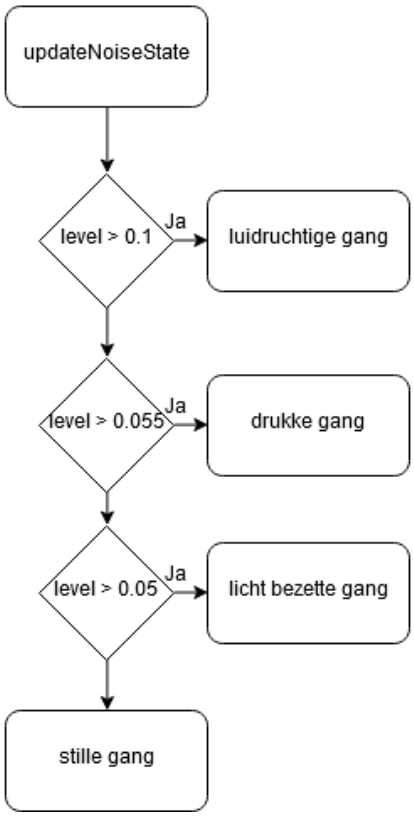
Noise measure

Deze methode neemt 100 (x) samples van de spanning van de sensor en neem daar het gemiddelde van om een nauwkeuriger meting van de spanning te bekomen. Hierna wordt het eerste gemiddelde bij een variabele opgeteld die zal gebruikt worden om een globaal gemiddelde te bekomen. Na 50 kleine metingen wordt het globaal gemiddelde berekend van deze 50 kleine metingen voor een globaal resultaat. Dit resultaat zal dus om de 50*100ms = 5 s geüpdatet worden, en zal daarna de noiseState updaten.



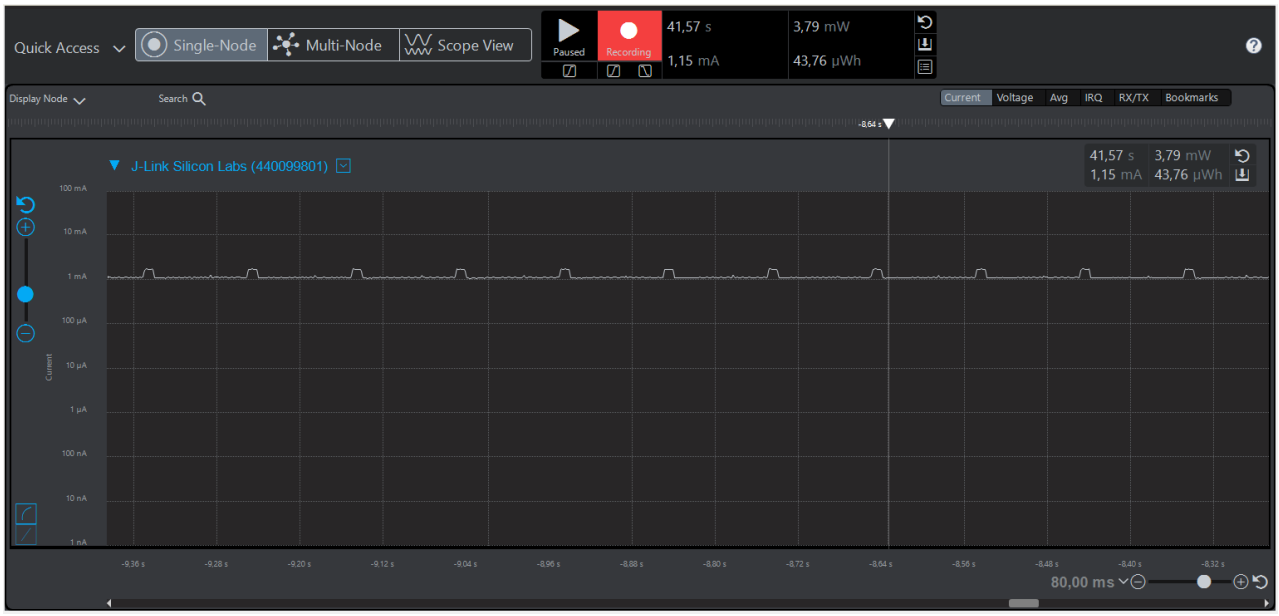
updateNoiseState

Deze methode bepaald de status van de drukte in de gang in een ifelse structuur. De waarden in de structuur, die overeenkomen met de spanning van de sensor, werden gekozen na verschillende tests in het labolokaal. Vanaf de spanning van de sensor hoger is dan 0,1 V dan komt dit overeen met een luidruchtige gang. Deze waarde werd vastgelegd door een test waarbij andere groepen in het labo aan het babbelen waren. Bij geen labogroepen aanwezig was de spanning gemiddeld lager dan 0,05V wat we in de toepassing als een 'licht bezette gang beschouwen. Deze waarden kunnen wellicht anders zijn in andere lokalen of op andere locaties van de sensor in het lokaal.



Stroomverbruik

Eerst wordt het energieverbruik weergegeven van de sensor gevoed door de VMCU pin van de Pearl Gecko zodat het volledige energieverbruik zichtbaar is. Er is een verbruik van ongeveer 1,15mA op het moment dat er niet gesampeld wordt. Dit is het verbruik van de processor in em2 met van 32µA en het samen met het verbruik van de sensor en de GPIO blok. Het stroomverbruik van de GPIO blok bedraagt volgens de hiervoor vermelde datasheet 5.32µA/MHz. Daaruit volgt dat de GPIO blok 21,28µA verbruikt. De sensor alleen verbruikt dus 1,15mA-32µA-21,28µA= 1,09mA.



Vervolgens hebben we een screenshot van het energieverbruik als de sensor niet meegerekend wordt. Hierop zien we dat een meting(100 keer samplen) ongeveer 0.5mA verbruikt gedurende 10,20ms. Deze tijd kan worden aangepast door bijvoorbeeld het aantal samples te verminderen.

Op onderstaande figuur zien we de werking van de acmp die kort een enkele meting doet. Dit zorgt voor een stroom piekje van 61,53µA wat dus zeer miniem is.

Stroomverbruik besluit

Door het aanpassen van de hoeveelheid samples en de tijd tussen de samples kan het stroomverbruik nog verlaagt worden. Het gemiddelde stroomverbruik is 1,15mA, wat een vermogen geeft van 43,76µWh. Het stroomverbruik is vrij hoog voor een lowpower toepassing. Het stroomverbruik van de sensor domineert het globale stroomverbruik. De Pearl Gecko werkt wel zeer lowpower omdat deze het grootste deel van de tijd in em2 zit.

Uitbreidingen

Het stroomverbruik heeft nog veel ruimte om gereduceerd te worden. De opamp op de sensor kan worden vervangen door een model dat nog steeds geschikt is voor het systeem maar met een lager stroomverbruik. Een andere mogelijkheid is om de microfoon zelf te veranderen naar een model dat minder verbruikt. De sensor kan natuurlijk ook volledig vervangen worden door een energiezuiniger model bijvoorbeeld een MEMS sensor.

Een mogelijke aanpassing in de code zodat de sensor ingeschakeld wordt net voor het uitlezen van de sensor met de ADC. Hierdoor kan de ACMP niet meer gebruikt worden omdat deze verreist dat de sensor altijd actief is.

Referenties

[1] Sparkfun, „https://learn.sparkfun.com,” [Online]. Available: https://learn.sparkfun.com/tutorials/sound-detector-hookup-guide?_ga=2.148141620.2017922931.1576501816-1469302152.1551103145.