Lab: Embedded system design 1

Thomas Feys Jona Cappelle

16 december 2019

Inhoudsopgave

1 Inleiding **3**

2 Sensor **3**

3 Systeem **3**
3.1 Overzicht 3
3.2 I2C 3

4 Code **3**
4.1 Functionaliteiten 3
4.2 Flow van de code 4

5 Power consumptie **4**
5.1 Sensor 4
5.2 Principe 5

6 Besluit **8**

1 Inleiding

Ons doel is om de wachtrij in de rabotaria te monitoren. Dit zullen we doen aan de hand van een IR grid sensor (AMG8833). Aan de hand van de uitgelezen waarden zullen we in het tweede semester bepalen hoeveel mensen er in de wachtrij staan. Deze informatie zullen we vervolgens via een app of website verspreiden.

2 Sensor

De sensor die we gebruiken om de wachtrij te monitoren is de AMG8833. Deze IR grid sensor heeft 8x8 pixels die de temperatuur weergeven. Volgens de datasheet kan a.d.h.v. de temperatuur een mens waargenomen worden vanop een afstand van 5 meter. De sensor kan gebruikt worden in 3 verschillende modes: normal, sleep en stand-by. In de normale mode heeft de sensor een verbruik van 4.5 mA. De stand-by mode heeft twee opties; de waardes kunnen geüpdatet worden om de 60 seconden of om de 10 seconden. In deze mode is er een verbruik van 0.8 mA. Als laatste is er de sleep mode deze verbruikt 0.2 mA. Een overzicht van alle modes en de commando's die verstuurd moeten worden om in deze modes te gaan, wordt weergegeven in figuur 1. Tijdens het testen van de sensor hebben we periodiek geswitched tussen de verschillende power modes. Het resultaat van deze test is te zien in figuur 2. De sensor heeft ook een interrupt pin. Deze pin geneert een interrupt als een van de pixels over of onder een bepaalde waarde gaat. Deze waarde is instelbaar.

3 Systeem

3.1 Overzicht

De AMG8833 communiceert met de EFM32 via I2C. Naast de I2C communicatie is er ook een interrupt pin voorzien op de sensor. Deze pin genereert een interrupt als er een van de pixels een bepaalde, instelbare waarde overschrijdt. Een volledig overzicht van het systeem is te zien in figuur 3.

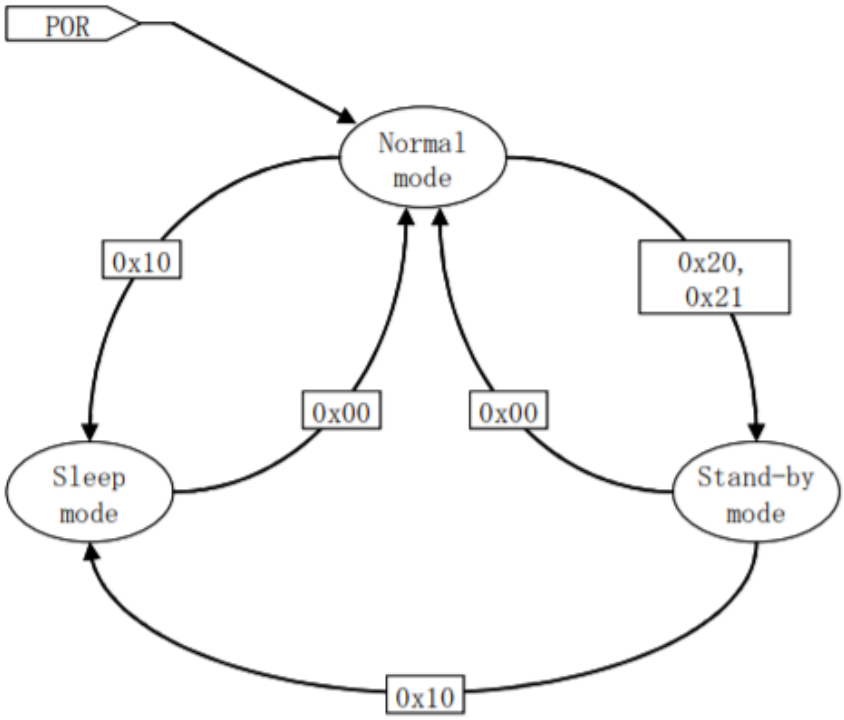
3.2 I2C

De communicatie met de sensor gebeurt via I2C. Het adres van de sensor is een 7 bit adres namelijk: 1101001. Aangezien we met een 7 bit adres zitten moet deze 1 bit geshift worden naar links. Dit adres moet telkens naar de sensor verstuurd worden vooraleer er iets gelezen of geschreven kan worden. De code die gebruikt wordt om de sensor uit te lezen, in andere powermodus te zetten en te interrupten wordt in de volgende sectie besproken.

4 Code

4.1 Functionaliteiten

Om gemakkelijk met de sensor te werken werd er een library geschreven voor de AMG8833. Er werden verschillende methodes geschreven om vlot te kunnen omgaan met de sensor. Er



Figuur 1: Overzicht operating modes

werd een functie geschreven om alle pixels uit te lezen. Naast de pixels is er ook een thermistor aanwezig in de sensor, ook hiervoor werd een functie geschreven. Er werden ook verschillende functies geschreven om makkelijk tussen de verschillende powermodes te kunnen wisselen. Een volledig overzicht van deze library en de rest van de code is terug te vinden op github: `https://github.com/jonacappelle/Embedded_Project`.

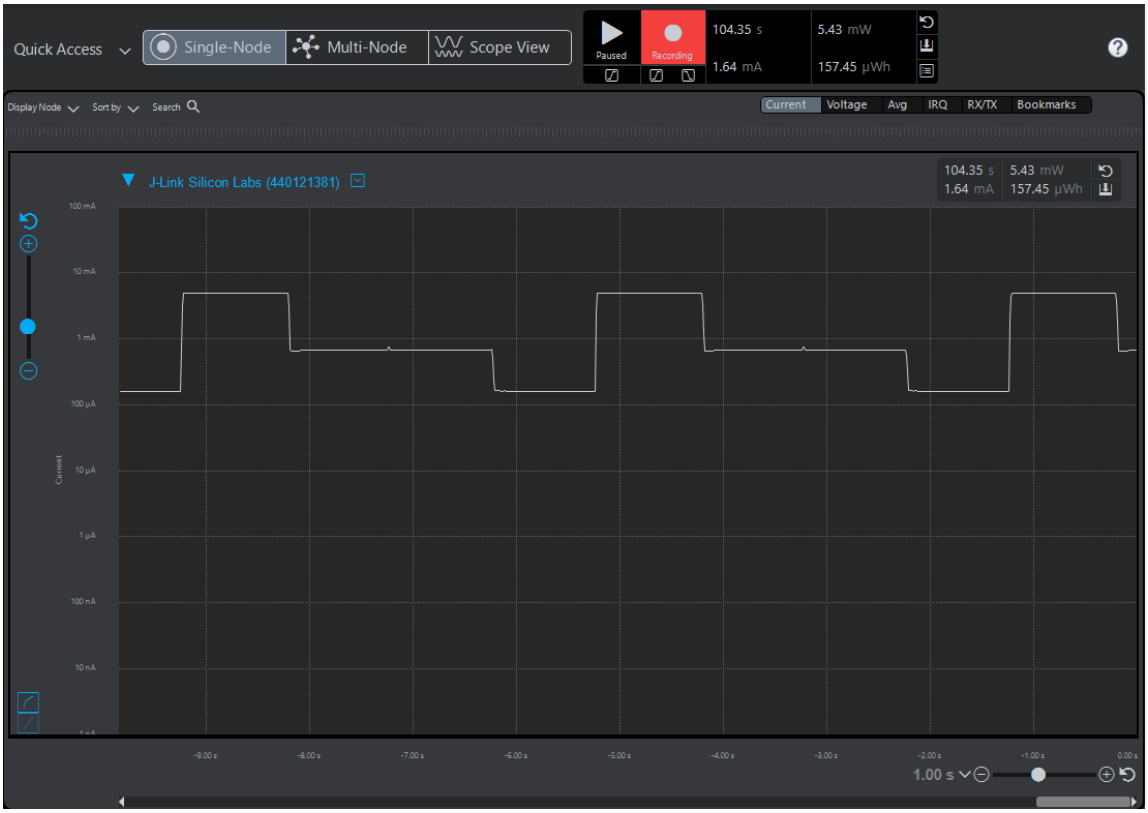
4.2 Flow van de code

Aangezien de sensor in stand-by een verbruik van 0.8 mA heeft wordt deze altijd in sleep gezet. De sensor wordt om de 5 minuten uit sleep gehaald om alle pixels uit te lezen. Hierna wordt ze weer in sleep gezet. Een volledig overzicht van de code is terug te vinden in figuur 4.

5 Power consumptie

5.1 Sensor

De sensor heeft 3 verschillende powermodes, deze zijn te zien in figuur 5. Deze hebben we ook zelf opgemeten zoals te zien is in figuur 2. De gemeten waarden liggen in dezelfde grote orde als deze die opgegeven zijn in de datasheet, maar wijken toch een klein beetje af. In normal mode gebruikt



Figuur 2: Test van de verschillende powermodes

onze sensor 4.95 mA, in stand-by 0.66 mA en in sleep 0.156 mA.

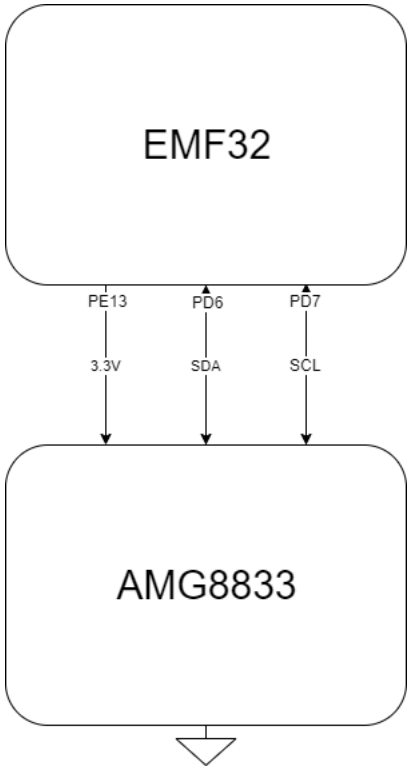
5.2 Principe

Sensor: Door de methode toe te passen die besproken werd in section 4.2 wordt de power consumptie sterk verminderd. De sensor zit enkel in stand-by tijdens het uilezen van de sensor waarde. De sensor zit ook 105 ms in normal mode, omdat je altijd naar de normal mode moet gaan als je uit sleep komt. Er werd een meting uitgevoerd om dit verbruik te controleren, dit is terug te vinden in figuur 6. Uit deze meting is te zien dat de sensor 105 ms in normal mode zit met een verbruik van 4.8 mA, 50ms in stand-by met een verbruik van 2.83 mA[1] en verder zit de sensor in sleep met een verbruik van 0.156 mA. Hierdoor bekomen we volgend verbruik per meting:

$$3.3\ \text{V} \cdot \left(105\ \text{ms} \cdot 4.8\ \text{mA} + 50\ \text{ms} \cdot 2.83\ \text{mA} + 299.845\ \text{s} \cdot 0.156\ \text{mA}\right) = 156.4904\ \text{mJ/meting} \quad (1)$$

Een overzicht is te zien in tabel 1.

[1]Tijdens het uitlezen van de sensor ligt het verbruik hoger, hierdoor verbruikt de sensor 2.83 mA in stand-by ipv 0.8 mA



Figuur 3: Overzicht van het systeem

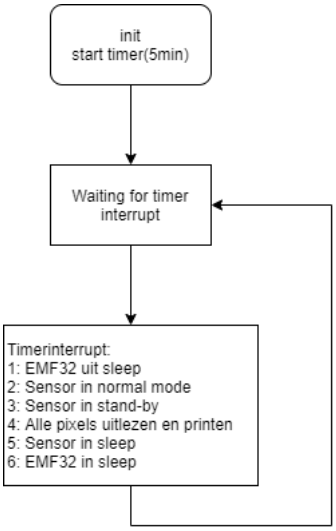
Power Mode	Verbruik [mA]	Tijd [s]	Energie [mJ]
Normal	4.8	0.105	1.6632
Stand-By	2.83	50	0.4669
Sleep	0.156	299.845	154.3602
		Totaal	156.4904

Tabel 1: Verbruik voor één meting

Als we om de 5 minuten een meting uitvoeren, bekomen we volgend verbruik:

$$12 \cdot 156.4904 \frac{\text{mJ}}{\text{meting}} = 1.877\,\text{mJ}$$
$$\Rightarrow \frac{1.877\ \text{J}}{3600\ \text{s}} = 0.52\,\text{mW}$$

In plaats van een constant verbruik van 0.8 mA verbrverbruikt de sensor maar 0.2 mA aangezien hij bijna constant in sleep zit. Het verbruik zal een klein beetje hoger zijn aangezien de sensor voor een kleine tijd weer in normal en stand-by mode gezet wordt.



Figuur 4: flowchart van de code

Energy Mode	Current
Normal	4.5 mA
Stand-by	0.8 mA
Sleep	0.2 mA

Figuur 5: powermodes van de sensor

Processor: De processor wordt enkel uit sleep gehaald als er een timerinterrupt afgaat. Hierdoor zit de processor voornamelijk in EM2 dit geeft een verbruik van 0.9 μA.

Figuur 6: Meting verbruik

6 Besluit

Bijlage

```
1 /**************************************************************************//**
2  * @file main.c
3  * @brief Code to read AMG8833
4  * @details Project for Lab Embedded Systems 1
5  * @version 1.0
6  * @author Jona Cappelle & Thomas Feys
7  * ******************************************************************************
8  *
9  *****************************************************************************/
10
11 /* System Includes */
12 #include "em_device.h"
13 #include "em_chip.h"       /* Needed to initialize the chip */
14 #include "em_cmu.h"
15 #include "em_emu.h"        /* Nedded to use different energy modes, EM0 - EM1 - EM2 - EM
16 #include "em_gpio.h"       /* Needed to use GPIO pins */
17 #include "em_rtc.h"        /* Needed to use RTC timers to generate low power interrupts
18 #include "em_core.h"       /* Needed to use core functionality */
19 #include "em_core.h"
20 #include "rtcdriver.h"
21 #include "em_wdog.h"
22
23 /* Self-written libraries */
24 #include "AMG8833.h"       /* Include our self-written AMG8833 class to interface with t
25 #include "I2C.h"          /* DRAMCO inspired I2C read - write - readwrite library */
26 #include "em_i2c.h"
27 #include "i2cspm.h"        /* I2C higher level library to easily setup and use I2C */
28
29 /* Fescron dbprint include */
30 #include "debug_dbprint.h"     /* Fescron dbprint arduino like library */
31 #include "em_usart.h"      /* Needed to use USART functionality for dbprint */
32
33
34 /* Switch - cases makes the code more user friendly */
35 typedef enum app_states{
36   INIT,
37   MEASURE,
38   SLEEP
39 } APP_State_t;
40
41 static volatile APP_State_t appState;
42 RTCDRV_TimerID_t xTimerForWakeUp;
43 WDOG_Init_TypeDef wInit = WDOG_INIT_DEFAULT;
44
```

```
45 float rBuffer_Thermistor[1];
46 float rBuffer_Pixels[64];
47
48 volatile uint32_t msTickCount;        /* Counts 1ms time ticks */
49 static volatile uint32_t msTicks; /* counts 1ms timeTicks */
50
51
52 /***************************************************************************//**
53  * @brief
54  *    SysTick_Handler needed for timer functionality
55  *
56  ******************************************************************************/
57 void SysTick_Handler(void)
58 {
59     msTicks++;        /* increment counter necessary in Delay()*/
60 }
61
62 /***************************************************************************//**
63  * @brief SysTick_Disable
64  * Disable systick interrupts
65  ******************************************************************************/
66 static void SysTickDisable(void)
67 {
68   SysTick->CTRL = 0x0000000;
69 }
70
71 /***************************************************************************//**
72  * @brief
73  *    Configure the SysTick to use cmuClock_CORE for 1 ms interrupts
74  *
75  ******************************************************************************/
76 void delay_init(void)
77 {
78     /* Setup SysTick Timer for 1 msec interrupts  */
79     if (SysTick_Config(CMU_ClockFreqGet(cmuClock_CORE) / 1000)) {
80       while (1) ;
81     }
82 }
83
84 /***************************************************************************//**
85  * @brief
86  *    Non-low-power delay, but non-blocking
87  *
88  * @param[in] dlyTicks
89  *    How many milli seconds delay
90  *
```

```
91  ******************************************************************************/
92 void delay(uint32_t dlyTicks)
93 {
94     uint32_t curTicks;
95
96     curTicks = msTicks;
97     while ((msTicks - curTicks) < dlyTicks) ;
98 }
99
100 /***************************************************************************//**
101  * @brief
102  *    Callback function for 5 min timer
103  *
104  ******************************************************************************/
105 void Callback_Sleep( void )
106 {
107   appState = MEASURE;
108 }
109
110
111 /***************************************************************************//**
112  * @brief GPIO Even IRQ for pushbuttons on even-numbered pins
113  ******************************************************************************/
114 void GPIO_EVEN_IRQHandler(void)
115 {
116   // Clear all even pin interrupt flags
117   GPIO_IntClear(0x5555);
118   appState = MEASURE;
119
120 #if DEBUG_DBPRINT == 1 /* DEBUG_DBPRINT */
121   dbprint("Interrupt fired! 1");
122 #endif /* DEBUG_DBPRINT */
123 }
124
125 /***************************************************************************//**
126  * @brief GPIO Odd IRQ for pushbuttons on odd-numbered pins
127  ******************************************************************************/
128 void GPIO_ODD_IRQHandler(void)
129 {
130   // Clear all odd pin interrupt flags
131   GPIO_IntClear(0xAAAA);
132   appState = MEASURE;
133
134 #if DEBUG_DBPRINT == 1 /* DEBUG_DBPRINT */
135   dbprint("Interrupt fired! 2");
136 #endif /* DEBUG_DBPRINT */
```

```
137 }
138
139 /**************************************************************************//**
140  * @brief
141  *   Main function
142  *
143  * @details
144  *  We used a state machine to control the states where our microcontroller is in
145  *
146  *****************************************************************************/
147 int main(void)
148 {
149   /* Infinite loop */
150   while (1)
151   {
152     switch(appState)
153     {
154     case INIT:
155     {
156       /* Chip errata */
157         CHIP_Init();
158
159         /* Enable GPIO clocks */
160         CMU_ClockEnable(cmuClock_GPIO, true);
161         CMU_ClockEnable(cmuClock_HFPER, true); // Needed to use GPIO
162         CMU_ClockEnable(cmuClock_CORE, true);
163
164         /* Watchdog setup - Use defaults, excepts for these :*/
165         wInit.em2Run = true;
166         wInit.em3Run = true;
167         wInit.perSel = wdogPeriod_4k; /* 4k 1kHz periods should give ~4 seconds in EM3
168
169         /* Initialize RTC timer. */
170         RTCDRV_Init();
171         RTCDRV_AllocateTimer( &xTimerForWakeUp);
172
173         /* Start I2C */
174         IIC_Init();
175
176         /* Start AMG8833 Temp sensor */
177         AMG8833_Init();
178
179         /* Start Delay */
180         delay_init();
181
182         /* Setup printing to virtual COM port */
```

```
183         #if DEBUG_DBPRINT == 1 /* DEBUG_DBPRINT */
184           dbprint_INIT(USART1, 4, true, false);
185         #endif /* DEBUG_DBPRINT */
186
187           appState = MEASURE;
188       } break;
189       case MEASURE:
190       {
191         /* Set to StandBy 10 sec */
192         AMG8833_Sleep( false );
193         AMG8833_StandBy(STBY_60);
194
195         RTCDRV_StartTimer( xTimerForWakeUp, rtcdrvTimerTypeOneshot, 105, NULL, NULL);
196         EMU_EnterEM2(true);
197
198         //AMG8833_Thermistor_Read( rBuffer_Thermistor );
199         AMG8833_Pixels_Read( rBuffer_Pixels );
200
201         appState = SLEEP;
202
203 #if DEBUG_DBPRINT == 1 /* DEBUG_DBPRINT */
204         AMG8833_Pixel_Print(rBuffer_Pixels);
205 #endif /* DEBUG_DBPRINT */
206
207         appState = SLEEP;
208       } break;
209       case SLEEP:
210       {
211         /* Sensor to sleep --> 0.2 mA */
212         AMG8833_Sleep( true );
213
214         /* EFM to sleep */
215         SysTickDisable();
216         RTCDRV_StartTimer( xTimerForWakeUp, rtcdrvTimerTypeOneshot, 5000, Callback_Sleep,
217         EMU_EnterEM2(true);
218       } break;
219       }
220     }
221 }
```

../Embedded_1_AMG8833_Temp_Sensor/src/main.c

```
1 /*
2  * AGM8833.h
3  *
4  *  Created on: Nov 18, 2019
5  *      Author: jonac
6  */
7
8 /**************************************************************************//**
9  * @file AMG8833.h
10 * @brief Interface with AMG8833 IR Temp camera
11 * @version 1.0
12 * @author Jona Cappelle & Thomas Feys
13 * *************************************************************************/
14
15 #ifndef _AMG8833_H_
16 #define _AMG8833_H_
17
18 /* Needed to use uintx_t */
19 #include <stdint.h>                     /* Needed to use uintx_t */
20 #include <stdbool.h>                    /* Needed to use booleans */
21
22 #define I2C_ADDRESS                       ( 0x69 << 1 ) /* MCU works with 10bit I2C addre
23
24 #define POWER_CONTROL           0x00      /* Power control address */
25 #define POWER_CONTROL_NORMAL      0x00      /* Power control set to normal mode */
26 #define POWER_CONTROL_SLEEP       0x10      /* Power control set to sleep mode */
27 #define POWER_CONTROL_STBY_60     0x20      /* Power control set to standby, measure ev
28 #define POWER_CONTROL_STBY_10     0x21      /* Power control set to standby, measure ev
29
30 #define AMG8833_POWER_PORT        gpioPortE   /* Power Port */
31 #define AMG8833_POWER_PIN       13        /* Power Pin */
32
33 #define AMG8833_INTERRUPT_PORT      gpioPortD   /* Interrupt Port */
34 #define AMG8833_INTERRUPT_PIN     4       /* Interrupt Pin */
35 #define AMG8833_INT_CONTROL_REGISTER  0x03      /* Interrupt control register */
36 #define AMG8833_INT_LEVEL_REGISTER    0x08      /* Interrupt level register (upper limi
37
38 #define RESET               0x01      /* Reset register */
39
40 #define THERMISTOR_OUTPUT_VALUE_L   0x0E      /* Begin address of thermistor output */
41
42 #define PIXEL_1_L           0x80      /* Begin address of IR pixel array (128 registers
43
44 #define THERMISTOR_RES          0.0625      /* Constant: thermistor resolution */
45 #define PIXEL_RES           0.25      /* Constant: pixel resolution */
46
```

```
47 #define THRESHOLD_VALUE          20        /* Threshold value for interrupt */
48
49 /* SWITCH CASES */
50 #define STBY_60      0x00              /* Register value to set stand-by mode to 60 sec
51 #define STBY_10      0x01              /* Register value to set stand-by mode to 10 sec
52
53 void AMG8833_Init( void );
54 void AMG8833_Thermistor_Read(float *rBuffer_Thermistor);
55 void AMG8833_Pixels_Read(float *rBuffer_Pixels);
56 void AMG8833_Sleep(bool enable);
57 void AMG8833_StandBy(uint8_t time);
58 void AMG8833_Reset(void);
59 void AMG8833_Power(bool enable);
60 void AMG8833_Interrupt(bool enable);
61
62 #endif
```

../Embedded_1_AMG8833_Temp_Sensor/AMG8833/AMG8833.h

```c
/*
 * AGM8833.c
 *
 *  Created on: Nov 18, 2019
 *      Author: jonac
 */

/**************************************************************************//**
 * @file AMG8833.c
 * @brief Interface with AMG8833 IR Temp camera
 * @version 1.0
 * @author Jona Cappelle & Thomas Feys
 * *************************************************************************/


#include "AMG8833.h"
#include "I2C.h"

/* dbprint include */
#include "debug_dbprint.h"
#include "em_usart.h"
#include "em_gpio.h"

/* Needed to use uintx_t */
#include <stdint.h>
#include <stdbool.h>


/**************************************************************************//**
 * @brief
 *   Setup AMG8833 Temperature sensor
 *****************************************************************************/
void AMG8833_Init( void )
{
  /* Set the pin high to supply power to the AMG8833 */
  GPIO_PinModeSet(AMG8833_POWER_PORT, AMG8833_POWER_PIN, gpioModePushPull, 1);
}

/**************************************************************************//**
 * @brief
 *   Read the thermistor value of the AMG8833 IR sensor
 *   The data is processed as described in the datasheet
 *
 * @param[in] rBuffer_Thermistor
 *   Address where the read thermistor value is stored
 *
```

```
47  ***************************************************************************/
48 void AMG8833_Thermistor_Read(float *rBuffer_Thermistor)
49 {
50   uint8_t rBuffer[2];
51   uint8_t wBuffer[2];
52   wBuffer[0] = THERMISTOR_OUTPUT_VALUE_L;
53   wBuffer[1] = 0x00;
54 
55   IIC_WriteReadBuffer(I2C_ADDRESS, wBuffer, 1, rBuffer, 2);
56 
57   *rBuffer_Thermistor = ( ( rBuffer[1] << 8 ) | rBuffer[0] ) * THERMISTOR_RES;
58 }
59 
60 
61 /***************************************************************************//**
62  * @brief
63  *   Read the value of all 64 IR pixels
64  *   The data is processed as described in the datasheet
65  *
66  * @param[in] rBuffer_Pixels
67  *   Begin address where the read values of the pixel array is stored
68  *
69  ***************************************************************************/
70 void AMG8833_Pixels_Read(float *rBuffer_Pixels)
71 {
72   uint8_t rBuffer[128];
73   uint8_t wBuffer[2];
74   wBuffer[0] = PIXEL_1_L;
75   wBuffer[1] = 0x00;
76 
77   IIC_WriteReadBuffer(I2C_ADDRESS, wBuffer, 1, rBuffer, 128);
78 
79   for (int i=0; i<64; i++)
80   {
81     rBuffer_Pixels[i] = ( ( rBuffer[2*i+1] << 8 ) | rBuffer[2*i] ) * PIXEL_RES;
82 
83   }
84 }
85 
86 
87 /***************************************************************************//**
88  * @brief
89  *   Print all the values of the IR pixel array
90  *
91  * @param[in] rBuffer_Pixels
92  *   Address where the read thermistor value is stored
```

```
 93  *
 94  ***************************************************************************/
 95 void AMG8833_Pixel_Print(float *rBuffer_Pixels)
 96 {
 97 #if DEBUG_DBPRINT == 1 /* DEBUG_DBPRINT */
 98
 99   for (int j=0; j<8; j++)
100   {
101     for (int i=0; i<8; i++)
102     {
103       dbprintInt( (int) rBuffer_Pixels[j*8+i] );
104       dbprint(" ");
105     }
106     dbprintln("");
107   }
108   dbprintln("-----------------------------------------------------------------");
109
110 #endif /* DEBUG_DBPRINT */
111 }
112
113
114 /***************************************************************************//**
115  * @brief
116  *    Function to set the AMG8833 to sleep
117  *
118  *@details
119  *   Power consumption sleep: 0.2 mA
120  *
121  * @note
122  *   In sleep mode, nothing can be read from the registers
123  *
124  * @param[in] enable
125  *    @li 'true' - sleep mode
126  *    @li 'false' - normal mode
127  *
128  ***************************************************************************/
129 void AMG8833_Sleep(bool enable)
130 {
131   uint8_t rBuffer[1];
132   uint8_t wBuffer[2];
133   wBuffer[0] = POWER_CONTROL;
134
135   if(enable)
136   {
137     wBuffer[1] = POWER_CONTROL_SLEEP;
138   }else{
```

```
139       wBuffer[1] = POWER_CONTROL_NORMAL;
140     }
141     IIC_WriteBuffer(I2C_ADDRESS, wBuffer, 2);
142
143 }
144
145
146 /**************************************************************************//**
147  * @brief
148  *    Set AGM8833 in stand by
149  *
150  * @details
151  *   Power consumption stand-by: 0.8 mA
152  *
153  * @note
154  *    Can't go from SLEEP to STAND-BY!
155  *
156  * @param[in] time
157  *    @li 'STBY_10' - Sets interval to 10sec
158  *    @li 'STBY_60' - Sets interval to 60sec
159  *****************************************************************************/
160 void AMG8833_StandBy(uint8_t time)
161 {
162   uint8_t wBuffer[2];
163   wBuffer[0] = POWER_CONTROL;
164
165   switch(time)
166   {
167   case STBY_10:
168     wBuffer[1] = POWER_CONTROL_STBY_10;
169     break;
170   case STBY_60:
171     wBuffer[1] = POWER_CONTROL_STBY_60;
172     break;
173   }
174
175   IIC_WriteBuffer(I2C_ADDRESS, wBuffer, 2);
176 }
177
178 /**************************************************************************//**
179  * @brief
180  *    Resets the AMG8833 sensor to default settings
181  *
182  *****************************************************************************/
183 void AMG8833_Reset(void)
184 {
```

```
185   //TODO
186 }
187
188 /**************************************************************************//**
189  * @brief
190  *   Supply Power to the AMG8833 VDD pin
191  *
192  * @note
193  *   Needs to be enabled for the sensor to work
194  *
195  * @param[in] enable
196  *   @li 'true' - Enable power
197  *   @li 'false' - Disable power
198  *****************************************************************************/
199 void AMG8833_Power(bool enable)
200 {
201   if(enable)
202   {
203     GPIO_PinOutSet(AMG8833_POWER_PORT, AMG8833_POWER_PIN); /* Enable VDD pin */
204   }else{
205     GPIO_PinOutClear(AMG8833_POWER_PORT, AMG8833_POWER_PIN); /* Disable VDD pin */
206   }
207 }
208
209 /**************************************************************************//**
210  * @brief
211  *   Sensor Interrupt functionality
212  *
213  * @details
214  *  @li 'bit0' - INT output active
215  *  @li 'bit1' - Absolute value interrupt mode
216  *
217  * @param[in] enable
218  *   @li 'true' - Enable interrupts
219  *   @li 'false' - Disable interrupts - other mode
220  *****************************************************************************/
221 void AMG8833_Interrupt(bool enable)
222 {
223   uint8_t rBuffer[1];
224   uint8_t wBuffer[2];
225   wBuffer[0] = AMG8833_INT_CONTROL_REGISTER;
226
227   if(enable)
228   {
229     wBuffer[1] = 0x03;
230   }else
```

```
231   {
232     wBuffer[1] = 0x00;
233   }
234 
235   IIC_WriteReadBuffer(I2C_ADDRESS, wBuffer, 1, rBuffer, 0);
236 }
237 
238 /**************************************************************************//**
239  * @brief
240  *   Sensor Interrupt upper limit set: TODO
241  *
242  * @param[in] value
243  *  Upper limit for interrupt
244  *****************************************************************************/
245 void AMG8833_Set_Interrupt_Upper_Value( int value )
246 {
247   uint8_t rBuffer[1];
248   uint8_t wBuffer[2];
249   wBuffer[0] = AMG8833_INT_LEVEL_REGISTER;
250 
251   uint16_t temp;
252 
253   temp = (uint16_t) (value / PIXEL_RES);
254 
255   wBuffer[1] = temp;
256   //TODO
257 
258   IIC_WriteReadBuffer(I2C_ADDRESS, wBuffer, 1, rBuffer, 0);
259 }
```

../Embedded_1_AMG8833_Temp_Sensor/AMG8833/AMG8833.c