LIS2DH12

We want to use LIS2DH12TR to wake up the battery device on PIC18F24Q10 when it is touched. I made the following circuit for testing, connected to our microcontroller via the I2C interface.



To check I use the following code.

#pragma warning disable 759  //expression generates no code

#pragma warning disable 760  //portion of expression has no effect

#pragma warning disable 520  //function is never called

#pragma warning disable 1498 //pointer (unknown) in expression may have no targets

#pragma warning disable 1510 //non-reentrant function appears in multiple call graphs and has been duplicated by the //compiler

#pragma warning disable 1090 //variable is not used

#pragma warning disable 2053 //function is never called

#include <stdint.h>

#include <stddef.h>

#include <stdbool.h>

#include <string.h>

#include <stdio.h>

#include "mcc\_generated\_files/mcc.h"

#include "lis2dh12\_reg.h"

/\*---------------------------------------------------------------------------\*/

static volatile \_\_persistent uint16\_t delay\_1ms;

static volatile \_\_persistent uint16\_t \_1ms;

static volatile \_\_persistent bool \_1ms\_flag;

void delay\_ms(uint16\_t ms)

{

 delay\_1ms = ms;

 while (delay\_1ms) CLRWDT();

}

void tmr0\_hnd(void)

{ //1000us

 if (delay\_1ms) delay\_1ms--;

 \_1ms++;

 \_1ms\_flag = true;

 ADCON0bits.ADON = 1;

 ADCON0bits.ADGO = 1;

}

void get\_ad(void)

{

 while(ADCON0bits.ADGO);   /\* Wait for AD conversion complete ~37u \*/

 int16\_t res = (int16\_t)ADCC\_GetConversionResult();

 ADCON0bits.ADON = 0;

}

/\*---------------------------------------------------------------------------\*/

static volatile  uint8\_t Int1Flag = 0;

static volatile  uint8\_t Int2Flag = 0;

static volatile  bool interrupted1 = false;

static volatile  bool interrupted2 = false;

void Int1(void)

{

 Int1Flag = 1;

 interrupted1 = true;

}

void Int2(void)

{

 Int2Flag = 1;

 interrupted2 = true;

}

/\*---------------------------------------------------------------------------\*/

/\*

                         Main application

 \*/

static int16\_t data\_raw\_acceleration[3];

static int16\_t acceleration\_mg[3];

static uint8\_t whoamI;

void main(void)

{

 // Initialize the device

 SYSTEM\_Initialize();

 RB0\_SetDigitalInput();

 RB1\_SetDigitalInput();

 RB0\_SetPullup();

 RB1\_SetPullup();

 Vsns\_LAT = 0;

 TMR0\_SetInterruptHandler(tmr0\_hnd);

 ADCC\_SetADIInterruptHandler(get\_ad);

 INT0\_SetInterruptHandler(Int1);

 INT1\_SetInterruptHandler(Int2);

 // If using interrupts in PIC18 High/Low Priority Mode you need to enable the Global High and Low Interrupts

 // If using interrupts in PIC Mid-Range Compatibility Mode you need to enable the Global and Peripheral Interrupts

 // Use the following macros to:

 // Enable the Global Interrupts

 INTERRUPT\_GlobalInterruptEnable();

 // Disable the Global Interrupts

 //INTERRUPT\_GlobalInterruptDisable();

 // Enable the Peripheral Interrupts

 INTERRUPT\_PeripheralInterruptEnable();

 EXT\_INT0\_InterruptDisable();

 printf("\e[2J\e[H"); //Clear screen and home

 printf("Hello!\n\r");

 delay\_ms(10);

 lis2dh12\_device\_id\_get(&whoamI);

 printf("LIS2DH12 ID: %d\n\r", whoamI);

 if (whoamI != LIS2DH12\_ID) printf("Device not found\n\r");

 else

   {//AN5005 6.3.3 Using the HP filter

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_CTRL\_REG0, 0x90); // Disconnect SDO/SA0 pull-up.

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_TEMP\_CFG\_REG, 0x00); //Temperature sensor disabled

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_CTRL\_REG1, 0x3F); // Turn on the sensor and enable X, Y, and Z

                                                           // low-power mode, ODR = 50 Hz

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_CTRL\_REG2, 0x09); // High-pass filter disabled

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_CTRL\_REG3, 0x40); // Interrupt activity 1 driven to INT1 pad

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_CTRL\_REG4, 0x00); // FS = ±2 g

    state\_resolution = LIS2DH12\_LP\_8bit;

    state\_scale = LIS2DH12\_2g;

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_CTRL\_REG5, 0x08); // Interrupt 1 pin latched

    //LIS2DH12\_Write1ByteRegister(LIS2DH12\_INT1\_THS, 0x10);  // Threshold = 250 mg

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_INT1\_THS, 0x8);  // Threshold = 125 mg

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_INT1\_DURATION, 0x00); // Duration = 0

                                                           // Dummy read to force the HP filter to

    LIS2DH12\_Read1ByteRegister(LIS2DH12\_REFERENCE);        // current acceleration value

                                                           // (i.e. set reference acceleration/tilt value)

    LIS2DH12\_Write1ByteRegister(LIS2DH12\_INT1\_CFG, 0x2A);  // Enable XH and YH interrupt generation (2A for XYZ)

    EXT\_INT0\_InterruptEnable();

    while(1)

    {

     lis2dh12\_int1\_src\_t src;

     if (Int1Flag)

      {

       Int1Flag = false;

       src.byte = LIS2DH12\_Read1ByteRegister(LIS2DH12\_INT1\_SRC);

       lis2dh12\_acceleration\_raw\_get(data\_raw\_acceleration);

       acceleration\_mg[0] = lis2dh12\_getAcceleration(data\_raw\_acceleration[0]);

       acceleration\_mg[1] = lis2dh12\_getAcceleration(data\_raw\_acceleration[1]);

       acceleration\_mg[2] = lis2dh12\_getAcceleration(data\_raw\_acceleration[2]);

       printf("\r\nInt1 %02X [mg]:% 5d % 5d % 5d\r\n", src.byte,

           acceleration\_mg[0], acceleration\_mg[1], acceleration\_mg[2]);

      }

     if (\_1ms\_flag)

      {

       if ((\_1ms%1000) == 0) printf(". ");

       \_1ms\_flag = false;

      }

     CLRWDT();

    }

   }

/\*\*

 End of File

 \*/

The code works, I see points on the terminal connected to the UART when no one touches the board, and interruptions when it is touched. But the microammeter connected to the sensor power supply circuit shows 167µA (instead of 20µA as shown in Table 12 of the datasheet). By the way, I see exactly the same current if I remove the pull-up resistors R1 and R2 and turn on the pull-up of the microcontroller SDA and SCA pins. After disabling the internal pull-up at the SA0 pin, reducing the data transfer rate to 25 Hz and the resolution to 8 bits, the current decreased to 5.3 µA that close to 4 µA.

Checking the temperature sensor showed its practical uselessness. The results are extremely unstable, their conversion to degrees is not described, the code for converting them to degrees in the ST source code looks strange, the result of its work strongly does not correspond to the real temperature. Turning on the temperature sensor adds 15..20 µA to the consumption.



