COE 306, Term 171

 Introduction to Embedded Systems

**Assignment# 5 Solution**

**Due date: Saturday, Dec. 23, 2017**

# It is required to write an embedded program to interface with the LSM303D Accelerometer using I2C interface. The accelerometer measures acceleration along the three dimensions, and makes them available in the following registers:

* OUT\_X\_L\_A (28h), OUT\_X\_H\_A (29h)

X-axis acceleration data. The value is expressed in 16 bits as 2’s complement.

* OUT\_Y\_L\_A (2Ah), OUT\_X\_H\_A (2Bh)

Y-axis acceleration data. The value is expressed in 16 bits as 2’s complement.

* OUT\_X\_L\_A (2Ch), OUT\_X\_H\_A (2Dh)

Z-axis acceleration data. The value is expressed in 16 bits as 2’s complement.

## Use the LPC1769’s I2C interface to read the accelerometer data from the LSM303D device. The slave address (SAD) associated to the LSM303D is 00111xxb, whereas the xx bits are modified by the SDO/SA0 pin in order to modify the device address. If the SDO/SA0 pin is connected to the voltage supply, the address is 0011101b, otherwise, if the SDO/SA0 pin is connected to ground, the address is 0011110b. This solution permits the connection and addressing of two different accelerometers to the same I2C lines. Consult the LSM303D datasheet for more details [https://www.pololu.com/file/0J703/LSM303D.pdf].

## Write a simple application to indicate different stationary positions. For example, indicate whether the device is tilted to the right or to the left, tilted forward or backward, and whether it’s facing upward or downward. Use print statements to reflect this data in real-time. The following table summarizes the readings corresponding to each of the six stationary positions.

##

# Include the source code of your solution along with a video snapshot to demonstrate correct functionality of your code.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* $Id:: i2c.h $

 \* Project: NXP LPC11xx I2C example

 \*

 \* Description:

 \* This file contains I2C code header definition.

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* Software that is described herein is for illustrative purposes only

 \* which provides customers with programming information regarding the

 \* products. This software is supplied "AS IS" without any warranties.

 \* NXP Semiconductors assumes no responsibility or liability for the

 \* use of the software, conveys no license or title under any patent,

 \* copyright, or mask work right to the product. NXP Semiconductors

 \* reserves the right to make changes in the software without

 \* notification. NXP Semiconductors also make no representation or

 \* warranty that such application will be suitable for the specified

 \* use without further testing or modification.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#ifndef** \_\_I2C\_H

**#define** \_\_I2C\_H

/\* If I2C SEEPROM is tested, make sure FAST\_MODE\_PLUS is 0.

For board to board test, this flag can be turned on. \*/

**#define** TRUE 1

**#define** FALSE 0

**#define** BUFSIZE 64

**#define** MAX\_TIMEOUT 0x00FFFFFF

**#define** I2CMASTER 0x01

**#define** I2CSLAVE 0x02

**#define** SLAVE\_ADDR 0x1D // slave address 0011101b. SDO/SA0 pin connected to VDD

**#define** I2C\_IDLE 0

**#define** I2C\_STARTED 1

**#define** I2C\_RESTARTED 2

**#define** I2C\_REPEATED\_START 3

**#define** DATA\_ACK 4

**#define** DATA\_NACK 5

**#define** I2C\_BUSY 6

**#define** I2C\_NO\_DATA 7

**#define** I2C\_NACK\_ON\_ADDRESS 8

**#define** I2C\_NACK\_ON\_DATA 9

**#define** I2C\_ARBITRATION\_LOST 10

**#define** I2C\_TIME\_OUT 11

**#define** I2C\_OK 12

**#define** I2CONSET\_I2EN (0x1<<6) /\* I2C Control Set Register \*/

**#define** I2CONSET\_AA (0x1<<2)

**#define** I2CONSET\_SI (0x1<<3)

**#define** I2CONSET\_STO (0x1<<4)

**#define** I2CONSET\_STA (0x1<<5)

**#define** I2CONCLR\_AAC (0x1<<2) /\* I2C Control clear Register \*/

**#define** I2CONCLR\_SIC (0x1<<3)

**#define** I2CONCLR\_STAC (0x1<<5)

**#define** I2CONCLR\_I2ENC (0x1<<6)

**#define** I2SCLH\_SCLH 0x00000019 /\* I2C SCL Duty Cycle High Reg \*/

**#define** I2SCLL\_SCLL 0x00000019 /\* I2C SCL Duty Cycle Low Reg \*/

**#define** CTRL0 0b00000000 //for normal mode, filters by-passed

**#define** CTRL1 0b01001111 //update after read, and all axes of acceleration enabled at 25Hz

**#define** CTRL2 0b11000001 //50Hz anti-alias, +/- 2g, no self-test, (SPI 3-wire)

**#define** CTRL3 0b00000000 //No INT1 actions

**#define** CTRL4 0b00001000 //accelerometer data ready on INT2.

**#define** CTRL5 0b00001110 //No temperature, low-res magnetic, 25Hz, latch interrupt on INT2

**#define** CTRL6 0b00000000 //+/-2gauss sensitivity.

**#define** CTRL7 0b00000000 //normal high-pass acceleration filter, no Temp, magnetic always on, continuous conversion mode

**extern** **void** **I2C0\_IRQHandler**( **void** );

**extern** uint32\_t **I2CInit**( uint32\_t I2cMode );

**extern** uint32\_t **I2CStart**( **void** );

**extern** uint32\_t **I2CStop**( **void** );

**extern** uint32\_t **I2CEngine**( **void** );

**#endif** /\* end \_\_I2C\_H \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* End Of File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* $Id:: i2c.c 4058 2010-07-30 01:03:21Z usb00423 $

 \* Project: NXP LPC11Uxx I2C example

 \*

 \* Description:

 \* This file contains I2C code example which include I2C initialization,

 \* I2C interrupt handler, and APIs for I2C access.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Software that is described herein is for illustrative purposes only

\* which provides customers with programming information regarding the

\* products. This software is supplied "AS IS" without any warranties.

\* NXP Semiconductors assumes no responsibility or liability for the

\* use of the software, conveys no license or title under any patent,

\* copyright, or mask work right to the product. NXP Semiconductors

\* reserves the right to make changes in the software without

\* notification. NXP Semiconductors also make no representation or

\* warranty that such application will be suitable for the specified

\* use without further testing or modification.

\* Permission to use, copy, modify, and distribute this software and its

\* documentation is hereby granted, under NXP Semiconductors'

\* relevant copyright in the software, without fee, provided that it

\* is used in conjunction with NXP Semiconductors microcontrollers. This

\* copyright, permission, and disclaimer notice must appear in all copies of

\* this code.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

#include "i2c.h"

volatile uint32\_t I2CMasterState = I2C\_IDLE;

volatile uint32\_t I2CSlaveState = I2C\_IDLE;

volatile uint32\_t timeout = 0;

volatile uint32\_t I2CMode;

volatile uint8\_t I2CMasterBuffer[BUFSIZE];

volatile uint8\_t I2CSlaveBuffer[BUFSIZE];

volatile uint32\_t I2CCount = 0;

volatile uint32\_t I2CReadLength;

volatile uint32\_t I2CWriteLength;

volatile uint32\_t RdIndex = 0;

volatile uint32\_t WrIndex = 0;

/\*

From device to device, the I2C communication protocol may vary,

in the example below, the protocol uses repeated start to read data from or

write to the device:

For master read: the sequence is: STA,Addr(W),offset,RE-STA,Addr(r),data...STO

for master write: the sequence is: STA,Addr(W),offset,RE-STA,Addr(w),data...STO

Thus, in state 8, the address is always WRITE. in state 10, the address could

be READ or WRITE depending on the I2C command.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name: I2C\_IRQHandler

\*\*

\*\* Descriptions: I2C interrupt handler, deal with master mode only.

\*\*

\*\* parameters: None

\*\* Returned value: None

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C0\_IRQHandler(void)

{

 uint8\_t StatValue;

 timeout = 0;

 /\* this handler deals with master read and master write only \*/

 StatValue = LPC\_I2C0->I2STAT;

 //printf("State value =%x \n", StatValue);

 switch ( StatValue )

 {

 case 0x08: /\* A Start condition is issued. \*/

 WrIndex = 0;

 RdIndex = 0;

 LPC\_I2C0->I2DAT = I2CMasterBuffer[WrIndex++];

 LPC\_I2C0->I2CONCLR = (I2CONCLR\_SIC | I2CONCLR\_STAC);

 break;

 case 0x10: /\* A repeated started is issued \*/

 RdIndex = 0;

 /\* Send SLA with R bit set, \*/

 LPC\_I2C0->I2DAT = I2CMasterBuffer[WrIndex++];

 LPC\_I2C0->I2CONCLR = (I2CONCLR\_SIC | I2CONCLR\_STAC);

 break;

 case 0x18: /\* Regardless, it's a ACK \*/

 if ( I2CWriteLength == 1 )

 {

 LPC\_I2C0->I2CONSET = I2CONSET\_STO; /\* Set Stop flag \*/

 I2CMasterState = I2C\_NO\_DATA;

 }

 else

 {

 LPC\_I2C0->I2DAT = I2CMasterBuffer[WrIndex++];

 }

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC;

 break;

 case 0x28: /\* Data byte has been transmitted, regardless ACK or NACK \*/

 if ( WrIndex < I2CWriteLength )

 {

 LPC\_I2C0->I2DAT = I2CMasterBuffer[WrIndex++]; /\* this should be the last one \*/

 }

 else

 {

 if ( I2CReadLength != 0 )

 {

 LPC\_I2C0->I2CONSET = I2CONSET\_STA; /\* Set Repeated-start flag \*/

 }

 else

 {

 LPC\_I2C0->I2CONSET = I2CONSET\_STO; /\* Set Stop flag \*/

 I2CMasterState = I2C\_OK;

 }

 }

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC;

 break;

 case 0x30:

 LPC\_I2C0->I2CONSET = I2CONSET\_STO; /\* Set Stop flag \*/

 I2CMasterState = I2C\_NACK\_ON\_DATA;

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC;

 break;

 case 0x40: /\* Master Receive, SLA\_R has been sent \*/

 if ( (RdIndex + 1) < I2CReadLength )

 {

 /\* Will go to State 0x50 \*/

 LPC\_I2C0->I2CONSET = I2CONSET\_AA; /\* assert ACK after data is received \*/

 }

 else

 {

 /\* Will go to State 0x58 \*/

 LPC\_I2C0->I2CONCLR = I2CONCLR\_AAC; /\* assert NACK after data is received \*/

 }

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC;

 break;

 case 0x50: /\* Data byte has been received, regardless following ACK or NACK \*/

 I2CSlaveBuffer[RdIndex++] = LPC\_I2C0->I2DAT;

 if ( (RdIndex + 1) < I2CReadLength )

 {

 LPC\_I2C0->I2CONSET = I2CONSET\_AA; /\* assert ACK after data is received \*/

 }

 else

 {

 LPC\_I2C0->I2CONCLR = I2CONCLR\_AAC; /\* assert NACK on last byte \*/

 }

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC;

 break;

 case 0x58:

 I2CSlaveBuffer[RdIndex++] = LPC\_I2C0->I2DAT;

 I2CMasterState = I2C\_OK;

 LPC\_I2C0->I2CONSET = I2CONSET\_STO; /\* Set Stop flag \*/

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC; /\* Clear SI flag \*/

 break;

 case 0x20: /\* regardless, it's a NACK \*/

 case 0x48:

 LPC\_I2C0->I2CONSET = I2CONSET\_STO; /\* Set Stop flag \*/

 I2CMasterState = I2C\_NACK\_ON\_ADDRESS;

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC;

 break;

 case 0x38: /\* Arbitration lost, in this example, we don't

 deal with multiple master situation \*/

 default:

 I2CMasterState = I2C\_ARBITRATION\_LOST;

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC;

 break;

 }

 return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name: I2CStart

\*\*

\*\* Descriptions: Create I2C start condition, a timeout

\*\* value is set if the I2C never gets started,

\*\* and timed out. It's a fatal error.

\*\*

\*\* parameters: None

\*\* Returned value: true or false, return false if timed out

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint32\_t I2CStart( void )

{

 uint32\_t timeout = 0;

 uint32\_t retVal = FALSE;

 /\*--- Issue a start condition ---\*/

 LPC\_I2C0->I2CONSET = I2CONSET\_STA; /\* Set Start flag \*/

 /\*--- Wait until START transmitted ---\*/

 while( 1 )

 {

 if ( I2CMasterState == I2C\_STARTED )

 {

 retVal = TRUE;

 break;

 }

 if ( timeout >= MAX\_TIMEOUT )

 {

 retVal = FALSE;

 break;

 }

 timeout++;

 }

 return( retVal );

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name: I2CStop

\*\*

\*\* Descriptions: Set the I2C stop condition, if the routine

\*\* never exit, it's a fatal bus error.

\*\*

\*\* parameters: None

\*\* Returned value: true or never return

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint32\_t I2CStop( void )

{

 LPC\_I2C0->I2CONSET = I2CONSET\_STO; /\* Set Stop flag \*/

 LPC\_I2C0->I2CONCLR = I2CONCLR\_SIC; /\* Clear SI flag \*/

 /\*--- Wait for STOP detected ---\*/

 while( LPC\_I2C0->I2CONSET & I2CONSET\_STO );

 return TRUE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name: I2CInit

\*\*

\*\* Descriptions: Initialize I2C controller

\*\*

\*\* parameters: I2c mode is either MASTER or SLAVE

\*\* Returned value: true or false, return false if the I2C

\*\* interrupt handler was not installed correctly

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint32\_t I2CInit( uint32\_t I2cMode )

{

 LPC\_SC -> PCONP |= 1 << 7; //I2C0 interface power/clock control bit.

 //Configure P0.28 to I2C CLK pin SCL0

 LPC\_PINCON -> PINSEL1 |= 1 << 24;

 //Configure P0.27 to data pin SDA0

 LPC\_PINCON -> PINSEL1 |= 1 << 22;

 /\*--- Clear flags ---\*/

 LPC\_I2C0->I2CONCLR = I2CONCLR\_AAC | I2CONCLR\_SIC | I2CONCLR\_STAC | I2CONCLR\_I2ENC;

 // initialize clock registers

 LPC\_I2C0->I2SCLL = I2SCLL\_SCLL;

 LPC\_I2C0->I2SCLH = I2SCLH\_SCLH;

 /\* Enable the I2C Interrupt \*/

 NVIC\_EnableIRQ(I2C0\_IRQn);

 // Enable I2C

 LPC\_I2C0->I2CONSET = I2CONSET\_I2EN;

 return( TRUE );

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name: I2CEngine

\*\*

\*\* Descriptions: The routine to complete a I2C transaction

\*\* from start to stop. All the intermitten

\*\* steps are handled in the interrupt handler.

\*\* Before this routine is called, the read

\*\* length, write length, I2C master buffer,

\*\* and I2C command fields need to be filled.

\*\* see i2cmst.c for more details.

\*\*

\*\* parameters: None

\*\* Returned value: true or false, return false only if the

\*\* start condition can never be generated and

\*\* timed out.

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint32\_t I2CEngine( void )

{

 RdIndex = 0;

 WrIndex = 0;

 /\*--- Issue a start condition ---\*/

 LPC\_I2C0->I2CONSET = I2CONSET\_STA; /\* Set Start flag \*/

 I2CMasterState = I2C\_BUSY;

 while ( I2CMasterState == I2C\_BUSY )

 {

 if ( timeout >= MAX\_TIMEOUT )

 {

 I2CMasterState = I2C\_TIME\_OUT;

 break;

 }

 timeout++;

 }

 LPC\_I2C0->I2CONCLR = I2CONCLR\_STAC;

 return ( I2CMasterState );

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* End Of File

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

===============================================================================

 Name : Ass5i2c.c

 Author : $(author)

 Version :

 Copyright : $(copyright)

 Description : main definition

=====================================================================\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

#include "i2c.h"

extern volatile uint32\_t I2CCount;

extern volatile uint8\_t I2CMasterBuffer[BUFSIZE];

extern volatile uint8\_t I2CSlaveBuffer[BUFSIZE];

extern volatile uint32\_t I2CMasterState;

extern volatile uint32\_t I2CReadLength, I2CWriteLength;

int main(void) {

 // TODO: insert code here

 // Force the counter to be placed into memory

 volatile static int i = 0 ;

 // Enter an infinite loop, just incrementing a counter

 int16\_t accX, accY, accZ;

 int16\_t absX, absY, absZ;

 if ( I2CInit( (uint32\_t)I2CMASTER ) == FALSE ) /\* initialize I2c \*/

 {

 printf("Error in initializing I2C"); /\* Fatal error \*/

 }

 /\* Writing the 8 control registers \*/

 I2CWriteLength = 10;

 I2CReadLength = 0;

 I2CMasterBuffer[0] = SLAVE\_ADDR << 1;

 I2CMasterBuffer[1] = 0x1F |0x80; /\* CNTR0 register with bit 7 set for address auto increment \*/

 I2CMasterBuffer[2] = CTRL0;

 I2CMasterBuffer[3] = CTRL1;

 I2CMasterBuffer[4] = CTRL2;

 I2CMasterBuffer[5] = CTRL3;

 I2CMasterBuffer[6] = CTRL4;

 I2CMasterBuffer[7] = CTRL5;

 I2CMasterBuffer[8] = CTRL6;

 I2CMasterBuffer[9] = CTRL7;

 I2CEngine();

 /\* Reading the 8 control registers \*/

 I2CWriteLength = 2;

 I2CReadLength = 8;

 I2CMasterBuffer[0] = SLAVE\_ADDR << 1;

 I2CMasterBuffer[1] = 0x1F |0x80; /\* CNTR0 register with bit 7 set for address auto increment \*/

 I2CMasterBuffer[2] = SLAVE\_ADDR << 1|1; // set read bit

 I2CEngine();

 for (int i=0; i<8; i++){

 printf("CNT %d = %x \n", i, I2CSlaveBuffer[i]);

 }

 while(1) {

 for(int i=0;i<10000000;i++);

 /\* Reading the 6 acceleration registers \*/

 I2CWriteLength = 2;

 I2CReadLength = 6;

 I2CMasterBuffer[0] = SLAVE\_ADDR << 1;

 I2CMasterBuffer[1] = 0x28 |0x80; /\* OUT\_X\_L\_A register with bit 7 set for address auto increment \*/

 I2CMasterBuffer[2] = SLAVE\_ADDR << 1|1; // set read bit

 I2CEngine();

 accX = (int)(I2CSlaveBuffer[1] << 8) | I2CSlaveBuffer[0];

 accY = (int)(I2CSlaveBuffer[3] << 8) | I2CSlaveBuffer[2];

 accZ = (int)(I2CSlaveBuffer[5] << 8) | I2CSlaveBuffer[4];

 printf("accX = %d \n", accX);

 printf("accY = %d \n", accY);

 printf("accZ = %d \n\n", accZ);

 if (accX < 0) { absX = -accX;} else {absX = accX;}

 if (accY < 0) { absY = -accY;} else {absY = accY;}

 if (accZ < 0) { absZ = -accZ;} else {absZ = accZ;}

 printf("absX=%d absY=%d absZ=%d \n",absX, absY, absZ);

 if (absX > absY && absX > absZ)

 if (accX > 0)

 printf("X Up \n");

 else

 printf("X Down \n");

 else if (absY > absX && absY > absZ)

 if (accY > 0)

 printf("Y Up \n");

 else

 printf("Y Down \n");

 else if (absZ > absX && absZ > absY)

 if (accZ > 0)

 printf("Z Up \n");

 else

 printf("Z Down \n");

 else

 printf("Titlted in several directions \n");

 }

 return 0 ;

}

# Write an embedded program that interfaces the LPC1769’s UART interface with the PC using CoolTerm. Display first the message “Welcome to COE 306.” Then, display all the text that the user types on the terminal in upper case. All punctual characters should remain as is. Use a 9600 baud rate, 8-bit word, no parity bit and one stop bit (default settings in CoolTerm). Include the source code of your solution along with a video snapshot to demonstrate correct functionality of your code.

/\*

===============================================================================

 Name : UART.c

 Author : $(author)

 Version :

 Copyright : $(copyright)

 Description : main definition

===========================================================================\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

void init\_UART(uint32\_t baud\_rate) {

 LPC\_PINCON->PINSEL0 |= (1 << 4) | (1 << 6); // Pin 0.2 is TX, Pin 0.3 is RX

 LPC\_UART0->FCR |= (1 << 0) | (1 << 1) | (1 << 2); // Enable FIFO, reset RX FIFO, and reset TX FIFO

 LPC\_UART0->LCR |= (1 << 0) | (1 << 1); // 8-bit word

 uint32\_t pclk\_value, temp;

 pclk\_value = 25E6; // PCLK is 25 MHz

 LPC\_UART0->LCR |= (1 << 7); // Enable Access to Divisor Latches

 // The Equation

 // baud\_rate = (pclk\_value) / (16 \* ((256 \* DLM) + DLL) \* (1+(DivAddVal/MulVal)))

 // Assuming (DivAddVal/MulVal) is equal to 0

 temp = (pclk\_value / (16 \* baud\_rate));

 LPC\_UART0->DLL = temp & 0xFF;

 LPC\_UART0->DLM = (temp >> 0x08) & 0xFF;

 LPC\_UART0->LCR &= ~(1 << 7); // Clear Access to Divisor Latches

}

uint8\_t getByte() {

 volatile uint8\_t ch;

 while (!(LPC\_UART0->LSR & 1))

 ; // wait until data is ready

 ch = LPC\_UART0->RBR;

 return ch;

}

void sendByte(uint8\_t ch) {

 while (!(LPC\_UART0->LSR & (1 << 5)))

 ; // wait until transmit signal is available

 LPC\_UART0->THR = ch;

}

int main(void) {

 init\_UART(9600);

 volatile int i;

 char\* welcomeMessage = "Welcome to COE 306\n";

 for (i = 0; i < strlen(welcomeMessage); i++) {

 sendByte(welcomeMessage[i]);

 }

 volatile uint8\_t ch;

 while (1) {

 ch = getByte();

 if (ch >= 'a' && ch <= 'z') {

 ch -= 32;

 sendByte(ch);

 }

 return 0;

}