Lab 4 – I²C and Accelerometer

Inter-integrated circuit interface. Accelerometer interfacing.

Introduction

The Inter-Integrated Circuit (I^2C) interface is a popular 2-wire interface that is used for synchronous medium-speed serial communication with local peripherals. Many peripherals, such as analog-to-digital converters, digital-to-analog converters, Flash memory, real-time clocks, temperature monitors, accelerometers, etc. come with built-in I^2C interfaces, making it easy for them to connect to a variety of microcontrollers. All devices share the I^2C "bus" and communication can take place between one master and multiple slaves or there can be multiple masters.

The MMA8451Q, 3-Axis, 14-bit digital accelerometer has low power, high resolution, and embedded signal processing for jolt detection, orientation, freefall, and taps. The applications of the accelerometer are numerous: e-compasses, orientation detection, tumble and freefall detection, pedometer step counting, motion detection, shock and vibration monitoring and user interfaces. Most modern mobile devices now incorporate accelerometers.

Objectives

1. To implement a hardware abstraction layer for the I^2C Interface.
2. To interface to an accelerometer.
3. To perform simple signal processing on a signal.
4. To expand the implementation of the Tower serial protocol.

Equipment

- 1 TWR-K70F120M-KIT – UTS
- 1 USB cable – UTS
- NXP Kinetis Design Studio

Safety

This is a Category A laboratory experiment. Please adhere to the Category A safety guidelines (issued separately).

Cat. A lab

L4.2

Software Requirements

1. The software is to incorporate all the features of Lab 3.
2. A hardware abstraction layer (HAL) is to be written for the Inter-Integrated Circuit (I^2C) module. The HAL should support the setting up of the I^2C module as a master only (no slave functionality is required). It must be able to address multiple slaves. It should also provide an interface for writing a single byte to a slave device and for reading multiple bytes from a slave device.
3. For reading multiple bytes from a slave device, we do not want to waste time polling. Therefore, you should implement an I^2C interrupt service routine that handles read operations by placing received bytes into a caller's data array and invoking a callback function when the receive operation is complete.
4. Refer to the Tower schematics (TWR-K70F120M-SCH.pdf) to determine how the accelerometer has been set up, and to determine appropriate ports and pins to manipulate on the K70.

   An accelerometer software module should be written that uses the I^2C to read accelerometer values for the X, Y and Z directions. In addition, the module should filter the returned results, using a median filter of the last three results, i.e. a "sliding window" of the three most recent values.

   The resolution of the accelerometer should be set to **8-bits**.

   The accelerometer is to use a baud rate that is as close to **100 kbps** as possible

   The accelerometer should be able to be placed in one of two modes:

   1. **Polling mode**: the accelerometer is passive and waits for the master to initiate a read of the data.
   2. **Interrupt mode**: the accelerometer takes samples periodically and asserts an interrupt to the microcontroller when data is ready.

   The datasheet for the accelerometer is readily available on the Internet.

In the main program, two different modes of PC communication are to be implemented: synchronous and asynchronous (the default mode).

- **Asynchronous mode**: the accelerometer should be put in polling mode at a **frequency of 1 Hz**. Accelerometer values should be sent to the PC **only if any** of the X, Y or Z values of the accelerometer have changed. If any axis has changed, all axes should be sent. It is important to only send data packets to the PC when a change in accelerometer values occurs, rather than continuously, so that the PC is not bombarded with extraneous packets.
- **Synchronous mode**: the accelerometer should be put in interrupt mode at a **frequency of 1.56 Hz**. Accelerometer values (X, Y and Z) should be sent to the PC regardless of whether the accelerometer values **have changed or not**.

The Tower should still respond to packets sent from the PC whilst in either mode.

5. The green LED should be toggled when the accelerometer is read.

6. Extra commands of the Tower serial protocol to be implemented are:

Tower to PC	PC to Tower
0x0A Protocol – Mode	0x0A Protocol – Mode
0x10 Accelerometer – Values	

7. On startup, or in response to reception of a "0x04 Special –Startup" packet from the PC, the Tower should send, in addition to all other "startup" packets:
   - a "0x0A – Protocol – Mode" packet
8. The Tower PC Interface has a tab specifically for Lab 4. It graphs the values sent back by the Tower. You should use this to verify that your software is working correctly.

9. Git must be used for version control. Note that version control will be assessed in Lab 5 based on the development of the software from Lab 1 through to Lab 5.

Hints

1. The accelerometer's device address is given in Table 10 of the datasheet. You will also need to observe the Tower schematics to determine whether pin SA0 is tied high or low – this is the last bit of the device address.

2. To set up the I2C unit, you will need to set the baud rate (i.e. the bits per second). See section 55.3.2 of the K70 Reference Manual for information on setting the baud rate. For a list of values corresponding to each ICR setting, see Table 55-41. Do an exhaustive search to find an achievable baud rate that is closest to the requested baud rate.

3. Don't confuse the K70's I2C Address Register with the accelerometer's address register – we're not using the K70 as a slave device, so there is no need to set a value in this register.

4. No I2C glitch filtering is used.

5. An example I2C timing diagram is given in Figure 11 of the accelerometer datasheet. You will need to manipulate the `TX, MST, RSTA, TXAK` bits of `I2C Control Register 1` when writing your routines for the I2C module.

6. The `I2C_S_BUSY_MASK` is used to determine whether the I2C bus is idle. You should check this before you initiate a communication. The `I2C_S_IICIF_MASK` is used to indicate that an event has occurred on the I2C bus, such as a one byte transfer.

7. A flowchart for a typical I2C interrupt service routine can be found in the *K70 Reference Manual* as Figure 55-42. The ISR only needs to support the receiving of characters in master mode. Note that the STARTF and STOPF bits (start flag and stop flag respectively) are not implemented in the K70. Therefore, you should only implement Fig 55-42 starting from the first box Clear IICIF.

L4.6

8. In using the accelerometer, we need to write ISRs to handle 2 interrupts – one for the I^2C bus, and one for when the accelerometer generates an interrupt (e.g. data is ready). Set the accelerometer up to use INT1 which is tied to PTB4 (see the Tower schematic). You will need to place the two ISR addresses into the vector table (I2C0 and PORTB).

Marking

1. **The software to be assessed must reside in the remote git repository before the start of your timetabled activity on the date specified in the Timetable in the Learning Guide.**
2. **Please create a "tag" called "Lab4Submission" (no spaces allowed) to the particular commit that you want marked. Markers will then create a branch from this tag called "Lab4Marking" in order to assess it.**
3. **Software marking will be carried out in the laboratory, in the format of a code review.**
4. **Refer to the document "Software Style Guide" for more details of some of the assessment criteria.**

L4.8

Assessment Criteria

Your lab will be assessed according to the following criteria:

Item	Detail	Evaluation	Mark
Opening comments / function descriptions	File headers and function descriptions are correct.	**G A P**	**/0.5**
Naming conventions / code structure	Names and code structure conform to the Software Style Guide.	**G A P**	**/0.5**
Doxygen comments	Comments for all functions, variables and modules.	**G A P**	**/0.5**
I^2C HAL	Public and private functions.	**G A P**	**/2**
Accelerometer HAL	Public and private functions.	**G A P**	**/1**
Median filtering	Efficient implementation.	**G A P**	**/1**
I^2C receive interrupts	For reading multiple bytes.	**G A P**	**/1**
Accelerometer interrupts	Interrupts initiated by the accelerometer.	**G A P**	**/1**
Protocol implementation	Protocol expanded.	**G A P**	**/0.5**
		TOTAL	**/8**

Evaluation

When we evaluate an assessment item, we will use the following criteria:

- **G =** All relevant material is presented in a logical manner showing clear understanding, and sound reasoning. For software - evidence of correct coding style, efficient implementation and / or novel (and correct) code.
- **A =** Most relevant material is presented with acceptable organisation and understanding. For software – some code may be prone to errors under certain operating conditions (e.g. input parameters) or usage, style may have inconsistent sections, occasional inefficient or incorrect code.
- **P =** Little relevant material is presented and/or code displays poor organisation or understanding of the underlying concepts.

Oral Defence

During the demonstration session you will be asked a number of questions based on material which you have learnt in the subject and then used to implement the assignment. You are expected to know exactly how your implementation works and be able to justify the design choices which you have made. If you fail to answer the questions with appropriate substance then you will be awarded **zero** for that component.