MASTER'S THESIS

Handheld Vibration Logger for Android Platforms

Tony Persson

Master of Science in Engineering Technology
Mechanical Engineering

Luleå University of Technology
Department of Engineering Sciences and Mathematics

LULEÅ UNIVERSITY OF TECHNOLOGY

Handheld Vibration Logger for Android Platforms

Tony Persson

Luleå University of Technology
Dept. of Engineering Sciences and Mathematics

Abstract

The aim of this thesis was to develop a handheld, battery-powered vibration logger based on an Android platform. The system consists of an Android device, for data presentation and user input, and a vibration sensor that would connect to the Android device by USB. The requirements of the system were to allow the user to configure the measurement settings, plot the measurement data and give the ability to save the measurement data from the Android device. The system was also to be adapted for industrial environments.

The development process involved choosing an Android platform, developing the electronics and housing for the accelerometer unit, develop an application for the Android-platform and firmware for the accelerometer unit's microcontroller.

The result is a vibration logger that fulfills the requirements stated. The Android-platform used for the system was a Motorola Xoom tablet. The tablet was equipped with a protective case and the accelerometer unit was a closed case that shields its components from the surrounding environment.

The user can configure the measurement settings and give the measurement a name and an index from the user interface. After each measurement the acceleration data is plotted on the Android screen to allow visual inspection of the data. If the user wishes to save the measured acceleration data this can also be done from the user interface.

From measurements performed on a test rig, the accelerometer unit case was evaluated. The results proved satisfactory and showed that the frequencies of interest are still preserved with the accelerometer placed inside the developed sensor case.

Preface

This thesis was done as a part of the Master of Science in Mechanical Engineering programme at Luleå University of Technology. The thesis work was carried out at Rubico AB in Luleå from October 2011 to March 2012.

I would like to thank every one at Rubico AB for giving me the opportunity to do this thesis and for support along the way. A special thanks to my supervisor, Patrik Pääjärvi, and my examiner Jan-Olov Aidanpää.

Tony Persson
Luleå, 25 March 2012

Contents

Chapter 1

Introduction

This chapter will introduce the subject to the reader by giving a background, describe the problem to be solved, list the project aims and lay out the scope and delimitations of the thesis.

1.1 Background

Rubico develops products and services for condition monitoring of rolling-element bearings based on vibration analysis. Vibrations from bearings are measured and analyzed during operation in order to detect damages early on and thereby facilitate planned maintenance.

Vibration analysis is a proven and commonly used technique in the heavy industry [15]. However, Rubico has developed a patented signal processing approach that has shown very promising performance, including applications where existing methods often fail for reasons such as slow rotating machinery.

1.2 Project Motivation

The system used by Rubico to gather vibration data today consists of a laptop, an AD (analog-to-digital) converter and an analog accelerometer. This concept has proven unsuitable because of the harsh environment around the measurement objects. Laptops and AD converters are often too fragile for the industry environment and unless the laptop and the AD converter is placed onto a surface, this equipment is hard to operate.

1.3 Project Aims

The aim of this thesis is to develop a prototype of a handheld, battery-powered vibration logger based on an Android platform. The system should consist of two units connected by cable, where the first unit is a handheld Android device that present data to the user and process user input. The second unit is an accelerometer unit that connects to the measurement objects by magnet to measure the vibrations.

The prototype should meet the following requirements:

- Allow the user to setup measurement settings from a user interface.
- Plot the measured vibration data to allow visual inspection.
- Give the option to save vibration data to a file for future analysis.
- Give the possibility to add additional information, such as a name, to the measurements.
- Be adapted for use in harsh industrial environments.

1.4 Scope and Delimitations

The main focus of this thesis is to create a prototype that collects, visualize and stores vibration data from an accelerometer. No processing of the vibration data is required. The handheld device should be based on an already existing device, such as an Android tablet or smartphone. The choice of the accelerometer is already predetermined.

Already existing solutions, such as open source libraries, should be utilized during development to the extent possible. A casing for the accelerometer unit should be designed as time permits.

Chapter 2

Theory

This chapter will give an overview of some of the theories and facts used during the development of the measurement system. This includes information about the accelerometer and some basic knowledge about USB and Android development.

2.1 Accelerometer

An accelerometer is an instrument that senses the motion of a surface to which it is attached and produces a signal that is proportional to this motion[14]. The accelerometer measures the change in motion, thus the acceleration, of the surface or object it is attached to. Accelerometers can be built to measure accelerations very precisely, which makes them able to measure everything from big motions down to small vibrations.

Accelerometers can be analog or digital. In the case of an analog accelerometer it will produce an electrical signal proportional to the motion. In the case of a digital accelerometer the signal can be read digitally via an interface such as SPI or I^2C.

2.1.1 ADIS16228

The ADIS16228 is an accelerometer adapted to vibration analysis, condition monitoring and diagnostics. It is a triaxial accelerometer with a measurement range of $\pm$ 20g. It has a 5kHz flat frequency band and the sample rate can be set in discrete steps between 20Hz and 20kHz. Detailed information about the ADIS16228 can be found in its datasheet[12].

The ADIS16228 uses SPI (Serial Peripheral Interface Bus), at a clock frequency up to 2.5MHz, for communication. The accelerometer has a set of registers to manage configuration input and data outputs. Each register is 16-bits, has its own unique 7-bit address

and can be accessed from the SPI-interface using read and write commands.

The raw acceleration data can be accessed in two ways. Either the collected acceleration samples are saved to a buffer during the measurement, when the mesurement is done this buffer becomes accessible for reading. This will limit the length of the measurement, since the ADIS16228 buffer only can hold 512 samples per axis. The other way to access measurement data is real-time mode, where the acceleration data can be read continuously. A status pin, called DIO1, can be configured to be given a low output level whenever a new sample is measured, indicating to the external device connected to read the corresponding axis register.

The accelerometer's internal temperature and supply voltage are measured for each acceleration measurement. This data can be accessed by reading the corresponding registers using the SPI-interface after the measurement is done.

SPI communication

To give external access to the accelerometer registers the ADIS16228 uses SPI for communication. The access to the registers are based on read and write operations to and from the accelerometer registers. Each operation, or command, consists of one or two 16 bits SPI cycles depending on the operation. The ADIS16228 SPI interface supports full duplex communication, which means support of simultaneous transmitting and receiving of data.

All transmissions to the accelerometer starts with the first bit being a read or write bit, where a "1" indicates to write to a register and "0" indicates to read a register. The read/write bit is followed by the 7-bit register address to perform the operation on.

Each write cycle consist of the read/write bit set to "1" and the register address to the upper or lower byte of register as the first byte. The second byte contains the data wished to be written. In other words, a write command only writes to half the register and a second write cycle needs to be performed if we want to write to the complete register.

A read command always takes two SPI cycles to complete. The first cycle consists of the read/write bit set to "0" and the register address as the first byte. The second byte of the cycle will not be processed by the accelerometer for a read command and can be set to anything. To receive the content of the register a second SPI cycle needs to be performed where the register data is transmitted. This is where full duplex can be utilized and perform another operation if needed, otherwise a dummy operation can be performed just to provide a clock sequence for the second cycle.

2.2 USB

USB (Universal Serial Bus) is a serial data bus designed to standardize the connection between computers and computer peripherals. Today, USB is one of the most widely used method to connect computers and computer accessories such as keyboards, printers and external storage media. USB has also found its way into other products such as mobile phones and video game consoles. Information about USB can be found in the USB 2.0 specification[17].

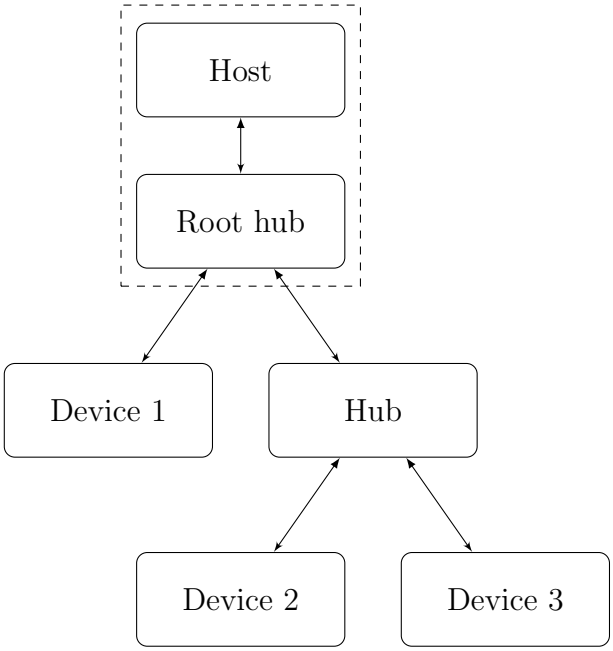

Figure 2.1: Example of physical bus topology.

2.2.1 Bus Topology

The USB connects devices to a host. The host is the coordinating entity of the bus and is responsible to detect attachment and removal of devices, manage control and data flow between host and devices, collect status and activity statistics and provide power to the attached USB devices. The USB devices are the peripherals attached to the host, providing device specific functionality to the host.

The physical bus topology is illustrated in Figure 2.1. The devices connect to the host in a tree-like configuration. Attachment points are provided by a special class (more about

classes in Section 2.2.4) known as a hub. A hub allows the tree configuration to branch into one or more devices. The host is usually combined with a root hub.

While the end user's view would simply be a host connected to a physical device, the implementer's viewpoint is more complicated and illustrated in Figure 2.2. Three layers can be recognized in the communication:

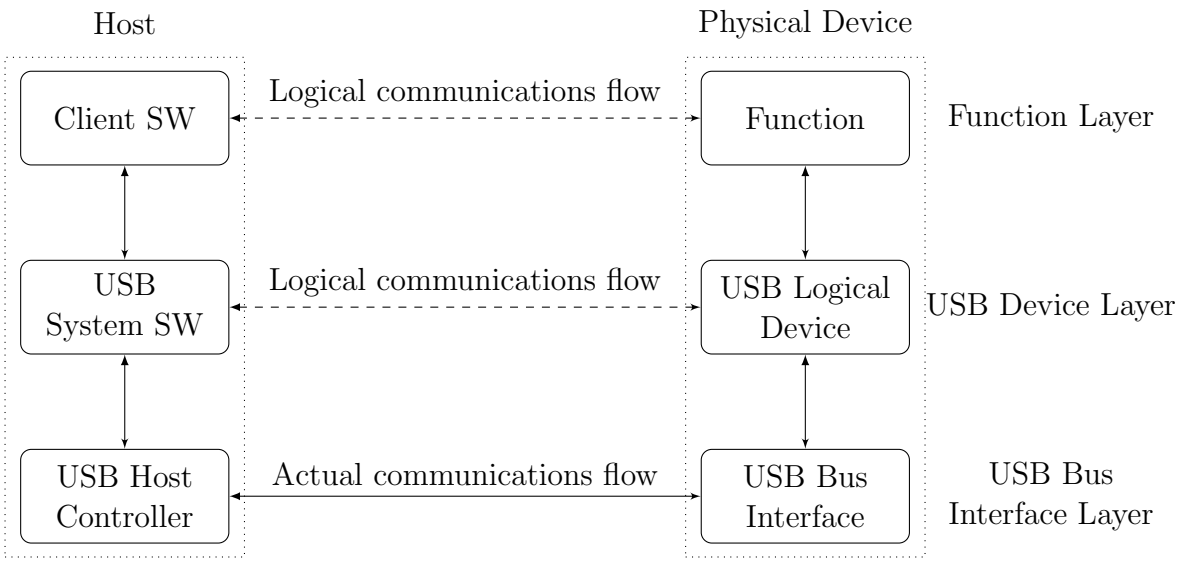

Figure 2.2: USB implementation areas.

- **USB Bus Interface Layer**: Provides the physical/signaling/packet connectivity between host and a device. The actual communication flow takes place on this layer between the *USB Host Controller* and the *USB Bus Interface*, both of which contain the hardware and firmware that allows the host and the device to connect.

- **USB Device Layer**: A logical layer that the *USB System Software* use for performing generic USB operations on the *USB Logical Device*. All Logical Devices present the same basic interface to the host. The USB System Software is usually implemented in the host at operating system level.

- **Function Layer**: The function layer is a logical layer for communication between device specific *Functions* and the device specific *Client Software* running on the host. The Functions vary between the different device classes.

2.2.2 Communication Flow

The USB provides a communication service between the client software running on the host and the device functions. Since different device functions has different requirements for the client-to-function interactions, the USB provides separation between different communication flows, where each communication flow is terminated at an *endpoint* on the device.

Endpoints

An endpoint can be seen as the terminus of a communication flow in a USB device. Each USB logical device is composed of a collection of endpoints. At attachment time, the host assigns each logical device a unique address and give the endpoints a device-determined number. Endpoints has a device-determined direction of data flow: IN for communication from the device to the host and OUT for communication from the host to the device.

Each endpoint has its required transfer characteristics specified. The characteristics include latency and bandwidth requirements, endpoint number, error handling behaviour requirements, maximum packet size, *transfer type* (see Section 2.2.3) and data direction.

All USB devices must implement both an IN and an OUT default endpoint, both with endpoint number zero (referred to as endpoint zero). The USB System Software uses this default endpoint to initialize and manipulate the logical device. Endpoint zero is always accessible once a device is attached, powered and has received a bus reset.

2.2.3 Transfer Types

There exists four different transfer types that are optimized to more closely match the requirements of a device function. Each transfer type determines various characteristics of the communication.

Control Transfers

Control transfers are intended to support configuration, command and status type of communication flows. The transfers are error-free and the USB system will make a "best effort" to deliver control transfers. The zero endpoint uses control transfers.

A control transfer is composed of three stages. The first stage is the Setup Stage and consists of three packets, see Figure 2.3. The Setup Stage starts with the host initiating the transaction by sending two packets, a Setup token that contains address and endpoint number and a data packet that details the type of request. The final packet is a handshake packet sent by the device to indicate a successful transaction or errors.

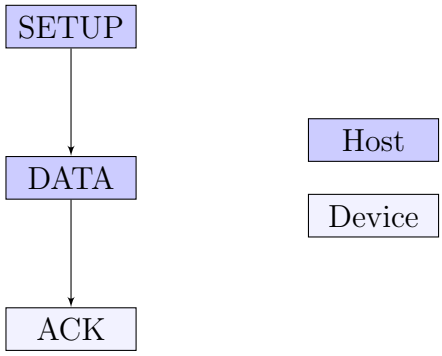

Figure 2.3: Setup stage of control transaction.

The second stage is the optional Data Stage and consists of three phases, see Figure 2.4. When the host is ready to send or receive, depending on the direction set in the Setup Stage, it issues an IN or OUT token. In the case of an IN token, the device will respond by sending one of the following:

- DATA packet: Containing the requested data.
- STALL packet: Indicating the endpoint has had an error.
- NAK packet: Indicating that the endpoint is working, but has no data to send.

In the case of an OUT token as the first packet in the Data Stage, the host will follow it with the DATA packet. The device will then respond with an ACK, STALL or NAK in the same fashion as the host would have done in the case of an IN packet.

The third and final stage of a control transaction is the Status Stage that reports the outcome of the previous Setup and Data stages of the transfer. The Status Stage is identical to the Data Stage, but the direction will be reversed. An IN packet as the first stage in the Data Stage will give an OUT packet as the first packet in the Status Stage and the other way around, but with the difference that the DATA packet is empty.

Bulk Transfers

Bulk transfers are intended to be used to transfer larger amounts of data. The communication is error-free, but the bandwidth taken up by bulk transactions can vary depending on other bus activities. Typical devices where bulk transactions are used are printers and scanners.

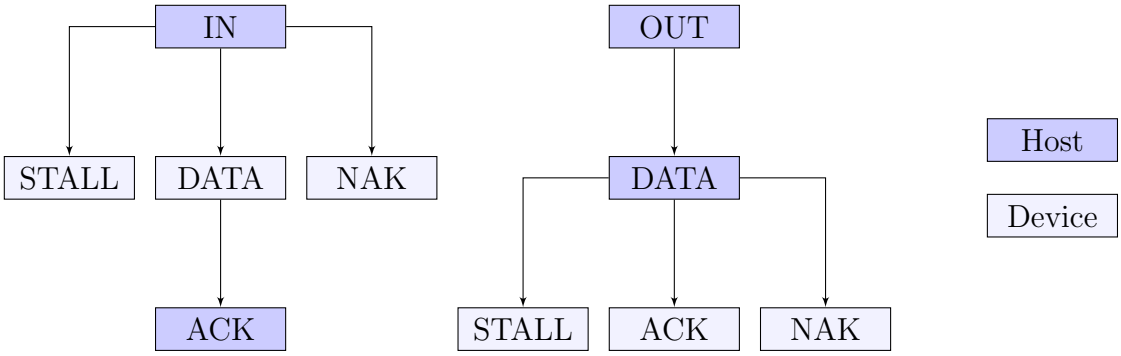

Figure 2.4: Data stage of control transaction.

A bulk transaction is performed as the Data Stage in a Control transaction, see Figure 2.4. The host issues an IN or OUT packet depending on the transfer direction. In the case of an OUT packet the host then sends the DATA packet and the device answers with either a STALL, ACK or NAK depending on if the transaction was successful or not. In the case of an IN packet, the device answers with a STALL, DATA or NAK and the host finishes the transaction with an ACK.

Interrupt Transfers

The interrupt transfer type is designed to be used when transactions will happen infrequently, but with bounded service periods. Typical devices that use interrupt transfers are pointing devices or keyboards. More information about interrupt transfers can be found in the USB 2.0 specification [17].

Isochronous Transfers

Isochronous transfers are continuous and periodical. Low latency is prioritized over error-free communication. A typical use of isochronous transfers are audio or video streams. More information about isochronous transfers can be found in the USB 2.0 specification [17].

2.2.4 USB Classes

USB devices report their attributes using data structures called descriptors. Device descriptors or interface descriptors can specify USB class codes. The class code consists of three bytes: Class, SubClass and Protocol, and are used to identify a device's functionality. A class code is assigned to a group of related devices and the subclass code divides the devices in class. The protocol code can define how to communicate with the device.

The class codes can specify a standard class such as Audio, Mass Storage or Printer, but the class code can also be set to Vendor Specific, which allows vendors to use it as they please.

2.2.5 USB On-The-Go

Many portable devices use USB for PC connectivity. These devices sometimes need to communicate directly with a peripheral without a PC available. It would be impossible for the portable device and the peripheral to communicate if both where USB devices, since the bus would miss a host, but USB OTG (On-The-Go) solves this problem.

A USB OTG device can take the role as both a host or a device. This means that if the USB OTG device is connected to a host, such as a PC, it would take the role as a USB device. If the USB OTG device is connected to a device, such as a keyboard, it would take the role as a host.

The OTG device is informed about its role by the cable that connects the devices. The Standard A/B USB connectors have four pins: Ground, Data-, Data+ and 5V. The mini and micro USB plugs used by OTG devices have a fifth pin called ID. If the ID-pin is floating the OTG-device will take the role as a device, but if the ID-pin is grounded it takes the role as a host[16].

2.3 Android

Android is an open source software stack for mobile devices. The software stack consists of a kernel based on the Linux kernel, libraries, an application framework and a set of core applications (also known as apps). Android is developed by the Open Handset Alliance led by Google. Information about Android and application development can be found at the Android Developer website[1].

2.3.1 Android SDK

The Android SDK is a software development kit that provides a variety of tools for developing applications for the Android platform. The SDK includes a debugger, a handset emulator, a system logger, necessary libraries, documentation, sample code and many other tools to support application development.

2.3.2 Android and USB

Most Android devices are equipped with a USB port. On the early Android handsets the USB port's main role was to allow the Android device to connect to a computer for synchronization or file sharing. The USB port is also often used to charge the Android device's battery.

The USB port opens up for the possibility to connect the Android device to a variety of USB peripherals. The ability to utilize the USB-port from applications was introduced with Android 3.1 and the *USB API*.

Android USB API

The Android USB API was introduced with Android version 3.1 and allows the Android applications to communicate with external USB hardware. The API supports two modes: Accessory mode and host mode.

When set to accessory mode, the Android device can connect to host hardware specifically designed for Android-powered devices. The host hardware must utilize a communication protocol called Android accessory protocol. In this mode the Android device acts as an USB-device and the external USB hardware must be the host.

When the Android device is set to host mode, the Android device acts as USB host, which means powers the bus and enumerate connected USB devices. The USB hardware of the Android device must support USB OTG to utilize this mode.

Android NDK and usblib

The Android NDK is a toolkit that allows parts of an Android applications to be written in a native-code language such as C or C++. To use the NDK can be beneficial for CPU-intensive operations to increase speed, or to reuse existing code.

The Android NDK can be used to implement libusb, which is an open source C library intended to give easy access to USB devices, in an Android application. By cross-compiling libusb for Android and by creating an interface using Java native interface (JNI), libusb can be utilized by the applications[13]. A downside of this method is that to acquire USB communication, the application must gain permissions to the USB hardware. In order to gain the permissions the Android device must give the phone user super-user privileges (also known as to "root" the device).

Chapter 3

Hardware Design

This chapter will describe how the system hardware was chosen and designed, including selection of platform for the handheld device, design of the electronics involved in the accelerometer unit and design of the casing for the accelerometer unit.

3.1 System Overview

The measurement system's hardware would consist of two units: A handheld device fitted with a screen for data presentation and either a touch interface or a hardware keyboard for data input. The second unit was a sensor unit to be to magnetically mounted on the measurement object. The two units were to be connected by cable for data transfers and sharing of battery supply.

The sensor unit would use an accelerometer for vibration measurements. The sensor unit, hereafter referred as accelerometer unit, would be magnetically connected to the measurement object. The accelerometer unit would, in addition to the accelerometer, consist of necessary electronics to communicate with the hand held unit.

The measurement system would be used according to Figure 3.1. The user will attach the accelerometer unit to a surface of the measurement object. A cable is connecting the accelerometer unit to the handheld unit. From the handheld unit the user will operate the system and setup measurements, view the data and save the data to files.

3.2 Selecting the Handheld Platform

The handheld unit, used for user input and data output, was decided to be a tablet computer. A tablet would be suitable since it is a device which can easily be operated

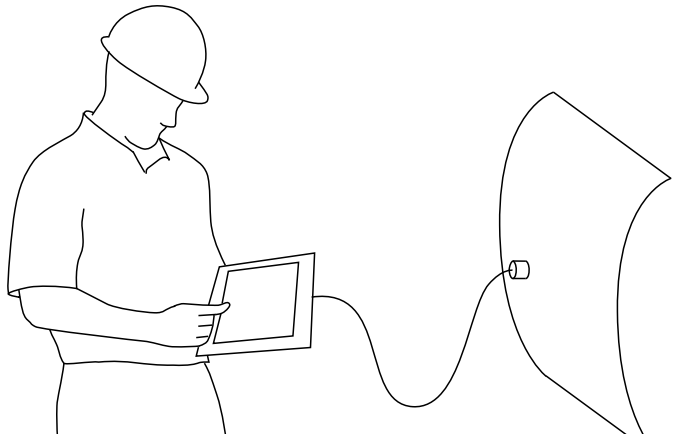

Figure 3.1: Concept sketch of the system in use.

without placing the device onto a surface. The user can control a tablet computer by holding the device in one hand and operate the touch user interface with the other hand. A touch screen tablet computer would also allow an easy interface without unnecessary buttons and controls, unlike a device with a hardware keyboard. A tablet computer is also usually more dust resistant than a laptop because of its closed case.

The next step was to decide which type of tablet to use. The common tablet types were Android tablets, Windows tablets and Apple iPad. Each type had its pros and cons. An Android tablet was preferred based on the following features:

- **USB-host**: Android offers support for USB-host from version 3.1 and up. USB-host would give the advantage of supplying power to the accelerometer unit and eliminating the need for a battery in the accelerometer unit. USB is also a widespread standard for communications between computers and peripherals which would open up for the possibly to use the accelerometer unit together with other devices as well, as long as the host device has the correct client software implemented.

- **Rugged devices**: Both Motorola[9] and Panasonic[11] were developing enterprise and industrial adapted Android tablets when this thesis was carried out. While a rugged Android tablet would be the best platform for the system, a consumer tablet could be used for development and then later be changed to a rugged tablet, since Android applications are not device specific.

- **Battery time**: The battery time on most Android tablets are specified between eight and ten hours, which is relatively long given the low weights of the tablets.

Since none of the rugged tablet models were released when the thesis was carried out, a consumer tablet would be used for the development of the system. A wide variety of consumer Android tablets were available and in the end the Motorola Xoom was chosen based on the following features:

- **Android 3.2**: The latest version of Android was available for the tablet at the time when the platform was to be chosen. This meant a high likelihood that the USB-host feature was enabled on the device.
- **Accessories**: Since the Motorola Xoom is a consumer tablet it was not adapted to be used in harsh environments, but there existed accessories such as protective cases that would give the tablet more resistance to the surrounding environment.
- **Robustness**: The tablet's shell was made from aluminum and was a more robust construction than some of the competing tablets made from plastics.

To make the Xoom more rugged and adapted to industrial environments the following accessories were acquired:

- **Otterbox Defender Series Case**: An impact absorbing silicone skin that wraps around the outside of the tablet to give a more rugged protection.
- **Belkin ScreenGuard**: Provides a protective layer over the screen.
- **Otterbox Utility Series Latch**: A clip to strap the tablet to the users hand and thereby make it easier to carry and operate.

3.3 Accelerometer Unit Hardware

The accelerometer unit, the unit that would be magnetically mounted to the measurement objects, was to be designed. The accelerometer unit's main tasks were to provide a way to communicate with the accelerometer, to initiate measurements and receive acceleration data, and communicate with the Android device to receive measurement settings and forward acceleration data.

The accelerometer unit would consist of the accelerometer for vibration measurements, a printed circuit board (PCB) containing all electronic components needed, a case to shield the components from the surrounding environment, a USB connector mounted onto the case for connection to the Android device and a magnet to connect the accelerometer unit to the measurement object.

3.3.1 Electronics

The choice of accelerometer model to use was out of scope for this thesis and was already decided at the start of the project. An Analog Devices ADIS16228[12] would be used. The ADIS16228 was a triaxial vibration sensor that offered measurement characteristics that would suit the system. It used SPI for communication and offered a real-time mode where acceleration data could be read continuously during the measurements. The real-time mode gave the advantage of not limiting the number of acceleration samples collected in each measurement.

An NXP LPC11U14[4] microcontroller was chosen to be used for the accelerometer unit. The LPC11U14 offered both a SPI interface for communication with the accelerometer and USB capabilities for the communication with the Android device. Another NXP microcontroller from the same series had been used in an earlier project, so knowledge about the microcontroller already existed within the company.

Components and PCB

All components, except for the accelerometer and the USB connector, would be mounted on to a PCB. The accelerometer and USB connector would still be connected to the PCB, but by cable connection. The following list shows the main components of the PCB:

- **Microcontroller**: The microcontroller was going to be mounted on to the PCB.
- **FFC connector**: The ADIS16228 was equipped with a flat flexible cable (FFC) for connection. The corresponding connector was needed on the PCB.
- **USB connector**: A Bulgin Buccaneer Mini-USB connector was chosen as the USB-connector. This connector was equipped with a five-way cable for connection. A corresponding connector was needed on the PCB.
- **JTAG connector**: To program and debug the microcontroller, a JTAG-connector was needed.
- **Voltage regulator**: The USB host, the Android tablet in this case, delivers 5V supply voltage to the USB bus, but the microcontroller and the accelerometer need 3.3V supply voltage. Therefore a voltage regulator was needed.

PCB Design Considerations

Since the PCB dimensions would set the limitations of the case dimensions and the case dimensions would set the limitations of the PCB dimension, the case designs had to be taken into consideration while designing the PCB. For that reason the two were designed in parallel to simplify the development process. More information about the case design

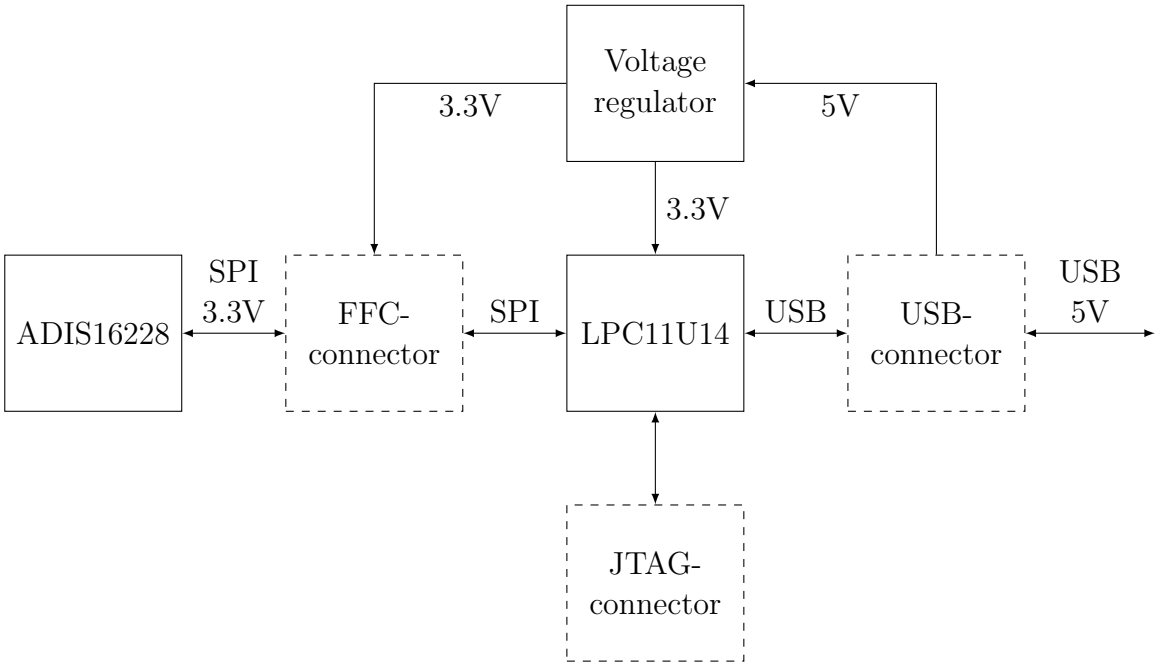

Figure 3.2: Main parts of the PCB.

can be read in Section 3.3.2.

The components would have to be placed onto the PCB in a way that would minimize the PCB dimensions and thus minimize the case dimensions. The FFC connector needed to be placed in a way that would allow the accelerometer and the PCB to be mounted according to the design described in Section 3.4.

The USB connector chosen was equipped with a standard five-ways pin-header for connection to the PCB. This connector would occupy a big area of the PCB in comparison to the other components, because of this the pin-header was removed and the PCB was prepared for a smaller connector instead. In the end the USB connector cables were soldered directly to the PCB.

Modeling Tool

To design the PCB used for the accelerometer unit CadSoft EAGLE[2] was used. EAGLE is a tool to design circuit schematics and PCB designs. EAGLE also offers a library containing a wide range of components to be used in the designs.

During the electronics development EAGLE was first used to create some of the components missing in the components library, a circuit schematic of the electronics was then

created and finally the PCB design was created.

Final PCB Design

The final PCB design was given the dimensions 29 x 29 mm. The populated PCB can be seen in Figure 3.3. In the figure the accelerometer is connected to the FFC connector and the USB connector is soldered directly to the PCB by cables.

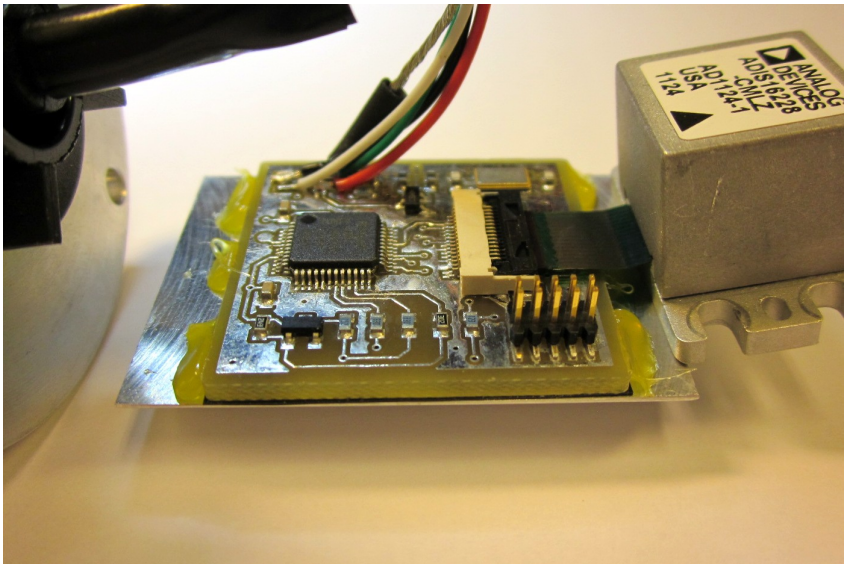

Figure 3.3: The PCB with USB connector and accelerometer connected.

3.3.2 Case Design

To shield the accelerometer unit's components from the surroundings, and allow the prototype to be used in the field, a case was designed. The case was made to hold the accelerometer and the PCB mounted inside it and allow the USB-connector and a magnet to be mounted from the outside.

Modeling Tool

The accelerometer unit case was designed using the software Siemens NX[8]. The parts were created using solid modelling and then drafts from the solid models were created and sent for manufacturing.

Design Considerations

Since the accelerometer unit case was of low priority and was to be designed as time would permit, the accelerometer case was designed very simple. No calculations or simulations were made of the design, it was simply designed with the following requirements:

- Hold and shield the accelerometer and the PCB from dust and liquids.
- Allow the USB connector to be attached to the case.
- Be equipped with a 10-32 UNF threaded hole for mounting of a magnet.
- Allow easy mounting of the components.
- Have relatively small outer dimensions.

Since the case dimensions would set the limitations of the PCB dimensions and the PCB dimensions would set the limitations of the case dimension, the PCB designs had to be taken in to consideration while designing the case. For that reason the two were designed in parallel to simplify the development process. More information about the PCB design can be read in Section 3.3.1.

A hollowed cylindrical shape was chosen for the accelerometer case body, since a cylindrical shape would be easy to manufacture. The material was decided to be aluminum to keep the weight light.

The ADIS16228 accelerometer's connection was of FFC type and was placed out from the side of the accelerometer. Due to the length of the FFC, the circuit board holding the FFC-connector had to be placed close to the accelerometer body. If the accelerometer would be mounted to the bottom of the hollowed cylindrical case, the best way to mount the circuit board would be to place it verticaly next to the accelerometer body.

Since the case had a cylindrical shape and the circuit board is flat, a circuit board holder was designed in order to mount the circuit board verticaly. The holder is rounded with the same radius as the inner radius of the cylindrical case on one side and flat on the other side. The circuit board holder was also designed with two threaded holes on the rounded side so it could be fastened to matching holes on the case body.

Final Case Design

The final case design can be seen as an exploded view drawing in Figure 3.4 and assembled in Figure 3.5. The USB connector **(C)** is mounted to the case cap **(B)** using a screw-nut **(A)**. The circuit board **(D)** is fastened to the to the circuit board holder **(E)** by glue. The circuit board was then connected to the accelerometer **(F)** and the USB connector

by cable. The whole package was then put inside the case body **(G)**. The accelerometer was mounted to the bottom of the case body by four screws and the circuit board holder was connected to the case body by two screws from the side of the body. Finally the cap was connected to the top of case body using four screws and an insulating washer **(H)** and the magnet **(I)** was screwed to the bottom of the case body.

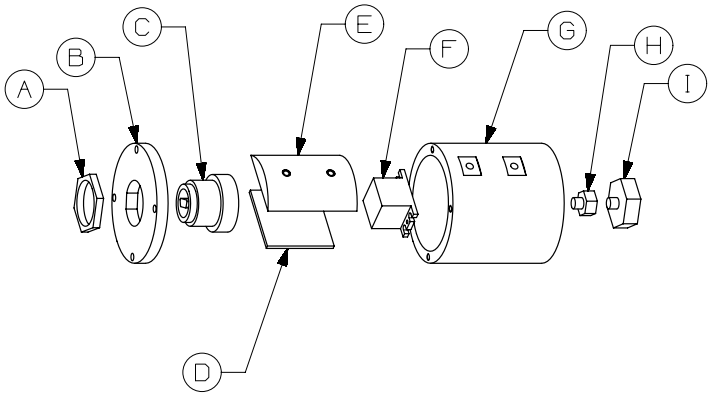

Figure 3.4: Exploded view drawing of the accelerometer unit.

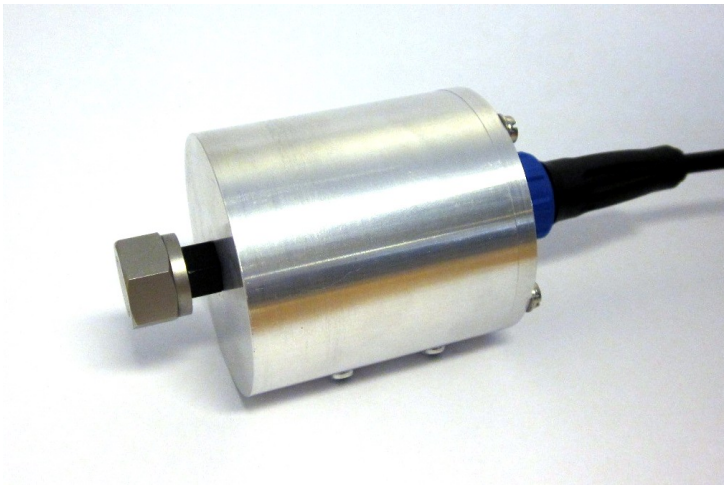

Figure 3.5: The assembled accelerometer unit.

Chapter 4

Software Design

This chapter will describe the key design considerations for the development of the software used in the accelerometer unit and the Android application.

4.1 System Overview

To structure the software development, a methodology for how the system should operate was developed from the project aims. The methodology shows how the user interacts with the system software and some of the key operations performed by the system in response to the user input:

1. The user selects a measurement axis (x, y or z), a sample rate(Hz) and a measurement length(sec.) using the Android device. If the user wishes to name or give the measurement an index, this information can be added as well.

2. The measurement is started by pressing a touch button and the Android device sends the measurement settings to the accelerometer unit.

3. When the accelerometer unit receives the measurement settings, it configures the accelerometer accordingly and start the accelerometer in real-time sampling mode.

4. The accelerometer unit forwards the acceleration data, received from the accelerometer, to the Android device.

5. When the measurement is done, the Android unit requests additional information such as accelerometer temperature. The accelerometer unit fulfills this request by returning the information.

6. The Android application visualizes the measurement data to let the user inspect the data.

7. If the user wishes to save the measurement data this can be done by pressing a touch button and filling in a filename. The measurement data will then be saved to a file.

Design Considerations

Since the measurement system consists of two units, the Android tablet and the accelerometer unit, two different softwares were developed. The development of each of these two are described in Section 4.3 and 4.4.

It was decided that the Android device would be given the ability to read and write accelerometer registers directly. This was done by adding USB requests for reading and writing accelerometer registers. This decision was based on the fact that it would give the ability to modify, add or remove accelerometer calls by simply updating the Android application instead the accelerometer firmware which would simplify the development. These requests were used for configuring the accelerometer according to the selected measurement settings.

During the acceleration measurements, the accelerometer would be set to real-time mode in order to capture measurement series of sufficient length. Because of its design, the transaction type to read a single register would be unsuitable for sendning acceleration data. A single register read transfer would not use full duplex SPI communication and would only send a single register in each USB transaction. The transfer rate would not be high enough for the higher sampling rates.

In order to transfer the acceleration data another type of USB transaction had to be used, but to initiate a measurement, a request would be used. This request would include the measurement axis and the number of data samples to acquire based on the sample frequency and the length of the measurement.

When the accelerometer is set to real-time mode it starts the measurement by measuring the temperature and the supply voltage. This information was also desired and had to be transmitted to the Android device. The method used was to read the corresponding accelerometer registers in the end of each measurement and save the information temporarily. To forward the information to the Android device a fourth type of request was added where the temperature, the supply voltage, the version of the accelerometer unit software and a status flag was sent.

4.2 Custom USB Class

A vendor specific USB class was developed for communication between the Android device and the accelerometer unit. The class was developed since none of the standard classes would fit the functions provided by the accelerometer unit and a new class would give the ability to customize the types data transactions.

Five different types of data transactions between the Android device and the accelerometer unit were needed:

- **Read accelerometer register**: This type instructs the accelerometer unit to read a specified accelerometer register and return the register content. This type was used during configuration of the accelerometer.
- **Write accelerometer register**: This type instructs the accelerometer unit to write data to a specified accelerometer register. This type was used during configuration of the accelerometer.
- **Start the measurement**: Instructs the accelerometer unit to start the measurement and also provides the measurement axis and the number of samples to collect.
- **Information**: This type instructs the accelerometer to return information about a previous measurement. The information consists of accelerometer temperature and supply voltage, software version of the accelerometer module and also a status flag register.
- **Acceleration data:** This type consists of acceleration data collected during the measurement.

From the type of communication and the amount of data to be transmitted, the five different types were separated into two categories: Small and irregular data transactions for configuration purposes, including the first four types in the first category and large transactions, including the fifth type, in the second category.

For the first four types it was decided to use *control transactions*, since this type is error-free and the transaction type is intended to support configuration, command and status type of communication flows. It was also decided to use the zero endpoint that is always implemented for all USB devices. The control transactions would be two-way communication between the accelerometer unit and the Android device.

For the acceleration data *bulk transactions* was chosen since this type of communication would include larger data amounts and a need for error-free communication. The acceleration data would also be one-way communication from the accelerometer unit to the Android device. Bulk transactions would give a variable bandwidth, but since the accelerometer unit is the only device connected to the bus, this would not be a problem.

Control Transactions

In the Setup Stage of a control transaction two 16-bit fields allow parameters to be passed with the request. These registers were used for the start measurement request and the read/write single accelerometer register requests to append the accelerometer register address wished to operate on, or in the case of a start measurement request, contain the register address to the to the axis data buffer wished to be read.

Bulk Transactions

The maximum size of bulk transactions in a full-speed USB 2.0 are 64 bytes of data in each transaction. The accelerometer provides the acceleration data as 16 bit values, which gives 32 measurement values in each bulk transaction. The Android device will send IN packets continuous during the accelerometer data transmissions and the accelerometer unit would respond with DATA packets containing the acceleration data.

4.3 Android Software

The software for the Android device consists of an Android application, or app, that was developed for the system. The Android application handles user input, USB communication, data storage and data presentation.

4.3.1 Development Tools

The software for the Android tablet was written in the Java programming language using the development platform MOTODEV Studio for Android[6]. MOTODEV Studio is a development platform based on Eclipse IDE[3] and the Android SDK.

4.3.2 Xoom Firmware Version

In the early stages of the software development it was discovered that the USB API did not work on the Motorola Xoom tablet. Motorola's customer support stated that the Xoom tablet did not have official support for USB host. This was strange since reports were found where people successfully used the USB API on the Xoom [18].

It turned out that Motorola had chosen to disable the USB API. Any reason for this decision was never found, but the U.S. version of the WiFi-only Motorola Xoom (MZ604) is a Google Experience Device [10]. This meant that Google manages software updates themselves without interference from Motorola. After changing from the European 3G version (MZ601) firmware to the U.S. WiFi-only version (MZ604) firmware the USB API was working properly.

4.3.3 Android Software Overview

The Android software's main functions were:

- Handle user input and data presentation from a user interface.
- Visualize measurement data as a chart.
- Implement the USB class described in Section 4.2. Which includes the four control transactions on endpoint zero and the bulk transaction for acceleration data.
- Give the ability to configure measurement settings and save acceleration data to a file.

4.3.4 User Interface

The user interface was designed according to Figure 4.1. As seen in the figure the main area of the user interface is occupied by a chart where the measurement data is visualized after each measurement. The chart shows time in seconds on the X-axis and acceleration in g in the Y-axis. This chart is emptied when the application starts and when the user chooses to make a new measurement.

There are four text fields in the user interface. Two text fields are read only and shows the status of the accelerometer unit, if it is connected or not, and the second read only field shows the accelerometer temperature after each measurement. The two other text fields are used to give the measurement a name and an index.

There are three drop-down menus, also called spinners, where the user can select the measurement axis, the sample rate and measurement length in seconds. Drop-down menus were used since the accelerometer only offers a fixed number of choices for axis and sample rate settings and it is easier to select the measurement length from a menu rather than entering a number using the touch keyboard.

Finally there are three touch buttons to create a new measurement, save a measurement or start a measurement. These buttons are enabled or disabled depending on the state of the system. During measurements and file saving all buttons are disabled. After a measurement the start measurement button is disabled and after a new measurement is created the new button is disabled.

4.3.5 Chart Engine

A chart class was developed to visualize the measurement data. The chart occupies the main part of the interface as seen in Figure 4.1. When a measurement is finished the chart objects gets called with the new data as arguments. The chart will plot the data and adjust the scale on both axes to fit the data.

The accelerometer does not have any way to warn the user if the signal exceeds the maximum measurement range. The measurement data would be corrupted if the signal hits the maximum range and the user has to be warned if this might happen. To warn the user about possible data corruption the color of the measurement data line will be red, instead of white, if the signal exceeds a threshold value of 19 g, which is 1 g under the maximim range.

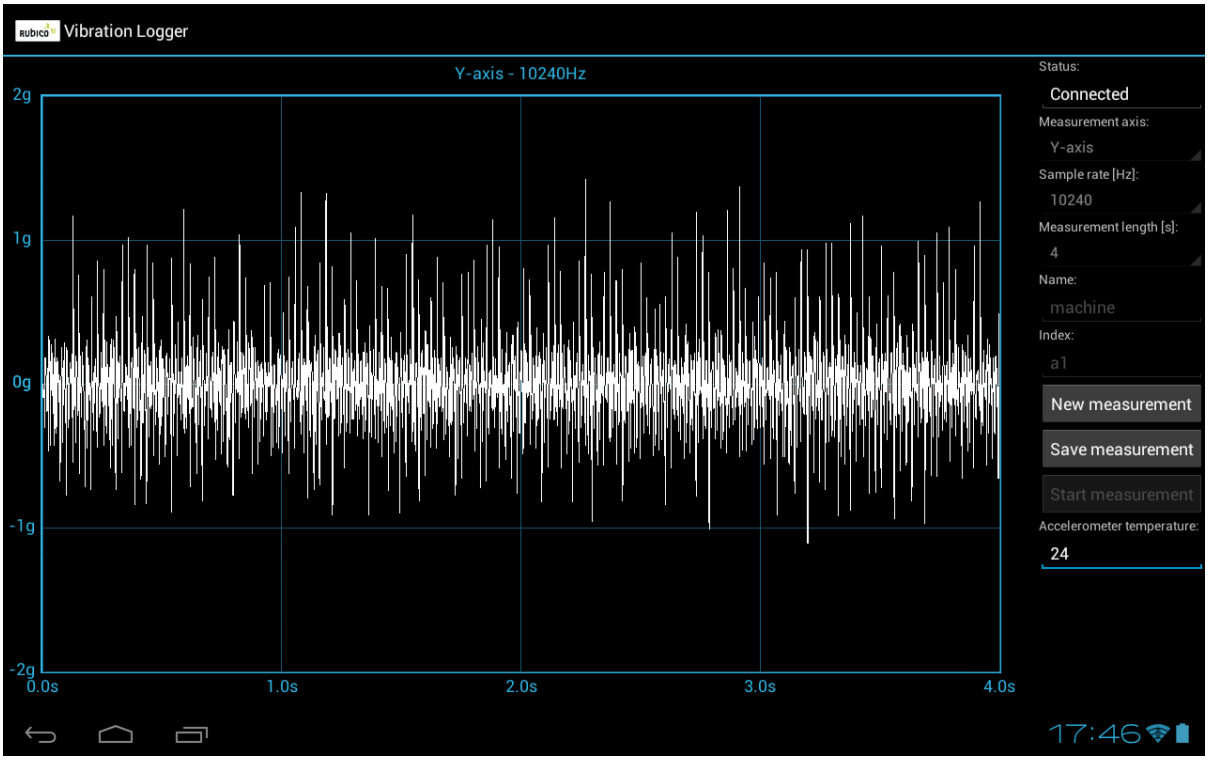

Figure 4.1: The user interface.

4.3.6 USB Communication

The USB API was chosen over the Android NDK and libusb method to keep the software less complex and avoid the need to "root" the Android tablet. The developed application implements the USB class described in Section 4.2.

The bulk transfers, where acceleration data is received from the accelerometer unit, are done asynchronous by a function that queues the requests and let the USB hardware handle the transaction. The synchronous way of performing the bulk transactions are used because it is non-blocking and the other type, synchronously, would require more communication between the application and the USB hardware, thus lower the maximum transmission rate. The bulk transfers are executed in a separate thread to avoid blocking of the interface while receiving data.

4.3.7 Data Storage

A data storage class was developed for saving measurement data to files. The file format was decided to be ASCII data to make it easy to inspect the data using a normal text editor. The files also contain a file header with the following information:

- Date and time when the measurement was done.
- Name and index, if these fields were set from the user interface.
- Measurement axis, sample frequency and number of samples collected.
- Accelerometer temperature and supply voltage.
- Software version of the Android application and the accelerometer unit software.

When the user press the Save measurement button on the interface, a dialog box asking for a file name will be shown. A standard file name is always generated based on the name and index of the measurement, the sample axis and the measurement frequency. The user can change this file name if desired.

4.4 Accelerometer Unit Software

The software for accelerometer unit was the software developed to run on the LPC11U14 microcontroller used in the accelerometer unit. The accelerometer unit software handles USB communication, accelerometer SPI communication and collection of acceleration data during measurements.

4.4.1 Development Tools

The software for the Accelerometer unit was written in the programming language C. To simplify the development the development platform LPCXpresso[5] was used together along with an LPCXpresso LPC11U14 development board[4]. LPCXpresso is a development platform based on Eclipse IDE[3] and supports LPC micro-controllers.

4.4.2 Accelerometer Unit Software Overview

The accelerometer unit software's main functions are:

- Handle accelerometer communication by reading and writing accelerometer registers using the SPI connection.
- Implement the USB class described in Section 4.2. Including the four control transactions on endpoint zero and the bulk transaction for acceleration data.
- Collecting real time data from the accelerometer and forward the data to the Android tablet.

4.4.3 Accelerometer Communication

To manage communication with the accelerometer a library was developed to provide a simple interface to the accelerometer. The library handles configuration of the micro-controller's synchronous serial port (SSP) interface to SPI communication and provides functions for reading and writing accelerometer registers.

Three different communication functions were developed.

- **Single read command**: This function reads a single register from the accelerometer by sending a read command as the first SPI cycle and a dummy command as the second cycle to provide a clock sequence during which the accelerometer returns the register content. This function was used to read single registers such as the temperature and voltage registers.
- **Single write command**: This function writes a single register by performing two write commands. One write command to write the upper byte of the register and one write operation to write the lower byte of the register. This function was used to write single registers for accelerometer settings.
- **Full duplex read**: This function works as the single read command, but instead of a dummy command as the second cycle, it sends another read operation to the register address, thus prepares the next read. This function was used during accelerometer measurements to provide fastest possible accelerometer communication.

4.4.4 nxpUSBlib

Since developing a complete USB driver would be out of scope for this project, an open source library called nxpUSBlib[7] was used. nxpUSBlib is a USB library designed to run on all USB capable LPC micro-controllers from NXP. It aims to be a full-featured USB library supporting USB 2.0, host and device modes, up to high speed transfer rates and control, bulk, interrupt and isochronous transfer types.

When this thesis was carried out, nxpUSBlib had reached version 0.92 beta, which meant that all features of the library were not implemented yet. The library included some demonstration software to show its capabilities. These demonstration softwares were tested and showed enough functionality that, although the library was incomplete, the library would be a good start for the accelerometer unit software.

The accelerometer unit software was developed using nxpUSBlib's demonstration softwares as a starting point. The custom USB class was developed as a part of nxpUSBlib and was included the same way as the demonstration classes.

4.4.5 Control Transactions

All USB communication, except sample data transactions, are done using the zero endpoint. By doing interface requests, nxpUSBlib will let the USB class handle the request. The zero endpoint was used for the simple reason that it was already implemented by nxpUSBlib, since it is used during initialization of the USB device.

When the USB host sends the first two packets in the Setup Stage of a controlling transaction, where the interface is set as the recipient, a class specific function to handle the request is called, see Figure 4.2. The function was designed to recognize the type of request and do one of the following depending on the request:

- **Read Register Request**: The Setup Stage will provide the accelerometer register wished to be read, so the software will first read the corresponding accelerometer register and then clear the Setup Stage. In the Data Stage the content of the accelerometer register will be sent to the host and finally the Status Stage will be cleared by receiving a zero-length data packet from the host.

- **Write Register Request**: The Setup Stage will provide the accelerometer register wished to be written. The software starts by clearing the Setup Stage. The data to be written will be received during the Data Stage. Then the software will write the data to the corresponding accelerometer register, wait for the accelerometer's busy-pin (DIO1) get a low output level, indicating that the accelerometer operation has finished, and then clear the Status Stage by sending a zero-length packet to the host.

- **Start Measurement Request**: The Setup Stage will provide the accelerometer register corresponding to the axis data buffer. The software starts by clearing the Setup Stage. In the Data Stage a packet containing the number of samples to collect in the measurement is received. The Status Stage will be cleared by sending a zero-length packet to the host. The number of samples received in the Data Stage will set a "to send" variable used for the measurement. Finally a pin change interrupt will be enabled to initiate the real-time data collection.

- **Get information Request**: The temperature and voltage of the accelerometer will be put in a packet together with the software version of the accelerometer unit software and a status report for the measurement. The Setup Stage will then be clear. In the Data Stage the prepared information packet will be sent to the host and finally the Status Stage is cleared by receiving a zero-length packet from the host.

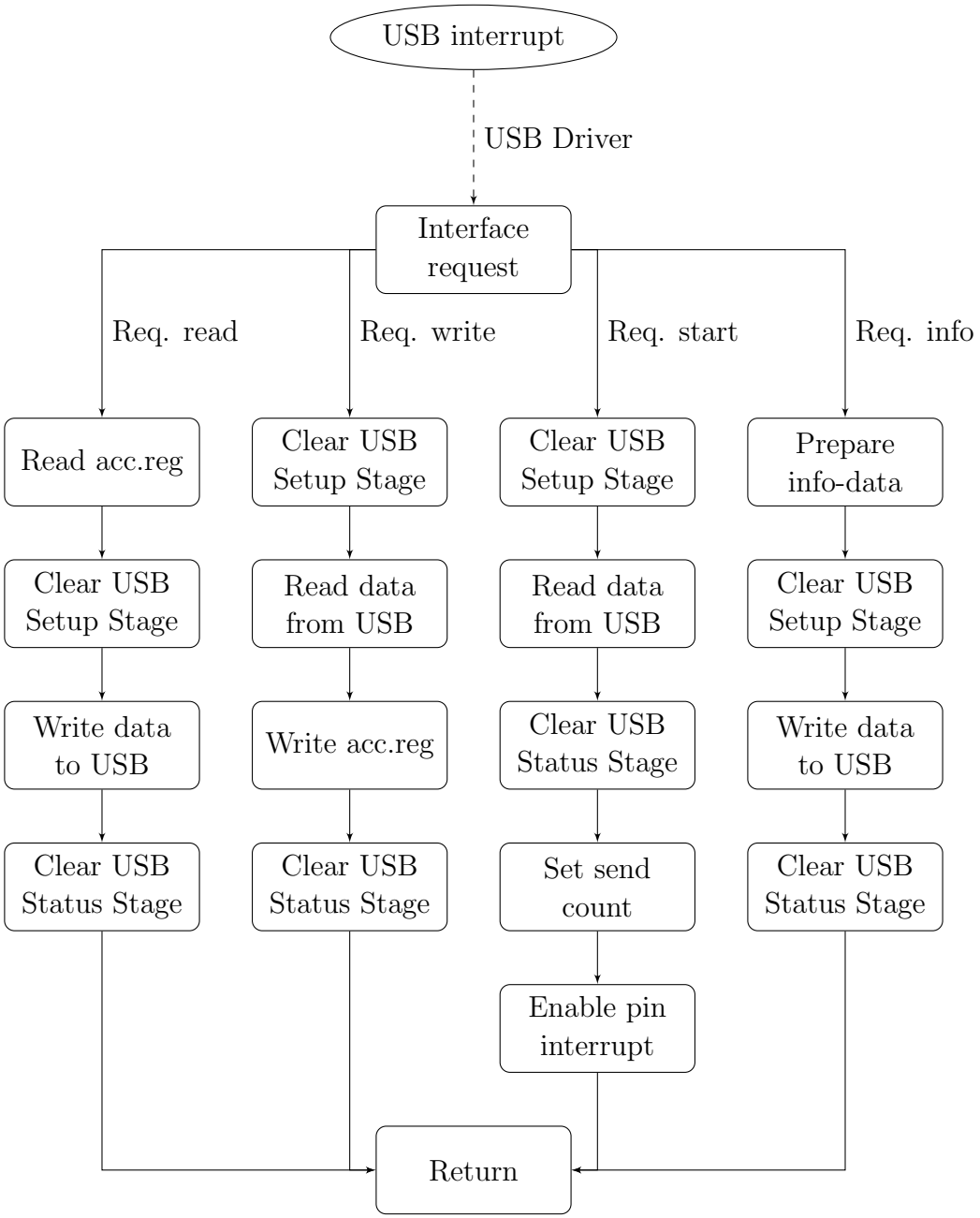

Figure 4.2: Flow chart for USB interrupt.

4.4.6 Collecting Real Time Data

When the ADIS16228 accelerometer is configured to real-time mode, it will sample one axis and provide acceleration data at the sample rate set during configuration. The acceleration data can be accessed by reading the corresponding axis register using the SPI

interface. To inform the device responsible to read data, in our case the microcontroller, that a new sample value is ready to be read, the acccelerometer pin DIO1 will go from a high to low output level.

By enabling a falling edge interrupt for the microcontroller pin connected to the accelerometer's DIO1 pin, an interrupt service routine (ISR) will be called whenever the accelerometer reports that a new sample value is ready to be read. The ISR was designed according to Figure 4.3.

When the accelerometer measurement was started, using the start measurement request sent to the zero endpoint, a "to send" variable was set to the number of samples to collect. As long as the value of this variable is more than zero the ISR will read the new sample value from the accelerometer using the full duplex read function described in Section 4.4.3.

Since the USB class was designed to send the sample data by bulk transactions with the size 64 bytes and each sample value is two bytes, the bulk transactions will contain 32 samples. This meant that the software will have to make a bulk transaction every 32:nd sample value read. Because of this the samples are first written to a buffer and each time the buffer is full (32 sample values) the accelerometer unit will send the content of the buffer as a bulk transaction. When a bulk transaction has finished the buffer will be reseted and the send count will be decreased by 32.

When the send count, the number of samples left to send, is zero, the pin change interrupt will be disabled. This is done so that the ISR will not run for the upcoming accelerometer operations and destroy the software flow. Then the accelerometer temperature and voltage is read from their corresponding registers and finally the accelerometer will be reset to make sure that everything is set to normal.

Overhead Problem

The normal flow of nxpUSBlib, when you want to do an IN bulk transaction, is to call a write-function that takes a pointer to a data buffer and the length of the buffer as arguments. nxpUSBlib will use the buffer pointer and buffer length and copy the data to a reserved part of the memory and then instruct the USB controller to send the data. Unfortunately this operation will create a lot of overhead and takes over 100 μs to finish. Since the higher sample rates produce samples at a shorter interval, this would mean that we miss data samples during bulk transactions.

This problem was solved by using the part of the memory, reserved for USB transaction data, as buffer and thus removing the need to copy the complete buffer as one chunk. A modified write function, that skips the buffer-copying and only instructs the

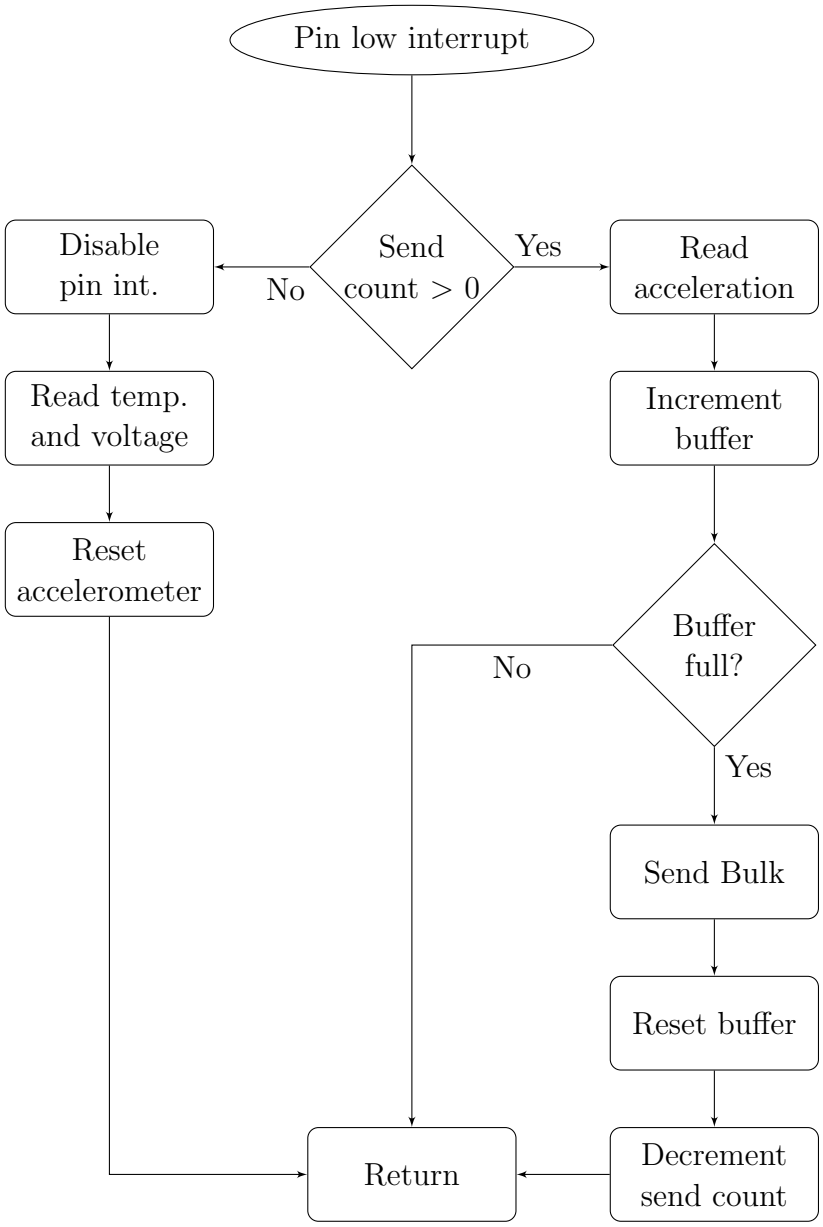

Figure 4.3: Flow chart for pin interrupt.

USB controller to start sending the data, was also added to nxpUSBlib.

Buffer Over-Write Problem

One thing needed to take into account was that bulk transaction will not happen instantaneously. There will be a delay between when the USB controller is instructed do send data and when the transaction finishes. When using the technique described in previous

section together with a high sample rate there is a possibility that the buffer would be over-written with new sample values before the bulk transaction was finished.

This problem was solved by adding a double buffer system where the buffers are used alternately. First buffer A is written to and send, then buffer B is written to and send, thus the pin change interrupt function and the USB hardware never operates on the same buffer at the same time.

Missed Bulk Transactions

The accelerometer unit has a limited time frame to send the bulk transactions, due to the continuous stream of new samples throughout the measurement. Because of this the software does not have time to retry a failed bulk transaction. A missed bulk transaction would mean a loss of 32 acceleration samples and this would be non-tolerable.

A missed bulk transaction would most likely happen if the accelerometer unit tries to send the DATA packet, in the bulk transaction, before the IN packet is received from the host. Even if this scenario never was experienced in the final version of the software, a warning flag is set if this would happen. The status flag is then set in the status flag register sent to the host in the information request following the measurement.

Chapter 5

Test Measurements

This chapter will describe and present the results of test measurements performed in a test rig using the developed measurement system.

5.1 Test Rig

To evaluate the developed measurement system (foremost the accelerometer unit case design) tests were performed using a test rig. The rig was consisting of a electrical motor that was driving a shaft via a belt transmission. The shaft was fixed to the rig by two permanent ball bearings. The rig offered the ability to add a third ball bearing to the shaft, on to which a load could be applied.

5.2 Measurements

A defect ball bearing, with an introduced defect on the inner race, was used for the test measurements. Three measurements were then carried out with the following setups.

- **Test 1**: The ADIS16228 glued directly to the test rig without the accelerometer case.
- **Test 2**: The assembled accelerometer unit screwed to the test rig.
- **Test 3**: The assembled accelerometer unit magnetically mounted to the test rig.

5.3 Measurement Results

Figures 5.1 and 5.2 shows the power spectral density (PSD) for the observed signal. The PSD plots show the power distribution over the frequencies in the signal. By comparing the PSD plots between the different signals the following observations were made:

- The highest power levels were found in the 0-1kHz frequency band for all three test cases. The maximum power level was 25-35dB higher in this band compared to the other frequencies.
- By comparing the highest levels between the plots a decrease of about 5dB and 10dB can be observed for the test cases with the screwed and magnetically mounted accelerometer unit, compared to test case with the accelerometer glued to the rig.
- A smoothing effect can be observed in the 3-5kHz band for test case 3 where the power level decreases with about 10dB.

The interesting frequency band in this test is 0-1kHz. By comparing the magnetically mounted accelerometer unit (test case 3) to the other two, we notice that the frequency response differ and that the maximum power levels are lower in this band. However, the power levels are still about 30 dB higher in this band than the other frequencies and would still be valid for the purpose of this system.

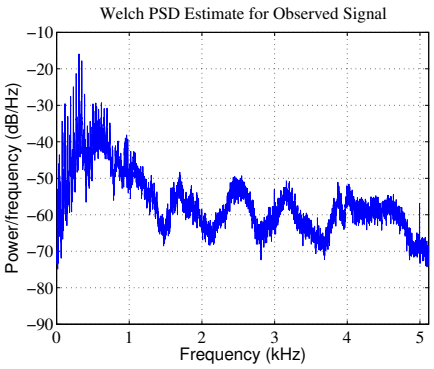

Figure 5.1: PSD for test case 1.

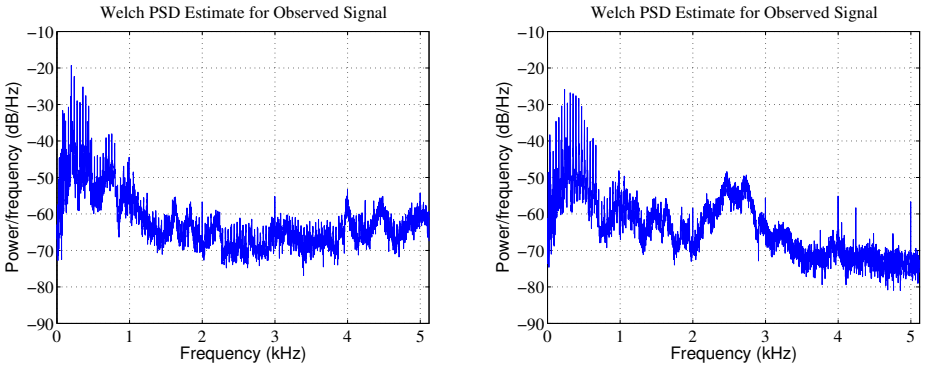

Figure 5.2: PSD for test case 2 (to the left) and test case 3 (to the right).

Chapter 6

Conclusions

In this chapter the results of the thesis will be compared to the aims stated in the introduction along with an evaluation of some of the design choices made during the development.

The aim of this thesis was to design a prototype for a handheld vibration measurement system. The measurement system was supposed to be based on an Android platform and a sensor unit that was to be developed. The system was required to allow the user to setup measurement settings, add additional information about the measurement, view measurement data and save measurement data to a file from a user interface on the Android platform. The system was also to be adapted for industrial environments.

The result of the thesis fulfills all the requirements stated above. The system is based on an Motorola Xoom Android tablet that can be connected to an accelerometer unit via USB. The tablet is equipped with a protective case and the accelerometer unit is a closed case that shields its components from the surrounding environment.

The user can configure the measurement settings and give the measurement a name and an index from the user interface. After each measurement the acceleration data is plotted on the Android screen to allow visual inspection of the data. If the user wishes to save the measured acceleration data this can also be done from the user interface.

From measurements performed on a test rig, the accelerometer unit case was evaluated. The results proved satisfactory and showed that the signals measured using the developed system would meet the requirements of this system.

Android has proven to be a good choice for the platform. The development tools of-

fered by the Android SDK has proven to be a big help during the application development. The USB API offered an easy interface to the USB bus and the accelerometer unit.

One downside was that it turned out that the USB API, used to establish a connection to the accelerometer unit, was not activated on all Android tablets. This problem might limit the number of choices for a future change of platform to an industrially adapted Android tablet.

A beta version of the open source library nxpUSBlib was used in the microcontroller in the accelerometer unit. nxpUSBlib offered a good starting point for the software since developing a complete USB-driver would be out of scope for the thesis. Thanks to the modifications described in Section 4.4.6 sufficient transfer rates were reached for the USB communication to forward acceleration data at the highest sample rate offered by the accelerometer.

References

[1] Android developers. Available from: http://developer.android.com [cited March 21, 2012].

[2] CadSoft EAGLE. http://www.cadsoftusa.com/eagle-pcb-design-software/.

[3] Eclipse. http://www.eclipse.org/.

[4] LPC11U14 LPCXpresso Board. http://www.embeddedartists.com/products/lpcxpresso/lpc11U14_xpr.php.

[5] LPCXpresso. Available from: http://ics.nxp.com/lpcxpresso/ [cited March 27, 2012].

[6] MOTODEV Studio for Android. http://developer.motorola.com/.

[7] nxpUSBlib. Available from: http://www.lpcware.com/content/project/nxpusblib [cited March 27, 2012].

[8] Siemens NX. http://www.plm.automation.siemens.com/en_us/products/nx/.

[9] Building an enterprise-class Android device from the ground up, October 2011. Available from: www.motorola.com/web/Business/Products/Tablets/ET1/_Documents/_StaticFiles/ET1-Tech-Brief-Ground-Up.pdf [cited March 23, 2012].

[10] No MotoBlur for Motorola Xoom, pure Android 3.0 Honeycomb device, January 2011. Available from: http://www.intomobile.com/2011/01/21/motorola-xoom-android-motoblur/ [cited March 27, 2012].

[11] Panasonic toughbook to address market void by delivering enterprise-grade android tablet, June 2011. Available from: www.panasonic.com/business/toughbook/downloads/CM111_Launch_Press_Release.pdf [cited March 23, 2012].

[12] Inc Analog Devices. *ADIS16228 Digital Triaxial Vibration Sensor with FFT Analysis and Storage*.

[13] Manuel Di Cerbo and Andreas Rudolf. Using android in industrial automation. Technical report, University of Applied Sciences Northwestern Switzerland for the Institute of Automation, 2010.

[14] Inc. Dytran Instruments. Accelerometer Mounting Considerations. Available from: `www.dytran.com/img/tech/a8.pdf` [cited March 27, 2012].

[15] N. Tandon and A. Choudhury. A review of vibration and acoustic measurement methods for the detection of defects in rolling element bearings. *Elsevier Science Ltd.*, 1999.

[16] Inc USB-IF. *On-The-Go and Embedded Host Supplement to the USB Revision 2.0 Specification*. Available from: `http://www.usb.org/developers/docs/usb_20_101111.zip` [cited March 27, 2012].

[17] Inc USB-IF. *Universal Serial Bus Specification, Revision 2.0*. Available from: `http://www.usb.org/developers/docs/usb_20_101111.zip` [cited March 27, 2012].

[18] Peter van der Linden. Code to Launch Foam Missiles Over USB!, August 2011. Available from: `http://community.developer.motorola.com/t5/MOTODEV-Blog/Code-to-Launch-Foam-Missiles-Over-USB/ba-p/17889` [cited March 27, 2012].