



ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO DE TELECOMUNICACIÓN

Título del proyecto:

DESIGN AND IMPLEMENTATION OF THE OPERATING
SOFTWARE FOR THE HARDWARE PROTOTYPE FOR PERSONAL
HEALTH DATA IDENTIFICATION (PHDI)

Elsa Moriones Idiazabal

Supervisor: Dr. Ing. Luis Serrano Arriezu

Examinador 1: DI Ferenc Gerbovics

Examinador 2: Dr. Stefan Sauermann

Viena, 04/05/2012

MASTER'S THESIS

for an academic degree
"Master of Science in Engineering"

Design and implementation of the operating software for the hardware prototype for Personal Health Data Identification (PHDI)

written by

Elsa Moriones Idiazabal
1190, Wien, Hutweidengasse, 35/8/3

Examiner 1: DI Ferenc Gerbovics

Examiner 2: Dr. Stefan Sauermann

Vienna, 04.05.2012



Written at University of Applied Sciences Technikum Wien

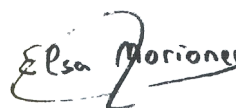
Study Programme, Master Biomedical Engineering

Affidavit

"I hereby declare by oath that I have written this paper myself. Any ideas and concepts taken from other sources either directly or indirectly have been referred to as such. The paper has neither in the same nor similar form been handed in to an examination board, nor has it been published."

Vienna, 04/05/2012

Place, Date

A handwritten signature in black ink that reads "Elsa Moriones". The signature is written in a cursive style with a large, stylized initial 'E'.

Signature

Resumen

El patrón de la población en Europa está cambiando. El aumento del número de ancianos y el consecuente aumento de las enfermedades crónicas deriva en una situación en la que los sistemas sanitarios deben encontrar nuevos métodos para el cuidado de las personas enfermas. La telemonitorización está ganando importancia para proveer la solución a estos cambios. Además, una estrategia de prevención para evitar la aparición de enfermedades crónicas consiste en animar a la población a practicar más ejercicio físico. Como consecuencia existe una gran variedad de aparatos de telemonitorización de las constantes vitales durante la práctica deportiva.

El PHDI (Identificador personal de datos médicos) permite que el empleo de dispositivos de salud personal (PHD) sea más fácil y cómodo. Este dispositivo proporciona un método innovador para relacionar a cada persona con sus mediciones de datos sanitarios. El PHDI se asigna a una persona que emplea PHDs. Esta persona identifica el PHD que va a utilizar mediante RFID. El PHDI se encarga de transmitir el identificador del PHD empleado a un servidor mediante la tecnología ANT. Las medidas del PHD se transmiten directamente al servidor, donde son asignadas al usuario en cuestión. Este producto da solución a situaciones en las que los PHDs son empleados por varias personas. Así mismo, puede ser beneficioso para residencias de ancianos y centros deportivos.

Abstract

The pattern of population in Europe is changing. The increasing number of elderly people and the consequent increase of chronic diseases drive to a situation where health care systems have to find a new way to take care of ill people. Telemonitoring is gaining importance to provide a solution for these changes. In addition, a preventive strategy against chronic diseases consists of encouraging people to practice more physical exercise. Consequently, there is a wide variety of fitness and health telemonitoring devices. The PHDI makes easier and more comfortable the use of personal health devices (PHD). This device provides an innovative way to relate each person with their health data measurements. The PHDI is assigned to a person who uses PHDs. This person has to identify the PHD via RFID before using the medical device. The PHDI transmits the ID of the used PHD to a server environment via ANT technology. The measurements from the PHD are transmitted directly to the server environment, where they are assigned to the correct ID. This product provides a solution for situations where PHD are shared by many people. Nursing homes and fitness centers can benefit from this product.

Keywords: Telemonitoring, personal health device, health, data identification, software

Acknowledgements

I would like to thank

my project leader Dipl.-Ing Ferenc Gerbovics for his support and advices,

the members of the project team, Gregor Wiktorin, Xin Lu, Jiliang Wang and Mona Janbozorgi, for their work making this project possible,

Adrián Catón Oteiza, for his support, his encourage and the design of the PHDI hardware prototype,

and finally to my family and friends, for their encourage and support during all these years.

Table of Contents

1 Introduction.....	1
1.1 Motivation.....	2
1.2 Scope of Work/Requirements.....	3
1.3 PHDI hardware prototype.....	5
1.4 Development tools.....	7
1.4.1 MPLAB Integrated Development Environment (IDE)	7
1.4.2 PIC32 Starter Kit board.....	7
1.4.3 PIC32 I/O Expansion Board.....	8
1.4.4 PICtail Daughter Board for SD and MMC Cards.....	9
1.4.5 Speech Playback PICtail Plus Daughter Board.....	9
2 Methods and material	10
2.1 Audio interface.....	10
2.1.1 SD-Card.....	10
2.1.2 WAV/WAVE (WAVEform) audio file format	13
2.1.3 Audio player.....	15
2.1.4 Volume control audio amplifier.....	17
2.2 RFID and the SkyTek Protocol.....	18
2.2.1 The Request message.....	18
2.2.2 The Response message.....	21
2.3 ANT	23
2.3.1 ANT interface.....	23
2.3.2 ANT Data Types.....	25
2.3.3 ANT channel configuration.....	27
2.3.4 Channel configuration messages.....	30
2.3.5 Auto Shared Channel.....	33
2.3.6 ANTware II and ANT USB stick.....	37
2.4 Accelerometer and gyroscope.....	37
2.4.1 Programmable Interrupts.....	38

2.4.2 Register Map.....	39
2.4.3 Fall detection.....	40
3 Results	44
3.1 Audio interface.....	44
3.1.1 Audio play function.....	44
3.2 RFID.....	45
3.2.1 Initialization.....	47
3.2.2 Request for the Tag ID.....	48
3.2.3 Process the response.....	49
3.2.4 RFID and UART disabling.....	52
3.2.5 Power problems.....	52
3.3 ANT.....	52
3.3.1 ANT interface.....	53
3.3.2 Initialization	55
3.3.3 Channel Configuration.....	56
3.3.4 Data messages.....	60
3.3.5 Getting a unique identifier.....	62
3.3.6 Handshaking.....	63
3.3.7 ANT data exchange.....	65
3.4 MPU 6050 and fall detector.....	67
3.4.1 Sensors calibration.....	67
3.4.2 Interrupts configuration.....	69
3.4.3 Fall detection.....	70
4 Discussion.....	72
Bibliography.....	74
List of Figures.....	77
List of Tables.....	79
List of Abbreviations.....	81

1 Introduction

In the last century, Europe has experienced an increase in life expectancy. Since 1840, life expectancy has increased by a quarter of a year each year and the predictions say it will continue growing this way. [1]

This change in life expectancy combined with low levels of fertility for decades is responsible for the aging of the population. According to Eurostat 2008-based population projections, the population pyramid in EU-27 in 2060 will be as illustrated by the figure 1. [2]

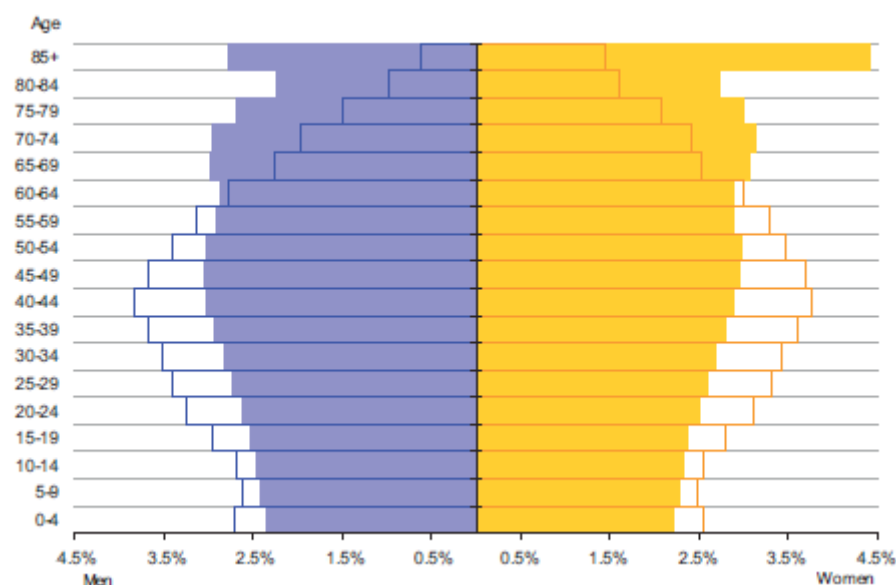


Figure 1: Population pyramids EU-27, 2010 and 2060 [2]

The projections show the population pyramid will be narrower in the middle and the base. These parts of the pyramid correspond to the working age (from 20 to 64) and the younger people respectively. In contrast, the top of the pyramid corresponding to people over 65 will be wider. The proportion of people aged 65 years or over will increase from 17.4% to 30% of the total of the EU-27 population by 2060. [2] This demographic change is affecting not only Europe but also other industrial countries. The proportion of people aged 65 or over in the total of the OECD (Organisation for Economic Cooperation and Development) countries will change from 17% in 2005 to 20.7% in 2020).

The absolute number of elderly people in Europe is growing as well. For the EU 25, the population aged 65 or above is predicted to change from 75.3 million in 2004 to 134.5 million in 2050.

Health problems increase considerably with age. According to studies of Eurostat, in 2009, a 68.1% of the population of the EU-27 aged 16 or more claimed to be in good health.

However, more than half of the population aged from 65 to 74 claimed to have a long-standing illness or health problem. (See figure 2.) [3]

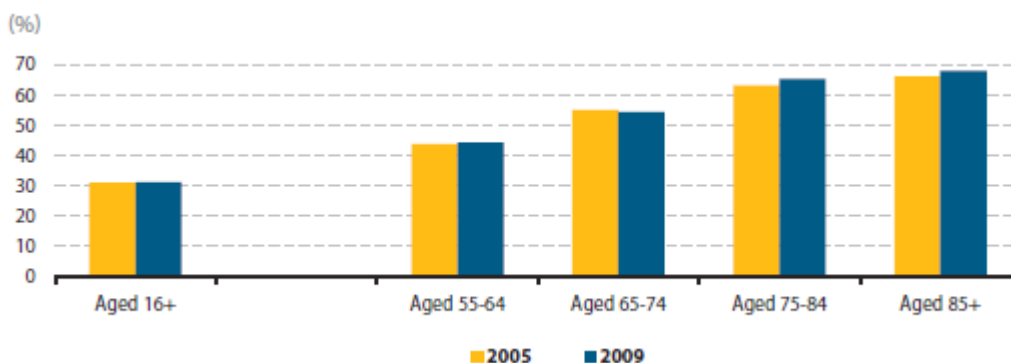


Figure 2: Proportion of the people with a long-standing illness or health problem, EU-27 [3]

The demographic changes lead to an increase in the demand for health and long-term care. Health and social services have to be adapted to the new patterns of illness as the number of chronic diseases will increase. [4]

Certain programs such as Ambient Assisted Living (AAL) try to provide a solution to health demand. *“AAL aims to prolongate the time people can live in a decent way in their own home by increasing their autonomy and self-confidence, the discharge of monotonously everyday activities, to monitor and care for the elderly or ill person, to enhance the security and to save resources. New technologies - especially in combination with new services and caring organisations -offer enormous opportunities for improving the living standard at home”.* [1]

Physical inactivity is a major risk for people suffering from certain chronic diseases such as type 2 diabetes, cardiovascular diseases, obesity, hypertension and some cancers. [5] Hence, the new health and social services should emphasize on preventive strategies with more physical and mental activity and with a healthier nutrition. [6] In addition, it is important to monitor vital constants while exercising. [7]

1.1 Motivation

Telemonitoring provides the solution to the demographic changes and the consequent increase of chronic diseases. *“Many people prefer home care to any other option. Home is a place of emotional and physical associations, memories and comfort. Although many people can be happy in assisted-living facilities, retirement communities or nursing homes – and for many people these are better options – leaving home can be disruptive and depressing for some people”.*[8] Nowadays, telemonitoring allows to decrease the costs in the health system reducing the hospitalization of patients. [9] [10]

There is a wide variety of health care and fitness monitoring devices that allow to monitor some vital parameters. However, sometimes they are too difficult to handle for elderly people. [1] There is a need of a user-friendly device which enhances the use of personal health devices (PHD). In a situation where many people share some health or fitness monitoring devices it would be desirable to be able to save every measurement and to assign them to the correspondent user. Even though some monitoring devices have a memory to store some measurements, this is not sufficient in a situation where there is more than one user. It would be convenient to be able distinguish the measurements of each user. Moreover, it is desirable to store the measurements and access them with an easy-to-use tablet or mobile application. This could be used in nursing homes as well as in fitness centers or simply in a house where somebody uses PHDs.

Considering this necessity, the innovative project of the Personal Health Data Identifier (PHDI) was proposed by the Biomedical engineering department of the University of Applied Sciences Technikum Vienna. The PHDI has been designed as a wrist wearable device. The main target of the device is to work as an identifier of PHD users. The PHDI allows to assign each user their health data measured by the PHD. An Android Tablet takes care of collecting the measurements. This can be achieved with minimum intervention of the user. The measurements are stored in a data base and are accessible to the user.

Furthermore, to take advantage of being a comfortable wearable device, the PHDI has some modules that allow to implement additional applications. For instance, a desirable application for the elderly could be a fall detector to warn relatives if there is a fall.

1.2 Scope of Work/Requirements

The main target of this master thesis is to deal with the design and implementation of a software model for the hardware prototype for identification of personal health data by using RFID technology and ANT technology. The PHDI will be assigned to a person. This person will identify personal health devices via RFID before using the medical device for the conduction of measurements. The PHDI will transmit the ID of the used PHD to an Android tablet via ANT technology. The measurements from the PHD will be transmitted directly to the Android Tablet, where they can be assigned to the correct ID. A speech output is used to give the user feedback about the success or failure of the communication. Finally, a message using the HL7 (Health Level Seven) protocol and containing the measurements will be sent from the Android Tablet to the server via WiFi.

The figure 3 shows the different communications between devices that take place in the project.

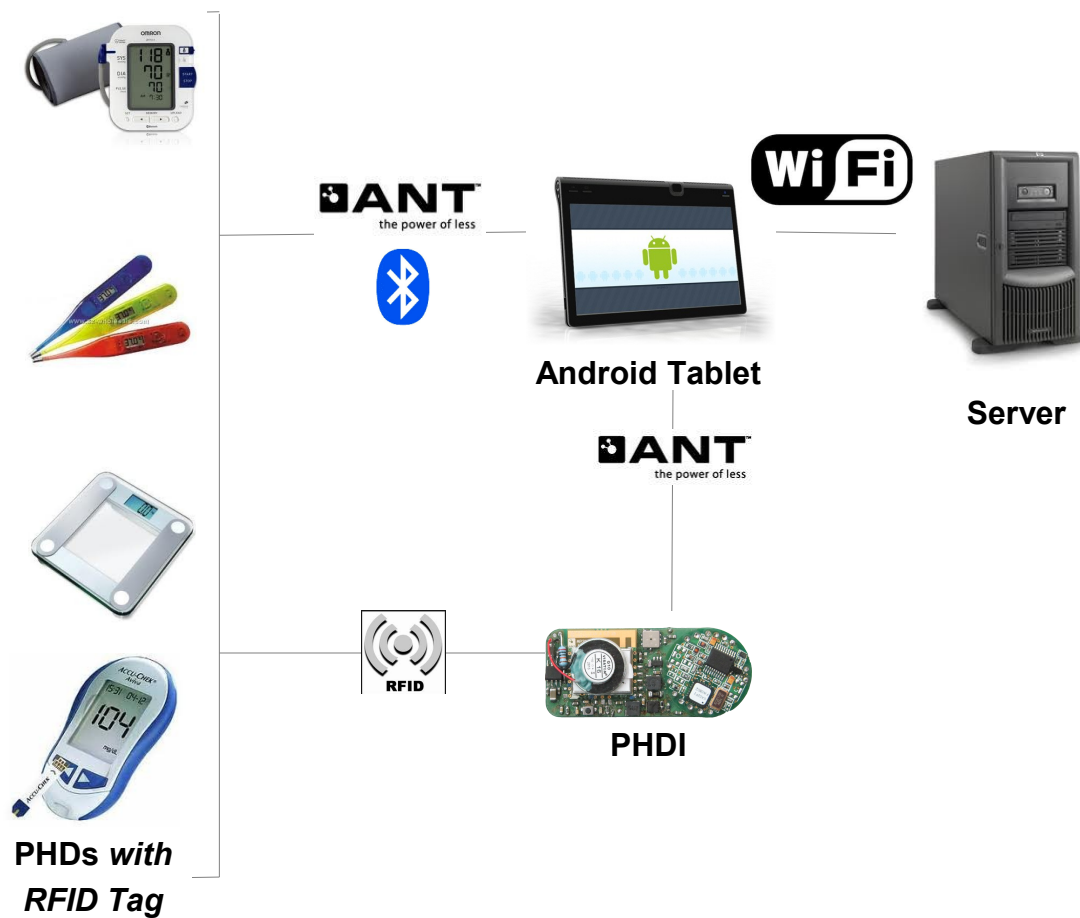


Figure 3: PHDI Infrastructure Overview

The PHDI is a useful device for fitness centers, nursing homes or homes where personal health devices are used. In these places, each person will use one PHDI, with a unique identifier that allows to associate each measurement with the owner. In addition, it is important to ensure the coexistence of the devices. More than one PHDI should be able to communicate with the Android tablet at the same time.

There are some additional application that can be interesting depending of the users. Anyone can fall down but the consequence of a fall can be fatal for the elderly. Thus, a fall detector could be implemented for this sector of the population. For the development of this application, the PHDI has been designed containing numerous sensors such as an accelerometer, an altimeter and a gyroscope.

To achieve these requirements, the software developed should be capable of fulfilling the following tasks:

- Read the RFID tag placed on the personal health device
- Send the ID of the used PHD to an Android tablet via ANT
- Interact with the user via speech output

- Assure coexistence with more PHDIs
- Creation of a unique ID for each PHDI
- Handle data read from the accelerometer
- Handle data read from the gyroscope
- Handle data read from the altimeter [11]
- Detect a fall using the data obtained with the different sensors
- Send data to the Android tablet

1.3 PHDI hardware prototype

The PHDI is a wrist wearable device. The main target is to work as an identifier of the user. Therefore a RFID module and an ANT module are necessary. Moreover, the device has some

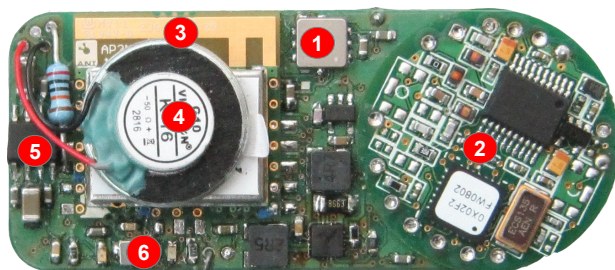


Figure 5: PHDI front side

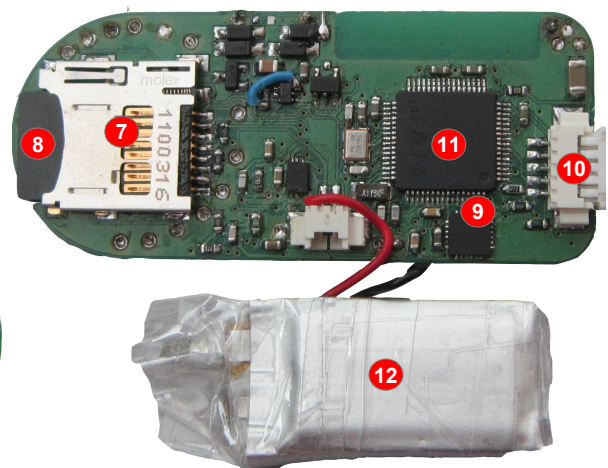


Figure 4: PHDI back side

sensors that allow to monitor the user movements. The front side and the back side of the hardware prototype and its different modules are shown in the figures 5 and 4.

As a wrist wearable device, the PHDI has to be small, comfortable and lightweight. It is also desirable to have a low battery consumption in order to keep working without charging during a long time. The design of the PHDI hardware prototype is faced by Adrián Catón and is explained in the following reference. [9] The different modules of the PHDI are explained below:

1. Altimeter BMP085 [12]. It is a digital pressure sensor from Bosch. This altimeter allows to measure from 300 hPa to 1100 hPa. The sensor is also capable of measuring temperature.

2. SkyeModule M1-mini RFID module [13]: It is a RFID read and write radio module with an internal antenna, compatible with most standard 13.65MHz RFID Tags. The module is small and has a low power consumption.
3. ANT module ANTAP281M4IB: This is a module from Dynastream Innovations Inc. It is an ultra low power consumption module that allows a bidirectional wireless communication using the ANT protocol. The module works in the 2.4GHz ISM band.
4. Speaker 716-6453: This speaker from Visaton was selected due to its small size. The speaker allows to inform the user about the success or failure of the operation of the PHDI.
5. Audio amplifier TDA 8551[14]: It is an amplifier from NXP Semiconductors. The module has a digital volume control.
6. A button is placed to let the user interact with the PHDI. On both sides of the button there is a LED. These LEDs are used to give information about the operation of the device.
7. SD slot used to insert the Micro SD Card.
8. Micro SDHC-Card: This is a 4GB card used to store the audio files and some data exchanged with the Android Tablet.
9. Accelerometer and gyroscope MPU 6050 [15]: This is a sensor from InvenSense Inc. The module contains a 3-axis accelerometer and a 3-axis gyroscope. The accelerometer can measure from -16g to 16g. The gyroscope can measure values from -2000°/sec to 2000°/sec. The module also includes a temperature sensor.
10. The ICSP (In-Circuit Serial Programming) is used for programming and debugging the PHDI.
11. Microcontroller PIC32MX795F512H [16]. This is a 32 bit microcontroller from Microchip Technology Inc. The maximum frequency is 80MHz. The microcontroller contains some peripherals such as six UART modules, four SPI modules, five I²C modules, five 16-bit timers and one real time clock and calendar (RTCC).
12. Battery of 100 mAh with a small size.

The file HardwareProfile.h defines labels to abstract the programmer from the hardware connections between the MCU and the modules. To see this in more detail please refer to the following reference [11].

1.4 Development tools

The software design of the PHDI is made in parallel with the Hardware design of the PHDI. Thus, at the beginning of the implementation of the Software it is important to find a way to test the software during the development. Therefore, a tool like the PIC (Peripheral Interface Controller) 32 Starter Kit is very useful. Another additional boards will provide the possibility to test some functionalities of the PHDI such as the audio interface. The important tools used are listed below:

- MPLAB IDE
- ICD 3
- PIC32 Starter Kit board
- PIC32 I/O expansion board
- Speech Playback PICtail Plus Daughter Board, Data Sheet
- PICtail Daughter Board for SD and MMC Cards

1.4.1 MPLAB Integrated Development Environment (IDE)

MPLAB IDE is a free, integrated toolset for the development of embedded applications employing Microchip's PIC microcontrollers. The program has hardware debugging and programming tools. It supports the MPLAB ICD 3 In-Circuit Debugger, which allows to program or debug at a high speed.

The IDE debugging tool allows to set breakpoints at nearly every part of the program and allows to see the value of the different variables. It is also possible to jump into a desired part of the program. [17]

The programming language used for the Software development is C. As a result, a C compiler is also needed. At first, the compiler used was the MPLAB C compiler for PIC32 v2.02. Finally, the version used is the v1.12 due to problems with exceptions in the other version.

1.4.2 PIC32 Starter Kit board

The PIC32 Starter Kit board is a USB powered board. The board includes the following parts:

1. PIC32MX360F512L microcontroller

2. Power indicator LED
3. 3.3V power supply
4. Crystal 8MHz
5. USB connector for on-board debugger communications and power supply
6. PIC18LF4550 microcontroller for debugging and programming
7. Debug indicator LED
8. 3 switches
9. 3 LEDs
10. Connector for an expansion board [18]

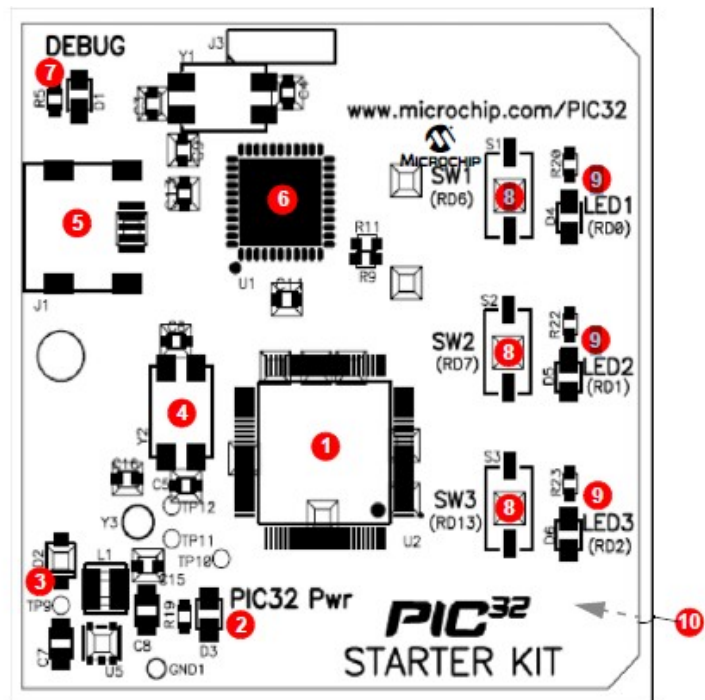


Figure 6: PIC32 Starter Kit demo board layout [18]

1.4.3 PIC32 I/O Expansion Board

The PIC32 Starter Kit Board can be placed in the PIC 32 I/O Expansion board. See figure 7. This board has different sockets and pin headers. The PICtail Plus socket allows to connect different Daughter Boards such as the Daughter Board of the SD-Card and the Daughter Board of the speaker that will be explained below. The expansion board provides access to the MCU Signals via test points. The ICD3 debugger can be connected to the board using the RJ12 connector.

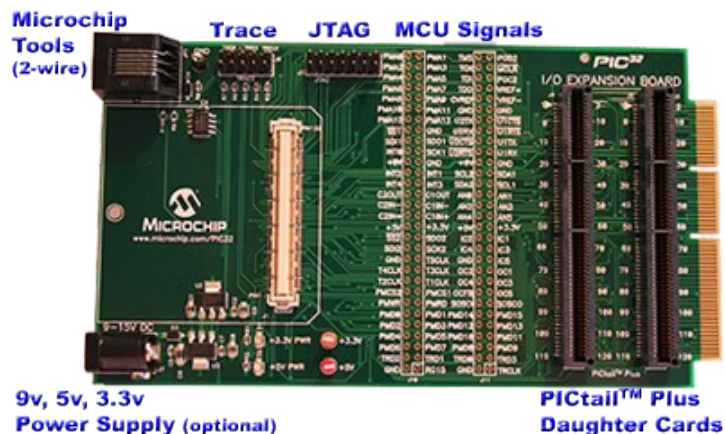


Figure 7: I/O Expansion Board [19]

The Starter Kit board can provide power to the I/O Expansion Board and vice versa. If additional power is required, an optional power supply of 9V, 5V or 3.3V can be connected. [19]

1.4.4 PICtail Daughter Board for SD and MMC Cards

A Secure Digital (SD) Card is used to store audio files. In order to be able to use the SD Card or a Multi Media Card (MMC), a socket is necessary. In the first stage of the development of the software the PICtail Daughter Board for SD and MMC Cards is used. This is a demonstration board for evaluating reading and writing data on SD or MMC cards. It is compatible with the PICtail Plus interface of the Expansion Board.

1.4.5 Speech Playback PICtail Plus Daughter Board

The PIC 32 starter kit does not have an audio module. Then, it is necessary to use another board for playing the audio files. The one that is used for the software development is the Speech Playback PICtail Plus Daughter Board. This board fits into the expansion slot of the PIC32 Starter Kit. The board contains a potentiometer for volume control, a forth order Low Pass Filter, a speaker with mono output, an amplifier with shutdown control and a EEPROM (Electrically Erasable Programmable Read-Only Memory) for voice storage. [20]



Figure 8: Speech Playback PICtail Plus Daughter Board

2 Methods and material

2.1 Audio interface

“Any human interface can be greatly enhanced by using sound to provide feedback, to capture the attention of the user with alerts and error messages, or to enhance the user experience.” [21] The PHDI has to interact with the user via speech output in order to give feedback of the success or failure of the operation. A SD-Card is used to store the audio files and a audio module to play the file.

2.1.1 SD-Card

The Figure 9 shows the interconnections between the PICtail Daughter Board for SD Card and the MCU (Microcontroller Unit). [22]

The default mode of communication between the SD card and the microcontroller is the SD bus mode. However, it is possible to change the communication protocol to Serial Peripheral Interface (SPI). To do so, the card has to be selected (this is done with the pin CS) and a command of reset has to be sent. To configure the SPI, the pins for the SDO and the SCK have to be set as outputs. WD and CD have to be configured as inputs. [21]

The steps to initialize the SD Card and read and write data from the SD Card are summarized below:

Initialization

1. The CS line is initially high (the card is not selected).
2. Wait for more than 74 clock pulses for the card to become capable of receiving commands.
3. Select the card.

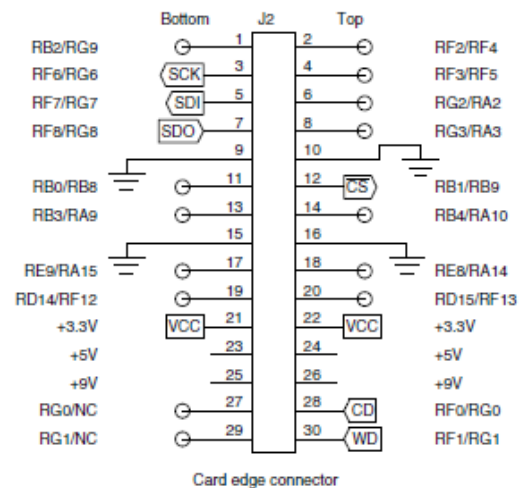


Figure 9: Connections of the PICtail Daughter Board for SD and MMC Cards [22]

4. Send the RESET (CMD0) command. The card should respond by entering the Idle state and activating the SPI mode.
5. Send the INIT (CMD1) until the card exits the Idle state.

Reading data

1. Send a READ_SINGLE command and wait for the SD card to respond with a DATA_START . Set a timeout of less than one millisecond.
2. Once the DATA_START token is received, read in a rapid sequence all 512 bytes composing the requested block of data. They will be followed by a 16-bit Cyclic Redundancy Check (CRC) value that can be discarded.
4. Deselect the memory card and terminate the entire read command sequence.

Writing data

1. Send a WRITE_SINGLE command and check the SD card response to make sure that the command is accepted.
2. Send the DATA_START token and immediately after it, in a short loop, all 512 bytes of data.
3. Send 2 bytes for the CRC (any value since the CRC check is not enabled in SPI mode)
4. Check the SD card response. The token DATA_ACCEPT will confirm that the entire block of data has been received and the write operation has started.
5. Wait for the completion of the write command. While the card is busy writing, it keeps the SDO line low. Wait for the SDO line to return high. A timeout must be set to limit the amount of time to complete the operation (100 ms).

[21]

SC Card types

There are different sizes of SD Cards. In the first stage of the development, a normal SD Card is used. However, in the final prototype a microSD card is employed. Fortunately, the only difference between them is the size, logically and electrically they are identical. [21]

Moreover, there are different types of SD cards depending on the capacity. The SD standard is used for cards up to 2GByte and this type of cards use FAT 12 or FAT16 file system. The capacity for the Secure Digital High Capacity (SDHC) Cards goes from 2GB to 32GB and the

FAT 32 file system is employed. Finally, the Secure Digital Extended Capacity (SDXC) standard is used in cards from 32GB up to 2TB and the exFAT file system is used. [23]

The card employed in the prototype is a 4GB card with the standard SDHC. Unfortunately, there are some differences between the standards. Therefore, there are some changes in the initialization procedure indicated above. [24]

FAT file system module

The FAT (File Allocation Table) file system is a method for storing and organizing computer files and the data they contain. This file system was created by Microsoft. There are many versions of the FAT file system: FAT12, FAT16 and FAT32. FAT 16 and FAT 32 are supported by almost every operating system and most of the mass storage devices. [21]

Fortunately, there are FAT file system software modules already implemented. The FatFs module used in the prototype can be found in the following reference [25]. The module is free software and it is generic, independent from the hardware architecture.

Only some functions are important for the software development. These functions are summarized below:

- `f_open`: This function opens an existing file or creates a new file. The file can be opened in write or read mode.
- `f_read`: This function allows to read an specified number of bytes from a file and store them in a buffer.
- `f_write`: The function writes a given number of bytes into the desired file.
- `f_lseek`: This function moves the file read/write pointer.
- `f_mount`: This function registers/unregisters a work area to the FatFs module.
- `f_close`: This function closes the opened file.
- `f_unlink`: The function removes a file or a directory.

Disk I/O interface

Since the FatFs module is separated from disk I/O layer, it requires some functions to access the physical media. It is necessary a disk I/O interface. However the low level disk I/O

module is not a part of FatFs module. It must be provided by user. The I/O module contains the following functions:

- `disk_initialize`: This function initializes the disk drive.
- `disk_status`: This function returns the current disk status (not initialized, no medium in the drive or write protected)
- `disk_read`: This function reads sectors from the disk drive.
- `disk_write`: This function writes sectors to the disk.
- `disk_ioctl`: This function controls device specified features and miscellaneous functions other than disk read/write.
- `get_fattime`: This function gets current time.

[25]

A free software implemented by Microchip contains these functions. Thus, it is used to read and write from the SDHC Card.

2.1.2 WAV/WAVE (WAVEform) audio file format

The format chosen for the audio files is the Microsoft and IBM WAVE format. This format is compatible with almost any audio application and it is often the default lossless destination format for extracting files from a music CD. [21]

The WAVE files contain different chunks. The header is the Resource Interchange File Format (RIFF) chunk which specifies that the file is a RIFF file of type WAV. The second chunk is the format chunk which contains some different fields with important information in order to play the audio file. The last important chunk is the Data Chunk.

Wave file header – RIFF Chunk

The different fields of the file header are explained in the table 1:

Offset	Size	Description	Value
0x00	4	Chunk ID	“RIFF” (0x52494646)
0x04	4	Chunk Data Size	File size - 8
0x08	4	RIFF Type	“WAVE” (0x57415645)

Table 1: RIFF chunk content [26]

Format chunk

The different fields found in the format chunk are explained in the table 2:

Offset	Size	Description	Value	Explanation
0x00	4	Chunk ID	0x666D7420	"fmt"
0x04	4	Chunk size	16+ extra format bytes	Size of the standard wave format data plus the size of any extra format bytes
0x08	2	Compression code	Unsigned int	Compression code of the wave data
0x0a	2	Number of channels	Unsigned int	Number of separate audio signals encoded in the wave data chunk
0x0c	4	Sample rate	Unsigned long	Number of samples per second that has been used to record the data. The number of channel doesn't affect this field.
0x10	4	Average bytes per second	Unsigned long	Number of bytes of data per second that must be streamed to a D/A converter in order to play the audio file.
0x14	2	Block align	Unsigned int	Number of bytes per sample. This number is affected by the number of channels. The formula 1 shows this dependence [26]
0x16	2	Significant bits per sample	Unsigned int	This number is not affected by the number of channels.
0x18	2	Extra format bytes	Unsigned int	Number of additional bytes that come afterwards. This field doesn't exist if there aren't any extra format Bytes.

Table 2: The fmt chunk content [26] [21]

$$BlockAlign = \frac{SignificantBitsPerSample}{8} NumberChannels \quad (1)$$

Data chunk

After all of these fields there could be other chunks containing additional information about the file. Therefore, it is important to scan the file until the data chunk is found.

As shown in the table 3, it is easy to know where the data chunk starts because the first 4 Bytes of it are the word DATA with ASCII characters. The next four Bytes of the data chunk specify the size of the data. Afterwards, the data can be read. [26]

Offset	Length	Description	Value
0x00	4	Chunk ID	“data“ (0x064617461)
0x04	4	Chunk size	It depends on sample length and compression
0x08	Sample data		

Table 3: Data chunk format [26]

2.1.3 Audio player

Once the audio files can be read from the SDHC Card, the audio module has to be used to play the sound. For this purpose, the Pulse Width Modulation (PWM) is used to conditioning the audio signal.

PWM

The speaker plays analogue signals. Since the signal obtained from the SDHC Card is digital, a Digital-to-Analog (D/A) converter is required. The PHDI requires to play audio files containing synthesized speech. Consequently, the device does not need high resolution D/A, a simple and cheap D/A converter can be generated using the PWM and an analogue Low Pass Filter (LPF). [27]

“The idea behind realizing digital-to-analog (D/A) output from a PWM signal is to analog low-pass filter the PWM output to remove most of the high frequency components, ideally leaving only the D.C. Component.” [28]

A PWM signal has a fixed base frequency and a variable width. A timer is used to set the frequency of the signal that can be controlled with the period register. The pulse width is programmable and it can vary between 0 and 100 percent of the timer period. The variation of the width is directly proportional to the amplitude of the audio signal.

A Fourier analysis of the PWM signal shows three different parts (see figure 10): A DC component, a peak in the fundamental frequency and an infinite number of harmonics with a frequency multiple of the fundamental.

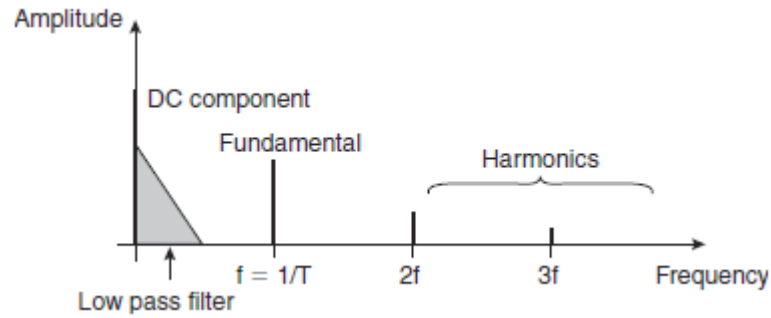


Figure 10: Frequency spectrum of a PWM signal [21]

The DC component is directly proportional to the duty cycle of the PWM signal and therefore proportional to the amplitude of the original audio signal. A LPF is used to eliminate the peaks which are undesired noise and get only the desired original signal converted to analog.[21]

The Output Compare (OC) Module of the microcontroller can work as a PWM modulator and it can generate a continuous stream of pulses with the desired duty cycle. The Output Compare Module can be configured using the OpenOCx function from the Peripheral library [29].

It is important to choose the correct value for the frequency of the PWM signal and to calculate the optimal configuration for the timer. The value of the Period Register (PR) of the timer depends on the sample rate of the audio file and on the peripheral bus clock frequency. This dependence is shown in the formula 2

$$PRx = \frac{FPB}{\text{samplerate}} - 1 \quad (2)$$

[21]

The selection of the PWM frequency influences the DAC resolution. This resolution depends on the duty cycle resolution and on the harmonic ripple. There is a trade off between the duty cycle resolution and the harmonic ripple. On one hand, reducing the PWM frequency allows to increase the duty cycle resolution. On the other hand increasing the PWM frequency allows to reduce the harmonic ripple. [30] It is very important to consider this in order to select the sample rate of the audio file that will influence the PWM frequency. The figure 11 shows the dependence of the DAC resolution with the PWM frequency using different peripheral bus clocks and using a certain first order LPF.

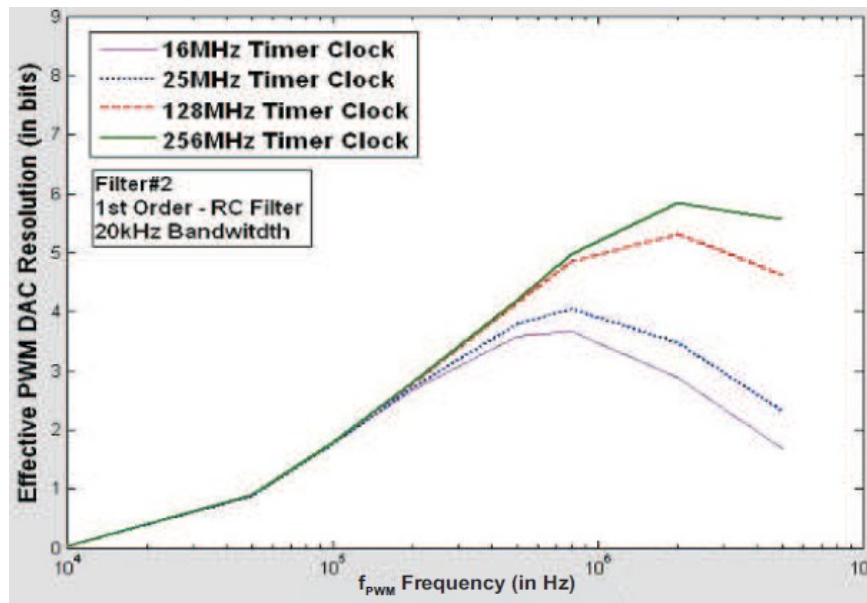


Figure 11: DAC resolution vs PWM frequency [30]

Finally, the samples are filtered. In the first stage of the development, the output of the OC Module of the Starter Kit is plugged in the input of the Audio Board. The signal goes through a 4th order Low Pass Filter to the speaker. In the PHDI prototype, the output signal of the OC Module goes to the first order LPF. This filter has a theoretical cutoff frequency of 4.81KHz. The filtered signal goes to the audio amplifier. Finally, the amplified analog signal goes to the speaker.

2.1.4 Volume control audio amplifier

The audio amplifier has a pin to control the volume. This pin could be controlled by external buttons or without external component. As the PHDI only has one button, the volume is

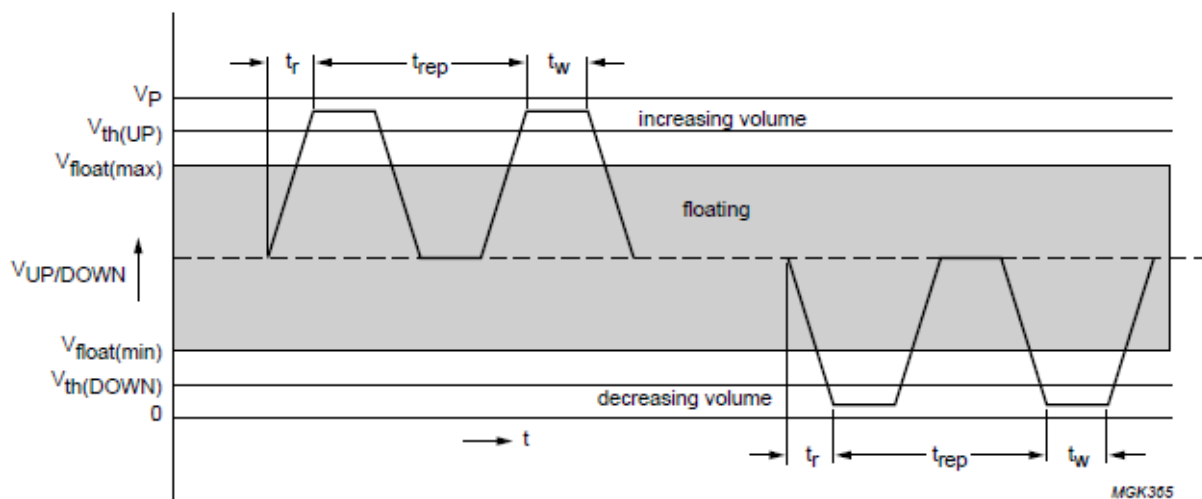


Figure 12: Volume control [14]

controlled by Software. “The volume control circuit responds to the trailing edge of the pulse on the volume pin” [14] Connecting the pin to V_p results in an increase of the volume and connecting it to ground results in a decrease of the volume. This is illustrated in figure 12.

The pulse repetition time (t_{rep}) has to be set to a value of at least 100 ns. “The rise time (t_r) and the width of the pulse (t_w) are not critical”.[14]

2.2 RFID and the SkyeTek Protocol

The RFID module has different possibilities for the host interface connection. The three possibilities are Transistor Transistor Logic (TTL) Serial, SPI and I²C. This choice is made by the hardware designer.[12]. The interface used for this connection is the Universal Asynchronous Receiver Transmitter (UART).

The RFID module allows to read the ID of a RFID Tag attached to a PHD. This tag is passive and obtains its energy from the reader. [31] The study of the SkyeTek protocol will be focused on achieving this purpose.

The SkyeTek protocol describes the format of the messages exchanged between the SkyeTek RFID module and the host. First of all, the host has to start with the request-response sequence and define if the format used is ASCII or Binary. For simplicity the used format is Binary. Consequently, the study of the protocol implementation is focused on this format.

Every binary message starts with a Start of Transmission (STX) field with the value 0x02. Moreover, there are two different kinds of messages with different fields: the Request and the Response.

2.2.1 The Request message.

The request message is sent by the host. This message has two different field as shown in table 4. These fields are the STX and the request.

Message:	STX	REQUEST
Number of bits:	8 bits	Unknown number of bits

Table 4: Request message sent by the host [32]

Meanwhile, the request field is divided into many different fields as can be found in the table 5. Note that the coloured fields are optionals while the rest of the fields are mandatory. Only the used fields are explained below.

REQUEST										
MSG LEN	FLAGS	COMMAND	RID	TAG TYPE	TID	AFI	STRATING BLOCK	NUMBER OF BLOCKS	DATA	CRC
8 bits	8 bits	8 bits	8 bits	8 bits	64 bits	8 bits	8 bits	8 bits	n*8 bits	16 bits

Table 5: Fields of a request in the binary format [32]

MSG LEN

MSG LEN is the message length in number of bytes. All fields are taking to account to calculate the length except for the MSG LEN and the STX fields.

FLAGS

FLAGS: The different flags are summarized in the table 6. Only the used flags are explained, the rest are set to zero.

FLAGS							
RID_F	TID_F	CRC_F	AFI_F	RF_F	LOCK_F	INV_F	LOOP_F
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Table 6: Flags of the binary request message [32]

- CRC_F indicates the presence (value 1) or absence (value 0) of the CRC field in the request.
- If the value of LOOP_F is 1 it repeats the tag command and gives a response for every pass. Otherwise, if the value is 0 there is a single execution of the command and one response that can be pass or fail.

COMMAND

The command field sets the command type and the target of the request. The different bits of the command are specified in the table 7

COMMAND Type				COMMAND Target			
Reserved (set to 0)	Write_Bit	Read_Bit	Set_Bit	Reserved (set to 0)	Tag_Bit	Sys_Bit	Mem_Bit
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Table 7: COMMAND field [32]

The only important command for the project is the SELECT_TAG, with the value 0x14.

TAG TYPE

This field is present when there is a Tag Command. The TAG TYPE specifies the code of the RFID Tag Type which will communicate with the radio module. It is also possible to set the code as “auto” if the Type of Tag is unknown (value 0x00).

CRC

This field is mandatory in the Binary mode. Therefore, the CRC_F flag must be set to 1. The CRC is calculated including all the fields except for the STX field.

The CRC calculation uses the polynomial $x^{16}+x^{12}+x^5+1=0x8408$. The C language Implementation of the CRC calculation is given in the following paragraph:

```
1 // *dataP is a pointer to the byte array over which the CRC
  is to be calculated
2 // n is the number of bytes in the array pointed to by
  *dataP
3 //
4 unsigned int crc16(unsigned char *dataP, unsigned char n)
5 {
6     unsigned char i, j; // byte counter, bit counter
7     unsigned int crc_16; // calculation
8     crc_16 = 0x0000; // PRESET value
9     for (i = 0; i < n; i++) // check each byte in the array
10    {
11        crc_16 ^= *dataP++; //
12        for (j = 0; j < 8; j++) // test each bit in the byte
13        {
14            if(crc_16 & 0x0001 ) //
15            {
16                crc_16 >>= 1;
17                crc_16 ^= 0x8408; // x^16 + x^12 + x^5 + 1
18            }
19            else
20            {
21                crc_16 >>= 1;
22            }
23        }
24    }
25    return( crc_16 ); // returns calculated crc (16 bits)
26 }
```

[32]

2.2.2 The Response message

After receiving the Request from the host, the radio module sends a Response message.

Message:	STX	RESPONSE
Number of bits:	8 bits	Unknown number of bits

Table 8: Response message sent by the radio module

The response field is divided into different fields. Some fields are optional and will or won't appear depending on the flags of the table 6 and on the RESPONSE CODE field. The fields of the response message are summarized in the table below.

RESPONSE					
MSG LEN	RESPONSE CODE	RID	TAG TYPE	RESPONSE DATA	CRC
8 bits	8 bits	8 bits	8 bits	n * 8 bits	16 bits

Table 9: Binary Response Message [32]

The optional fields appear with a colour in the table 9. Only the important fields are explained.

MSG LEN

The message length specifies the number of bytes without including the STX and the MSG LEN field.

RESPONSE CODE

The response code field appears in every response message. This code can report errors in the content of the message, and the success or failure of a request.

Response code	Description
0x14	SELECT_TAG pass
0x1C	SELECT_TAG LOOP activate
0x94	SELECT_TAG fail
0x9C	SELECT_TAG_LOOP cancel
0x21	READ_MEM pass
0x22	READ_SYS pass
0x24	READ_TAG pass
0xA1	READ_MEM fail
0xA2	READ_SYS fail

Response code	Description
0xA4	READ_TAG fail
0x32	EVENT_report
0xC2	EVENT_error
0x41	WRITE_MEM pass
0x42	WRITE_SYS pass
0x44	WRITE_TAG pass
0xC1	WRITE_MEM fail
0xC2	WRITE_SYS fail
0xC4	WRITE_TAG fail
0x80	Non ASCII character in request
0x81	BAD CRC
0x82	FLAGS do not match COMMAND
0x83	FLAGS do not match TAG TYPE
0x84	Unknown COMMAND
0x85	Unknown TAG TYPE
0x86	Invalid STARTING BLOCK
0x87	Invalid NUMBER OF BLOCKS
0x88	Invalid Message Length

Table 10: Response codes [32]

TAG TYPE

This field is present if the request has a SELECT TAG code with TAG_TYPE set to AUTO (TAG_TYPE=0x00).

RESPONSE DATA

This field is present when the response contains data requested by the host, for example the Tag ID.

CRC

The CRC of the response is calculated the same way as in the request.

2.3 ANT

ANT is a wireless protocol running on the 2.4GHz ISM band and it is designed for ultra low power applications. The protocol can handle peer-to peer, star, tree and practical mesh topologies. Moreover, it provides a reliable way of communication and a flexible and adaptive network. It also provides immunity to cross-talk.

The ANT communication protocol takes care of four of the OSI (Open Systems Interconnection) model layers: Physical, Data Link, Network and Transport layers. (See figure 13)

- Physical layer: Physical transmission of the bits.
- Data link layer: Secure data transmission using frames. This layer takes care of the synchronization and some errors and flow control.
- Network layer: Responsible for establishing, maintaining and closing connections.
- Transport layer: Provides security, data transmission between the end points, error control and flow control.



Figure 13:OSI layers [33]

[33]

The protocol contains a key low-level security which allows to implement a secure network.

[34]

The ANT technology is available in different formats. The one used in this project is an ANT module (ANTAP281M4IB). This module is a certification ready PCB, that incorporates an ANT chipset. It can be mounted onto an existing PCB with an immediate integration. [35]

2.3.1 ANT interface

There are two possible interfaces between the ANT and the Host MCU, Asynchronous serial interface and Synchronous serial interface.[34] The interface used in this prototype is the asynchronous Serial interface.

The UART configuration settings are: 1 start bit, one stop bit, 8 bits of data and no parity. The baud rate is selected by hardware to 50000 bauds. [34]

Reset and initialization

The ANT chipset is ready for communication after the initial power-up phase. There is an RESET pin available to clear the configuration, close all the channels and go back to the default values. This reset pin is active low. [34]

Flow control

When the ANT is configured in asynchronous mode, there is a flow control mechanism for data sent from the Host to the ANT module. This flow control is performed by the RTS signal which is available in the RTS pin. The procedure of the flow control is shown in the figure 14

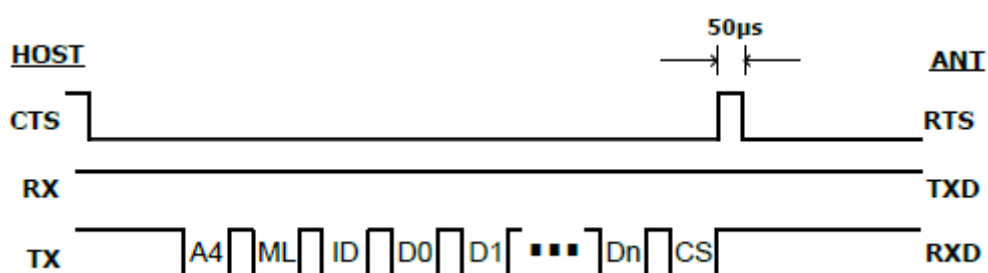


Figure 14: Flow control with a RTS signal in a transfer from Host to ANT [34]

“The RTS signal is de-asserted for approximately 50µs after each correctly formatted message has been received”. When ANT raises the RTS signal, the host MCU should wait until the signal is low again to send more data. Otherwise, the message will be wrong or incomplete. There is no flow control in the opposite direction. Thus, the host must be able to receive data at any time. [34]

Power conservation

There are two signals that allow to activate a low power state and terminate all RF and serial port activity. The signals are SLEEP and SUSPEND.

In order to enter the suspend mode, the SLEEP signal must be raised before asserting the SUSPEND signal. Moreover, to exit the suspend mode and go to the power-up state, both signals, the SLEEP and the SUSPEND must be de-asserted. This is shown in more detail in the figure 15.

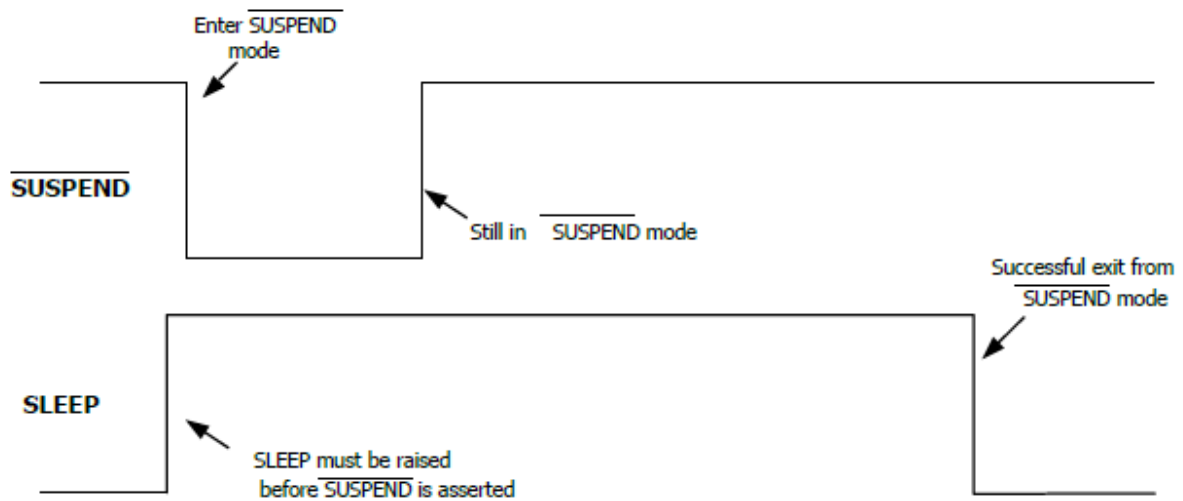


Figure 15: Suspend mode control [34]

Coming out from the suspend mode will automatically trigger a reset of the ANT chip.

Message structure

A typical serial message between the host and the ANT module has the format of figure 16.



Figure 16: ANT Serial Message Structure[35]

- Sync: Is a 1-Byte field with a fixed value of 0xA4 or 0xA5
- Message length: Is a 1-Byte field which specifies the number of data bytes in the message (N)
- Message ID: Is a 1-Byte field with the message identifier.
- Data: N data bytes
- Checksum: 1 Byte field specifying the XOR of all previous bytes including the SYNC byte.

2.3.2 ANT Data Types

There are three data types supported by ANT: Broadcast Data, Acknowledged Data and Burst Data.

Broadcast data is sent from the master to the slave in every channel timeslot and from the slave to the master only if expressly requested by the slave's MCU. Broadcast data is not acknowledged. Therefore, the originating node will not notice the possible loses. This type of data consumes the least amount of RF bandwidth and system power. Thus, it is the best for communications where some occasional loss can be tolerated. The value of the Message ID field to specify Broadcast data is 0x4E.

The Acknowledged Data is a good choice for applications that can not have any loss. This kind of data can be sent in any direction, from master to slave or from slave to master. The node that receives the acknowledged data has to respond with an acknowledgement message if the acknowledged data was received correctly. In the case of a failure, there is no automatic retransmission of the unacknowledged packet. The value of the Message ID field that specifies the data type as Acknowledged is the 0x4F.

Burst data allows to send large amounts of data between devices. In this data type there is also a notification for the success or failure of the data transfer. However, in this case, the notification is for the entire burst. Notice that bursting can create interference for other devices operating in the same RF frequency. The value of the Message ID field that specifies the data type as Burst is the 0x50 [35]

The format of the the Broadcast and the Acknowledge messages is shown in the table 11.

Sync	Len	Msg ID	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Check sum
0xA4	9	0x4E	Channel number	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	XOR
0xA4	9	0x4F	Channel number	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	XOR

Table 11: Format of a Broadcast and a Acknowledge message [35]

When one of those messages are sent, a response is received to express the success or failure. This message is a Channel Event and is shown in the table 12.

Parameters	Bytes	Description
Channel Number	Byte 1	Channel number of the channel associated with the event
Message ID	Byte 2	ID of the message being respond to. Set this to 1 for an RF event.
Message Code	Byte 3	Code for a specific response or event.

Table 12: Channel Response event message [35]

For the broadcast and acknowledge messages, the message ID will be set to 1 because it is a RF event. In addition, the message code will explain the success or failure. The possible message codes are the following:

- EVENT_TX (Value 3): A broadcast message has been transmitted successfully. Note that this does not necessarily mean it has been received correctly.
- EVENT_TRANSFER_TX_COMPLETED (Value 5): An acknowledge data message or a burst transfer has been completed successfully.
- EVENT_TRANSFER_TX_FAILED (Value 6): An acknowledge data message or a burst transfer failed to complete successfully.

[35]

2.3.3 ANT channel configuration

In order to get the communication of two devices, they require a common channel configuration. To configure the channel it is necessary to set some parameters:

- Channel type
- RF Frequency
- Channel ID
- Channel Period
- Network

[35]

Channel Type

The channel type specifies the communication that will occur on the channel. Some possible channel types are shown in the table 13.

Value	Description
0x00	Bidirectional Receive Channel (Slave)
0x10	Bidirectional Transmit Channel (Master)
0x20	Shared Bidirectional Receive Channel (Slave)
0x30	Shared Bidirectional Transmit Channel (Master)
0x40	Receive Only Channel (Slave)
0x50	Transmit Only Channel (Master)

Table 13: ANT Channel Types [35]

In the bidirectional channel, data can flow in both directions, from the master to the slave and from the slave to the master. The master node will primary transmit but it can also receive.

However, in the Transmit/Receive Only channels, data can only flow from master to slave. This type of channel should not be used for applications that requires a confirmation or acknowledgement.

A shared channel is a bidirectional channel used when a single ANT node must receive and process data from several nodes. Multiple slaves share one channel. While in the other channel types the number of simultaneous communications is limited to the number of channels, the use of a shared channel allows to have communication with 255 slaves (if 1 byte of shared address is used) or 65K slaves (if 2 bytes of shared address are used) at the same time. [35]

Since many nodes (PHDIs) will communicate with one single node (Android Tablet), the shared channel is the best choice for this application. Therefore, from now on, the study will be focused in this type of channel.

To set the value of the channel type, the Assign Channel(0x42) function must be used.

RF Frequency

The RF Frequency is an 8-bit field. The frequency can go from 2400MHz to 2524MHz. The value of the field is calculated with the formula 3

$$RFFrequencyVal = \frac{DesiredRFFrequency(MHz) - 2400(MHz)}{1(MHz)} \quad (3)$$

Notice that ANT employs Time Division Multiple Access (TDMA). Therefore, is not necessary to use different RF frequencies to support multiple coexisting channels at the same time. The function used to set this value is the Channel RF Frequency (0x45) [35]

Channel ID

The channel ID contains three different fields: the Transmission Type, Device Type and Device Number. The master has to define its channel ID (every field different from 0), and the slave has to define the channel ID of the master it wants to connect to. In the case of the slave, any of these fields can be set to zero to use it as a wild card and be able to connect to any master that matches the rest of the channel parameters. In the master's case, these fields can be user defined.

The Transmission Type is an 8-bit field to define some characteristics of the transmission (TX). The two least significant bits are used to indicate the presence and size of a shared address field at the beginning of the data payload. A value of 3 in this field means that there is a shared address of two bytes at the beginning of the Data Payload.

The device type is an 8-bit field to specify the device type or class. The most significant bit of this field is a device pairing bit. If this bit is set to zero that means that there is no pairing.

The device number is a 16-bit field that is supposed to be unique for each device. This field is usually related to the serial number of the device.

To configure the channel ID the function SetChannelID(0x51) must be used. [35]

Channel Period

The channel period is a 16-bit field that allows to configure the message rate of the data packets sent by the master. The possible values for the message rate go from 0.5Hz to 200Hz. The value of the field is calculated with the formula 4

$$ChannelPeriodVal = \frac{32768}{MessageRate(Hz)} \quad (4)$$

The default message rate is 4Hz. Notice that the message rate is directly proportional to the power consumption. The function employed to set this parameter is Channel Message Period (0x43) [35]

Network

Two characteristics of the network can be set: The Network Number and the Network Key.

The Network Number is an 8-bit field that identifies the available networks on an ANT device. The default network number 0 is assigned to the Public Network by default. [35]

The Network Key is an 8-byte number that uniquely identifies a network and provides a measure of security and access control. By default every network numbers have the default

public network key. If another key is desired one should contact to the Dynastream company and ask for a key. The function used to perform the key assignment is SetNetworkKey(0x46). [35]

The public network with a public key is available for all ANT developers and is free to use. It is used for product development, prototyping and demonstrations. The ANT managed networks are administered by ANT. This management provides security and integrity to each of the participants. [36]

2.3.4 Channel configuration messages

There are some specific functions to set the channel configuration parameters. The channel Period, Channel Search Timeout, Channel RF Frequency and the Network Key have default values and only require setting if a different value is desired. Otherwise, the rest of the parameters have no default values and the application must set the desired values. [35]

Notice that all the configuration commands return a response to indicate the success or failure of the configuration.

Assign Channel (0x41)

Parameters	Bytes	Description
Channel Number	Byte 0	Channel used for the communication.
Channel Type	Byte 1	Values show in table 13. 0x20: Shared Bidirectional Receive Channel 0x30: Shared Bidirectional Transmit Channel
Network Number	Byte 2	Network address to be used for the channel. Set it to 0 to use the default public network.
Extended assignment	Byte 3	0x01: Background Scanning Channel Enable 0x04: Frequency Agility Enable All other bits reserved

Table 14: Assign Channel message [35]

This message must be the first message of the channel configuration. Once assigned the channel, the rest of the parameters with default values are set to the default values.

Set Channel ID (0x51)

The following message sets the channel ID for the channel specified in the assign channel message.

Parameters	Bytes	Description
Channel Number	Byte 0	Channel used for the communication.
Device Number	Bytes 1 and 2	Little endian, Least Significant Byte (LSB) first. In a slave use 0 to match any device
Device Type MSb Pairing Request	Byte 3, bit 7	Pairing Request. Set it to 1 to activate it.
Device Type ID	Byte 3, bits: 0-6	The device type. In a slave set it to 0 to match any device type
Transmission Type	Byte 4	Transmission type. If it is a shared channel the value indicates the number of bytes of the shared address.

Table 15: Set Channel ID message [35]

The channel ID is an ID that identifies the master. When the slave wants to communicate with the master, it can set one or more of the fields as wildcards. However, if the slave knows the master's Channel ID, it should set it and the pairing bit will be irrelevant.

Note that Transmission Type and the Device Type ID are assigned and regulated by the company, except for the free default network.

Channel Messaging Period (0x43)

The table 16 shows the content of the message that configures the channel messaging period.

Parameters	Bytes	Description
Channel Number	Byte 0	Channel used for the communication.
Messaging Period	Byte 1 and 2	Channel messaging period in seconds * 32768 Maximum messaging period is proximately 2 seconds Default value 8192 (4Hz) Little endian (LSB first)

Table 16: Channel messaging period message [35]

Note the messaging period parameter does not need to match in the slave and the master devices. A slave can configure the messaging period to a lower data rate if it does not want to receive data as fast as it is being transmitted. In this case, the rate must be an integer divisor of the transmission data rate. [35]

Channel Search Timeout (0x44)

Parameters	Bytes	Description
Channel Number	Byte 0	Channel used for the communication.
Search Timeout	Byte 1	Time that the receiver will search for a channel before timing out. Default value: 10 (10*2.5 seconds= 25 seconds) 0: Disable high priority search mode 255: Infinite search timeout

Table 17: Channel Searching Timeout message [35]

This message sets the time that the slave will search for a channel before timing out. The default value is 25 seconds.

Channel RF Frequency

The message sent to configure the channel frequency is shown in the table 18.

Parameters	Bytes	Description
Channel Number	Byte 0	Channel used for the communication.
Channel RF Frequency	Byte 1	Channel Frequency = 2400MHz + Channel RF Default value: 66 (2466MHz)

Table 18: Channel RF Frequency message [35]

Set Network Key(0x46)

The message used to set the network key is shown in the table 19.

Parameters	Bytes	Description
Network Number	Byte 0	The network number
Network key 0-7	Bytes 1 to 8	Network key 0-7

Table 19: Set Network Key message

This command is not required when using a public network. The default public network key is 0. Moreover, if the Set Network Key message is sent with an invalid network key, a RESPONSE_NO_ERROR will be received and the network key will remain unchanged.

Open Channel (0x4B)

After setting all the channel parameters, the channel should be opened. Therefore, the Open Channel message should be sent with only one parameter: the number of the channel to be opened. It is good to ensure that the master channel is opened prior to the slave. [35]

Channel Response Events for Channel Configuration messages

After sending each configuration message, a response is received indicating the success or failure. Thus, a state machine can be configured. This state machine advances the state only when a RESPONSE_NO_ERROR is received for the current command. However, the state machine resends the configuration message if there is an error.

The Channel Response Event has the format specified in the table 12. The RESPONSE_NO_ERROR message has the value 0 and it means the operation was successful.

When a receiving channel has timed out on searching, a channel response is received with the message code EVENT_RX_SEARCH_TIMEOUT (value 1). This means that the search is terminated and the channel is automatically closed.

2.3.5 Auto Shared Channel

In a shared channel there is a master that communicates with different slaves through the same channel. Every slave has a unique shared channel address (1 or 2 Bytes), only known by the master and by the slave itself. [35] In a system where the number of slaves is fixed is very easy for the master to know all the addresses. Nevertheless, if the number of slaves is not fixed, or if it is necessary to add or remove dynamically slaves, the ANT shared channel is not sufficient on its own.

ANT auto shared channel is an extension of an ANT shared channel. This extension provides the shared channel with a method to dynamically and automatically add or remove slave devices to a shared channel. The ANT auto shared channel can be implemented at the application level.

When a new slave wants to participate in an auto shared channel it needs a shared address. In order to get one it is necessary to perform the Handshaking process. The steps of the handshaking are the following:

1. Address available message
2. Request address message
3. Busy acquiring message
4. Confirm acquire message

Note that the explanation below is for a case where a shared address of two bytes is used.

Address available (0xFF)

Byte	0	1	2	3	4	5	6	7
Content	Shared address LSB	Shared address MSB	Message ID	Channel Period Decimation LSB	Channel Period Decimation MSB	Data Time out	Next Address LSB	Next Address MSB
Value	0xFF	0xFF	0xFF	0-65535		0-255	1-65535	

Table 20: Address available message content [37]

The table 20 shows the message sent by the master when it has an available shared addresses. The message is sent as a broadcast message. If a slave wants to receive this message it has to set its shared address as 0xFFFF permanently.

The field of the channel period decimation is a value that the master can suggest to the slave to decimate its channel period in order to save power. However, it is not recommended because of problems of synchronization. The value is specified in seconds * 31768.

The data timeout is the timeout a slave should wait before giving up its shared address. The unit of this value is 2 s.

Finally, the field of the next address specifies the next shared address available for the slave.

Request Address (0xFD)

Byte	0	1	2	3	4	5	6	7
Content	Shared address LSB	Shared address MSB	Message ID	Reserved	Unique ID 0	Unique ID 1	Unique ID 2	Unique ID 3
Value	0xFF	0xFF	0xFD	0	0-0xFF	0-0xFF	0-0xFF	0-0xFF

Table 21: Request address message content [37]

When a slave wants to get a shared address, it sends a request address message (table 21) after receiving the address available message of table 20. This message has to be sent as an acknowledged message. If the slave does not receive the acknowledgement it has to wait a certain time and retry the request. If there is no successful acknowledgement after some retries, the slave has to start the handshaking process again.

In this message, the slave has still the shared address 0xFFFF. Moreover, there is also a unique identifier that uniquely identifies the slave during the handshaking process. The identifier can be a random number generated by the slave. Finally, if the slave receives the successful acknowledgement, it has to change its shared address to the temporary one of 0xFFFFE in order to proceed with the next steps.

Busy Acquiring (0xFE)

Byte	0	1	2	3	4	5	6	7
Content	Shared address LSB	Shared address MSB	Message ID	Reserved	Unique ID 0	Unique ID 1	Unique ID 2	Unique ID 3
Value	0xFE	0xFF	0xFE	0	0-0xFF	0-0xFF	0-0xFF	0-0xFF

Table 22: Busy acquiring message content [37]

After receiving the request address message and sending the acknowledgement, the master sends the message busy acquiring shown in table 22. This message is sent as a broadcast message. In this message the shared address changes to 0xFFFE. The unique ID is the ID of the slave to which the master is assigning the shared address.

Confirm Acquiring (0xFC)

Byte	0	1	2	3	4	5	6	7
Content	Shared address LSB	Shared address MSB	Message ID	Reserved	Unique ID 0	Unique ID 1	Unique ID 2	Unique ID 3
Value	0xFE	0xFF	0xFC	0	0-0xFF	0-0xFF	0-0xFF	0-0xFF

Table 23: Confirm acquiring message content [37]

This is the final step of the handshaking. The confirm acquiring message show in table 23 is sent by the slave to confirm the reception (RX) of the busy acquiring message (table 22). This message has to be sent as an acknowledge message. Once the slave gets the confirmation, it sets its address to the address specified in the field “next address” of the message Address Available (table 20). However, if the slave does not receive a successful acknowledgement, it will return back to the beginning of the handshake and try to acquire a new shared address. [37]

The state machine that implements the handshaking process in the slave is shown in the figure 17.

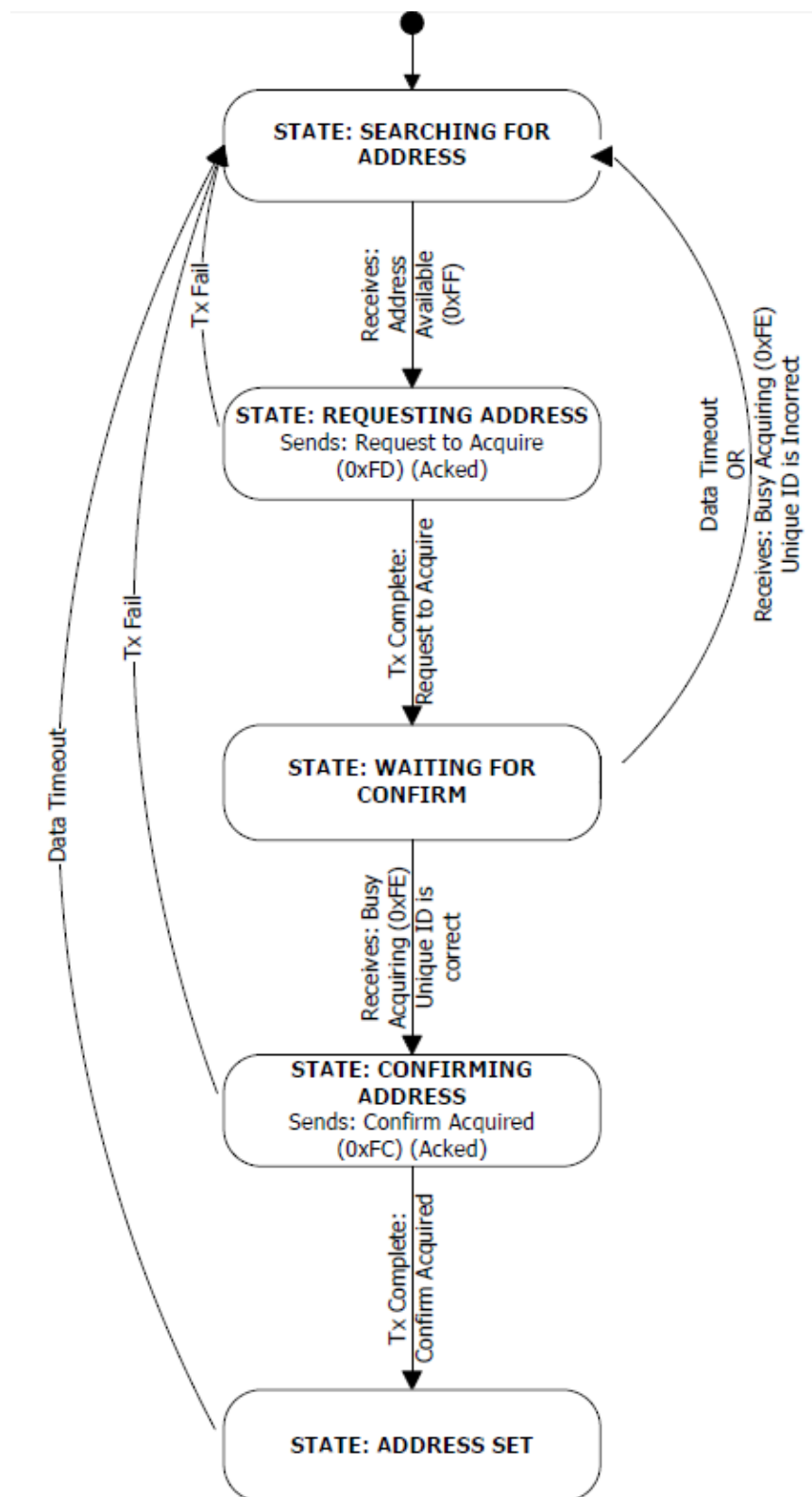


Figure 17: Auto Shared slave state machine

2.3.6 ANTware II and ANT USB stick

The ANT USB stick is a computer dongle that provides connectivity to ANT wireless networks. The ANTware application (figure 18) can be used with the ANT dongle to test the operation of the ANT software of the PHDI. The ANTware tool allows to set the ANT dongle as a master or a slave and to configure the desired channel parameters. Once the channel is configured and opened, the application permits to send the desired Broadcast , ACK or Burst message.

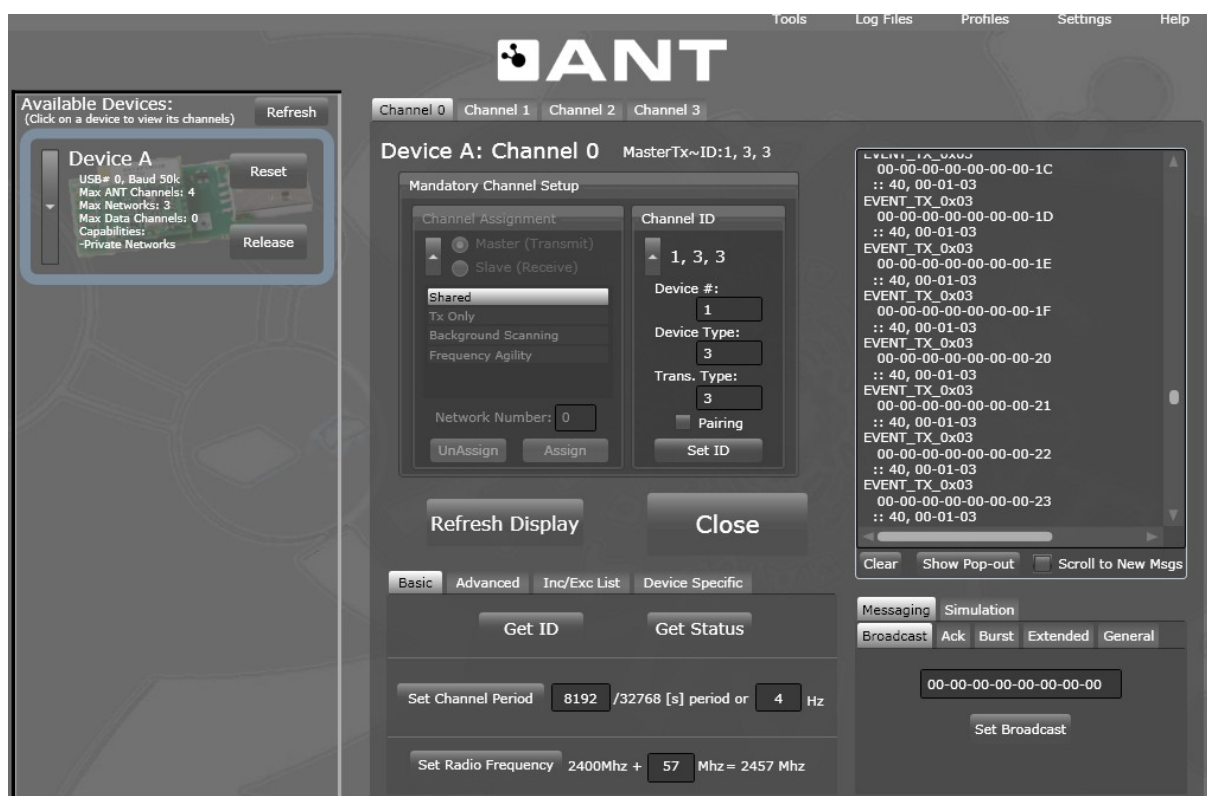


Figure 18: ANTware II development application

The window of the right side shows the events of TX and RX of messages. [36]

2.4 Accelerometer and gyroscope

The MPU 6050 has an embedded 3 axis MEMS (Micro Electro-Mechanical System) gyroscope, 3 axis MEMS accelerometer and a Digital Motion Processor (DMP). The device communicates with the MCU via I²C port.



Figure 19: MPU-6050 [45]

The gyro has a 16-bit Analog-to-Digital-Converter (ADC) for each axis. The full scale range of the digital outputs can be programmed to ± 250 , ± 500 , ± 1000 or ± 2000 degrees per second. The sample rate can be programmed from 8000 samples per second to 3.9 samples per second. There is a programmable digital low pass filter (DLPF) that allows to set the desired cut-off frequency to condition the signal.

The accelerometer has also a 16-bits ADC for each axis. The full scale range can be programmed to $\pm 2g$, $\pm 4g$, $\pm 8g$ or $\pm 16g$. When the device seats on a flat surface, it will measure 0g on the X and Y axes and 1g on the Z axis.

The DMP allows to process the accelerometer and gyro data. Therefore, the DMP offloads computation of motion processing algorithms from the host microcontroller. The DMP has also access to one of the MPU external pins which can be used to generate interrupts. [15]

2.4.1 Programmable Interrupts

The MPU-6050 has a programmable interrupt system that can generate an interrupt in the INT external pin

Free Fall interrupt

The free fall is detected when the absolute value of all the three axes of the accelerometer is below the free fall threshold. The free fall interrupt is triggered when this condition lasts at least for a time specified in the free fall duration register. [15]

Motion interrupt

First of all, the measures of the accelerometer are conditioned with a configurable digital high pass filter. A motion interrupt is triggered when the conditioned sample for any axis has a value exceeding the user programmable motion threshold for a time longer or equal than the one specified in the motion duration threshold. [15]

Zero Motion interrupt

The Zero motion detection also uses a digital high pass filter for the measures of the accelerometer. A zero motion is detected when each axis has an absolute value below the value specified in the zero motion threshold. An interrupt is triggered when this condition remains for a time longer than the value specified in the zero motion duration threshold. Moreover, another zero motion interrupt is triggered when the zero motion is no longer detected. [15]

2.4.2 Register Map

The following table shows the main registers of the MPU-6050, their addresses and a brief description.

Register name	Hex. address	Description
SMPRT_DIV	19	Divider from the gyro. output rate used to generate the sample rate. Sample Rate = Gyroscope Output Rate / (1 + SMPRT_DIV) Gyro Output Rate 8KHz when the DLPF is disabled and 1KHz when the DLPF is enabled.
CONFIG	1A	This register configures the external Frame Synchronization (FSYNC) pin sampling and the Digital Low Pass Filter (DLPF) settings for both the gyroscopes and accelerometers.
GYRO_CONFIG	1B	This register is used to trigger gyroscope self-test and configure the gyroscopes' full scale range.
ACCEL_CONFIG	1C	This register is used to trigger accelerometer self test and configure the accelerometer full scale range. This register also configures the Digital High Pass Filter (DHPF). The DHPF is available for the free fall, zero motion and motion detection, but not for the data registers.
FF_THR	1D	This register configures the detection threshold for Free Fall event detection.
FF_DUR	1E	This register configures the duration counter threshold for Free Fall event detection. The duration counter ticks at 1kHz, therefore FF_DUR has a unit of 1 LSB = 1 ms.
MOT_THR	1F	This register configures the detection threshold for Motion interrupt generation.
MOT_DUR	20	This register configures the duration counter threshold for Motion interrupt generation. The duration counter ticks at 1 kHz, therefore MOT_DUR has a unit of 1 LSB = 1 ms.
ZRMOT_THR	21	This register configures the detection threshold for Zero

Register name	Hex. address	Description
		Motion interrupt generation.
ZRMOT_DUR	22	This register configures the duration counter threshold for Zero Motion interrupt generation. The duration counter ticks at 16 Hz, therefore ZRMOT_DUR has a unit of 1 LSB = 64 ms.
INT_PIN_CFG	37	Configures the behaviour of the interruption pins.
INT_ENABLE	38	Enables each of the interrupt sources.
INT_STATUS	3A	Shows the status flag of each interruption source.
ACCEL_XOUT_H ACCEL_XOUT_L ACCEL_YOUT_H ACCEL_YOUT_L ACCEL_ZOUT_H ACCEL_ZOUT_L	3B-40	Each of the X, Y, Z registers stores the acceleration value in each axis. In addition, every axis is divided into H: higher part of the word and L: lower part of the word.
TEMP_OUT_H TEMP_OUT_L	41-42	These registers store the last temperature measurement. Divided into H: high and L: low bytes.
GYRO_XOUT_H GYRO_XOUT_L GYRO_YOUT_H GYRO_YOUT_L GYRO_ZOUT_H GYRO_ZOUT_L	43-48	Each of the X, Y, Z registers stores the gyroscope measurement in each axis. In addition, every axis is divided into H: higher part of the word and L: lower part of the word.
MOT_DETECT_STAUS	61	The register reports the status of the motion and zero motion detection.
MOT_DETECT_CTRL	69	The register selects the delay of the accelerometer power on time and configures the free fall and motion detection decrement rate.
PWR_MGMT_1 PWR_MGMT_2	6B-6C	These are the register for the power management. The first one allows the configuration of the power mode and the clock source. There is a bit for resetting the device and another bit for disabling the temperature sensor. The second register configure the frequency of the wake ups in the Accelerometer Only low power mode. Moreover, the individual axes of the accelerometer and of the gyro can be set into standby.

Table 24: MPU-6050 main registers [38]

2.4.3 Fall detection

“Falls affect over one in every three elderly people”. [39]. Different systems have been developed to detect a fall. It is difficult to define the characteristics of a fall and therefore, fall detection systems usually have some false positives and some false negatives. There is not

a unique solution for the fall detection problem and depending on the system different sensors are used. Some systems use only accelerometers, others use more sensors such as gyroscopes or tilt sensors. In addition, there are differences on the number of sensors and on the place where they are attached to the body. Some systems need only one module attached to the body while some others use more than one module. A review can be made in the existing fall detection systems.

Fall detectors using accelerometers and gyroscopes

The university of Virginia has implemented a fall detector using sensors for acceleration and angular velocity. The fall detector has two nodes, one placed on the chest and the other one placed on the upper leg. It analyses the activity intensity, the posture and the transitions. [40]

The Seoul National University has developed a fall detection algorithm using an accelerometer, a gyroscope and a tilt sensor. The system is attached to the chest. The measurements of the different sensors are compared to different thresholds to detect the fall. [41]

Fall detectors using accelerometers

The University of New South Wales has developed a physical activity monitor employing an accelerometer placed on the waist. The system is addressed to the elderly or to people suffering from chronic obstructive pulmonary disease (COPD) or from congestive heart failure (CHF). Thus, it is supposed that the movements will be slow. A fall will be detected when an acceleration greater than 2.5g is measured. [42]

The university of Limerick has implemented a fall detection algorithm using an accelerometer placed at the trunk and thigh. The algorithm is based on detecting when the acceleration norm exceeds a certain threshold. [39]

The Swiss Federal Institute of Technology has developed a wrist worn fall detector. It uses an accelerometer. The algorithm uses thresholds for the acceleration norm and for the velocity. [43]

Analog devices has also implemented an algorithm using an accelerometer. The algorithm consists of using the available interrupts of the accelerometer to detect the pattern of a fall. [44]

Acceleration patterns during a fall

The acceleration characteristics during the process of a human falling are represented in the figure 20.

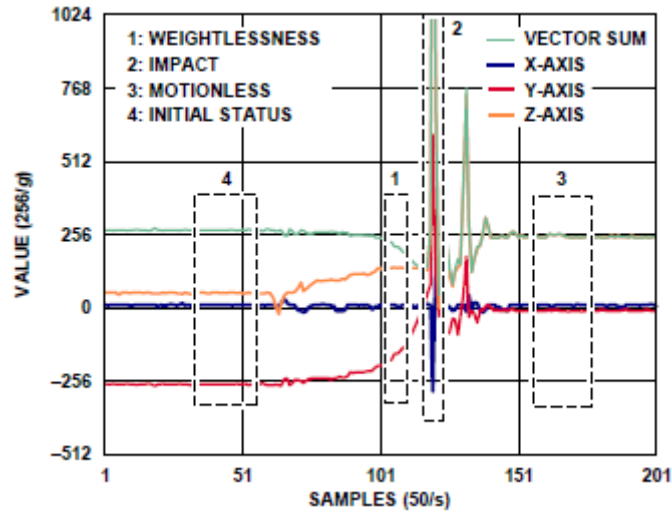


Figure 20: Acceleration change curves during the process of falling [44]

The first stage of a fall is the weightlessness (number 1 of figure 20). This means that the vector sum of acceleration reduces to near 0 g. The duration of the state depends on the height of the fall. This phenomenon can be detected with the free fall interrupt. The second stage of the fall is the impact with the floor (number 2 of figure 20). This can be detected with the motion detection interrupt. The final stage of a fall is the motionless (number 2 of figure 20). Generally, a person after falling remains motionless during a short period. This state can be detected with the zero motion interrupt. Finally, when a fall occurs, the final status is different to the initial status. Therefore, this is the final step to detect a fall. It is important to check that the difference between the final and the initial status exceeds a threshold. Analog Device proposes an algorithm using the acceleration pattern during a fall. The flow chart of the algorithm is shown in the figure 21.

After the initialization, the system waits for the free fall interrupt. The configuration for this interrupt is a threshold of 0.75g and a duration of 30ms. Once the free fall is detected, the motion interrupt has to be enabled with a threshold of 2g. The time between the free fall and the motion detection must be lower or equal to 200ms. If this condition is not satisfied the fall detection starts from the beginning.

After the motion detection, the system waits for the zero motion interrupt. The threshold of the zero motion must be set to 0.1875g and the duration is set to 2 sec. This zero motion interrupt must be detected within 3.5 sec after the motion interrupt (impact). Otherwise, the result is invalid and the system starts again from the beginning waiting for a free fall.

Finally, the acceleration in the final status has to be compared with the acceleration in the initial status. If the difference is greater than 0.7g a valid fall has been detected. Otherwise, the fall is not valid.

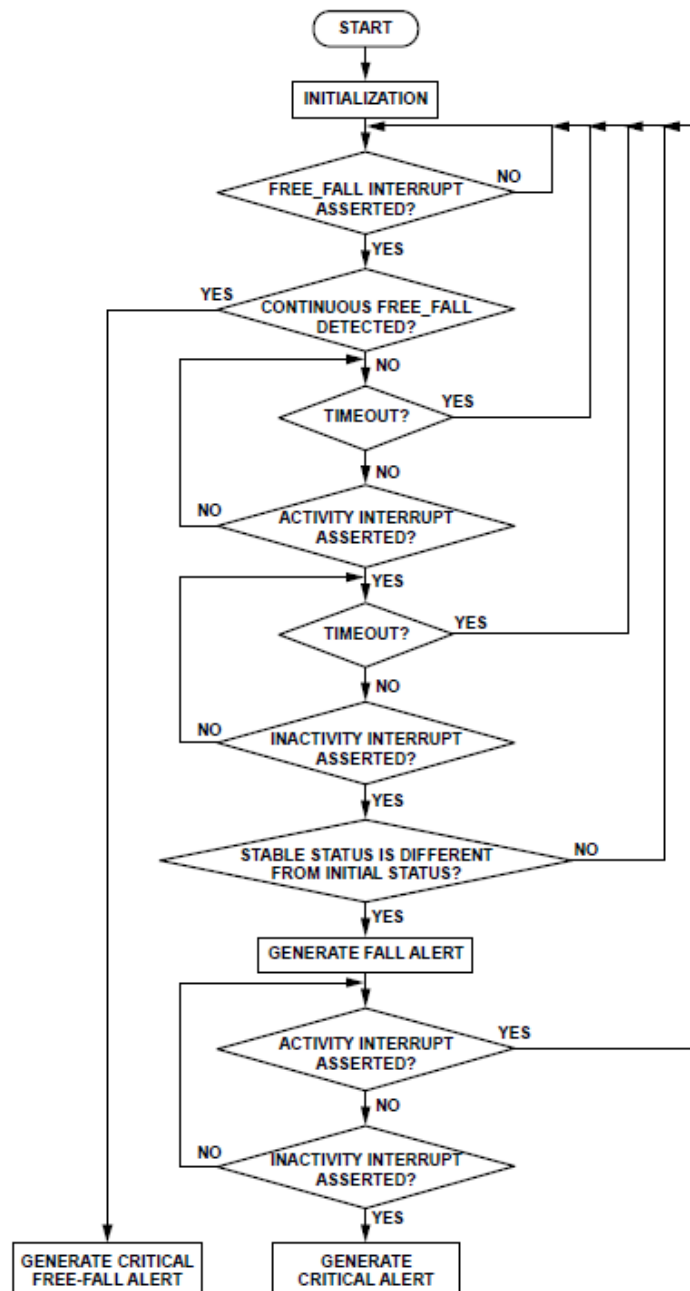


Figure 21: Fall detection algorithm flow chart [44]

After detecting a fall, the motion and zero motion interrupts can be controlled. If there is no activity within the next 10 sec a critical alert can be raised.

Moreover, there is a possibility to detect a continuous free fall. If a free fall interrupts is triggered during 300ms a critical alert can be raised.

This method was implemented using another accelerometer and it was tested with falls forward, backward, to the left and to the right. It was tested 80 times and every time the fall was successfully detected.[44]

3 Results

3.1 Audio interface

An interface with the user is necessary to give feedback to the user about the operation of the PHDI. The audio files are stored in the SDHC Card and played with the audio module. To achieve this, the audio files have to be read and converted from digital to analog in order to be amplified and played in the speaker.

The audio files are created in a computer using synthesized speech. The use of synthesized speech allows to lower the level of noise. The frequency used for recording the audio is 48KHz. This frequency was selected because it provides the best sound quality.

The first step is to have access to the SDHC Card and be able to manage the FAT file system. For this purpose, the free software explained in section 2.1.1 was used and adapted to be used with the PHDI prototype.

Once the access to the content of the SDHC Card was achieved successfully, the audio player was implemented.

3.1.1 Audio play function

First of all, the function opens the WAVE file and verifies that the file is a WAVE. This is done checking the values of the chunk ID and the RIFF type. If the result is successful, the chunk ID of the format chunk should be verified. It is important to store the useful information of the format chunk that will help to play the audio file: the number of channels, the sample rate, the number of bytes per second and the number of bytes per sample. Finally, the data chunk has to be found and the size of the data has to be stored.

The sample rate must be taken into consideration to determine whether the requested value can be played. If the sample rate is too high, some samples can be skipped in order to decrease the sample rate.

Once the final sample rate is known, it is important to set the correct value of the PWM period. This value will be set in the period register of the desired timer and can be calculated with the formula 2.

Two buffers of 512 bytes each are used to store the data read from the wave file. This allows to play the data from one buffer while the other is being filled.

The PWM duty cycle is updated in the timer Interrupt Service Routine (ISR). If the audio is mono, one Output Compare module is employed and if the audio is stereo one Output Compare is used for each channel. Once all the data has been read, the audio file has to be

closed, the timer interrupts have to be disabled and the audio amplifier must be disabled. [21]

The audio amplifier has a digital volume control. Since the PHDI has only one button, the volume is fixed to the maximum value.

3.2 RFID

The RFID module of the PHDI has to read the Tag ID placed on the PHD and save the ID number to be able to send it via ANT later. In order to save battery, the RFID module should be turned on and off every time it is used. Therefore, an initialization routine and a finalization routine have been implemented. The functions used in the routines are `int RFIDInit(void)` and `void RFIDEnd(void)`. An intervention of the user is necessary to start the RFID module. The RFID module initialization starts when the button is pressed. Once the RFID module is initialized, the MCU requests the RFID module to read the tag ID. This is done using the function `void RFIDRequestTagID(void)`. Finally, the MCU processes the response and saves the ID number using the functions `int RFIDLoopActiveCheck(void)`, `int RFIDReadTagID(void)` and `void RFIDGetTagID(unsigned char*)`. It is important to give feedback to the user about the success or failure of the activity via speech output. This procedure is explained in more detail in the following subsections.

The flow chart of figure 22 can clarify the operation of the RFID module. Notice that a timer will be used to prevent the timer enters an infinite loop.

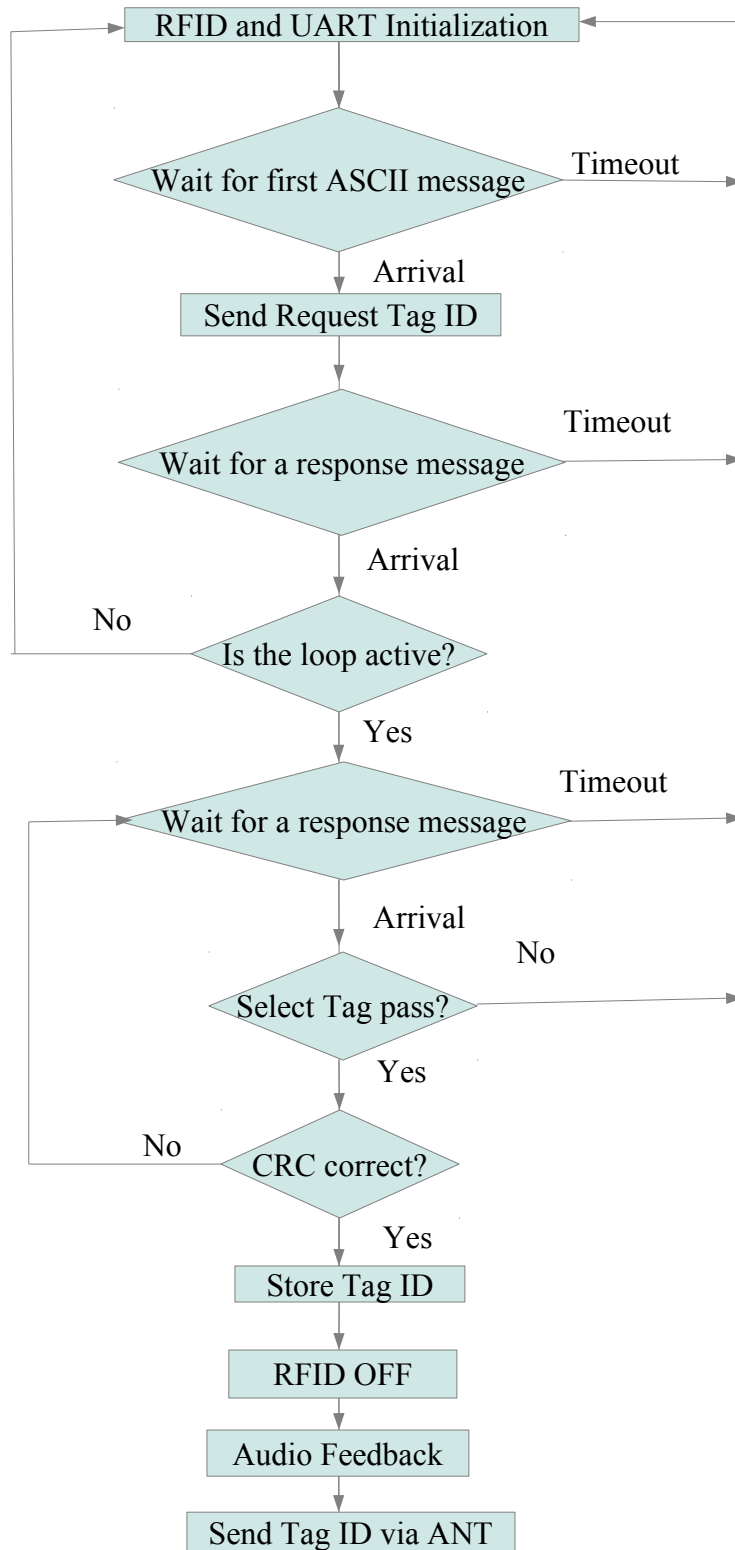


Figure 22: Flow chart of the operation of the RFID module

3.2.1 Initialization

The `RFIDInit()` function initializes the RFID module and the UART. This function is shown below:

```
1 | int RFIDInit(void) {
2 |     RFIDSetENDirection();
3 |     RFIDSetRSTDirection();
4 |     RFID_EN = 1;
5 |     RFID_RST = 0;
6 |     RFID_RST = 1; // RFID on
7 |     First_ascii = 0;
8 |     UART1Init();
9 |     if(RFIDWaitFirst())
10 |         return 1;
11 |     return 0;
12 | }
```

The first two lines set the Reset and the Enable pins as outputs. The next three lines set the correct values to initiate the RFID module. The `RFID_RST` pin of the MCU is connected to the reset pin of the RFID module (pin number 4 of figure 23). The `RFID_EN` is a pin in the microcontroller that works as a switch to open or close the voltage supply of the pin number 12 of figure 23.

As was said in the section 2.2, the RFID uses the UART, and more specifically, the UART1 to communicate with the MCU. Thus, the next step is to initialize the UART1 of the MCU. The `UART1Init()` function enables the UART1, the UART1 TX and RX and the interrupts for reception. Furthermore, the function also sets the configuration of the UART1 according to the configuration of the UART of the RFID module. The default configuration of the RFID module, and the one used in the PHDI, is 9600 bauds, one start bit, one stop bit and no parity [13].

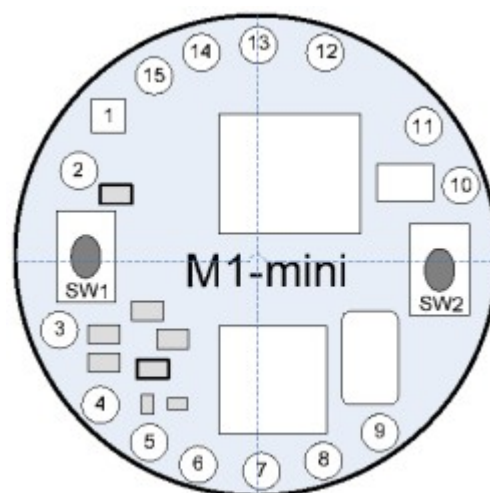


Figure 23: M1-mini Pins [13]

Due to the previous initialization, the RFID module sends to the MCU two messages in ASCII which do not give additional information. The `RFIDWaitFirst()` function takes care of waiting for those messages before starting to search for the Tag ID.

3.2.2 Request for the Tag ID

The `RFIDRequestTagID()` function creates the request message necessary to generate a response with the Tag ID number. Moreover, the function enables the TX interrupts to send

the request message to the RFID module. The implementation is shown in the source code below:

```

1 void RFIDRequestTagID(void) {
2     unsigned int crc_16;
3
4     RFID.TX.data[0] = RFID_STX; // command: autodetect rfid-
tag, binary mode
5     RFID.TX.data[1] = 0x05; //MSG LEN, number of bytes
without including the MSG LEN and STX
6     //Flags
7     RFID.TX.data[2] = 0x21; //CRC field is present
8     //No RID, TID or AFI field present.
9     //Turns off the RF transmitter after the command has
been executed
10    //Select the tag command to detect only one tag
11    //Loop execution of a tag command
12    //Command
13    RFID.TX.data[3] = 0x14; //Select Tag
14    //Tag Type
15    RFID.TX.data[4] = 0x00; //Auto detect
16    //CRC 16 bits
17    crc_16 = crc16(RFID.TX.data, 5);
18    RFID.TX.data[5] = (crc_16 >> 8) & 0x00ff; //LSB
19    RFID.TX.data[6] = crc_16 & 0x00ff; //MSB
20
21    while (RFID.TXUpdate == 1); // Block until last command
is shifted out of buffer
22
23    RFID.TXLength = 6;
24    RFID.RXCPointer = 0;
25    RFID.TXCPointer = 0;
26    RFID.TXUpdate = 1;
27
28    // Enable UART1 RX Interrupt
29    IEC0bits.U1TXIE = 1;
30 }

```

The section 2.2.1 explains the format of the binary request messages in the skyetek protocol. According to the protocol, the previous source code generates the Request tag ID message described in the table 25

STX	MSG LEN	Flags	Command	Tag Type	CRC LSB	CRC MSB
0x02	0x05	0x21	0x14	0x00	0xC5	0x41

Table 25: Content of the Request Tag ID message

The flags of the message allow to configure several options of the RFID. First of all, the flags specify which fields appears in the request: The value 0x21 means the CRC field is present in the message but the RID, TID and AFI fields are not present. Another option selected by the flags is to turn off the RF transmitter after the command has been executed. This option helps to save power. Furthermore, it is desired to get only one Tag ID. Then, the flag field is configured to select only the first tag ID detected. In addition, the flags allow to select a loop that searches for an ID all the time and sends a message for every successful response. At first, this option was not active and all the responses were failure, probably due to a problem with the ground of the antenna. Consequently, the configuration was changed, and the RFID module activates the loop that waits until it has a successful response.

The command specified by the message is the Select Tag. This is the command that generates a response with the Tag ID. The Tag type is present in the message because the command is a tag command. Even though the value 0x00 specifies that the tag can be of any type, there shouldn't be any problem of undesirable tag detection. This is due to the fact that the distance between the PHDI and the RFID Tag has to be around 10 mm in order to get a response.

Finally, the last two bytes of the message form the CRC. The algorithm used to calculate the CRC value is shown in the section 2.2.1.

After creating the previous message, the function `RFIDRequestTagID()` is responsible for enabling the transmission interrupts. It is important to activate the TX interrupts only when the message is going to be sent, because otherwise the interrupts will trigger all the time. Once enabled, the ISR takes care of sending the message through the UART 1 and disabling the TX interrupts once the desired message has been sent.

3.2.3 Process the response

As a result of the Request Tag ID message sent through the UART to the RFID module, the latter activates the loop mode and starts looking for a Tag. The activation of the loop generates a response message indicating that the select tag loop is working. The content of this message must be the following:

STX	MSG LEN	Response code	CRC LSB	CRC MSB
0x02	0x03	0x1C	0xF0	0x85

Table 26: Response message indicating Select Tag Loop active

The reception of this message is very important because it means the RFID module has started searching for tags. Thus, the reception of this message is controlled by the `RFIDLoopActiveCheck()` function. The source code is shown below:

```

1 | int RFIDLoopActiveCheck(void) {
2 |     if (RFIDWaitForResponse())
3 |         return 1;
4 |     RFID.RXUpdate = 0;
5 |     RFID.RX.bin_data.response_code = RFID.RX.data[2];
6 |     if (!(RFID.RX.bin_data.response_code ==
7 |     RES_SELECT_TAG_LOOP_ACT))
8 |         return 1; //Tag loop not active
9 |     return 0;
10| }

```

When a response is received, the ISR takes care of saving the message to be able to analyse it afterwards. The `RFIDWaitForResponse()` function (line 2) is a loop that waits until a complete message from the RFID module has been received. Notice that this function has a timer that ends the loop if there is a reception error.

Finally, when the message is complete, the function shown above verifies the response code of the message. The meaning of the response code of table 26 can be verified in the table 10. The value 0x1C is the response code for Select Tag loop active. The line 5 of the source code checks this response code. If the received message has the content specified in the table 26, the response of the `RFIDLoopActiveCheck()` function returns a 0 (no error). Otherwise, if the message is not the expected one, the function returns a 1 and the RFID will restart the process again.

If the previous function returns no error, it is known that the RFID module will search for tags until there is a successful response. A successful response has the following format:

STX	MSG LEN	RESP. CODE	TAG TYPE	TAG ID								CRC LSB	CRC MSB
0x02	0x0C	0x14	0x01	0xE0	0x04	0x01	0x00	0x11	0x1A	0x9E	0x88	0x1C	0x8A

Table 27: Example of a successful response message for the Select Tag request

If no error occurred, the response code must be a “Select Tag pass” (value 0x14). As the number of fields is fixed, the message length field allows to calculate the number of bytes of the tag ID (8 bytes in the example shown above).

The `RFIDReadTagID()` function is responsible for waiting for a successful response message and processing it. This is shown in the following source code:

```

1 | int RFIDReadTagID(void) {
2 |     unsigned char i;
3 |     unsigned int crc_16 = 0x0000;
4 |     unsigned int crc1 = 0x0000;
5 |     unsigned int crc2 = 0x0000;
6 |

```

```

7   if (RFIDWaitForResponse())
8       return 1; //Start from the beginning
9   RFID.RXUpdate = 0;
10  RFID.RX.bin_data.response_code = RFID.RX.data[2];
11  if (!(RFID.RX.bin_data.response_code ==
RES_SELECT_TAG_PASS))
12      return 1; //Start from the beginning
13  RFID.RX.bin_data.msg_len = RFID.RX.data[1];
14  for (i = 0; i < RFID.RX.bin_data.msg_len; i++) {
15      RFID.TagID[i] = RFID.RX.data[i + 4];
16  }
17  //check CRC
18  crc_16 = crc16(RFID.RX.data, RFID.RX.bin_data.msg_len -
1);
19  crc1 = crc_16 & 0x00ff; //CRC LSB
20  crc2 = (crc_16 >> 8) & 0x00ff; //CRC MSB;
21
22  if (!(crc1 = RFID.RX.data[RFID.RX.bin_data.msg_len]) &
(crc2 = RFID.RX.data[RFID.RX.bin_data.msg_len + 1]))
23      // The CRC is not correct, wait for response again
24      return 2;
25  for (i = 0; i < RFID.RX.bin_data.msg_len - 4; i++) {
26      RFID.TagID[i] = RFID.RX.data[i + 4];
27  }
28  return 0; //Everything was ok
29 }

```

First of all, in the line 7, the function waits until the reception of a message has completed. Once the reception is finished, the function checks if the received message is the desired response (see line 10 of the source code). This can be done checking the data response and comparing it to Select Tag Pass code. If the response code is not the expected one, the function returns a error and the whole process is restarted.

Nevertheless, if the response code is the right one, the next step is to check the CRC (lines from 16 to 23). If the CRC is not correct, the function returns a 2 and the process starts from the beginning of the `RFIDReadTagID()` function. If the CRC is correct, the function returns a value of success and the next step will be to save the Tag ID.

Finally, the `RFIDGetTagID(unsigned char*)` function is responsible for saving the tag ID in the pointer given to the function as a parameter. The content of the array saved by the function is shown in the table below:

Tag ID length	Tag ID							
0x08	0xE0	0x04	0x01	0x00	0x11	0x1A	0x9E	0x88

Table 28: Content of the array saved by the function `RFIDGetTagID()`

As can be seen in the table 28, the first byte indicates the length of the tag ID, and the next bytes are the ID.

In conclusion, the value obtained in the last step is saved in an array that can be used later to send the value to the android tablet via ANT.

3.2.4 RFID and UART disabling

Finally, it is important to disable the RFID and the UART to save energy. The `RFIDEnd()` function takes care of that.

```

1 void RFIDEnd(void) {
2     //Disables the RFID module and the correspondig UART
3     //to reduce power consumption.
4     RFID_EN = 0;
5     CloseUART1();
6 }
```

3.2.5 Power problems

During the development of the software for the RFID module there was a problem with the power of the RFID module. When the PHDI was powered by the ICD 3, the RFID module stopped working when the command “Select Tag” was sent. Using an oscilloscope it was discovered that the pin of UART reception (pin 7 of figure 23) behaved in a different way when this command was sent. While sending the rest of the commands, the line remained in the high state. However, once the “select tag” command was sent, the line changed into the low state. This problem was solved changing the power source. Fortunately, the battery provides enough power to read the tag ID.

3.3 ANT

The PHDI must communicate with the Android Tablet via ANT. This communication is necessary mainly to send the Tag ID read by the RFID module. However, an exchange of data between the PHDI (slave) and the android tablet (master) will also be necessary in other occasions. For instance, this communication will also be used to initialize the PHDI and get a unique ID, to send data about the battery state, and to get data for the Real Time Clock and Calendar (RTCC).

First of all, it is necessary to be able to exchange messages between the ANT module and the microcontroller via UART2. Hence, the UART2 and the ANT module have to be initialized. Secondly, the channel has to be configured and opened. Note that the PHDI will be configured to be a slave since the Android tablet will be the master. This will help to save battery of the PHDI because it will only open the channel when it wants to communicate with the Android Tablet. Finally, the handshaking process is necessary for the slave to get an address. Once all the previous steps are finished successfully, the PHDI will be able to send data to the Android Tablet via ANT.

3.3.1 ANT interface.

It is necessary an exchange of messages between the ANT module and the MCU. Therefore, a function called compose packet will be employed to form the desired packets. The UART2 ISR will manage the transmission and reception of the packets.

```

1 void antComposePacket(unsigned char type, unsigned char *
  data, unsigned char len) {
2     unsigned char checksum = ANT_SYNC;
3     ant.tx.wr = 0; //Read pointer
4     ant.tx.rd = 0; //Write pointer
5     ant.tx.buffer[ant.tx.wr++] = ANT_SYNC;
6     ant.tx.buffer[ant.tx.wr++] = len;
7     checksum ^= len;
8     ant.tx.buffer[ant.tx.wr++] = type;
9     checksum ^= type;
10
11     while (0 != len--) {
12         ant.tx.buffer[ant.tx.wr++] = *data;
13         checksum ^= *data++;
14     }
15     ant.tx.buffer[ant.tx.wr++] = checksum;
16     IEC1bits.U2TXIE = 1; //Transmission interrupts enable
17 }

```

This function uses three arguments, the type of message, a pointer to the data and the length of the data. The ant.tx.buffer array is used to store the message that will be sent through the UART afterwards. The content of the packet will be the following:

SYNC	LENGTH	MESSAGE ID	DATA: 0-LENGTH -1	CHECKSUM
-------------	---------------	-------------------	--------------------------	-----------------

Table 29: Content of the ant.tx.buffer array

The ant.tx.wr and ant.tx.rd are two pointers. The first one is used to fill the content of the ant.tx.buffer array and the second one is used to fill the transmission buffer of the UART2.

In addition, the checksum is calculated with the XOR operator of all the fields of the message and is included in the end of the ant.tx.buffer.

Finally, the transmission interrupts are enabled. The ISR will take care of sending the message to the ANT module through the UART2. Notice that the TX interrupts of the UART have a flag indicating if there has been overflow or not. This flag must be checked and it is necessary to wait to send the next byte until the overflow flag indicates that the overflow is over. Otherwise, problems will appear while trying to send a message longer than 11 bytes.

Sometimes there are errors in the reception of the of the UART data. There is also an error interrupt available. It is important to clear the error interrupt flag to be able to send or receive more data. Otherwise, once there is an error it is not possible to communicate with the ANT module.

The ISR for reception interrupts stores the receiving message in the array ant.rx.buffer. The pointer used to do so is the ant.rx.ptr. The field of the length of the data will be stored in the variable ant.rx.len. Finally, the antProcessRX is called in order to process the response. This function takes care of changing the value of the ant.status variable that allows to know the response received. Therefore, some different status have been defined:

```
1 //Status
2 #define ANT_OFFLINE 0 //No startup message received
3 #define ANT_ONLINE 1 //Startup message received
4 #define ANT_ADDR_AVAILABLE 2 //Address available message
  received
5 #define ANT_ACK 3 //ACK received (Event Transfer tx
  completed)
6 #define ANT_BUSY_ACQ 4 //Busy Acquiring message received
7 #define ANT_UN_ASSIGNED 5 //Channel status unassigned
  received
8 #define ANT_ASSIGNED 6 //Channel status assigned received
9 #define ANT_SEARCHING 7 //Channel status searching
  received
10 #define ANT_TRACKING 8 //Channel status tracking received
11 #define ANT_ACK_FAILURE 9 //ACK failure received (Event
  transfer tx faild)
12 #define ANT_EVENT_TX 10 //Broadcast message transmitted
  successfully
13 #define ANT_EVENT_SEARCH_TIMEOUT 11 //Timeout on searching.
  Channel closed
```

All of these status will be used afterwards to check if the received response is the desired or if it is not. The function ProcessRX sets the correct status for every response and it clears the first position of the ant.rx.buffer, the ant.rx.ptr and the ant.rx.len to be ready to receive a new message.

3.3.2 Initialization

The UART2 and the ANT have to be initialized. The function `int antInitialize(void)` shown below takes care of that.

```
1 int antInitialize(void) {
2     int pbClk;
3     pbClk = powerGetPBClock();
4     OpenUART2(UART_EN | UART_EN_WAKE | UART_BRGH_FOUR, // |
5     UART_TX_PIN_NORMAL, // Module is ON
6     UART_RX_ENABLE | UART_TX_ENABLE, // Enable TX &
7     RX
8     pbClk / 4 / 50000 - 1);
9     // Configure UART2 RX Interrupt
10    ConfigIntUART2(UART_INT_PR2 | UART_RX_INT_EN |
11    UART_TX_INT_DIS);
12    U2STAbits.URXISEL0 = 0; //TX Interrupt mode selection:
13    Interrupt generated
14    U2STAbits.URXISEL1 = 0; // when the buffer contains at
15    least one empty space
16    ant.status = ANT_OFFLINE;
17    ANTSetRTSDirection();
18    ANTSetRSTDirection();
19    ANTSetSLEEPDirection();
20    ANTSetSUSPENDDirection();
21    ANT_SUSPEND = 1;
22    ANT_SLEEP = 0;
23    ANT_RST = 0; // ant startup
24    ANT_RST = 1;
25    //Wait until ANT is ready to receive more data
26    while (ant.status != ANT_ONLINE) {
27        if (Timeout <= 0) {
28            Timeout = 100;
29            return 1;
30        }
31    }
32    return 0;
33 }
```

In order to initiate the UART2, the function `OpenUART2` from the peripheral library can be used [29]. This function enables the UART, the transmission, the reception and the interrupts. Moreover, it allows to configure the baud rate to the desired value of 50000 bauds. The UART interrupt can also be configured using a function from the peripheral library. This function, called `ConfigIntUART2` configures the interrupts with priority 2, enables the RX interrupts and disables the TX interrupts. The TX interrupts are configured to trigger when the TX buffer contains at least one empty space (lines 9 and 10). Then, this interrupts will be

enabled just when a transmission is desired and they will be disabled when the transmission is finished.

For the UART initialization, the SUSPEND and the SLEEP signals must be disabled. The reset pin can be asserted and de-asserted in order to be sure that the UART is in a known state with the previous configuration cleared. After the initialization of the ANT, the ANT module sends a notification to the microcontroller. This notification is the startup message, and takes place after a power up or reset event. [35] The format of this message is shown in the table 30:

SYNC	LENGTH	MSG ID	STARTUP MESSAGE	CHECKSUM
0xA4	0x01	0x6F	0-255	XOR

Table 30: Startup message

The Startup message field specifies the kind of reset that has occurred. Once this notification has been received, the ANT module is ready for the channel configuration.

3.3.3 Channel Configuration

The channel configuration takes care of setting the parameters of the channel. This is achieved by sending the channel configuration messages from the MCU to the ANT module. Some parameters can be set as desired by the user, but some values have to match in the slave and the master. An agreement was reached to set the same values in the master's and in the slave's configuration. The parameters are explained in the table 31:

Parameter	Value Master	Value Slave	Description
Channel number	-	0	Host's channel numbers do not need to match.
Channel Type	0x30	0x20	Transmit/Receive shared channel
Network Number	0	0	Default Public Network is 0
Device Number	0x01	0x01	Device number of the master
Device Type	0x03	0x03	Device type of the master.
Transmission Type	0x03	0x03	2 Bytes of shared address
Channel Period	0x2000	0x2000	Default 4 Hz Message Rate
Channel Search Timeout	-	0x08	Default value 25 seconds.(2.5 * 10 seconds) The Value 8 sets it as 20 seconds
Channel RF Frequency	0x39	0x39	The default value is 2466MHz. With the value set it is 2400+57 (MHZ) =2457MHz

Table 31: Channel configuration parameters

As was specified in a previous section, the channel Period, Channel Search Timeout and the Channel RF Frequency have default values and only require setting if a different value is desired. Otherwise, the rest of the parameters have no default values and the application must set the desired values. [35]

In order to be able to have communication with the Android Tablet and many PHDI's at the same time, a shared channel will be used. The shared channel avoids the restriction of the number of communications because it allows to have more simultaneous communications than the number of channels. One channel can be shared by up to 65k slaves.

After sending a configuration message there is a response message to indicate whether the configuration succeeded or not. A list of the channel configuration messages sent and received is explained below. Note that these messages are for the slave's configuration. To see the values for the master check the table 31.

Assign Channel messages

SYNC	LEN	MSG ID	CHANNEL NUMBER	CHANNEL TYPE	NETWORK NUMBER	EXTENDED ASSIGNMENT	CHECK SUM
0xA4	0x04	0x42	0x00	0x20	0x00	0x00	XOR

Table 32: Assign Channel message from the host to the ANT module

SYNC	LEN	MSG ID	CHANNEL NUMBER	MSG ID	MGS CODE	CHECKSUM
0xA4	0x03	0x40	0x00	0x42	0x00	XOR

Table 33: Successful channel response event to the Assign Channel message

Once the Assign channel message has been sent, the host microcontroller has to wait for the successful response to be able to continue with the channel configuration.

The message code of the table 33 is the RESPONSE_NO_ERROR and it is the response of the message 0x42 (Assign Channel).

Set Channel ID messages

SYNC	LEN	MSG ID	CHANNEL NUMBER	DEVICE NUMBER	DEVICE TYPE	TRANSMISSION TYPE	CHECK SUM
0xA4	0x05	0x51	0x00	0x01	0x0003	0x03	XOR

Table 34: Set Channel ID message from the host to the ANT module

SYNC	LEN	MSG ID	CHANNEL NUMBER	MSG ID	MGS CODE	CHECKSUM
0xA4	0x03	0x40	0x00	0x51	0x00	XOR

Table 35: Successful channel response event to the Set Channel ID message

When the Set Channel ID message of the table 34 is sent, the host MCU has to wait for the successful response of table 35 to be able to continue with the channel configuration. The message code means RESPONSE_NO_ERROR and it is the response of the message 0x51 (Set Channel ID).

Set Channel Period messages

SYNC	LEN	MSG ID	CHANNEL NUMBER	MESSAGE PERIOD LSB	MESSAGE PERIOD MSB	CHECK SUM
0xA4	0x03	0x43	0x00	0x00	0x20	XOR

Table 36: Set Channel Period message from the host to the ANT module

SYNC	LEN	MSG ID	CHANNEL NUMBER	MSG ID	MGS CODE	CHECKSUM
0xA4	0x03	0x40	0x00	0x43	0x00	XOR

Table 37: Successful channel response event to the Set Channel Period message

The message Set Channel Period only needs to be sent if the value of the messaging period is going to be different from the default value. In the case that a Set Channel Period message is sent, the host microcontroller has to wait for the successful response to be able to continue with the channel configuration. The successful response event expected when this message is sent, is shown in the table 37. The code is the RESPONSE_NO_ERROR and it is the response of the message 0x43, Set Channel Period.

Set Channel Timeout messages

SYNC	LEN	MSG ID	CHANNEL NUMBER	SEARCH TIMEOUT	CHECKSUM
0xA4	0x02	0x44	0x00	0x08	XOR

Table 38: Set Channel Timeout message from the host to the ANT module

SYNC	LEN	MSG ID	CHANNEL NUMBER	MSG ID	MGS CODE	CHECKSUM
0xA4	0x03	0x40	0x00	0x44	0x00	XOR

Table 39: Successful channel response event to the Set Channel Timeout message

The message Set Channel Timeout only needs to be sent if the value of the timeout is going to be different from the default value of 25 seconds. Once a Set Channel Timeout message is sent, the host MCU has to wait for the successful response to be able to continue with the channel configuration. The message code is the same as in previous responses but the message ID is 0x44 (Set Channel Timeout).

Channel RF Frequency messages

SYNC	LEN	MSG ID	CHANNEL NUMBER	RF FREQUENCY	CHECKSUM
0xA4	0x02	0x45	0x00	0x39	XOR

Table 40: Channel RF Frequency message from the host to the ANT module

SYNC	LEN	MSG ID	CHANNEL NUMBER	MSG ID	MGS CODE	CHECKSUM
0xA4	0x03	0x40	0x00	0x45	0x00	XOR

Table 41: Successful channel response event to the Channel RF Frequency message

The RF Frequency has to be set when a value different from the default one is desired. As usual, once the Channel RF Frequency message is sent, the host microcontroller has to wait for the successful response to be able to continue with the channel configuration. The successful response differs from previous ones in the message ID value 0x45 (Channel RF Frequency).

Open Channel messages

When all the parameters has been successfully configured, the channel has to be opened. To do so, the message shown in the table 42 has to be sent.

SYNC	LEN	MSG ID	CHANNEL NUMBER	CHECKSUM
0xA4	0x01	0x4B	0x00	XOR

Table 42: Open Channel message from the host to the ANT module

SYNC	LEN	MSG ID	CHANNEL NUMBER	MSG ID	MGS CODE	CHECKSUM
0xA4	0x03	0x40	0x00	0x4B	0x00	XOR

Table 43: Successful channel response event to the Open Channel message

The whole procedure of the channel configuration is a state machine that after sending a message checks the response and advances in the case of the desired response. Otherwise, if the desired response is not received the configuration message that obtained the failure has to be resend. This is shown in the flow chart of figure 24.

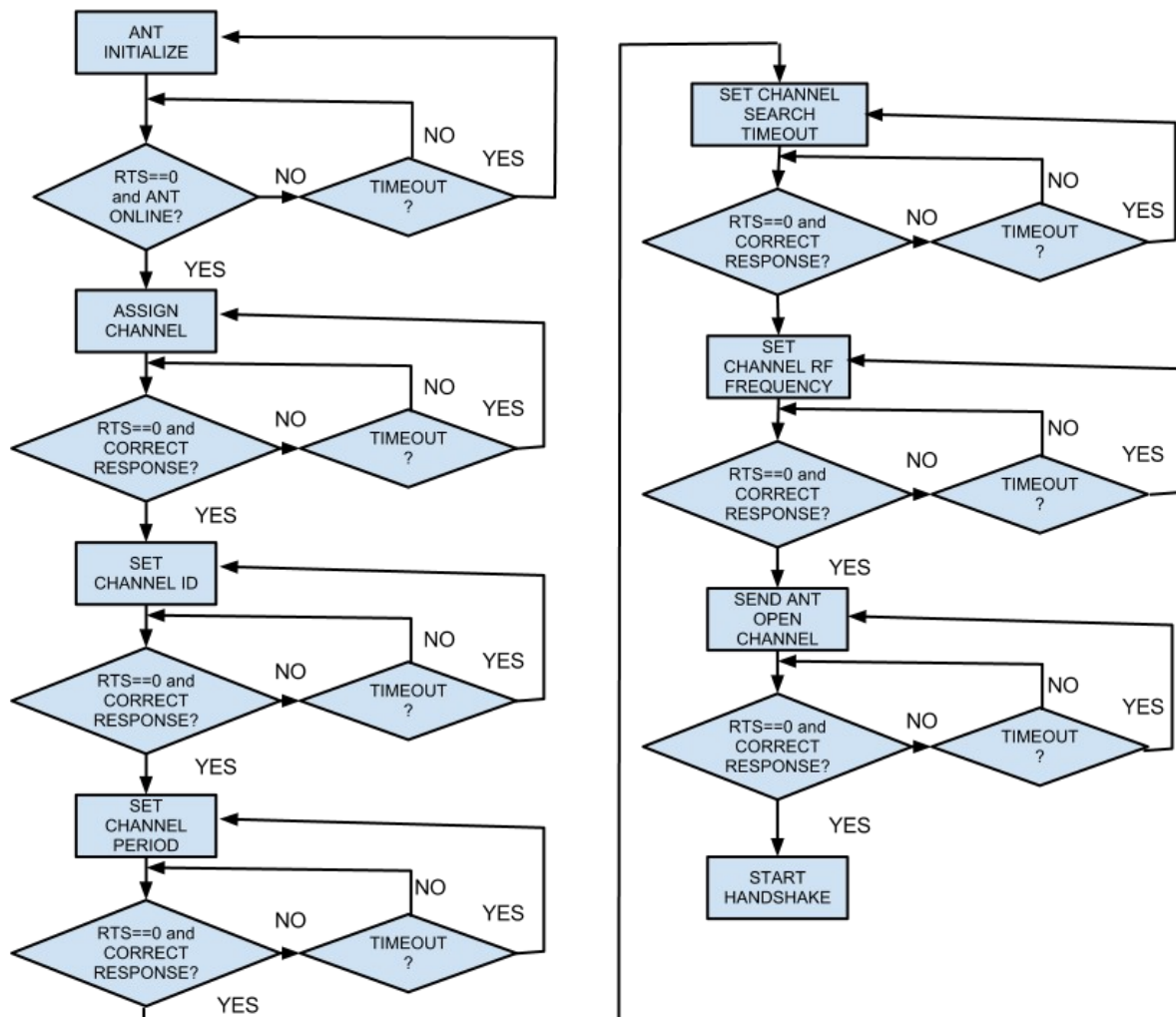


Figure 24: Flow chart of the channel configuration

To set the values of the timeouts a timer was employed to check the normal duration of these events. Finally, the values of the timeouts are set to 10ms. These timeouts are implemented using the timer 1 which is configured to trigger interrupts every millisecond because of requirements of the FAT file system. Notice that the flow control is implemented checking the RTS signal before sending a new message.

Finally, if the responses were successful for all the channel configuration messages, the Handshaking can start. Before implementing the handshaking procedure it is important to be able to send and receive data messages.

3.3.4 Data messages

The data messages are the messages exchanged between the ANT module and the Android Tablet via ANT. As was said in the section 2.3.1, there are three data types: Broadcast data, acknowledge data and burst transfer data. Only the first two will be used in the project.

Broadcast message

A function has been implemented to form a Broadcast message.

```
1 | int antSendBroadcastData(unsigned char* data3) {
2 |     int i;
3 |     unsigned char data[40];
4 |     data[0] = 0;
5 |     for (i = 0; i < 8; i++) {
6 |         data[i + 1] = data3[i];
7 |     }
8 |     antComposePacket(0x4E, data, 9);
9 |     Timeout=20000;
10 | while ((ANT_RTS == 1) || (ant.status != ANT_EVENT_TX)) {
11 |     if (Timeout <= 0) {
12 |         return 1;
13 |     }
14 | }
15 | return 0;
16 | }
```

As the first four fields of the message are fixed (Sync (0xA4), LEN (9), MESSAGE ID (0x4E) and CHANNEL NUMBER (0)), the pointer given to the function contains only the data from the second byte to the ninth byte. The antSendBroadcastData function takes care of adding the channel number and calling the antComposePacket with the proper parameters. Moreover, the function takes care of the flow control checking the RTS pin and waiting until it is asserted again. Finally, the function waits until the proper response has been received. Furthermore, there is a timeout to avoid an infinite loop. If there is an error in the response, the function returns a 1. However, if there is no error, the function returns a 0.

Acknowledge message

Another function, antSendAckData, has been implemented to send an acknowledge message. The code of this function is shown below:

```
1 | int antSendAckData(unsigned char* data3) {
2 |     int i;
3 |     unsigned char data[40];
4 |     data[0] = 0;
5 |     for (i = 0; i < 8; i++) {
6 |         data[i + 1] = data3[i];
7 |     }
8 |     antComposePacket(0x4F, data, 9);
9 |     Timeout=3000;
10 | while ((ANT_RTS == 1) || (ant.status != ANT_ACK)) {
11 |     if (Timeout <= 0) {
12 |         return 1;
13 |     }
14 | }
```

```

13 |         }
14 |     }
15 |     ant.status=100;
16 |     return 0;
17 | }

```

This function receives a pointer to the data bytes. In addition, it takes care of adding the channel number 0 and calling the function `antComposePacket` with the correct parameters. Finally, the function takes care of the flow control and it waits until the ACK has been received. In addition, there is a timeout that breaks out of the loop if the proper response is not received and returns an error in that case. If everything was successful, the function returns a 0.

3.3.5 Getting a unique identifier

There is a need for a unique identifier to be able to identify the user who wears the PHDI and relate the health measurements to him or her. A unique identifier is also necessary for the handshaking procedure that is explained in the section 3.3.6.

There are several possibilities to create a unique identifier. First of all, many ANT modules have a serial number which allow to easily create a unique identifier. However, unfortunately the ANT module used in the PHDI prototype does not have a serial number. The next option is to check if the rest of the modules have a serial number that can be used to create a unique identifier for the PHDI and the user. The RFID module has the option to save a user programmable serial number, but it is desirable to make this process automatically, without the need to program the PHDI one by one changing the number of each device.

Finally, the best option is to let the Android Tablet assign the ID to the PHDI. This process will be the initialization of the PHDI. As this unique ID is necessary for every data exchange with the Android Tablet, the PHDI will not be able to do anything else until the initialization process is complete. The initialization of the PHDI will allow to insert the name of the user in the Android application and the relation with the specific PHDI.

Initialization procedure

The unique ID will be created just the first time the PHDI is initialized. The number will be saved in a file in the SDHC Card.

The first time the PHDI is initialized, the button will be used to start the initialization process. Once pressed the button of the PHDI, the device will set its shared address to `0xFFFFD`, which is an address reserved for this procedure. Meanwhile, if the correct button is activated in the Android application, the tablet will start sending the message with the ID. This message contains broadcast data and it is sent to the shared address `0xFFFFD`. In the data fields it contains an identifier of 4 bytes. The expected message is shown in the table 44.

Sync	Len	Data type	Channel number	Addr LSB	Addr MSB	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Check sum
0xA4	9	0x4E	0	FD	FF	ID1	ID2	ID3	ID4	0	0	XOR

Table 44: Message to provide the unique ID sent from the tablet to the PHDI

Once the PHDI receives and stores the ID, an audio message will inform of the success of the initialization. The application of the Android tablet will take care of remembering the identifiers and the name of the users. **Note:** It is very important that no more than one PHDI is initialized at the same time because two PHDI could hear the same message and save the same identifier.

3.3.6 Handshaking

The handshaking is the procedure that has to be followed to get a shared address in an auto shared channel. This procedure was explained in the section 2.3.5. The state machine to implement the handshaking in the slave is shown in the figure 17. First of all, the slave has to set its shared address to 0xFFFF to be able to receive the Address Available from the master. To do so, the slave has to send a broadcast message indicating in the second and third byte of data the shared address.

Meanwhile, the master has to send the message Address Available explained in section 2.3.5. Once the slave receives the message, it saves the address available (0x0001 in the example of table 45) and sends the Request Address message. When the master receives that message, it starts sending the message Busy Acquiring to the shared address 0xFFFE. Consequently, the slave has to set its address to 0xFFFE to be able to receive the message from the master. Once the Busy Acquiring message is received with the correct unique ID, the slave has to send the message Confirm Acquire. When the ACK is received, the slave can set its shared address to the one available (0x0001) by sending the broadcast message. The master will set this address as not available and it will save the relation between the shared address and the unique ID. This relation between the unique ID and the address is not permanent, the PHDI will ask for a new shared address every time it wants to communicate with the Android Tablet. Then, the tablet should set a timer to release the shared address free after a while.

Sync	Len	Msg ID	Ch. #	Addr LSB	Addr MSB	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Check sum
0xA4	9	0x4E	0	0xFF	0xFF	0	0	0	0	0	0	XOR
0xA4	9	0x4E	0	0xFF	0xFF	0xFF	0	0	0xFF	0x01	0x00	XOR
0xA4	9	0x4F	0	0xFF	0xFF	0xFD	0	ID1	ID2	ID3	ID4	XOR
0xA4	9	0x4E	0	0xFE	0xFF	0xFE	0	ID1	ID2	ID3	ID4	XOR
0xA4	9	0x4E	0	0xFE	0xFF	0	0	0	0	0	0	XOR

Sync	Len	Msg ID	Ch. #	Addr LSB	Addr MSB	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Check sum
0xA4	9	0x4F	0	0xFE	0xFF	0xFC	0	ID1	ID2	ID3	ID4	XOR
0xA4	9	0x4E	0	0x01	0x00	0xFA	0	0	0xFF	0	0	XOR
0xA4	9	0x4E	0	0x01	0x00	0	0	0	0	0	0	XOR

Table 45: Messages Handshaking sent by the Master (white) and the slave (Blue)

Following the State Machine of the Slave shown in figure 17, the flow diagram of figure 25 has been implemented. Notice that timeouts are used to avoid infinite loops while waiting for the desired messages.

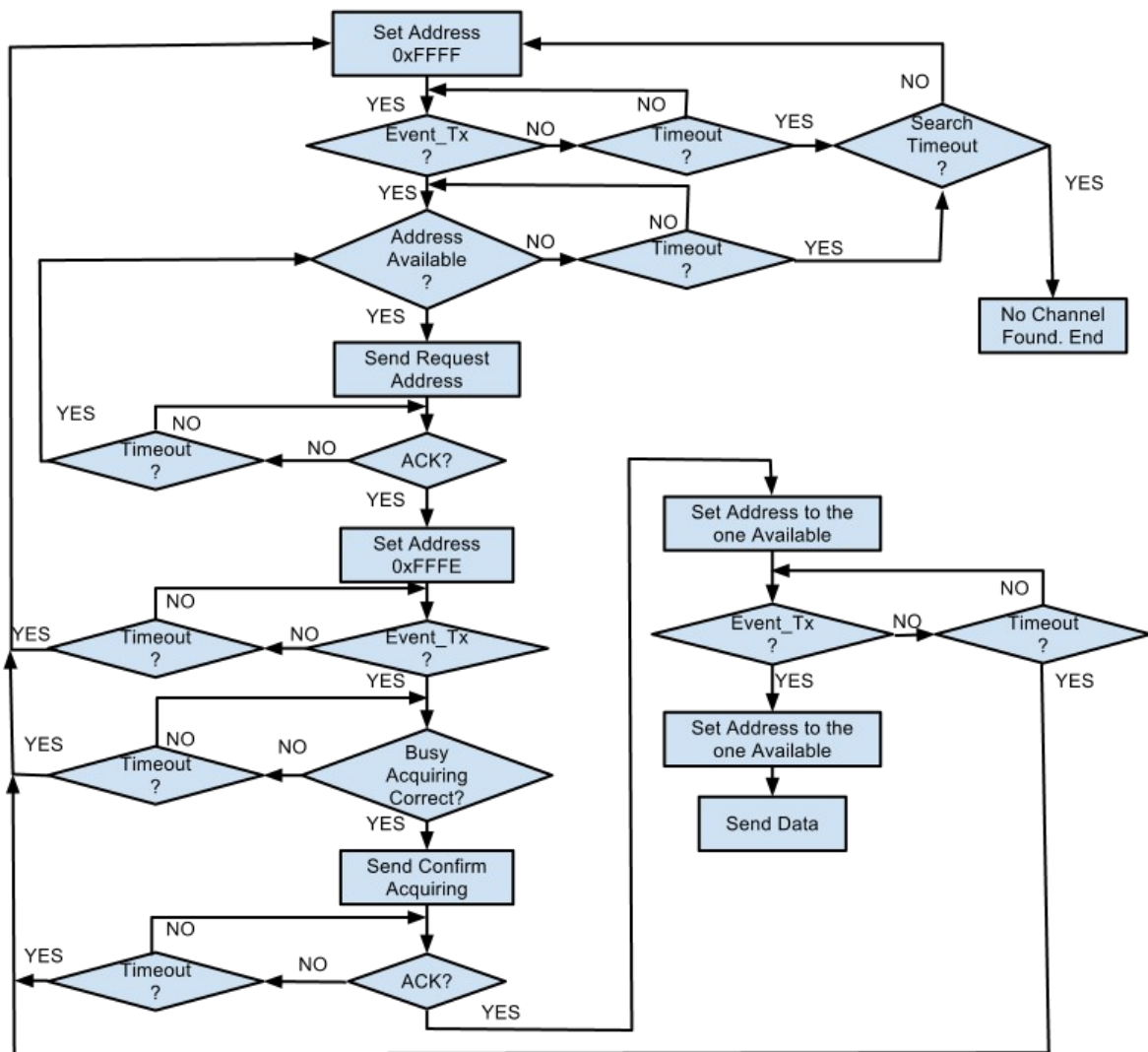


Figure 25: ANT Handshaking flow chart for the slave

3.3.7 ANT data exchange

The initialization of the UART and the ANT module, the channel configuration and the handshaking process, allow the communication and exchange of data between the PHDI and the android tablet.

As was said previously, the exchange of data will be used in several situations. Therefore, there is a need of a protocol to be able to distinguish one data exchange from another. The first six fields of the message and the last one are given by the ANT protocol. Moreover, the data type will always be acknowledged to get the confirmation of the correct reception of the message. However, the rest of the data fields have to be fixed by another protocol.

Type of message

The ANT protocol will be used to exchange data in several situations. To distinguish the situation, a message type field has been defined, and a different value is given to each type of message:

- RFID data exchange: Message type 0.
- Battery status messages: Message type 1
- RTCC request time: Message type 2
- RTCC request date: Message type 3
- RTCC request time for medicines: Message type 4
- Fall detected, trigger an alarm: Message type 5

RFID data exchange (0x00)

The length of the Tag ID is unknown. Thus, the first byte of data of the RFID message is the length of the Tag ID. Consequently, the Android Tablet is able to know where the Tag ID starts and when it finishes. Once the Tag identifier is written, the remaining fields of data will be filled with zeros.

Sync	Len	Data type	Channel number	Addr LSB	Addr MSB	Type msg	Data 1	Data 2	Data 3	Data 4	Data 5	Check sum
0xA4	9	0x4F	0	01	00	00	8	0xE0	0x04	0x01	0x00	XOR
0xA4	9	0x4F	0	01	00	00	0x11	0x1A	0x9E	0x88	0x00	XOR

Table 46: Example RFID Tag ID send

In the example shown in table 46, the Tag ID length is 8 bytes. Therefore, two messages are necessary to send the ID. Note that only the first message specifies the ID length. After the

length, the Tag ID is written (the highlighted numbers). Finally, a Zero-padding is used to complete the second message.

Battery status messages

The battery status messages allow to communicate the state of the battery. The available messages go from the PHDI to the tablet and are used to communicate to the user that the battery is low, charging or completely charged. Therefore, there are three different messages. These messages are shown in the table 47.

Sync	Len	Data type	Channel number	Addr LSB	Addr MSB	Type msg	Data 1	Data 2	Data 3	Data 4	Data 5	Check sum
0xA4	9	0x4F	0	01	00	01	0x01	0	0	0	0	XOR
0xA4	9	0x4F	0	01	00	01	0x02	0	0	0	0	XOR
0xA4	9	0x4F	0	01	00	01	0x03	0	0	0	0	XOR

Table 47: Battery status messages

Notice that the type of message is 01. The next field specifies the state of the battery: the value 1 means low battery, the value 2 means charging and the value 3 means charge complete. Note that the rest of the fields are filled with zero padding. These messages have been designed but have not been implemented.

RTCC messages

The RTCC must communicate with the tablet in certain occasions. To implement the medicine reminder it is important to know the time and the date. Moreover, if the PHDI runs out of battery the RTCC will stop counting. Therefore, it is necessary to request the time and date every time the PHDI initializes. The content of the messages sent between the tablet and the PHDI is shown in the table 48.

Sync	Len	Data type	Channel number	Addr LSB	Addr MSB	Type msg	Data 1	Data 2	Data 3	Data 4	Data 5	Check sum
0xA4	9	0x4F	0	01	00	02	0	0	0	0	0	XOR
0xA4	9	0x4F	0	01	00	02	Hour	Min	Sec	0	0	XOR
0xA4	9	0x4F	0	01	00	03	0	0	0	0	0	XOR
0xA4	9	0x4F	0	01	00	03	Year	Month	Day	Day Week	0	XOR

Table 48: RTCC Time and date request and response.

The first message is the time request. This message goes from the PHDI to the tablet. The second message contains the current time and is a message sent from the tablet to the PHDI.

The third message is the date request and the last message contains the current date. Note that every extra data of the message is filled with zero padding. The RTCC uses the time and date in the Binary Coded Decimal (BCD) format. These messages have been designed but have not been implemented.

3.4 MPU 6050 and fall detector

The MPU 6050 can read the acceleration and the angular velocity in 3 axis. These two sensors can be used to implement a fall detector. The device communicates with the MCU via I²C port. Before trying to implement a fall detector it is important to be able to configure the device and check its correct operation.

A I²C library is used to deal with the communication between the MPU-6050 and the microcontroller. Only four functions from the library will be used:

```

1 | unsigned char I2C1LDWriteByte(unsigned char, unsigned char,
   | unsigned char);
2 | unsigned char I2C1LDWriteByteArray(unsigned char, unsigned
   | char, unsigned char *, unsigned char);
3 | unsigned char I2C1LDReadByte(unsigned char, unsigned char);
4 | void I2C1LDReadByteArray(unsigned char, unsigned char,
   | unsigned char *, unsigned char);

```

These functions allow to read and write in the desired register of the accelerometer or gyroscope. The first argument of all these functions is the device address which is described with a label called ACC_ADDRESS. The second argument is the desired register.

These function with the correct parameters allow to configure the device and read the acceleration and angular velocity in every axis.

3.4.1 Sensors calibration

The sensors of the MPU 6050 can be damaged due to thermal or mechanical stress. Therefore it is convenient to practice a self test before trying to implement any application. The test requires to place the device motionless.

Axes	X	Y	Z
Accelerometer	Pass	Pass	Failed
Gyroscope	Pass	Pass	Pass

Table 49: MPU-6050 Self test results

The results of the self test are shown in the table 49. Every axes of the gyroscope passed the self test. However, the z axes of the accelerometer does not pass the self test.

To verify the malfunctioning of the accelerometer, the values of the axes of the accelerometer were checked. The obtained measurements for the device in a stationary flat position were:

- x axis: Value around 0g
- y axis: Value around 0g
- z axis: Value around 750mg

The values obtained for the x and y axes are the expected ones. However, the value for the z axis should be around 1g.

The device was soldered again and its operation improved. The z axis of the accelerometer gets a value closer to 1g while it is motionless and in a flat position. Even though the results improved, the z axis of the accelerometer does not pass the self test. The figure 26 shows the acceleration of the PHDI in every axes while the device was motionless and in a flat position. The module of the acceleration measured is around 900mg. There is still a deviation from the expected value of 1g.

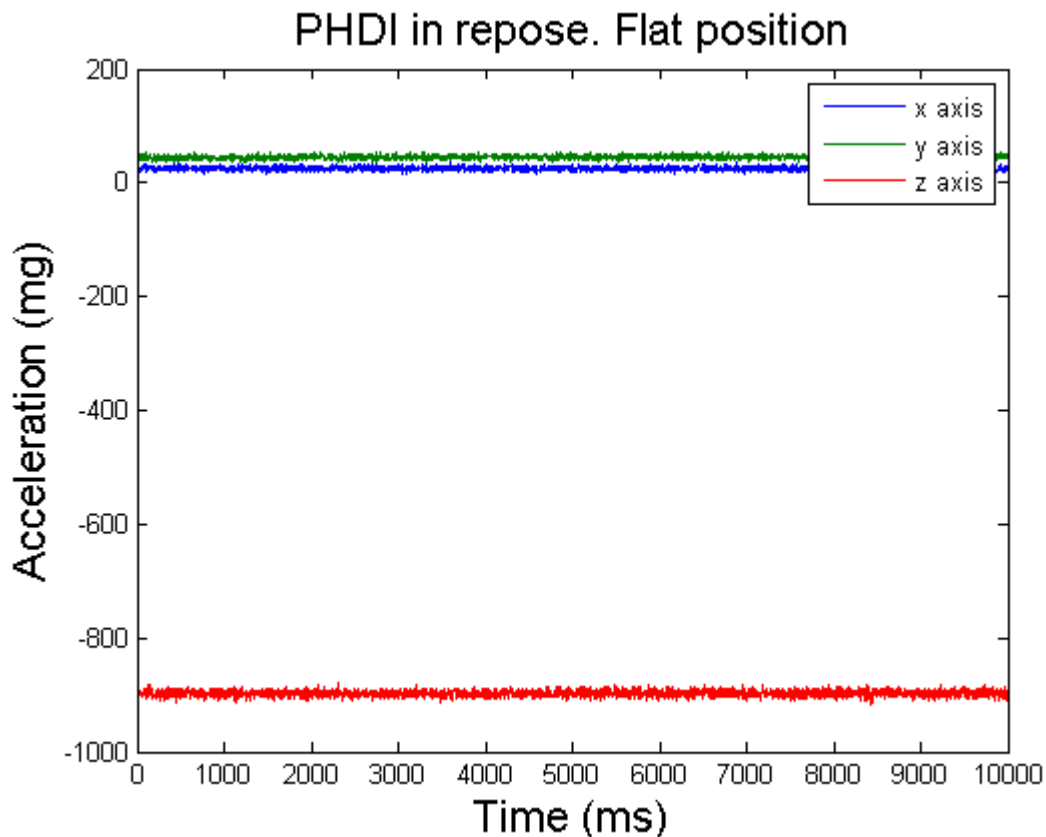


Figure 26: Acceleration in the 3 axes: PHDI in repose an in a flat position.

In the same conditions the angular velocity in the 3 axes was measured. The results are shown in the figure 27. The angular velocity of all the axes is close to 0°/s. There is a small deviation of 2°/s and 4°/s in the z axis and in the y axis respectively.

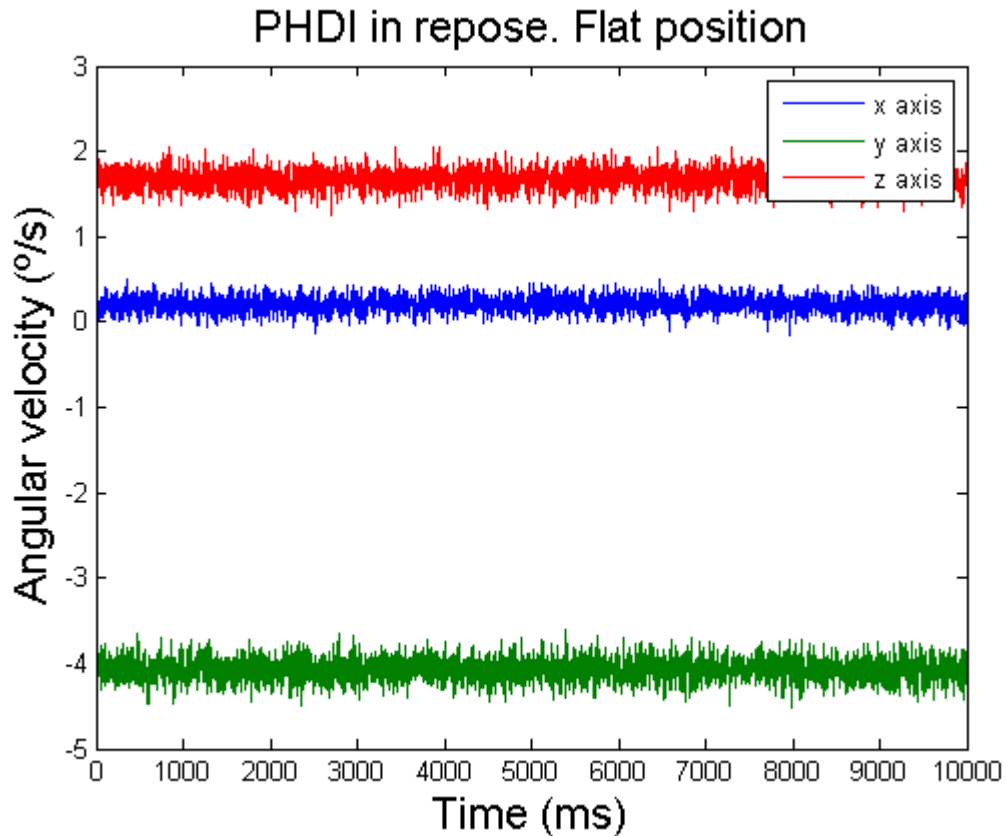


Figure 27: Angular velocity in the 3 axes: PHDI in repose and in a flat position.

3.4.2 Interrupts configuration

The accelerometer includes three interrupts that can be used for the implementation of different application. The available interrupts are:

- Free fall detection
- Motion detection
- Zero Motion detection

All of these interrupts have a threshold for the acceleration and for the duration. The interrupts can be easily configured with the corresponding register. However, it is important to notice that the threshold of the interrupts is limited to a value from 0g to 255mg. Therefore, the algorithm suggested in the section 2.4.3 can not be implemented using these interrupts.

3.4.3 Fall detection

The PHDI is a wrist wearable device. This place is very difficult for detecting a fall because any abrupt movement of the hand could be easily confused with a fall. A software has been developed for recording the measurements of the MPU 6050 while a fall is simulated. Recording the measurements of different falls allows to find a pattern of acceleration and angular velocity repeated in every fall.

It is probable to get some false positives. Therefore, it would be convenient to use a voice message to inform the user that a fall has been detected. If the detection was a false positive the user would be able to cancel the emergency alarm using the button. Furthermore, sometimes a fall occurs but medical assistance is not necessary. In that case the button is also useful to cancel the alarm.

An example of the PHDI acceleration and the angular velocity during a fall is illustrated in the figures 28 and 29. The recording starts some seconds before the fall and finishes some seconds after the fall.

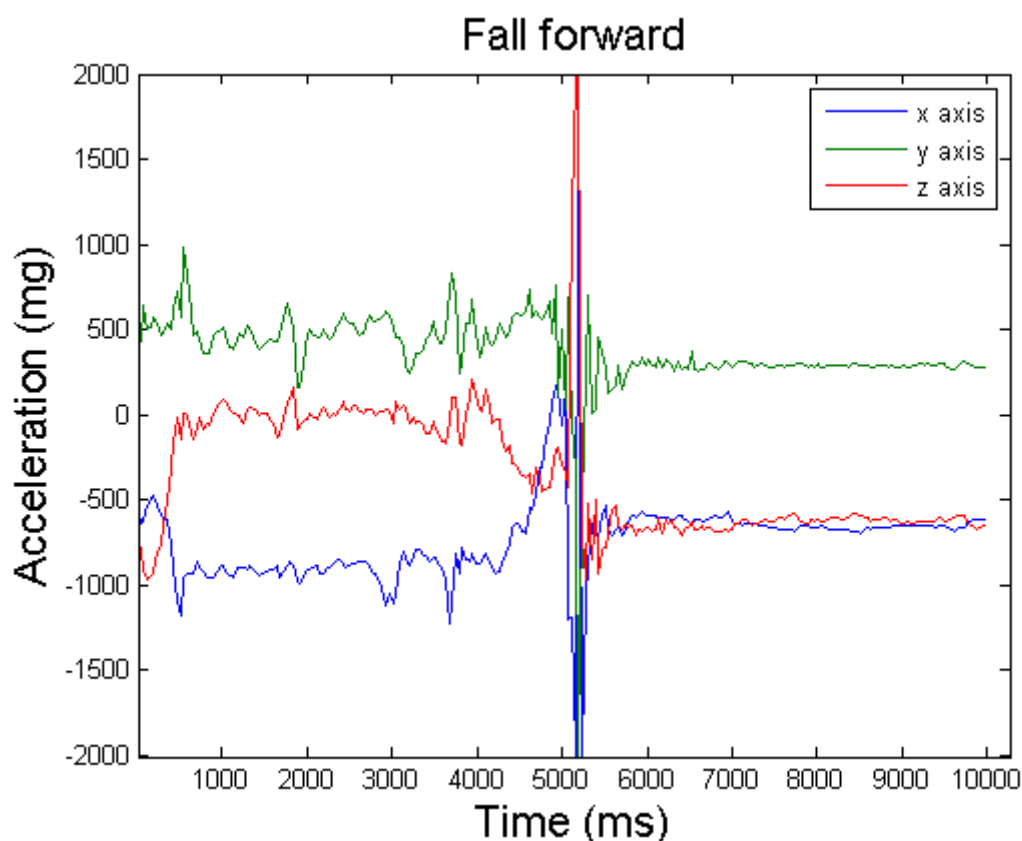


Figure 28: PHDI acceleration in 3 axes during a fall forward

The values of the acceleration were measured with the scale range of the accelerometer configured to $\pm 2g$. The free fall, the impact with the floor and the motion-less states can be distinguished. Also the initial and final status are different as was explained in in section 2.4.3.

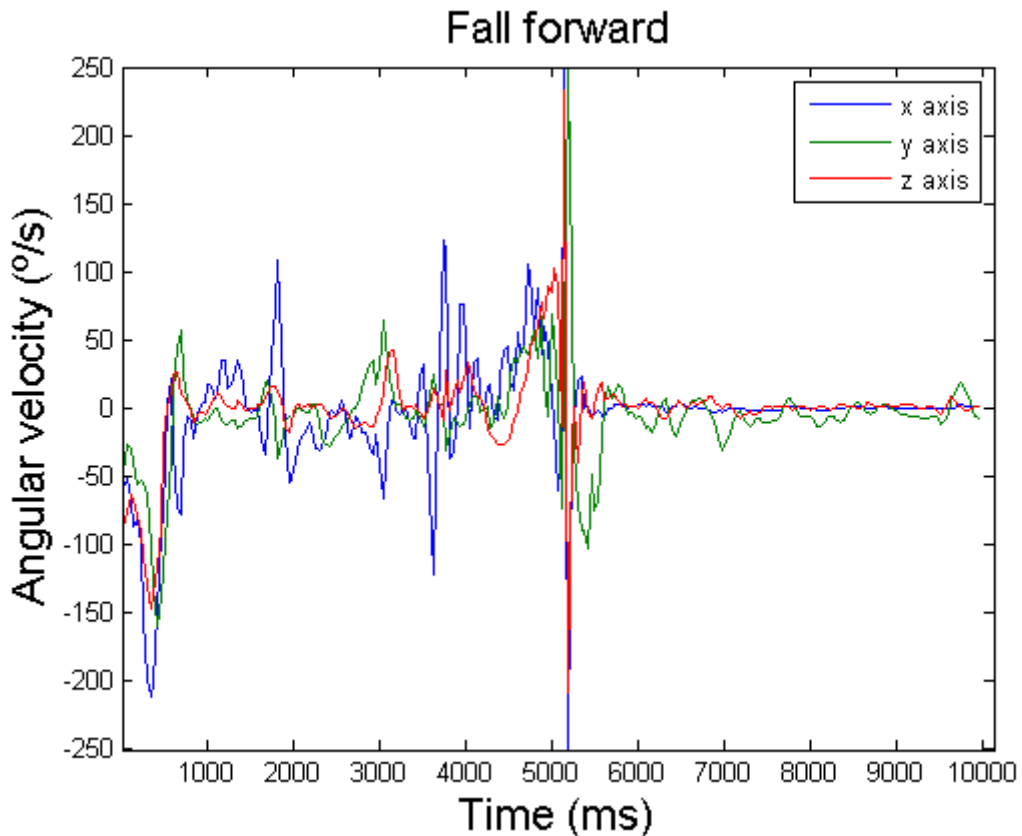


Figure 29: PHDI angular velocity in the 3 axes during a fall forward

The angular velocity was measured with the scale range of $\pm 250^\circ/\text{s}$. Notice that in the moment of the impact with the floor, the acceleration and the angular velocity exceed the scale range.

By the simulation of more falls a pattern could be found and it would allow to generate a fall detector.

4 Discussion

In this paper, the software designed and implemented for the new hardware prototype PHDI is presented. The PHDI is an innovative device for the telemonitoring field which makes easier and more comfortable the use of PHDs. The device provides an innovative way to relate each person with their health data measurements while the measurements are stored in a server. This process requires a minimum intervention of the user. The PHDI can be used anywhere where PHDs are employed, since it makes easier the collection of the measurements. However, people who will benefit from this product the most are those who share the PHDs with more people. This is the case of nursing homes, fitness centers or a house where the family shares PHDs.

The software design deals with the implementation of several interfaces. First of all, a communication interface between the Android Tablet and the PHDI is developed. This is done by the use of ANT technology. This communication is employed to exchange data between the PHDI and the Android Tablet. The initialization process of the PHDI also uses this communication interface. This initialization consists on getting a unique ID number which allows to distinguish the user that is communicating with the Android Tablet. This process is only made once.

The software developed deals with a communication interface between the PHDI and the PHD. RFID is used in this case. The PHDI allows to read a RFID Tag placed on the PHD. Once this is done the PHDI automatically sends the identification number of the PHD to the Android tablet via ANT. Then, the Tablet collects the data from the PHD and it assigns the measurements to the user.

The PHDI has a user interface. This interface has been designed simple and easy to use. Therefore, the PHDI has one button and a speaker. At first, the functionality of the button consists of activating the initialization procedure. Once the PHDI is initialized, the functionality of the button changes and it is used to activate the RFID Tags search. The PHDI gives feedback to the user via speech output. A voice message informs the user of the success or failure of every activity. The voice messages have been recorded with a voice synthesizer in order to avoid external noise.

The PHDI is a wrist wearable device what makes its use as comfortable as wearing a watch or a bracelet. Taking advantage of being a comfortable, user-friendly device, the hardware of the PHDI has been designed with the capability of implementing additional applications which will be convenient depending on the usecase.

The software has been designed to allow the coexistence of multiple PHDI communicating with the same Android Tablet. To achieve this a shared channel is used, and it can be used

by around 65000 PHDI at the same time. However, this coexistence has not been tested because only one hardware prototype has been built. For the future it would be convenient to test this capability.

A future improvement for the device could be the implementation of a buffer function. It is possible to lose communication between the Android Tablet and the PHDI. If that happens, the PHDI could store the data measured by the PHDs and send this data to the Android Tablet once the communication is recovered. This could be done using the existing ANT module to communicate also with the PHD.

This thesis makes also an approach to a fall detector. The acceleration and angular velocity during a fall have been recorded. However, the wrist is one of the most difficult parts of the body to place the sensors for the detection of falls. Activities of daily living can be easily detected as false falls. The fall detection application could be much easily implemented if the sensors were placed in other part of the body such as the chest or the waist, but then the device would not be that comfortable. The future work should focus on this implementation since falls are very common among elderly people and it is important to receive medical attention in time.

Moreover, future work can focus on the implementation of more elderly-oriented applications. The PHDI has the capability to implement a medicine reminder. The Android Tablet could provide the information about the timing of the medicines and the reminder can be set to the correct time using the RTCC. A life signal detector could also be implemented using the temperature sensor. In addition, a wake up detector could be implemented using the existing accelerometer. Finally, a button could be placed to set an alarm if there is an emergency.

The PHDI represents a breakthrough in the use of telemonitoring devices. With some future improvements this device can become an indispensable device for the elderly that will allow them to live longer at home without continuous medical assistance. The assistance can be get when there is a real necessity.

Bibliography

- [1] H. Steg, H. Strese, C. Loroff, J. Hull, and S. Schmidt, «Europe is facing a demographic challenge—ambient assisted living offers solutions», See <http://www.aal169.org/Published/Final%20Version.pdf> (last checked 6 July 2008), 2006.
- [2] European Commission, Directorate-General for Employment, Social Affairs and Inclusion, and Eurostat, «Demography report 2010. Older, more numerous and diverse Europeans».
- [3] European Commission and Eurostat, *Active ageing and solidarity between generations. A statistical portrait of the European Union 2012*. 2011.
- [4] Council (EPSCO / ECOFIN), «Joint report by the Commission and the Council on supporting national strategies for the future of health care and care for the elderly», Brussels, mar. 2003.
- [5] D. E. R. Warburton, C. W. Nicol, and S. S. D. Bredin, «Health Benefits of Physical Activity: The Evidence», *CMAJ*, vol. 174, n^o. 6, pp. 801–809, mar. 2006.
- [6] V. M. Conraads, C. Deaton, E. Piotrowicz, N. Santaularia, S. Tierney, M. F. Piepoli, B. Pieske, J.-P. Schmid, K. Dickstein, P. P. Ponikowski, and T. Jaarsma, «Adherence of Heart Failure Patients to Exercise: Barriers and Possible Solutions A Position Statement of the Study Group on Exercise Training in Heart Failure of the Heart Failure Association of the European Society of Cardiology», *Eur J Heart Fail*, vol. 14, n^o. 5, pp. 451–458, ene. 2012.
- [7] L. Burkow-Heikkinen, «Non-invasive physiological monitoring of exercise and fitness», *Neurol. Res.*, vol. 33, n^o. 1, pp. 3–17, ene. 2011.
- [8] R. Tarricone and A. D. Tsouros, «The solid facts. Home care in Europe», World Health Organization. Universita Commerciale Luigi Bocconi.
- [9] E. Seto, E. P., and S. M., «Cost Comparison Between Telemonitoring and Usual Care of Heart Failure: A Systematic Review», Centre for Global eHealth Innovation, University Health Network,, Toronto, Ontario, Canada., sep. 2008.
- [10] R. Antonicelli, P. Testarmata, L. Spazzafumo, C. Gagliardi, G. Bilo, M. Valentini, F. Olivieri, and G. Parati, «Impact of Telemonitoring at Home on the Management of Elderly Patients with Congestive Heart Failure», *J Telemed Telecare*, vol. 14, n^o. 6, pp. 300–305, ene. 2008.
- [11] A. Catón, «Design and implementation of a hardware prototype for Personal Health Data Identification (PHDI)», FH Technikum Wien.
- [12] «BMP085 Digital pressure sensor, Data sheet», Bosch Sensortec GmbH.
- [13] «SkyeModule M1-mini Reference Guide», SkyeTek, Inc.
- [14] «TDA8551 1 W BTL audio amplifier with digital volume control. Data sheet», NXP Semiconductors.
- [15] «MPU-6000 and MPU-6050 Product Specification», InvenSense Inc.
- [16] «PIC32MX5XX/6XX/7XX Family Data Sheet», Microchip Technology Inc.

- [17] «MPLAB Integrated Development Environment». [Online]. Available: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002. [Accessed: 03-feb-2012].
- [18] «PIC32 Starter Kit User's Guide, Data Sheet», Microchip Technology Inc.
- [19] «Starter Kit I/O Expansion Board». [Online]. Available: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en535444. [Accessed: 04-feb-2012].
- [20] «Speech Playback PICtail Plus Daughter Board, Data Sheet», Microchip Technology Inc.
- [21] L. D. Jasio, *Programming 32-bit Microcontrollers in C: Exploring the PIC32*. Newnes, 2008.
- [22] «PIC tail Daughter Board for SD and MMC Cards DS51583B, Data Sheet», Microchip Technology Inc.
- [23] «Home - SD Association». [Online]. Available: <https://www.sdcard.org/home/>. [Accessed: 29-mar-2012].
- [24] P. Kardos, «MPC5121e SDHC Controller. Application Note», Rožnov Czech System Center Czech Republic.
- [25] «ELM - FatFs Generic FAT File System Module». [Online]. Available: http://elm-chan.org/fsw/ff/00index_e.html. [Accessed: 03-feb-2012].
- [26] «Wave File Format - The Sonic Spot». [Online]. Available: <http://www.sonicspot.com/guide/wavefiles.html#filestructure>. [Accessed: 03-feb-2012].
- [27] A. Palacherla, «Using PWM to Generate Analog Output. Application Note 538», Microchip Technology Inc., 1997.
- [28] D. M. Alter, «Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Controller. Application Report», Texas Instruments Inc.
- [29] «PIC32 Peripheral Libraries for MPLAB C32 Compiler», Microchip Technology Inc.
- [30] B. Nisarga, «PWM DAC Using MSP430 High-Resolution Timer. Application Report», Texas Instruments Inc., jul. 2011.
- [31] B. Glover and H. Bhatt, *RFID Essentials (Theory in Practice)*, 1.^a ed. O'Reilly Media, 2006.
- [32] «SkyeTek Protocol Guide», SkyeTek, Inc.
- [33] W. Stallings, *Data and Computer Communications*, 6.^a ed. Prentice Hall, 2000.
- [34] «Interfacing with ANT General Purpose Chipsets and Modules», Dynastream Innovations Inc.
- [35] «ANT Message Protocol and Usage», Dynastream Innovations Inc.
- [36] «This is ANT, the Wireless Sensor Network Solution». [Online]. Available: <http://www.thisisant.com/>. [Accessed: 12-mar-2012].
- [37] *ANT Auto Shared Channel*. Dynastream Innovations Inc., 2011.
- [38] «MPU-6000 and MPU-6050 Register Map and Descriptions», InvenSense Inc.

- [39] A. K. Bourke, J. V. O'brien, and G. M. Lyons, «Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm», *Gait & posture*, vol. 26, n^o. 2, pp. 194–199, 2007.
- [40] Q. Li, J. A. Stankovic, M. Hanson, A. Barth, and J. Lach, «Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information», University of Virginia.
- [41] J. Y. Hwang, J. M. Kang, Y. W. Jang, and H. C. Kim, «Development of Novel Algorithm and Real-time Monitoring Ambulatory System Using Bluetooth Module for Fall Detection in the Elderly», in *Proceedings of the 26th Annual International Conference of the IEEE EMBS San Francisco, CA, USA • September 1-5, 2004*, 2004.
- [42] M. J. Mathie, J. Basilakis, and B. G. Celler, «A system for monitoring posture and physical activity using accelerometers», in *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd annual international conference of the IEEE*, 2001, vol. 4, pp. 3654–3657.
- [43] T. Degen, H. Jaeckel, M. Rufer, and S. Wyss, «SPEEDY:a fall detector in a wrist watch», in *Wearable Computers, 2003. Proceedings. Seventh IEEE International Symposium on*, 2003, pp. 184 – 187.
- [44] N. Jia, «Fall Detection Application by Using 3-Axis Accelerometer ADXL345. Application Note -1023», Analog Devices, Application Note.
- [45] «MEMS Gyro-Accel | Gyroscope | Accelerometer | Processing - MPU-6000/6050». [Online]. Available: <http://invensense.com/mems/gyro/mpu6050.html>. [Accessed: 25-mar-2012].

List of Figures

Figure 1: Population pyramids EU-27, 2010 and 2060 [2].....	1
Figure 2: Proportion of the people with a long-standing illness or health problem, EU-27 [3].	2
Figure 3: PHDI Infrastructure Overview.....	4
Figure 4: PHDI back side.....	5
Figure 5: PHDI front side.....	5
Figure 6: PIC32 Starter Kit demo board layout [18].....	8
Figure 7: I/O Expansion Board [19].....	8
Figure 8: Speech Playback PICtail Plus Daughter Board.....	9
Figure 9: Connections of the PICtail Daughter Board for SD and MMC Cards [22].....	10
Figure 10: Frequency spectrum of a PWM signal [21].....	16
Figure 11: DAC resolution vs PWM frequency [30].....	17
Figure 12: Volume control [14].....	17
Figure 13:OSI layers [33].....	23
Figure 14: Flow control with a RTS signal in a transfer from Host to ANT [34].....	24
Figure 15: Suspend mode control [34].....	25
Figure 16: ANT Serial Message Structure[35].....	25
Figure 17: Auto Shared slave state machine.....	36
Figure 18: ANTware II development application.....	37
Figure 19: MPU-6050 [45].....	38
Figure 20: Acceleration change curves during the process of falling [44].....	42
Figure 21: Fall detection algorithm flow chart [44].....	43
Figure 22: Flow chart of the operation of the RFID module.....	46
Figure 23: M1-mini Pins [13].....	47
Figure 24: Flow chart of the channel configuration.....	60
Figure 25: ANT Handshaking flow chart for the slave.....	64
Figure 26: Acceleration in the 3 axes: PHDI in repose an in a flat position.....	68
Figure 27: Angular velocity in the 3 axes: PHDI in repose and in a flat position.....	69
Figure 28: PHDI acceleration in 3 axes during a fall forward.....	70
	77

Figure 29: PHDI angular velocity in the 3 axes during a fall forward.....71

List of Tables

Table 1: RIFF chunk content [26].....	13
Table 2: The fmt chunk content [26] [21].....	14
Table 3: Data chunk format [26].....	15
Table 4: Request message sent by the host [32].....	18
Table 5: Fields of a request in the binary format [32].....	19
Table 6: Flags of the binary request message [32].....	19
Table 7: COMMAND field [32].....	19
Table 8: Response message sent by the radio module.....	21
Table 9: Binary Response Message [32].....	21
Table 10: Response codes [32].....	22
Table 11: Format of a Broadcast and a Acknowledge message [35].....	26
Table 12: Channel Response event message [35].....	26
Table 13: ANT Channel Types [35].....	28
Table 14: Assign Channel message [35].....	30
Table 15: Set Channel ID message [35].....	31
Table 16: Channel messaging period message [35].....	31
Table 17: Channel Searching Timeout message [35].....	32
Table 18: Channel RF Frequency message [35].....	32
Table 19: Set Network Key message.....	32
Table 20: Address available message content [37].....	34
Table 21: Request address message content [37].....	34
Table 22: Busy acquiring message content [37].....	35
Table 23: Confirm acquiring message content [37].....	35
Table 24: MPU-6050 main registers [38].....	40
Table 25: Content of the Request Tag ID message.....	49
Table 26: Response message indicating Select Tag Loop active.....	49
Table 27: Example of a successful response message for the Select Tag request.....	50
Table 28: Content of the array saved by the function RFIDGetTagID().....	52

Table 29: Content of the ant.tx.buffer array.....	53
Table 30: Startup message.....	56
Table 31: Channel configuration parameters.....	57
Table 32: Assign Channel message from the host to the ANT module.....	57
Table 33: Successful channel response event to the Assign Channel message.....	57
Table 34: Set Channel ID message from the host to the ANT module.....	57
Table 35: Successful channel response event to the Set Channel ID message.....	57
Table 36: Set Channel Period message from the host to the ANT module.....	58
Table 37: Successful channel response event to the Set Channel Period message.....	58
Table 38: Set Channel Timeout message from the host to the ANT module.....	58
Table 39: Successful channel response event to the Set Channel Timeout message.....	58
Table 40: Channel RF Frequency message from the host to the ANT module.....	59
Table 41: Successful channel response event to the Channel RF Frequency message.....	59
Table 42: Open Channel message from the host to the ANT module.....	59
Table 43: Successful channel response event to the Open Channel message.....	59
Table 44: Message to provide the unique ID sent from the tablet to the PHDI.....	63
Table 45: Messages Handshaking sent by the Master (white) and the slave (Blue).....	64
Table 46: Example RFID Tag ID send.....	65
Table 47: Battery status messages.....	66
Table 48: RTCC Time and date request and response.....	66
Table 49: MPU-6050 Self test results.....	67

List of Abbreviations

ACK	Acknowledge
ADC	Analog-to-Digital Converter
AFI	Application Field Identifier
ASCII	American Standard Code for Information Interchange
BCD	Binary Coded Decimal
CD	Card Detect
CHF	Congestive Heart Failure
COPD	Chronic Obstructive Pulmonary Disease
CRC	Cyclic Redundancy Check
CS	Chip Select
D/A	Digital-to-Analog
DAC	Digital-to-Analog Converter
DC	Direct Current
DLPF	Digital Low Pass Filter
DMP	Digital Motion Processor
EEPROM	Electrically Erasable Programmable Read-Only Memory
FAT	File Allocation Table
FPB	Frequency Peripheral Bus
FSYNC	Frame Synchronization
HL7	Health Level 7
I ² C	Inter-Integrated Circuit
ICD	In Circuit Debugging
ID	Identifier
IDE	Integrated Development Environment
ISM	Industrial, Scientific and Medical
ISR	Interrupt Service Routine
LED	Light Emitting Diode
LPF	Low Pass Filter
LSB	Least Significant Byte
MCU	Microcontroller Unit
MEMS	Micro Electro-Mechanical System
MMC	Multi Media Card
MSB	Most Significant Byte
OC	Output Compare
OECD	Organisation for Economic Cooperation and Development
OSI	Open Systems Interconnection
PHD	Personal Health Device
PHDI	Personal Health Data Identifier
PIC	Peripheral Interface Controller
PR	Period Register
PWM	Pulse Width Modulation
RF	Radio Frequency
RFID	Radio Frequency Identification
RID	Reader Identifier
RIFF	Resource Interchange File Format
RTCC	Real Time Clock and Calendar

RX	Reception
SCK	Serial Clock
SD	Secure Digital
SDHC	Secure Digital High Capacity
SDI	Serial Data Input
SDO	Serial Data Output
SDXC	Secure Digital Extended Capacity
SPI	Serial Peripheral Interface
STX	Start of Transmission
TDMA	Time Division Multiple Access
TID	Tag Identifiers
TTL	Transistor Transistor Logic
TX	Transmission
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
WAVE	WAVEform audio file format
WD	Write Detect
WiFi	Wireless Fidelity