Universidade de Aveiro
Ano 2011

Departamento de Electrónica, Telecomunicações e Informática

Krzysztof Jan Stelmachowski

Unidade de Áudio sem Fios

Wireless Audio Unit

Universidade de Aveiro
Ano 2011
Departamento de Electrónica, Telecomunicações e Informática

Krzysztof Jan Stelmachowski

Unidade de Áudio sem Fios

Wireless Audio Unit

Bezprzewodowa Jednostka Audio

Universidade de Aveiro
Ano 2011

Departamento de Electrónica, Telecomunicações e Informática

Krzysztof Jan Stelmachowski

Unidade de Áudio sem Fios

Wireless Audio Unit

Bezprzewodowa Jednostka Audio

Dissertation submitted to the University of Aveiro as part of its Master's in Electronics and Telecommunications Engineering Degree. The work was carried out under the scientific supervision of Professor Telmo Reis Cunha of the Department of Electronics, Telecommunications and Informatics of the University of Aveiro

Mr. Krzysztof Jan Stelmachowski is a registered student of Technical University of Lodz, Lodz, Poland and carried out his work at University of Aveiro under a joint Campus Europae program agreement. The work was followed by Professor Marcin Janicki at Technical University of Lodz as the local Campus Europae Coordinator.

o júri
presidente

Professor Doutor Dinis Gomes de Magalhães dos Santos
Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor Carlos Miguel Nogueira Gaspar Ribeiro
Professor Adjunto do Departamento de Engenharia Electrotécnica da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria

Professor Marcin Janicki
Associate Professor of Department of Microlectronics and Computer Science of Faculty of Electrical, Electronic, Computer and Control Engineering,Technical University of Lodz, Lodz, Poland

Professor Doutor Telmo Reis Cunha
Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

palavras-chave

Sistema Wireless, Microcontrolador, Processamento do sinal digital, Transceptor, Processamento de áudio.

resumo

A presente tese pretende descrever o desenvolvimento de um sistema electrónico, cuja funcionalidade se baseia na transmissão de sinais áudio através da rede Wireless.

Inicialmente foi estudada a família de microcontroladores PIC32, no qual se incluiu a sua forma de programação. Foi ainda realizada pesquisa acerca dos possíveis métodos de compressão de áudio, culminando com o desenvolvimento de algoritmos de compressão no software MATLAB.

Seguidamente foi desenvolvida a PIC32 Module – daughterboard do projecto. Esta é uma componente universal que contém um microcontrolador PIC32, de fácil utilização em outros projectos.

Posteriormente foi criado o dispositivo Wireless Audio Unit – o objectivo basilar desta tese. Este passo compreendeu a esquematização e PCB de ambas as partes: o transmissor e o receptor. Após a montagem, ambos os dispositivos forma colocados em caixas.

O firmware dos dois microcontroladores PIC32 foi criado em linguagem de programação C. O ADC e o DAC são controlados pelo firmware do PIC32, estando a ser executadas correctamente as suas funções.

No momento do desenvolvimento da componente escrita desta tese, ainda se mantêm alguns problemas associados à manipulação do transceptor. Por esta razão, o firmware WAU não foi terminado, e o dispositivo não cumpre, ainda, a sua funcionalidade.

keywords

Wireless system, microcontroller, digital signal processing, compression, transceiver, audio processing.

abstract

The thesis aims to report on the development of an electronic system, which task is to transmit wirelessly an audio signal.
The work was started by studying the PIC32 family of microcontrollers including the way of programming. The research on audio compression methods that was made, finished with development of compression algorithms in MATLAB software.
Following, the PIC32 Module – the daughterboard of project was designed. This part is universal unit containing PIC32 microcontroller, which could be easily used in many other projects.
Afterwards, it was created the proper Wireless Audio Unit device – the main objective of this dissertation. This step included design of schematics and PCB for two its parts: transmitter and receiver. After assembling, both devices was put into enclosures.
The firmware for two PIC32 microcontrollers was created in C programming language. The ADC and DAC are controlled by PIC32 firmware and are correctly realizing their functions.
At the moment of writing this document, the problem with handling transceiver was not solved. For this reason the firmware WAU was not finished and the device does not have its functionality.

Słowa kluczowe

System bezprzewodowy, mikrokontroler, obróbka cyfrowa sygnału, kompresja, transceiver, obróbka audio.

Streszczenie

Celem niniejszego dokumentu jest opis wykonanego systemu elektronicznego, którego zadaniem jest bezprzewodowa transmisja sygnału audio.
Praca została rozpoczęta od zapoznania się z rodziną mikrokontrolerów PIC32, włączając w to poznanie metod ich programowania. Badania nad istniejącymi metodami kompresji audio, zostały uwieńczone opracowaniem algorytmów kompresji w oprogramowaniu MATLAB.
Następnie został zaprojektowany moduł rozszerzenia - PIC32 Module. Jest to uniwersalna jednostka zawierająca mikrokontroler PIC32, która może być łatwo wykorzystana również w innych projektach.
Kolejnym krokiem było stworzenie właściwego urządzenia – Wireless Audio Unit (Bezprzewodowa Jednostka Audio), będącego głównym celem tej pracy. Etap ten zawierał projekt schematu oraz płytki obwodu drukowanego dwóch części projektu: WAU Transmitter (Nadajnik) i WAU Receiver (odbiornik). Po montażu, oba urządzenia zostały umieszczone w obudowach.
Oprogramowanie dla mikrokontrolerów PIC32 zostało stworzone w języku programowania C. Przetworniki a/c oraz c/a są kontrolowane przez mikrokontroler i poprawnie realizują swoje funkcje.
W chwili powstawania tego raportu, problem z obsługą transceivera nie został rozwiązany. Z tego powodu, oprogramowanie dla mikrokontrolerów nie zostało ukończone i urządzenie nie posiada założonej funkcjonalności.

I dedicate this work to my parents.

CONTENTS

List of Tables

List of Figures

CHAPTER 1
INTRODUCTION

Digital processing of audio is becoming more and more common. Increase of hardware performance with its prices drop, generates more possibilities for electronic audio engineers.

The development of digital wireless systems is growing too. Recently it has been a trend to use wireless applications wherever possible. The examples might be: wireless sensors, lighting control systems and many more. This tendency also given in wireless audio systems. There have been growth in number of applications like wireless microphones, headsets, wireless solutions for musical instruments.

Most of existing wireless audio devices are using digital transmission in except of analogue. An analogue system requires complex techniques to provide the performance level constant, since in analogue circuits almost always exists the problem of variable performance and adjustments of parts. In contrast, digital audio wireless transmission systems are almost free from such difficulties. Therefore the choice to widely use these kind of applications is reasonable.

1.1. Objectives

The objective of this work was to build from scratch a transmitter/receiver device that is able to transmit audio signal that is captured from musical instrument or playing unit. This device further in this document will be referred as Wireless Audio Unit (WAU).

Wireless Audio Unit is simply a substitute for a cable connecting two audio equipment. On Figure 1, there is an example of functionality of WAU – connection between musical instrument and speaker.

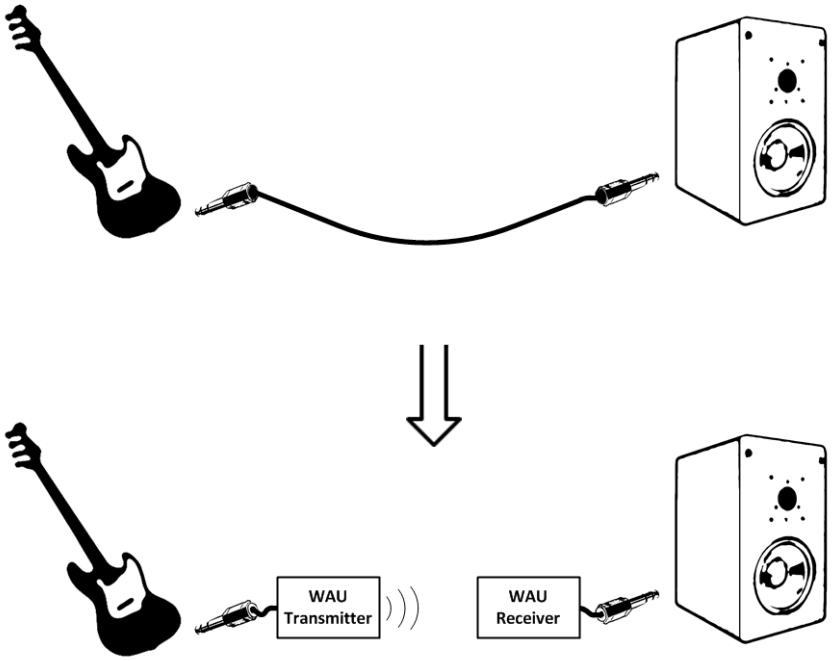


Figure 1: Functionality of Wireless Audio Unit

The proposed order of tasks necessary to develop the project was as follows:

- Study and understand the PIC32 family of microcontrollers. This part includes learning the programming process in 32-bit processors.
- Study on methods of audio compression
- Development of PIC32 Module – the daughterboard of WAU
- WAU schematics design and component selection.
- Design of PCB for WAU.
- Soldering the board and mounting it in enclosure.
- Development of firmware for WAU. This task consists of creating the program code for PIC32 in both transmitter and receiver parts.
- Final testing of the prototype

1.2. Dissertation structure

The dissertation is organized as follows:

Chapter 2 presents the state of the art survey connected with the subject area of project. There are described basic definitions from the field of analog and digital audio signals. Chapter presents the architecture of digital processing units, which includes the description of processors, analog-to-digital and digital-to-analog converters. This section ends with the specification of analog signal conditioning circuits.

Chapter 3 contains the description of design part of the work. It presents all steps that were made during the designing process from choosing appropriate components and making schematics to creating the PCB. It also includes the description of final assembly of Wireless Audio Unit. This chapter is divided in two sections. First characterizes the PIC32 Module – a daughterboard of WAU and the second describes Wireless Audio Unit device.

Chapter 4 is dedicated to the specification of compression methods. First, it is briefly described the idea of compression and methods of assessment. Further are presented different types of compression algorithms, starting with basic, statistical and dictionary techniques and ending with audio compression methods. The chapter closes with the presentation of developed and tested compression algorithms.

Chapter 5 is concentrated on two aspects: description of firmware for microcontrollers and presentation the results of device tests. This section characterizes certain components in category of their handling by microcontroller. The results of laboratory tests are illustrated with the use of several photos captured from oscilloscope.

Chapter 6 summarizes realized work. It contains the conclusions, that were gathered during creating the project. It also states the proposed future work on the Wireless Audio Unit.

CHAPTER 2
STATE OF THE ART

2.1. Properties of audio signals

2.1.1. Sound

A definition of sound can be stated it two ways[Ballou 91]:

- Psychophysical perception detected by ears and interpreted by a brain.
- Physical disturbance in a medium. It propagates it a medium as a pressure wave by the movement of atoms or molecules.

The most common medium in which sound is propagating is air. However it can propagate also in different medias like water, wood, metal. The best isolator for a sound is vacuum where there are no particles to vibrate. In Table 1 there are some examples of speed of a sound in different medias [Ballou 91].

Table 1: Speed of sound in different medias.

Medium	Speed [m/s]
Air, 21^{o}C	344
Water	1480
Wood	3350
Concrete	3400
Mild steel	5050
Glass	5200
Gypsum board	6800

As any wave, sound can by characterized by its amplitude and period.

The frequency, that is a number of periods per one second expressed in hertz (Hz) of sound wave can vary in wide range. Normally the range of audio frequencies that can be heard by human auditory system is between 20Hz and 20kHz depending of human age and health[Ballou 91].

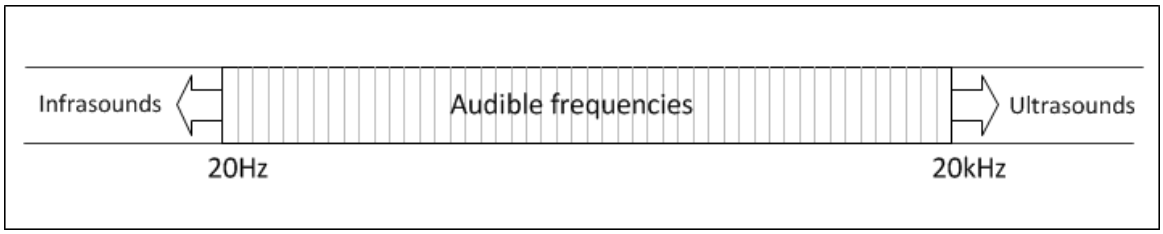


Figure 2: Range of audible frequencies

Under 20Hz there are located infrasound, and above 20kHz ultrasound.

The other property of sound is its amplitude. To human auditory system, the amplitude of a sound is identified as a loudness. Because the range of amplitudes which are audible to human ear is very wide, it is inconvenient to measure it in linear scale. In exchange the units of sound loudness are given in logarithmic scale of base 10. The level of sound is measured in two units: decibel (dB) and sound pressure level decibel (dB SPL).

Bel unit is defined as a 10-base logarithm of a ratio of two quantities expressed as power. If the result is multiplied by 10, it gives a unit of decibel.

$$\text{Level} = 10 \cdot \log_{10} \frac{P_1}{P_2} [dB]$$

The sound pressure level is a quantity in logarithmic scale describing the proportion of effective sound pressure of a sound and a reference value:

$$\text{Level} = 20 \log_{10} \left(\frac{p_{rms}}{p_{ref}} \right) [dB\ SPL].$$

To see what are the typical amplitude values of real sounds, in table 2 there are some examples of decibel levels for various sounds [Katz-Gentile 06]:

Table 2: Decibel (dBSPL) values of typical sounds.

Source	dB SPL
Threshold of hearing	0
Normal conversation	60-70
Busy traffic	70-80
Loud factory	90
Power saw	110
Discomfort	120
Threshold of pain	130
Jet engine (30 meters away)	150

The conclusion from this table, is that the range of level which is confortable to hear is up to 120dB and the audio systems should not exceed that value of signal amplitude.

2.1.2. Audio signals

Before describing what audio signal is, and how it is processed in digital systems, first it is necessary to define what signal is.

A signal is physical quantity which varies in time, space or any in other variable. It can be defined as a function of one or more independent variables[Proakis – Manolakis 96]. For example these functions

$$s_1 = 2\sin(3t)$$

$$s_2 = 2\mathrm{t} - 5$$

describe two signals which are changing one sinusoidally and other linearly in time.

If in these examples t ∈ $\mathbb{R}$, the signals are *continuous-time signals.* If t is defined only for a sequence of time instances:

$$\ldots < t_{-2} < t_{-1} < t_0 < t_1 < t_2 < \ldots, \quad (t_n \in \mathbb{R}),$$

than these signals are *discrete-time signals*[Morche 99]. Below, there is an example of continuous-time signal and his discrete-time equivalent.

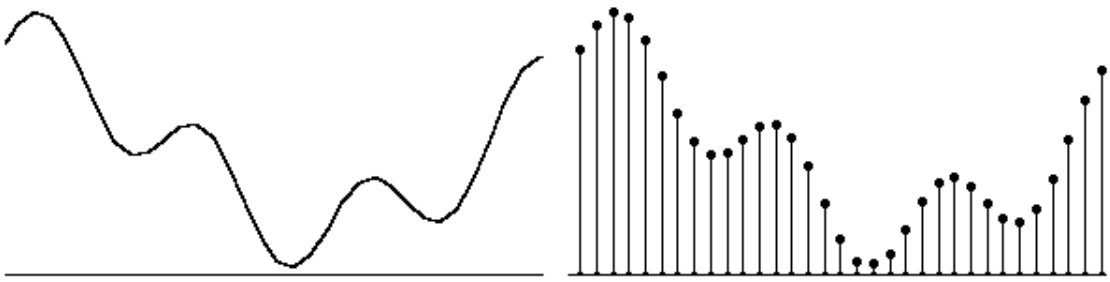


Figure 3: Analog and digital representation of a signal

Analog audio

Analog audio signal is a continuous-time signal, which has a frequency that can be detected as a sound to human ear.

Real audio signals like speech and music waves differ from simple sine wave. Nevertheless due to Fourier analysis, each periodic wave can be reduced to sine components. In the other hand each complex wave can be constructed from sinusoids of different frequency, amplitude and phase.

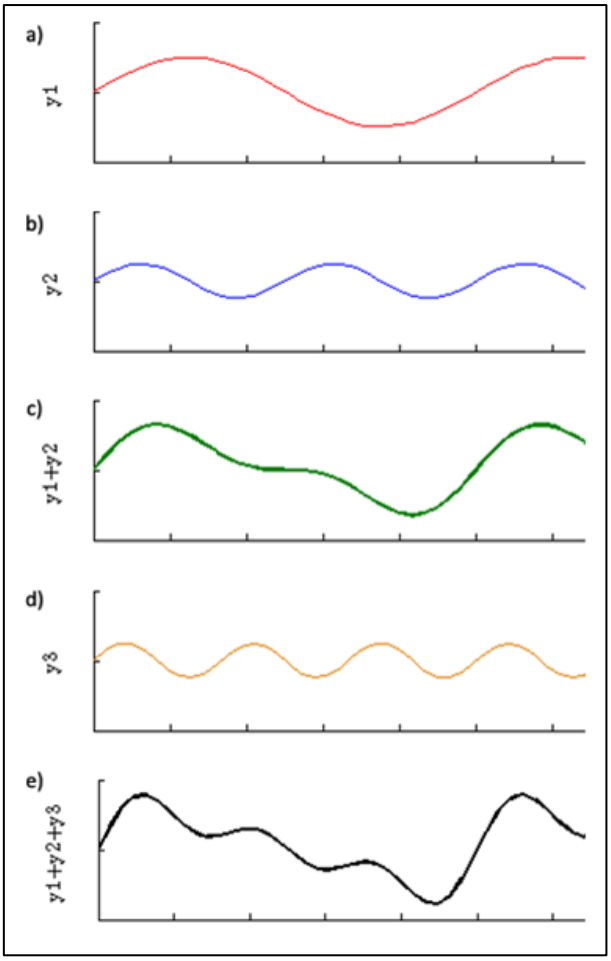


Figure 4: Combination of sine waves

On Figure 4a there is a simple sine curve y_1 of frequency f_1 and amplitude a_1. Figure 4b shows another sinusoid of frequency $f_2 = 2f_1$ and amplitude $a_2 = 0.5a_1$. The wave shape from Figure 4c is the combination (sum at each point in time) of sinusoids y_1 and y_2. On Figure 4d there is third sinusoid y_3 of amplitude $a_3 = 0.5a_3$ and frequency $f_3 = 2f_1$. Adding this one to y_1 and y_2, wave from Figure 4e is obtained. Therefore, a sinusoid y_1 has been distorted by adding other waves to it and it was created new wave shape.

The first wave of lowest frequency is called the *fundamental*. The second one (y2) is named *second harmonic* and y3 – *third harmonic*. If more waves (of frequencies which are successive multiples of f_1) were added, they would be called *fourth harmonic*, *fifth harmonic* etc.

The components of this signal can be represented in frequency domain graph as its *spectrum*. Figure 5 represents a spectrum of wave from Figure 4e:

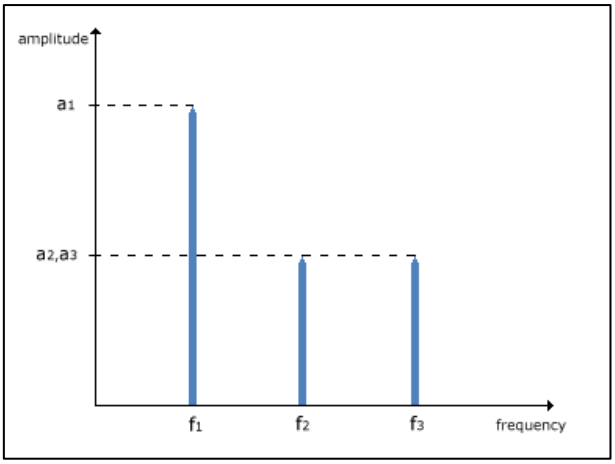


Figure 5: Frequency spectrum of a wave from Figure 2.3e

The frequency spectrum shows the fundamental of signal and all its harmonics.

Audio and electronics engineers use the idea of harmonics as they are connected to physical characteristic of sound. Musicians rather use different notation which is called *octaves* [Everest 2009]. This is a logarithmic concept which is implanted in musical scales because of its association with ear's characteristic. Harmonics and octaves are compared in Figure 6.

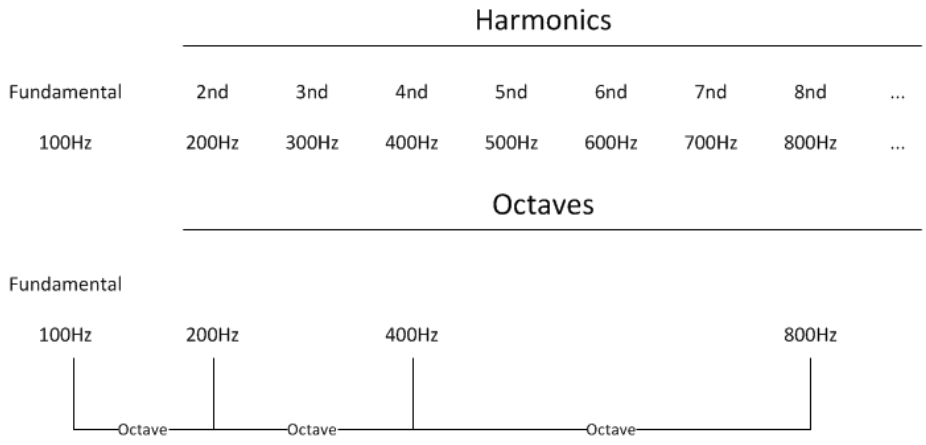


Figure 6: Comparison of harmonics and octaves

As it can be seen, if the fundamental is 100Hz, the first octave is between 100Hz and 200Hz. The second finishes at 400Hz which is twice of 200Hz. Consequently third octave starts at 400Hz and ends at 800Hz which is doubled 400Hz.

Digital audio

In order to process any analog signal (also audio) in digital system, first it is necessary to reduce signal to discrete samples in discrete time domain. Operation which converts analog signal to digital is called *sampling*. It is done by taking the values of continuous-

time signal at the time moments that are multiple of T_s, called the signal interval. The quantity $F_s = 1/T_s$ is called the *sampling rate* or *sampling frequency*.[Rochesco 3]

Intuitively higher sampling rate would effect in higher fidelity of digital signal. However, if bigger value of the sampling frequency is, the result file occupies more space in memory. From the other side the sampling rate cannot be too low, because it would not be possible to reproduce original signal from sampled.

The Nyquist-Shannon sampling theorem says[Rochesco 03], that a continuous-time signal which is band-limited to F_b can be recovered from its sampled version, if the sampling rate F_s satisfies an inequality:

$$F_s > 2F_b$$

When apply this theorem to audio signals, which are band-limited to 20kHz, it can be seen that the sampling frequency should be over 40kHz. That is why the music stored on Compact Disc (CD) is sampled with 44,1kHz. In table 3 there are some few examples of sampling frequencies in different systems.

Table 3 Commonly used sampling frequencies [Katz 154]

System	Sampling frequency [kHz]
Telephone	8
Compact Disc	44,1
Professional audio	48
DVD Audio	96

The sampling rate in telephone which was originally designed for speech communication is 8kHz. That is because frequency band in human speech is about 4kHz [Rabiner 78][Katz 06]. Therefore any sound with higher frequency than 4kHz send over the phone line would be distorted.

When an analog signal is sampled with any sampling rate, first it is *digitalized*, which means that a signal is broken into series of very short samples of amplitude equal to instantaneous value of amplitude of analog signal(Figure 7b).

After this procedure it is necessary to transform samples to discrete values that are suitable for digital processing and can be stored in memory. This is done by *sample and hold* circuits, which make that the amplitude of sample have constant value during the sampling period (Figure 7c). This process is called *quantization.*

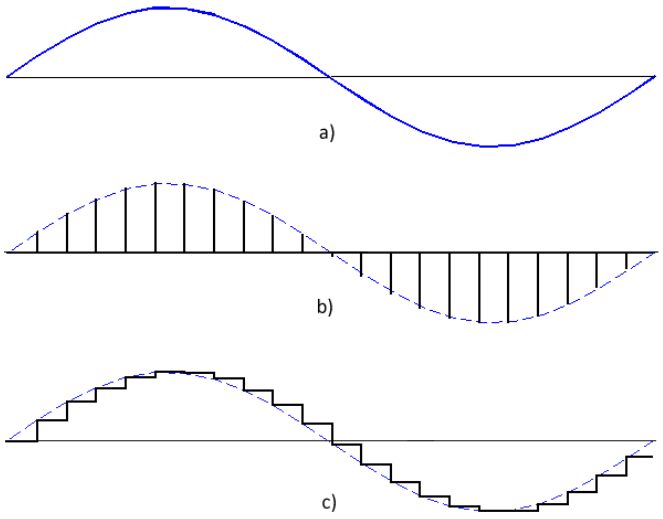


Figure 7: a) Analog signal, b) Digitalization, c) Quantization

The other aspect of transforming continuous-time signal to digital-time domain, is size of samples. After quantization each sample becomes a number. The size of this number, which is called *resolution* is represented in amount of bits. Higher resolution effects in higher fidelity of quantized wave to original one. This means that in high resolution two (or more) adjacent samples can have different values, while in lower resolution this two samples are noted as the same number. This case is presented on Figure 8 where at Figure 8c there is higher resolution of sampling than at Figure 8b which leads to better representation of original wave (Figure 8a). The bit resolution, that is usually used in practical applications has values 8, 16 or 24 bits depending on desired quality.

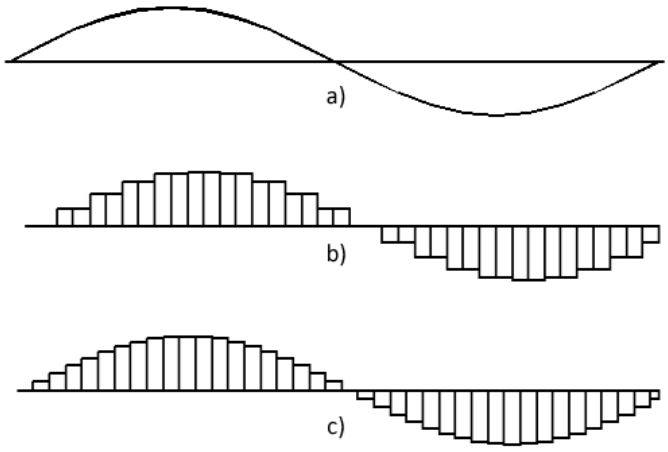


Figure 8: a) Analog signal, b) digitalized signal c) more precised digital signal

2.2. Digital processing units for audio signals

A digital processing unit begins with a conversion from analog input signal and ends with a conversion to analog output signal as it is shown in Figure 9.

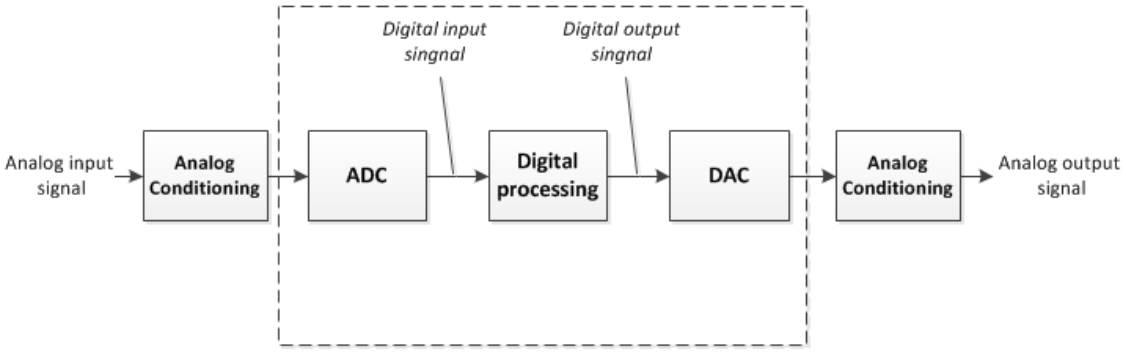


Figure 9: Block diagram of digital processing unit

Analog conditioning blocks contain antialiasing filters which in input are needed to remove high frequencies, that can interfere the baseband and are not carrying any information. At output, antialiasing filter removes the aliases that were produced during sampling.

After the analog conditioning, the signal is transformed into digital form by analog-to-digital converter (ADC). Then the signal is processed by microcontroller or other kind of processor. After processing digitalized signal is converted again to analog voltage or current by digital-to-analog converter.

Digital processing units can include also other type of digital or analog operations on signal, but only those listed above will be described.

2.2.1. Processors

The choice of processor can be made from many different options. Each of them has his advantages and disadvantages. Selecting the processing device is very important for performance of digital processing unit, since it is the main element of the system. The basic available choices are:

- Application-Specific Integrated Circuit (ASIC)
- Field-Programmable Gate Array (FPGA)
- Microcontroller (MCU)
- Digital Signal Processor (DSP)

ASICs [Smith M.J.S 97] are type of integrated circuits, designed to realize a predetermined specific tasks. Because of this, they can be optimized for both power and performance. ASICs usually are implemented after successful prototype. They can be applied to reduce the cost of final device which is produced in large quantities.

FPGAs [Smith G. 10] have generally the same functionality as ASICs, but it can be reprogrammed many times after manufacturing and mounting on target device. Because FPGAs are composed of optimized hardware blocks for a given collection of functions, they can achieve better performance than programmable processors. For example, it is common on FPGAs to do parallel computation in range, that is not possible on standard microcontrollers or DSP. In the other hand FPGAs are usually large, expansive, and consume a lot of power, therefore they are not suitable for portable applications. Also in systems which not require the performance of FPGA, it is reasonable to realize application using simpler processors.

MCU [Crisp 04] typically acts as system controller. It specializes in situations, when many conditional operations and frequent changes in program flow are needed. The code of microcontroller is usually written in C or C++ which are among the most popular programming languages. There are many types of microcontrollers, that can have different performance from 4-bit, 32kHz models to even 32-bit, 500MHz devices. The need of connecting digital processing units to media sources and targets, in most cases eliminates microcontrollers that are not 32-bit. The real-time operations performed in digital processing units requires the bandwidth and computational power of 32-bit MCUs. This does not means, that 8-bit processors are not useful. They can be used in simpler applications specially with low-power requirement. Additionally they can act as a companion to 32-bit MCU in more complex systems. The advantage of MCUs is that they have implemented many useful peripherals like high speed USB, Ethernet connectivity etc. That makes them easy to use in complex applications. In digital processing unit, MCU can perform not only signal processing but also can serve as system controller.

Digital signal processors (DSP) [Lyons 10] are used in applications, where MCUs computing power is not sufficient. DSPs are specialized and optimized to run in tight, efficient loops, performing as many multiply-accumulate (MAC) operations as possible in a single clock cycle. Achieving DSP full performance usually requires writing optimized assembly code. DSPs are running at very high clock rates. They are usually compared in number of MAC operations they can perform per second. However, a DSP is not ideal standalone processor. Often, when it is focused on signal processing, it is not able to deal with whole system control. That is why often DSP are used together with MCUs, which provides asynchronous system control functionality.

2.2.2. Analog-to-digital converters

Analog-to-digital converter (ADC) is a device, which turns analog quantities into digital form. The structure of basic ADC can be presented as it is shown on Figure 10.

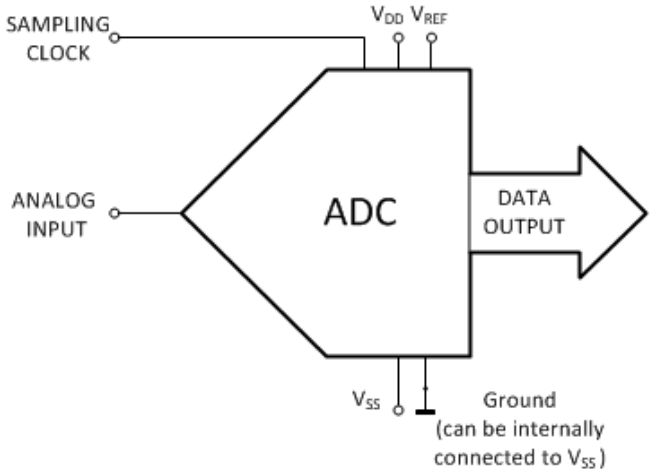


Figure 10: Basic Analog-to-digital converter

The main two signals in ADC is an analog input and data output. Input for an ADC is an analog signal such as voltage or current. Sometimes ADCs can have multiple analog inputs (i.e. multiple channels). The output data of transducer, which is a quantized signal, can be passed further using a single serial pin. In some types of ADCs, output data is served in parallel form with use of multiple pins.

Often analog-to-digital converters have some other input signals. The sampling rate can be fixed by internal oscillator in ADC, but can be also set by the external sampling clock. Usually the manufactures inform the users in datasheets, in what range the sampling frequency can be situated.

The other very important aspect of ADC is *reference voltage (V_{REF})*. This factor determines the range of voltage in which ADC can operate. V_{REF} value has to be lower or equal to the value of V_{DD}. The minimum value is defined separately for different models of ADCs.

The important part of internal structure of ADC are *sample and hold circuits S/H (*named sometimes also as *sample hold amplifiers SHA)*. They are indispensable for analog-to-digital converters to process non-DC signals[Kefauver 07]. The S/H function is added do ADC encoder as it is shown on Figure 11.

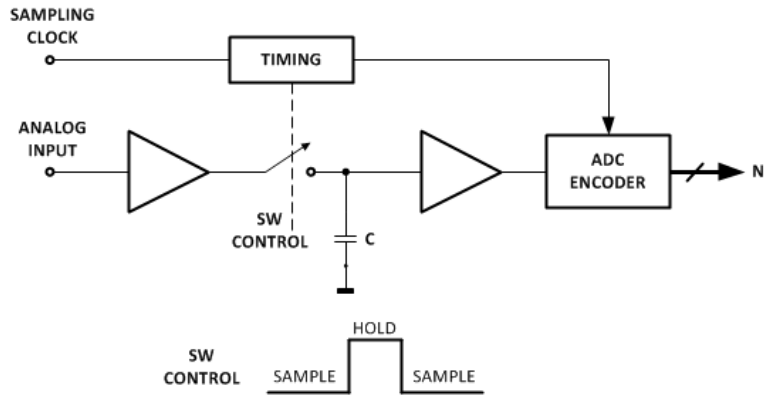


Figure 11: Sample and hold circuit

The ideal SHA is a switch driving a hold capacitor trailed by the high input impedance buffer. The input impedance of buffer must be high enough to discharge the capacitor by less than 1LSB during the hold time. The SHA samples the signal during the *sample mode,* and holds it constantly during *hold mode*. The timing has to be set in the way, that the conversion is done during hold time.

There are many architectures of analog-to-digital converters. Some of them are dedicated to work at high sampling frequencies, the others are committed to operate with high resolution. Further in this chapter will be briefly presented three popular types of ADCs.

Successive Approximation ADC

The basic structure of successive approximation ADC[Kester 05] is shown on Figure 12. On the assertion of signal START, the sample-hold block is placed in hold mode. All bits in the successive approximation register is set to zero, with exception of MSB that is set to 1. The output of SAR is passed to DAC. The output of DAC is compared now in comparator with output of S/H. If it is greater, the MSB in SAR is reset 0, otherwise it is left set. The next most significant bit is then set to 1. If the DAC output is greater than analog input this bit in SAR is reset, otherwise it is left set. This procedure is repeated until all bits in SAR have been tested. After this the content of SAR corresponds to the value of analog input and the conversion is done.

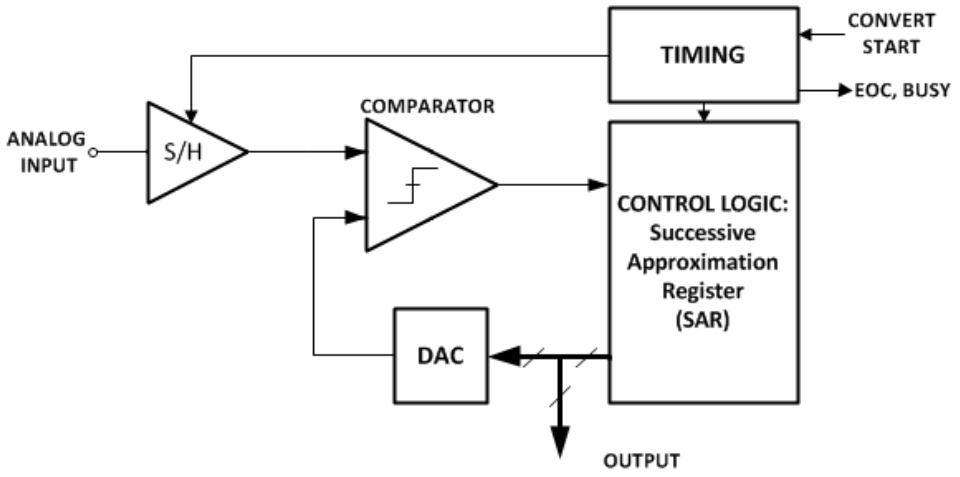


Figure 12: Basic Successive Approximation ADC

The example of timing diagram is shown on Figure 13. The conversion starts with rising (can be falling, depending on device) edge of start signal. The end of conversion is indicated with signal EOC (End of conversion), BUSY or some similar signal fulfilling this functionality. Depending on SAR ADC model, the output data can be accessible after some delay. The output data can be transmitted further in serial or parallel mode.

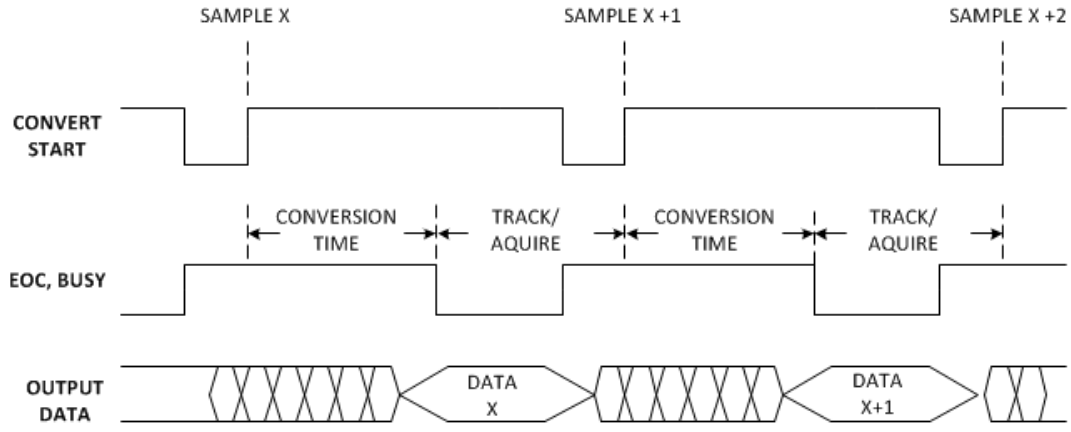


Figure 13: Typical SAR ADC timing

Assuming that N-bit conversion takes N steps, the conversion time in 16-bit SAR ADC should be twice longer that conversion in 8-bit ADC. However in SAR ADC's this condition is not met, because of internal DAC, which processing time is not linear in proportion to number of bits. For 8-bit SAR ADC conversion time is about hundreds of nanoseconds, while for 16-bit it can be counted in microseconds[Kester 05].

The successive approximation ADC's are very popular on the market of signal converters, thus there are available in many resolutions and sampling rates.

Pipelined ADC

Pipelined ADC are multi stage converters. This means that they consist numerous identical cascaded stages that are separated by sample and hold amplifiers. On Figure 14 there is a structure of pipelined ADC.

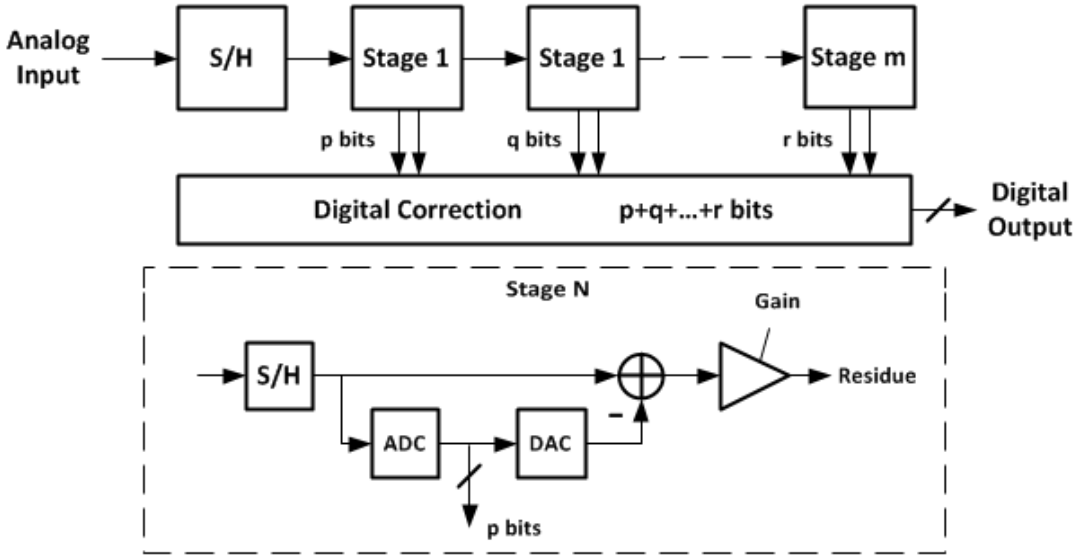


Figure 14: Structure of pipelined ADC

The first part of each stage consists a Sample Hold circuit followed by a sub ADC. This converter drives directly a p-bit DAC to reconstruct the quantized analog signal. Then this signal is subtracted from the sampled analog input signal of the stage. The result of subtraction (residue) is amplified and after this sent to following sub-converter stage.

The pipelining allows optimization between maximum sampling clock and the speed of the circuits used. In the first stage maximum accuracy is obligatory. After this stage the accuracy can be reduced with low influence to overall accuracy. The resolution of each stage depends on resolution of whole pipelined ADC.

Sigma-delta ADC

A sigma-delta ADC (Σ-Δ ADC) in this structure consists some simple analog components (a comparator, voltage reference, a switch, integrators and analog summing circuits) and more complex digital circuitry. This circuitry contain a digital signal processor (DSP) which function is a low-pass filter.

The diagram of first order Sigma-Delta ADC is shown o Figure 15.

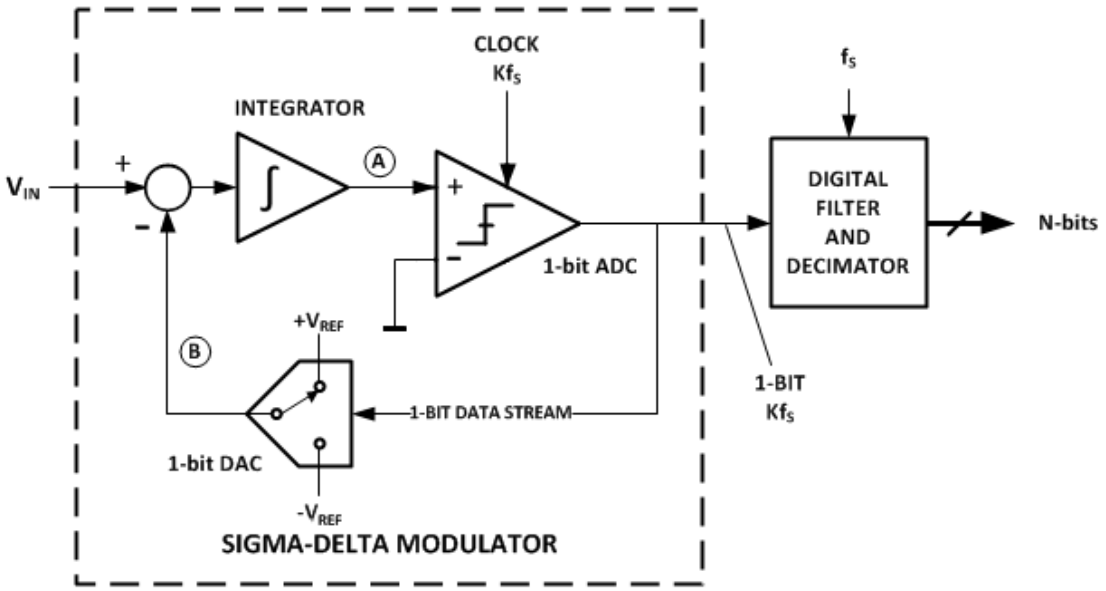


Figure 15: First Order Sigma-Delta ADC

To present how converter works it is easy to consider a DC input voltage. At point A, the integrator is repetitively ramping up or down. The output signal of the comparator (1-bit ADC) is going through the 1-bit DAC to the summing input at point B. This negative feedback loop is forcing the average dc voltage at node B to be identical to V_{IN}. This means that the average DAC output voltage has to be equal to input voltage V_{IN}. The average voltage of DAC is controlled by number of “ones” in 1-bit data stream from comparator output. With increment of input voltage in relation to $+V_{REF}$, the number of “ones” in serial bit stream also is rising, and the number of “zeros” decreasing. Likewise, if the input signal is going negative towards $-V_{REF}$, the number of “zeros” is incrementing, and number of “ones” is reducing. Therefore the serial bit stream from output of comparator is representing the average value of input voltage. Next this stream is processed in digital filter and decimator, which produces final output data.

Sigma-delta Analog-to-Digital converters are used in applications where a low cost, low bandwidth, low power and high resolution ADC is required.

2.2.3. Digital-to-analog converters

Digital-to-analog converters (DAC) are devices which transforms the data in digital form to an analog signal. The basic structure od DAC is presented on Figure 16.

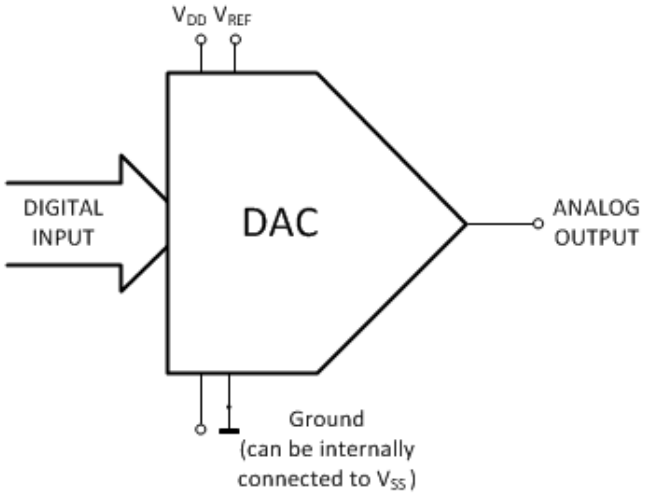


Figure 16: Basic structure of digital-to-analog converter

The digital input in DAC can be a single serial pin or a set of multiple parallel pins. The analog output is a single pin (in one channel DACs) on which the result of DAC operation is present.

Some DACs use external voltage references, while others have an internal sources of references. The value voltage reference, like in ADC determines the voltage span on which DAC can operate (i.e. the analog output voltage range).

As in case of ADCs, there also many kind of DACs architectures. The short descriptions of some types of DAC will be further presented.

The Kelvin divider (String DAC)

The basic and the simplest architecture of DAC is the Kelvin divider also known as String DAC. The example of 3-bit DAC is shown on Figure 17. The general N-bit string DAC consists of 2^N resistors and the same number of switches (usually CMOS). These switches are placed between nodes of serial connection of resistors and the output. The output is taken from a voltage divider, which is created by closing one of the switches at the moment. The control of the switches is handled by the decoder (in example it is 3 to 8 decoder) which input is N digital bits that has to be converted to analog signal.

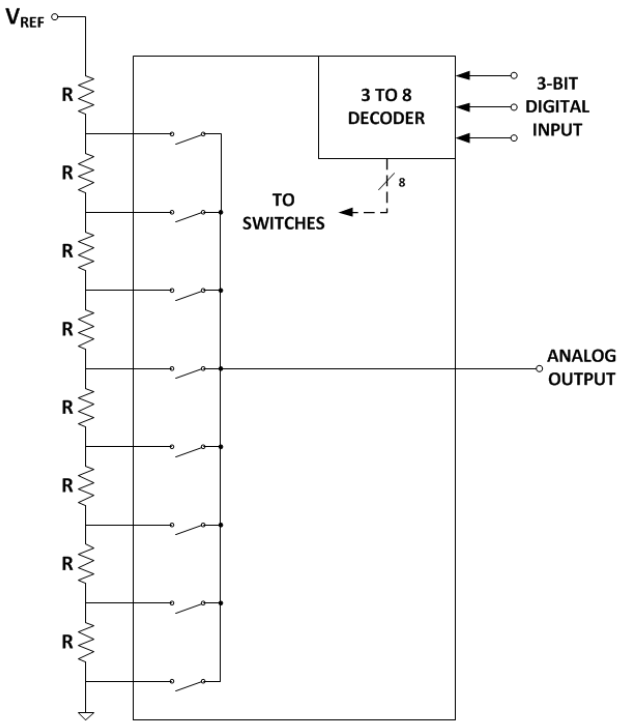


Figure 17: The Kelvin divider (string DAC)

This configuration can be linear if all resistors have the same values or nonlinear if resistances are different. The limitation of this DAC is a fact that, the number of resistors and switches needed in a structure is growing very fast (exponential) with the grow of resolution. Therefore this architecture is rather dedicated to low resolution DACs. Nowadays this type of DACs serve for example as digital potentiometers or as a components of more complex DAC architectures.

R-2R DAC

One of the most popular architectures of DAC is R-2R. It is based on resistor ladder network, composed of two values of resistors, which one has doubled resistance of the other. An N-bit DAC requires 2N resistors in his structure.

There are two ways, in which R-2R DAC can work: *voltage mode* and *current mode*. Both modes have its advantages and disadvantages.

In the voltage mode (Figure 18), the positions of switches are moved between the V_{REF} and the ground. The output voltage is taken from the end of the ladder. The voltage output is an advantage of this mode. The output impedance is constant, which is preferable when connecting amplifier to output of DAC. The disadvantage of this mode is that the switches operate within the V_{REF} and ground, which is usually a wide voltage range. This is an inconvenience from a design and manufacturing point, because the reference voltage must be driven from a very low impedance source.

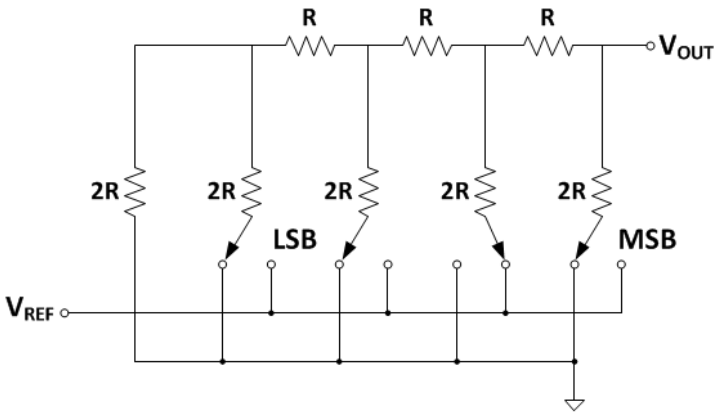


Figure 18: R-2R DAC voltage mode

In the current mode (Figure 19) the gain of the DAC can be adjusted with the series resistor next to V_{REF} input (this feature is not possible in voltage mode). The positions of switches are changing between the ground (or sometimes an "inverted output" at ground potential) and an output, which has to be held at ground potential. Usual the output is connected to an operational amplifier configured as current-to-voltage converter. Nevertheless the stabilization of this amplifier is problematic, due to non-stable output impedance of DAC.

Because the switches of a current-mode ladder network are always at ground potential, the problem of their design, which existed in voltage mode R-2R DAC is eliminated. Also their voltage rating does not influence the reference voltage rating. Moreover, since the switches are always at ground potential, the maximum reference voltage can exceed the logic voltage of circuit.

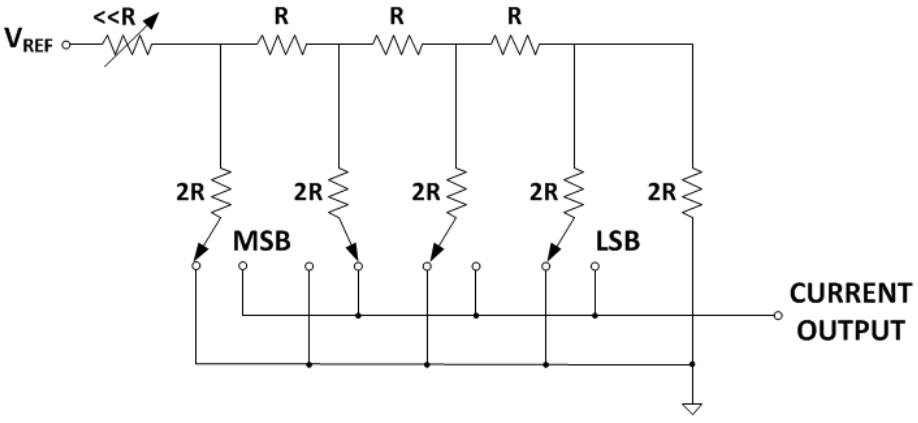


Figure 19: R-2R DAC current mode

One of the advantages of R-2R DAC is that, existence of only two values of resistors, make them more easily to manufacture in technology of integrated circuits.

Sigma-delta DAC

Sigma-delta DACs operate very similar to sigma-delta ADC, however there are some differences. The block schemes of one bit and multibit transducer is shown on Figure 20.

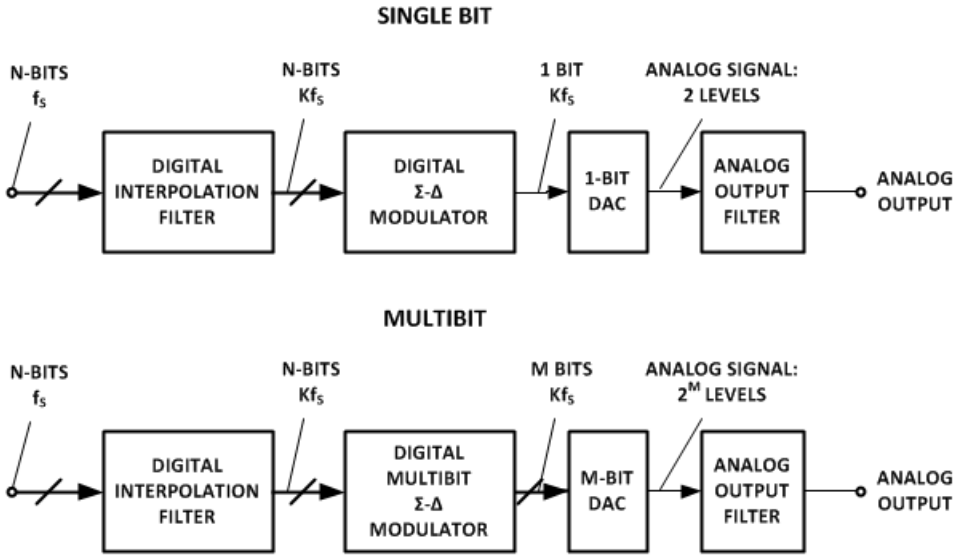


Figure 20: Sigma-Delta DACs

A Σ-Δ DAC is mostly digital. It contains of interpolation filter, which is a digital circuit that accepts data at a low rate, inserts zeros at a high rate, and then applies a digital filter algorithm and outputs data at a high rate[Kester 05]. A Σ-Δ modulator acts like low pass filter to the signal and like high pass filter to quantization noise. It converts the data to a high speed bit stream. The 1-bit DAC (in single bit converter) is acting like a switch, that connects his output either to positive or negative reference voltage. The output is filtered with low pass filter.

2.2.4. Communication Protocols

Communication between MCU and ADC or DAC can be provided by various types of protocols. Below there will be described most popular among them.

SPI

Serial Peripheral Interface (SPI) [SPI.specification] also known as Microwire is primarily used for synchronous serial communication between host processor and peripherals. Nevertheless it is possible to connect two processors via this protocol.

SPI specifies four signals for communication: clock (SCK), data output (SO), data input (SI) and chip/slave select (CS/SS). Devices communicate using a master/slave relationship. The master provides a clock signal and controls the chip select line i.e. activates a slave when wants to communicate with him. In Figure 21 there is an example connection between master and single slave.

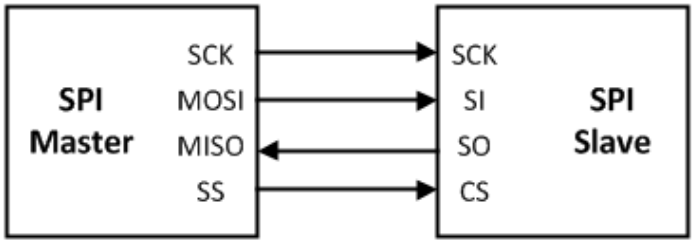


Figure 21: SPI connection between master and single slave

MOSI(Master Output Slave Input) carries data from master to slave and MISO(Master Input Slave Output) back from slave to master.

A SPI device is basing on idea of shift register. Commands and data values are serially transferred, pushed into shift register and then internally available for parallel processing. The length of shift registers can vary. The most popular is 8 bit length, but also in some devices exists 16 and 32 bit lengths.

There is also a possibility to connect multiple independent slaves to one master. This solution is shown on Figure 22. The SCK, MOSI and MISO lines are common to all slaves. Only the chip select lines are connected separately to provide for the master possibility to choose with which slave it is communicating at the moment.

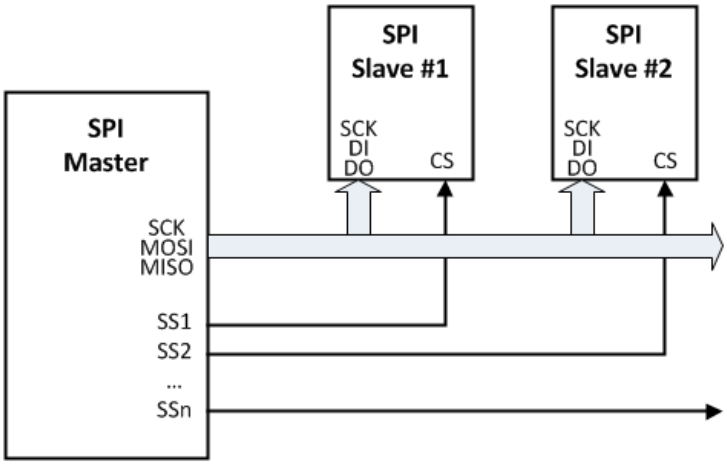


Figure 22: Connecting multiple slaves to one master

Each transmission starts with a falling edge of SS controlled by master, which selects the particular slave. If SS/CS is pulled high, SPI device is not selected, and its data output goes in high-impedance state(Hi-Z).

During transmission, commands and data are controlled by SCK and SS. In master device, the clock can be configured to shift data either on rising on falling edge depending which kind of shifting is implemented in slave device.

On Figure 23 it is shown an example of reading 8 bit status byte from SPI device.

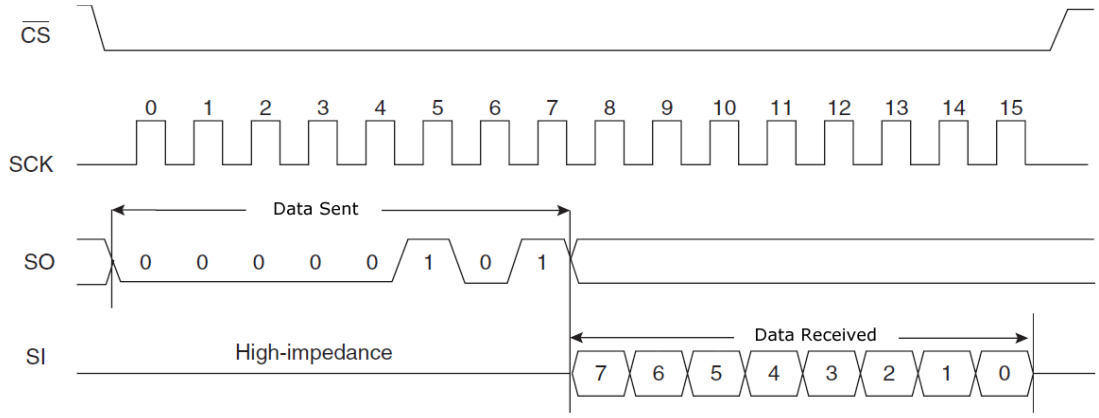


Figure 23: Timing sequence in SPI communication

The CS pin is pulled low by master, which activates the slave and starts producing the clock signal. First master is sending on SO pin 8bit address (Data Sent) of register that he wants to read. At this time SO pin of slave (which is SI of master) is in Hi-Z state. To complete the reading process, master has to send a second (dummy) byte to capture the data on SI pin (Data Received). Dummy byte is usual composed by only "1" values (in this case eight of them). After completing the reading process the CS pin is pulled high to end the communication with slave.

I^2C

I^2C (also known as Inter Integrated Circuit bus) interface[I2C.specification] is also bidirectional serial communication protocol working in Master/Slave mode. In Figure 24 there is a typical connection between master and slave devices.

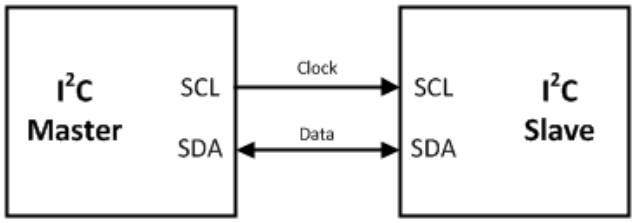


Figure 24: Connection between master and slave in I^2C protocol

As distinct from SPI it is using two wires to communicate: a clock (SCL) line which is controlled by master device and a bidirectional data line (SDA).

It is possible to connect more than one device to master. Each of them is recognized by unique address which is sent on the beginning of transmission. Devices can be transmitters, receivers or both, depending on their function in all system. I^2C also provides a possibility to connect multiple master devices. This means that in system can exist few devices which can take over the control of steering the signals.

During transmission of data, signal on line SDA has to be stable, when SCL is in high state. Changes on line SDA during high state of clock are interpreted as control signals. Therefore some rules of transmission were settled:

- START condition – falling edge of SDA during high state of SCL
- STOP condition - rising edge of SDA during high state of SCL
- Data valid – states of line SDA represents valid data, when (after START condition) they are stable during high state of SCL signal. Change of data can occur during low state of clock signal.

On the Figure 25, there is a timing sequence during transfer of data.

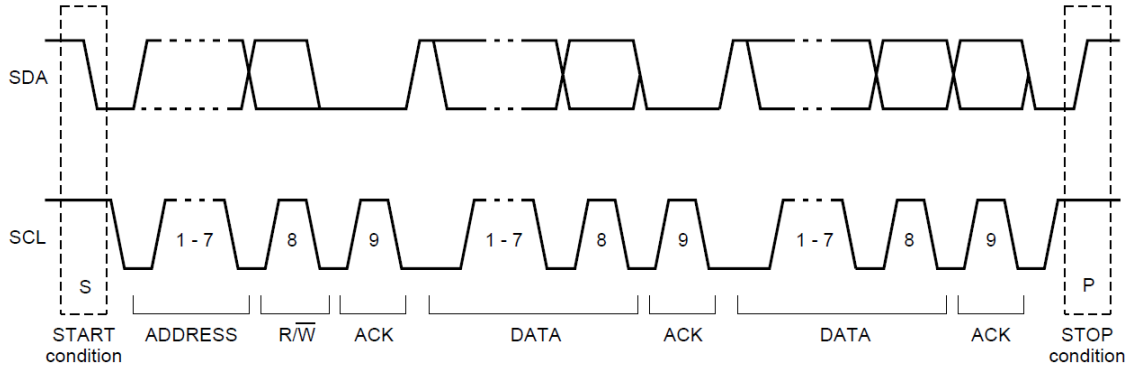


Figure 25: Timing sequence in I²C communication

The transmission starts by START condition after which, it is sent 7bit address of slave device. Direction of data transfer is determined by R/W bit. Then, takes place the proper exchange of data, confirmed by slave after each byte by acknowledge but (ACK). The end of transmission is determined by STOP condition.

I²S

I²S is a serial bus interface[I2S.specification] dedicated for connecting digital audio devices, mostly for stereo signals. Like the previous protocols, I²S is using master/slave organization of devices. As distinct from them I²S is providing one direction data transfer. The master can be a device which transmits data or the one which receives it. In Figure 26 there is typical connection between transmitter and receiver in two possible versions.

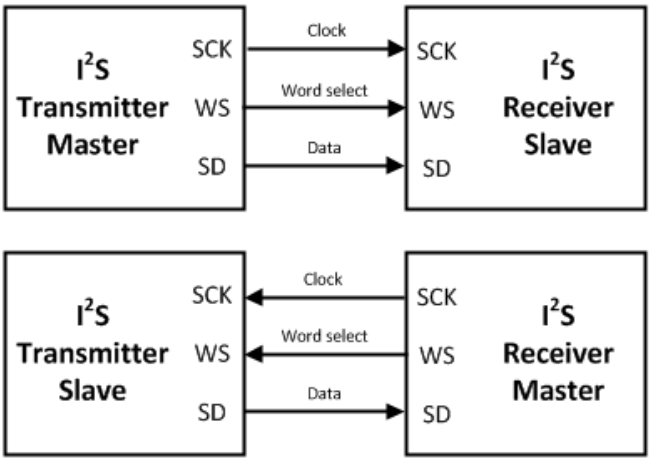


Figure 26: Connection between master and slave in I²S bus

The protocol is using 3 lines to communicate: clock line (SCK), word select line (WS) and data line. Generation of SCK and WS signals is the responsibility of master device.

The serial data is sent in two parts: left and right channel with the MSB first. It is possible that transmitter and receiver have different word lengths and it is not necessary for transmitter to know how many bits receiver can handle. In the other hand receiver does not to be informed how many bits transmitter is sending. Data transmission can be synchronized to be sent either on falling or rising edge of CLK signal. However serial data has to be latched in receiver on leading edge of clock signal.

Word select line is responsible for switching between transmitting left and right channel. During low state of WS, transmitter is sending left channel and during high state right channel. The WS signal does not to be symmetrical, which means that more audio samples can be sent by one of the channels. In slave device, this signal is latched on the rising edge of clock signal. The WS is changing one clock signal before sending data from specific channel.

The Figure 27 represent the timing sequence during transmitting data.

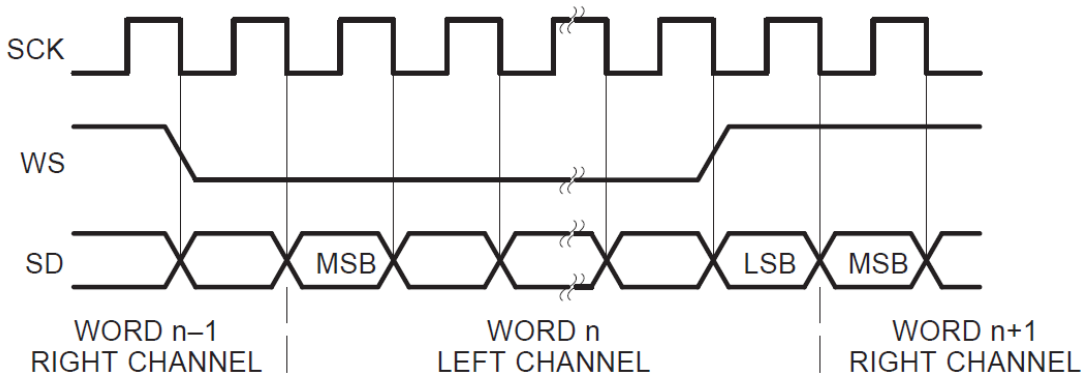


Figure 27: Timing sequence in I2S protocol

As it can be seen, the data of left and right channels are sent alternately. There is delay of one clock cycle between changing the channel and sending MSB of next sequence of data.

Comparison of protocols

Each serial communication interface has its advantages and disadvantages. In Table 3 there is a comparison of three described protocols[Jasio 08].

Table 3: Comparison of serial communication protocols

	SPI	I^2C	I^2S
Max bit rate	20 Mbit/s	1 Mbit/s	3,125 Mbit/s
Max bus size	Limited by number of pins	128 devices	1 device
Number of pins	3 + (n × CS)	2	3
Advantages	Two way transmission, high speed, low power, not limited to 8-bit words.	Two way transmission ,small pin count, allows multiple masters	Simple application for transmitting stereo audio signals
Disadvantages	Single master, requires more pins than I^2C	Slowest, limited size of bus	Connection between only 2 devices, one way transmission
Typical application	Direct connection to many common peripherals on the same PCB	Bus connection to peripherals on the same PCB	Transmitting stereo audio signals between two devices
Examples	Serial EEPROMs, ADC, DAC converters, temperature sensors, etc.	Serial EEPROMs, ADC, DAC converters, temperature sensors, etc.	ADC, DAC converters dedicated for stereo audio signals, DSP

The conclusion is following: if in a system high speed of transmission or connecting more than 128 slaves is necessary, it is better to use SPI protocol. Otherwise, for application that can work at slower data rate I^2C can be used. As it was stated I^2S interface is strictly dedicated for stereo audio signals transmission and it can be applied in this kind of systems.

2.3. Signal Conditioning

A digital processing unit is usually dedicated to fulfill some specific task. This means, that it is prepared for an input signal that has certain characteristics in terms of amplitude, frequency etc. Also, the output signal should be prepared for interaction with a device

connected to the output of digital processing unit. To ensure the correct behavior of device it is necessary to use the analog signal conditioning circuits.

Input conditioning block is driven directly by analog input signal. It is usually performing two functions. First is fulfilled by low pass filter which cuts off the frequencies that are not useful for processing unit. In audio systems, pass band is usually limited with the range of human auditory system, but can be also narrower for example in speech processing applications. Filters can either be passive or active. Usually active filters are adapted, because they have better shapes of response and quality factors than passive ones.

Second function in input analog conditioning block is modification of the signal amplitude. Usually, the input signals have low amplitude and need to be amplified. The amplification is indispensible to exploit full dynamic range of an ADC. However, the signal should fit the input range of ADC. Too low amplitude, degrades the signal to noise ratio(SNR)[Self 10] as the top bits are not used. Too high amplitude will cause undesirable clipping of signal. Most of ADCs are using only positive reference voltage. Input signal often has both positive and negative values. For this reason a circuit that is adding an offset to signal is in some cases essential. To perform those tasks, usually are chosen circuits based on operational amplifiers. They are easy to implement, because it exists many of operational amplifiers produced as integrated circuits. Also only a few external elements have to be added to create complete circuitry.

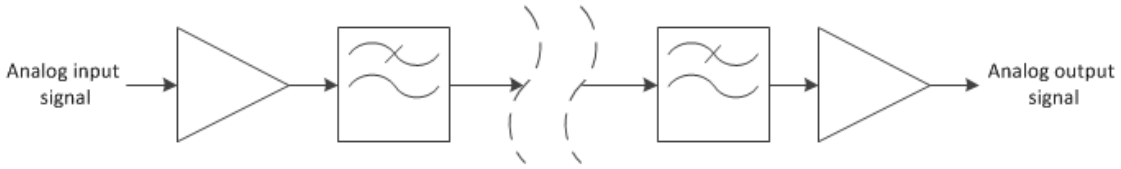


Figure 28: Analog signal conditioning

The output signal conditioning has very similar structure to input block. It also consists of low pass filter, that removes high frequencies caused by aliasing. The cut off frequencies of output and input filters is usually equal. After filter is often placed function block which modifies the amplitude. It can be a circuit, which amplifies or attenuates the signal, depending on destination of output. If the offset was added to signal in the input conditioning block, it can be removed by adding a serial capacitor. Nevertheless it is recommended to place an operational amplifier on the output the circuit, which will separate the device from the output.

CHAPTER 3
DESIGN OF WIRELESS AUDIO UNIT

This chapter is divided in two sections. It begins with description of "PIC32 Module" and ends with specification of the design of Wireless Audio Unit. This partition is caused by the fact that whole Wireless Audio Unit device was created in two parts. Firstly PIC32 Module was designed in collaboration with my colleague Marcin Janiszewski, who has also applied it in his project. After this, the proper Wireless Audio Unit – the main objective of this dissertation was developed.

3.1. PIC32 Module

The idea of creating PIC32 Module was to design a universal unit containing PIC32 microcontroller, which could be easily used in many projects. The problem of using 32bit microcontrollers in prototypes and simple projects is that all of them, with few exceptions exists only in SMD packages. This implicates the necessity of producing precise PCB for each project. The main advantage and convenience of using PIC32 Module, is that microcontroller is already soldered on PCB and all its pins are connected to output pins of PIC32 Module which are "trough-hole" pins. This makes, that the whole module can be treated as PIC32 in DIP package. The basic block schematic of PIC32 Module is shown of Figure 29.

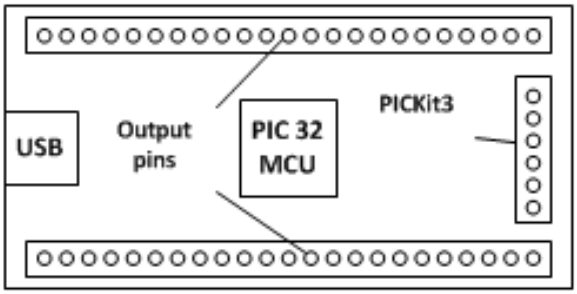


Figure 29: PIC32 Module block schematic

On PIC32 Module board were located all necessary components, so that just after providing power supply it is ready to use. Design includes also a USB functionality, therefore all required circuitry was placed in board. Moreover PIC32 Module is adapted to use with PICkit 3 in-circuit debugger and programmer.

The spacing between two rows of output pins have a value, that PIC32 Module can be easily used on universal boards and breadboards.

3.1.1. Choice of microprocessor

PIC32 module consists a 32-bit microprocessor from family of PIC32. This family is composed from 34 models, divided in five groups: PIC32MX3, PIC32MX4, PIC32MX5, PIC32MX6 and PIC32MX7. Difference between microcontrollers in each group is possession of various peripherals and functionality. Generally the power of MCUs (in those terms) grows with the order of group (form 3 to 7). In each of these groups microprocessors exists in in two kind of packages – 64 and 100pin.

For PIC32 Module, the selected model was PIC32MX795F512H from group PIC32MX7[PIC32.datasheet]. This version of PIC32 is the most powerful microcontroller in 64pin case form PIC32 family. Since the idea of creating PIC32 module was not only applying it to WAU project, but also to other future designs, the decision of using the best possible chip was made.

The properties of PIC32MX795F512H microcontroller are as follows:

- 64 pin case
- 512KB Flash memory + 12KB Boot Flash
- 128KB SRAM memory
- 80MHz of maximum operating frequency
- 3 channels of SPI
- 4 channels of I^2C
- 6 channels of UART (Universal Asynchronous Receiver and Transmitter)
- 8 general and 8 dedicated DMA channels
- USB 2.0-compliant full-speed device and On-The-Go (OTG) controller
- 10/100 Mbps Ethernet MAC with MII and RMII interface
- 5 channels of CAN2.0b
- 5 capture inputs, 5 compare outputs and 5 PWM outputs
- 16 channels of 10bit ADC 1Msps
- 2 analog comparators
- One 32bit and five 16bit timers
- RTTC
- Parallel master port
- JTAG Program, Debug, Boundary Scan

The PIC32MX795F512H microcontroller offers therefore many possibilities for system designers and programmers and can be used in many complex projects.

3.1.2. Schematics

The circuit of PIC32 Module is shown of Figure 30. It contains few parts which are described below:

- P1 - Junction to PICkit3. It was realized using 6-pin male header. The connection to microcontroller was taken from PICkit3 User's Guide [PICkit3].
- J1 – USB socket. For minimizing the size of module, it was selected the mini USB type AB socket. The connection to PIC32 was found in PIC32 Family Reference Manual[USB.specification]. As it can be seen, to use USB functionality in PIC32 it is necessary to make four connections between socket and microcontroller and include two capacitors (C7 and C9).
- Y1, Y2 – quartz crystals. To enhance functionality, two different sources of clocking were placed. Y1, that is 8MHz crystal is the main clock supply, from which PIC32 can obtain (with correct setup of configuration bits) 80Mhz of operating frequency. The frequency of second crystal (Y2) is 32.768kHz. This part is required to use RTTC (Real time clock and calendar)[PIC32.datasheet]
- P1 and P3 – output pins. These two 23-pin male headers are connected to all ports of microcontroller. On pins going form case of PIC32, particular ports are not grouped (Port D, Port E etc.). This was done in PIC32 Module, so that individual ports can be more easily accessed and used. Headers also includes the VDD and VSS pins.
- S1 – reset button. This switch, when pushed shorts to ground the MCLR pin. This results in reset of microcontroller.
- D1 – Power indicating diode. A low current LED, which is tuned on, while the power supply is provided to PIC32 module.
- Capacitors C1-C5 – decoupling capacitors. Their values and amount was taken from PIC32 datasheet[PIC32.datasheet].
- Resistors R1,R3 – connection from PIC32 datasheet[PIC32.datasheet].

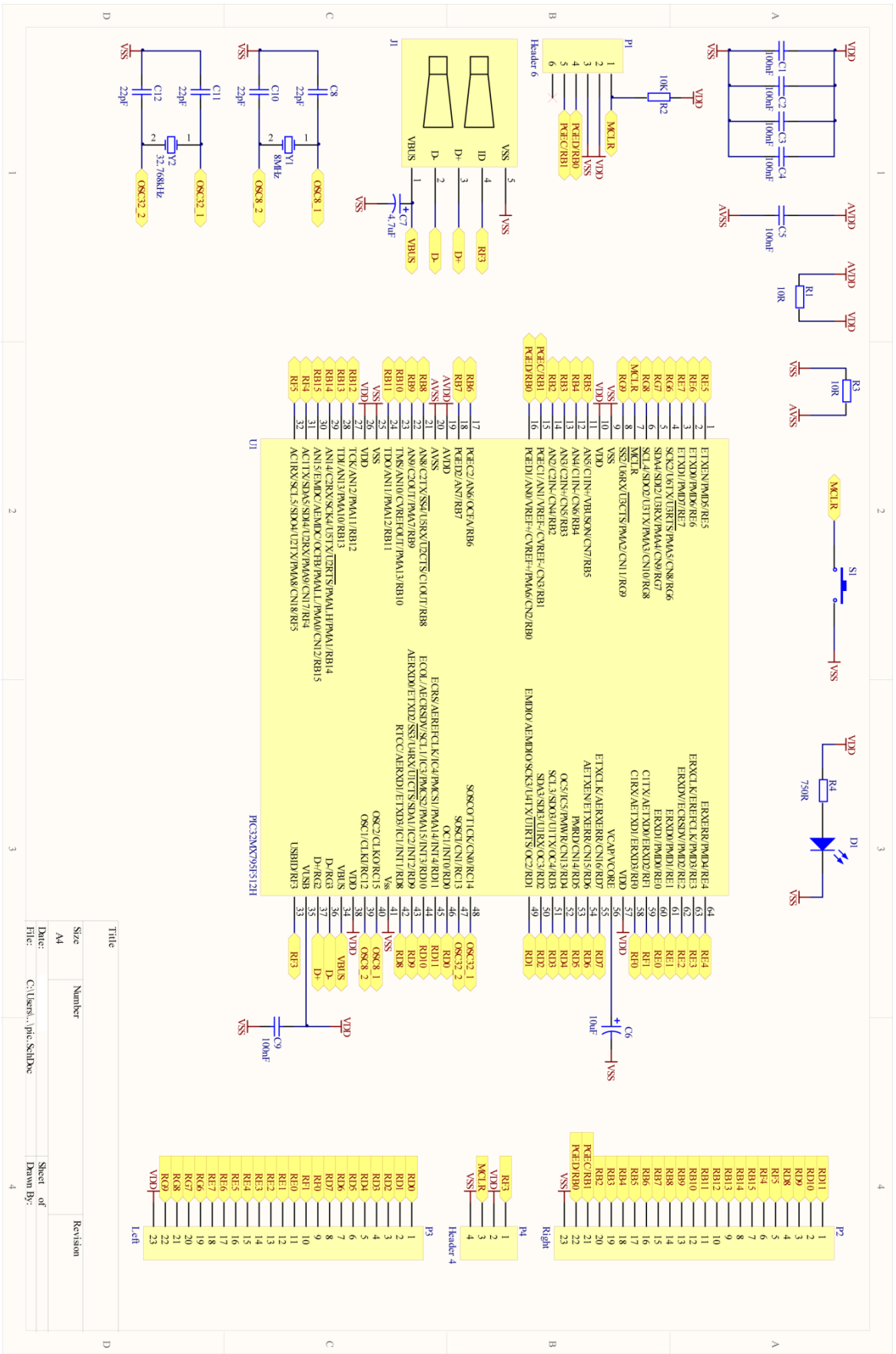


Figure 30: Schematic of PIC32 Module

3.1.3. PCB

The project of PCB was created using Altium Designer software. The printouts are present in appendix.

Following rules were applied in PCB design

- Track width:
  - 20mil (0,508mm) for VDD, VSS, AVDD, AVSS tracks
  - 10mil (0,254mm) for other tracks
- Clearance constraint between tracks:
  - 20mil (0,508mm) for VDD, VSS, AVDD, AVSS tracks (in some places there are some exceptions and clearance is 10mil)
  - 10mil (0,254mm) for other tracks
- Size of vias : diameter 1mm, hole size 0,5mm

To minimalize the size of board, all used elements (except for headers) are in SMD packages. The capacitors and resistors are in 0805 package which dimensions are 2.0 × 1.3 mm.

3.2. Wireless Audio Unit Design

3.2.1. General architecture

As it was written at the beginning of this document, the task of Wireless Audio Unit, is to substitute the cable connection between audio equipment. To achieve this, it was designed two devices: transmitter and receiver. The general block diagram is presented on Figure 31.

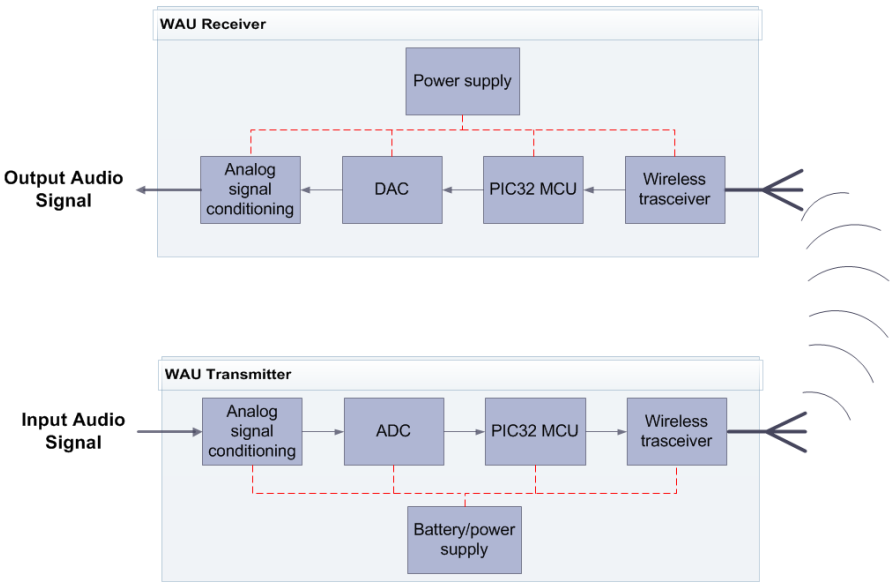


Figure 31: General architecture of Wireless Audio Unit

The input of transmitter is analog audio signal. Since the WAU is mainly dedicated to work with musical instruments (electric guitar), expected input is a wave of maximum amplitude $2V_{PP.}$ It also means, that WAU has only one input one output audio channel. The signal from input goes to "Analog signal conditioning" block. This section performs two functions: cutting off high (not audible) frequencies and preparing the signal for its digitalization. This preparation includes the amplification of signal and adding a DC offset to it. After conditioning, the signal is quantized in ADC. The samples of audio signal are being gathered by PIC32 microcontroller. PIC32 is performing a lossless compression on gathered stream of data. After this, the compressed samples are sent to wireless transceiver, configured as transmitter.

In WAU Receiver part, the transceiver is obtaining the data sent by WAU Transmitter. This data, which are compressed samples, are directed to microcontroller. In PIC32 is done the decompression procedure. The reproduced samples are sent to DAC. The analog signal produced by this transducer is now being conditioned. The last block is removing the constant component and amplification to cause that output signal is identical to input.

The power supply for both parts are provided by AC adapter. In WAU transmitter there is possibility to switch to battery supply.

The specification of project can be stated as follows. The signal reproduced by the receiver has to be an exact copy of input signal. To achieve this, the proposed quality of digital processing is 16 bit resolution and 40kHz of sampling rate. Also the compression which is performed in microcontroller has to be based on lossless algorithm.

3.2.2. Choice of components

To fulfill those requirements and realize the project, the components were carefully selected on the ground of characteristics, functionality and price.

In both input and output signal conditioning it was necessary to use a low-pass filter to sift high, not useful frequencies. To implement this filter, two solutions were possible. First option, was to design an active filter in one of the known topologies (Sallen-Key, MFB, etc.) based on operational amplifiers. Second opportunity was to use a ready integrated circuit , which serves as low-pass filter. In order to save more space on the PCB and have more confidence on reliability of filter, second option was chosen.

In both input and output signal conditioning, it was used a low-pass filter LTC1569-7 from Linear Technology. This is a 10^{th} order analog filter featuring linear phase in the pass band. The cutoff frequency is configurable by the external resistor, up to 300kHz. According to datasheet the filter attenuation is 57dB at $1.5f_{CUTOFF}$ and 80dB at $6f_{CUTOFF}$. The frequency response for three different cutoff frequencies is shown on Figure 32 [LTC1569-7.datasheet]. The chip can work at single or symmetrical power supply (from 3 to ±5).

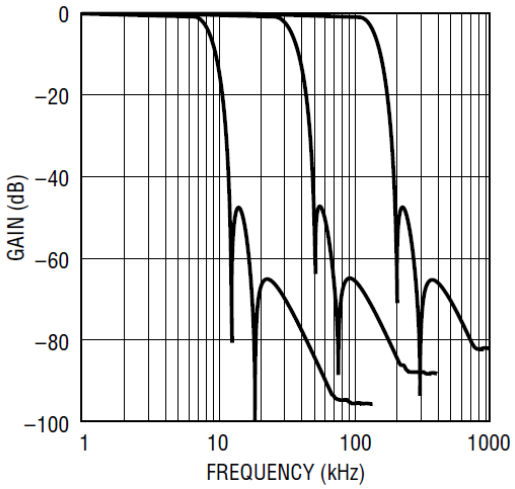


Figure 32: Frequency Response, f_{CUTOFF} = 128kHz/32kHz/8kHz

To perform other signal conditioning such as amplification it was necessary to choose an operational amplifier. There are plenty of them available on the market, almost each with different characteristics. Most important properties, according to which the amplifier was chosen, was a rail-to-rail feature, single supply operation at 3.3V, low supply current and sufficient bandwidth. The operational amplifier that meets these conditions is AD8531 from Analog Devices[AD8531.datasheet]. It has 750µA supply current and 3MHz of bandwidth.

The choice of ADC was limited by four factors:

- 16 bit samples resolution,
- minimum 40kHz sampling rate,
- single supply of 3,3V,
- serial output, compatible with SPI protocol.

Among the chips available on the market, it was selected LTC1864L ADC from Linear Technology. It is one channel, successive approximation ADC with 16 bit resolution and maximum sampling rate 150kHz. The communication is performed via 3-wire interface, that fits SPI standard. The supply voltage can have value between 2,7 and 3,6V. The maximum speed of conversion is 2µs per sample.

The same requirements were put to the selection of DAC. The searches led to the chip from Linear Technology - LTC2641. The resolution of this transducer is 16 bit and it is compatible with SPI protocol. It can be supplied from a voltage of value between 2,7 and 5,5V.

Both ADC and DAC are low-power devices. Supply current for ADC is around 100µA (for sampling rate 40kHz) and for DAC 120µA.

To provide a wireless communication between two parts of Wireless Audio Unit, a proper wireless transceiver had to be selected. The choice was dictated by conditions:

- The throughput of transceiver must be sufficient to let the transmission and reception of audio data at given quality. Since the resolution of samples is 16 bits, and expected sampling rate is 40kHz, the predicted bitrate of audio data can be calculated:

$$Bitrate = 16\ bit\ \times 40kHz = 640\ kbit/s$$

  The compression, that would be done in microprocessor is estimated to be in order of 50% of size of input stream (see Chapter 4.4.5). Therefore the throughput of transceiver has to be at minimum 320 kbit/s.
- The second aspect of choosing transceiver, was the range in which two devices can communicate. The minimum condition was few tens of meters.
- Searches was also limited to transceiver modules, on which is already included the antenna, Balun and RF matching. In RF applications the accuracy of circuit of transceiver is essential to make them work properly. In those type of units, all passive components have precise values, inductances are often created with the use of track on PCB and the module is already tested.
- As the other elements, the power supply should has a value of 3.3V
- The communication between PIC32 should be provided via SPI interface

The selected part is QFM-TRX1-24G from Quasar company[QFM-TRX1-24G.datasheet]. It is ready to use module with internal antenna, possible to mount on PCB board. The photo of this part is on Figure 33.

Figure 33: Photo of QFM-TRX1-24G transceiver module

This module is based on CC2500 chip from Texas Instruments[CC2500.datasheet]. It is a 2,4GHz RF transceiver, with maximum throughput of 500kbps. The output power is

programmable, and at +1dBm the range is up to 50m in open area. The power supply is between 1,8 and 3,6V. The communication with transceiver (i.e. configuration and data interface) is done through the SPI interface.

The assumed value of voltage supply for both WAU Transceiver and Receiver was 3,3V. The AC adapter, which was provided to use with project, produces 5V of DC voltage. To obtain 3,3V, it was necessary to use a power management system such as linear voltage regulator or DC-DC converter. Due to better efficiency, the second option was selected. The search led to TSR 1-2433 from Traco Power company [TSR-1.datasheet]. This component is a step-down switching regulator with high efficiency up to 96%. It produces 3,3VDC from wide input voltage from 4,75 to 36V. The maximum output current is 1A, which is sufficient to designed device. The main advantage of this part, in relation to others available on market, is that any external element such as inductor or diode is not necessary to be added. It is only recommended to place one input capacitor.

Since in WAU Transmitter part, was added also possibility to power the device from batteries, proper cells had to be selected. To minimize the space occupied by them and maintain the sufficient value of voltage, it was chosen two 3V batteries, in 2032 case. After mounting them in proper battery holder, where they are connected in series the total voltage is 6V. The capacity of these batteries is 225mAh.

The choice of microprocessor was previously made in design of PIC32 Module and this module was used in both WAU Transmitter and WAU Receiver.

3.2.3. Schematics

After selecting the components, it was necessary to connect them together on schematics. As two sub devices supposed to be created, the schematics is divided into two parts: Transmitter and Receiver.

The schematics as well as PCB was created in Altium Designer software.

WAU Transmitter

On Figure 34 there is a block diagram of WAU Transmitter part. Each block represents a separated circuit. Connections between blocks signifies the signals, which are connecting those circuits together in one schematic of WAU Transmitter.

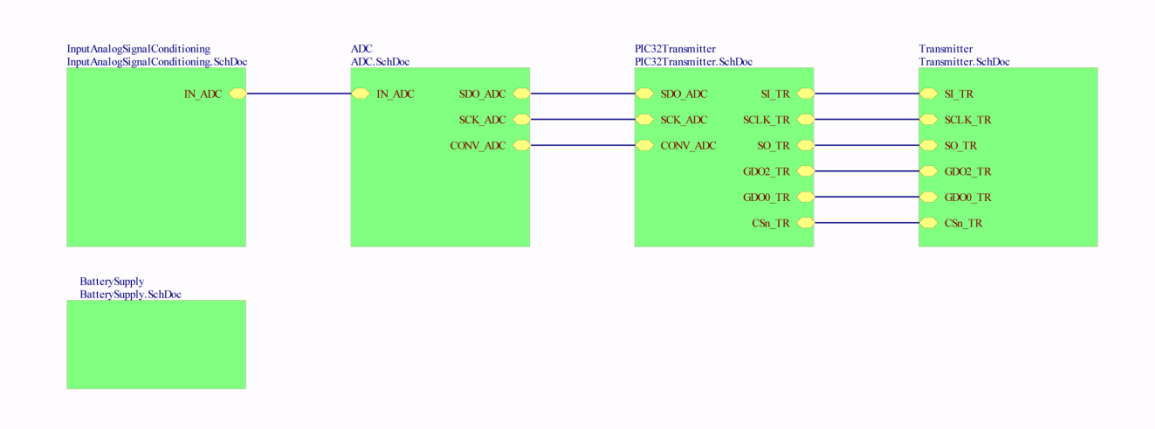


Figure 34: Block diagram of WAU Transmitter

Battery Supply

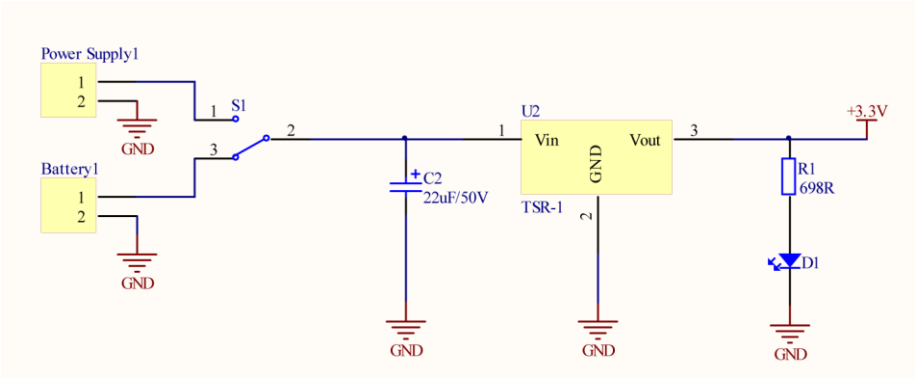


Figure 35: WAU Transmitter: Power supply circuit

As it was written, WAU Transmitter has two possible sources of supply: AC adapter (connected to Power Supply1 junction) or batteries (connected to Battery1 junction). The selection is done using the 3-pole switch S1 (ON-OFF-ON).

The place and value of capacitor C2 was taken from TSR-1 datasheet.

The role of LED D1 is to indicate if the power supply is provided to circuit. The forward current of this diode is I_f = 2mA and forward voltage U_f = 1,9V. Taking this into account, the value of resistor R1 was calculated:

$$R1 = \frac{U_{R1}}{I_{R1}} = \frac{3{,}3V - U_f}{I_f} = \frac{3{,}3V - 1{,}9V}{2mA} = 700\Omega$$

The closest value form available resistors is 698Ω.

Input Analog Signal Conditioning

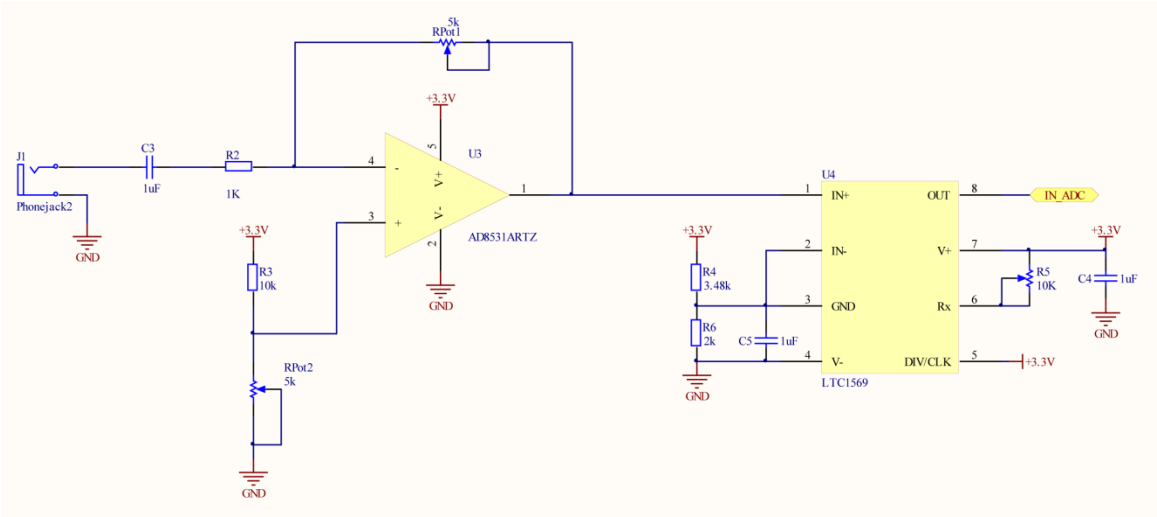


Figure 36: WAU Transmitter: Input Analog Signal Conditioning circuit

The input analog audio signal is connected to junction J1, which is a mono Jack socket. The possible DC component in input signal is eliminated using the C3 capacitor. Further the signal is amplified in inverting operational amplifier configuration. The amplification depends on setting of potentiometer RPot1. Its value is given as:

$$K_u = -\frac{RPot1}{R2}$$

Therefore the maximum possible amplification is -5V/V (for RPot1 = 5k).

During amplification, also second operation on signal is done. With use of voltage divider composed by R3 and RPot2, connected to non-inverting input of operational amplifier, a DC offset is added to signal.

This two changes of signal are performed to prepare the signal for the input of ADC. The amplification is done to exploit the full range of ADC resolution. Adding DC offset is caused by the fact that ADC is single supplied, therefore it cannot sample the negative voltage signals.

After this stage, the signal is put on the input of low-pass filter LTC1569-7. External elements are connected according to its datasheet. Feeding the DIV/CLK pin with 3,3V and the value of potentiometer R5 are settling the value of cut-off frequency given by the equation:

$$f_{CUTOFF} = \frac{128kHz(\frac{10k\Omega}{R5})}{16}$$

After filtration, the signal is going to the input of ADC

ADC

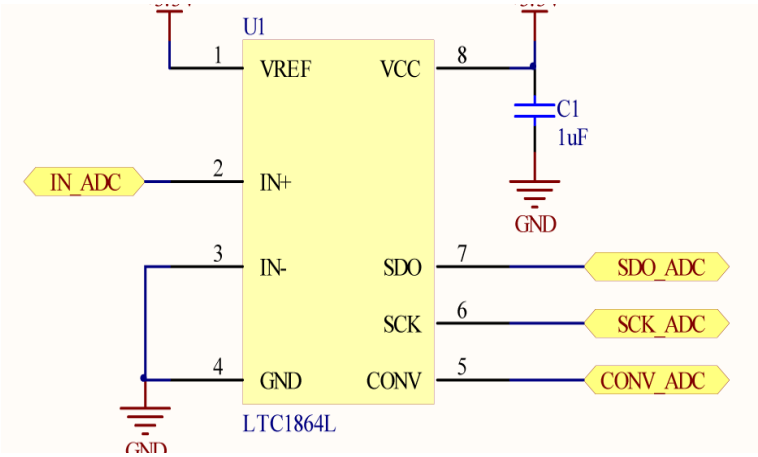


Figure 37: WAU Transmitter: ADC circuit

As it can be seen, only one external element, which is decoupling capacitor C1 had to be added to ADC circuit. LTC1864L possess a differential input, but in WAU the input signal is referred to the ground, therefore only positive input (IN+) was used. The reference voltage of ADC, which is determined by the potential on pin VREF was set to value 3,3V. The communication with microprocessor is performed with use of three wires, compatible with SPI interface. SDO is the serial output of ADC, SCK is SPI clock input and CONV is equivalent of CS signal in SPI protocol.

PIC32

Connections between PIC32 Module and the rest of the circuit are shown on Figure 38.

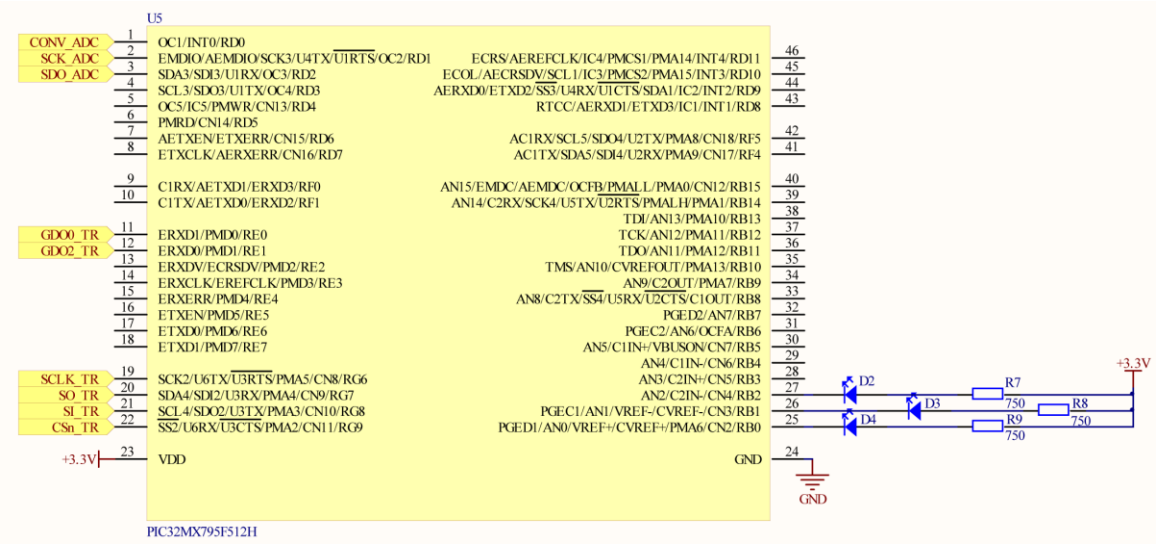


Figure 38: WAU Transmitter: PIC32 Module circuit

Signals: *CONV_ADC*, *SCK_ADC* and *SDO_ADC* are connected to pins of PIC32, which share the function of SPI channel. According to numeration in PIC32, the number of this channel is 3.

Signals *SCLK_TR*, *SO_TR*, *SI_TR*, *CSn_TR*, *GDO0_TR* and *GDO2_TR* are associated with control of transceiver.

Additionally three diodes D2, D3, D4 were connected to first three pins of port B. Their role is to assist during testing the device.

Transceiver

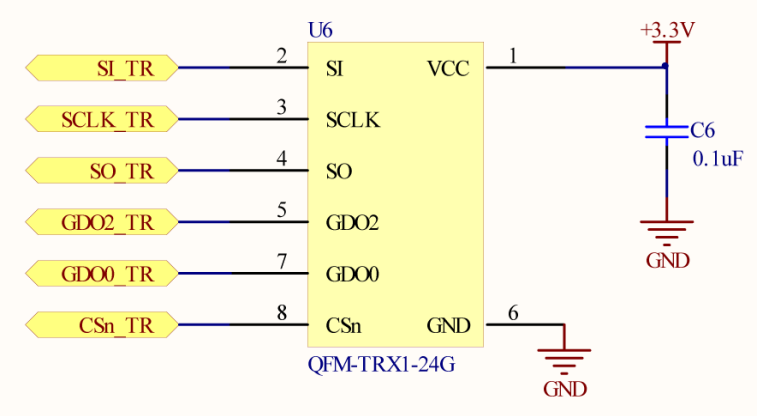


Figure 39: WAU Transmitter: Transceiver circuit

Since all necessary elements are placed on transceiver module board, only additional decoupling capacitor C6 was placed.

The signals *SCLK_TR*, *SO_TR*, *SI_TR*, *CSn_TR* which are connected to PIC32, serve as SPI interface. The SPI channel is 2. Two additional connections *GDO0_TR* and *GDO2_TR* are digital pins for general use. They are helpful during communication between this module and microprocessor, because they can indicate some states in which transceiver is at the moment. Moreover these pins can serve as serial communication interface, when SPI is not used. The detailed theory of operation of this transceiver will be described in chapter 5.

WAU Receiver

As in the WAU Transmitter part, block diagram shown on Figure 40, represents particular schematics connected together with drawn connections. Diagram is also composed from five circuits and some of them are very similar to those from WAU Transmitter part.

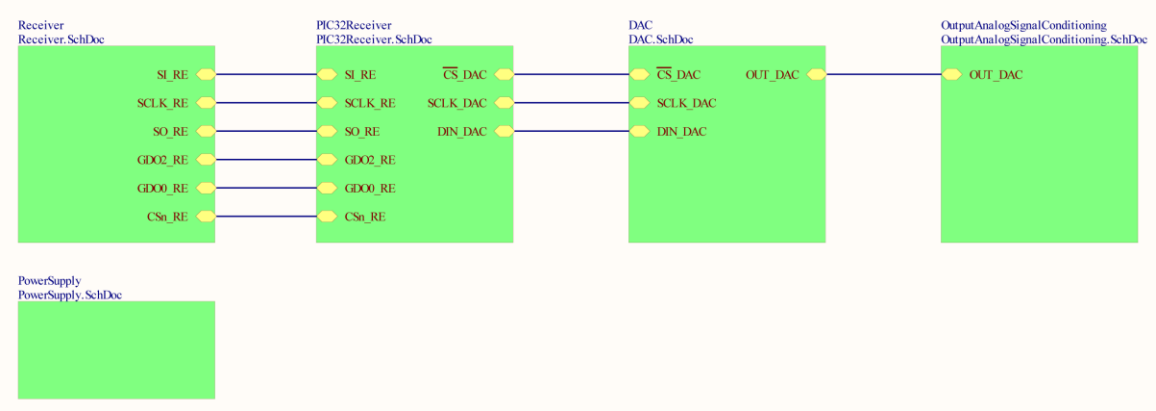


Figure 40: Block diagram of WAU Receiver

Power Supply

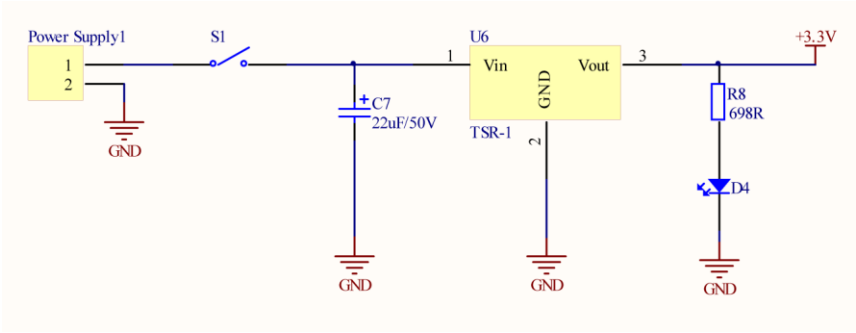


Figure 41: WAU Receiver: Power supply circuit

The circuit of power supply almost identical to this from the WAU Transmitter part. The only difference is the removal of the battery supply and replacing the 3 pole switch to 2 pole S1.

Transceiver

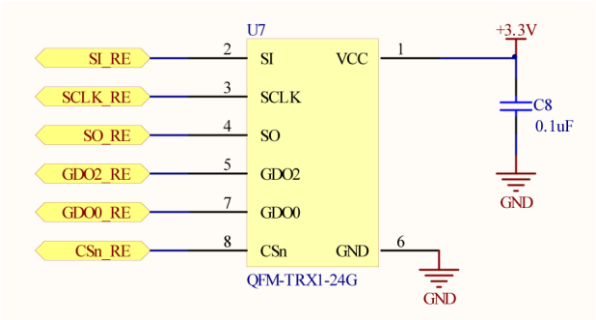


Figure 42: WAU Receiver: Transceiver circuit

This schematic is an exact copy of circuit from WAU transmitter.

PIC32

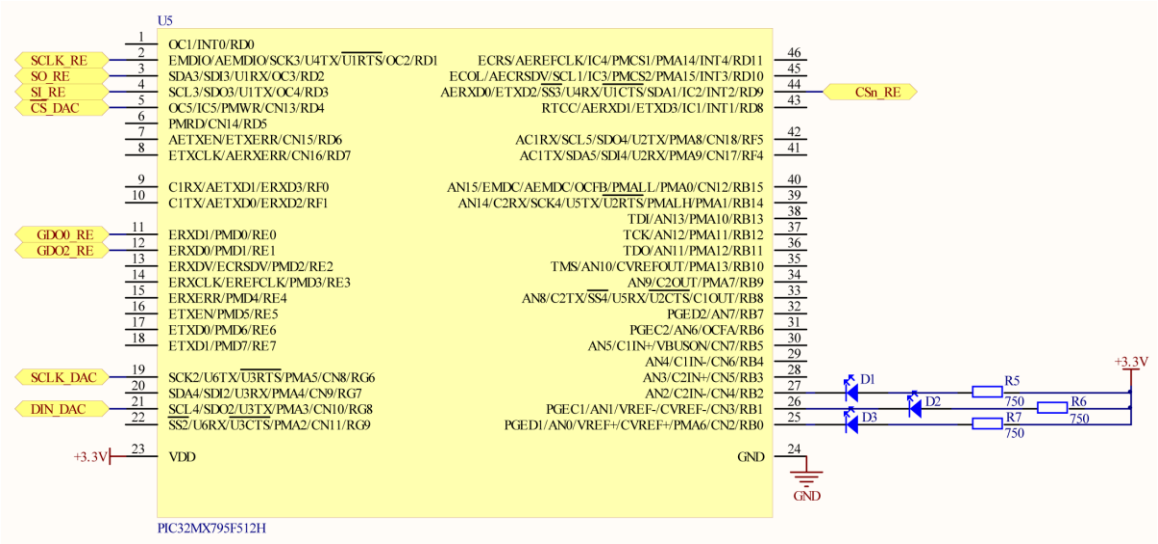


Figure 43: WAU Receiver: PIC32 Module Circuit

Transceiver is connected to SPI channel no. 3 and DAC to channel number 2. Like in previous circuit, three test diodes are connected to port B.

DAC

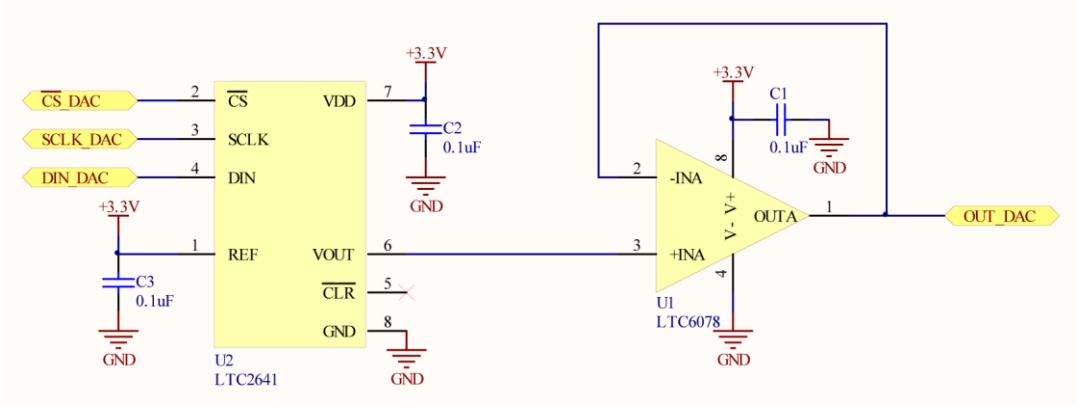


Figure 44: WAU Receiver: DAC circuit

The DAC circuit includes two external decoupling capacitors(C2,C3). The pin *REF* in DAC is connected to 3,3V, which means that this value is the reference voltage for this transducer. The signals *CS_DAC, SCLK_DAC* and *DIN_DAC* are the SPI interface for this chip. The analog output (VOUT) is directly connected to operational amplifier configured as voltage follower. Adding this element was dictated by the recommendation from datasheet of DAC. In circuit it was placed the same model of operational amplifier as in the suggested application circuit.

Output Analog Signal Conditioning

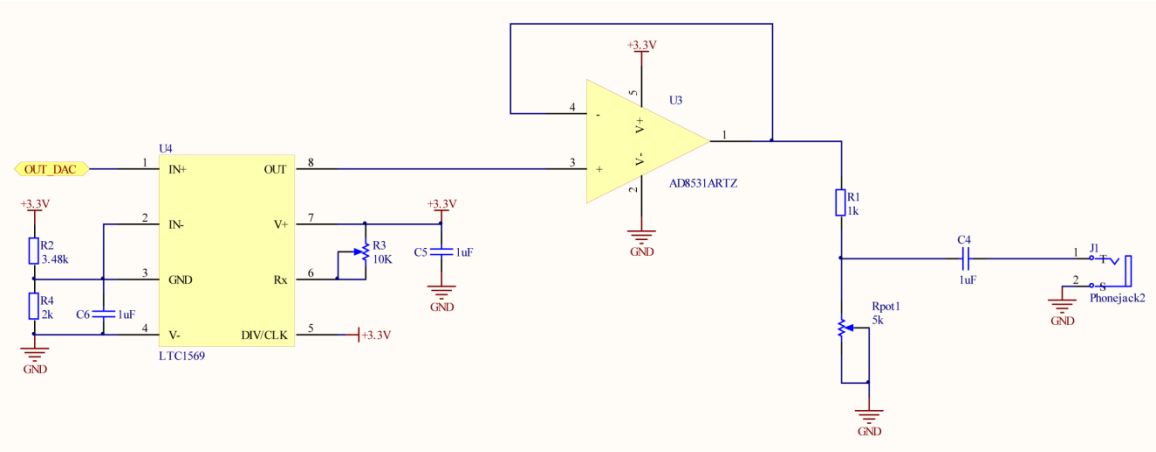


Figure 45: WAU Receiver: Output Analog Signal Conditioning circuit

This circuit contains a low pass filter LTC1569-7 configured identically as in WAU Transmitter part. The output of filter is feeding the voltage follower, which is placed here to counter the effects of loading of the source.

On the output of the operational amplifier there is a voltage divider composed from R3 and Rpot1. Its role is to restore the level of amplitude from the input of device. The DC offset added in Input Analog Conditioning, is removed when signal passes through the capacitor C4.

The output analog audio signal is connected to junction J1, which is a mono Jack socket.

Corrections in schematics

After assembling the device and start the tests, some mistakes were detected.

- The input capacitor C3 in WAU Transmitter was removed. With potentiometer RPot2, it was creating an undesirable low pass filter, which was blocking the frequencies below 400Hz.
- The value of resistor R3 in WAU Transmitter , was changed to 4,7kΩ. Only just with this resistance it was possible to obtain demanded 1,65V on the non-inverting input of operational amplifier U3.
- The both LTC1569-7 filters were supplied from voltage of 5V. This was caused by the fact, that the output voltage of this part turned out to be in the range of 50mV to (V_{CC} - 0,8V). This led to a situation in which ADC could not work at full resolution. The 5V voltage was taken from the AC adapter.
- The SDO and SCK pins in ADC had been switched. This mistake was caused by wrong assignment of these pins during creating the schematic model for ADC.

- The CLR pin in DAC was connected to V_{CC}. When this pin was floating, DAC had been working properly only for some time after turning on the device. Connecting CLR pin to 3,3V had eliminated this problem.
- GDO0_TR, GDO2_TR pins in WAU Transmitter and GDO0_RE, GDO2_RE pins in WAU Receiver were connected to RD10 and RD11 pins in PIC32 instead to RE0 and RE1. It was caused by the need of using external interrupts of PIC32 when handling transceiver. RD10 and RD11 are input pins for external interrupts INT3 and INT4. Also, it was necessary to connect pull-down resistors to those pins.

The corrected schematics are presented on Figure 46.

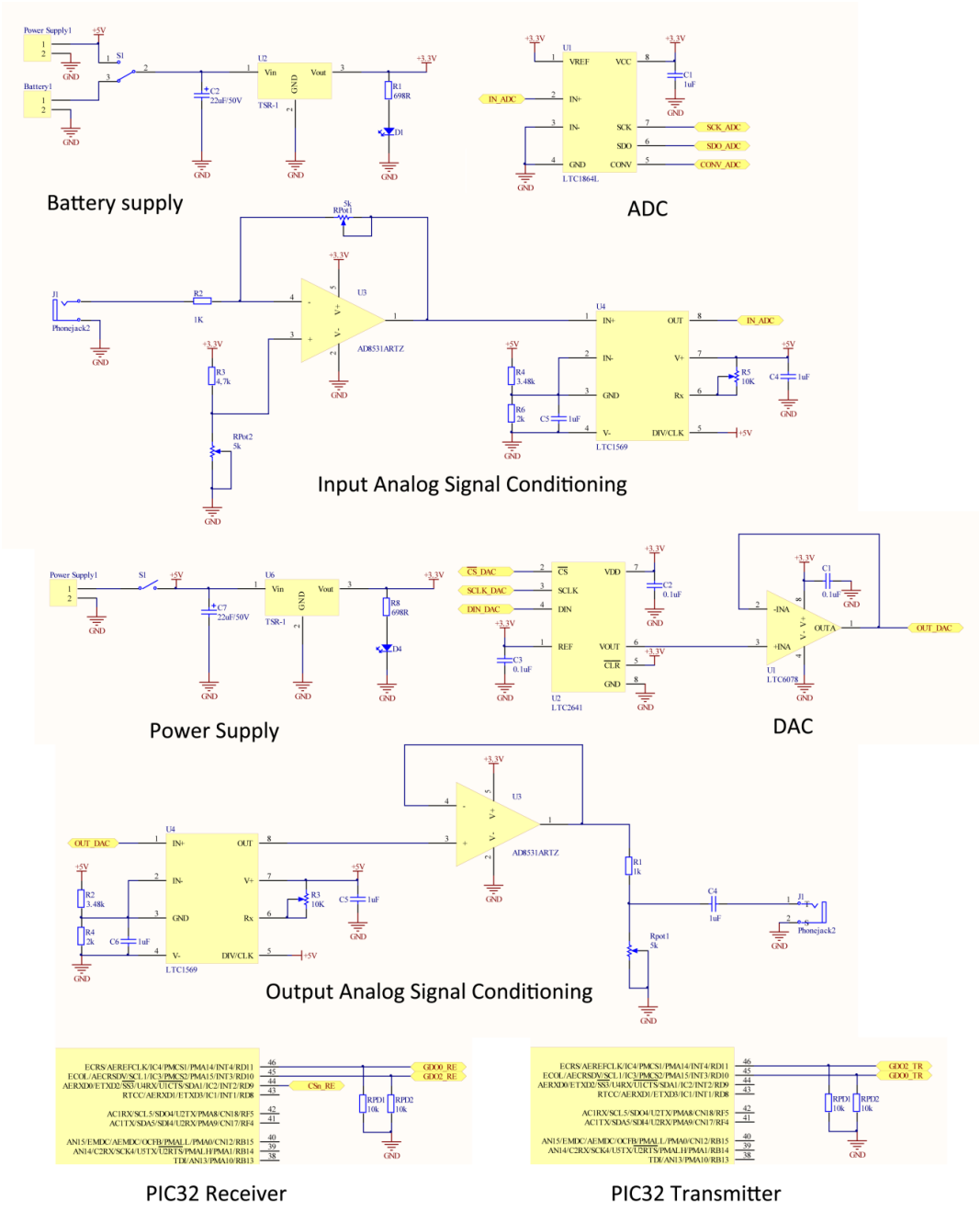


Figure 46: Corrected schematics of WAU

The corrected schematics can be found also on DVD added to dissertation.

3.2.4. PCB

The printouts of PCB can be found in appendix.

Following rules were applied in PCB design:

- Track width:
  - 15mil (0,508mm) for VDD, VSS, AVDD, AVSS tracks
  - 10mil (0,254mm) for other tracks
- Clearance constraint between tracks:
  - 15mil (0,508mm) for VDD, VSS, AVDD, AVSS tracks (in some places there are some exceptions and clearance is 10mil)
  - 10mil (0,254mm) for other tracks
- Size of vias : diameter 1mm, hole size 0,5mm

During the design, efforts were made to place capacitors as close as possible to integrated circuits. Also all tracks were routed with the minimum length wherever possible.

The boards are a double layer PCBs. On both layers, the polygon was poured and connected to ground net. In order to the ground potential was equal on both sides of the boards, some additional vias on polygon plane was placed. The ground tracks were not routed, since the polygon took care of connecting ground pins.

To minimize the size of the boards, most of the SMD elements were placed under the PIC Module. Trimmers were located in places, where they can be easily accessed.

To make possible mounting the boards in cases, the holes for screws was placed on each corner of the boards.

To connect external elements, such as power switches Jack and AC adapter sockets the trough hole screwed terminal blocks was used.

The dimensions of both WAU Receiver and WAU Transmitter are equal: 56 × 61 mm.

3.2.5. Final Assembly

The photos of produced PCBs are shown on Figure 47. As it can be seen the PIC Module is plugged into WAU motherboard.

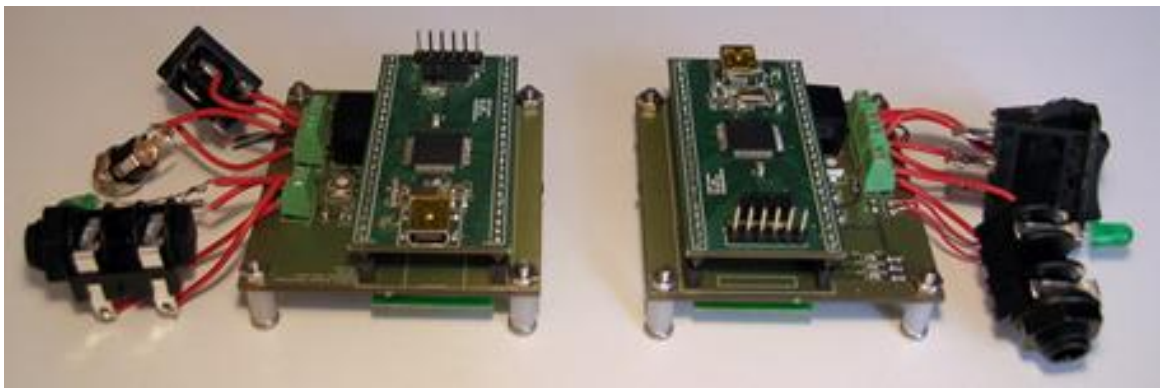


Figure 47: Produced PCBs of Wireless Audio Unit

Finally, the both WAU Transmitter and WAU Receiver were put in the enclosures. Both has the same dimensions: 110×82×44 mm.

Figure 48: Final assembly of Wireless Audio Unit

CHAPTER 4
COMPRESSION

Compression is the process whose objective is to reduce the size of a data set by writing the contained information in a different and more efficient way. The result of this transformation is a reduction of the number bits needed to store the information in the data set. This process is called *encoding*.

To reconstruct the information after encoding, the compression must be followed by a de-compression method which has to possess also a procedure known as *decoding.* This algorithm is responsible for restoring to the previous form the information of the original data set.

Compression techniques can be divided in two groups, when considering the loss of information during encoding:

- **Lossless** – the original dataset can be reconstructed from the encoded stream without any loss. These methods are used when the quality of information is essential, and also in some sort of data, where the loss of information makes impossible the decoding process. Lossless algorithms are usually used in text, computer files and high multimedia compression.
- **Lossy**–These techniques lose information after compressing the data set, thus in most cases they provide better results (in the final size of the compressed data) than the lossless methods. After decompression however, the final data is not identical to original. Lossy encoders are used especially in compression of music, video and images. In these cases, if the loss of data is small, the user cannot notice the difference.

There are many known methods of lossless data compression. They have different means of operation, and are dedicated to different types of data, but they all share one

in common characteristic - compression is performed by removing redundancy from original data. If a data set is generated from a non-random source, it has some structure. Compression methods are using this structure to represent a data set in a reduced form, where this structure is not directly recognizable.

In text streams, for example, this structure can be connected with probability of occurring different characters in words, or words in sentences. In images redundancy is associated with fact that, in most cases, adjacent pixels have similar color.

In the cases where the encoder and decoder are using the same algorithm (in opposite direction), the method of compression is called symmetrical. In contrast, the asymmetrical methods are those in which either compressor or decompressor have a more or less complex operation. For example, asymmetrical methods can be useful when files are rarely compressed, but their decompression is very often, and the decompression should be fast.

If the encoder and decoder do not know the statistics of the input stream, the compression method is called universal. In most cases, however dedicated compressors provide better compression, as they are constructed to deal with particular types of data. The advantage of universal compressors is that they can be widely applied.

The performance of compression can be represented by different factors listed below[Salomon 07]:

- **Compression ratio** –defined as ratio of output size to input size:

$$Compression\ ratio = \frac{size\ of\ the\ output\ stream}{size\ of\ the\ input\ stream}$$

  From this equation it can be assumed that the smaller the ratio is, the better is the compression. For example, a value 0.7 means that the output stream occupies 70% of the input stream. For a ratio equal to 1, the encoder is not providing any compression.
  In some cases the compression ratio is used as a factor defined as *100 × Compression ratio*. The result here is given in percentage, which is more suitable for presenting the outcome of compression.
- **Compression factor** – it is the inversion of compression ratio:

$$Compression\ factor = \frac{size\ of\ the\ input\ stream}{size\ of\ the\ output\ stream}$$

  In this factor, the values above 1 mean the positive result of compression, and below means data expansion instead of compression. Sometimes, using this

parameter is more readable and natural, because a higher value is symbolizing a better compression.

- **The compression gain** – is defined as:

$$Compression\ gain = 100 \log_e \frac{reference\ size}{compressed\ size}$$

The reference size can be the size of the original stream, or the size of the stream compressed with some standard lossless compression method. The use of the logarithm functions allows compression gain comparison simply by subtraction. The unit of this factor is called *percent log ratio* and has symbol– "$\frac{\circ}{\circ}$" .

- **Cycles per byte (CPB)** - This parameter measures the speed of compression. The result is the number of machine code cycles that, in average, is needed to compress one byte.

In the remaining of this chapter, some examples of compression methods will be shown. They are divided in four groups: Basic techniques, Statistical methods, Dictionary methods and, finally, Audio compression methods.

4.1. Basic techniques

4.1.1. Run-Length Encoding (RLE)

The idea behind this compression method is the following: If a data item *d* occurs *n* consecutive times in the input stream, the encoder replaces it with a single pair *nd*. The *n* consecutive occurrences of a data item are called a run length of *n*. To better show how this method works, here is an example of text compression using RLE.

Applying this method to the string ”2.␣this␣week␣will␣be␣good” would yield ”2.␣this␣w2ek␣wi2l␣be␣g2od” but it would not work because the decoder will not know if the character “2” stands for the number of occurrences of the following character or if it a character belonging to the original message (moreover in this case RLE does not provide any compression). To solve this problem it can be used a special character “@” to mark the repetitions. But, for the given string in this example, would result in ”2.␣this␣w@2ek␣wi@2l␣be␣g@2od” which is longer than the original string, since two repeated letters are substituted with three characters. This method can be modified so that only three or more repetitions of the same character will be replaced with a repetition factor. A diagram below shows a block diagram of this algorithm.

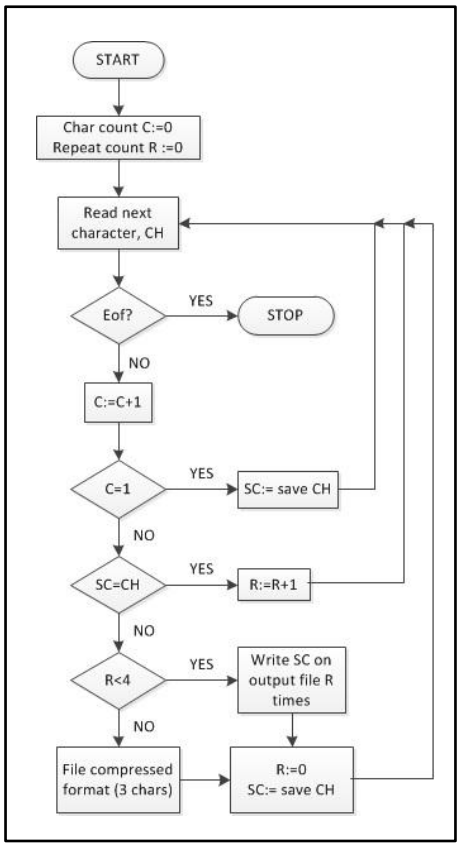


Figure 49: Block diagram of the RLE alghoritm.

According to the diagram of Figure 49, after reading the first character, the *repeat count* (R) is 1 and the character is saved in a temporary location. Consecutive characters are compared, with the first one already saved. If they are identical to it, R is incremented. When a different character is read, the following action depends of the current value of R. If it is smaller than 4, the saved character is written on the compressed file and the newly-read character is saved. Otherwise, an @ is written, followed by the repeat count and the saved character.

To evaluate how efficient RLE is, it can be assumed a string of N characters that needs to be compressed. The string has M repetitions of average length L. Each M repetition is substituted with 3 characters, so the size of compressed string is N – ML + 3M = N – M(L – 3). Therefore the compression factor is:

$$\frac{N}{N - M(L - 3)}$$

For example for N = 1000, M = 20, L = 5, the compression factor is 1.04. A better result is, for example, for N = 1000, M=50, L = 10 yielding a compression factor of 1.538.

RLE is used in image compression. For example in fax documents, where the dominating color is white, this compression is very efficient. This algorithm is also used as one of filters in PostScript and PDF documents.

In audio compression, RLE is applied for example in FLAC format (described further in this chapter), being effective in cases there the input audio signal keeps a constant value during some periods.

4.2. Statistical methods

To compress data, statistical methods use shorter codes for symbols (or groups of symbols), that occurs more often in input data, which means that they have higher probability of appearing. There are two aspects for constructing a statistical method of compressing:

- Creating a code that can be decoded unequivocally
- Creating a code with minimum average size

To go further it is necessary to define a factor called entropy. This factor measures the number of bits needed on average to represent single symbol from a stream of data. For a string of *n* symbols from a_1 to a_n entropy can be calculated from:

$$H = \sum_{i=1}^{n} P_i \log_2 \frac{1}{P_i} = -\sum_{i=1}^{n} P_i \log_2 P_i \,,$$

where P_i is the probability of occurrence of a symbol a_i. This equation shows that data entropy is connected with the probability of single symbols. This factor is largest when all P_i are equal. Concerning this fact, it was created other factor, called redundancy of data (R). It is the difference between the maximum possible entropy and the real entropy of a data stream, and is given by:

$$R = \left(-\sum_{i=1}^{n} P \log_2 P\right) - \left(-\sum_{i=1}^{n} P_i \log_2 P_i\right) = \log_2 n + \sum_{i=1}^{n} P_i \log_2 P_i$$

The principal statement of the statistical method, proved by Claude Shannon[Salomon 07], is that information of *n* elements can be compressed to a maximum of *n*H bits.

4.2.1. Variable-size codes

The statistical methods to compress data are using codes of changing length. They are called variable-size codes. Applying this method of compression can be shown better with an example.

Let us consider a string containing of set of four elements, with probability of occurrence written in a Table 4:

Table 4: Set of example elements with their probability of occurrence

Symbol	Probability
a_1	0,51
a_2	0,25
a_3	0,20
a_4	0,04

Without any coding, the number of bits necessary to represent each symbol is 2 (00,01,10,11).

The entropy of this data is:

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i = -\left[a_1 \log_2(a_1) + a_2 \log_2(a_2) + a_3 \log_2(a_3) + a_4 \log_2(a_4)\right] =$$
$$= -[0{,}56 \log_2(0{,}56) + 0{,}25 \log_2(0{,}25) + 0{,}20 \log_2(0{,}20) + 0{,}04 \log_2(0{,}04)] = 1{,}65$$

This result means that the smallest possible number of bits needed to represent one symbol is 1.65 on average.

The redundancy of this data set is:

$$R = \log_2 n - H = \log_2 4 - 1{,}65 = 0{,}35$$

The method to assign codes for each symbol is that the most probable element should have the shortest representation. Following variable-size codes can be assigned to symbols a_1 to a_4.

Table 5: Assigned variable-size codes

Symbol	Probability	Code
a_1	0,51	1
a_2	0,25	01
a_3	0,20	000
a_4	0,04	001

The code assignment has to be chosen carefully so that it can be decoded unambiguously. The properly selected codes should have a feature called “prefix property”. The key of this ability is that, when a starting bit or pattern of starting bits in code is assigned to one symbol, it cannot be assigned to other symbol. This is essential for the decoder to work faster and decode the string of bits precisely.

In the example given above, the assigned code for a_1 is “1”, and none of the other code for other symbols is starting with “1”. The same situation is with a_2. The code for this symbol is starting with “01”, and this sequence does not occur at the beginning of any other code.

The method of variable size coding can be applied now to sequence of 20 symbols given below:

$$a_3a_1a_2a_2a_2a_3a_1a_2a_1a_1a_1a_2a_4a_1a_3a_1a_1a_2a_2a_1$$

When the string is not compressed, it has length of 20 x 2bit = 40 bits. While applying variable-size coding, the given sequence assumes the form:

000|1|01|01|01|000|1|01|1|1|1|01|001|1|000|1|1|01|01|1

The length of this string is 35 bits, so the gain is of 5 bits. Of course, this example is very trivial, and applied to very short string. The variable-size coding method is most effective for very long sequences of compressing data, counted in thousands of bits.

4.2.2. The Golomb-Rice Coding

The Golomb coding is dedicated to encode streams of integers. In these streams there should be an assumption stating that: if the integer value is large, its probability of occurrence is low. The Golomb coding uses the *unary code*. This code for a positive integer *n*, is *n* “ones” followed by one “zero”. For example, the unary code for 5 is 111110, and for 8 is 111111110.

Salomon Wolf Golomb described his method of coding in a paper - “Run length encoding”[Golomb 66] where he presented the context for a problem as it is quoted below:

> *"Secret Agent 00111 is back at the Casino again, playing a game of chance, while the fate of mankind hangs in the balance. Each game consists of a series of favorable events (probability p), terminated by the first occurrence of an unfavorable event (probability q = 1 - p). More specifically, the game is roulette, and the unfavorable event is the occurrence of 0, which has a probability of q = 1 / 37. No one seriously doubts that 00111 will come through again, but the Secret Service is quite concerned about communicating the blow-by-blow description to Whitehall.*
> *The bartender, who is a free-lance agent, has a binary channel available, but he charges a stiff fee for each bit sent. The problem perplexing the Service is how to encode the vicissitudes of the wheel so as to place the least strain on the Royal Exchequer..."*
>
> S.W. Golomb. Run-length encodings. *IEEE*, July 1966.

The Golomb code is parameterized by a factor *m*>0. The positive number *n* can be represented in Golomb code of parameter *m,* using two quantities *q* and *r*, where:

$$q = \left\lfloor \frac{n}{m} \right\rfloor$$

and

$$r = n - qm$$

The number *q* is therefore a quotient and *r* the remainder. The quotient can take values 0 ,1 ,2 ,… and the remainder 0, 1 ,2 ,… ,*m*- 1. It is useful also to define a parameter *c*, which takes value:

$$c = \lceil \log_2 m \rceil$$

The $\lfloor x \rfloor$ operator rounds x to the integer that immediately precedes it (for instance, ⌊5,2⌋=5 and ⌊5,7⌋=5), and $\lceil x \rceil$ rounds *x* to the integer value that immediately succeeds it (for example, ⌈5,2⌉ = 6 and ⌈5,7⌉ = 6).

The Golomb code consists of two parts. The first is the value of *q* noted in unary code, and the second is the binary representation of *r* coded in a special way. The first $2^c - m$ values of *r*, are noted in (*c – 1*) bits and the rest of *r*'s are coded in *c* bits each. When the *m* is the power of 2, the coding is easier, because there is no *c – 1* bit codes ($2^c - m = 0$). This case of Golomb code was developed by Robert Rice and it is called Rice coding.

For example, Golomb code for m = 3 can be computed in following steps:

First, computation of c:

$$c = \lceil \log_2 m \rceil = \lceil \log_2 3 \rceil = \lceil 1{,}59 \rceil = 2$$

The *r* can take the values 0, 1, or 2. The amount of first *r* values coded using *c – 1* bits is:

$$2^c - m = 2^2 - 3 = 1$$

So for *r* equal to 0, it will be coded using c – 1 = 1 bits. For *r* = 1 and *r* = 2, it will be coded using *c* = 2 bits.

For *m* = 8 the value of *c* = 3, and *r* can take values from 0 to 7. Because 8 is the power of 2 it is, in fact, Rice coding and all values of *r* will be coded using *c* = 3 bits.

Following is the table of Golomb code words, for *m*=3 and *m* = 8:

	n	q	r	Codeword
0	0000	0	0	0|0
1	0001	0	1	0|10
2	0010	0	2	0|11
3	0011	1	0	10|0
4	0100	1	1	10|10
5	0101	1	2	10|11
6	0110	2	0	110|0
7	0111	2	1	110|10
8	1000	2	2	110|11
9	1001	3	0	1110|0
10	1010	3	1	1110|10
11	1011	3	2	1110|11
12	1100	4	0	11110|0

	n	q	r	Codeword
0	0000	0	0	0|000
1	0001	0	1	0|001
2	0010	0	2	0|010
3	0011	0	3	0|011
4	0100	0	4	0|100
5	0101	0	5	0|101
6	0110	0	6	0|110
7	0111	0	7	0|111
8	1000	1	0	10|000
9	1001	1	1	10|001
10	1010	1	2	10|010
11	1011	1	3	10|011
12	1100	1	4	10|100

As it can be seen, the number of bits needed to note the number in Golomb code is in some cases larger than number of bits in its binary representation. That is because Golomb code is not intended to compress the size of single numbers. Its purpose is to reduce the average length of a string of numbers, if the values in this string are from large range and the probability of occurrence the smallest numbers from this range is high. In other words, Golomb coding is dedicated to be used on strings of numbers which have a *geometric distribution*.

The Golomb coding (and especially Rice coding) are used often as a last stage of compression in lossless audio encoders.

4.3. Dictionary methods

As distinct from statistical methods, dictionary methods do not use any statistical model, or variable size codes. In exchange, they choose a string of symbols and encode it as a “token” using a dictionary which contains strings of symbols and the associated tokens. A dictionary can be static or dynamic. The second one can adapt to input data by adding new strings of symbols to the previous set and associate it with new tokens.

Compressors based on dictionary methods are entropy encoders. It means that a string of *n* elements can be at maximum compressed to *nH* bits, where *H* is the entropy of the string. These compressors have the best compression factors on vary large input files, and they are quite popular due to the good results in practical applications.

The simplest example of a static dictionary can be the dictionary of any language. When an input string of symbols (in this case a word) is read from the input data, the compressor searches for it in the dictionary. If this word is found, it is replaced with and

index to the dictionary, corresponding to the selected word. If this input string is not found, it is send to the output without compression.

The compressor with this static dictionary will only provide a good compression factor when working with an input data which is a text in the particular language (or, at least, most of the words are). Otherwise the output file would have the same, or even a larger (for example, due to flag bits), size. Therefore in general the compressor using an adaptive dictionary is more desirable. Such method starts with an empty dictionary, or filled with some basic symbols. During processing, new words are added to it, which are found in the input string. Also, it is desirable that processing method can delete old words from the dictionary, because a too large dictionary slows down the searching process.

The processing loop in this compressor consists of the following steps:

1) Reading the input string and parsing it into words.
2) Searching for the words in the dictionary:
    a. If it is found – write the respective token to the output stream
    b. If it is not found – write the uncompressed word to the output, and add it to the dictionary.
3) Checking if an old word should be deleted from a dictionary.

The described method has two main features. The first is that, except for numerical computations, it uses searching and matching procedures. The other one is that the decoder is not symmetric, as it is in statistical methods, but asymmetric. In an adaptive dictionary-based method, the decoder has to read the input stream and decide if the current element is a token, or an uncompressed word. The advantage of this decoder is that it does not have to parse the input data as in statistical methods.

4.3.1. LZ77

The full name of this algorithm is Lempel-Ziv 77, due to its creators – Abraham Lempel and Jacob Ziv. The number 77 stands for the year when it was developed(1977).

The idea of this method is to use a portion of previously encoded stream as a dictionary. The encoder creates a window and shifts it from right to left over the stream which is being encoded. It is called a *sliding window*. This window is divided in two parts: a *search buffer* and *a look-ahead buffer*. The search buffer contains a piece of recently encoded string, and it is treated like a dictionary of this method. The look-ahead buffer is the next portion of the stream to be encoded. An example of such sliding window is presented below.

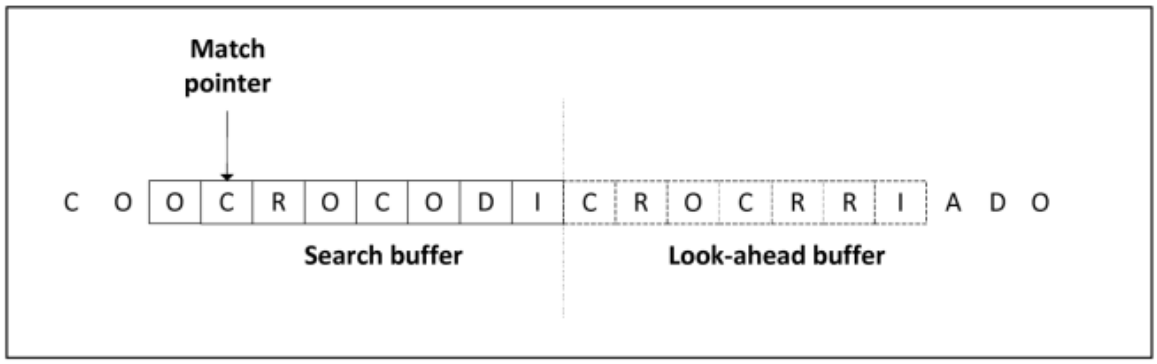


Figure 50: Example of sliding window in LZ77 method.

To encode the fragment of the stream in the look-ahead buffer, the encoder moves the match pointer back through the search buffer, until it matches the first symbol in look-ahead buffer. Than encoder analyses the consecutive symbols, to check if they match the next symbols in the look-ahead buffer. The number of equal consecutive symbols in both buffers is called the length of match. The distance between the match pointer and the look-ahead buffer is called offset. To be most efficient, the encoder is looking for the longest possible match. When it is found, the compressor encodes it with a triple *<o, l, c >*, where *o* is the offset, *l* is length of the match and *c* is a code word corresponding to the symbol in the look-ahead buffer that is following the match. In the example of Figure 50, the pointer is indicating the beginning of the longest match. The offset is 7, the length of match is 4, and the code word "r". The encoder is, therefore, substituting the fragment of the stream by the triple *< 7, 4, r>.*

If the encoder does not find any match in the search buffer, it sets the value of the offset and length to 0. The code word in this case is simply the symbol from the look-ahead buffer.

The decoding algorithm is based on restoring original string of symbols from encoded triples. If the triple from previous example *< 7, 4, r>* is encountered, the decoder moves back 7 characters and starts copying 4 symbols. Then it adds the code word, which in this case is the letter *"*r".

The LZ77 algorithm is an example of universal compressing method. It does not requires any additional information from that contained in the input stream. The LZ77 method has many variations like LZSS, LZX, LZ78, LZFG, LZRW and many others. Lempel and Ziv, therefore, significantly contributed to the development of dictionary compression methods.

4.4. Audio compression

The audio compression methods, as general compression algorithms, are usually divided in two groups – lossless and lossy. The first ones, as it was described in the introduction, compress the audio file without any loss of data. Lossy encoders are skipping some

information in audio samples to significantly reduce the data size. The size of audio files compressed by lossy methods is usually several times smaller than those compressed by lossless methods. In Table 1.3 is a comparison of a particular music track compressed using lossless encoder – FLAC, and lossy MP3 (MPEG-1/2 Audio Layer-3). The duration of this track is 9m46s(586s).

Table 6: Comparison of compressed track Miles Davis – "Freddie Freeloader".

		FLAC	MP3
Size	MB	64,1	17,8
	kb	525107,2	145817,6
Bit rate	kbps	896,09	248,84

As it can be seen, the size of the MP3 file is about 3,6 times smaller than the FLAC file. The cost of this, however, is a reduced quality of sound.

The lossy audio encoders are based on *psychoacoustics,* which means that they are involving a model of human auditory perception system. The idea of this model is that some sounds are not possible to hear by humans, which is closely connected with frequencies.

The first aspect of *psychoacoustics* is that some sounds which are equally loud cannot be perceived at different frequencies. This dependence is presented usually on a graph, where a sound pressure level (SPL) is plotted in a function of frequency.

The sound pressure level is a quantity, in logarithmic scale, describing the proportion of effective sound pressure of a sound when compared to a reference value:

$$L_P = 20 \log_{10} \left(\frac{p_{rms}}{p_{ref}} \right) [dB].$$

Figure 51 shows the audible region of human auditory system[Sayood 06].

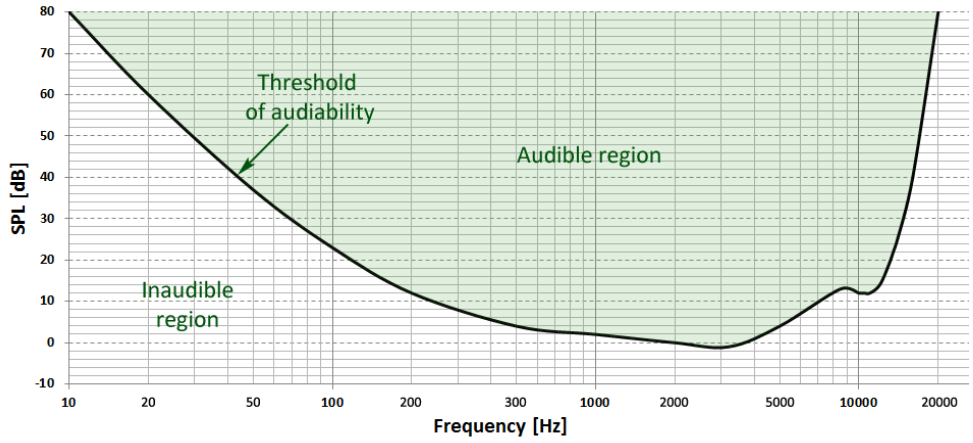


Figure 51: Audible and inaudible regions of human auditory system.

As it can be seen, a low amplitude sound for the frequencies on the order of 1-2kHz can be heard by the human ear, while for the same amplitude level but for low frequencies it cannot be heard. The lossy audio encoders use this characteristic to remove the parts of the recorded sounds which are not noticeable to listeners.

Lossy methods also makes use of the *spectral masking*. The idea of this process is that the occurrence of a tone at certain frequency will raise the threshold of audibility in some band around this frequency, in detriment of other regions of the spectrum.

Apart from lossy encoders there are also lossless audio compressors. They are not using any *psychoacoustic* model, but only algorithms which are compressing data without any loss in it, even when some parts of it are not hearable.

To compress audio files the lossless audio algorithms are using prediction methods and variable-size coding such as Golomb-Rice.

The result of lossless encoding can change significantly, depending on how much redundancy exists in the waveform under analysis. For example, the stream composed of only zeros can be compress to the single number representing the amount of samples, while a signal containing white noise would be hard to compress due to lack of redundancy.

Universal lossless compressors which are based on the LZ77 method (and its variations) are not very efficient in compressing audio files. That is because the dictionary methods do not match the statistical characteristics of audio waves. The Ziv-Lempel methods exploit the repeating sequences which occur in input data. Though the audio wave is, in most cases, somehow periodic, the samples do not repeat exactly. This property imposes a difference from audio compression and compressing text streams or program source code, which are more structured (thus, with a higher redundancy level).

As it was written, audio encoders usually use prediction algorithms to compress the data. These algorithms use the existing correlation between samples and, during encoding, predict the next samples from their predecessors. A general model of prediction is shown in Figure 52.

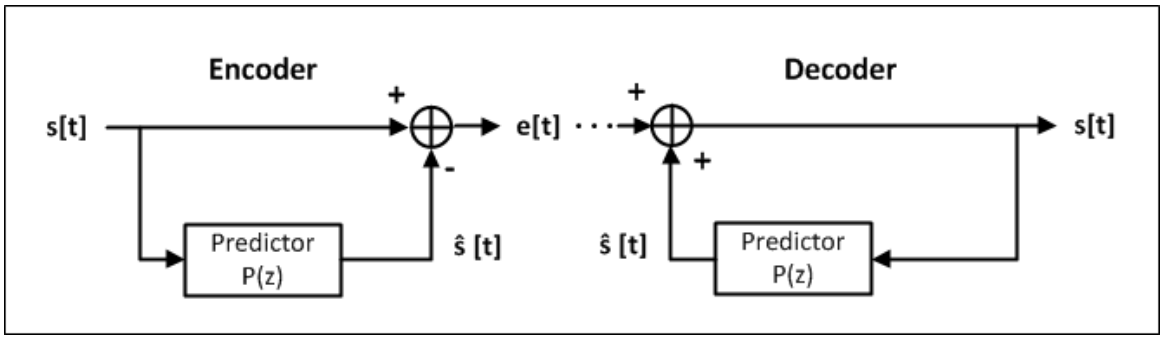


Figure 52: General prediction structure.

The input samples *s[t]* are the input for predictor *P(z).* It is creating the signal estimate *ŝ[t],* which is further subtracted from original signal, producing an error signal *e[t].* As the error signal contains less information that the samples, it can be coded with less bits than the original samples, thus creating compression. It is the error signal that is transmitted to the decoder.

The decoder is symmetrical to the encoder. The values *e[n]* are added to the values of *ŝ[n]* produced using the decoder predictor. The final signal after decompression is identical to original.

4.4.1. Shorten

Shorten is a simple lossless compressor for waveform files. It was developed by Tony Robinson [Robinson 94] who has dedicated it especially for speech compression. Nevertheless, this method can be used to any file whose data items (samples) are formed like in a wave signal. Shorten does the best compression for files with low-amplitude and low-frequency samples.

This method encodes the file by initially partitioning it into blocks. The size of the blocks is typically 128 to 256 samples. In audio files which consist of more than one channel, Shorten separates each channel in each block. Blocks are necessary for applying prediction algorithms, which cannot operate on single samples. Shorten has also a lossy mode, in which the samples are quantized before compressing.

After making a block, the samples in each of them are predicted, and the differences are computed. A predicted value *ŝ(t)* for the current sample *s(t)* is computed from the *p* preceding samples (depending of the order of prediction). A possible implementation of the predictor algorithm is:

$$\hat{s}(t) = \sum_{i=1}^{p} a_i s(t-i)$$

The difference between estimated and actual sample, which is also called the error, is computed as follows:

$$e(t) = s(t) - \hat{s}(t)$$

If the compression is properly performed, the error will have a small (positive or negative) value. Prediction can be achieved in different ways, involving different numbers of preceding samples. In other words, the order of prediction can vary.

For the zero-order prediction each sample *s(t)* is predicted as zero. A first-order prediction predicts each sample *s(t)* only from its predecessor *s(t-1)* (see Figure 53a). Consistently, the second-order prediction computes *s(t)* from a linear combination of the two previous samples *s(t-1)* and *s(t-2)* (see Figure 53b). Extending this idea, the

third-order prediction is a computation of the 2nd degree polynomial from 3 preceding samples (*s(t-3), s(t-2)* and *s(t-1)*) and extrapolated to *s(t)* (see Figure 53c).

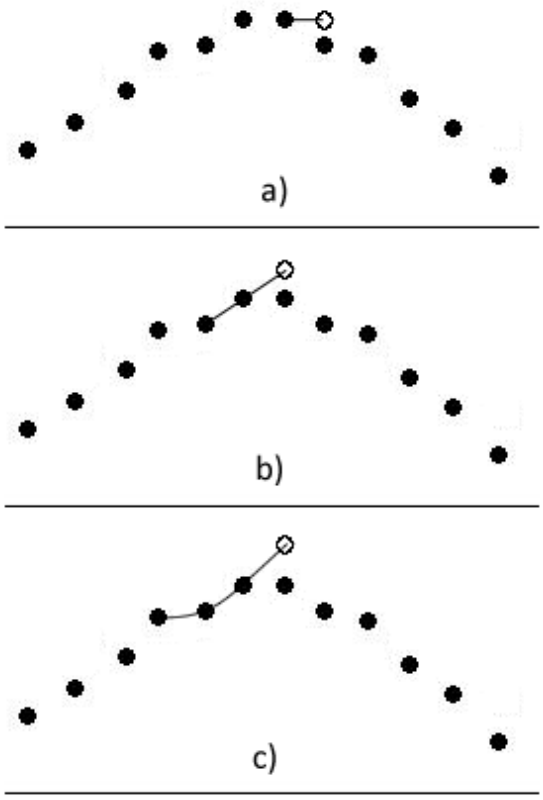


Figure 53: Predictors of orders 1, 2 and 3

The method of computing the second-order predictor is as follows. From the two given points: $P_2 = (t\text{-}2, s_2)$ and $P_1 = (t\text{-}1, s_1)$, it is possible to write the parametric equation of the straight segment connecting them:

$$L(u) = (1-u)P_2 + uP_1 = (1-u)(t-2, s_2) + \mathrm{u}(t-1, s_1) =$$
$$= (u + t - 2, (1-u)s_2 + us_1), \qquad u \in [0;1]$$

P_2 is the point from which the line starts, so $L(0) = P_2$ and $L(1) = P_1$. Extrapolation to the next point gives $L(2) = (t, 2s_1\text{-}s_2)$.According to this, the second-order predictor predicts the sample *s(t)* from the linear combination $2s(t\text{-}1) - s(t\text{-}2)$.

For the third-order there are three points $P_3 = (t\text{-}3, s_3)$, $P_2 = (t\text{-}2, s_2)$ and $P_1 = (t\text{-}1, s_1)$. The degree-2 polynomial that passes through those points is given by the uniform quadratic Lagrange interpolation polynomial:

$$L(u) = [u^2, u, 1]\begin{bmatrix} \frac{1}{2} & -1 & \frac{1}{2} \\ -\frac{3}{2} & 2 & -\frac{1}{2} \\ 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} P_3 \\ P_2 \\ P_1 \end{bmatrix} =$$

$$= \left(\frac{u^2}{2} - \frac{3u}{2} + 1\right) P_3 + (-u^2 + 2u) P_2 + (\frac{u^2}{2} - \frac{u}{2}) P_1$$

To verify this equation, it can be checked that: *L(0) = P3, L(1) = P2,* and *L(2) = P1.* Extrapolating equation to the next point gives $L(3) = 3P_1 - 3P_2 + P_3$. Therefore the prediction of the next sample is $\hat{s}(t) = 3s(t-1) - 3s(t-2) + s(t-3)$.

The predictors of first four orders are:

$$0^{th} \rightarrow \hat{s}_0(t) = 0,$$
$$1^{st} \rightarrow \hat{s}_1(t) = s(t-1),$$
$$2^{nd} \rightarrow \hat{s}_2(t) = 2s(t-1) + s(t-2),$$
$$3^{rd} \rightarrow \hat{s}_3(t) = 3s(t-1) - 3s(t-2) + s(t-3).$$

From these predictors the respective error values can be computed as:

$$0^{th} \rightarrow e_0(t) = s(t) - \hat{s}_0(t) = s(t),$$
$$1^{st} \rightarrow e_1(t) = s(t) - \hat{s}_1(t) = s(t) - s(t-1) = e_0(t) - e_0(t-1),$$
$$2^{nd} \rightarrow e_2(t) = s(t) - \hat{s}_2(t) = s(t) - 2s(t-1) + s(t-2) = e_1(t) - e_1(t-1),$$
$$3^{rd} \rightarrow e_3(t) = s(t) - \hat{s}_3(t) = s(t) - 3s(t-1) - 3s(t-2) + s(t-3) =$$
$$= e_2(t) - e_2(t-1).$$

As seen above the computation is recursive. For maximum compression it is possible to count all predictors and errors and choose the smallest one. The Shorten, in default mode, uses the second-order (linear) prediction.

4.4.2. FLAC

The FLAC acronym stands for Free Lossless Audio Codec. According to the respective license, the specification of format can be used by anyone without any previous permission.

The FLAC format evolves in some way from the Shorten method, following the idea of Josh Coalson [sourceforge.flac 06]. Similarly to many other audio codecs, it is composed of the following stages:

- Blocking – The input signal is divided in adjacent blocks. The size of the blocks can vary and depends on different factors, like sample rate, spectral characteristics over time, etc. Default FLAC uses fixed block sizes, but it provides also possibility to vary the block size within a stream.
- Interchannel decorrelation – When the audio stream has two channels, the encoder will create two signals: *mid = (left + right)/2* and *side = left – right*. Then it chooses the best (smaller in terms of number of bits) result and sends it to the next stage.
- Prediction – FLAC uses four types of prediction, which will be described later.

- Residual Coding –If the prediction is effective, this method ensures that the residual signal will require fewer bits per sample than the original signal. The method of residual coding used in FLAC is Rice Coding.

As mentioned earlier, FLAC uses four prediction methods, but only two are general for providing compression (the last two in the following list):

- Verbatim –This is the zero-order prediction considered in the Shorten method. It always predicts the next sample as equal to zero, so the difference is always equal to the original sample. The Verbatim predictor should provide almost the same compression as the others, when the input data follows a random pattern.
- Constant – FLAC uses this method when an input block is pure DC (for example, in the case of silent parts of an audio stream). In this case, the signal is run-length encoded (RLE). The result of this method contains information about the sample value and the number of repetitions.
- Fixed linear predictor–This is a predictor that fits a polynomial to the audio samples. The method is the same as presented in the Shorten method, but also includes a fourth-order predictor which is an extension of third-order predictor, involving one more previous sample. Skipping the mathematical transformations, the predicted sample is calculated by:

$$\hat{s}_4(t) = 4s(t-1) - 6s(t-2) + 4s(t-3) - s(t-4).$$

The error value sent further is:

$$e_4(t) = s(t) - \hat{s}_4(t) = s(t) - 4s(t-1) - 6s(t-2) + 4s(t-3) - s(t-4).$$

- FIR linear prediction –This is more complex method, supporting 32nd order FIR linear prediction. The algorithm consists on Linear Predictive Coding (LPC) where the maximum depth of the FIR filter is chosen from 1 to 32. Generally, the larger the depth of the filter is, more accurate is the prediction. The cost of this process is longer computational time. The LPC method is described below.

LPC

LPC is an algorithm that can predict an element a_i from a set of correlated values $\{a_i\}$, using n of its predecessors from $a_{i-1}$to a_{i-n}. The aim of this method is to try some values for the coefficient c_j and choose the set that minimizes the difference:

$$d_i = \left(a_i - \sum_{j=1}^{n} c_j a_{i-j} \right)^2$$

To find the set of coefficients that minimizes this equation it is necessary to differentiate the expected value of d_i with respect to a coefficient c_p, and to set such derivative to zero. This must be considered for all possible values of p, from 1 to n. The result is:

$$-2\left[\left(a_i - \sum_{j=1}^{n} c_j a_{i-j} \right) a_{i-p}\right] = 0, \qquad for\ 1 \le p \le n,$$

or,

$$\sum_{j=1}^{n} c_j E[a_{i-j} a_{i-p}] = E[a_i a_{i-p}], \qquad for\ 1 \le p \le n,$$

where E[·] returns the expected value of its argument.

Before proceeding with these equations it is necessary to describe a parameter called autocorrelation (*R*). The Autocorrelation $R_V(d)$ of a random variable *V* is the correlation of *V* with a copy of itself, shifted by *d* positions. For example, from an array *a* of *n* elements two arrays *x* and *y* are constructed, each of *n-1* samples:

$$a = (a_1, a_2, \dots, a_n),$$
$$x = (a_1, a_2, \dots, a_{n-1}),$$
$$y = (a_2, a_3, \dots, a_n),$$

Next it is computed the Pearson correlation coefficient [Salomon 07]*R* between *x* and *y*. This is the autocorrelation $R_a(1)$ of array *a* with a shift of 1 position. Additionally, it should be noticed that autocorrelation can also be expressed as:

$$R_V(k) = E[V_i V_{i+k}]$$

Therefore, after replacing, in the above equations, the expectations with the autocorrelation coefficients, such equations can be written the form of a system of *n* linear equations with the *n* unknown coefficients c_j:

$$\begin{bmatrix} R(0) & R(1) & R(2) & \cdots & R(n-1) \\ R(1) & R(0) & R(1) & \cdots & R(n-2) \\ R(2) & R(1) & R(0) & \cdots & R(n-3) \\ \vdots & \vdots & \vdots & & \vdots \\ R(n-1) & R(n-2) & R(n-3) & \cdots & R(0) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ R(3) \\ \vdots \\ R(n) \end{bmatrix}$$

which, in a matrix form yields:

***RC** = P*.

It can be solved by Gaussian elimination or by inverting matrix **R**, but it will involve $O(n^3)$ operations.

FLAC uses a different, more efficient method called the Levinson-Durbin algorithm. It was first proposed by Norman Levinson in 1947 , improved by J. Durbin in 1960, and further improved by several researchers[Salomon 07]. In its present form it requires only $3n^2$ multiplications.

4.5. Implementation of algorithms

Before implementing one of the compression methods in firmware of microcontroller, first it was desirable to realize them in appropriate computer software. For this purpose, the MATLAB application was used. All code listing ("m-files") are available on DVD attached to dissertation.

Algorithms created in MATLAB was based on prediction method. Two of them was derived from Shorten: second and third order prediction. The other one, was based on fourth order prediction known from FLAC method.

To properly test each of algorithms, it was necessary to do following steps:

- Create a compression function. In this step, it was built a predictor of order dependent on the method. Based on it, was created an error stream, which was result of compression. It is worth noting that few (depending of predictor order) first samples had to be copied to compressed stream, to make possible the decompression procedure.
- Create a function which compares input and compressed streams in category of size. It was calculated average number of bits saved per sample. Basing on this it was possible to compute the compression ratio.
- Create the decompression function. This function is inversion of compression algorithm. Like it was written, first few samples from input stream (e.g. for second order predictor it was 3 samples) were necessary to rebuilt the compressed data.
- Create a function which evaluates if input and decompressed streams are identical. This function simply subtracted both streams sample by sample. Derisible result, was a string composed only of zeros.

The algorithms were tested with eight different input streams, created for this purpose. These streams were recorded sounds from an electric guitar. The resolution of samples were 16 bits and sampling rate was 40kHz. These "tracks" can be found also on DVD attached to this document.

The example results of algorithms operation applied to the same input stream (track1) are shown at Figures 54, 55 and 56.

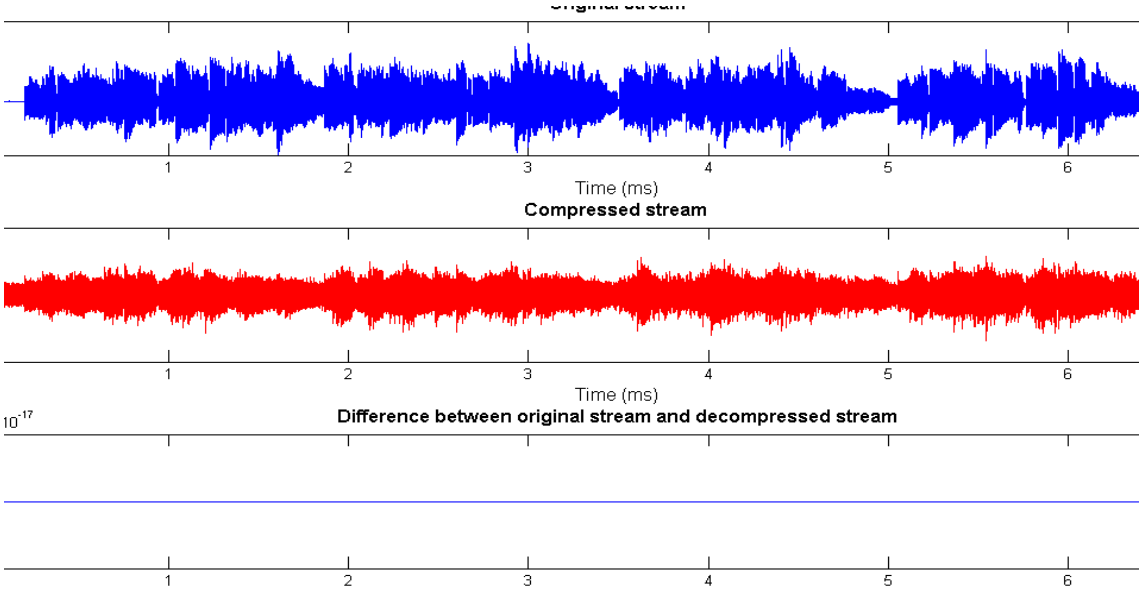


Figure 54: Result of the algorithm based on second order predictor (Shorten_2ord.m)

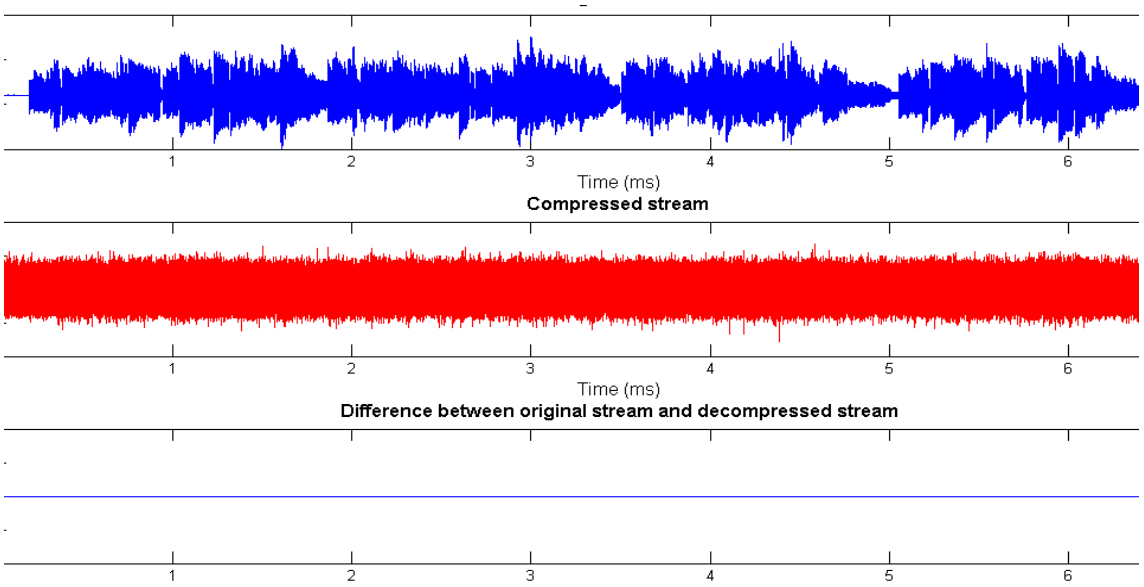


Figure 55: Result of the algorithm based on third order predictor (Shorten_3ord.m)

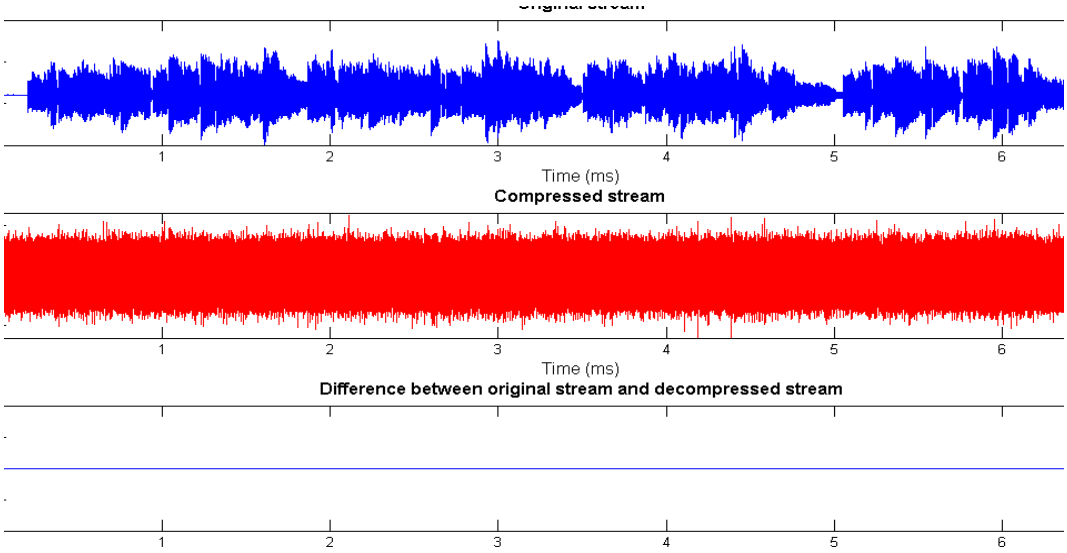
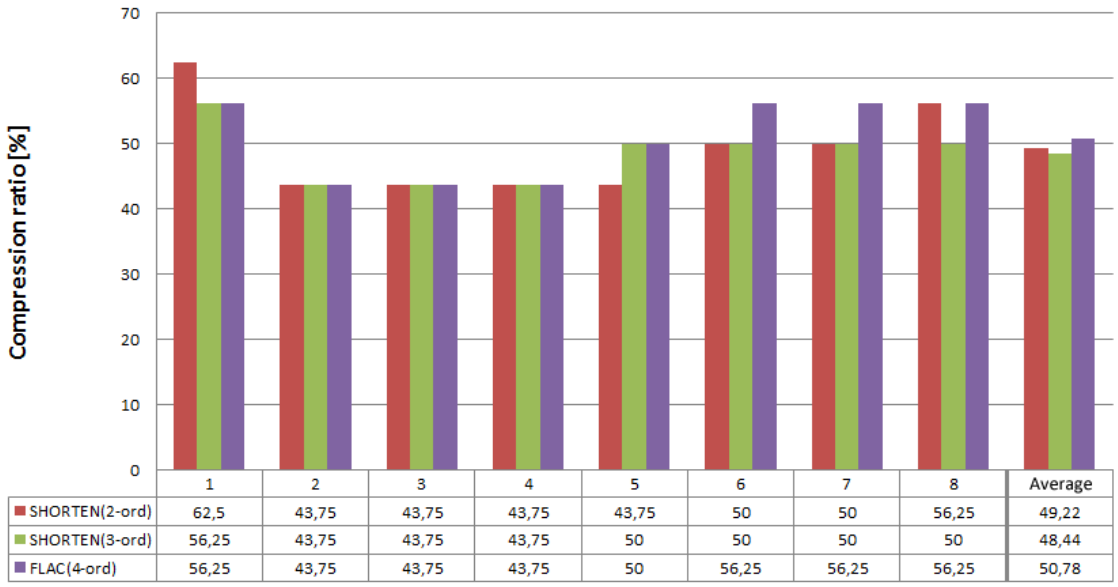


Figure 56: Result of the algorithm based on fourth order predictor (FLAC.m)

As can be seen, the average amplitude of compressed stream is two order of magnitude smaller than original. It can also be noted, that in each of algorithm, the decompressed stream is identical to input one.

All three algorithms were tested on all eight tracks. Every time, the compression ratio was calculated. The results were collected on the chart, that is presented on Figure 57. To remind, the lower value has the compression ratio, the compression method is more efficient (output stream is smaller).



	1	2	3	4	5	6	7	8	Average
SHORTEN(2-ord)	62,5	43,75	43,75	43,75	43,75	50	50	56,25	49,22
SHORTEN(3-ord)	56,25	43,75	43,75	43,75	50	50	50	50	48,44
FLAC(4-ord)	56,25	43,75	43,75	43,75	50	56,25	56,25	56,25	50,78

Figure 57: Comparison of tested compression algorithms

First observation can be that, all three algorithms have different results for different input streams. This is caused by the fact that each of the input files have different redundancy.

Comparing all the methods, it can be concluded that, results are very similar. The average compression ratio is about 50% for each method. Fourth order prediction, which intuitively should have the best compression ratio, proved to have the worse. Therefore, when the results are comparable, the most reasonable choice is to select second order prediction, because its implementation requires the least number of calculations.

CHAPTER 5
SOFTWARE DESIGN AND TESTING

The purpose of this chapter is to describe how was formed the software of Wireless Audio Unit. This includes the firmware of PIC32 in WAU Transmitter and WAU Receiver. Also some laboratory tests was done to check the operation of WAU.

At the moment in which this document is written, the author did not finished developing the software. It was caused by the difficulties in creating the firmware for transceiver module. This obstacle made impossible to finish the project in a assumed form. Description of the problem and attempted solutions will be presented in this chapter.

The listing of all described codes can be found on DVD attached this dissertation.

5.1. Components handling

Firstly will be explained the theory of operation of three main elements which has to be handled by software of microcontroller. These three components are ADC, DAC and Transceiver.

5.1.1. ADC

As has been written, LTC1864L analog-to-digital converter is interfacing with PIC32 via 3-wire protocol, which is compatible with SPI standard. These three signals are: SDO (serial data output), SCK (serial clock), CONV (equivalent to CS/SS signal). LTC1864L is configured as slave while PIC32 is master. This means that both clock and CONV signals are provided by microcontroller.

The operating sequence of ADC is shown on Figure 58 [LTC1864L.datasheet].

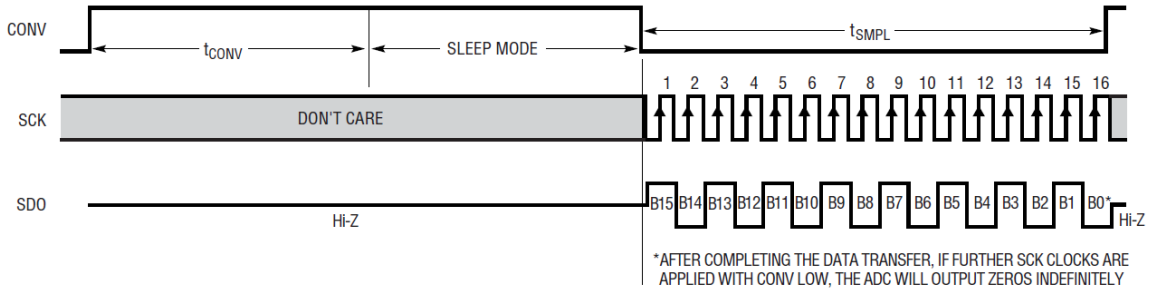


Figure 58: LTC1864L Operating sequence

The conversion starts with rising edge of CONV signal. It is finished after the time equal to t_{CONV}. LTC1864L datasheet states that duration of this time is typically 3,7µs. After this time, if CONV is still on high level, the ADC is going into sleep mode, drawing only leakage current (±1µA). During these two times, (t_{CONV} and t_{SLEEP}) even if master is providing clock, the ADC does not react to it. Also the SDO pin it in high impedance state (Hi-Z). On falling edge of CONV, LTC1864L is going into sample mode and SDO is enabled. SCK synchronizes the data transfer with each bit being transmitted from SDO on the falling SCK edge. The PIC32 therefore should capture the data on rising edge of SCK. The speed of receiving the data by microprocessor is determined by the frequency of clock signal. The maximum frequency that can be safely applied to LTC1864L is 8MHz. This means that the minimum sampling time can be calculated as:

$$t_{SMPL} = \frac{1}{f_{SCK}} \cdot 16 = \frac{1}{8MHz} \cdot 16 = 2\mu s$$

After analyzing the timing sequences, it can be concluded that, the sampling rate of ADC is determined by the frequency of *CONV*. To obtain assumed 40kHz, the period of this signal has to be set to 25µs (1/40kHz).

This was achieved with use of internal timer of PIC32. Timer 3 was configured to generate interrupt every 25µs. To do this, it was necessary to set the period of timer by putting into register PR3 a proper value. It was calculated from an equation presented below. F_{PB} is the clock frequency of PIC32 which is 80MHz and the prescale of timer 3 was set to 1.

$$PR3 = \frac{T_{T3} \cdot F_{PB}}{(T3\ clock\ prescale)} - 1 = \frac{25\mu s \cdot 80MHz}{1} - 1 = 1999$$

The handler of this interrupt is calling the function, which is setting the CONV signal to “0” and after sampling time to “1”.

5.1.2. DAC

Communication between LTC2641 and PIC32 is also performed using 3-wire interface compatible with SPI. These 3 wires are SCLK (serial clock), DIN (serial input) and CS (chip select). DAC is also like ADC configured as slave device to PIC32. The operating sequence is presented on Figure 59[LTC2641.datasheet].

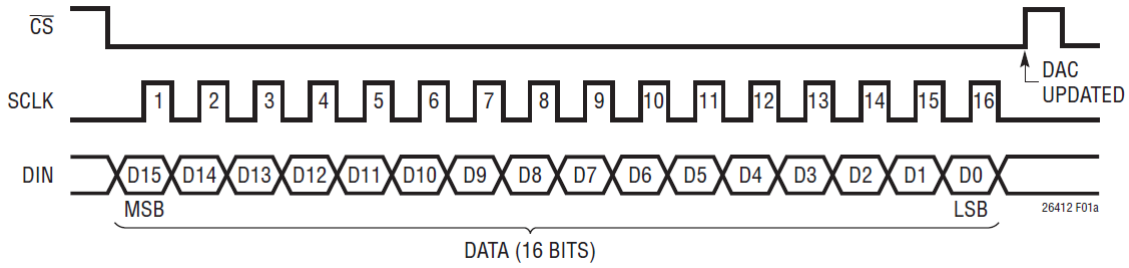


Figure 59: LTC2641 operating sequence

The falling edge of CS signal starts loading the serial data on DIN pin into shift register. The bits are shifted on rising edge of SCLK, MSB first. After loading 16 bits, a rising edge of CS signal transfers the data to the 16-bit DAC latch, updating the DAC output. To acquire correct data, exactly 16 periods of SCLK must occur. While CS pin remains high, the input shift register is disabled.

The maximum frequency of SCLK is 50MHz, which means loading of the data can last at minimum 320ns. Nevertheless the maximum frequency of SPI bus in PIC32 working at 80Mhz clock is around 26MHz so it this case the minimum time lengthens to 615ns.

To test the functionality of DAC, in PIC32 firmware, a dedicated function has been created. This function fills an array with values from one period of sinusoid. These values of 16 bit resolution are being successively sent to DAC in infinite loop. The test were performed on three sinusoid frequencies. The results are shown on Figure 60.

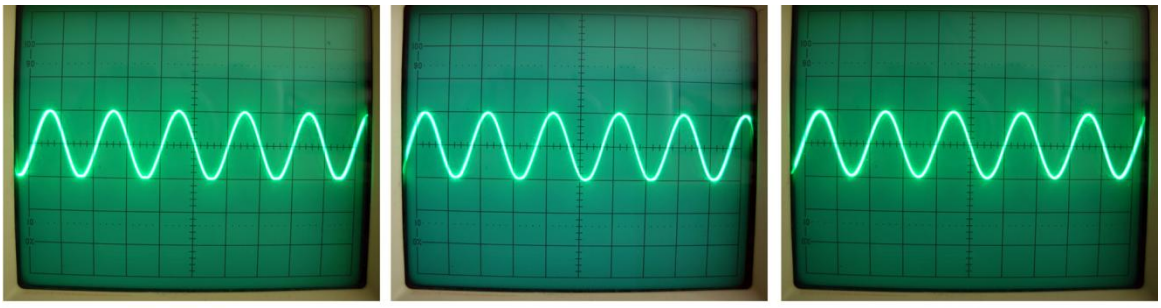


Figure 60: DAC testing; a) 100Hz (1V/DIV;5ms/DIV) b) 1kHz 100Hz (1V/DIV;0,5ms/DIV) c) 10kHz (1V/DIV;50µs/DIV)

Shown signals were captured at the output jack socket. After conversion in DAC, they have been filtered and the DC offset was removed. Also the amplitude of signal was

weakened from 3,3Vpp to 2Vpp with use of output voltage divider. Captured sinusoids can prove correct operation of DAC.

5.1.3. Transceiver

The RF Transceiver CC2500, mounted on module QFM-TRX1-24G is communicating via 4-wire SPI compatible interface (SI, SO, SCLK, CS). SPI is used to configure the radio and can be also used to read and write buffered data. All addressing and data transfer is done MSB first.

The microcontroller must be configured to operate in SPI master mode. The clock should be configured so the data is sent or received on rising edge(Figure 61 [CC2500.datasheet]). The SPI clock can run at maximum 9MHz for single access and 6,5MHz for burst access.

CC2500 has 47 configuration registers (address 0 to address 0x2E). They are responsible for the aspects like: type of modulation, throughput, form of packet etc. The values of these registers was fixed with help of SmartRF Studio software [SmartRF] provided by the manufacturer.

Before sending any data to CC2500 it is necessary to load the address of demanded register. The address header is shown on Figure 62. The R/W bit controls, if register should be written or read. The B bit determines if the access is single or burst.

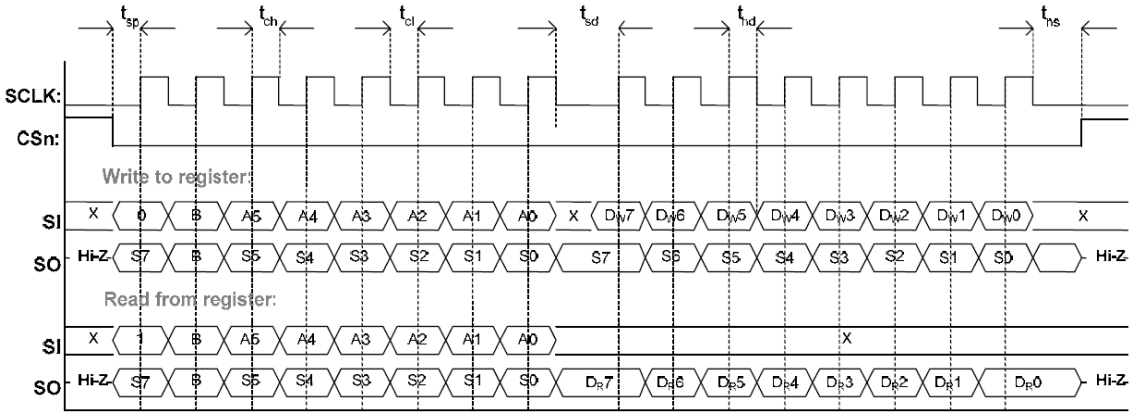


Figure 61: CC2500 SPI timing

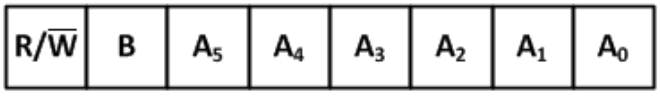


Figure 62: CC2500 adress header

The communication over SPI starts with falling edge of CS. It is necessary to wait for the MISO to go low before sending the address header.

For the single access to registers (bit B is cleared) it is possible to write or read one data byte depending on value of R/W bit. After the data byte, a new address is expected (CS can remain low). For the burst access, CC2500 expects one address header and then the consecutive data bytes, until terminating by setting CS high.

Another type of communication over SPI are *command strobes*. They are single byte instructions which starts an internal sequences like start TX, enter IDLE mode etc. Command strobes share the addresses (from 0x30 to 0x3F) with other type of registers which are called *status registers.* They belong to read only memory and can be accessed only in burst mode. The command strobes can be written or read only in single access mode. This distinction allows sharing the addresses.

When the header byte, data byte or command strobe is sent on the SPI interface, the chip status byte is sent from the radio on the MISO pin. The status byte contains key status signals, useful for the MCU like the current main state machine mode.

The other accessible directions are RX FIFO and TX FIFO. When using SPI mode to transmit data between two transceivers, in these two 64 byte registers data is loaded before sending (TX FIFO) or available to gather after receiving(RX FIFO). Both FIFO share the same address 0x3F. When the R/W bit is zero, the TX FIFO is accessed, and the RX FIFO is accessed when the R/W bit is one. The TX FIFO is write-only, while the RX FIFO is read-only. Both registers can be singe or burst accessed.

To summarize, Figure 63 gives a brief overview of different register access types possible.

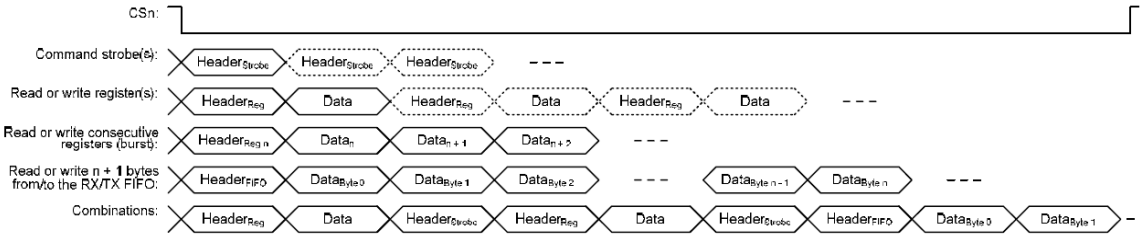


Figure 63: CC2500 Register access types

Besides 4-wire interface, also two additional pins GDO0 and GDO2 are connected to microcontroller (in CC2500 also exists GDO1 pin and is shared with SO pin). These are general control pins configured using proper registers. There are several different signals, that can be observed on these pins and can be useful for MCU which controls the transceiver. There are 64 possibilities to configure GDO pins. For example, GDO pins can help the microcontroller to handle the communication by indicating when packet is received or sent.

CC2500 has implemented packet handling support in hardware. Packet can be configured, to add following components to its content:

- A programmable number of preamble bytes
- A two/four byte synchronization word.
- A CRC checksum computed over the data field

It is not possible to only insert preamble or only insert a sync word. In the RX FIFO, it is optional to add two status bytes with RSSI value (Received Signal Strength Indicator), Link Quality Indication, and CRC status. The general packet format is shown on Figure 64.

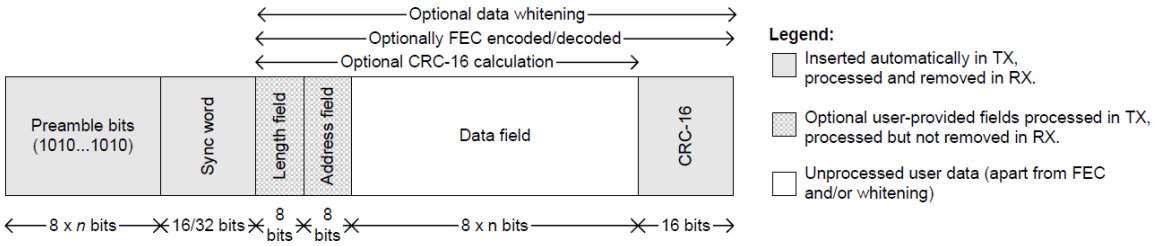


Figure 64: Packet format

It is possible to send packets of any length, but the code becomes more complex, if the packet size is longer than 64 bytes (the FIFO size). The packet size is understood as all bytes, that follow sync word except the optional CRC16 bytes.

There are three possible packet length modes:

- Fixed. In fixed mode, the preprogrammed value of appropriate register (PKTLEN) indicates the length of the packet.
- Variable. For variable size mode, PKTLEN register only tells the receiver what is maximum allowed length of packet. The size of packet is determined by the value of first byte after a sync word, which is called length field.
- Infinite. This mode is used to send packets of length greater than 255 bytes, which is maximum for fixed and variable modes.

Besides the possibility of using SPI protocol together with TX and RX FIFOs, in CC2500 also exists different type of communication called *serial synchronous mode*, which does not use this 4-wire interface[AN095]. In this mode data is transmitted or received serially over a two wire interface. This interface is provided by general control pins GDO0 and GDO2, where one of them carries the clock signal and the other data. In both RX and TX modes, the clock is provided by the CC2500 (Figure 65). The exchange of data is performed bit-by-bit. The clock is configured, that in RX mode data is set up on the falling edge of clock and in TX mode, data is sampled by CC2500 on the rising edge of the serial clock.

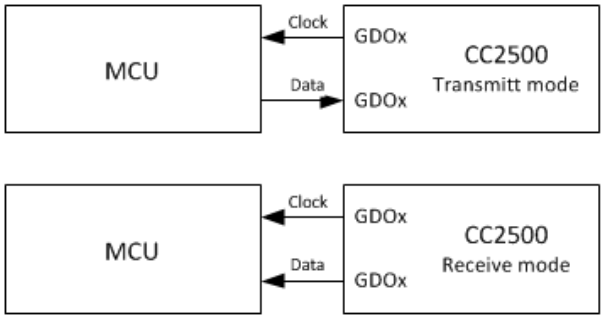


Figure 65: CC2500 Serial synchronous mode

Regardless of which method is used, before configuring transceiver and just after providing power supply, it is recommended to perform proper reset procedure. This procedure is illustrated on Figure 66[DN503] and has following steps:

- Strobe CS low / high.
- Hold CS high for at least 40 µs relative to pulling CS low
- Pull CSn low and wait for SO to go low.
- Issue the SRES strobe on the SI line.

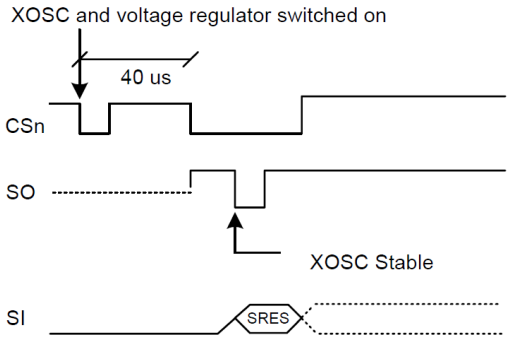


Figure 66: CC2500 Power-On-Reset with SRES

After this process the chip is in idle mode and can be further configured.

5.2. Software description

In this subsection will be described several attempts of creating properly working firmware for WAU, particularly the different types of transceiver handling.

However at the beginning, there will be brief description of originally assumed firmware algorithm for WAU. The flowchart of this algorithm is shown on Figure 67.

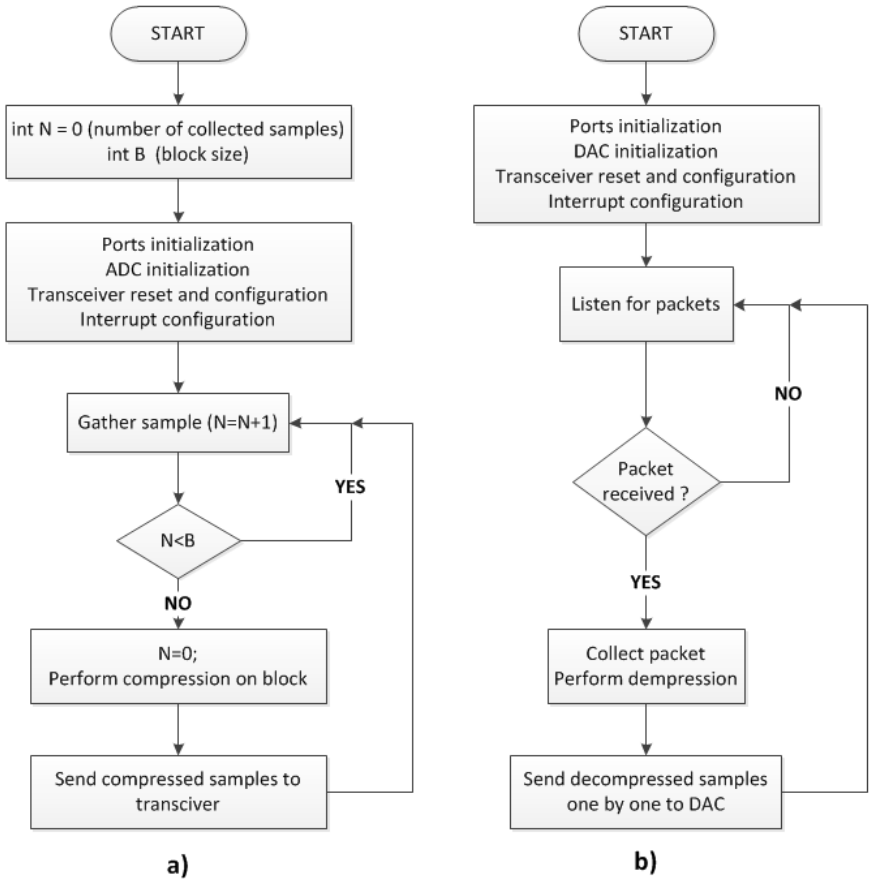


Figure 67: Flowchart of assumed firmware algorithms: a) WAU Transmitter b) WAU Receiver

In WAU Transmitter, first it is necessary to define a variable N, which is counting number of collected samples from ADC. Since the compression method that uses any prediction algorithm can be applied only to a set of data, it is necessary to define a variable B equal to number of samples on which compression is performed. Next step is the PIC32 port initialization. This includes determining which ports are input and which output. To output ports, initial values are assigned. Transceiver is reset and configured. The configuration registers are written with appropriate values as well as power table register (PATABLE), which corresponds to output power of radio. Further the interrupts are configured. One is them is the timer interrupt required for ADC operation. The others are connected with transceiver. Their number and purpose depends on mode in which transceiver is configured. After enabling interrupts, the samples from ADC are being gathered by MCU. When the number of collected samples reaches the size of block, the compression algorithm is applied on collected set of samples. After this, the compressed stream is sent to WAU Receiver with use of transceiver configured as transmitter.

In WAU Receiver the initialization includes ports, transceiver, DAC and interrupts (associated with transceiver operation). After setting the transceiver to operate as receiver in one of the modes, it is listening for incoming packets. After receive one, it

sends it to MCU, where decompression procedure is performed. Next, the recovered samples is being sent one by one to DAC.

Both algorithms (WAU Transmitter and WAU Receiver) should work in infinite loop to provide constant transmission of audio.

The first attempt to creating firmware was to develop a software, which transmits audio signal without providing any compression on it. For this purpose, the sampling rate is halved to 20kHz. The ADC and DAC was configured as it was described in section 5.1.

Four modes of communication between PIC32 and CC2500 were tested. First, was tried the mode in which TX FIFO is filled with one sample (two bytes) and then the packet containing length field and data is sent to receiver. The result of this operation was not satisfactory, because the delay between consecutive packets was too big. This interval was measured with use of oscilloscope (Figure 68a). One pin of PIC32 was programmed to toggle each time the packet has been transmitted (in WAU Transmitter) or received (in WAU Receiver). In both cases, these delays are equal to about 2ms. Therefore it can be estimated that, in one second would be transmitted/receiver 1000 bytes. That gives the throughput of 8kbps, which is two orders of magnitude smaller than required 500kbps.

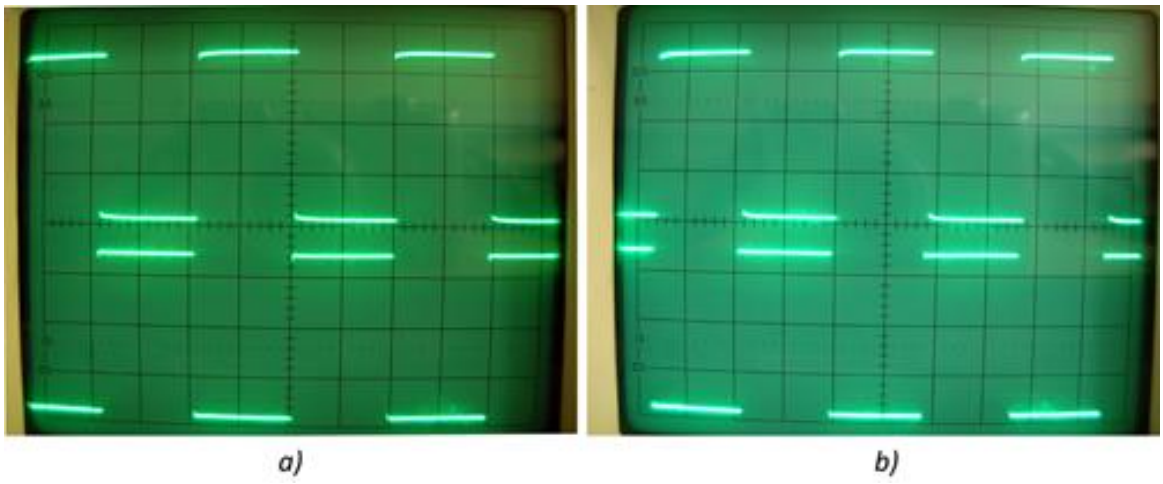


Figure 68: a) One sample transmission (1V/DIV, 1ms/DIV); b) 30 samples transmission (1V/DIV, 2ms/DIV)

In the second attempt, TX FIFO was filled with 30 samples (60 bytes) each iteration. This time, the delay between packets was 4ms (Figure 68b). Consequently, the expected bitrate was 120kbps.

Since these two options did not produce desirable throughput, other had to be considered. Because increasing the number of samples in the package improved throughput, the reasonable step was to go further in packet size. To achieve this, the combination of infinite packet mode and fixed packet mode was adapted. This solution allows to transmit packets of length greater than 255. The procedure proposed in CC2500 design note[DN500] assumes switching between fixed and infinite packet

modes. However, the bitrate has improved slightly. The average throughput of channel when sending 135 samples (270 bytes) was about 135ksps, when sending 225 samples (450 bytes) about 138ksps and when sending 500 samples (1kB) about 148ksps. It can therefore be notices a very slight increase in throughput, with a relatively large increase in the length of the packet. Because this method also proved to be not efficient enough, another was adapted.

The last option which remained, was to use Serial Synchronous Mode. In this case, only configuration of CC2500 was performed with use of SPI interface. Exchange of data was done with use of two-wire interface, where GDO2 pin was configured as serial clock of frequency 500kHz and GDO0 pin as data carrier. Since the exchange of data is done bit by bit with the frequency of clock, the throughput in this mode is equal to 500kbps. To test the operation of transceiver in this mode, PIC32 was programmed to send a single 16bit number in infinite loop. To meet the requirement of this type of communication, this number was sent bit by bit, MSB first. The example results captured with use of oscilloscope are shown on Figure 69. The square wave on the top is the clock signal provided by CC2500. The signals on the bottom are the data loaded serially by the PIC32. The examples show transmissions of two values: Figure 69a $AAAA_{16}$ (1010101010101010_2) and Figure 69b $B5A3_{16}$ (1011010110100011_2). First results were satisfactory, because the bitrate of transfer was 500kbps and the WAU Receiver was collecting the values sent by WAU Transmitter (Figure 70).

The further tests showed however that, after some time of operating, the transmission is becoming unstable. This manifests itself with the fact that some bits are being lost during transmission. The example WAU Receiver output analog signal is shown on Figure 71. It was measured before output capacitor, to keep the DC component. The WAU Transmitter is continually sending a constant 16 bit value ($B5A3_{16}$). Therefore, the output analog signal is expected to have a constant value in time. However, the obtained signal varies in time, which may prove that there is some disturbance in transmission.

At the moment of writing this document, the problem with handling CC2500 transceiver was not solved. For this reason the firmware WAU was not finished and the device does not have its functionality.

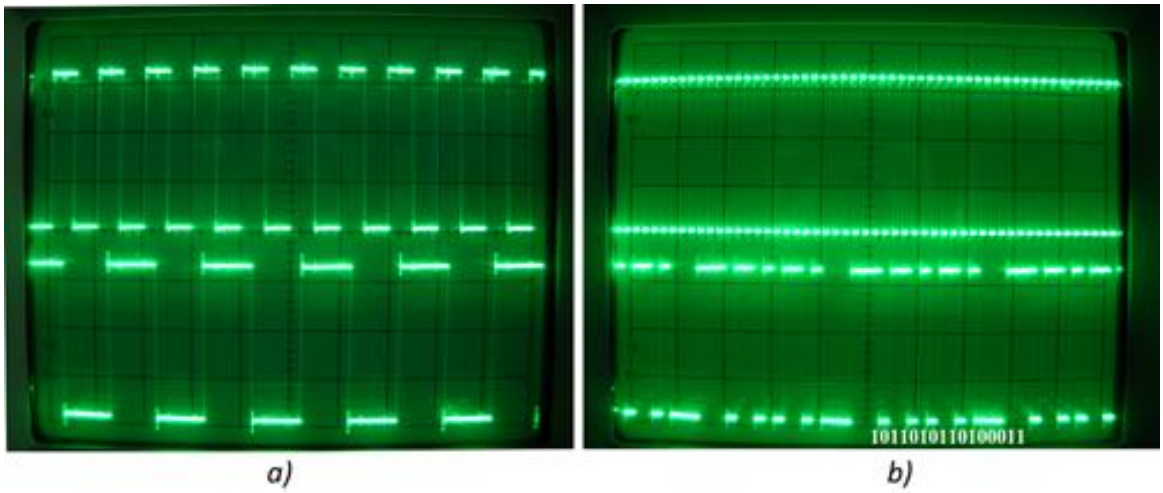


Figure 69: Serial synchronous mode transmission a) Transmission of $AAAA_{16}$ (1V/DIV, 2µs/DIV); b) Transmission of $B5A3_{16}$ (1V/DIV, 10µs/DIV)

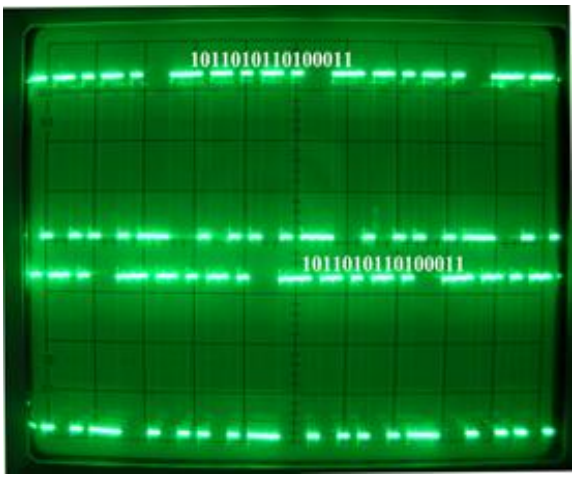


Figure 70: Serial synchronous mode; top signal - WAU Receiver data output, bottom signal - WAU Transmitter data input (1V/DIV, 10µs/DIV)

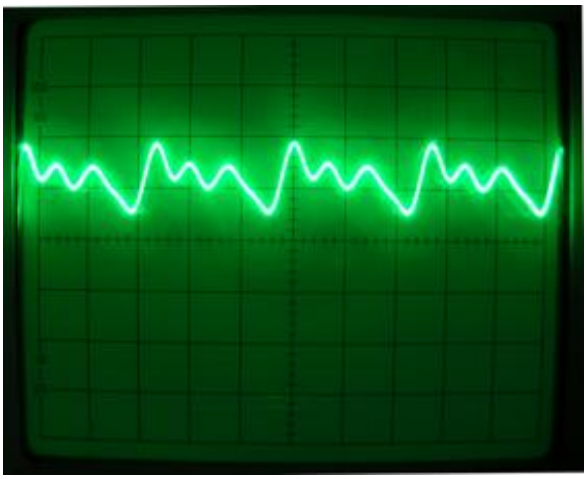


Figure 71: Incorrect WAU Receiver output signal

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

6.1. Conclusions on Wireless Audio Unit

Summarizing the work, which has been done, the following objectives were accomplished:

- The PIC32 Module has been fully realized. Both schematics and PCB were designed, and this unit is working as assumed. The main advantage of this part, is that it can be also easily used in other projects.
- The schematics of Wireless Audio Unit was created and adopted on designed PCB.
- The input analog signal conditioning circuit is performing his task which is preparing the signal for processing. Its output counterpart also fulfills his role.
- The ADC and DAC have been handled by PIC32 firmware and are correctly realizing their functions.
- The compression algorithms were developed and examined in MATLAB software. The tests proved, that they are providing sufficient compression to be used in WAU application.

The difficulties with handling the transceiver module made impossible to complete the project. There were tested different transceiver modes and none of them has not brought satisfying effects. Author believes that the problem lies not in the hardware side, but on the side of the PIC32 firmware.

6.1. Future work proposed

Since the project was not finished and does not have assumed functionality, the primary objective of future development is to complete the work by solving the software problem connected with handling transceiver.

If Wireless Audio Unit had been fully realized, following ideas could be proposed to extend project further:

- To improve quality of transmitted sound, a higher sampling rate can be applied. Both ADC and DAC can work at frequencies higher than 40kHz. The PIC32 firmware can be optimized to provide this possibility. Since the both transducers have 16-bit resolution, increasing of sampling rate can be only possibility to improve quality without interfering in WAU hardware.
- To achieve the objective presented above, a more efficient compression algorithm can be developed. If the audio samples were more compressed, the bitrate of transmitted sound could be greater.

REFERENCES

I2S.specification is: *I2S bus specification*. 1996, Philips Semiconductors.

LTC1569-7.datasheet is: *LTC1569-7 Datasheet*. 1998, Linear Technology Corporation.

LTC1864L.datasheet is: *LTC1864L Datasheet*. 2001, Linear Technology Corporation.

sourceforge.flac is: *FLAC Documentation*. 2006; Available from: http://sourceforge.net/projects/flac/.

LTC2641.datasheet is: *LTC2641 Datasheet*. 2007, Linear Technology Corporation.

CC2500.datasheet is: *CC2500 Datasheet*. 2007, Texas Instruments.

I2C.specification is: *UM10204, I2C-bus specification and user manual*. 2007, Philips Semiconductors.

AD8531.datasheet is: *AD8531 Datasheet*. 2008, Analog Devices.

QFM-TRX1-24G.datasheet is: *QFM-TRX1-24G Datasheet*. 2008, Quasar (UK).

PICkit3 is: *PICkit™ 3 Programmer/Debugger User's Guide*. 2009, Microchip.

TSR-1.datasheet is: *TSR-1 Datasheet*. 2009, Traco Power.

SPI.specification is: *PIC32 Family Reference Manual, Section 23. Serial Peripheral Interface*. 2009, Microchip.

USB.specification is: *PIC32 Family Reference Manual, Section 27. USB On-The-Go (OTG)*. 2009, Microchip.

PIC32.datasheet is: *PIC32MX5XX/6XX/7XX Family Data Sheet*. 2010, Microchip.

SmartRF is: *SmartRF Studio 7*. 2011; v1.5.0 (Rev. I):[Available from: http://focus.ti.com/docs/toolsw/folders/print/smartrftm-studio.html.

Ballou, G., *Handbook for Sound Engineers*. 1991: SAMS.

Crisp, J., *Introduction to Microprocessors and Microcontrollers*. 2004: Newnes.

David J. Katz, R.G., *Embedded Media Processing*. 2006, Norwood, MA: Elsevier.

Everest, F.A., *The Master Handbook of Acoustics*. 2009: McGraw-Hill/TAB Electronics.

Golomb, S.W., *Run-length encodings* 1966: Transactions of the Information Theory Group of the IEEE.

H. G. ter Morsche, G.M., *Signals and Systems*. 1999, Eindhoven: University of Eindhoven.

Jain, S., *Application Note AN095*. 2011, Texas Instruments.

Jasio, L.D., *Programming 32-bit Microcontrollers in C: Exploring the PIC32 (Embedded Technology)*. 2008: Newnes.

John G. Proakis, D.G.M., *Digital Signal Processing: Principles, Algorithms, and Applications*. 1996, New Jersey: Prentice-Hall.

Kefauver, A.P., *Fundamentals of Digital Audio*. 2007: A-R Editions.

Kester, W., *Data Conversion Handbook*. 2005, Norwood, MA: Elsevier.

L.R. Rabiner, R.W.S., *Digital Processing of Speech Signals*. 1978, New Jesrey: Prentice-Hall.

Lyons, R.G., *Understanding Digital Signal Processing (3rd Edition)* 2010: Prentice Hall.

Namtvedt, S., *Design Note DN503*. 2007, Texas Instruments.

Namtvedt, S., *Design Note DN506*. 2007, Texas Instruments.

Namtvedt, S., *Design Note DN500*. 2009, Texas Instruments.

Robinson, T., *SHORTEN: Simple lossless and near-lossless waveform compression*. 1994: Cambridge University Engineering Department.

Rocchesso, D., *Introduction to Sound Processing*. 2003, Verona: University of Verona.

Salomon, D., *Data Compression The complete reference*. 2007, London: Springer-Verlag.

Sayood, K., *Introduction to Data Compression, 3rd Edition*. 2006: Morgan Kaufmann Publishers.

Self, D., *Small Signal Audio Design*. 2010, United Kingdom: Elsevier.

Smith, G., *FPGAs 101: Everything you need to know to get started*. 2010: Newnes.

Smith, M.J.S., *Application-Specific Integrated Circuits*. 1997: Addison-Wesley Professional.