SOLID STATE VOICE RECORDER SYSTEM

By

Muhammed BEYAZ

Engineering Project Report

Yeditepe University

Faculty of Engineering and Architecture

Department of Computer Engineering

2009

SOLID STATE VOICE RECORDER SYSTEM

APPROVED BY:

Assoc. Prof. Dr. Mustafa Türkboyları (Supervisor)	…………………………
	…………………………
	………………………….
Prof. Dr. Şebnem Baydere	…………………………
	…………………………
	………………………….
Assoc. Prof. Dr. Birol Aygün	…………………………
	…………………………
	………………………….

DATE OF APPROVAL:

ACKNOWLEDGEMENTS

Many thanks to my supervisor, Mustafa Türkboyları, for spending his weeks and effort to develop this project board, debug hardware and software and also for his financial support on project.

Special thanks to Bilal Hatipoğlu for his guidance about project and his help to debug software and hardware with me.

ABSTRACT

SOLID STATE VOICE RECORDER SYSTEM

In this project, a voice recording system that has been composed of a microcontroller and a high density NAND flash was implemented. The microcontroller collects the voice samples, passes through the ADPCM codec and stores on the NAND flash. A simple file allocation table has been designed for the files located in the NAND flash. In addition to the hardware part of this project a PC application has been developed. The purpose of the PC application is to download and playback of the stored voice recordings on the NAND flash. The PC application communicates to the microcontroller and transfers the stored voice recordings on the NAND flash to the PC. The PC application converts the stored voice recordings to a common audio format of WAV (WAVE).

ÖZET

KATI HAL SES KAYIT SİSTEMİ

Bu projede, bir mikro denetleyici ve yüksek yoğunlukluklu NAND flaştan oluşan katı hal ses kayıt sistemi gerçekleştirdik. Mikro denetleyici ses örneklerini toplayıp, bu örnekleri ADPCM den geçirip çıkan sonucu NAND flaş üzerine kaydediyor. NAND flaşta bulunan ses kayıtları içinse basit dosya yerleşim tablosu dizayn edildi. Projenin donanımsal kısmına ek olarak bilgisayar uygulamasıda geliştirildi. Bilgisayar uygulamasının amacı ise NAND flaşta bulunan ses kayıtlarını indirmek ve bu kayıtları çalmaktır. Bilgisayar uygulaması mikro denetleyici ile haberleşip bu dosyaları bilgisayar üzerine transfer eder ve ses kayıtlarınının formatını genel olarak bilinen bir format olan WAV (dalga) formatına dönüştürür.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

ADC	Analog to Digital Converter
ADPCM	Adaptive Differential Pulse Code Modulation
AEG	Aallgemein Elektricitat Gesellschaft
AIFF	Audio Interchange File Format
AL	Address Latch
BASF	Baden Aniline and Soda Factory
BSL	Bootstrap Loader
CD	Compact Disk
CL	Command Latch
CPU	Central Processing Unit
DAC	Digital to Analog Converter
DB9S	Data Bus 9 Pins
DCE	Data Communication Equipment
DMA	Direct Memory Access
DPCM	Differential Pulse Code Modulation
DRAM	Dynamic Random Access Memory
DS0	Digital Signal 0
DTE	Data Terminal Equipment
DVD	Digital Versatile Disc
DVI	Digital Video Interface
DVR	Digital Video Recorder
E	Enable
EEPROM	Electronically Erasable Programmable Read-Only Memory
EIA/TIA	Electronic Industry Association and the Telecommunications Industry Association
EPROM	Erasable Programmable Read Only Memory
FAT	File Allocation Table

FSM	Finite State Machine
GSM	Global System for Mobile Communications
IBM	International Business Machines
IFF	Interchange File Format
IMA	Integrated Media Association
I/O	Input/Output
I^2C	Inter-Integrated Circuit
IrDA	Infrared Data Association
ITU-R	International Telecommunication Union Radio Communications Sector
LCD	Liquid Crystal Display
JTAG	Joint Test Action Group
LIN	Local Interconnect Network
LPCM	Linear Pulse Code Modulation
MCU	Microcontroller Unit
MIPS	Millions of Instruction Per Second
MP3	MPEG Audio Layer III
MS-DOS	Microsoft Disk Operating System
NTFS	New Technology File System
ONFI	Open NAND Flash Interface
OPAMP	Operational Amplifier
OS	Operating System
PC	Personal Computer
PCM	Pulse Code Modulation
PCB	Printed Circuit Board
PIC	Programmable Intelligent Computer
P/E/R	Program/Erase/Read
QFN	Quad Flat No Leads
R	Read
RAM	Random Access Memory
RB	Ready Busy
RIFF	Resource Interchange File Format

RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
SDHC	Secure Digital High Capacity
SPI	Serial Peripheral Interface
TSSOP	Thin Small Outline Package
TTL	Transistor Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
VOIP	Voice over IP
W	Write
WP	Write Protect

1. INTRODUCTION

Everyday a new technology product is coming into our houses. One of the most promising technologies is solid state storage. Until now, we have seen MP3 players and video players which are using this new technology. Also, in short term mass storage devices such as harddrives are changing their structures to solid state technology. We want to adopt ourselves to this new technology so we implemented a new solid state voice recording system. Our system is recording on flash but playback function is implemented on PC by using Java programming language.

We want to adopt ourselves to this new technology so we implemented a new solid state voice recorder system. This system is based on a high density NAND flash which is used to store records and a microcontroller which is processing samples, data and transmission of records. First, ADC of microcontroller is capturing the samples by using a carbon microphone. Second, CPU of microcontroller is processing samples and passing them through an ADPCM encoder because raw samples need too many spaces and a recording system must use its space in an intelligent way. Then, these encoded samples are stored on the NAND flash. Voice samples are now on flash but they do not mean anything because a user cannot understand which samples belong to which record so we develop a file system which gives the information about record time, start address of the record and size of the record. Until now, we get samples, record them on NAND flash and have the information about the record but this means nothing to ordinary users and we must get back the records to real world so we develop the PC application which can download and playback the records. To do so, it communicates to MCU with RS-232 industrial standard by using our own communication protocol format and it guarantees the reliability of communication by using Fletcher checksum algorithm. Also, the PC application stores voice recordings as a common audio format WAV.

The organization of this report is as follows: Chapter 2 contains the information about sound recording and its history. Chapter 3 is the design and implementation part of the project.

At this chapter, hardware structure of the project, microcontroller technology, flash memory technology, file system, PC application, sampling, quantization, coding, audio format, communication protocol, microphone circuitry and finite state of project is expressed. Chapter 4 is analysis part of the project about memory usage, timing and power consumption.

Finally, we provided the results and conclusion in the conclusion chapter.

2. BACKGROUND

2.1 Sound Recording

For almost 130 years the technology to preserve and replay sounds, that would otherwise be lost the moment they occurred, have been used. In saving these brief sonic events, people thought, to some extent, personal interaction in "real time" has to be replaced with a one-way communication process that is independent of time. For example, people used to either perform their own music or attend a public performance. Now people are more likely to listen to recordings (or records) than gather around the piano or go to a concert. Music and, to a lesser extent, the spoken word have become products that are consumed rather than an activity that are shared. Thanks to the mass production of records and players, people now have access to more diverse styles and more expertly performed music than ever before. Ironically, few of us can now sing the songs our grandparents sang.

Sound recording and reproduction is an electrical or mechanical inscription and re-creation of sound waves, such as spoken voice, singing, instrumental music, or sound effects. The two main classes of sound recording technology are analog recording and digital recording.

2.2 Types of Recording

2.2.1 Analog recording

Analog (or analogue) recording (Greek, ana is "according to" and logos "relationship") is a technique used to store signals of audio or video information for later playback.

In analog recording, original sound signal is modulated onto another physical medium as a continual wave in or on the media. The wave might be stored as a physical texture on a phonograph record, or a fluctuation in the field strength of a magnetic recording. This is different from digital recording, which converts audio signals into discrete numbers and also the physical quality in the medium (e.g., the intensity of the magnetic field or the path of a record groove) is directly related, or analogous, to the physical properties of the original sound (e.g., the amplitude, phase, etc.)

Analog recording methods store audio signals as a continual wave in or on the media. The first successful demonstration of analog recording for audio was by Thomas Alva Edison.

2.2.2 Digital Recording

A digital recording is produced by converting the physical properties of the original sound into a sequence of numbers, which can then be stored and played back for reproduction. The accuracy of the conversion process depends on the sampling rate (how often the sound is sampled and a related numerical value is created) and the sampling depth (how much information each sample contains, which can also be described as the maximum numerical size of each sampled value). However, unlike analog recording which depends critically on the long-term durability of the fidelity of the waveforms recorded on the medium, the physical medium storing digital samples is essentially immaterial in playback of the encoded information so long as the original sequence of numbers can be recovered.

2.3 History of Recording

2.3.1 Phonograph

There was sound recording before the phonograph, but not sound reproduction. The record player, phonograph or gramophone was the most common device for playing recorded sound from the 1870s through the 1980s.

The earliest known invention of a phonographic recording device was the phonautograph, invented by Frenchman Edouard-Leon Scott de Martinville and patented on March 25, 1857. It could transcribe sound to a visible medium, but had no means to play back the sound after it was recorded. In 2008, phonautograph recordings were for the first time played back as sound by American audio historians, using computers to decode the transcribed waveforms. [1]

Charles Cros, a French scientist, produced a theory (April 18, 1877) concerning a phonograph, but he did not manufacture a working model. His theory was submitted to the French Academy of Sciences, and was read to the public in December 1877, by which time Edison had produced a working model. Cros and Edison apparently discovered their theories independently.

Thomas Alva Edison conceived the principle of recording and reproducing sound between May and July 1877 as a byproduct of his efforts to "play back" recorded telegraph messages and to automate speech sounds for transmission by telephone. [2][3] He announced his invention of the first phonograph is shown in Figure 2.1, a device for recording and replaying sound, on November 21, 1877, and he demonstrated the device for the first time on November 29 (it was patented on February 19, 1878 as US Patent 200,521). Edison's early phonographs recorded onto a tinfoil sheet phonograph cylinder using an up-down ("hill-and-dale") motion of the stylus. [4][5]

The tinfoil sheet was wrapped around a grooved cylinder, and the sound was recorded as indentations into the foil. Edison's early patents show that he also considered the idea that sound could be recorded as a spiral onto a disc, but Edison concentrated his efforts on cylinders, since the groove on the outside of a rotating cylinder provides a constant velocity to the stylus in the groove, which Edison considered more "scientifically correct". [6]

Edison's patent specified that the audio recording be embossed, and it was not until 1886 that vertically modulated engraved recordings using wax coated cylinders were patented by Chichester Bell and Charles Summer Tainter. They named their version the graphophone.

Emile Berliner patented his gramophone in 1887. The gramophone involved a system of recording using a lateral (back and forth) movement of the stylus as it traced a spiral onto a zinc disc coated with a compound of beeswax in a solution of benzene. The zinc disc was immersed in a bath of chromic acid; this etched the groove into the disc where the stylus had removed the coating, after which the recording could be played.

Figure 2.1: Edison Phonograph

2.3.2 Magnetic Tape

Magnetic tape (in Figure 2.2) is a continuous plastic strip covered with magnetic oxide. Nearly all recording tape is of this type, whether used for recording audio or video or for computer data storage.

Practical magnetic tape recording originated with an Austrian inventor living in Dresden, Germany, named Fritz Pfleumer. In the 1920s, sales of cigarettes boomed, and the manufacturers wanted a cheap alternative to the expensive gold leaf band that decorated the tips of the most expensive cigarette brands. An expert in special paper and related processes for industrial uses, Pfleumer succeeded in creating a process for striping a cheap gold-colored bronze band on the cigarette paper. Pfleumer was interested in the magnetic recording process, and reasoned that he could use his cigarette-paper manufacturing process to create a magnetizable paper recording tape that would replace expensive and impractical wire and steel.

The inventor glued pulverized iron particles onto a 16 millimeter-wide strip of paper, creating the first magnetic tape. He also built the first tape recorder in the modern sense. For his discovery, Pfleumer received German patent DE 500 900 on 31 January 1928, titled "Lautschriftträger" (sound record carrier) and, amongst others, British patent GB 333,154, "Improvements in or relating to Sound Records", on February 5, 1929. The audio tape recorder produced poor quality sound that nonetheless offered great promise. The paper tape tore readily, but could also easily be repaired with glue instead of the welding method of steel and wire. In his promotion of the invention, Pfleumer pointed out that editing and assembling the edited pieces of his recording tape were as easy as handling movie film, a well-established technology. [7]

AEG realized that this invention was one of the most promising technology. AEG signed contract with Pfleumer. The AEG engineers quickly found they were better at designing electronics and tape transports than they were at experimenting with the chemical processes of iron powder, glues, and coating methods. So they contacted the logical people to help them with the job, the BASF division in Ludwigshafen of the giant chemical combine I.G. Farbenindustrie. The two made a gentlemen's agreement", later formalized in a written contract, that AEG would build the machines and BASF would make the tape, with their development teams cooperating closely.

There was Second World War so this invention was kept in secret. It was only after the war that Americans, particularly Jack Mullin, John Herbert Orr and Richard H. Ranger were able to bring this technology out of Germany and develop it into commercially viable formats.

A wide variety of recorders and formats have developed since, most significantly reel to reel and Compact Cassette.

Figure 2.2: Magnetic Tape

3. DESIGN AND IMPLEMENTATION

3.1 Development Board

Embedded system project needs a development board which is designed by considering project requirement. We do not have such a board so I have learnt to draw PCB (Printed Circuit Board) by using Mentor Graphics' PADS Logic and Layout program and then I have designed three types of board in order to develop this project.

First board is based on MSP430F169 and NAND Flash but its features are not good enough to solve problem. Its data pins are separate from each other around different ports and this creates a big problem about timing of data.

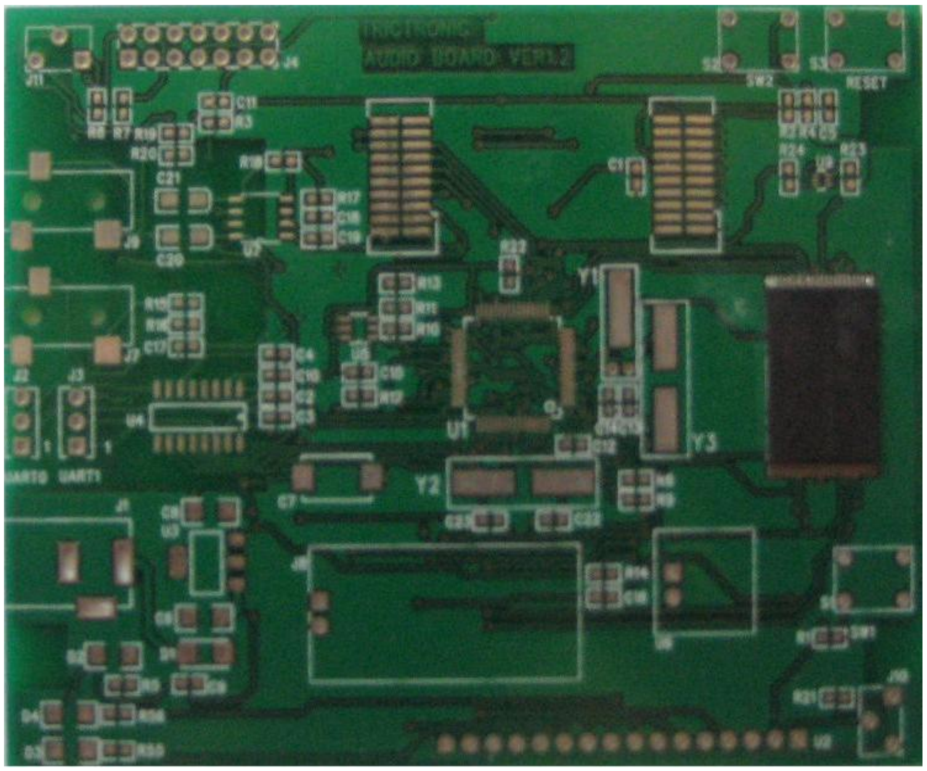


Figure 3.1: First development board

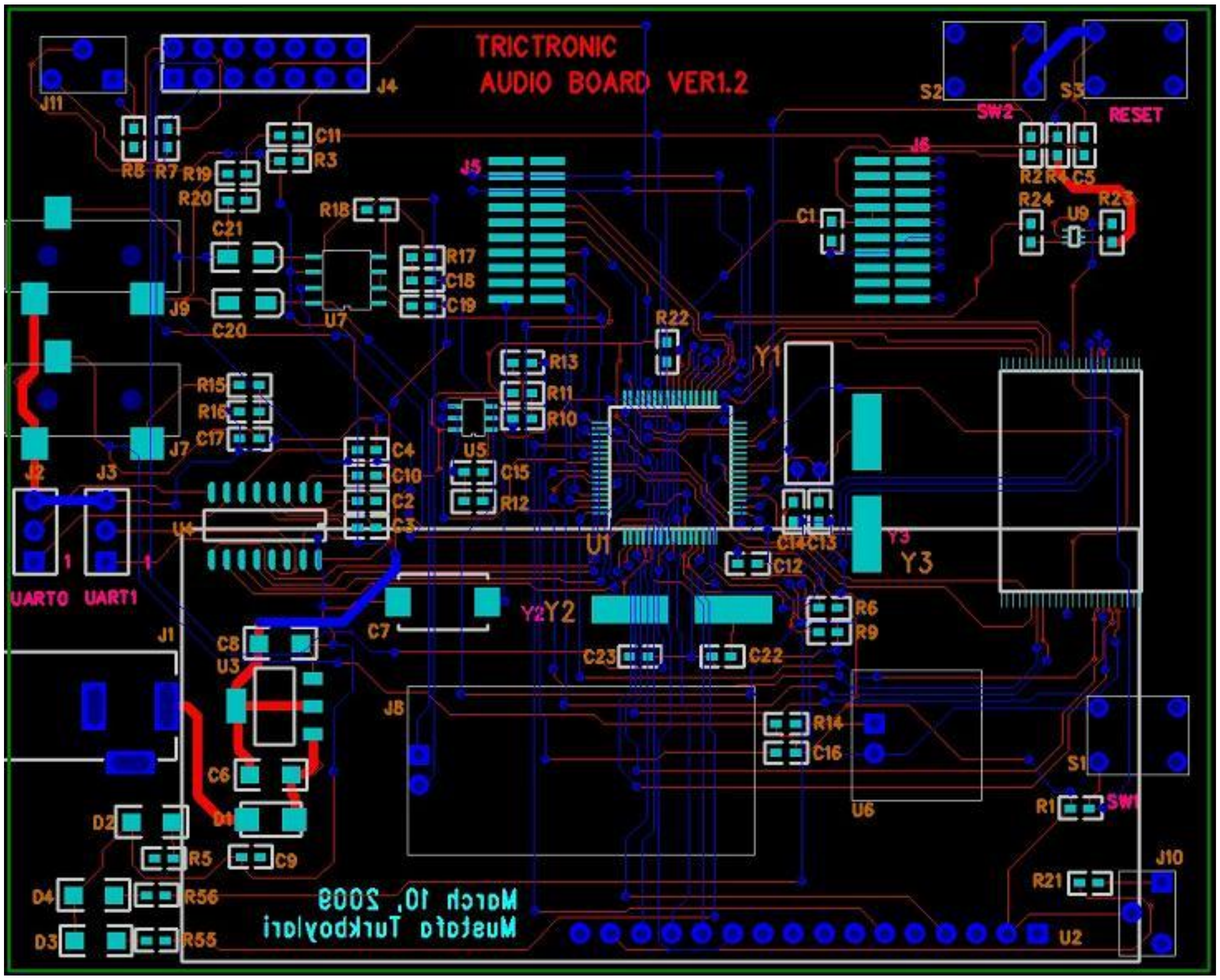


Figure 3.2: PCB of first development board

Second board is based on MSP430F2274 and NAND Flash but it has two big problems. First problem is about flash's PCB design, flash does not have some open drain pull up resistance. Second problem is about microphone circuitry. Microphones pins are connected to ground so there is no a voltage difference between pins and we can not get any sample.

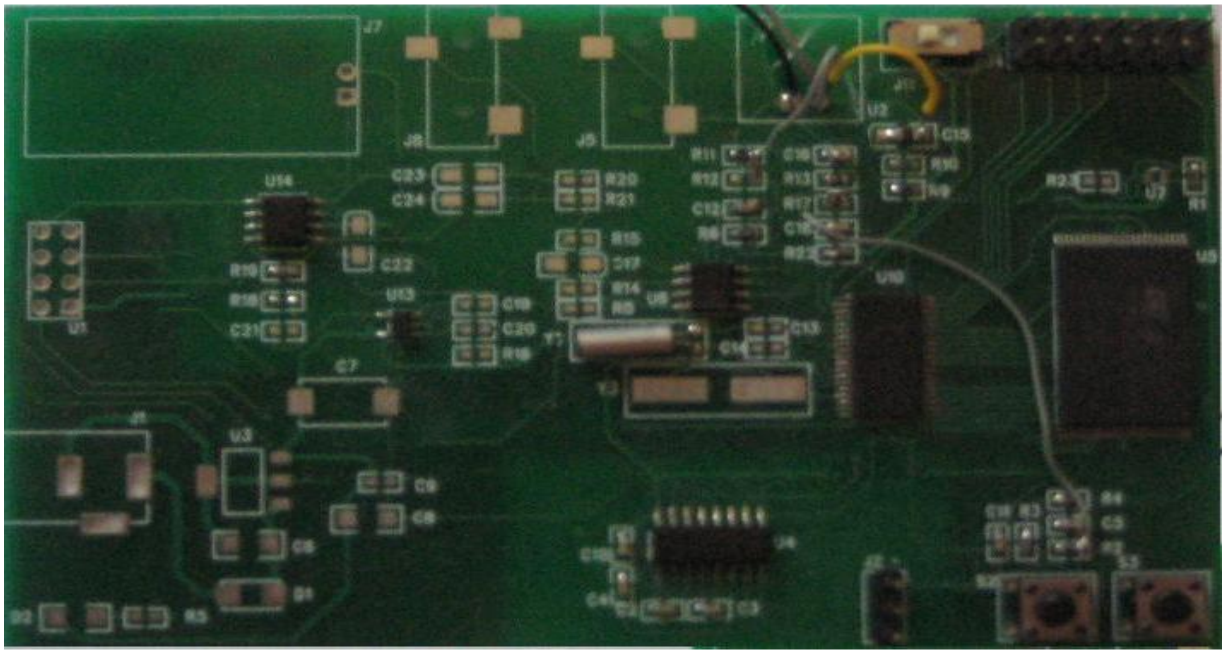


Figure 3.3: Second development board

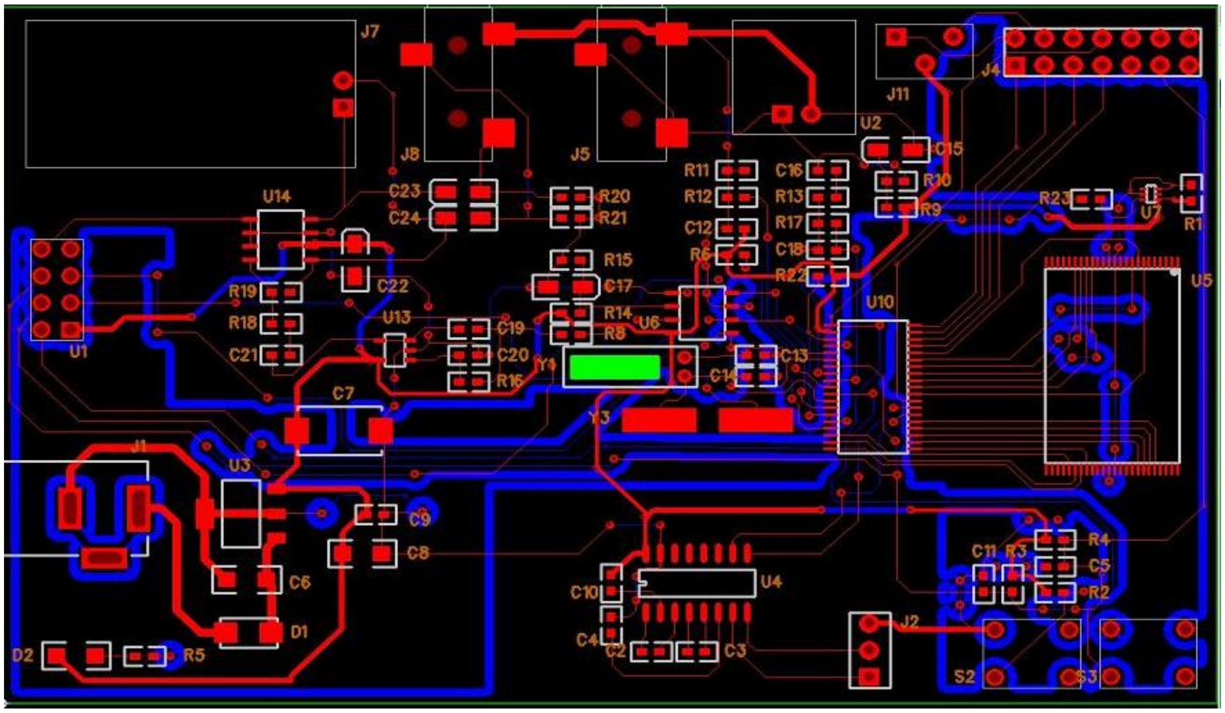


Figure 3.4: PCB of second development board

Third board is also based on MSP430F2274 and NAND Flash. Its designed is like same second board but third one has two microphone circuits, one of the circuits is active but I can choose which one will be active. Third one has two more additional connectors, RJ10 and RJ12. Open drain pull up resistances are also designed. It also has test points to test flash's pin configuration.

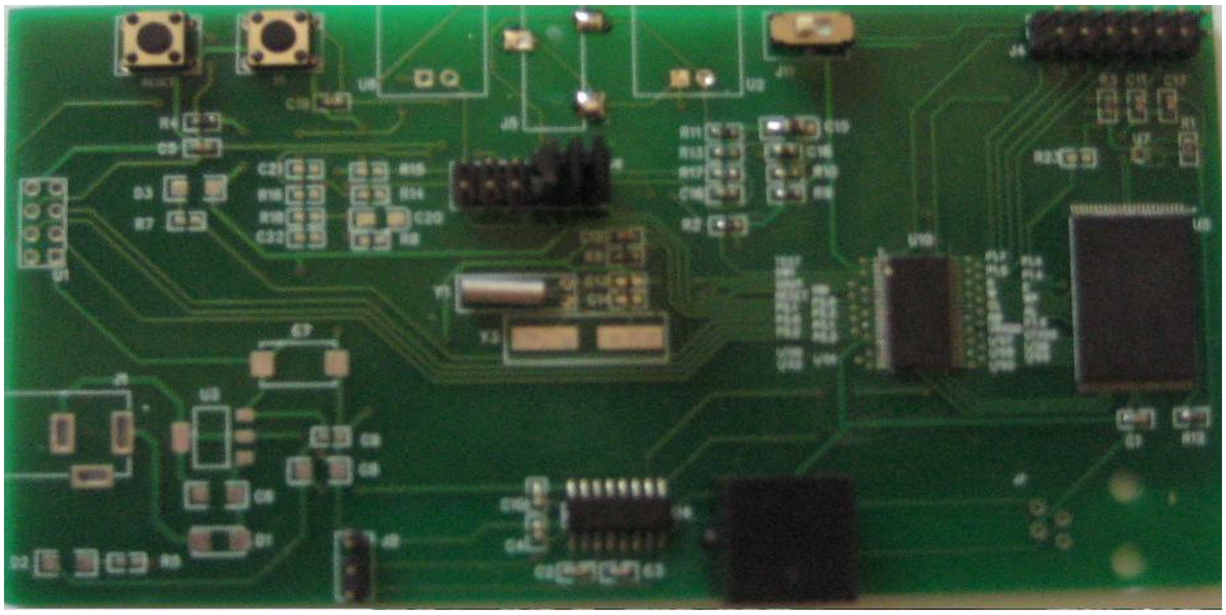


Figure 3.5: Third development board

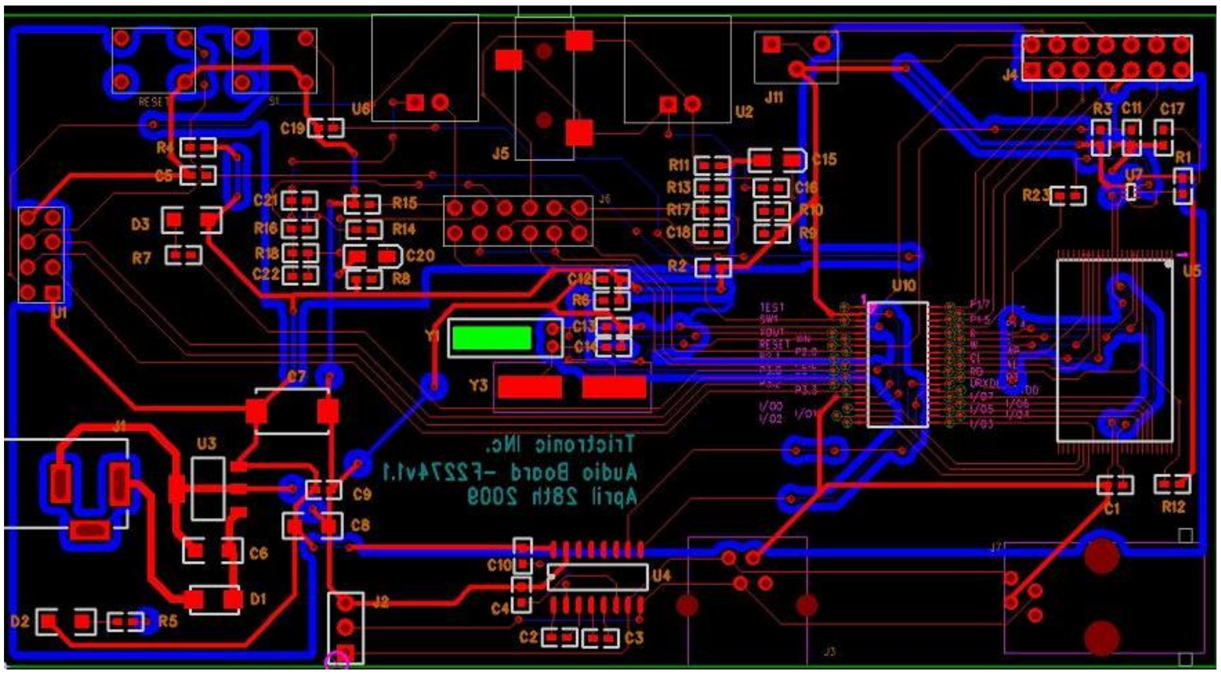


Figure 3.6: PCB of third development board

Three boards have flash memory, UART module, buttons, microphone circuit and external clocks. First board has MSP430F169, wireless module slot and LCD. The second and third one have MSP430F2274 and operational amplifiers. We chose third one because it has more functional units and does not have any problematic structure.

3.2 Microcontroller Technology

Microcontroller (also called as microcontroller unit) is a kind of small computer on a single integrated circuit consisting of relatively small CPU compared with a normal PC and also supported with crystal oscillator, timers, watchdog and analog I/O. It also has other functionalities such as ROM or a flash which is small capacity and has simple read/write operations. Microcontroller is used in small embedded systems. It usually waits for an input from a sensor or an interrupt which is usually generated by pressing a button and can sleep if there is no event. It operates at very low frequencies such as 32 KHz so it is used at low power consumption, long life stable, battery used embedded applications such as wireless sensors, remote control, engine control, etc. There are many different well known microcontroller families such as PIC, 8051 and MSP430. [8]

PIC (Programmable Intelligent Computer) is the first microcontroller unit which uses Harvard architecture and is created by Microchip Company at middle of 1970s. Harvard architecture is the computer architecture which has separate instruction memory and data memory. PIC's 14 cores model is the most well known model at industry because it is good to program and affordable but the separate memories makes it much slower than other structures. [9]

8051 is another well known microcontroller unit which is created by Intel Company at 1980s. It is an ancient model but its basic structure is well organized so many different companies (such as Atmel, Dallas Semiconductors and Philips) buy it and make their enhancement. Because of these reasons, it has wide application areas. [10]

MSP430 is one of the most promising microcontroller families from Texas Instruments. It is built around 16 bit CPU and designed for low cost, low battery consumption embedded applications. MSP430 enables system designers to simultaneously interface to analog signals, sensors and digital components while maintaining unmatched low power. Typical applications include utility metering, portable instrumentation, intelligent sensing, and consumer electronics. [36]

Its peripherals are usually watchdog timer, timers, DMA, ADC, DAC, flash memory, I^2C, SPI, USART, OPAMPs, brownout reset and LCD. Apart from some older EPROM (PMS430E3xx) and high volume mask ROM (MSP430Cxxx) versions, all of the devices are in-system programmable via JTAG or a built in bootstrap loader (BSL) using RS-232 but the included peripherals vary from device to device.

MSP430 is a good choice to be used in low powered embedded applications. In idle mode its power consumption is lower than 1 micro ampere. Its clock frequency varies and the top speed is 25 MHz. MIPS and MHz do not mean the same thing. MIPS means millions of instructions per second and MHz is the number of clock signals generated by CPU divide by one million, so sometimes lower clock frequencies have bigger MIPS rates.

Consumers usually need more than the system has but the designers need some specifications and limitations to make it much simpler. MSP430 does not have external memory bus so it has a limited flash memory from 10 KB to 32 KB so it is not suitable for larger memory and buffer needed application.

MSP430 uses von Neumann architecture which means a single address space for data and instructions. Its memory is byte addressable and pair of bytes are mapped as little endian format to get a 16-bit word.

CPU has 16 bit registers and these registers are used as program counter, stack pointer, constant generators and general purpose registers.

The instruction set is very simple; there are 27 instructions in three families. Most instructions are available in 8-bit (byte) and 16-bit (word) versions, depending on the value of a B/W bit - the bit is set to 1 for 8-bit and 0 for 16-bit. Byte operations to memory affect only the addressed byte, while byte operations to registers clear the most significant byte. [11][12]

3.2.1 Key Features

Key features of the MSP430x2xx family include:

- Ultra low-power architecture extends battery life
    - 0.1-µA RAM retention
    - 0.8-µA real-time clock mode
    - 250-µA / MIPS active

- High-performance analog ideal for precision measurement
    - Comparator-gated timers for measuring resistive elements

- 16-bit RISC CPU enables new applications at a fraction of the code size.
    - Large register file eliminates working file bottleneck
    - Compact core design reduces power consumption and cost
    - Optimized for modern high-level programming
    - Only 27 core instructions and seven addressing modes
    - Extensive vectored-interrupt capability

- In-system programmable Flash permits flexible code changes, field upgrades and data logging

Details and specification of MSP430 is not included in this report. You can find very detailed description and specifications of MSP430F2274 in MSP430x2xx Family User Guide, MSP430F2274 Datasheet and MSP430F2274 Device Erratasheet. [11][12][13]

The block diagram of MSP430F2274 is shown in Figure 3.7:

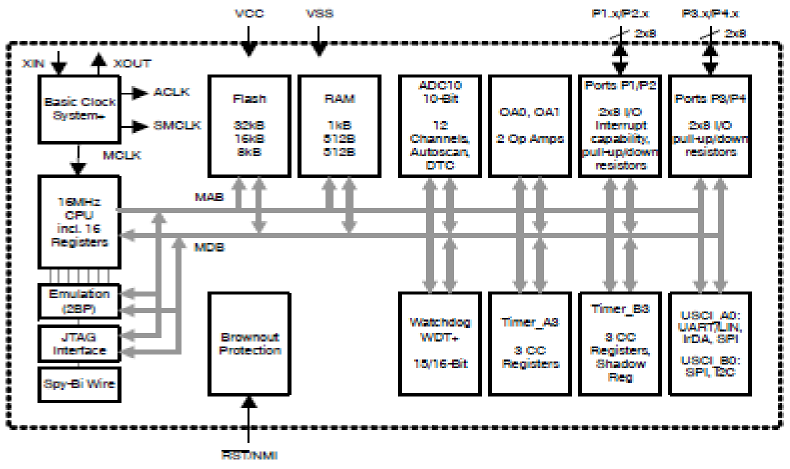


Figure 3.7: Block diagram of MSP430F2274

3.2.2 History of MSP430

MSP430 family is established approx. 10 years ago. In the beginnings there are no UARTs and LCD-display. Many peripherals are not included, they are just in mind so it is not attractive and there are no good development tools. At the end of nineties , the MSP430 family and its development tools became very attractive because the cost of system has been lowered dramatically and JTAG based programming & debugging in flash based devices made development tools cheaper. Then MSP430 became a good competitor for 8051 and PIC for low budget wide range industry.

3.2.3 Pros

MSP430 is used in a wide range application size from smaller to bigger and its current competitors are usually PIC and 8051 architecture.

MSP430 uses von Neumann architecture so it does not have an external bus. It has linear address space for 8 bit and 16 bit instructions. By the use of a stable well known linear address space; programming from memory to register, register to memory, peripherals to memory and vice versa have an appropriate way to communicate over an efficient protocol. Single 16 bit transfer from memory to peripheral and vice versa changes the value of variables by interrupts so usually C programmers make their biggest mistakes to think programming MSP430 as usual C programming so they must consider about this before starting to program.

Harvard architecture was used more common in the past. Using separate address space and data space made programming easier in old days. The needs for using larger memory are starting to grow up rapidly, so then everything turns back because very large external memory and flash are not easy to manage. It became complicated to program. Codes were not modular and reusable. All of these reasons make MSP430 usage more common. MSP430’s linear address space creates good code utilization for programmers.

3.2.4 Cons

MSP430s market usage is growing up rapidly but there are some small problems. In past, many companies used PIC and 8051 so the development tools were created according to them so MSP430 does not have sufficient number of programming environment. Another problem is MSP430s development tools are based on C and assembly programming languages but programmers want to have higher level more modular programming environment. These small problems are being solved in a few years because of growing market rate.

3.2.5 Why do we choose MSP430F2274?

Other key features of MSP430F2274:

- Basic Clock Module Configurations:
  - Internal Frequencies up to 16 MHz With
  - Four Calibrated Frequencies to ±1%
  - Internal Very-Low-Power Low-Frequency
  - Oscillator
  - 32-kHz Crystal
  - High-Frequency Crystal up to 16 MHz
  - Resonator
  - External Digital Clock Source
  - External Resistor

- 16-Bit Timer_A With Three Capture/Compare Registers

- 16-Bit Timer_B With Three Capture/Compare Registers

- Universal Serial Communication Interface
    - Enhanced UART Supporting
    - Auto-Baudrate Detection (LIN)
    - IrDA Encoder and Decoder
    - Synchronous SPI
    - I2C™

- 10-Bit, 200-ksps A/D Converter With Internal Reference, Sample-and-Hold, Autoscan, and Data Transfer Controller

- Two Configurable Operational Amplifiers

- Brownout Detector

- Serial Onboard Programming, No External Programming Voltage Needed Programmable Code Protection by Security Fuse

- Bootstrap Loader

- On Chip Emulation Module

- 1 KB RAM

- 32 KB + 256 B Flash Memory

In out project, we need operational amplifiers for microphone circuit, ADC to convert voice to digital pulse, timer_a for voice sampling, timer_b for time calculation, clock module for 32 KHz crystal, enhanced UART support for PC communication RS-232, data transfer controller to write voice to flash without usage of CPU and other features when we need. MSP430F2274 has two types of package: one of is 40-pin QFN and the other one is 38-pin

TSSOP. We choose 38-pin TSSOP package type because it is easy to be populated and soldered. [11][12]

3.3 Flash Memory

Flash memory technology is a mix of EPROM and EEPROM technologies. The term “flash” was chosen because a large chunk of memory could be erased at one time. The name, therefore, distinguishes flash devices from EEPROMs, where each byte is erased individually. Flash memory technology is today a mature technology. It is a strong competitor to other nonvolatile memories such as EPROMs and EEPROMs, and to some DRAM applications. Flash is a solid state device so for the rest of this thesis, we refer solid state to flash.

In the internal circuit configuration of NOR Flash, the individual memory cells are connected in parallel, which enables the device to achieve random access. This configuration enables the short read times required for the random access of microprocessor instructions. NOR Flash is ideal for lower-density, high-speed read applications, which are mostly read only, often referred to as code-storage applications.

NAND Flash was developed as an alternative optimized for high-density data storage, giving up random access capability in a tradeoff to achieve a smaller cell size, which translates to a smaller chip size and lower cost-per-bit. This was achieved by creating an array of eight memory transistors connected in a series. Utilizing the NAND Flash architecture’s high storage density and smaller cell size, NAND Flash systems enable faster write and erase by programming blocks of data. NAND Flash is ideal for low-cost, high-density, high-speed program/erase applications, often referred to as data-storage applications. [14][15]

Comparison of NOR and NAND Flash is shown in Figure 3.8:

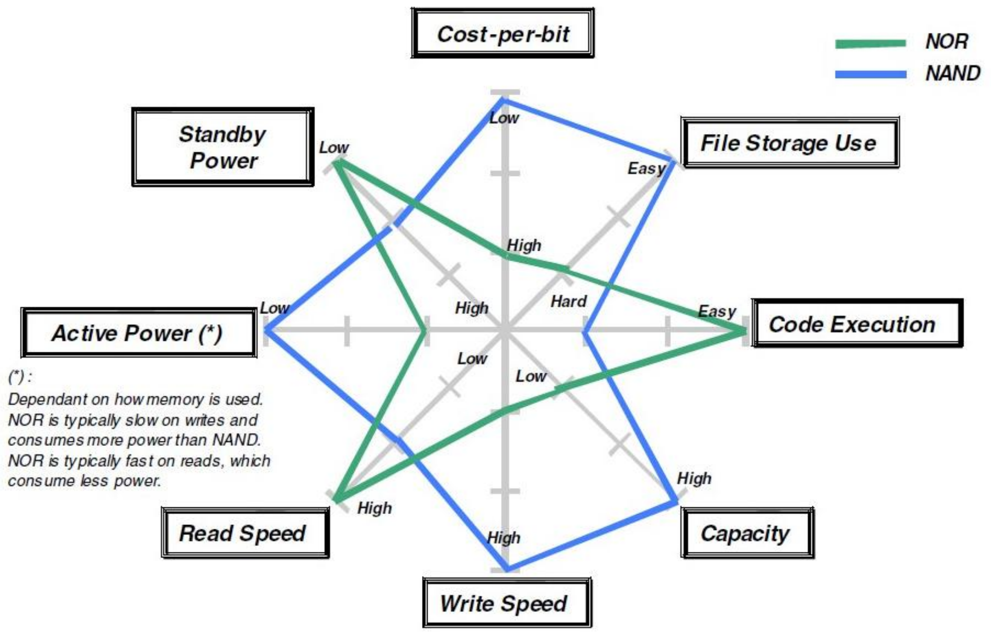


Figure 3.8: Comparison of NAND and NOR flash

The communication between flash and MCU is divided into two categories; one is serial and the other one is parallel. Serial communication needs SPI communication interface so the clock rate is much more important. Parallel communication does not need any SPI communication interface but reading is slower than serial one. Serial communication sends data as bits but parallel communication usually sends data as byte or two bytes. [16]

3.3.1 Which one do we choose?

Flash memory based voice recording system depends on two key features. The first feature is selection of MCU and we did it above. The second important feature is flash memory technology.

Voice recording is a data storage processing. Writing speed is very important at voice recording system because the system needs no sample loss to get a good voice record and data integrity. Reading speed is not so important for this development because reading from flash is done by RS-232 communication in order to transfer data to PC application.

According to above properties, we choose parallel NAND flash memory. Parallel communication is required to write samples at the same time. NAND technology is used as data storage. The writing speed is much better in NAND and reading speed is not so important but the sequential reading speed is also sufficient to carry communication busy and not to lose connection.

The used product is Numonyx NAND02G-B2D.

3.3.2 Key Features

Numonyx NAND02G-B2D's key features are :

- High density NAND flash memory
    - Up to 2 Gbits of memory array
    - Cost-effective solution for mass storage applications

- NAND interface
    - x8 or x16 bus width
    - Multiplexed address/data

- Supply voltage: 1.8 V or 3 V device

- Page size
    - x8 device: (2048 + 64 spare) bytes
    - x16 device: (1024 + 32 spare) words

- Block size
    - x8 device: (128 K + 4 K spare) bytes
    - x16 device: (64 K + 2 K spare) words

- Multiplane architecture
    - Array split into two independent planes
    - Program/erase operations can be performed on both planes at the same time

- Page read/program
    - Random access: 25 us (max)
    - Sequential access: 25 ns (min)
    - Page program time: 200 us (typ)
    - Multiplane page program time (2 pages): 200 us (typ)

- Copy back program with automatic EDC (error detection code)

- Cache read mode

- Fast block erase
    - Block erase time: 1.5 ms (typ)
    - Multiblock erase time (2 blocks): 1.5 ms(typ)

- Status register

- Electronic signature

- Chip Enable ‘don’t care’

- Serial number option

- Data protection:
  - Hardware program/erase disabled during power transitions
  - Non-volatile protection option

- ONFI 1.0 compliant command set

- Data integrity
  - 100,000 program/erase cycles (with ECC)
  - 10 years data retention

- ECOPACK® packages

We choose x8 device because MCU does not have enough pins (x8 uses 15 pins andx16 uses 23pins) and x8 is enough to save voice samples. [17]

3.3.3 Programming Flash

NAND flash is a non volatile memory structure so programming NAND flash is not as easy as programming a volatile memory.

NAND flash basically has two structures. First one is page which has 2048 byte data and 64 byte spare area. Data part is the part where audio samples are stored and spare are is the part where error correction and other stuff can done. Second structure is block which consists of 64 pages. If error correction is implemented and on then each block can be programmed and erased up to 100,000 cycles.

Each page can read and program individually. Up to 4 (included) different programming attempts can be done to a page but if there is a need for fifth attempt then the block ,which programming page is in, must be erased completely. [17]

Used architecture is based on x8.Block diagram of flash is shown in Figure 3.9:

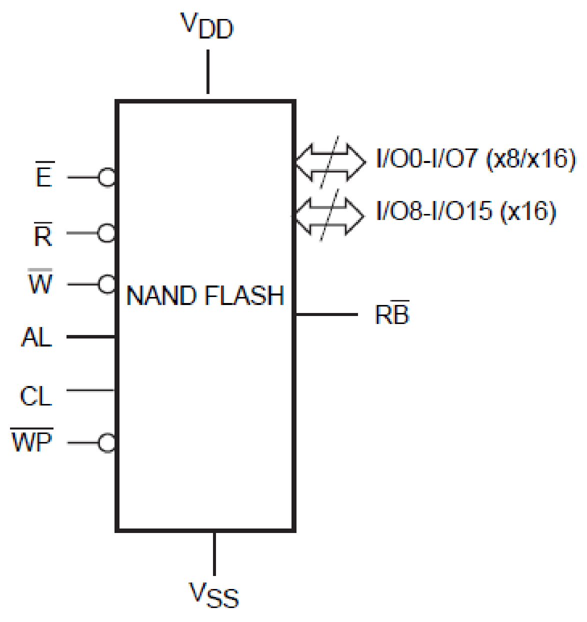
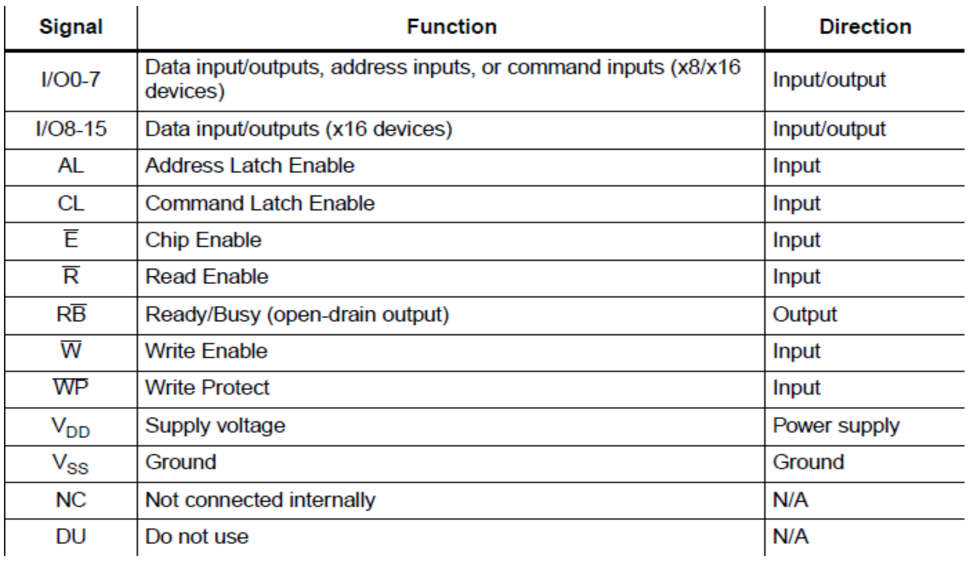


Figure 3.9: Block diagram of flash

Signal names of flash are shown in Table 3.1:

Table 3.1: Signal names of flash



Signal	Function	Direction
I/O0-7	Data input/outputs, address inputs, or command inputs (x8/x16 devices)	Input/output
I/O8-15	Data input/outputs (x16 devices)	Input/output
AL	Address Latch Enable	Input
CL	Command Latch Enable	Input
$\overline{E}$	Chip Enable	Input
$\overline{R}$	Read Enable	Input
$R\overline{B}$	Ready/Busy (open-drain output)	Output
$\overline{W}$	Write Enable	Input
$\overline{WP}$	Write Protect	Input
V_{DD}	Supply voltage	Power supply
V_{SS}	Ground	Ground
NC	Not connected internally	N/A
DU	Do not use	N/A

Signal description of flash is [17]

- Inputs/outputs (I/O0-I/O7)

  Input/outputs 0 to 7 input the selected address, output the data during a read operation, or input a command or data during a write operation. The inputs are latched on the rising edge of Write Enable. I/O0-I/O7 are left floating when the device is deselected or the outputs are disabled.

- Address Latch Enable (AL)

  The Address Latch Enable activates the latching of the address inputs in the command interface. When AL is High, the inputs are latched on the rising edge of Write Enable.

- Command Latch Enable (CL)

  The Command Latch Enable activates the latching of the command inputs in the command interface. When CL is High, the inputs are latched on the rising edge of Write Enable.

- Chip Enable (E)

  The Chip Enable input, E, activates the memory control logic, input buffers, decoders and sense amplifiers. When Chip Enable is Low, V_{IL}, the device is selected. If Chip Enable goes High, V_{IH}, while the device is busy, the device remains selected and does not go into standby mode.

- Read Enable (R)

  The Read Enable pin, R, controls the sequential data output during read operations. Data is valid t after the falling edge of R. The falling edge of R also increments the internal column address counter by one.

- Write Enable (W)

  The Write Enable input, W, controls writing to the command interface, input address, and data latches. Both addresses and data are latched on the rising edge of Write Enable. During power-up and power-down a recovery time of 10 us (min) is required before the command interface is ready to accept a command. It is recommended to keep Write Enable high during the recovery time.

- Write Protect (WP)

  The Write Protect pin is an input that gives a hardware protection against unwanted program or erase operations. When Write Protect is Low, VIL, the device does not accept any program or erase operations. It is recommended to keep the Write Protect pin Low, VIL, during power-up and power-down.

- Ready/Busy (RB)

  The Ready/Busy output, RB, is an open-drain outputs that can be used to identify if the P/E/R controller is currently active. When Ready/Busy is Low, V_{OL}, a read, program or erase operation is in progress. When the operation completes, Ready/Busy goes High, V_{OH}. The use of an open-drain output allows the Ready/Busy pins from several memories to be connected to a single pull-up resistor. A Low then indicates that one or more of the memories is/are busy. During power-up and power-down a minimum recovery time of 10 us is required before the command interface is ready to accept a command. During this period the RB signal is Low.

- V_{DD} supply voltage

  V_{DD} provides the power supply to the internal core of the memory device. It is the main power supply for all operations (read, program and erase). Each device in a system should have V_{DD} decoupled with a 0.1 uF capacitor. The PCB track widths should be sufficient to carry the required program and erase currents.

- V_{SS} ground

  Ground, V_{SS}, is the reference for the power supply. It must be connected to the system ground.

There are six standard bus operations that control the memory:

- Command input

  Command input bus operations give commands to the memory.

- Address input

  Address input bus operations input the memory addresses. Five bus cycles are required to input the addresses.

- Data input

  Data input bus operations input the data to be programmed.

- Data output

  Data output bus operations read the data in the memory array, the status register, the electronic signature, and the unique identifier.

- Write protect

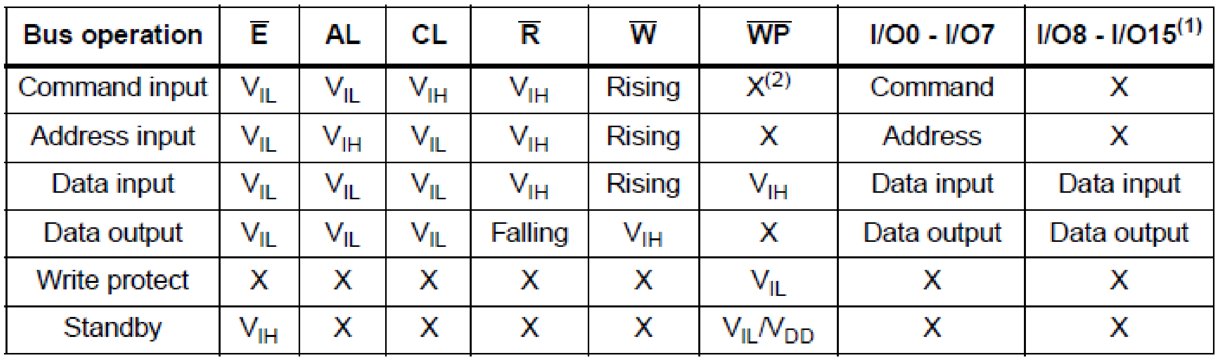
  Write protect bus operations are used to protect the memory against program or erase operations. When the Write Protect signal is Low, the device does not accept program or erase operations, and, therefore, the contents of the memory array cannot be altered. The Write Protect signal is not latched by Write Enable to ensure protection, even during power up.

- Standby

  When Chip Enable is High the memory enters standby mode, the device is deselected, outputs are disabled, and power consumption is reduced.

Pin descriptions of bus operations are shown in below Table 3.10:

Table 3.2: Pin description of bus operations



Bus operation	$\overline{E}$	AL	CL	$\overline{R}$	$\overline{W}$	$\overline{WP}$	I/O0 - I/O7	I/O8 - I/O15(1)
Command input	V_{IL}	V_{IL}	V_{IH}	V_{IH}	Rising	X(2)	Command	X
Address input	V_{IL}	V_{IH}	V_{IL}	V_{IH}	Rising	X	Address	X
Data input	V_{IL}	V_{IL}	V_{IL}	V_{IH}	Rising	V_{IH}	Data input	Data input
Data output	V_{IL}	V_{IL}	V_{IL}	Falling	V_{IH}	X	Data output	Data output
Write protect	X	X	X	X	X	V_{IL}	X	X
Standby	V_{IH}	X	X	X	X	V_{IL}/V_{DD}	X	X

Four types of commands are used in the project. First one is sequential which reads data sequentially reading from flash memory. Second one is sequential programming page which write to flash sequentially. Third one is block erase which erase whole pages in the given block. The last one is reading signature of flash memory which reads properties of the flash.

3.4 File System

Most users of computers are roughly familiar with what a file system does, what a file is, what a directory is, and so on. This knowledge is gained from direct experience with computers. Instead of basing our discussion on prior experiences, which will vary from user to user, we will start over again and think about the problem of storing information on a computer, and then move forward from there.

The main purpose of computers is to create, manipulate, store, and retrieve data. A file system provides the machinery to support these tasks. At the highest level a file system is a way to organize, store, retrieve, and manage information on a permanent storage medium such as a disk. File systems manage permanent storage and form an integral part of all operating systems.

There are many different approaches to the task of managing permanent storage. At one end of the spectrum are simple file systems that impose enough restrictions to inconvenience users and make using the file system difficult. At the other end of the spectrum are persistent object stores and object-oriented databases that abstract the whole notion of permanent storage so that the user and programmer never even need to be aware of it. The problem of storing, retrieving, and manipulating information on a computer is of a general-enough nature that there are many solutions to the problem.

There is no “correct” way to write a file system. In deciding what type of filing system is appropriate for a particular operating system, we must weigh the needs of the problem with the other constraints of the project. For example, a flash-ROM card as used in some game consoles has little need for an advanced query interface or support for attributes. Reliability of data writes to the medium, however, are critical, and so a file system that supports journaling may be a requirement. Likewise, a file system for a high-end mainframe computer needs extremely fast throughput in many areas but little in the way of user-friendly features, and so techniques that enable more transactions per second would gain favor over those that make it easier for a user to locate obscure files. [18]

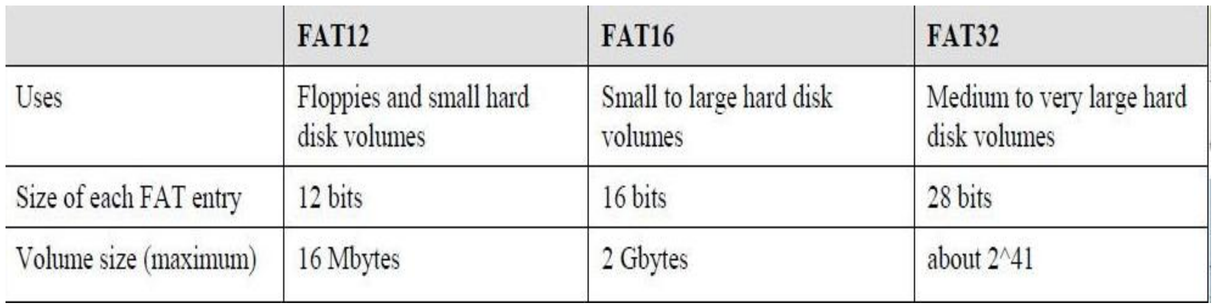
There are many different types of file systems such as: FAT16/FAT32, NTFS, etc. or creating your own file system.

3.4.1 FAT

FAT (File Allocation Table) is a partially patented file system developed by Microsoft for MS-DOS®. It involves the use of a table that centralizes the information about the areas belonging to files. This information helps to identify which areas are free or usable, and where each file is stored on the disk. It also identifies the areas that are unusable. To reduce the management complexity, disk space is allocated to files in contiguous groups of hardware sectors called as clusters. The maximum number of clusters possible that can be addressed by a file system depends on the number of bits being used to identify a cluster in FAT.

Currently, there are three types of FAT file systems: FAT12, FAT16, and FAT32. The basic difference between them is the number of bits in the FAT entries on a disk. FAT12 uses 12-bit entries, FAT16 uses 16-bit entries, and FAT32 uses 28-bit entries. The FAT standard has also evolved in several ways, preserving backward compatibility with existing software. Table 3.11 shows basic differences between the different FAT file systems. [20][21]

Table 3.3: Differences between different FAT file systems



	FAT12	**FAT16**	**FAT32**
Uses	Floppies and small hard disk volumes	Small to large hard disk volumes	Medium to very large hard disk volumes
Size of each FAT entry	12 bits	16 bits	28 bits
Volume size (maximum)	16 Mbytes	2 Gbytes	about 2^41

FAT is still a normal file system for removable media (with the exception of CDs and DVDs), with FAT12 used on floppies and FAT16 used on most other removable media (such as flash memory cards for digital cameras, mobile phones, and USB flash drives). Most removable media are not yet large enough to benefit from FAT32, although some larger flash drives such as Secure Digital High-Capacity (SDHC) cards use it. FAT16 is used on these drives for reasons of compatibility and size overhead.

There are many commands such as: listing in current directory files and folders, creating a directory, copying a file, reading a file, searching a file, navigating through sub-folders in current directory, or deleting a file or directory.

3.4.2 Creating Your Own File System

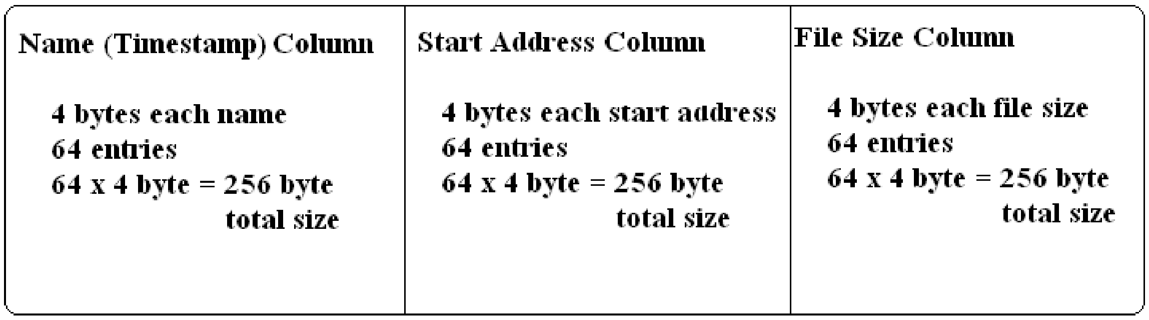
If a programmer creates its own file system, the reason to do so could be

- Known file systems are not suited well to the programmer's structure.
- Known file systems can be very large and needs too many disk space.
- Known file systems are very complicated to implement.
- User interactions are not permitted or system is not using external commands which are given by users.

In the voice recording system, file system is not directly reachable by users. It just gets the requests of the users and gives the right output by using UART communication. Recording a file and putting its properties into the file system is done by MCU. In this system, user can not generate its directory or file, he just requests a file and receive it. Because of these reasons, we generate our own file system. In our system, there is a file system table and records (files). In file system table there are three columns: first one is name column (also called as timestamp because system uses timestamp as file name), the second one is start address column which gives the start address of a record in external flash and the last one is size column which gives the size of a record.

In file system, name column and size column are kept together and start address column is kept in a different location. File system is implemented in internal flash. The reason to do so is external flash has blocks and pages (each of which is subpart of a block), each page can be programmed individually but system can only do four write attempts, then in order to do fifth attempt the whole block need to be deleted. The size of a block is very big and RAM of MCU is not enough to keep all data of a block. If we choose to use a small part of a block then whole remaining part of the block is wasted. In internal flash, we have four different chunks (big buffer) each of has 512 byte. We wrote encoded samples to internal flash. If size of samples in a chunk reaches 512 byte then we write new encoded samples to next chunk. If all four chunks are full of encoded samples which mean that we have 2048 byte encoded sample then we write 2048 byte to external flash as its page and empty big buffer.

The general structure of file system is shown in Figure 3.10:



Name (Timestamp) Column	Start Address Column	File Size Column
4 bytes each name 64 entries 64 x 4 byte = 256 byte total size	4 bytes each start address 64 entries 64 x 4 byte = 256 byte total size	4 bytes each file size 64 entries 64 x 4 byte = 256 byte total size

Figure 3.10: General structure of our own file system

Each column has 256 byte total size. As we mentioned above, name column and file size column are kept together because they are sent together to PC application but start address column is different, there is no reason to send it to user because user does not use this information. If there is a file request user chooses file name and send it to MCU, then MCU finds corresponding start address of file name and sends the requested file to user. Also system has just 1 KB RAM and there is not enough to space to keep all values if table is bigger so system internally restricts the size of file system as 256 bytes for each column.

The timestamp of the system is kept as minute according to system internal epoch 01.01.2009 (DD.MM.YYYY) 00:00 (HH:MM) but PC application converts the minutes which is based on our epoch to a human readable way like DD.MM.YYYY HH:MM.

Voice recoding is a sequential process so in external flash we write samples as page one after another. Each record can be downloaded to PC by itself and downloaded record is not deleted in external flash. User can download whatever he wants but when he wants to delete then whole table must be deleted because file allocation system is implemented to do it so there is no segmentation in external flash and all records are kept as one after another.

3.5 Sampling Theorem

In our lives, continues information keeps to many space and time. Maybe the receiver cannot handle too many information at a time or the required bandwidth to do this process is not always maintained. To keep communication alive and handle all other problems system does not want to process all information, it just wants some part of the information this information is called as sample and getting samples is known as sampling so sampling means picking some values of a continues signal and keeping the value of continues signal as discrete quantities. In order to keep information related, sampling must be done in a given time period at a determined number of samples over a continues signal.

Continues signals are frequency limited waveforms. If we are sampling a continues bandwidth limited waveform, in order to reproduce it we need to follow a theorem which is called Sampling Theorem. Sampling Theorem says that if a waveform is bandwidth-limited to W Hz, then it can be represented exactly by its samples if they are taken at a rate of 2W samples/second or more. If two times of frequency is bigger than sampling rate then there are some aliasing frequencies and information is corrupted. In the world of video, several forms of visual aliasing are possible. For example, temporal aliasing occurs where the wheel of a car seems to turn backwards. [22]

The Nyquist-Shannon Sampling Theorem is a fundamental tenet in the field of information theory, in particular telecommunications. This theorem states that, when converting from an analog signal to digital(or otherwise sampling a signal at discrete intervals), the sampling frequency must be greater than twice the highest frequency of the input signal in order to be able to reconstruct the original perfectly from the sampled version. If the sampling frequency is less than this limit, then frequencies in the original signal that are above half the sampling rate will be “aliased” and will appear in the resulting signal as lower frequencies. The theorem was first formulated by Harry Nyquist in 1928 ("Certain topics in telegraph transmission theory" reference book), but was only formally proved by Claude E. Shannon in 1949 ("Communication in the presence of noise" reference book). Mathematically, the theorem is formulated as a statement about the Fourier Transform. [23]

In digital world, every sampling needed project is based on this theory. As an example, the sampling rate used for CDs nowadays is 44,100 samples per second. That means when you play a CD, the speakers in your stereo system are moved to a new position 44,100 times per second, or once every 23 microseconds. Controlling a speaker this fast enables it to generate any sound in the human hearing range because we cannot hear frequencies higher than around 20,000 cycles per second, and a sampling rate more than twice the highest frequency in the sound guarantees that exact reconstruction is possible from the samples. [24]

In voice recorder project, the Nyquist-Shannon sampling theorem is also used. Sampling frequency is 8 KHz, voices are passed through a low pass filter and their frequencies are between 338.5 Hz to 3000 Hz. Sampling rate is bigger than twice of the highest frequency.

3.6 Quantization

A digital audio signal is usually represented by uniformly sampled values with a fixed word length N, which means that each sample can have a value between – (2^{N-1}) and (2^{N-1}-1). The digital sample value represents the signal amplitude at a specified instant (the sample instant) as shown in figure 2. The number of samples per second is specified by the sampling frequency f_S. This technique is called linear quantization or LPCM (Linear Pulse Code Modulation).

3.7 Coding

3.7.1 PCM

Pulse-code modulation (PCM) is a digital representation of an analog signal where the magnitude of the signal is sampled regularly at uniform intervals, then quantized to a series of symbols in a numeric (usually binary) code. PCM has been used in digital telephone systems and 1980s-era electronic musical keyboards. It is also the standard form for digital audio in computers and the compact disk red books format. It is also standard in digital video, for example, using ITU-R BT-601. Uncompressed PCM is not typically used for video in standard definition consumer applications such as DVD or DVR because the bit rate required is far too high. [25][26][27]

3.7.1.1 Modulation

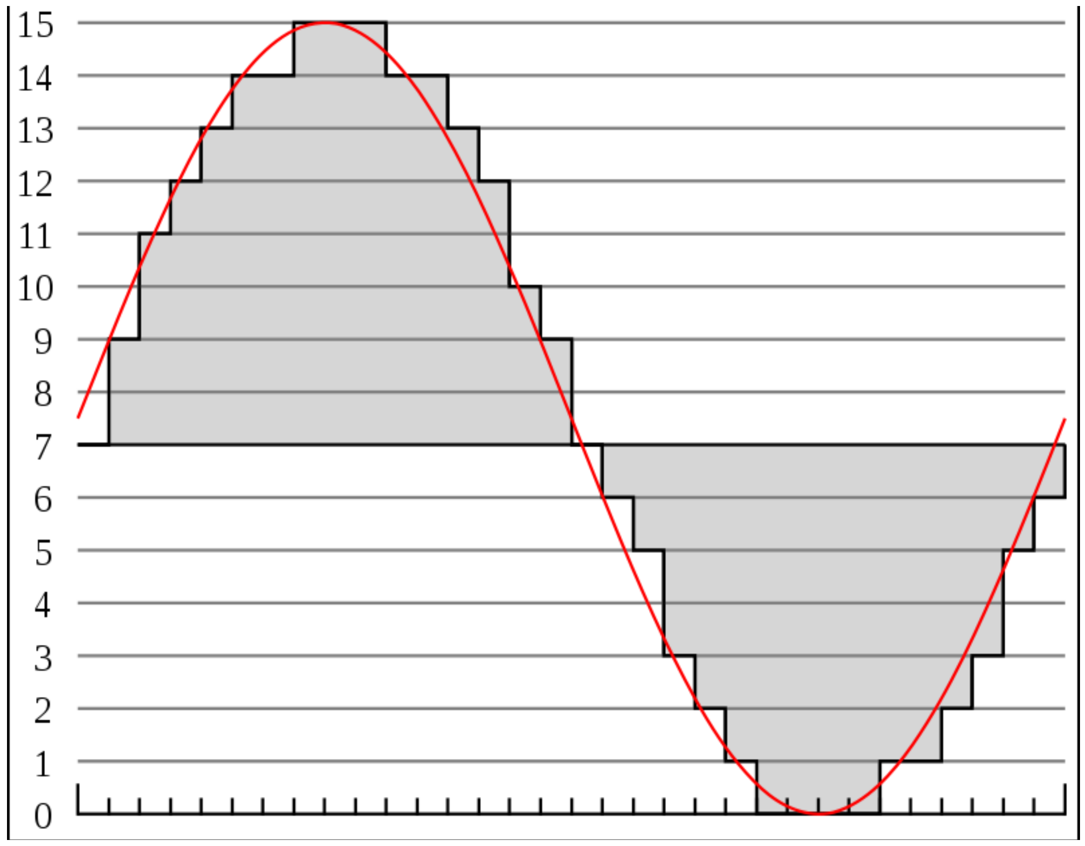


Figure 3.11: Sampling and quantization of a sine wave for 4-bit PCM.

In Figure 3.11, a sin wave (red curve) is sampled and quantized for PCM. The sine wave is sampled at regular intervals, shown as ticks on the x-axis. For each sample, one of the available values (ticks on the y-axis) is chosen by some algorithm (in this case, the floor function is used). This produces a fully discrete representation of the input signal (shaded area) that can be easily encoded as digital data for storage or manipulation. For the sine wave example at right, we can verify that the quantized values at the sampling moments are 7, 9, 11, 12, 13, 14, 14, 15, 15, 15, 14, etc. Encoding these values as binary numbers would result in the following set of nibbles: 0111, 1001, 1011, 1100, 1101, 1110, 1110, 1111, 1111, 1111, 1110, etc. These digital values could then be further processed or analyzed by a purpose-specific digital signal processor or general purpose CPU. Several Pulse Code Modulation streams could also be multiplexed into a larger aggregate data stream, generally for transmission of multiple streams over a single physical link. This technique is called time division multiplexing, or TDM, and is widely used, notably in the modern public telephone system.

3.7.1.2 Demodulation

To produce output from the sampled data, the procedure of modulation is applied in reverse. After each sampling period has passed, the next value is read and a signal is shifted to the new value. As a result of these transitions, the signal will have a significant amount of high-frequency energy. To smooth out the signal and remove these undesirable aliasing frequencies, the signal would be passed through analog filters that suppress energy outside the expected frequency range (that is, greater than the Nyquist frequency $f_s / 2$). Some systems use digital filtering to remove some of the aliasing, converting the signal from digital to analog at a higher sample rate such that the analog filter required for anti alising is much simpler. In some systems, no explicit filtering is done at all; as it's impossible for any system to reproduce a signal with infinite bandwidth, inherent losses in the system compensate for the artifacts — or the system simply does not require much precision. The sampling theorem suggests that practical PCM devices, provided a sampling frequency that is sufficiently greater than that of the input signal, can operate without introducing significant distortions within their designed frequency bands.

3.7.2 DPCM

Differential pulse code modulation (DPCM) is a procedure of converting an analog into a digital signal in which an analog signal is sampled and then the difference between the actual sample value and its predicted value (predicted value is based on previous sample or samples) is quantized and then encoded forming a digital value.

DPCM code words represent differences between samples unlike PCM where code words represented a sample value.

Basic concept of DPCM - coding a difference, is based on the fact that most source signals show significant correlation between successive samples so encoding uses redundancy in sample values which implies lower bit rate.

Realization of basic concept (described above) is based on a technique in which we have to predict current sample value based upon previous samples (or sample) and we have to encode the difference between actual value of sample and predicted value (the difference between samples can be interpreted as prediction error).
Because it's necessary to predict sample value DPCM is form of predictive coding.

DPCM compression depends on the prediction technique, well-conducted prediction techniques lead to good compression rates, in other cases DPCM could mean expansion comparing to regular PCM encoding. [28]

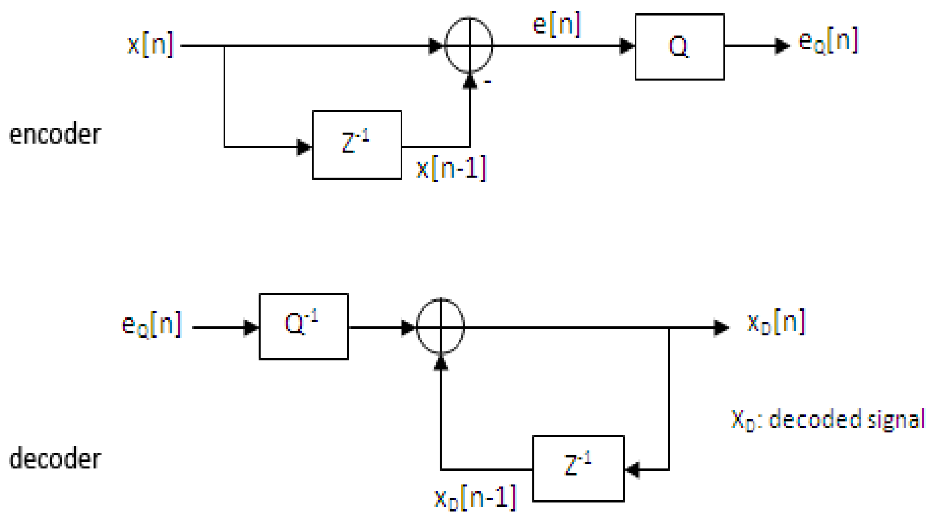


Figure 3.12: Encode and decode mechanisms of DPCM

3.7.3 ADPCM

Adaptive DPCM (ADPCM) is a variant of DPCM (differential pulse-code modulation) that varies the size of the quantization step, to allow further reduction of the required bandwidth for a given signal to noise ratio. Typically, the adaptation to signal statistics in ADPCM consists simply of an adaptive scale factor before quantizing the difference in the DPCM encoder.

ADPCM was developed in the early 1970s at Bell Labs for voice coding, by P. Cummiskey, N. S. Jayant and James L. Flaganan.

In telephony a standard audio signal for a single phone call is encoded as 8000 analog samples per second, of 8 bits each, giving a 64 kbit/s digital signal known as DS0. The default signal compression encoding on a DS0 is either mu-law PCM (North America and Japan) or A law PCM (Europe and most of the rest of the world). These are logarithmic compression systems where a 12 or 13 bit linear PCM sample number is mapped into an 8 bit value. This system is described by international standard G.711. Where circuit costs are high and loss of voice quality is acceptable, it sometimes makes sense to compress the voice signal even further. An ADPCM algorithm is used to map a series of 8 bit u-law (or a-law) PCM samples into a series of 4 bit ADPCM samples. In this way, the capacity of the line is doubled. The technique is detailed in the G.726 standard. [29]

The Interactive Multimedia Association (IMA) recommended a standard for ADPCM-codec in multimedia applications, known as IMA or DVI ADPCM, in the early 1990s. This algorithm is now used in most cross-platform ADPCM-based audio applications. The proposed IMA-standard, now widely used in multimedia applications, uses an adaptive quantizer, but a fixed predictor to limit the computational complexity. The predictor is identical to the one in the DPCM. The compression level is 4:1, which means the 16-bit original signal is quantized to 4-bits. The stepsize of the quantization depends on the input signal, thus making it adaptive.

ADPCM encoder and decoder mechanisms are shown in Figure 3.13:

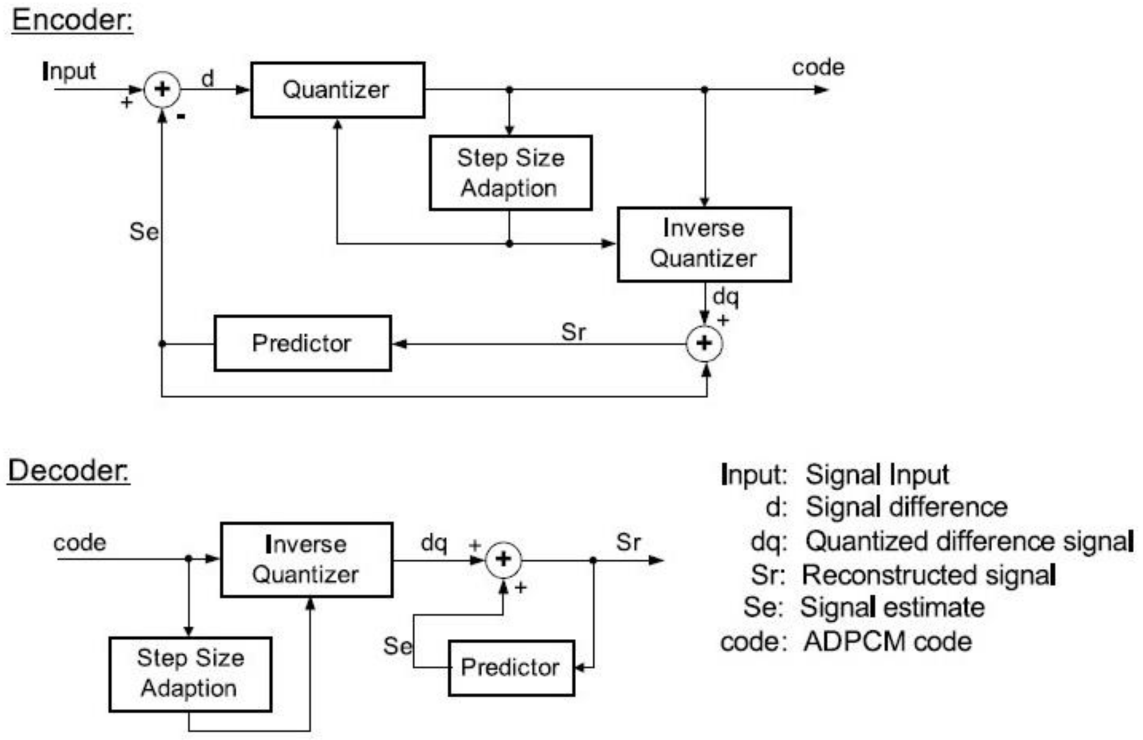


Figure 3.13: Encoder and decoder mechanisms of ADPCM

In this project, sampling rate is used as 8 KHz, quantization is done over 10 bit and used coding is ADPCM IMA.

Sampling, quantization and coding is shown in Figure 3.14:

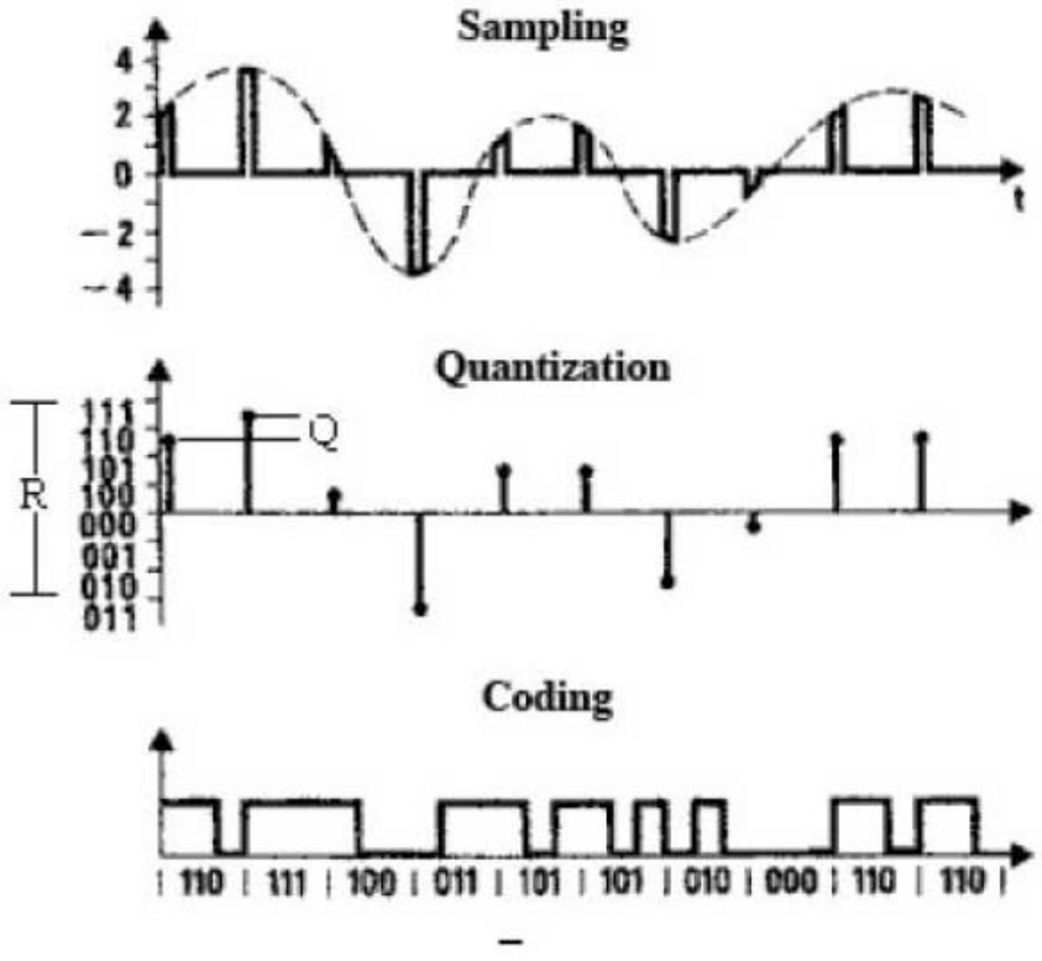


Figure 3.14: General perspectives of sampling, quantization and coding

3.8 Microphone Circuitry

A microphone, sometimes referred to as a mic or mike, is an acoustic-to-electric transducer or sensor that converts sound into an electrical signal. Microphones are used in many applications such as telephones, tape recorders, hearing aids, motion picture production, live and recorded audio engineering, in radio and television broadcasting and in computers for recording voice, VOIP, and for non-acoustic purposes such as ultrasonic checking. There many different types of microphone systems such as dynamic microphones, carbon microphones, laser microphones, etc. [30]

Carbon microphone is a capsule containing carbon granules pressed between two metal plates. A voltage is applied to across metal plates. One of the plates is diaphragm; sound waves vibrate the diaphragm and apply a varying pressure to carbon. Carbon granules are deformed. The distances between granules are changed and this changes the resistance of system. The electrical resistance is changed so this produces electrical signal.

Carbon microphones are very cheap systems and easily found in everywhere. It is also used as an amplifier so I choose carbon microphone in my project.

Human can hear frequencies between 20 Hz and 20 KHz. Human usually use frequencies between 300 Hz to 3500 Hz. To reduce the noise and have good quality of recording , frequencies which are out of this range must be suppressed. In order to suppress the frequencies I need to use filter.

Filter is a system which suppresses higher or lower frequencies than the cutoff frequency of the system. There two basic types of filter: the first one is low pass filter which suppresses the frequencies higher than cutoff frequency and the second one is high pass filter which suppresses the frequencies lower than cutoff frequency. I need to use a high and a low pass filter. The combination of this filter is called as bandpass filter. A bandpass filter is a filter which has two different filters: low pass and high pass filter. The bandpass filter suppresses the frequencies higher than the cutoff frequency of the low pass filter and also suppresses the frequencies lower than the cutoff frequency of the high pass filter.

Calculation of cut of frequency of a simple filter has the formula of:

$$\text{Frequency} = 1 / (2 x \pi x C x R)$$

R : resistance value of resistor in ohm

C : capacitance value of capacitor in F.

Human can hear frequencies between 20 Hz to 20 KHz. Used frequencies, while we are talking, are between 300 Hz to 3 KHz. I try to suppress the other frequencies by using a bandpass filter.

Our microphone circuitry is shown in below Figure 3.15:

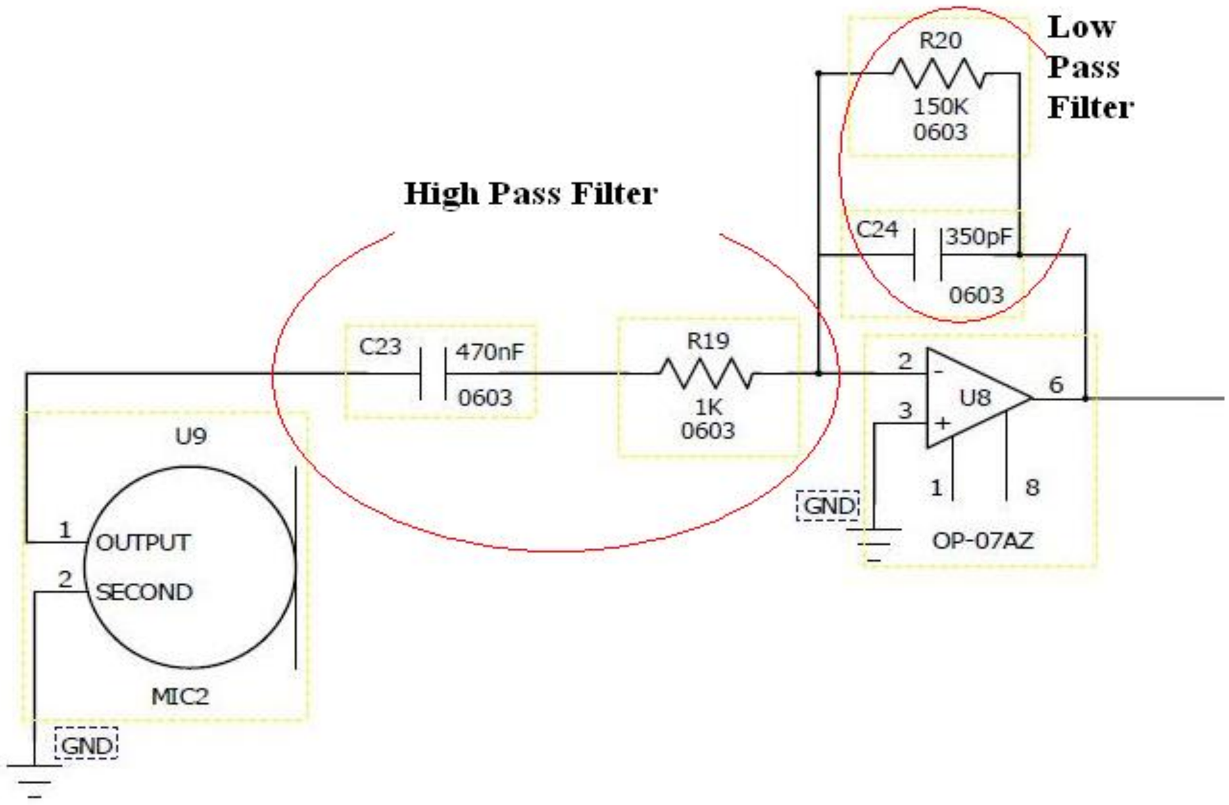


Figure 3.15: Microphone circuitry of the project

The high pass filter has two elements. First one is the capacitor has the value of 470 nF and second one is the resistor has the value of 1 KOhm.

The cutoff frequency of high pass filter is:

Frequency = $1/(2x\pi xCxR)$

R : 1 Kohm

C : 470 nF

Frequency = 338.5 Hz

The cutoff frequency of low pass filter is:

Frequency = Frequency = $1/(2x\pi xCxR)$

R : 150 Kohm

C : 350 pF

Frequency = 3030.3 Hz .

My bandpass filter passes frequencies between 338.5 Hz and 3030.3 Hz.

3.9 Voice Recording Mechanism

Our system recording mechanism is a digital recording mechanism. Digital recording is converting the physical properties of the original sound into a sequence of numbers. Accuracy of recording depends on two things

1. Sampling Rate

   Sampling rate means how many sample we have in one second.

2. Sampling Depth

   Sampling depth means how many bits we use in order to show original sound of data in digital world.

Our system sampling rate is 8192 Hz and sampling depth (resolution) is also 10 bit. Voice coding mechanism, is also mentioned before, is ADPCM encoding which is 32 kbps, requires low computation and gives acceptable quality for voice.

Voice recording mechanism is shown in Figure 3.16:

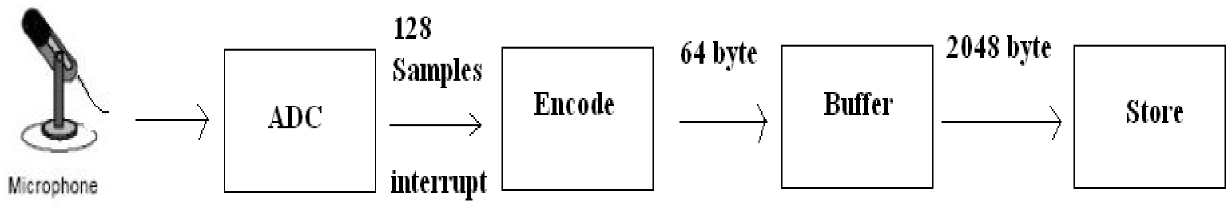


Figure 3.16: Voice recording mechanism

Microphone circuitry catches voice and sends it to ADC. ADC gets these samples and puts them into a small buffer. If buffer gets 128 samples then it gives an interrupt. Interrupt is taken by MCU and MCU sends the small buffer to encoding. ADPCM encoding gets buffer, processes it and gives the output as 64 byte to a big buffer which is located in internal flash. Big buffer keeps these encoded samples. If the big buffer's size reaches 2048 byte then these samples are written to external flash so we write every 2048 byte to external flash because a page in external flash is 2048 byte and it is the smallest unit which can be programmed at once in external flash.

A page is 2048 byte and it keeps 4096 samples in it. Our sampling rate is 8192 sample so a page keeps a 0.5 second voice record in it.

Each record has a size field in file table. Size field is 4 byte which is equal to 32 bit. Size filed of a record keeps how may pages it has in external flash so a record may have size of 2^{32} pages.

Maximum size of a record = 2^{32} pages x 2048 byte/page = 2^{43} bytes.

Maximum record time of a record = 2^{32} pages x 0.5 second/page $\cong$ 2^{31} second.

Our external flash's size is 2 Gbit (256 MB). Maximum record time of a record in our external flash is equal to equation1.

Equation1 = 256 MB x 1 second / 4KB x 1/60 minute/second x 1/60 hour/minute
$\cong$ 18.2 hours.

3.10 Wav Format

A waveform audio file, also known as a wave file, or simply .wav after its extension, is a common type of sound file. Microsoft and IBM introduced the wav file in 1991 for use on the Microsoft Windows 3.1 operation system (OS). It is the main format used on windows systems for raw and typically uncompressed audio. The usual bitstream encoding is the Pulse Code Modulation(PCM) format.

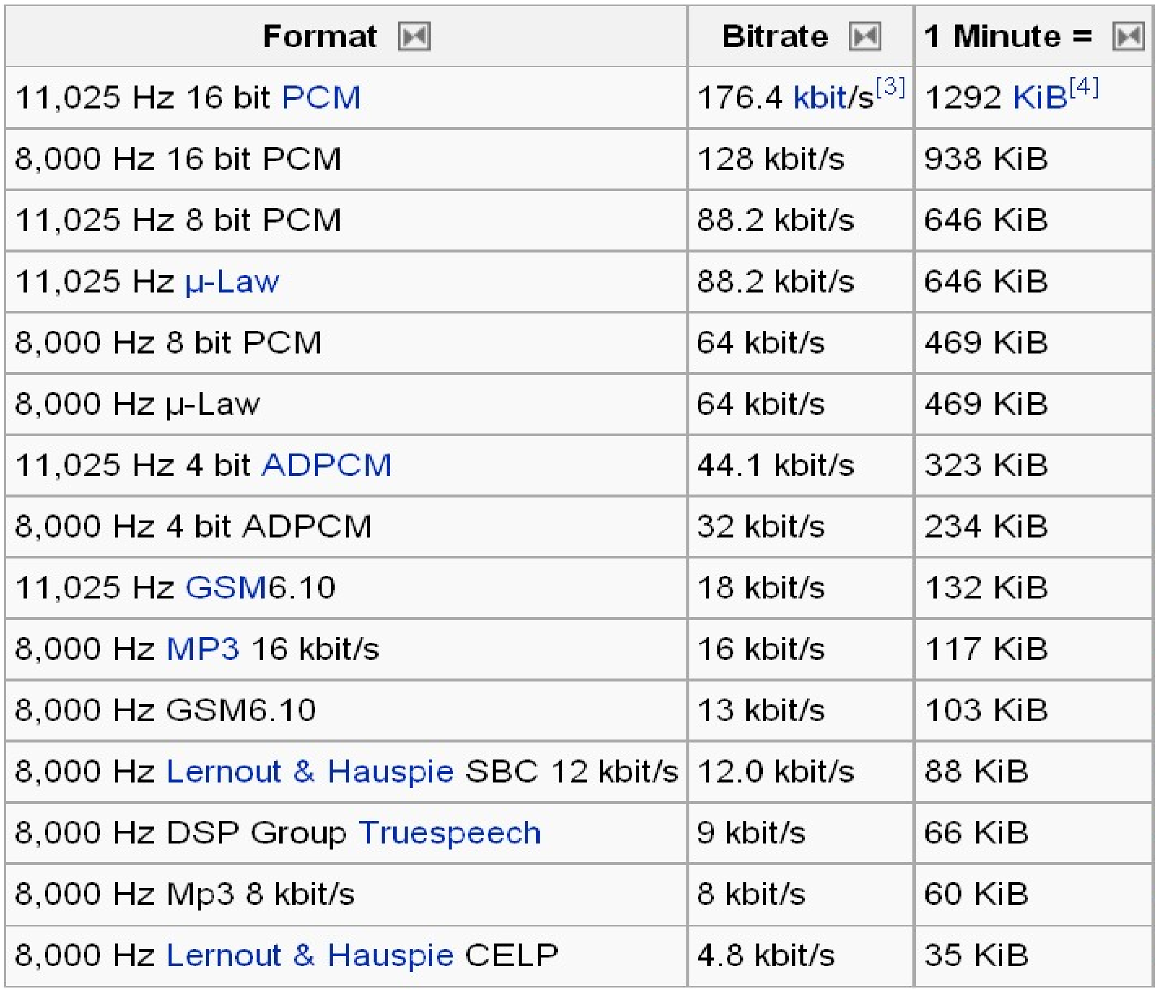
The wav format is based on the Resource Interchange File Format (RIFF), which stores audio files in indexed “chunks” and “sub-chunks.” RIFF is in turn based on the earlier Interchange File Format (IFF), established by Electronic Arts in 1985 for use in electronic gaming. Apple’s version, known as Audio Interchange File Format (AIFF), was released in 1988 for Macintosh computers. Due to the common roots of these various audio formats, however, the audio files will play on any computer system, IBM or Apple.

The wav file had two very big things going for it when introduced. Firstly, it could digitize sounds 100% faithful to the original source because it is a lossless format. “Lossless” means that the wav file format does not compromise audio quality even when it holds compressed data. Secondly, the wav file is very easy to edit and manipulate with software. Luckily for audiophiles, free wav file editing software has been available nearly as long as wav files themselves. A WAV file can hold compressed audio, the most common WAV format contains uncompressed audio in the linear pulse code modulation (LPCM) format. The standard audio file format for CDs, for example, is LPCM-encoded, containing two channels of 44,100 samples per second, 16 bits per sample. Since LPCM uses an uncompressed, lossless storage method, which keeps all the samples of an audio track, professional users or audio experts may use the WAV format for maximum audio quality. [31]

Wav files can be encoded with a variety of codecs to reduce the file size (for example the GSM or MP3 codecs). Below picture explains the format, bitrate and one minute record size of a wav file.

Well known wav audio formats are shown in Table 3.4:

Table 3.4: Wav audio formats



Format	Bitrate	1 Minute =
11,025 Hz 16 bit PCM	176.4 kbit/s[3]	1292 KiB[4]
8,000 Hz 16 bit PCM	128 kbit/s	938 KiB
11,025 Hz 8 bit PCM	88.2 kbit/s	646 KiB
11,025 Hz µ-Law	88.2 kbit/s	646 KiB
8,000 Hz 8 bit PCM	64 kbit/s	469 KiB
8,000 Hz µ-Law	64 kbit/s	469 KiB
11,025 Hz 4 bit ADPCM	44.1 kbit/s	323 KiB
8,000 Hz 4 bit ADPCM	32 kbit/s	234 KiB
11,025 Hz GSM6.10	18 kbit/s	132 KiB
8,000 Hz MP3 16 kbit/s	16 kbit/s	117 KiB
8,000 Hz GSM6.10	13 kbit/s	103 KiB
8,000 Hz Lernout & Hauspie SBC 12 kbit/s	12.0 kbit/s	88 KiB
8,000 Hz DSP Group Truespeech	9 kbit/s	66 KiB
8,000 Hz Mp3 8 kbit/s	8 kbit/s	60 KiB
8,000 Hz Lernout & Hauspie CELP	4.8 kbit/s	35 KiB

In the project we choose wav format because lossless storage important. In MCU, we can use ADMPC to compress voice samples and wav file format support ADMPC coding. Our sample rate is 8192 Hz and every 10 bit ADC sample is converted into 4 bit sample by ADMPC. The format we chose in the project is in Figure 3.17:

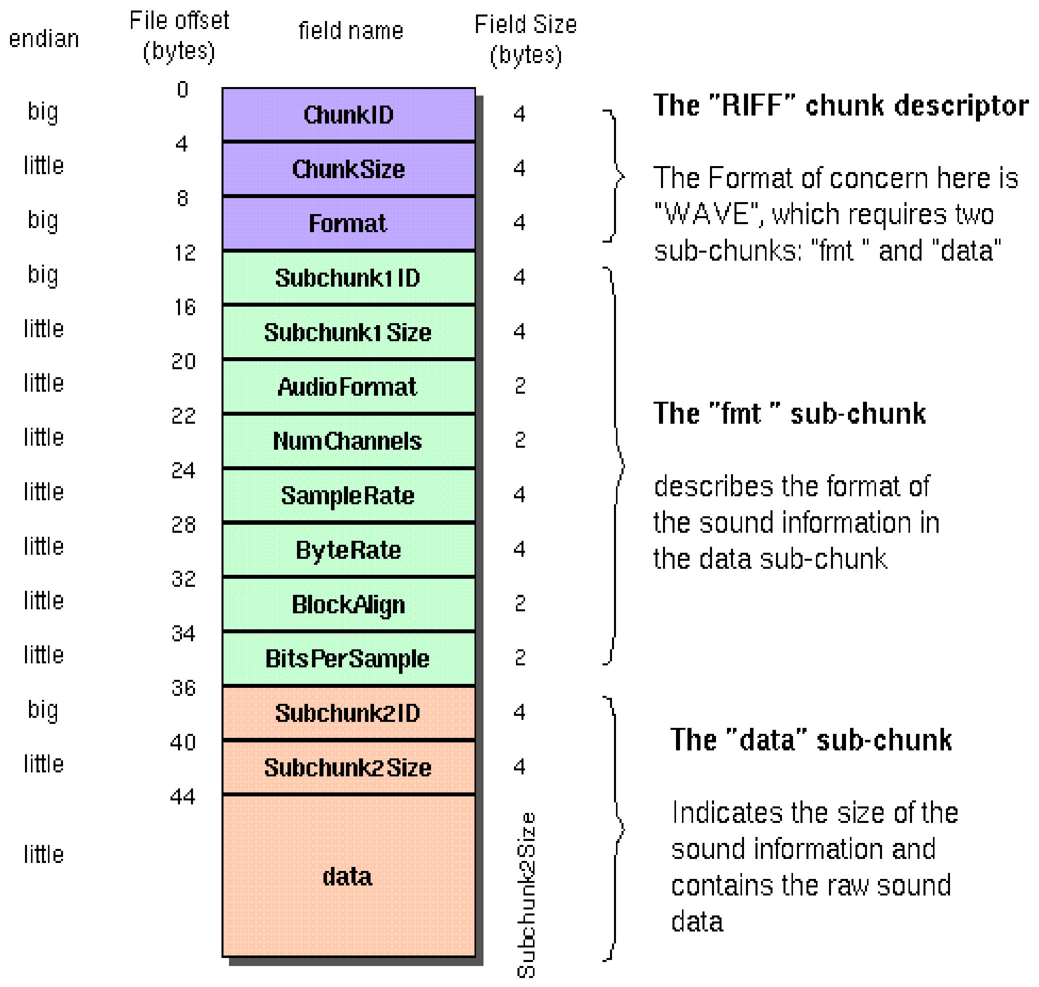


Figure 3.17: General format of wav file

3.11 Communication

A communication protocol is a convention for data transmission that includes such functions as timing, control, formatting, and data presentation. There are two categories depending on the clocking of the data on the serial link: [32]

- Synchronous protocols: each successive datum in a stream of data is governed by a master clock and appears at a specific interval in time.

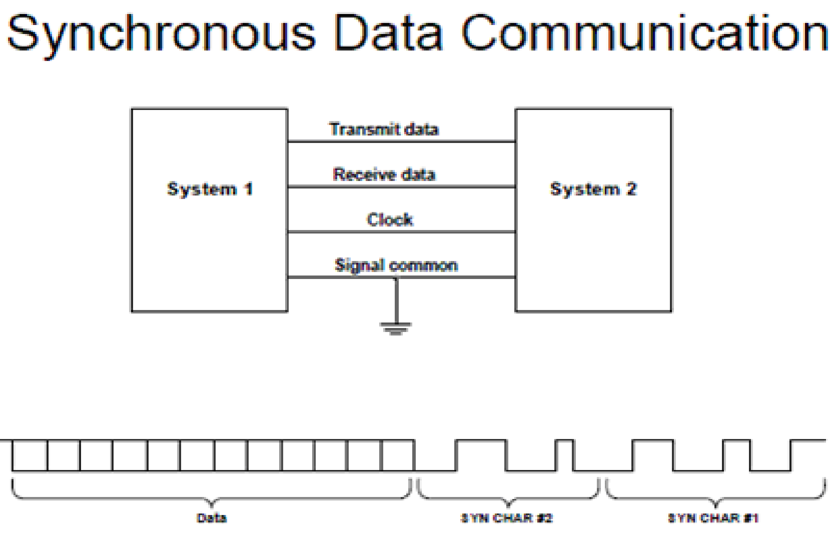


Figure 3.18: Synchronous data transmission format

- Asynchronous protocols: successive data appear in the data stream at arbitrary times, with no specific clock control governing the relative delays between data.

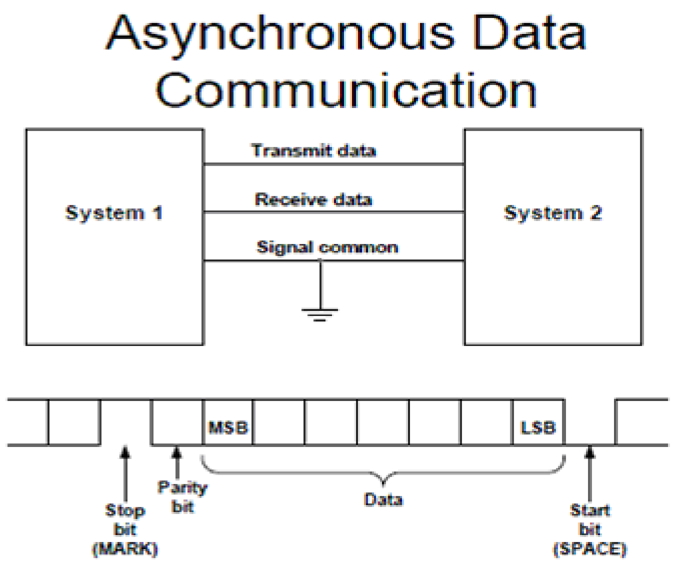


Figure 3.19: Asynchronous data transmission format

There are special IC chips made for serial data communications. These chips are called UART (universal asynchronous receiver-transmitter) and USART (universal synchronous-asynchronous receiver-transmitter).

Synchronous communication is usually used in small distance between two communicating units. Synchronous communication types are SPI, I^2C, etc.

Asynchronous communication is used in large distance and there is no clock need to communicate. Asynchronous communication types are RS-232, etc.

In project, asynchronous communication is chosen with RS-232.

3.11.1 RS-232 and Its Specifications

The EIA/TIA-232-E standard was introduced in 1962 and has since been updated four times to meet the evolving needs of serial communication applications. The letter "E" in the standard's name indicates that this is the fifth revision of the standard. [33]

3.11.1.1 RS-232 Specifications

RS-232 is a complete standard. This means that the standard sets out to ensure compatibility between the host and peripheral systems by specifying:

1. Common voltage and signal levels
2. Common pin-wiring configurations
3. A minimal amount of control information between the host and peripheral systems.

Unlike many standards which simply specify the electrical characteristics of a given interface, RS-232 specifies electrical, functional, and mechanical characteristics to meet the above three criteria. Each of these aspects of the RS-232 standard is discussed below. [33]

3.11.1.1.1 Electrical Characteristics

The electrical characteristics section of the RS-232 standard specifies voltage levels, rate of change for signal levels, and line impedance.

As the original RS-232 standard was defined in 1962 and before the days of TTL logic, it is no surprise that the standard does not use 5V and ground logic levels. Instead, a high level for the driver output is defined as between +5V to +15V, and a low level for the driver output is defined as between -5V and -15V. The receiver logic levels were defined to provide a 2V noise margin. As such, a high level for the receiver is defined as between +3V to +15V, and a low level is between -3V to -15V. Figure 3.20 illustrates the logic levels defined by the RS-232 standard. It is necessary to note that, for RS-232 communication, a low level (-3V to -15V) is defined as a logic 1 and is historically referred to as "marking." Similarly, a high level (+3V to +15V) is defined as a logic 0 and is referred to as "spacing".

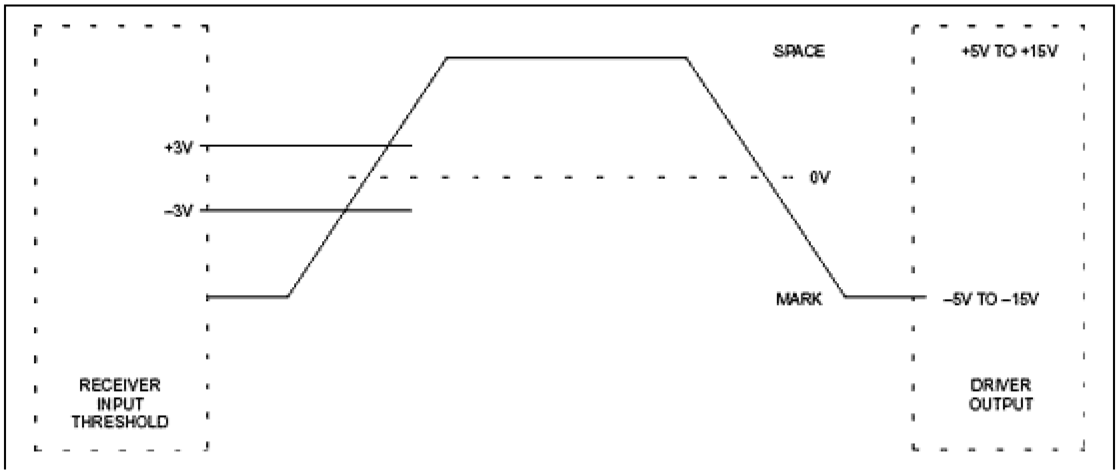
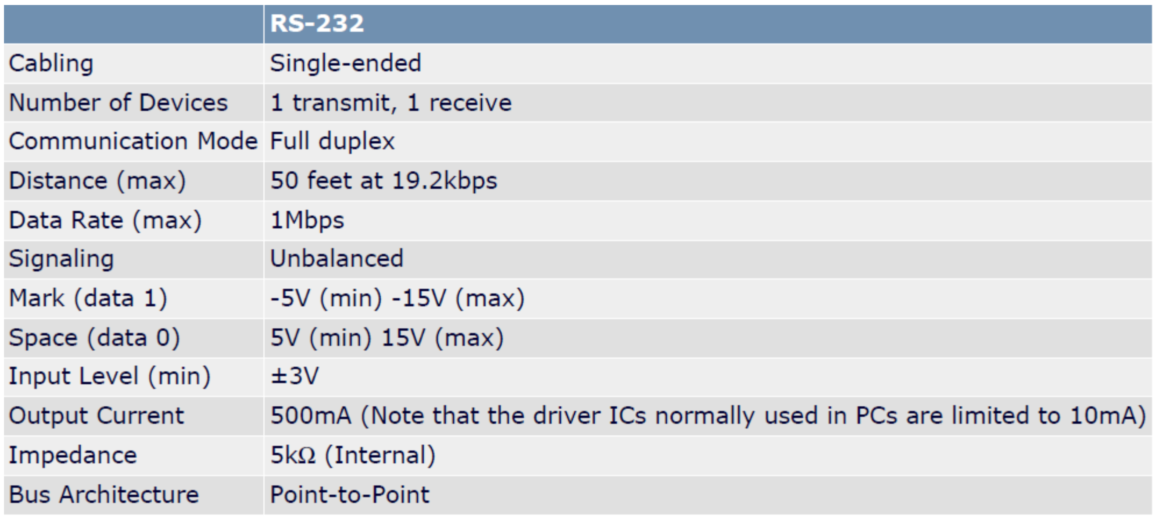


Figure 3.20: Logic levels defined by RS-232 standard

Table 3.5 summarizes the electrical specifications in the current standard.

Table 3.5: Electrical specification of the RS-232 standard

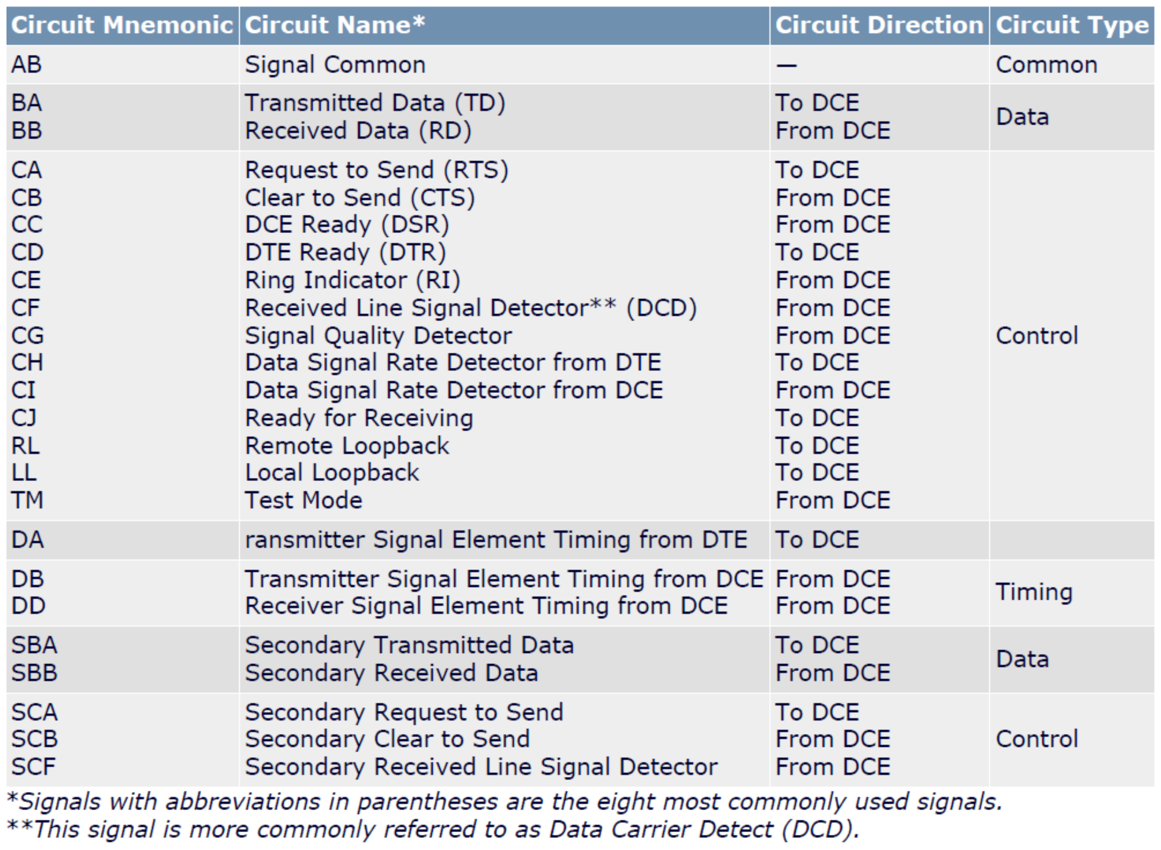


	RS-232
Cabling	Single-ended
Number of Devices	1 transmit, 1 receive
Communication Mode	Full duplex
Distance (max)	50 feet at 19.2kbps
Data Rate (max)	1Mbps
Signaling	Unbalanced
Mark (data 1)	-5V (min) -15V (max)
Space (data 0)	5V (min) 15V (max)
Input Level (min)	±3V
Output Current	500mA (Note that the driver ICs normally used in PCs are limited to 10mA)
Impedance	5kΩ (Internal)
Bus Architecture	Point-to-Point

3.11.1.1.2 Functional Characteristic

The standard also addresses the functional characteristics of the interface which is shown in Table 3.6. This essentially means that RS-232 defines the function of the different signals used in the interface. These signals are divided into four different categories: common, data, control, and timing. See table below. The standard provides abundant control signals and supports a primary and secondary communications channel. Fortunately few applications, if any, require all these defined signals. For example, only eight signals are used for a typical modem.

Table 3.6: Functional specification of the RS-232 standard



Circuit Mnemonic	Circuit Name*	Circuit Direction	Circuit Type
AB	Signal Common	—	Common
BA BB	Transmitted Data (TD) Received Data (RD)	To DCE From DCE	Data
CA CB CC CD CE CF CG CH CI CJ RL LL TM	Request to Send (RTS) Clear to Send (CTS) DCE Ready (DSR) DTE Ready (DTR) Ring Indicator (RI) Received Line Signal Detector** (DCD) Signal Quality Detector Data Signal Rate Detector from DTE Data Signal Rate Detector from DCE Ready for Receiving Remote Loopback Local Loopback Test Mode	To DCE From DCE From DCE To DCE From DCE From DCE From DCE To DCE From DCE To DCE To DCE To DCE From DCE	Control
DA	ransmitter Signal Element Timing from DTE	To DCE	
DB DD	Transmitter Signal Element Timing from DCE Receiver Signal Element Timing from DCE	From DCE From DCE	Timing
SBA SBB	Secondary Transmitted Data Secondary Received Data	To DCE From DCE	Data
SCA SCB SCF	Secondary Request to Send Secondary Clear to Send Secondary Received Line Signal Detector	To DCE From DCE From DCE	Control

**Signals with abbreviations in parentheses are the eight most commonly used signals.*
***This signal is more commonly referred to as Data Carrier Detect (DCD).*

3.11.1.1.3 Mechanical Interface Characteristics

The third area covered by RS-232 is the mechanical interface. Specifically, RS-232 specifies a 25-pin connector as the minimum connector size that can accommodate all the signals defined in the functional portion of the standard. The pin assignment for this connector is shown in Figure 3.14. The connector for DCE equipment is male for the connector housing and female for the connection pins. Likewise, the DTE connector is a female housing with male connection pins. Although RS-232 specifies a 25-position connector, this connector is often not used. Most applications do not require all the defined signals, so a 25-pin connector is larger than necessary. Consequently, other connector types are commonly used. Perhaps the most popular connector is the 9-position DB9S connector, also illustrated in Figure 3.21. This

9-position connector provides, for example, the means to transmit and receive the necessary signals for modem applications.

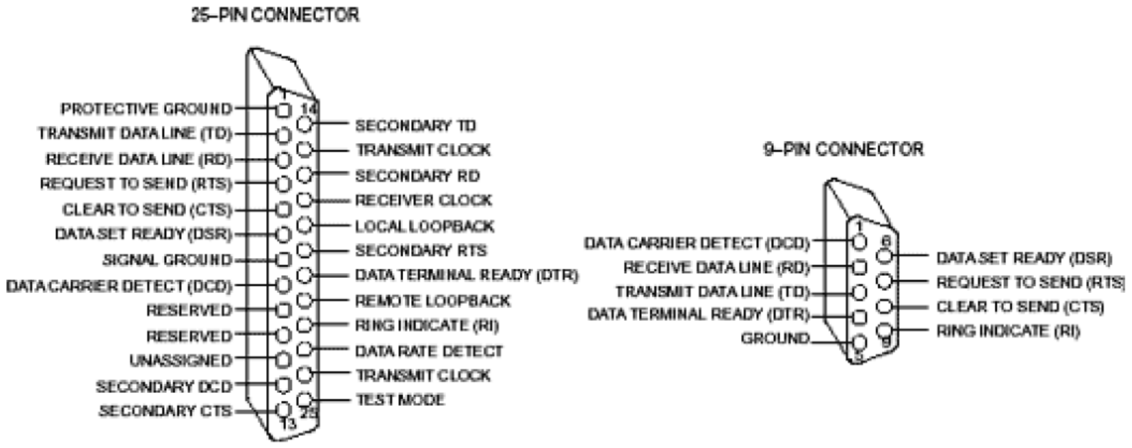


Figure 3.21: Mechanical interfaces of RS-232 standard

3.11.2 Communication Principles of Flash Memory Based Voice Recorder System

In every communication there is standard between communication devices such as baud rate, data bits, parity bit, stop bit and communication port. Communication port is chosen by user. Baud rate is automatically set to 115200. Data bits are set to 8 bits. Parity bit is used as none. Stop bit is used as one stop bit. In order to guaranty the reliability of communication and reliable data transfer, Fletcher16 is used as checksum algorithm and calculated on the given size of data.

3.11.2.1 Communication Commands

There are five different communication commands between MCU and PC application. These are file system table request, file request, setting date of MCU, getting systems internal time and reset file system.

1. File System Table Request

This command gets the file system table of the system from internal flash to PC application. First, PC application sends “T” character to MCU. MCU receives “T” character and learn that it must send the file system table to PC application. It sends 256 bytes data, calculates checksum of 256 bytes data and sends 2 bytes checksum. PC application receives 256 byte data, calculates checksum of it and compares the calculated checksum and received checksum if they are equal then PC application sends “C” character as continue, if they are not equal then MCU sends 256 bytes data and 2 bytes checksum until comparison becomes equal. MCU does these operations until all data of file system table is sent.

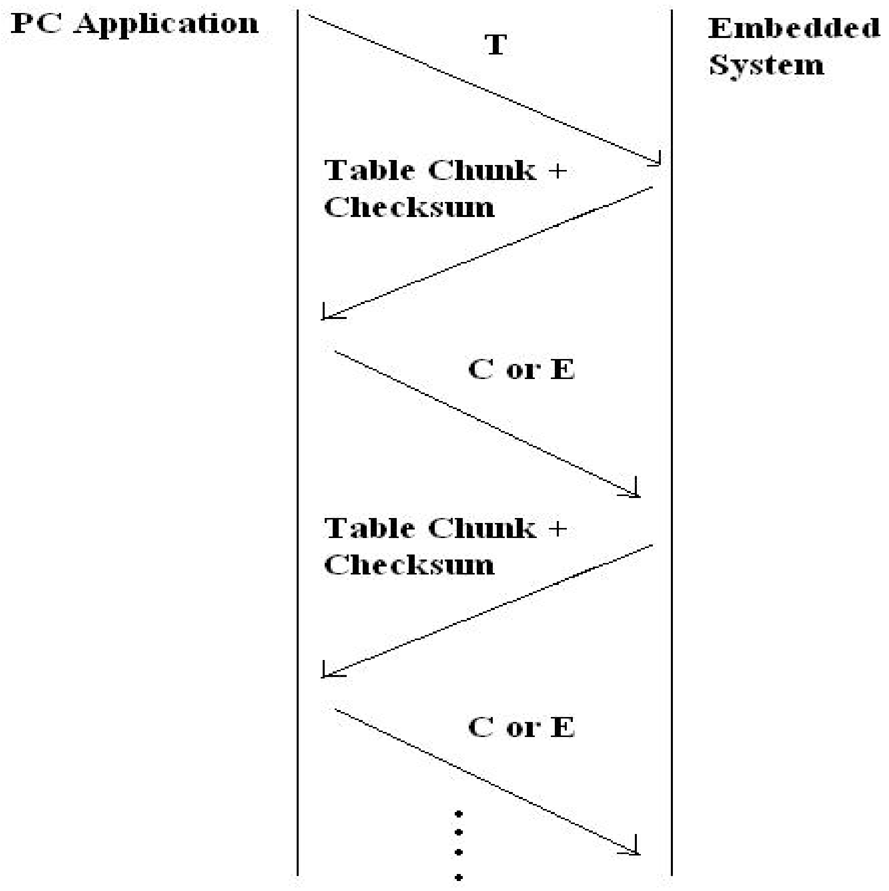


Figure 3.22: File table request

2. File Request

This command gets the requested file from internal flash to PC application. First, PC application sends “F” character to MCU and then sends 4 bytes name (timestamp) of file and its 2 byte checksum. MCU receives “T” character, 4 bytes name of file and 2 bytes checksum of name of file. Then MCU calculates checksum of name of file and compares checksum if they are equal it sends “C” character as continue, if they are not equal then MCU requests timestamp one more time until comparison becomes equal. After they become equal, MCU sends 256 bytes data of requested file, calculates checksum of 256 bytes data and sends 2 bytes checksum. PC application receives 256 byte data, calculates checksum of it and compares the calculated checksum and received checksum if it is equal then PC application sends “C” character as continue, if it is not equal then MCU sends 256 bytes data and 2 bytes checksum until comparison becomes equal. MCU does these operations until all data of file system table is sent.

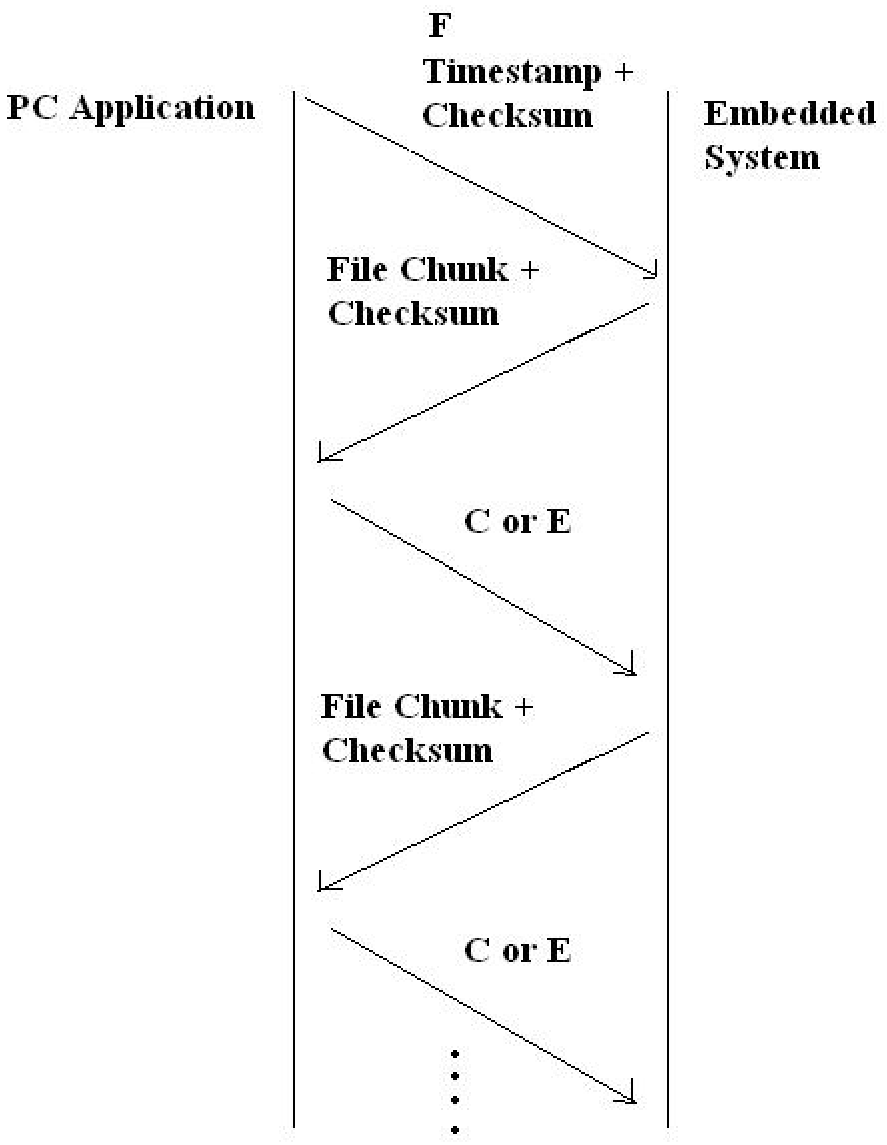


Figure 3.23: File request

3. Set the Time of MCU

This command sets the time of MCU, time is given by user as format DD.MM.YYYY HH:MM according to systems internal epoch 01.01.2009 00:00. First, PC application sends "D" character to MCU and then sends 4 bytes timestamp, which is calculated as minutes based on epoch of the system, and its 2 byte checksum. MCU receives "D" character, 4 bytes timestamp and 2 bytes checksum of timestamp. Then MCU calculates checksum of timestamp and compares checksum if they are equal it sends “C” character as continue, if they are not equal then MCU requests timestamp and checksum one more time until comparison becomes equal. If they become equal, MCU sets its internal time.

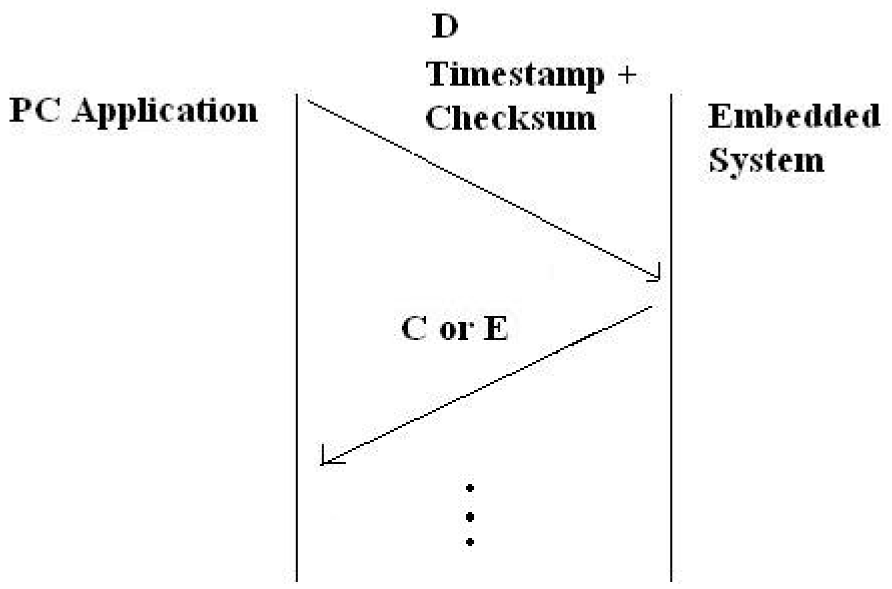


Figure 3.24: Setting time of MSP430

4. Get System Internal Time

This command gets system internal time. First, PC application sends “S” character to MCU. MCU receives “S” and then sends 4 bytes timestamp and its 2 byte checksum. MCU receives “D” character, 4 bytes timestamp and 2 bytes checksum of timestamp. Then MCU calculates checksum of timestamp and compares checksum if they are equal PC application sends “C” character as continue, if they are not equal then MCU sends timestamp and checksum one more time until comparison becomes equal. If they become equal, PC application calculates real time based on epoch and shows it on the screen.

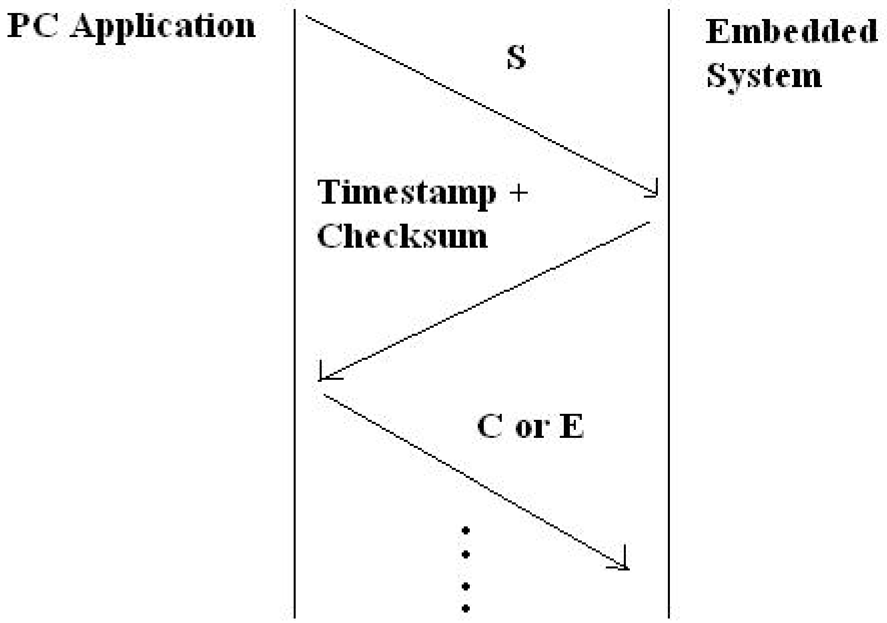


Figure 3.25: Getting system internal time

5. Reset File System

This command resets file system. First, PC application sends “R” character, “ESET” as 4 bytes and 2 bytes checksum of “ESET” to MCU. MCU receives “R”, 4 bytes “ESET and “ESET”’s 2 bytes checksum. Then MCU calculates checksum of “ESET” and compares checksum if they are equal PC application sends “C” character as continue, if they are not equal then MCU requests “ESET and its checksum one more time until comparison becomes equal. If they become equal, MCU resets file system.

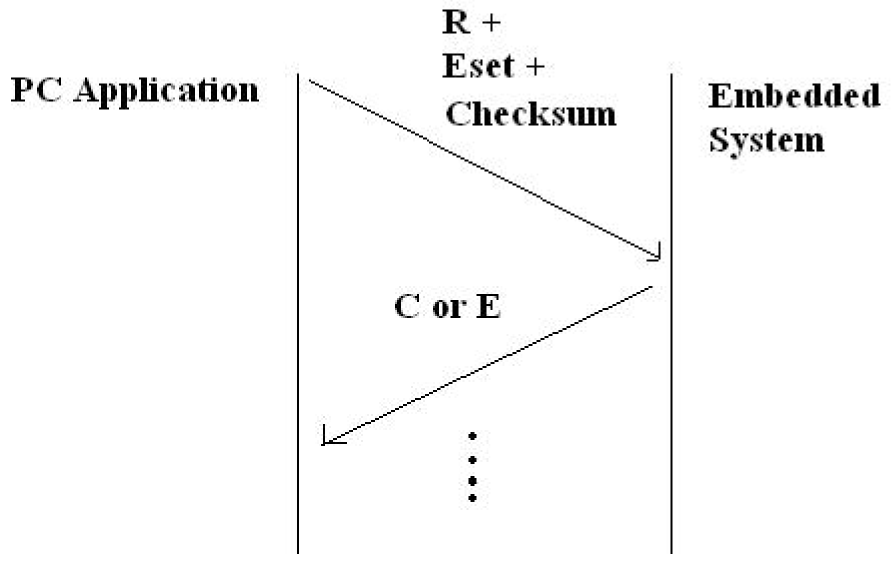


Figure 3.26: Resetting file system

3.11.3 Guaranty Reliable Communication: Fletcher’s Checksum Algorithm

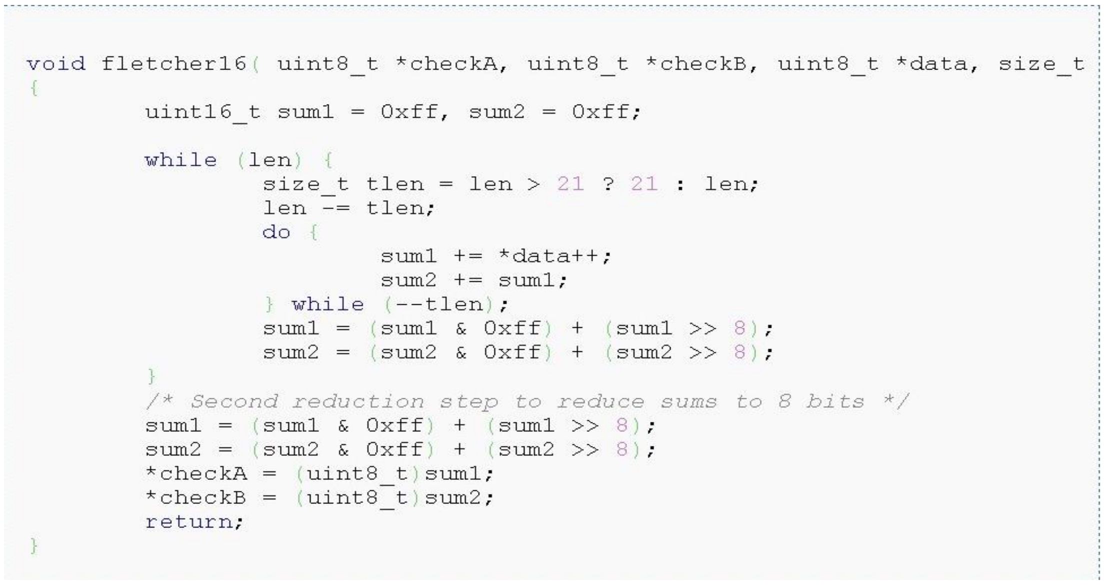
In every communication there can be loss or change in data because of noise that are generated by other factors so there must be a way to check there is a loss or not. The way to do so is using a checksum algorithm which is implemented at both side of the communication. If receiver part of communication gets data as wrong then sender part of communication resends data. By the use of this way, it ensures data integrity and guaranties reliability of connection.

Fletcher's checksum is one of several types of checksum algorithms, which are relatively simple processes used by computers to check the integrity of data.

The implementation of the Fletcher-32 is very similar to the Adler-32 algorithm but several differences should be noted. Fletcher wraps around at modulo 65535 while Adler wraps at the prime 65521. In other words, Fletcher adds overflow bits (16-31) into its sum; while Adler multiplies those bits by 15, and then adds the product into its sum. Fletcher-32 works on 16 bit data while Adler works on 8 bit data. [34][35]

It is designed to overcome some of the inadequacies of simply summing all the bytes as in the original checksum. Fletcher's checksum, unlike the original checksum, can detect the inserting/deleting of zero value bytes, the reordering of bytes, and the incrementing and decrementing of bytes in opposite directions. [34]

Fletcher-32 is slightly more reliable than Adler-32. Fletcher-16 is chosen because 8 bits data is used in communication. The C implementation of Fletcher-16 is given in Figure 3.27:



```
void fletcher16( uint8_t *checkA, uint8_t *checkB, uint8_t *data, size_t
{
        uint16_t sum1 = 0xff, sum2 = 0xff;

        while (len) {
                size_t tlen = len > 21 ? 21 : len;
                len -= tlen;
                do {
                        sum1 += *data++;
                        sum2 += sum1;
                } while (--tlen);
                sum1 = (sum1 & 0xff) + (sum1 >> 8);
                sum2 = (sum2 & 0xff) + (sum2 >> 8);
        }
        /* Second reduction step to reduce sums to 8 bits */
        sum1 = (sum1 & 0xff) + (sum1 >> 8);
        sum2 = (sum2 & 0xff) + (sum2 >> 8);
        *checkA = (uint8_t)sum1;
        *checkB = (uint8_t)sum2;
        return;
}
```

Figure 3.27: Fletcher checksum algorithm is implemented in C programming language

3.12 Finite State Machine

We have listed hardware and other software properties above. Many different software can be implemented using this system. Our project is about Flash Memory Based Voice Recording, Flash Memory is known as Solid State.

All embedded application is also a finite state machine. System is implemented according to below Figure 3.28:

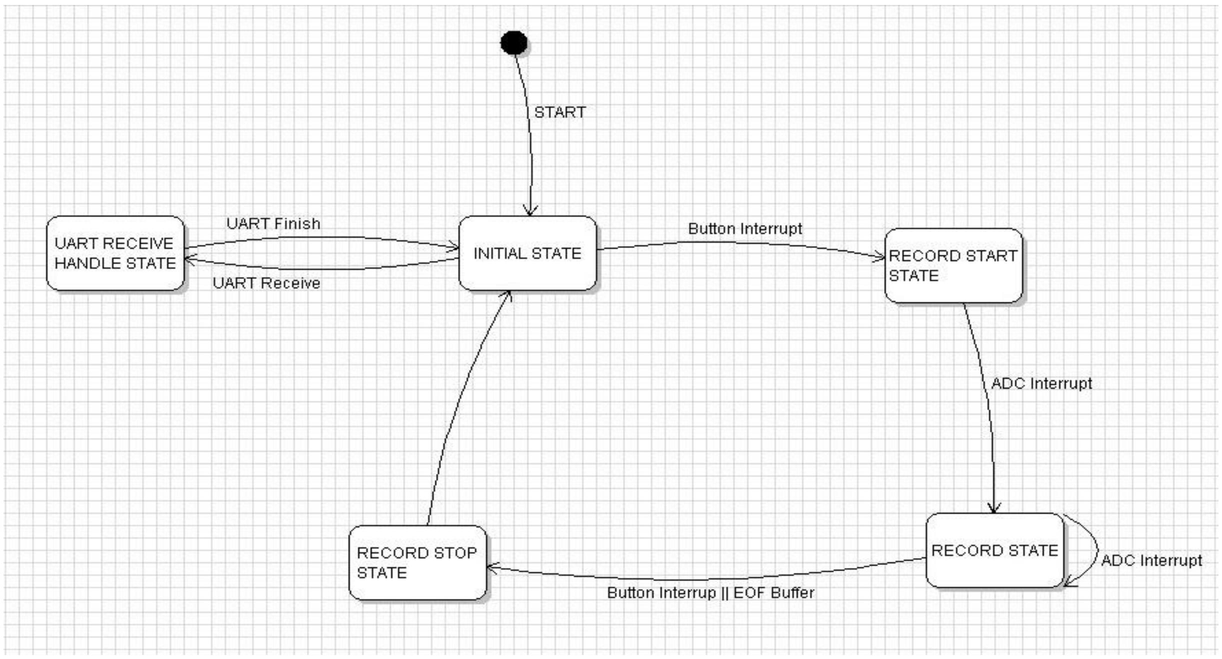


Figure 3.28: Finite state machine of the project

States and their properties are listed below:

1. Initial State

This state is the start state of the project and it is also the final return state of UART Receive Handle and Record State. In this state; board, MCU, FSM and sampling initialized. ADC is stopped. Button and UART receive are enabled. In this state, if record button is pressed then system goes to record start state. If a communication is started with PC application by using UART then system goes to UART receive handle state.

2. Record Start State

This state is the state which prepares system to record state. UART receive is disabled. Button is enabled. ADC is initialized. Sampling is started. When an ADC interrupt is occurred then this state goes to record state.

3. Record State

This state is the state which all recording is handled. Voice samples, which are prepared by ADC, is encoded and stored. If there is an ADC interrupt occurs then state returns back to itself but if there is a button interrupt or EOF buffer is reached then system goes to record stop state.

4. Record Stop State

This state is stop recording ADC is stopped and updates are done. When all processes finish then it goes back to initial state.

5. UART Receive Handle State

This state does handle all UART communication protocol. In this state, UART receive and button interrupt is disabled. UART commands are handled according to our UART communication protocol properties.

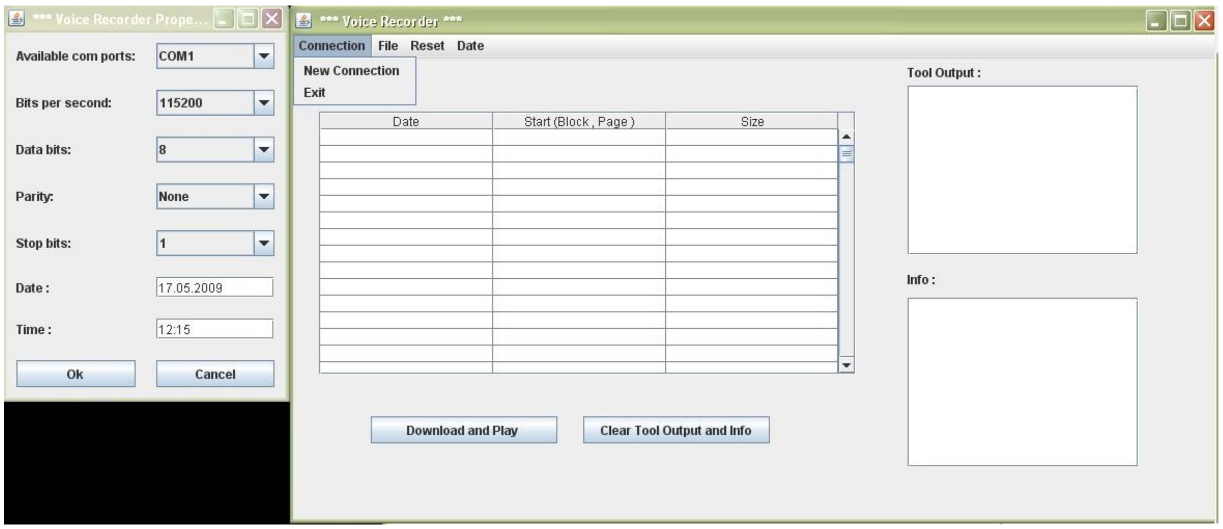
In all states, if there is an unknown incident occurs then system goes to a null state and blocks itself.

3.13 Communicating with Real World: PC Application

Most of embedded applications need a PC application in order to communicate with real world. They sense world, communicate with other embedded systems, process and store data. The raw data, is coming from MCU, is not perceptible to users. If a system does all stuff and does not give any perceptible output to users then it does not mean anything. We develop our own PC application to set MCU and get records.

PC application is developed by using Java Programming Language, JDK 1.6.0_07 and GiovynetSerialPort1.3 Library.

PC application is composed of a main window and a communication window. Communication window appears when a new communication is wanted to establish by choosing “Communication” tab on menu and clicking “New communication”. New communication window helps user to set communication’s properties, system date and time. MCU is configured to 115200 baud rate, 8 bit character, no parity bit and 1 stop bit so our configuration is based on those values.



Available com ports:	COM1
Bits per second:	115200
Data bits:	8
Parity:	None
Stop bits:	1
Date :	17.05.2009
Time :	12:15

Figure 3.29: Connection menu of PC application

All other processes are done on main window. On menu, there are four tabs and three of them are using in main window, which are File, Reset and Date tabs. There are also two different buttons which are “Download and Play” and “Clear Tool Output and Info” buttons.

On file tab, there is only one item which is “Get File List” and this item is used to get records file table from MCU. If it is clicked then a communication is established and MCU sends file table of records.

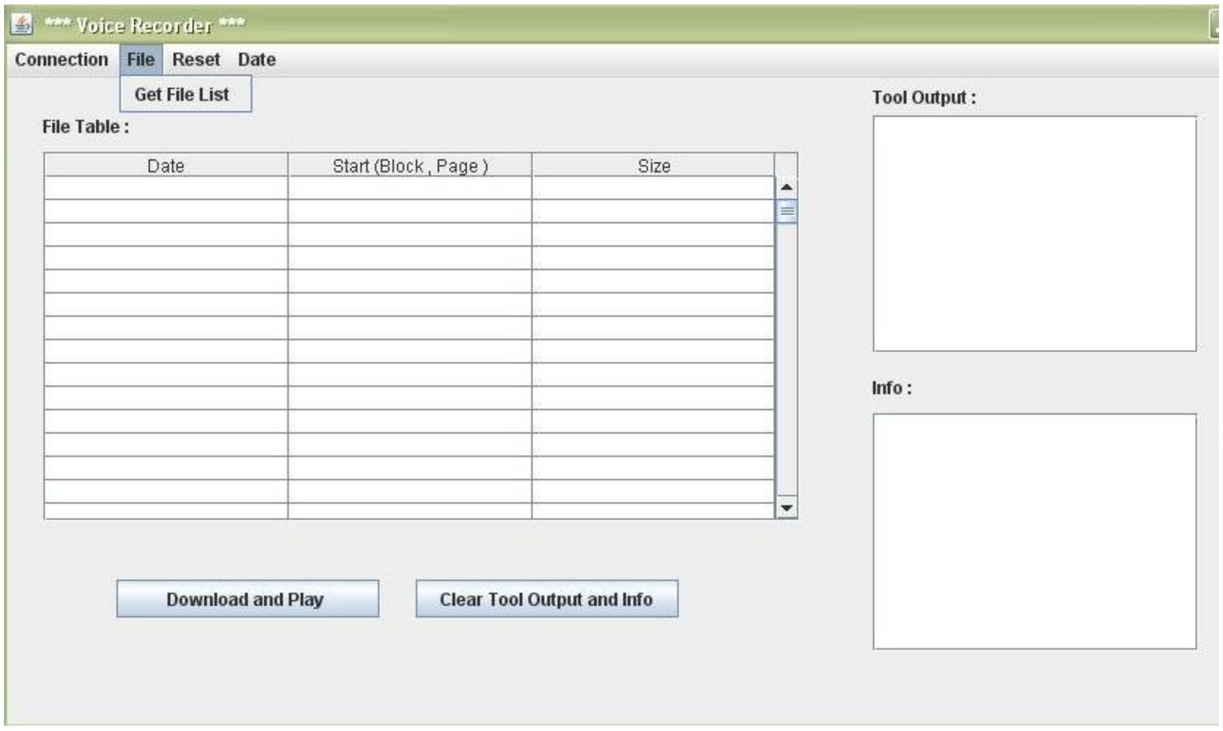


Figure 3.30: File menu of PC application

If user wants to set and get time of MCU then he must choose “Date” tab. If date tab is clicked there are three options, one of them is “Get MSP Date”, second one is “Set MSP Date” and third one is “Set Date”. Get MSP Date is used to learn the MCU internal time and if it is clicked then a window appears and shows the date of MCU. Set Date gets the date from user, to do so it shows a window to enter date and time setting. Set MSP Date sets MCU time according to our epoch 01.01.2009 00:00.

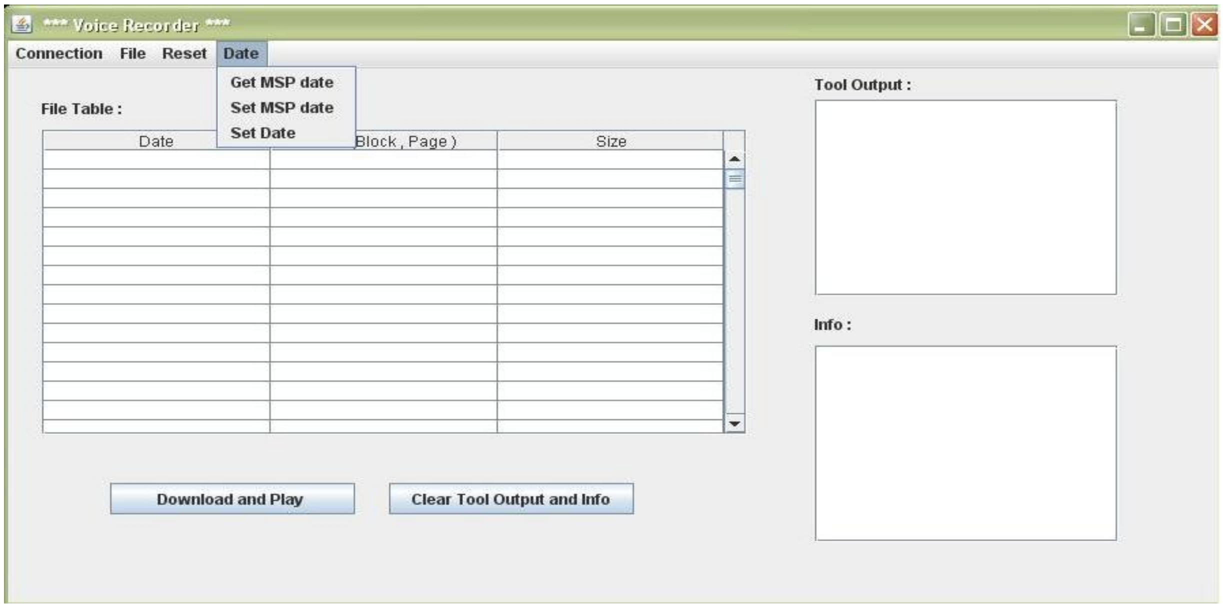


Figure 3.31: Date menu of PC application

On reset tab, there is only one item which is "Reset MSP430" and this item is used to reset file system of MCU. If it is clicked then a communication is established and MCU resets its file system table.

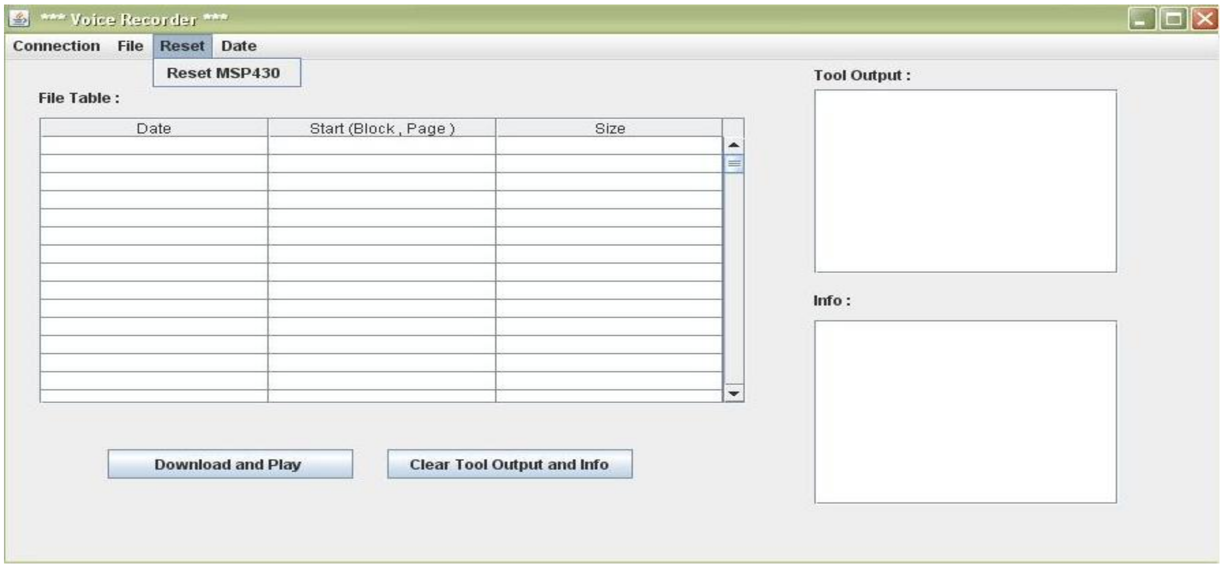


Figure 3.32: Reset menu of PC application

4. ANALYSIS

Analysis is an important part of developing an application. Especially embedded applications need much more analysis then PC application because in embedded systems memory, CPU speed and power is limited. In order to implement a good application, analysis must be done by considering power consumption, CPU usage and code size of application.

IAR Embedded Workbench is a good compiler and debugger interface and it has a good simulation module. By the use of this tool, CPU and memory analysis is done.

Power consumption of CPU is written on specific datasheet.

4.1 Memory Usage

According to linker output of IAR Compiler, the memory requirements are shown into following Table 4.1.

Table 4.1: Memory usage of whole code

	Code Memory	**Data Memory**	**Constant Memory**
No Optimized	3968 bytes	991 bytes	325 bytes
Size Optimized	3790 bytes	991 bytes	323 bytes
Speed Optimized	3804 bytes	991 bytes	323 bytes
Balanced Optimized	3794 bytes	991 bytes	323 bytes

The code size of some required modules is shown Table 4.2 which is based on Size Optimized Compilation:

Table 4.2: Memory requirements of some modules

	Code Memory	**Data Memory**	**Constant Memory**
ADPCM Module	202 bytes	4 bytes	186 bytes
Flash Module	50 bytes	15 bytes	-
Protocol Module	296 bytes	-	-
UART Module	200 bytes	4 bytes	-
Main Module	1286 bytes	736 bytes	73 bytes

4.2 Timing

4.2.1 CPU

The listed CPU usage analysis is done by IAR Embedded Workbench Simulator. The code is compiled at Size Optimized Mode.

The average values are calculated over 100 different executions in Table 4.3:

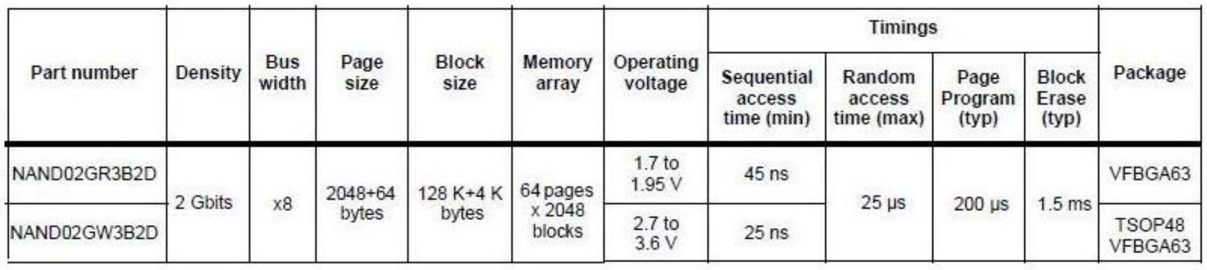
Table 4.3: CPU usage of key functions

	Cycles Per Call (Average)	**Theoretical time spent (for 12 MHz)**
Fletcher Checksum	2434 cycles	200 us
Encode	17024 cycles	1.5 ms
Erase Internal Flash	60 000 cycles	5 ms
Write Internal Flash	60 000 cycles	5 ms

4.2.2 Flash

External flash timing diagram is taken from its datasheet and results are shown below Table 4.4: [17]

Table 4.4: Timing of Numonyx NAND flash



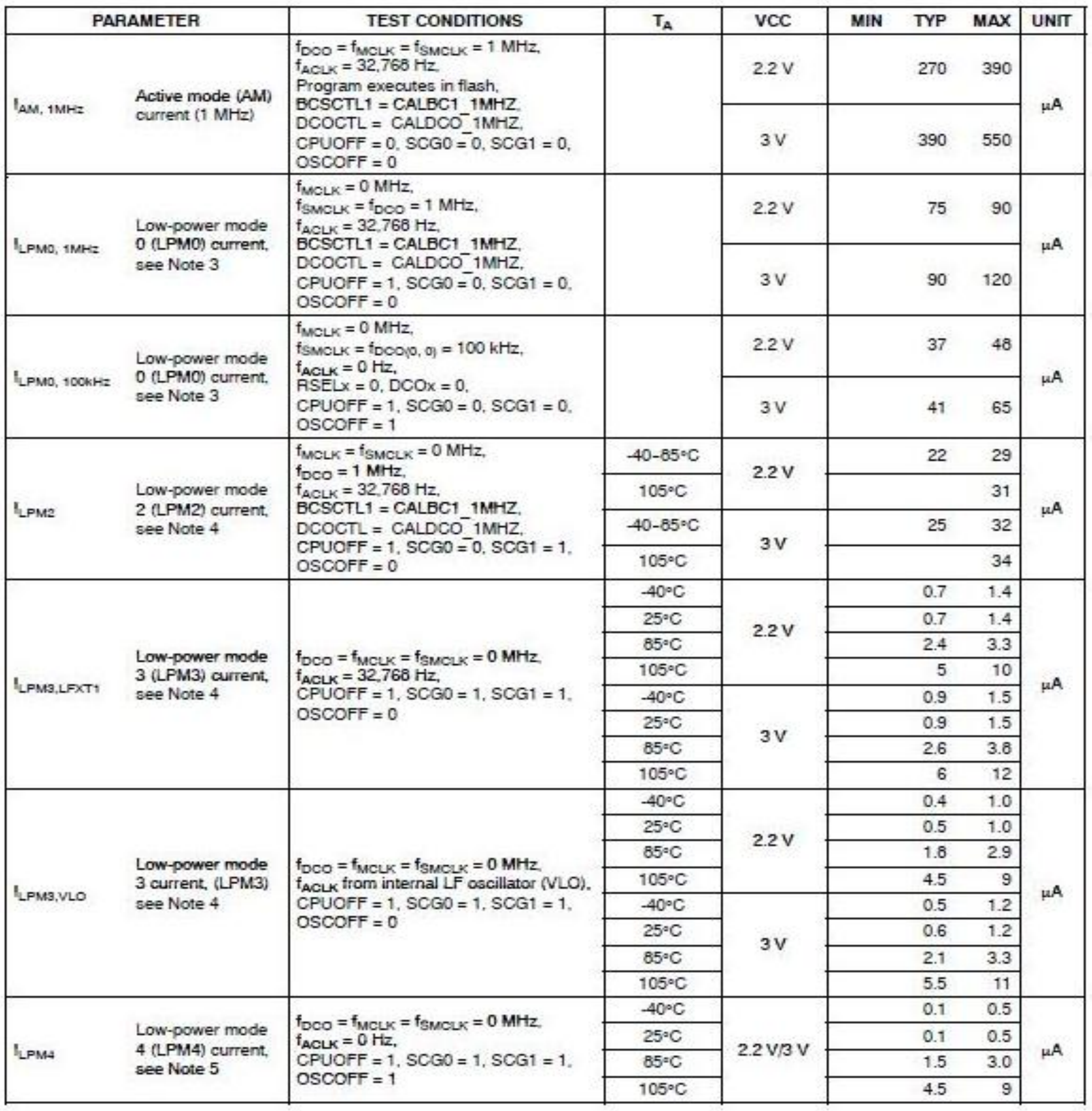
Part number	Density	Bus width	Page size	Block size	Memory array	Operating voltage	Timings				Package
							Sequential access time (min)	Random access time (max)	Page Program (typ)	Block Erase (typ)	
NAND02GR3B2D	2 Gbits	x8	2048+64 bytes	128 K+4 K bytes	64 pages x 2048 blocks	1.7 to 1.95 V	45 ns	25 µs	200 µs	1.5 ms	VFBGA63
NAND02GW3B2D						2.7 to 3.6 V	25 ns				TSOP48 VFBGA63

4.3 Power Consumption

4.3.1 CPU

According to datasheet the power consumption of MSP430 for different operating modes is as follows: [12]

Table 4.5: Current consumption of MSP430F2274 for different modes

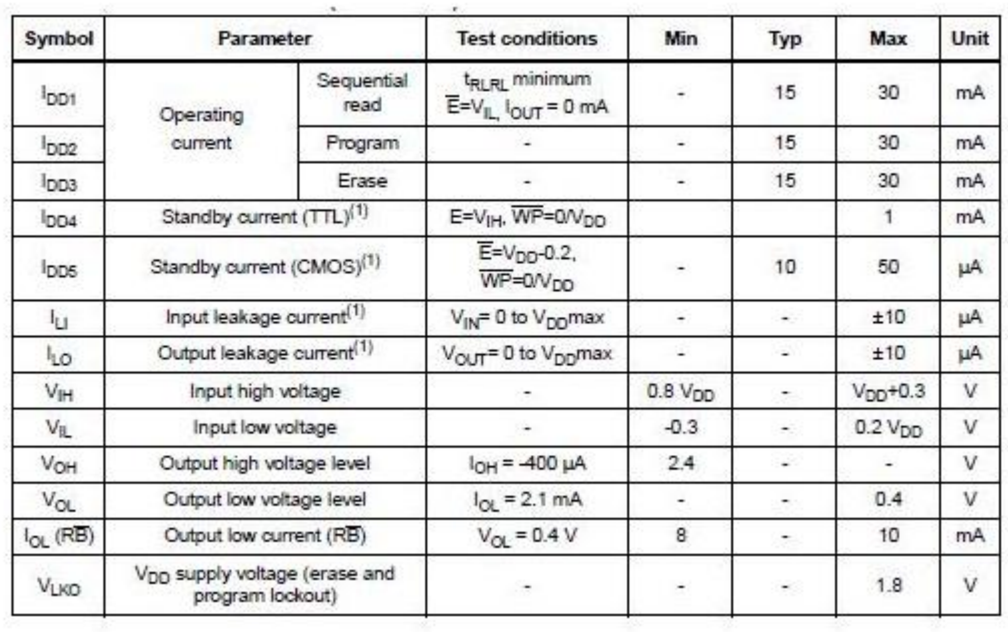


PARAMETER		TEST CONDITIONS	T_A	VCC	MIN	TYP	MAX	UNIT
$I_{AM, 1MHz}$	Active mode (AM) current (1 MHz)	$f_{DCO} = f_{MCLK} = f_{SMCLK} = 1$ MHz, $f_{ACLK} = 32{,}768$ Hz, Program executes in flash, BCSCTL1 = CALBC1_1MHZ, DCOCTL = CALDCO_1MHZ, CPUOFF = 0, SCG0 = 0, SCG1 = 0, OSCOFF = 0		2.2 V		270	390	μA
				3 V		390	550	
$I_{LPM0, 1MHz}$	Low-power mode 0 (LPM0) current, see Note 3	$f_{MCLK} = 0$ MHz, $f_{SMCLK} = f_{DCO} = 1$ MHz, $f_{ACLK} = 32{,}768$ Hz, BCSCTL1 = CALBC1_1MHZ, DCOCTL = CALDCO_1MHZ, CPUOFF = 1, SCG0 = 0, SCG1 = 0, OSCOFF = 0		2.2 V		75	90	μA
				3 V		90	120	
$I_{LPM0, 100kHz}$	Low-power mode 0 (LPM0) current, see Note 3	$f_{MCLK} = 0$ MHz, $f_{SMCLK} = f_{DCO(0, 0)} = 100$ kHz, $f_{ACLK} = 0$ Hz, RSELx = 0, DCOx = 0, CPUOFF = 1, SCG0 = 0, SCG1 = 0, OSCOFF = 1		2.2 V		37	48	μA
				3 V		41	65	
I_{LPM2}	Low-power mode 2 (LPM2) current, see Note 4	$f_{MCLK} = f_{SMCLK} = 0$ MHz, $f_{DCO} = 1$ MHz, $f_{ACLK} = 32{,}768$ Hz, BCSCTL1 = CALBC1_1MHZ, DCOCTL = CALDCO_1MHZ, CPUOFF = 1, SCG0 = 0, SCG1 = 1, OSCOFF = 0	-40–85°C	2.2 V		22	29	μA
			105°C				31	
			-40–85°C	3 V		25	32	
			105°C				34	
$I_{LPM3,LFXT1}$	Low-power mode 3 (LPM3) current, see Note 4	$f_{DCO} = f_{MCLK} = f_{SMCLK} = 0$ MHz, $f_{ACLK} = 32{,}768$ Hz, CPUOFF = 1, SCG0 = 1, SCG1 = 1, OSCOFF = 0	-40°C	2.2 V		0.7	1.4	μA
			25°C			0.7	1.4	
			85°C			2.4	3.3	
			105°C			5	10	
			-40°C	3 V		0.9	1.5	
			25°C			0.9	1.5	
			85°C			2.6	3.8	
			105°C			6	12	
$I_{LPM3,VLO}$	Low-power mode 3 current, (LPM3) see Note 4	$f_{DCO} = f_{MCLK} = f_{SMCLK} = 0$ MHz, f_{ACLK} from internal LF oscillator (VLO), CPUOFF = 1, SCG0 = 1, SCG1 = 1, OSCOFF = 0	-40°C	2.2 V		0.4	1.0	μA
			25°C			0.5	1.0	
			85°C			1.8	2.9	
			105°C			4.5	9	
			-40°C	3 V		0.5	1.2	
			25°C			0.6	1.2	
			85°C			2.1	3.3	
			105°C			5.5	11	
I_{LPM4}	Low-power mode 4 (LPM4) current, see Note 5	$f_{DCO} = f_{MCLK} = f_{SMCLK} = 0$ MHz, $f_{ACLK} = 0$ Hz, CPUOFF = 1, SCG0 = 1, SCG1 = 1, OSCOFF = 1	-40°C	2.2 V/3 V		0.1	0.5	μA
			25°C			0.1	0.5	
			85°C			1.5	3.0	
			105°C			4.5	9	

4.3.2 Flash

Flash power consumption is taken from specific datasheet and it is shown below Table 4.6: [17]

Table 4.6: Current consumption of Numonyx NAND flash



Symbol	Parameter		Test conditions	Min	Typ	Max	Unit
I_{DD1}	Operating current	Sequential read	t_{RLRL} minimum $\overline{E}=V_{IL}$, $I_{OUT} = 0$ mA	-	15	30	mA
I_{DD2}		Program	-	-	15	30	mA
I_{DD3}		Erase	-	-	15	30	mA
I_{DD4}	Standby current (TTL)(1)		$E=V_{IH}$, $\overline{WP}=0/V_{DD}$			1	mA
I_{DD5}	Standby current (CMOS)(1)		$\overline{E}=V_{DD}-0.2$, $\overline{WP}=0/V_{DD}$	-	10	50	µA
I_{LI}	Input leakage current(1)		$V_{IN}= 0$ to V_{DD}max	-	-	±10	µA
I_{LO}	Output leakage current(1)		$V_{OUT}= 0$ to V_{DD}max	-	-	±10	µA
V_{IH}	Input high voltage		-	0.8 V_{DD}	-	V_{DD}+0.3	V
V_{IL}	Input low voltage		-	-0.3	-	0.2 V_{DD}	V
V_{OH}	Output high voltage level		$I_{OH} = -400$ µA	2.4	-	-	V
V_{OL}	Output low voltage level		$I_{OL} = 2.1$ mA	-	-	0.4	V
I_{OL} ($R\overline{B}$)	Output low current ($R\overline{B}$)		$V_{OL} = 0.4$ V	8	-	10	mA
V_{LKO}	V_{DD} supply voltage (erase and program lockout)		-	-	-	1.8	V

5. CONCLUSION

In the scope of this project, we developed a solid state voice recorder system. Also we developed other useful hardware abstraction layer library in order to develop different project by using our library.

We used MSP430F2274 and Numonyx NAND02G-B2D. They are good choices to use because they both have low power mode and current consumption of devices are very small. This is an important point to develop low-power embedded systems.

RS-232 is an industry standard and very well known serial communication protocol to communicate embedded system with PC.

PC application is implemented by using Java Programming Language. Java is another good choice of the project because it is an open source language and it is widely supported by all types of computer.

IMA's ADPCM implementation is an optimum solution for this kind of limited applications, providing good sound quality. MSP430 has a limited computational power but this codec can be enhanced for future developments.

5.1 Engineering Criteria

MCU is one of the most important parts of the project. It must be chosen well. We have two different MCUs of Texas Instrument Inc. One of the MCU is MSP430F2274 and another one is MSP430F169. MSP430F169 has much more current consumption, almost twice, than MSP430F2274. Wake up time from standby mode of MSP430F169 is six times greater than MSP430F2274. MSP430F2274 has cycle time which is half of MSP430F169's. Another important point is price of MCUs, MSP430F2274's price is around 5 dollar but

MSP430F169’s price is around 13 dollar. Because of these reasons we choose MSP430F2274 as our MCU.

Flash technology is the hearth of the project. There are NAND or NOR and serial or parallel flashes. NOR flashes have higher read speeds but their write speed is smaller than NAND flashes. Serial flashes have smaller pin needs but their write speed is also smaller and for sequential access address of data must be set for every two bytes. Parallel flashes have higher write speed and sequential access needs once address initialization. Parallel flashes also divide two sub categories x8 and x16. X8 category has 8 data pins and x16 category need 16 data pins. Our MCU does not have too much pins to give the x16 parallel flashes. As a consequence of this reasons, we choose x8 parallel NAND flash.

Voice samples needs too many spaces but we have a limited capacity of flash. In order to keep maximum utilization space, we need to compress and encode samples. Many different alternatives can be used to compress voice samples but MCU has a limited calculation capacity. According to computation need and compression ratio, ADPCM is the optimum solution.

Many embedded applications also need PC applications and our PC application is developed by using Java programming language. There are many choices about programming language such as C, C++, C# and Java. C and C++ are good choices about speed but they are not good at visual programming and also platform dependent. C# is another good choice and easy to program but it is also platform dependent. Java is the best solution because it is platform independent, easy to program and widely supported open source system so we choose Java programming language.

Our RAM and computation speed are limited. To have better system, we need to optimize our program. We create our own functional libraries, use well organized efficient algorithms and try to keep MCU as busy as possible between sample’s times. Code is also complied according to different optimization such as speed, size or balanced.

File system is important for solid state voice recorder system. FAT and NTFS file systems are analyzed but they need to much MCU cycles and it is not easy to implement them. Our system is not reached directly from other systems. It gets commands, analyzes it and gives the need output. Because of these reasons, we create our own file system to get a better CPU utilization and keep records as safe from outsourced attacks.

PC application and MCU need a language to talk to each other in a proper way. We design a communication protocol. This communication protocol must use minimum commands and each command must need smaller time to be analyzed. There are too many noises that change the sent data. To have a reliable communication, we need a checksum algorithm. Cyclic Redundancy Check is on of the choices but it needs to many computation and it is not easy to implement. Simple checksum algorithms like sum all of the data and sending total sum is easy to implement but it needs also too many bytes as checksum. Fletcher checksum algorithm is a usual checksum algorithm, it does not need to many computation and its sum is just two bytes so we implement Fletcher checksum algorithm to keep our communication reliable.

5.2 Further Development

This project can also be enhanced in some ways:

- CPU can be worked at higher frequencies and so computational depending functions can changed to get better results but power consumption of system is increased.
- Multiplane programming can be implemented on flash library and writing speed is increased on average.
- Multiplane block erasing can be used to decrease erasing time of flash.
- UART communication protocol can use higher baud rates but better checksum and also error correction codes must be implemented to decrease resending of corrupted data.

- External codec implemented chips can be added on board to decrease the work-load of CPU.
- PC application may be enhanced to store voice records as different file format and merge records into one file or split records into different file.

REFERENCES

1. Jody Rosen, *"Researchers Play Tune Recorded Before Edison"*, New York Times, 2008.

2. Patrick Feaster, *"Speech Acoustics and the Keyboard Telephone: Rethinking Edison's Discovery of the Phonograph Principle"*, ARSC Journal, Spring 2007.

3. Oliver Berliner and Patrick Feaster, *"Letters to the Editor: Rethinking Edison's Discovery of the Phonograph Principle"*, ARSC Journal, Fall 2007.

4. Gelatt Roland, *"The Fabulous Phonograph: From Tin Foil to High Fidelity"*, Philadelphia J. B. Lippincott Company, 1955.

5. Allen, *"Edison Cylinder Records, 1889-1912"*, New York Stellar Production, 1969.

6. Leonard DeGraff., *"Confronting the Mass Market: Thomas Edison and the Entertainment Phonograph"*, 1995.

7. S. J. , "Magnetic Recording", New York Rinehart & Company, 1949.

8. Jan Axelson, *"The Microcontroller Idea Book Circuits"*, 1994.

9. Han-Way Huang, *"PIC microcontroller: an Introduction to Software and Hardware Interfacing"*, 2005.

10. Kenneth J. Ayala , *"The 8051 Microcontroller"*, 2004.

11. *"MSP430x2xx Family Users Guide"*, Texas Instrument Inc., 2007.

12. *"MSP430F2274 Device Datasheet"*, Texas Instrument Inc., 2007.

13. *"MSP430F2274 Device Erratasheet"*, Texas Instrument Inc., 2008.

14. *"NAND vs. NOR Flash Memory Technology Overview"*, Toshiba Inc., 2006.

15. *"Flash Memory Technology"*, Integrated Circuit Engineering Corporation.

16. *"Introduction to Serial Flash Memory"*, Next Flash Technologies Inc., 2003.

17. *"NAND02G-B2D Datasheet"*, Numonyx Inc., 2008.

18. Dominic Giampaolo, "*Practical File System Design with the Be File System"*, 1999.

19. *"Engineer-to-Engineer Note EE-329"*, 2007.

20. *"Extensible Firmware Initiative FAT32 File System Specification"*, Microsoft Corp., 2000.

21. John Leis, *"Disk File System Examples"*, 2006.

22. R.G. Gallager, *"The Sampling Theorem"*.

23. C. E. Shannon, *"Communication in the Presence of Noise"*, Proc. Institute of Radio Engineers, Jan. 1949. Reprint as classic paper in: Proc. IEEE, Vol. 86, No. 2, (Feb 1998).

24. Dan Lavry, *"Sampling Theory for Digital Audio"*, Lavry Engineering Inc.

25. *"PCM Tutorial"*, L3 Telecommunications.

26. Yao Wang, *"Source Coding Basics and Speech Coding"*, Polytechnic University, Brooklyn.

27. Nimrod Peleg, *"Introduction to Speech Coding"*, 2009.

28. Bernd Girod, *"EE368b Image and Video Compression DPCM Overview"*, Stanford University.

29. *"The Implementation of G.726 Adaptive Differential Pulse Code Modulation (ADPCM) on TMS320C54x DSP"*, Texas Instrument Inc., 1997.

30. Mark Valentino, *"Microphone Handbook"*, PCB Piezotronics Vibration Division.

31. Rob Ryan, *"RIFF WAVE (.WAV) File Format"*, Brown University.

32. Ph.D. Suree Pumrin, *"Introduction to Microprocessors Chapter 12 Serial Communications"*.

33. *"Application Note 83 Fundamentals of RS-232 Serial Communications"*, Maxim Integrated Products Inc., 2001.

34. *"RFC 1146 Fletcher Checksum Algorithm"*.

35. *"RFC 905 - ISO Transport Protocol Specification*".

36. Chris Nagy, *"Embedded Systems Design Using the TI MSP430 Series"*, 2003.