ASCII Reader

Ioana Cristescu

Submitted for the Digilent Digital Design Contest

at the Technical University of Cluj-Napoca in Romania

May 9, 2009

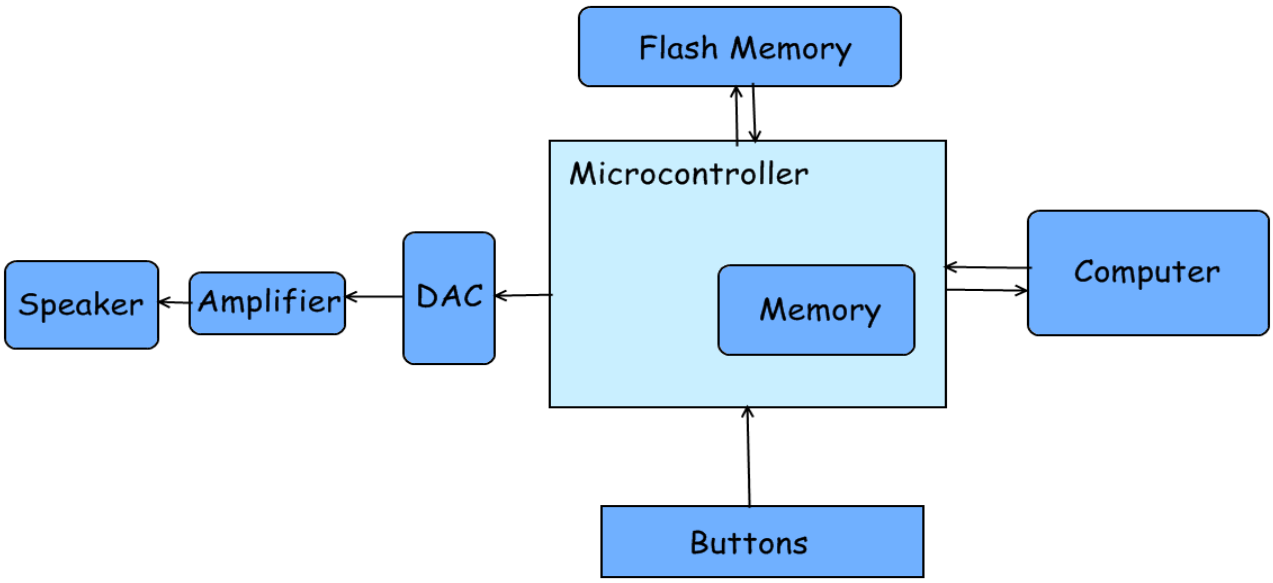
Technical University of Cluj-Napoca

ASCII READER

The product is a microcontroller based application. It reads the ASCII code of the letters sent by the computer to the microcontroller. It's a fun application, which you can use for didactic purposes or as a base for further development. Due to its structured code, extensions can easily be added.

Some features of the product:

- Send sounds recorded from your computer to the speaker.
- Communicate with the Flash Memory: read, write, erase, read status, identification.
- Communicate with a computer: from simple commands to files sending.
- User interaction provided through the buttons and the computer.



1. Introduction

Objectives

The ASCII reader is an application build on a microcontroller and on several other hardware components. It reads a text from the computer and it renders it on a speaker connected to the board. The project is only intent for letters reading, they don't form words since words ought to be divided in syllabus and not on letters. Nevertheless, there is only a matter of time to transform the current application on word reader, since all hardware is available and the code provided needs little modification for further development.

The motivation for developing this application was mostly because it was an interesting idea. It required the use for several components and it allowed me to use microcontroller architecture at its maximum capabilities.

Features-in-Brief

The principle feature is that to send to a speaker the waves that represent the letters to be rendered. Also the programmer can use the application to store the sounds in a non erasable memory.

Besides its main objective it allows the user to do several interesting things using the flash memory, such as erasing entirely or only parts of it, reading status or identification.

Project Summary

The project is divided into hardware components and this is due entirely to the fact that some features needed for the implementation of the project aren't provided by the Cerebot board. As a consequence, the code was also divided in a similar manner, in order to make reusability available as much as possible.

Another way of seeing this project is in a more abstract level. One part of the project is concerned with storing data in the flash memory and another component is the rendering component. The two components are also divided into two files, for several reasons. It is easier to use, since the developer doesn't have to deal with code it doesn't need. Also it seems to be a safe way to divide the project. After the writing part on the memory is done, it can be assumed you don't want to override that information.

Tools Required

Hardware tools:

- Cerebot II board
- PmodAMP1 Speaker/Headphone Amplifier
- Pmod DA1 Digital To Analog Module
- Pmod RS232Converter
- Pmod BTN Pushbutton Module
- PMOD Serial Flash

Software tools:

- A usart communication software from the computer to other devices through a cable, like the Hyper Terminal from Windows, or Real Terminal.
- A recording tool, like Realtek.

Design Status

The project is complete in the sense that it is an application that stands on its own, that offer a set of features that can easily be tested. But, from the point of view of what it was initially intended to do, there is one major feature, missing. The project cannot record the sounds by itself. The sounds that are transmitted through the amplifier need to be previously recorded and some mastering needs to be done. This feature would require a lot of work to be done properly and unless it is done properly it will be useless.

Another important feature that could be added to the current project is making it a word reader. This would require syllables to be stored in the memory instead of letters. The necessary code to be written in order to make this expansion is meticulous but it doesn't require any designing work and is therefore only a matter of time.

2. Background

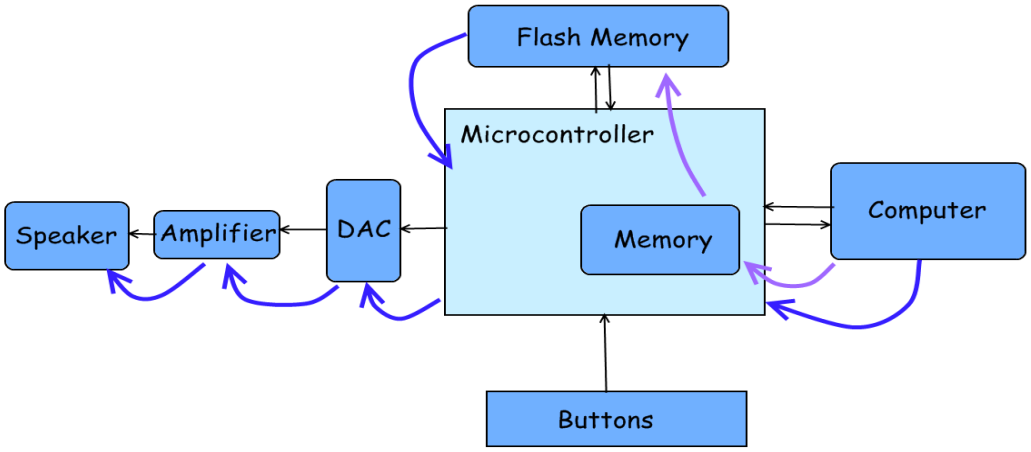
Why This Project?

The reason I decided to do this project it's because it seemed as a fun idea. I was able to work with several hardware components, to make them communicate, and as I anticipated, most problems appeared in this problem. I enjoy discovering how more experienced engineer design certain components (how to communicate with a flash memory for instance), and to grasp the idea behind their design choices. Also, it wasn't an idea I've heard it was implemented on a microcontroller, and I found it challenging to do something somewhat different of the usual usage of a microcontroller.

3. Design

Features and Specifications

Principal features of the project:



- A file of about 3 MB is sent from the Flash memory to the amplifier. Before being read by the amplifier it is converted to an analog file. From that the amplifier amplifies it and makes it an audio signal.
- A file of 3 MB representing a letter's pronunciation is stored in the flash memory. The file is initially created by recording the sound, transform it into a text file, send it to microcontroller through the usart channel. The file is initially stored in the microcontroller memory and afterwards it is transferred in the Flash memory.

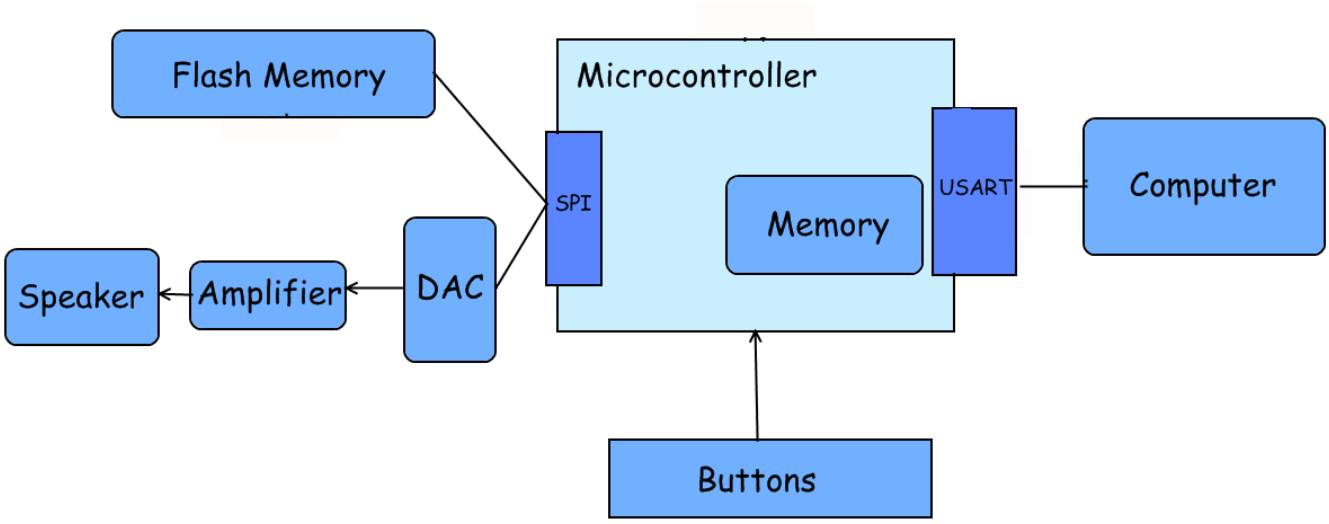
Other features of the projects are:

- Flash Memory:
  - The memory can be erased in bulk or by sector. It's possible to rewrite the memory, but in order to do this, you have to erase it first. Therefore, erasing is as important as reading and writing.
  - Status register can be read at any time from the memory. Most important usage is for reading the busy flag, while the memory is occupied with erasing or writing the execution of the program has to wait, or has to deal with something else.
  - The identification of the flash memory can be read for information like the manufacturer's or the device's identification.

- Microcontroller's memory:
  - Reading the microcontroller's memory and display it in a file or on the screen of a computer terminal. This can be used, in the context of this project, to verify the sound's representation in hexadecimal.
  - Erase the microcontroller's memory, not such an important features as turning the power off does this as well.

Design Overview

The top level diagram:

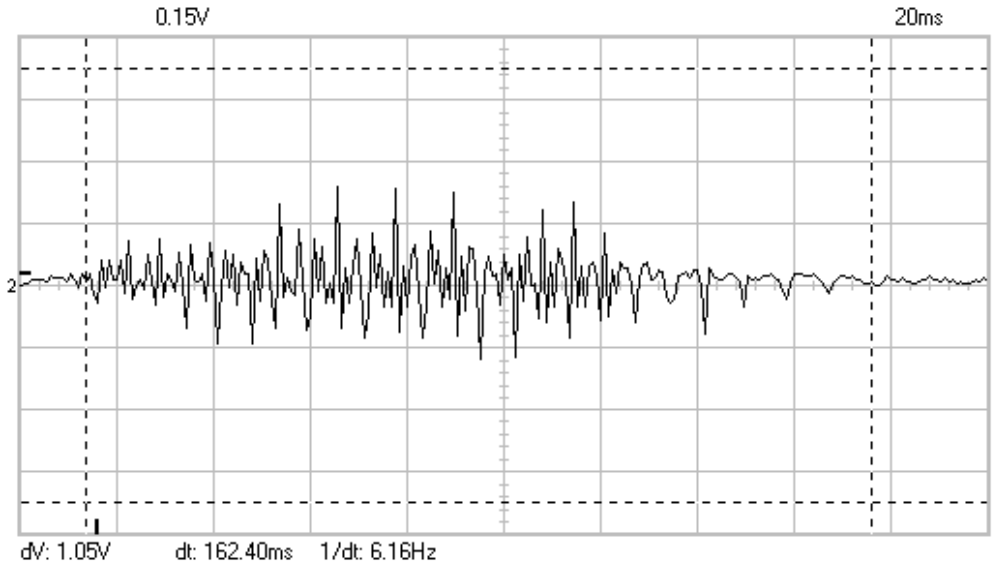


Component	Write the letters pronunciation	Read the letters
USART Unit	It will provide the 3MB of data that represent the sound and that needs to be stored in the microcontroller memory.	The computer will send the letters that it wants to be read.
SPI Unit	It will make the microcontroller's memory and the flash memory communicate in order to exchange data.	The flash memory and the amplifier communicate using the spi channels.
Amplifier		The low power audio signal is amplified and it is send to the speaker.
Flash Memory	The files corresponding to each letter are written in the flash memory.	The memory is read and data is send through the amplifier.

Detailed Design Description

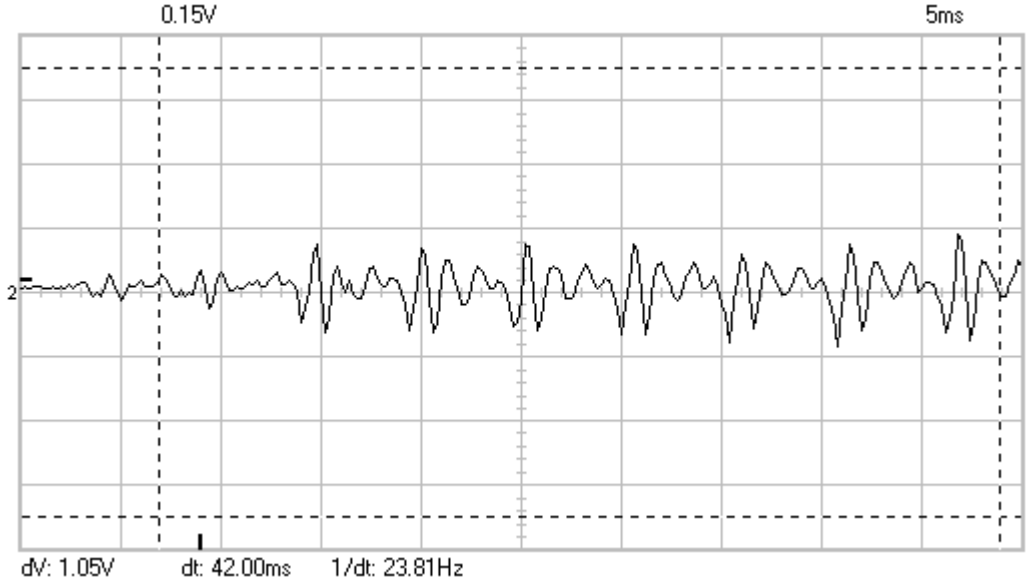
1. The Sounds

The document will present as example the sound waves of 2 letters.

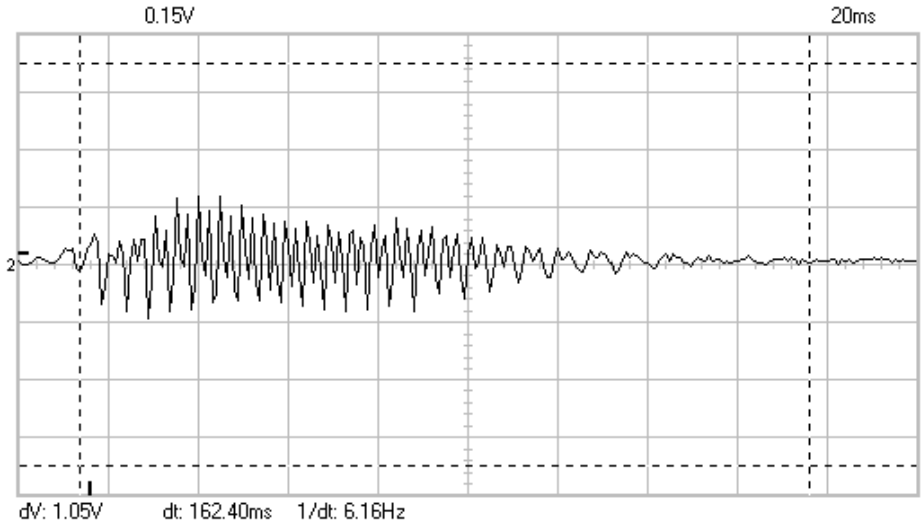


Letter A

This is sampled at 6Hz and it takes 162 ms.
Let's take the first part of the letter.



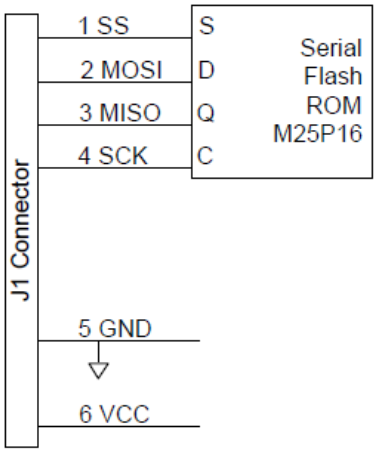
A good sampling frequency for this would be at every 0.1ms. Therefore for a letter, about 1600 samplings are needed.



Letter B

2. The Flash Memory

Hardware Component: Digilent PmodSF



The Flash memory has a size of 16Mbits and it communicates with the microcontroller using an SPI protocol.

Sector	Address Range	
31	1F0000h	1FFFFFh
30	1E0000h	1EFFFFh
2	020000h	02FFFFh
1	010000h	01FFFFh
0	000000h	00FFFFh

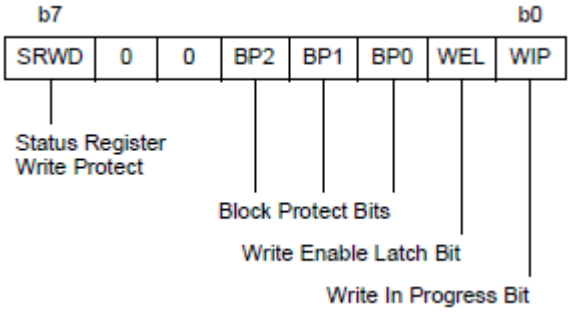
Memory Organization

Operations done on the flash memory:

Before calling an instruction the memory has to be selected by driving the chip select low. If the chip select is driven high before sending all necessary data the current instruction is ignored. During the instruction, the code for the instruction and all other necessary data is sent through the data in channel and the synchronization signal through the clock signal.

- Write Memory – Page Programming

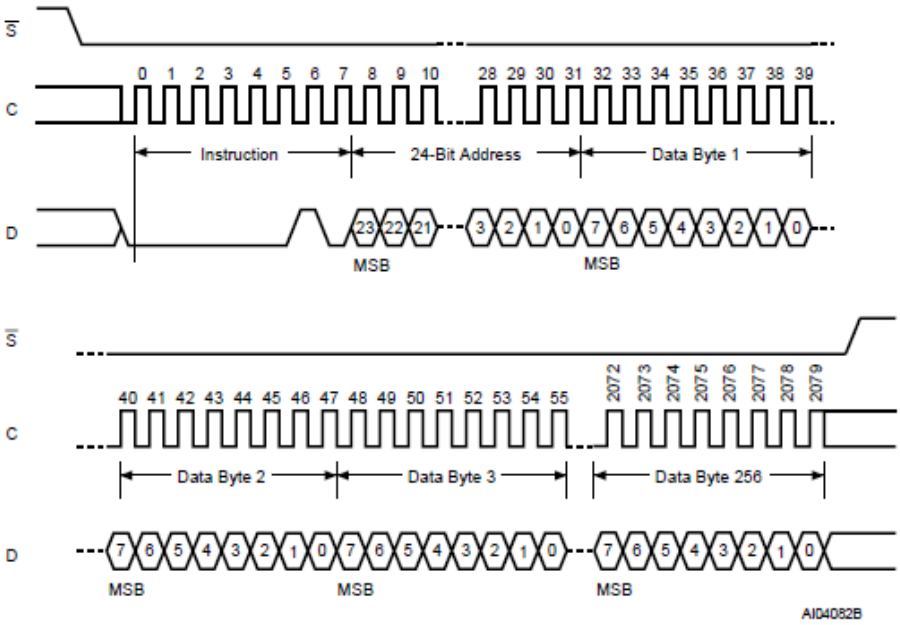
Before writing data in memory we first have to set the Write Enable Bit in Status Register. This is done by calling the Write Enable instruction.



Status Register.

The chip select is driven low and the following data has to be sent through the data in channel:

- The instruction code – 1 byte
- The start address – 3 bytes
- The data – can either be 8 bytes or 256 bytes at once.



Instruction Sequence

Each letter is stored in a separate control in order to make reading and writing easier to handle. So from letter A (sector 0) to letter T(sector 21), but not all letters are stored yet in the memory.

.

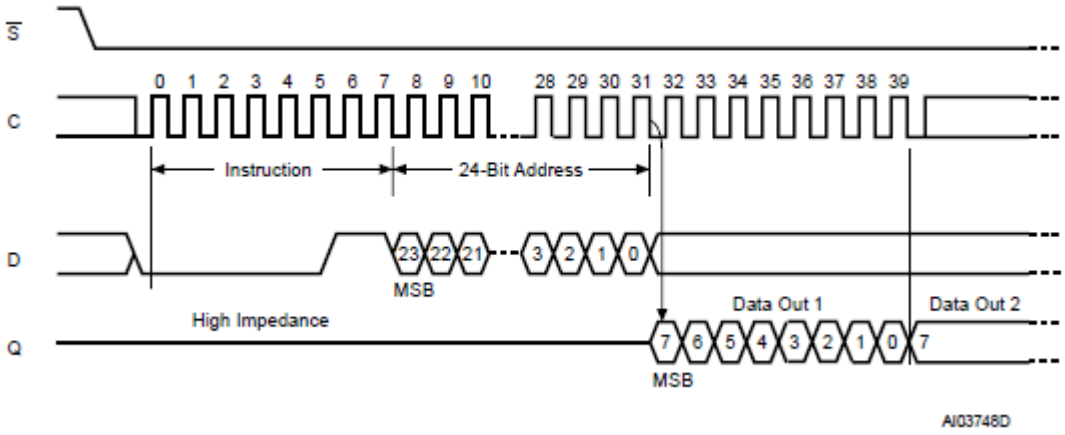
- Read Memory - Read Data Bytes

The chip select is driven low and the following data has to be sent through the data in channel:

- The instruction code – 1 byte
- The start address – 3 bytes

The following data is received through the data out channel:

- The data bytes – 8 bytes every cycle until the chip select is driven high. The direction of the data reading is from 000000 to 1FFFFF.



Instruction Sequence.

As bytes are read from the memory they are sent to the amplifier. This is done by controlling the synchronization signal, because the memory waits for that signal in order to send the next data on the data out channel.

- Erase all memory - Bulk Erase
  Erase a sector of the memory – Sector Erase

This instruction sets all bits to 1 (FFh). A write instruction actually is only performing a 1 to 0 transformation where needed, therefore setting all bits to 1 is a necessary step before overwriting something. Before it can be accepted, a Write Enable (WREN) instruction must previously have been executed.

The chip select is driven low and the instruction code has to be sent through the data in channel, followed by the address of the sector if needed.

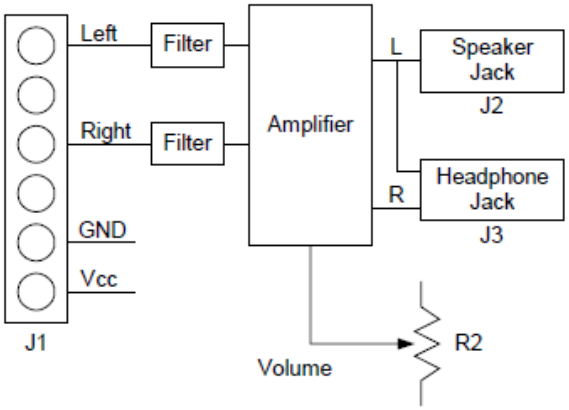
Other observations:

Memory write and erase are taking a little more time to be executed. During this time, the busy flag in the memory is set. Any other instruction send to the memory will be ignored. In the code, this waiting period is implemented as a delay.

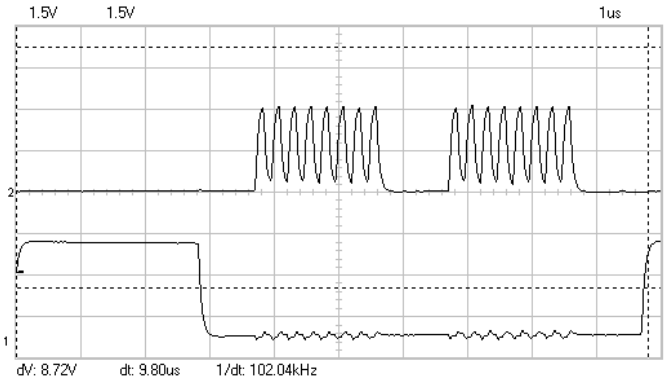
	Typical Delay	Maximum Delay
Page Program Cycle Time	1.4 ms	5 ms
Sector Erase Cycle Time	1 s	3 s
Bulk Erase Cycle Time 17 40 s	17 s	40 s

3. Amplifier

Hardware component: Pmod AMP1



The amplifier receives a analog signal, with an input voltage range of 0-Vcc, and it is send to the speaker. This signal is the output of the analog to digital converter available on the Cerebot board. The low pass filter on the input will again act as a reconstruction filter and remove the high frequency artifacts introduced by the sampling process.



The amplifier works with 12 bits for a sample. Therefore the content of the shift register has to be sent twice for each byte. The most significant 4 bits will be 0s and the 8 least significant bits will be the actual audio signal.

4. The USART Unit

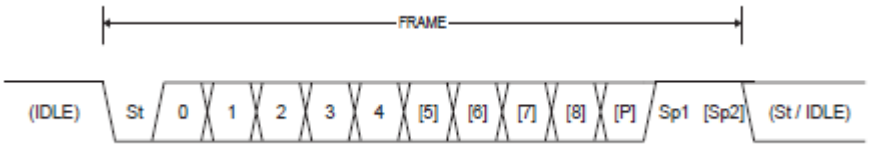
The 2 channels that make up the spi communications are:

- Rx – data in channel.
- Tx – data out channel.
- Clock

The 2 signals are sampled on opposite clock edges.

The unit has 3 procedures:

⌈ Initialization – the following format was set:



- St Start bit, always low.
- 8 Data bits
- P Parity bit. In my application it is not used
- Sp 2 Stop bits, always high.

⌈ Transmitter – when the procedure is called the data from the I/O Data register is shifted out.

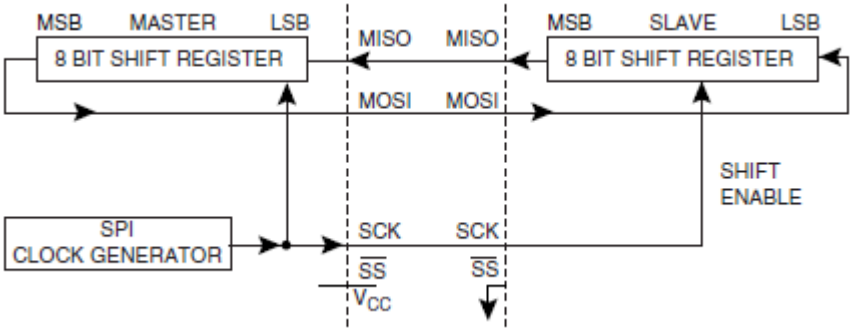
⌈ Receiver – when the computer sends something through the communication channel, a interrupt routine is called which handles the data received. The reason this operation works on an interrupt request is because the data has to be written in the memory, and that might create a delay during which sent data is lost. Therefore while the I/O shift register is being filled with bits from the computer, the microcontroller is writing the previously processed data in its memory. The interrupt service routine is only called when the shift register is full.

5. The SPI Unit

The 4 channels that make up the spi communications are:

- Synchronization signal – it sends the clock from the master to the slave and data is transmitted in accordance with this signal.
- Master Out Slave In – data out channel for the microcontroller, data in for the memory and amplifier.
- Master In Slave Out – data in channel for the microcontroller, data out for the memory.
- Chip Select – when high it selects the connected component for the spi communication. Since there are 2 components communicating with the microcontroller through the same spi channels, this is the one channel that is taken from two ports, one for each component.

The unit has 3 procedures:



- Initialization

  The following control values needs to be set:

  - Clock Polarity: gives the state of the synchronization when idle. When this bit is written to one, the synchronization signal is high when idle.
  - Clock Phase: determine if data is sampled on the leading (first) or trailing (last) edge of synchronization signal. It is changed from one spi slave (the flash memory data is read on rising edge) to the other (the amplifier data is read on the falling edge).
  - Enable Spi.
  - Set microcontroller as Master.
  - Set Clock Rate.

- Transmitter – when the procedure is called the data from the I/O Data register is shifted out.
- Receiver – for receiving the data from a slave, the master still has to send the clock and read the I/O register at the end of a 8 clock cycle run.

6. Buttons

The interrupt services routine are called from the interrupt vectors table. The routines are setting a flag, which is interpreted by the main program in a switch case kind of loop. This way the time consuming procedures are called outside the interrupt routines.

7. Testing

Testing was done at every stage of the project. First files we're transferred in the microcontroller memory and read back from there. Then the files were sent to the amplifier. In the process of making the amplifier work, the oscilloscope was used.

In the final phase of the project the testing was only done on the board.

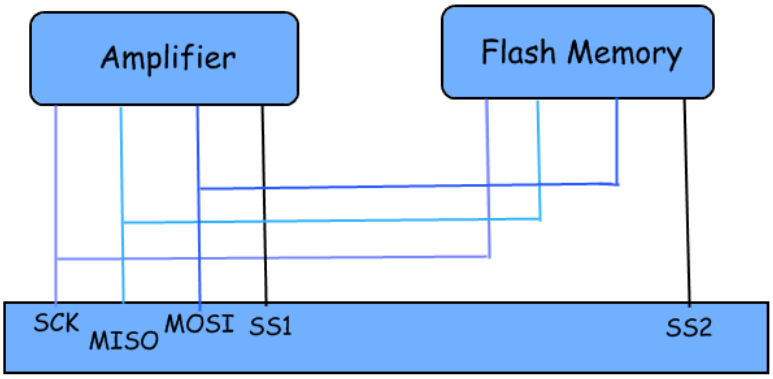
5. Discussion

Problems Encountered

Software or Hardware SPI

Two components have to speak to the microcontroller, through the spi ports. One byte is read from the Flash Memory and it is send to the amplifier. To implement this the following options were considered:

- Software SPI – instead of using the SPI communication protocol already available on the microprocessor, recreate the protocol through code. Then any available ports remained on the board could be used. Problems related are the fact that it is a relative slow approach, and secondly, the implementation is not easy to do.
- Hardware SPI – this is the implementation chosen. The idea is that instead of writing the code for the SPI protocol, use the already available one, by taking turns on the two components. An extra pin was used, for chip select. From the 6 pins, only 5 are common: clock, miso, mosi, vcc and Ground. The chip select has to be specific to each component, and it is chosen when chip select is driven low. It is faster and easier to implement.



C or assembly

- C code: It is easier to write and to implement a design in C. The application has to deal with arrays and their allocation in memory. The C compiler does this memory allocation; therefore it would have been a easier work. But it was not a fast approach, and the speed problem is essential for sound rendering.

- Assembly code: It is faster. The registers are easier to use, their value can be controlled by the user.
- Both: Some instructions can be written in assembly, a specific format has to be followed. But the problem with the methods I found for combining the 2 languages is that the variables are not common.

Engineering Resources Used

- Atmel AVR Tools / AVR Studio 4 – the gcc compiler for asm code. A .hex file is generated
- Digilent Tools / AVR Programmer – for loading the program memory with .hex file.
- Real Terminal – for the uasrt communication between computer and board.
- A recording tool like Realtek. The sampling frequency has to be set to 11.025KHz.

References

- Microcontroller:

ATmeag64 datasheet
Atmega64 instruction set.

- Digilent Pmods –available at www.digilentinc.com –

Digital to Analog Converter: Pmod_DA2_rm.pdf
Amplifier: Pmod_AMP1_rm.pdf
Buttons: Pmod_BTN_rm.pdf
RS232: Pmod_RS232_rm.pdf
Flash Memory: Pmod_SF_rm.pdf

- Other:

Cerebot_II_rm.pdf
MP25P16 Flash Memory datasheet

Appendix A:

ProjectExpanded.asm: -this file contains the implementation for writing files to the Flash Memory and other procedures/routines used as background fetures.

```
.include "m64def.inc"

.org 0x0000
      rjmp    main
.org 0x000A
      rjmp    sector_erase
.org 0x000C
      rjmp    button2_push
.org 0x000E
      rjmp    render_sound_memory
.org 0x0010
      rjmp    button4_push

.org 0x003C
        rjmp        USART_Receiver

.def usartdata = r16
.def spidata = r17
.def temp = r18

.def counter1 = r24
.def counter = r25

.def flag = r19
.def memAddr = r22 ;flash memory

.def temporary = r23

.def x1 = r27 ;access memory
.def x2 = r26

.def z1 = r31 ;clear memory
.def z2 = r30

.def y1 = r29 ;counter on 4bytes
.def y2 = r28

.def bytedata = r15
.def sector = r10

main:
      ;initialisation

    ldi   r16, LOW(RAMEND)
      out   SPL, r16
    ldi   r16, HIGH(RAMEND)
    out   SPH, r16

      ; set direction of port E (buttons) for input
      ldi r19, 0x04
```

```
	out DDRE, r19

	;init the usart communication
	rcall USART_Init
	;init the SPI communication
	rcall SPI_MasterInit

	;init the interrupts
	ldi temp,0xFF
	out EICRB, temp
	ldi temp,0xF0
	out EIMSK, temp
	sei

	;value of Y r29, r28 -as a counter on 4bytes
	ldi y1, 0x0A
	ldi y2, 0xFF

	;form the value in X
	ldi x1, 0x01
	ldi x2, 0x00

	;form the value in Z -for memory erase and usart receive
	ldi z1, 0x01
	ldi z2, 0x00

	;set flag to 0
	ldi flag, 0x00
	ldi temp, 0x00

	;deselect both spi devices
	sbi PORTB, 0
	sbi PORTE, 2

	rcall clear_mem

	ldi temp, 0x00
	mov sector, temp
main_loop:

	;value of Y r29, r28 -as a counter on 4bytes
	ldi y1, 0x12
	ldi y2, 0xFF

	;switch on the flag value
	cpi flag, 0x00
	breq main_loop

	cpi flag, 0x01
	breq sending_sound

	cpi flag, 0x02
	breq program_flash

	cpi flag, 0x03
	breq call_id_mem
```

```
      cpi flag, 0x04
      call read_flash

      cpi flag, 0x05
      call_id_mem: rcall id_mem

      cpi flag, 0x06
      call read_mem_microcontroller

      cpi flag, 0x07
      call render_sound_memory

      jmp main_loop

;sends sound in the memory of the microcontroller
sending_sound:
            ldi temp, 0x54
            out SPCR, temp

            ldi flag, 0x00
            dec y2
            cbi PORTB, 0
            cbi PORTE, 2

            ldi counter, 0x9F

            delay: dec counter
               cpi counter, 0x00
               brne delay

            ldi spidata, 0x00
            rcall SPI_MasterTransmit

            ld spidata, X+
            rcall SPI_MasterTransmit

            sbi PORTE, 2
            sbi PORTB, 0

            cpi y2, 0
            brne sending_sound
            cpi y1, 0
            breq main_loop
            dec y1
            ldi y2, 0xFF
            jmp sending_sound

jmp main_loop

program_flash:

      ;for flash memory the CPHA and CPOL must have the same value
      ldi temp, 0x50
      out SPCR, temp
```

```
ldi flag, 0x00

;form the value in X
ldi x1, 0x01
ldi x2, 0x00

;send data 256 bytes * 10: ldi y1, 0x0A
ldi y1, 0x0A

ldi memAddr, 0x00

program_page:

;---instruction write enable
      cbi PORTB, 0 ;chip select driven low
      cbi PORTE, 2

      ldi spidata, 0x06 ;write enable
      rcall SPI_MasterTransmit

      sbi PORTB, 0 ;chip select driven high
      sbi PORTE, 2

;---delay
      ldi counter, 0x07
      delay1: dec counter
           cpi counter, 0x00
           brne delay1

;---instruction page program
      cbi PORTB, 0 ;chip select driven low
      cbi PORTE, 2

      ldi spidata, 0x02 ;page program
      rcall SPI_MasterTransmit

      ;------------address
      ldi spidata, 0x12
      rcall SPI_MasterTransmit
      mov spidata, memAddr
      rcall SPI_MasterTransmit
      ldi spidata, 0x00
      rcall SPI_MasterTransmit

      ;set 256 bytes
      ldi y2, 0xFF

      dataSend:   ld spidata, X+
                  rcall SPI_MasterTransmit
                  dec y2    ;send 256B on one page
                  cpi y2, 0x00
      brne dataSend

      sbi PORTB, 0 ;chip select driven high
      sbi PORTE, 2
```

```
            rcall delay_tpp

            ldi usartdata, 0x14
            rcall USART_Transmit

            inc memAddr
            dec y1
            cpi y1, 0x00
      brne program_page
jmp main_loop



;transfer a flash memory sector to microcontroller memory
read_flash:

      rcall clear_mem

      ;for flash memory the CPHA and CPOL must have the same value
      ldi temp, 0x50
      out SPCR, temp

      ldi flag, 0x00

      ldi memAddr, 0x00

      ;form value of Y = 2700: ldi r29, 0x0A    ldi r28, 0x8C
      ldi y1, 0x0A
      ldi y2, 0xFF

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      cbi PORTB, 0 ;chip select driven low
      cbi PORTE, 2

      ldi spidata, 0x03 ;read data bytes
      rcall SPI_MasterTransmit

      ;------------address
      ldi spidata, 0x00
      rcall SPI_MasterTransmit
      mov spidata, memAddr
      rcall SPI_MasterTransmit
      ldi spidata, 0x00
      rcall SPI_MasterTransmit

      ;receive data bytes
      ldi spidata, 0x00
      receive_byte:
            rcall SPI_MasterTransmit

            in usartdata, SPDR
            st X+, usartdata
```

```
            rcall USART_Transmit

            dec y2
            cpi y2, 0
            brne receive_byte

            dec y1
            cpi y1, 0
      breq stop_reading
            ldi y2, 0xFF
            jmp receive_byte

stop_reading:
      sbi PORTB, 0 ;chip select driven high
      sbi PORTE, 2

      ldi usartdata, 0x33
      rcall USART_Transmit

      ;put the value 2700 in Y
      ldi y1, 0x0A
      ldi y2, 0x8C
      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

reti



;----------------------------------USART-----------------------------------------

USART_Init:
      ; Enable receiver and transmitter
      ldi temp, 0x98   ;(1<<RXCIE1)|(1<<TXEN1)|(1<<RXEN1);
      sts UCSR1B,temp

      ; Set frame format: 8data, 2stop bit, none parity
      ldi temp, 0x0e     ;0e parity  ;(1<<USBS0)|(3<<UCSZ00)
      sts UCSR1C,temp

      ; Set baud rate
      ldi temp, 0x33   ;0x19
      ldi spidata, 0x00
      sts UBRR1H, spidata
      sts UBRR1L, temp
ret

USART_Transmit:
      ; Wait for empty transmit buffer
      loop1:
            lds r20, UCSR1A         ;has the UDRE1 field
            sbrs r20,UDRE1    ;skip jump is bit is set. The UDREn flag indicates
if the transmit buffer (UDR) is ready to receive new data
                                    ;set after a reset to indicate that the
Transmitter is ready.
```

```
            rjmp loop1
      ; Put data (r16) into buffer, sends the data
      sts UDR1,usartdata
ret

USART_Receiver:
;     usart receive interrupt
;     it allows to write in the memory the word received
;     while another word is being received

cli
      lds usartdata, UDR1

      ;rcall USART_Transmit
      st X+, usartdata

      ldi temp, 0x00
      sts UCSR1A, temp
sei

ret

;----------------------------------SPI---------------------------------------
SPI_MasterInit:
      ;Set SS, MOSI and SCK output
      ldi temp, 0x07
      out DDRB, temp

      ;Enable SPI -SPE-, Master -MSTR-, set clock rate -SPI2X-
      ;set falling edge -CPHA-

      ldi temp, 0x54
      out SPCR, temp

      ldi temp, 0x01
      out SPSR, temp
reti

SPI_MasterTransmit:
      ;Start transmission
      out SPDR, spidata

      Wait_Transmit:
            ; Wait for transmission complete
            sbis SPSR, SPIF
            rjmp Wait_Transmit

      ;in usartdata, SPDR
      ;rcall USART_Transmit

reti

;--------------------------button interrupts routines-------------------------
button_push:
      ldi flag, 0x01
```

```
      ;put the value 2700 in Y
      ldi y1, 0x0A
      ldi y2, 0x8C

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ldi usartdata, 0x91
      rcall USART_Transmit
reti


button2_push:
      ldi flag, 0x02

      ;put the value 2700 in Y
      ldi y1, 0x0A
      ldi y2, 0x8C

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ldi usartdata, 0x92
      rcall USART_Transmit
reti

button3_push:
      ldi flag, 0x07

      ldi usartdata, 0x93
      rcall USART_Transmit

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ;value of Y r29, r28 -as a counter on 4bytes
      ldi y1, 0x0A
      ldi y2, 0x8C
reti

button4_push:
      ldi flag, 0x04

      ;mov usartdata, flag
      ;rcall USART_Transmit

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ;value of Y r29, r28 -as a counter on 4bytes
      ldi y1, 0x0A
      ldi y2, 0x8C
```

```
reti

;-----------------MicrocontrollerMemory-------------------

;read memory from address given in x number of locations given in y
read_mem_microcontroller:

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ldi y1, 0x02
      ldi y2, 0xFF

      loop_readMem:
            dec y2
            ld usartdata, X+
            rcall USART_Transmit
            cpi y2, 0
            brne loop_readMem
            cpi y1, 0
            breq fin
            dec y1
            ldi y2, 0xFF
            jmp loop_readMem
fin:

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ;value of Y r29, r28 -as a counter on 4bytes
      ldi y1, 0x0A
      ldi y2, 0x8C

reti

; clear the memory
clear_mem:  ;!form the z value before calling this
            dec y2
            st Z+, temp
            cpi y2, 0
            brne clear_mem
            cpi y1, 0
            breq endclear_mem
            dec y1
            ldi y2, 0xFF
            jmp clear_mem

endclear_mem:
      ;form the value in Z
      ldi z1, 0x01
      ldi z2, 0x00

      ;value of Y r29, r28 -as a counter on 4bytes
      ldi y1, 0x0A
```

```
      ldi y2, 0x8C

reti

;------------FlashMemory-------------

id_mem:
      ;for flash memory the CPHA and CPOL must have the same value
      ldi temp, 0x50
      out SPCR, temp

      ldi flag, 0x00

      cbi PORTB, 0
      cbi PORTE, 2 ;chip select driven low

      ldi spidata, 0x9F ;read data bytes
      rcall SPI_MasterTransmit

      ldi spidata, 0x00
      rcall SPI_MasterTransmit
      in usartdata, SPDR
      rcall USART_Transmit

      ldi spidata, 0x00
      rcall SPI_MasterTransmit
      in usartdata, SPDR
      rcall USART_Transmit

      ldi spidata, 0x00
      rcall SPI_MasterTransmit
      in usartdata, SPDR
      rcall USART_Transmit

      ldi spidata, 0x00
      rcall SPI_MasterTransmit
      in usartdata, SPDR
      rcall USART_Transmit

      sbi PORTE, 2 ;chip select driven high
      sbi PORTB, 0
reti

read_status:
      ;for flash memory the CPHA and CPOL must have the same value
      ldi temp, 0x50
      out SPCR, temp

      cbi PORTB, 0 ;chip select driven low
      cbi PORTE, 2

      ldi spidata, 0x05 ;read data bytes
      rcall SPI_MasterTransmit

      ldi spidata, 0x00
      rcall SPI_MasterTransmit
```

```
      in usartdata, SPDR
      rcall USART_Transmit

reti

;delay for page programming
delay_tpp:
      ;---delay----tpp = 5ms
            ldi counter, 0xFF
            ldi temp, 0xFF

            delay6:
                  dec counter
              cpi counter, 0x00
              brne delay6

                  cpi temp, 0x00
                  breq enddelay
                  ldi counter, 0xFF
                  dec temp
                  jmp delay6

enddelay:
            ldi counter, 0xFF
            delay2:
                  dec counter
                  cpi counter, 0x00
                  brne delay2

reti

;erase flash memory
bulk_erase:
      ldi usartdata, 0x91
      rcall USART_Transmit

      ;for flash memory the CPHA and CPOL must have the same value
      ldi temp, 0x50
      out SPCR, temp

      ldi flag, 0x00

      ;---instruction write enable
            cbi PORTB, 0 ;chip select driven low
            cbi PORTE, 2

            ldi spidata, 0x06 ;write enable
            rcall SPI_MasterTransmit

            sbi PORTB, 0 ;chip select driven high
            sbi PORTE, 2

      ;---delay
            ldi counter, 0x07
            delay7: dec counter
```

```
                cpi counter, 0x00
                brne delay7

            cbi PORTB, 0 ;chip select driven low
            cbi PORTE, 2

            ldi spidata, 0xC7 ;bulk erase
            rcall SPI_MasterTransmit

            sbi PORTB, 0 ;chip select driven high
            sbi PORTE, 2

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00
reti

sector_erase:

      ldi usartdata, 0x97
      rcall USART_Transmit

      ;for flash memory the CPHA and CPOL must have the same value
      ldi temp, 0x50
      out SPCR, temp

      ldi flag, 0x00

      ;---instruction write enable
            cbi PORTB, 0 ;chip select driven low
            cbi PORTE, 2

            ldi spidata, 0x06 ;write enable
            rcall SPI_MasterTransmit

            sbi PORTB, 0 ;chip select driven high
            sbi PORTE, 2

      ;---delay
            ldi counter, 0x07
            delay10: dec counter
                 cpi counter, 0x00
                 brne delay10

            cbi PORTB, 0 ;chip select driven low
            cbi PORTE, 2

            ldi spidata, 0xD8 ;sector erase
            rcall SPI_MasterTransmit

            ;------------address
            ldi spidata, 0x12
            rcall SPI_MasterTransmit
            ldi spidata, 0x00
            rcall SPI_MasterTransmit
            ldi spidata, 0x00
```

```
            rcall SPI_MasterTransmit

            sbi PORTB, 0 ;chip select driven high
            sbi PORTE, 2

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00


reti


;-----------------------------------------------------------------
render_sound_memory:
      ldi flag, 0x00

      ;form value of Y = 2700: ldi r29, 0x0A    ldi r28, 0x8C
      ldi y1, 0x0A
      ldi y2, 0xFF

      ;prepare memory

      ldi memAddr, 0x00
      ldi counter1, 0x00

      next_sound:

            ;select memory
            ldi temp, 0x50
            out SPCR, temp

            cbi PORTE, 2  ;mem

            ;access memory- read enough byte

            ldi spidata, 0x03 ;read data bytes
            rcall SPI_MasterTransmit

            ;------------address
            ldi spidata, 0x12
            rcall SPI_MasterTransmit
            mov spidata, memAddr
            rcall SPI_MasterTransmit
            mov spidata, counter1
            rcall SPI_MasterTransmit

            ;receive data bytes
            ldi spidata, 0x00
            rcall SPI_MasterTransmit

            in bytedata, SPDR

            ;deselect memory
            sbi PORTE, 2
```

```
            ;select amp
            ldi temp, 0x54
            out SPCR, temp

            cbi PORTB, 0

                  ldi counter, 0x7F

                  delay8: dec counter
                      cpi counter, 0x00
                      brne delay8

            ;send byte to amp

            ldi spidata, 0x00
            rcall SPI_MasterTransmit

            mov spidata, bytedata
            rcall SPI_MasterTransmit

            ;deselect amp
            sbi PORTB, 0

            dec y2
            inc counter1
            cpi y2, 0
            brne next_sound
            cpi y1, 0
            breq stop_send_sound_mem
            inc memAddr
            ldi counter1, 0x00
            dec y1
            ldi y2, 0xFF
      jmp next_sound

stop_send_sound_mem:

      ;form value of Y = 2700: ldi r29, 0x0A    ldi r28, 0x8C
      ldi y1, 0x0A
      ldi y2, 0xFF
reti
```

Project.asm:

```
.include "m64def.inc"

.org 0x0000
      rjmp    main
.org 0x000A
      rjmp    button_push
.org 0x000C
      rjmp    button2_push
.org 0x000E
      rjmp    listen_alfabet
.org 0x0010
      rjmp    listen_AB
```

```
.org 0x003C
        rjmp        USART_Receiver

.def usartdata = r16
.def spidata = r17
.def temp = r18

.def counter1 = r24
.def counter = r25

.def flag = r19
.def memAddr = r22 ;flash memory

.def temporary = r23

.def x1 = r27 ;access memory
.def x2 = r26

.def z1 = r31 ;clear memory
.def z2 = r30

.def y1 = r29 ;counter on 4bytes
.def y2 = r28

.def bytedata = r15
.def sector = r10

.def nbOfLetters = r12

main:
      ;initialisation

    ldi   r16, LOW(RAMEND)
      out   SPL, r16
    ldi   r16, HIGH(RAMEND)
    out   SPH, r16

      ; set direction of port E (buttons) for input
      ldi r19, 0x04
      out DDRE, r19

      ;init the usart communication
      rcall USART_Init
      ;init the SPI communication
      rcall SPI_MasterInit

      ;init the interrupts
      ldi temp,0xFF
      out EICRB, temp
      ldi temp,0xF0
      out EIMSK, temp
      sei

      ;value of Y r29, r28 -as a counter on 4bytes
      ldi y1, 0x00
```

```
	ldi y2, 0xFF

	;form the value in X
	ldi x1, 0x01
	ldi x2, 0x00

	;form the value in Z -for memory erase and usart receive
	ldi z1, 0x01
	ldi z2, 0x00

	;set flag to 0
	ldi flag, 0x00
	ldi temp, 0x00

	;deselect both spi devices
	sbi PORTB, 0
	sbi PORTE, 2

	rcall clear_mem

	;form the value in Z -for memory erase and usart receive
	ldi z1, 0x01
	ldi z2, 0x00

	ldi temp, 0x00
	mov sector, temp

	ldi counter1, 0x00
main_loop:

	;value of Y r29, r28 -as a counter on 4bytes
	ldi y1, 0x12
	ldi y2, 0xFF

	cpi flag, 0x00
	breq main_loop

	cpi flag, 0x05
	rcall listen_AB

	ldi usartdata, 0x77
	rcall USART_Transmit

	jmp main_loop


jmp main_loop

;------read memory
read_flash:

	rcall clear_mem

	;for flash memory the CPHA and CPOL must have the same value
	ldi temp, 0x50
	out SPCR, temp
```

```
	ldi flag, 0x00

	ldi memAddr, 0x00

	;form value of Y = 2700: ldi r29, 0x0A    ldi r28, 0x8C
	ldi y1, 0x0A
	ldi y2, 0xFF

	;form the value in X
	ldi x1, 0x01
	ldi x2, 0x00

	;ldi temp, 0x55
	;st X+, temp

	cbi PORTB, 0 ;chip select driven low
	cbi PORTE, 2

	ldi spidata, 0x03 ;read data bytes
	rcall SPI_MasterTransmit

	;------------address
	ldi spidata, 0x01
	rcall SPI_MasterTransmit
	mov spidata, memAddr
	rcall SPI_MasterTransmit
	ldi spidata, 0x00
	rcall SPI_MasterTransmit

	;receive data bytes
	ldi spidata, 0x00
	receive_byte:
		rcall SPI_MasterTransmit

		in usartdata, SPDR
		st X+, usartdata
		rcall USART_Transmit

		dec y2
		cpi y2, 0
		brne receive_byte

		;ldi temp, 0x55   ;e---mark each page
		;st X+, temp

		dec y1
		cpi y1, 0
	breq stop_reading
		ldi y2, 0xFF
		jmp receive_byte

stop_reading:
	sbi PORTB, 0 ;chip select driven high
	sbi PORTE, 2
```

```
      ldi usartdata, 0x33
      rcall USART_Transmit

      ;put the value 2700 in Y
      ldi y1, 0x0A
      ldi y2, 0x8C
      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

reti


;----------------------------------USART-----------------------------------------

USART_Init:
      ; Enable receiver and transmitter
      ldi temp, 0x98   ;(1<<RXCIE1)|(1<<TXEN1)|(1<<RXEN1);
      sts UCSR1B,temp

      ; Set frame format: 8data, 2stop bit, none parity
      ldi temp, 0x0e     ;0e parity  ;(1<<USBS0)|(3<<UCSZ00)
      sts UCSR1C,temp

      ; Set baud rate
      ldi temp, 0x33   ;0x19
      ldi spidata, 0x00
      sts UBRR1H, spidata
      sts UBRR1L, temp
ret

USART_Transmit:
      ; Wait for empty transmit buffer
      loop1:
            lds r20, UCSR1A         ;has the UDRE1 field
            sbrs r20,UDRE1    ;skip jump is bit is set. The UDREn flag indicates
if the transmit buffer (UDR) is ready to receive new data
                                    ;set after a reset to indicate that the
Transmitter is ready.
            rjmp loop1
      ; Put data (r16) into buffer, sends the data
      sts UDR1,usartdata
ret

USART_Receiver:
;     usart receive interrupt
;     it allows to write in the memory the word received
;     while another word is being received

cli
      lds usartdata, UDR1
      ;write memory:

      rcall USART_Transmit
      st X+, usartdata
```

```
      inc counter1

      ldi temp, 0x00
      sts UCSR1A, temp
sei

ret

;-----------------------------------SPI--------------------------------------
SPI_MasterInit:
      ;Set SS, MOSI and SCK output
      ldi temp, 0x07
      out DDRB, temp

      ;Enable SPI -SPE-, Master -MSTR-, set clock rate -SPI2X-
      ;set falling edge -CPHA-

      ldi temp, 0x54
      out SPCR, temp

      ldi temp, 0x01
      out SPSR, temp
reti

SPI_MasterTransmit:
      ;Start transmission
      out SPDR, spidata

      Wait_Transmit:
            ; Wait for transmission complete
            sbis SPSR, SPIF
            rjmp Wait_Transmit

      ;in usartdata, SPDR
      ;rcall USART_Transmit

reti
;--------------------------button interrupts routines-------------------------
button_push:
      ldi flag, 0x01

      ;put the value 2700 in Y
      ldi y1, 0x0A
      ldi y2, 0x8C

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ldi usartdata, 0x91
      rcall USART_Transmit
      ldi y2, 0x8C
reti

button2_push:
      ldi flag, 0x02
```

```
      ;put the value 2700 in Y
      ldi y1, 0x0A
      ldi y2, 0x8C

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ldi usartdata, 0x92
      rcall USART_Transmit
reti

button3_push:
      ldi flag, 0x01

      ldi usartdata, 0x93
      rcall USART_Transmit

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ;value of Y r29, r28 -as a counter on 4bytes
      ldi y1, 0x0A
      ldi y2, 0x8C
reti

button4_push:
      ldi flag, 0x05

      ;mov usartdata, flag
      ;rcall USART_Transmit

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      ;value of Y r29, r28 -as a counter on 4bytes
      ldi y1, 0x0A
      ldi y2, 0x8C
reti

clear_mem:  ;!form the z value before calling this
            ldi temp, 0x00
            dec y2
            st Z+, temp
            cpi y2, 0
            brne clear_mem
            cpi y1, 0
            breq endclear_mem
            dec y1
            ldi y2, 0xFF
            jmp clear_mem

endclear_mem:
```

```
      ;form the value in Z
      ldi z1, 0x01
      ldi z2, 0x00

      ;value of Y r29, r28 -as a counter on 4bytes
      ldi y1, 0x0A
      ldi y2, 0x8C

reti

listen_alfabet:


      ldi usartdata, 0x34
      rcall USART_Transmit

      ldi temporary, 0x00

      listen1:

            mov usartdata, temporary
            rcall USART_Transmit

            mov sector, temporary
            rcall render_sound_memory

            inc temporary
            cpi temporary, 0x12
            brne listen1

reti

listen_AB:
      mov nbOfLetters, counter1
      mov usartdata, counter1
      rcall USART_Transmit

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00

      read_data_comp:

            ld usartdata, X+
            rcall USART_Transmit

            cpi usartdata, 0x41
            brne compB

            ldi temp, 0x00
            mov sector, temp
            rcall render_sound_memory

            compB:
                  cpi usartdata, 0x42
                  brne compC
```

```
        ldi temp, 0x01
        mov sector, temp
        rcall render_sound_memory


compC:
        cpi usartdata, 0x43
        brne compD

        ldi temp, 0x02
        mov sector, temp
        rcall render_sound_memory

compD:
        cpi usartdata, 0x44
        brne compE

        ldi temp, 0x03
        mov sector, temp
        rcall render_sound_memory

compE:
        cpi usartdata, 0x45
        brne compF

        ldi temp, 0x04
        mov sector, temp
        rcall render_sound_memory

compF:
        cpi usartdata, 0x46
        brne compG

        ldi temp, 0x05
        mov sector, temp
        rcall render_sound_memory

compG:
        cpi usartdata, 0x47
        brne compH

        ldi temp, 0x06
        mov sector, temp
        rcall render_sound_memory


compH:
        cpi usartdata, 0x48
        brne compI

        ldi temp, 0x07
        mov sector, temp
        rcall render_sound_memory

compI:
```

```
        cpi usartdata, 0x49
        brne compL

        ldi temp, 0x08
        mov sector, temp
        rcall render_sound_memory

compL:
        cpi usartdata, 0x4C
        brne compM

        ldi temp, 0x09
        mov sector, temp
        rcall render_sound_memory

compM:
        cpi usartdata, 0x4D
        brne compN

        ldi temp, 0x0A
        mov sector, temp
        rcall render_sound_memory

compN:
        cpi usartdata, 0x4E
        brne compO

        ldi temp, 0x0B
        mov sector, temp
        rcall render_sound_memory

compO:
        cpi usartdata, 0x4F
        brne compP

        ldi temp, 0x0C
        mov sector, temp
        rcall render_sound_memory


compP:
        cpi usartdata, 0x50
        brne compR

        ldi temp, 0x0D
        mov sector, temp
        rcall render_sound_memory

compR:
        cpi usartdata, 0x52
        brne compS

        ldi temp, 0x0F
        mov sector, temp
        rcall render_sound_memory
```

```
            compS:
                  cpi usartdata, 0x53
                  brne compSh

                  ldi temp, 0x10
                  mov sector, temp
                  rcall render_sound_memory

            compSh:
                  cpi usartdata, 0x5B
                  brne compT

                  ldi temp, 0x11
                  mov sector, temp
                  rcall render_sound_memory

            compT:
                  cpi usartdata, 0x54
                  brne compTz

                  ldi temp, 0x12
                  mov sector, temp
                  rcall render_sound_memory

            compTz:
                  cpi usartdata, 0x5D
                  brne counter_handling

                  ldi temp, 0x13
                  mov sector, temp
                  rcall render_sound_memory


            counter_handling:

            dec nbOfLetters
            mov temp, nbOfLetters
            cpi temp, 0x00
            breq listen_AB_end
      jmp read_data_comp

listen_AB_end:
      ldi counter1, 0x00

      ;form the value in Z -for memory erase and usart receive
      ldi z1, 0x01
      ldi z2, 0x00

      rcall clear_mem

      ;form the value in X
      ldi x1, 0x01
      ldi x2, 0x00
reti

render_sound_memory:
```

```
ldi flag, 0x00

;form value of Y = 2700: ldi r29, 0x0A    ldi r28, 0x8C
ldi y1, 0x0A
ldi y2, 0xFF

;prepare memory

ldi memAddr, 0x00
ldi counter1, 0x00

next_sound:

      ;select memory
      ldi temp, 0x50
      out SPCR, temp

      cbi PORTE, 2  ;mem

      ;access memory- read enough byte

      ldi spidata, 0x03 ;read data bytes
      rcall SPI_MasterTransmit

      ;------------address
      mov spidata, sector
      rcall SPI_MasterTransmit
      mov spidata, memAddr
      rcall SPI_MasterTransmit
      mov spidata, counter1
      rcall SPI_MasterTransmit

      ;receive data bytes
      ldi spidata, 0x00
      rcall SPI_MasterTransmit

      in bytedata, SPDR

      ;deselect memory
      sbi PORTE, 2

      ;select amp
      ldi temp, 0x54
      out SPCR, temp

      cbi PORTB, 0

            ldi counter, 0x7F

            delay8: dec counter
                cpi counter, 0x00
                brne delay8

      ;send byte to amp

      ldi spidata, 0x00
```

```
            rcall SPI_MasterTransmit

            mov spidata, bytedata
            rcall SPI_MasterTransmit

            ;deselect amp
            sbi PORTB, 0

            dec y2
            inc counter1
            cpi y2, 0
            brne next_sound
            cpi y1, 0
            breq stop_send_sound_mem
            inc memAddr
            ldi counter1, 0x00
            dec y1
            ldi y2, 0xFF
      jmp next_sound

stop_send_sound_mem:

      ;form value of Y = 2700: ldi r29, 0x0A    ldi r28, 0x8C
      ldi y1, 0x0A
      ldi y2, 0xFF

      ;inc sector
reti
```