DJ HeMan

EITF40 - Digitala projekt
Mentor: Bertil Lindvall

by Robin Palmblad & Mattias Ahlström

Abstract

The project is about making a sound controller that has input sound from a normal 3.5mm cable and then processing this sound. Different beats could then be added to the sound with a keypad. The sound will then be played by connecting a 3.5mm cable to the output. A frequency spectrum of the sound will be shown on the display.

Table of Contents

1 Introduction

It's pretty common that when you go to a club there is a DJ there playing commonly known songs that has been modified in some way. For this project we thought it would be fun to create something similar to a DJ-board. First we had some extreme ideas about making a scratch and be able to modify the sound in numerous ways. After discussing this further we came to the conclusion that it would be best to make something a little bit simpler. We wanted to make a sound controller and also be able to display the characteristics of the sound. Some research was being made and we came up with the idea of making a sound controller with a 3.5mm cable connected to a sound source and another cable connected to a speaker. Beats would be able to be added to the sound and a frequency spectrum of the sound would be displayed on a LCD-display. This solution felt reasonable and doable so we started making a sound controller with these functions.

2 Requirements

The initial requirements weren't very specific. It only contains the minimum parts that could be needed for our implementation. When all parts of the construction was collected then the requirements became clear and is described below.

2.1 Hardware

The project is about making a sound controller that can display properties of the sound and add beats to the sound and therefore the following hardware parts were used:

- 3.5mm jack - for the input sound
- 3.5mm jack - for the output sound
- ATmega16 - the processor
- MM74C922 - 16-Key Encoder for the keypad
- 4x4 keypad - 16 buttons keypad
- TLC7524C - 8-BIT multiplying D/A converter for the output sound
- GDM-12864C - 128x64 graphical display

The ATmega16 has a built in A/D converter which could be connected to a 3.5mm jack for the input sound. The keypad is connected to the processor through a key encoder. The graphical display has a built in encoder and is therefore connected directly to the processor.

2.2 Software

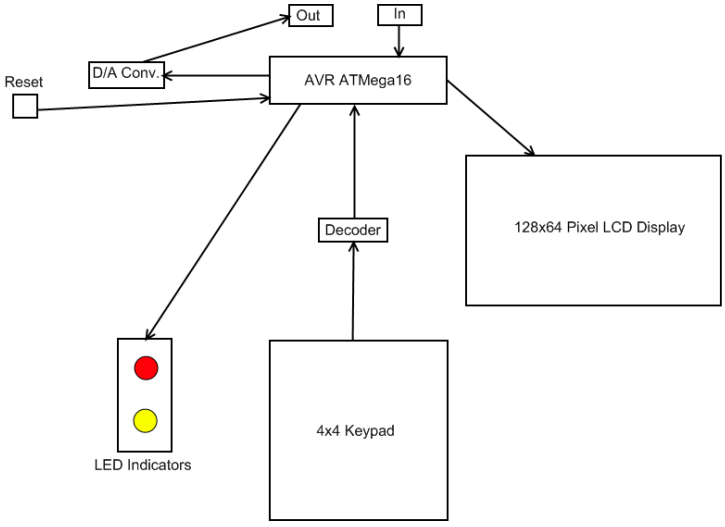
Input to the sound controller is from a sound source through a 3.5mm cable and the sound shall be played on a speaker connected with a 3.5mm cable. Samples of the sound will be made and its frequency response shall be displayed on a display. Beats can be added to the original sound by pressing the keypad. Beats can be played once, looped or in a pattern. Beats and patterns can be turned off by pressing the button again. The pace of the beats can be adjusted. Everything is in real time.

3 Construction

3.1 Hardware

Our construction consists of an ATmega16 microcontroller running at 16MHz. The microcontroller is connected to a 3.5mm jack for audio input, an external D/A converter which produces the output audio signal, a 4x4 keypad for user input and an LCD display with dedicated controller circuits. The keypad is connected to the microcontroller using a 4x4 decoder which converts the 8 signals used to control the keypad to a more manageable 4-bit data bus and interrupt signal. The D/A converter and the LCD display share a common 8-bit data bus which is routed to the correct device using their respective control signals.

The ATmega itself samples the input signal using the built-in A/D converter in differential mode (using both positive and negative terminals from the input signal instead of common ground) at 10x gain. Two LEDs on the board indicate if the input signal is too high (red LED flashing or constantly lit) or too low (yellow LED lit). Output from the D/A converter is connected to another 3.5mm jack where an external amplifier can be connected. Connecting headphones or speakers without a built-in amplifier directly to the 5V output is NOT recommended.



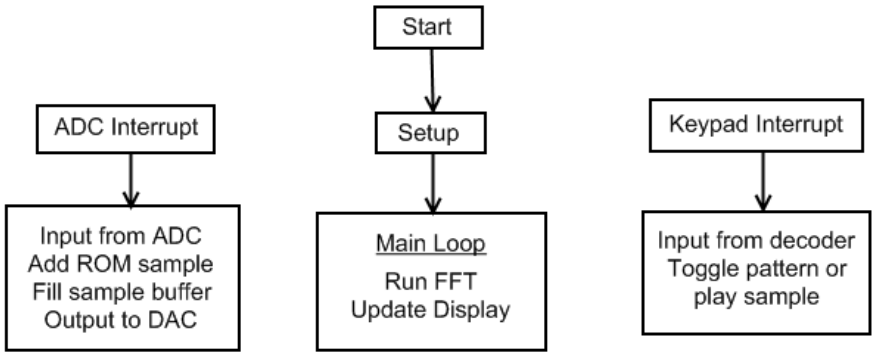
3.2 Software

The program starts in the Setup phase, where the microcontroller inputs and outputs are configured, the display is reset and turned on. The A/D converter is configured for free running mode where it will continuously sample the input signal and generate an interrupt when the conversion is complete. Using the ADC prescaler to divide the CPU frequency by x32 we were able to get a sample rate of approximately 38kHz (each conversion takes 13 ADC cycles), which suits our application very well.

At the end of the setup phase interrupts are enabled and the main loop kicks off, initially waiting for the sample buffer to fill from the ADC interrupt. When one of the two sample buffers is full (256 samples per buffer) the Fast Fourier Transform(FFT) function is executed on the buffer to produce the frequency spectrum to be displayed on the LCD. Once the display has been updated the sample buffer is returned to the ADC to be filled again.

The ADC interrupt routine reads the data registers of the A/D converter, steps through the pre-programmed patterns (if enabled) and adds any ROM samples which are currently playing. The data is then placed into the current sample buffer, rotating the two buffers as they fill up. Finally the status LEDs are updated and the data is written to the D/A converter.

The keypad interrupt routine simply reads the 4-bit value from the decoder and toggles the relevant state. Keys 0-3 trigger playback of each of the 4 ROM samples, keys 4-7 toggle simple repeating patterns while keys 8, 9 and C toggle more complex patterns. Keys B and F are used to increase or decrease the tempo (BPM) of these patterns. To avoid jitter in the sampling routine we only disable interrupts during a short time when the display update routine needs to access the data bus. In the final version of the code, interrupts are disabled for no more than 8 sequential instructions.



4 Problems

During the development, not entirely unexpected, we had some problems. It was especially hard in the beginning since we got a new problem as soon as we connected a new hardware part.

Our first problem was with just getting the debugger to work which didn't feel good at all. After checking with other groups which had gotten the debugger to work we asked our mentor to help us. It appears that the microcontroller was set to use an external clock, which we didn't have. The microcontroller was changed to use an internal clock and then it worked. This setting was changed back once we actually installed our external clock.

We used a LCD display which turned out to be more problematic than we thought. The display kept turning itself off every now and then. We tried to make it turn off by jiggling the cable but it didn't work so we weren't sure what the problem was. First we tried with other cables but the problem didn't go away and at last we changed display controller and with the new display controller it worked fine.

For our construction we needed three peripherals connected to the processor(key encoder, display, D/A converter) this led to that all pins of the processor were used and we needed more. Our first idea was to use a multiplexer but then we got to the better idea to use the same pins for both the display and the D/A converter.

Expectedly we had some problems with getting perfect sound as output. When we got sound we were able to hear what we played but it didn't sound especially good. We connected an oscilloscope and found that the negative values where over the positive. This is because the D/A converter can't handle negative values and therefore translates a negative value to a high positive value. To fix this a value was added to the output value that made the lowest value possible to be 0. After fixing the negative values the sound was pretty good but some noise could still be noticed. To remove this noise two 100nF capacitors were added, one at the input and one at the power supply for the A/D converter.

At the end of the work with the controller beats were added. Problems with the memory then occurred. First we had a problem that the data of the program couldn't fit into the RAM-memory. This was solved by modifying the beats and move them to the ROM-memory along with the sinus wave needed for FFT. Both the RAM- and the ROM-memory were now almost full and when we ran the program we still had some problems. The stack filled the RAM-memory so that strange things occurred. This was solved by optimizing the code and then the program ran again and with beats.

5 Add-ons

5.1 Added functions

A reset button was added to make debugging and restarting the program more simple.

Also for making the debugging easier LEDs were added to the construction. The LEDs lights when the output sound is maxed or when the sound has been off for a short time. It lights red for maximum volume and lights yellow for no sound.

To be able to adjust the volume a potentiometer was added to the construction. It was discovered that when the potentiometer wasn't set to maximum or minimum it produced noise to the output signal.

5.2 Further development

The current program fills the memory of the current processor, so to make any further development with the existing hardware will be difficult. However if the processor is changed to one with bigger memory and at least with the same clock frequency as the existing one improvements can be made.

- Some obvious improvements as adding more beats and patterns can be made.
- Changing between showing the audio loops patterns and the frequency spectrum on the screen.
- Make own patterns with the keypad.
- Add a scratch effect.
- Use better/more ADDA converters to allow 16-bit sound.
- Add support for stereo sound.

6 Discussion

6.1 Resulting construction

The project resulted in a nice sound controller that has input sound from a 3.5mm cable and sound out through a 3.5mm cable. A frequency spectrum that is shown on a graphical display. Beats that can be added to the sound by pressing buttons on a keypad. Lights that shows maximum output or no output. A volume controller.

6.2 Comments

We had some trouble starting out with unfamiliar and sometimes even broken hardware but in the end it turned out better than we had dared to hope for. The hardware works as intended, the software seems very stable and we fulfilled most of the original requirements.