EDA385

Project Proposal

Reverb

Martin Karlsson (et06mk2)

Daniel Malmgren (mt08dm6)

Mathias Bruce (d01mbr)

Theory

The reverb model that is going to be used is very simple. It uses multiple echoes to simulate the response of a room.

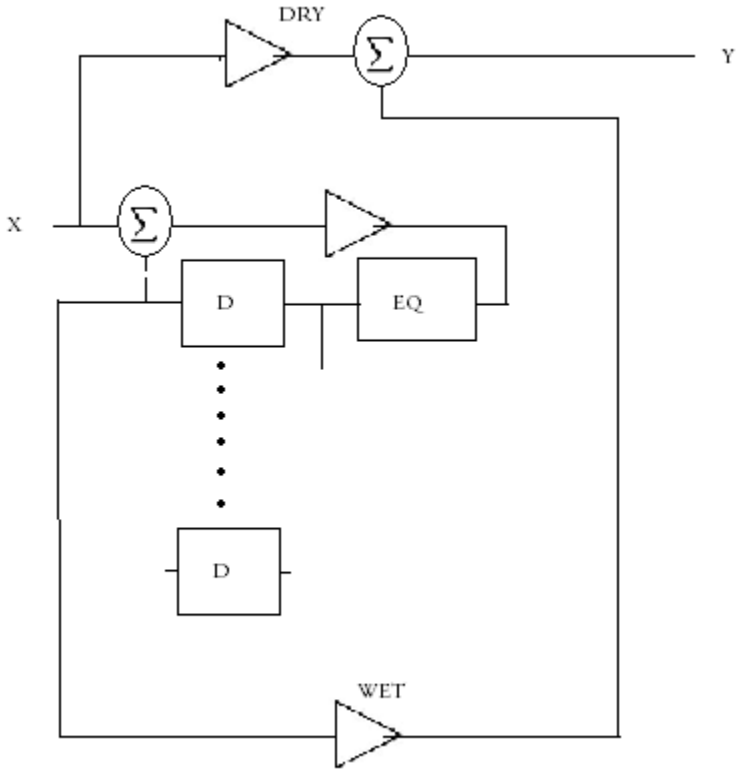

Figure.1. Reverb model

The amplifier Dry in the model decides how much of the original signal that passes through the device. The amplifier Wet in the model decides the amplitude of the reverb. The equalizer in the feedback models that the walls in a room absorbs certain frequencies. The different delay times are based on a function of the room size given by the user.

Implementation

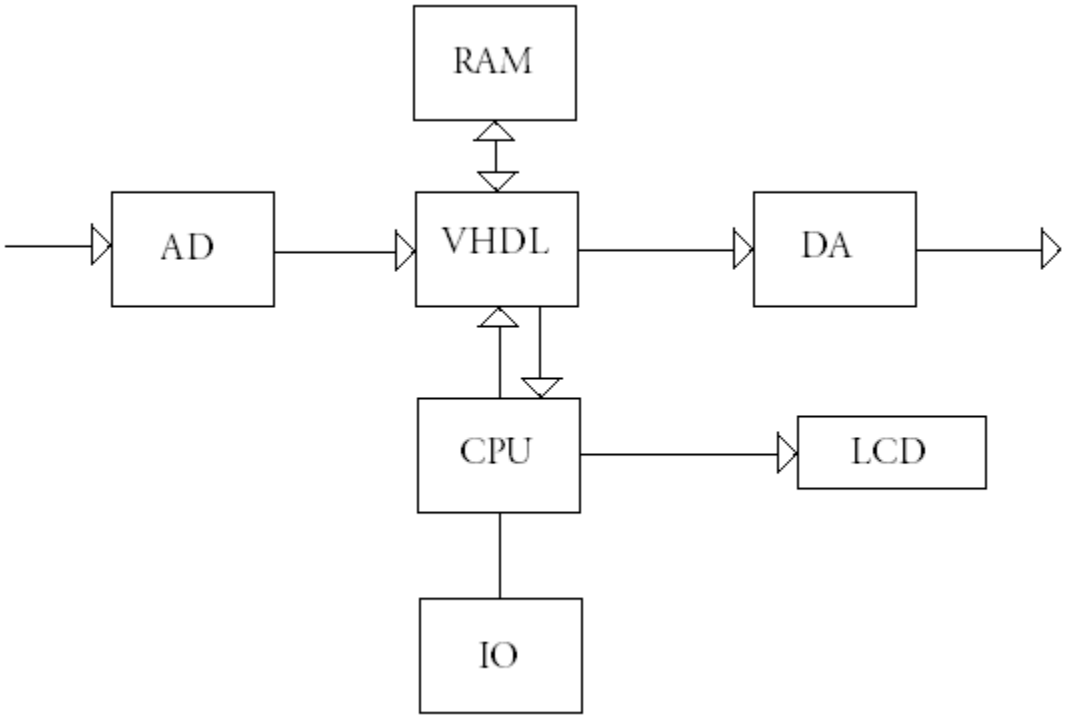

Figure.2. Block Diagram of the system

The audio path from sampling to output will be handled completely in hardware, i.e. from the input arrow into the A/D-converter to the output arrow from the D/A-converter. A PmodMIC expansion chip microphone (with integrated A/D-chip) will pick up the audio, which will be collected in 12-bit samples at 16 KHz by the path controller (The block named "VHDL" in the diagram). The sampling frequency is a result of our amplifier having a cutoff frequency of 16/2 = 8 KHz. After receiving the audio sample from the A/D-converter over a serial connection, the controller will store it into a ring buffer in RAM, for use in the output signal. Since we will be collecting 16.000 12-bit samples every second, the memory requirements will be 24 KB/s. As we probably will require a buffer of several seconds, we suspect this buffer would not fit in BRAM, and will therefore use the external 8 MB DRAM.

The output is generated by the controller, which collects a number of samples from the buffer in RAM and mixes them to produce the output signal. The mixing and signal processing is done according to the theoretical reverb model introduced earlier, with the

specific parameters for all processing given by user input via some controls connected to a MicroBlaze CPU. After mixing the samples and processing the signal, the controller outputs the result to a PmodAMP expansion chip amplifier with Pulse Width Modulation at a frequency of 16 KHz.

Also, some metric(s) will be transferred from the path controller to the CPU for displaying to the user, however exactly what metrics are not yet completely decided. A signal clipping indicator is one such possibility. The main system clock will be 50 MHz.

I/O and Software

This section covers the I/O and what software that is to be implemented on the Microblaze CPU core. The I/O handles user interface through PmodENC rotary encoder and the PmodCLS serial LCD module.

PmodENC

The PmodENC is a rotary encoder which uses a cam-type shaft which operates in contact with push-button switches. Turning the rotary encoder either to the right or to the left will close either A or B first, and thus generate a different signal pattern depending on which way the shaft is turned. The signal is shown in figure 2.

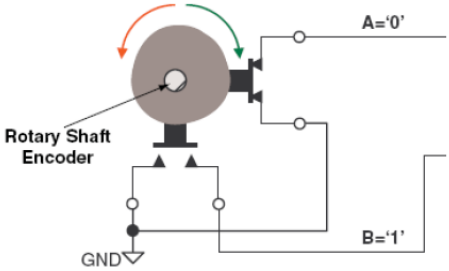

Figure 1 - Rotary encoder schematic.

The figure shows the signal pattern when the shaft is turned to the right. The principle of operation is the same when turned to the left with the only difference that B will be the first signal to rise and fall. The software must therefore implement a simple pattern analysis in order to determine which way the shaft is turned.

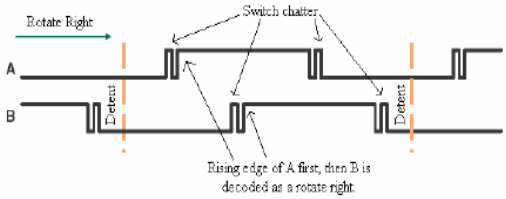

Figure 2 - Generated signal pattern.

PmodCLS

The PmodCLS is an LCD module which incorporates an Atmel ATmega48 microcontroller for display control. Communication is handled via an RS232 serial interface. Characters sent through the interface are displayed and certain sequences of escape characters are used for control purposes.

The display is able to show 2*16 characters, and every character consists of 8*6 or 8*5 pixels (The data-sheet is somewhat unclear on this matter).

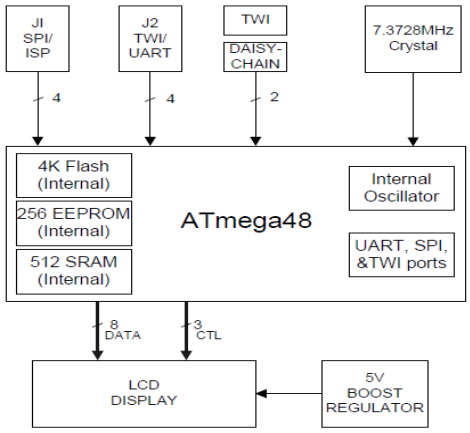

Figure 3 - PmodCLS block diagram.

Software

The software will be written in C and compiled for use with the Microblaze CPU core. The software will handle input from the PmodENC and output to the PmodCLS. Also, communication with the custom hardware design will be handled via an FSB (Fast Simplex Bus). This means sending data in order to change reverb settings, and receiving data to be displayed on the PmodCLS.

Timeplan

Week 3:

Be able to send audio through the system. That means have the AD and DA conversion done.

Initiated communication with the IO and LCD

Week 4

Establish communication with the ram and have a buffer.

Have a good function for the delay times implemented and IO and LCD working.

Week 5

Initiate communication between the VHDL-block and the CPU.

Get the hardware ready to receive parameters from CPU

Week 6

Have the parameter upload working and the hardware delays working.

Week 7

Finishing touches, writing report.