Hardware Accelerated Audio Manipulation Using An FPGA

Tom Johansson

Institute of Electronics and Information Technology, Lund University

March 6, 2018

Abstract

Hardware acceleration is a commonly used method in time critical or computationally heavy tasks. In this project, a two channel audio stream of 48 k samples/s with 16 bits/sample is being manipulated in various ways.

1 Introduction

The primary task of this project is to investigate the performance of an FPGA as an audio manipulator. However, in order to make the resulting product work well without needing external equipment, a board containing all the necessary components have been made. The components added to the board are as follows:

- FPGA (ARTIX-7 35T)
- Audio codec (TLV320AIC3101)
- Microcontroller (Atmega128)
- GPIO pins and buttons attached to the FPGA and microcontroller
- Two channel input and output RCA plugs

The end result should be a board which only needs power and audio input to generate a output signal modulated in the desired way.

2 Theory

2.1 FPGA

By using lookup tables instead of the multipurpose hardware habituating in microcontrollers and processors, FPGAs (*Fied Programmable Gate Arrays*) can often simulate actual hardware logic quite accurately. By being able to control every register to such detail, FPGAs are generally considered the second fastest digital construction type, only outranked by ASICs, which are far to expensive to create for anything other than mass production. Using its superior speed to microcontrollers, FPGAs are often used to accelerate time critical tasks where the used microcontroller can't perform well enough in a procedure known as *hardware acceleration*.

2.2 Audio codec

An audio codec is primarily used to sample an analog signal to a digital representation or to recreate a signal from digital data. The audio codec used in this project can produce digital representations with far better resolution than will be used in the project, both in the frequency and bit/sample domain. On the downside, however, the only mean of communication it supports to set its setting is I^2C. A communication protocol which relies on its ability to manage multiple devices on the same two wires. Since only one connection is established in this case, I^2C is unnecessarily tedious to implement.

3 Method

3.1 Preparation

Since the board on which the construction was going to be placed wasn't finished at the start of the project, a test board was set up to prepare the project as far as possible. The primary focus during this phase of the project was to get familiar with the Atmega128 microcontroller and to program it to handle the I^2C programming of the audio encoder.

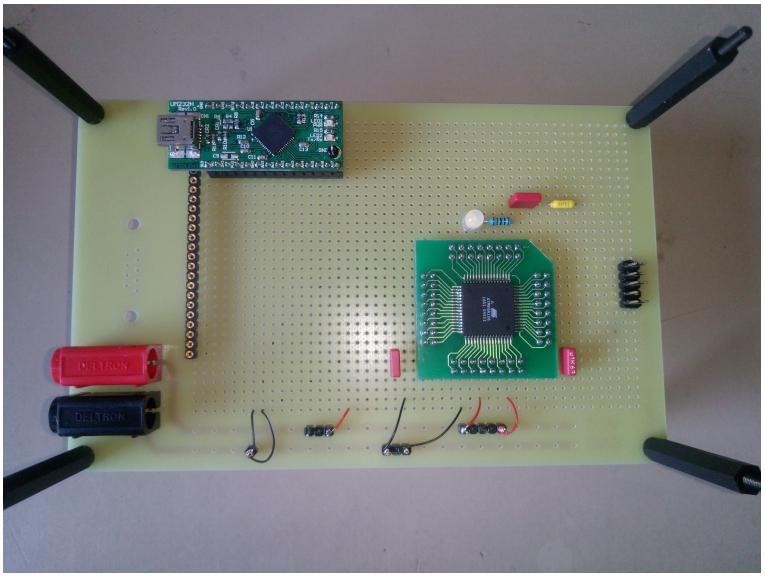

Figure 1: The preparation board.

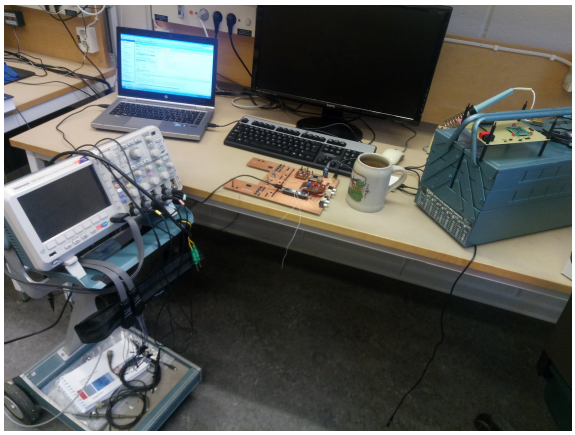

Figure 2: The lab setup during bud fixing with I^2C.

3.2 Programming the microcontroller and audio codec

Programming the audio encoder turned out to be a more time consuming task than expected. It was quickly decided that I^2C is to big of a protocol to implement within the scope of this project and that a library that implements the protocol should be used in the microcontroller instead. The first library that was imported was polling based, which means that the library is looking manually for ack bits in the communication. After a while, however, it was decided that it would be more elegant to use an interrupt based library instead, and so the old library became replaced. The new library wasn't working really as we wanted, so it was edited in some minor ways and some new helper functions was added to simplifying the sending and reading of data.

An annoying mistake that occurred here was due to miscommunication with Texas Instruments. They had labeled the device I^2C address of the audio to be 0x30 (0011 0000) which was simply because they helped us concatenate a write bit to the actual address 0x18 (0001 1000). We didn't know this so we added an extra write bit and wrote to 0x60 (0110 0000), which obviously never worked.

3.3 FPGA programming

3.3.1 IP

The FPGA programming was done using the Vivado 2017.2 tool. In it, two IP's are being used: The clock IP and the block memory generator IP.

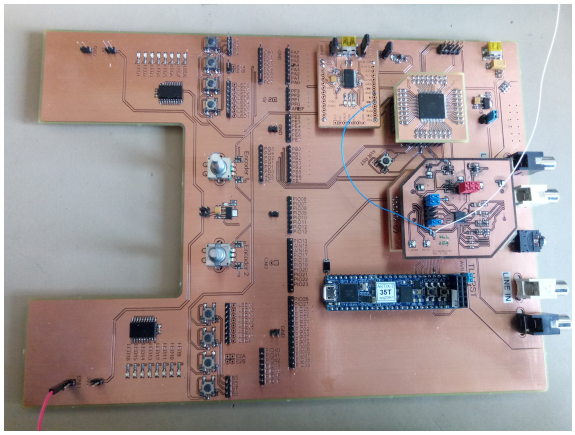

Figure 3: The complete final board.

The clock IP generates two clock frequencies: 100 MHz for the FPGA itself and 16 MHz for the clock input at the microcontroller and audio encoder.
The memory block generator generates a 32*48 000 bit large memory block. It is designed to contain a sample of both channels for each memory row and to have a depth of one second of sampling.

3.3.2 Code

Like most VHDL projects, also this one contains a top module to link the different signals and submodules together. All the logic to modify the audio lies within one of the submodules. In here, functionality to receive the audio data from the audio codec through I^2S has also been manually implemented.

4 Results & Discussion

So far, the only functionality that has been implemented is the ability to record audio and repeat it backwards. The primary reasons for that nothing more has been done yet are the time consuming problems with getting I^2C to work properly and general lack of time due to another ongoing project. With the hardest parts over, though, I'm optimistic to that a few more functions could be implemented before presentation.

Finally, disregarding of any end result, I've learned a tremendous amount about all the hardware I've used and can sincerely consider myself a better engineer after having done this project.