Audio transmissionsystem

Sara Källander, Josefine Strömsten

October 2017

Contents

1 Summary

During six laborations we where able to build a complete audio transmission system based on several different parts. The frequencies are altered with the help of two counters, one for the sampling and one for sending the information. The microphone was connected via an amplifier circuit, a low pass filter and a sample and hold circuit, to the A/D-converter. There is also headphones to which the information comes from the sender and goes through a serial-to-parallel code in the computer, an D/A-converter, a low pass filter and a power amplifier and then out through the headphones. These circuits were built during six labs and in the end it was tested by a collaboration with another group that sent sound through their microphone, which we could listen to in our headphones. It was not a very good sound but it worked.

2 Introduction

The digital sound transfer system was supposed to be able to function together with another groups project as either a sender or a receiver. It was built under certain specifications, it needed to work within 20-12000 Hz and have a 8 bits resolution. For everything to work together, the components needed to be dimensioned properly and calculated as an entire circuit, see the section about the D/A-converter for calculations. Every part of the circuit was supposed to be built and tested separately before connected with the rest. First, the D/A-converter was built and then the A/D-converter was built using the D/A as a base. The sample and hold, SAR and filters are there to make the circuit work as it should, and according to the specifications.

3 Subsystems

The different parts of the audio transmission system are all a small part of the entire circuit and it would not function as good without any of them. For more detailed information about the different parts, see the respective sections below, and for the code see appendix.

3.1 Counter

To be able to use different frequencies other than the standard clocks, two different counters for the two frequencies specified for the assignment where programmed. The code is built on two similar counters which creates different frequencies and pulse widths. One counter, T_s, was specified in the assignment to have a frequency of 960Hz and a 5 % duty-cycle. The other, T_b, was specified to 9600 Hz and a pulse width of only one clock pulse which is 20 ns with the 50 MHz clock used in the lab. To achieve these, 50 MHz was divided with 960Hz and 9600Hz respectively to get the values for the counters and then connected the triggers from the counters to square waves to see that it worked. For T_s the counter$_s$ goes up to 52083 for a full period time, with a positive trigger at 49479 which gives a pulse time of 5 %. Counter$_b$ goes up to 5208, with a positive trigger at 5207 for a pulse width of only one clock pulse. The code is attached in appendix.
In the tabular below are the measurements of the triggers T_S and T_B connected from the DE1-card to the breadboard. They were then connected to the oscilloscope to show the

analogue output. From this it was possible to see the values of amplitudes, period time, frequency and pulse width. As shown in the tabular below are the frequencies not exactly like the theoretical frequencies calculated for the program. It is however not a very big difference and it comes from the fact that the calculations are a bit rounded.

Signal	T_s	T_b
Amplitude [V]	3,36	4,36
Period time [ms]	1,04	0,104
Frequency [Hz]	959,7	9615
Pulse width [μs]	52	0,0202

From the measurements it was also clear that the pulse from T_B is a bit triangular, this is because it is not fast enough to be able to go all the way up before the trigger is set to zero again.

3.2 D/A converter

Below is a schematic of the D/A-converter used in the lab. In order for it to work according to the specifications, the resistances had to be dimensioned.

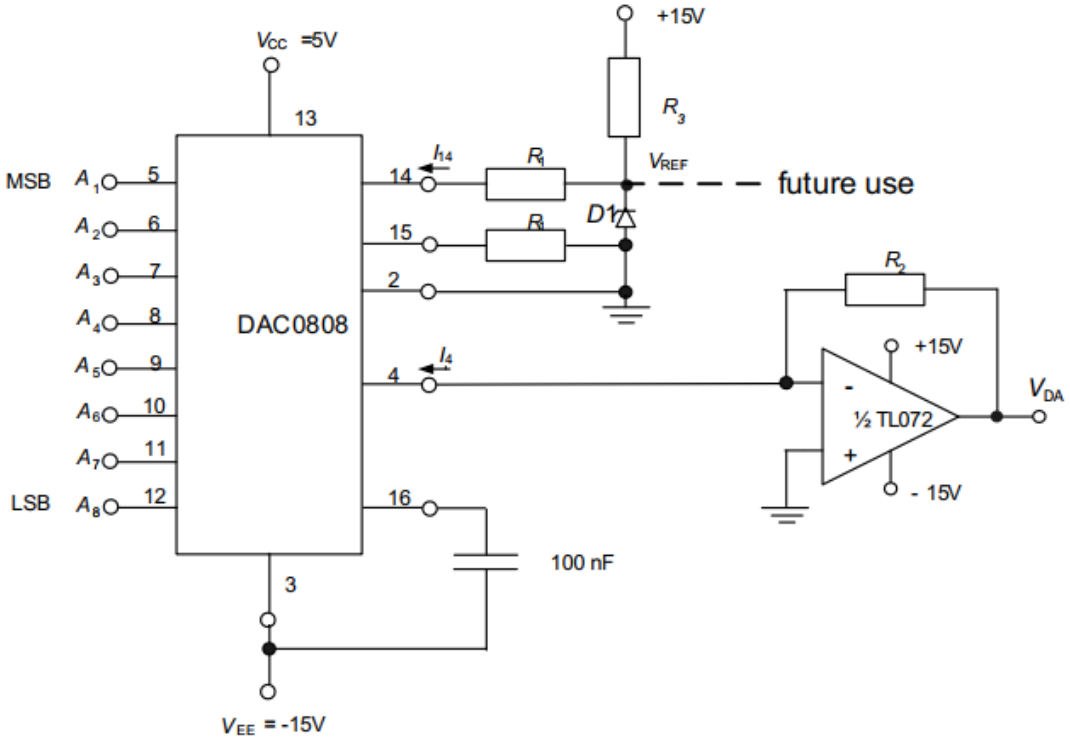

Figure 1: Schematic of the D/A-converter

R_1 was calculated by

$$R_1 = \frac{V_{ref}}{I_{14}} = \frac{5.6V}{2mA} = 2800\Omega$$

The closest actual resistance, from the E12-series in the lab is 2700 Ω.
With this R_2 could be calculated.

$$V_{DA} = \frac{V_{ref} R_2}{R_1} \times \frac{2^8 - 1}{2^8} => R_1 = \frac{R_2 V_{ref}}{V_{DA}} \times \frac{255}{256} = \frac{2700 \times 5.6}{10} \times \frac{255}{256} = 4840\Omega$$

And the closest actual value from the E12-series is 4700 Ω.

To calculate R_3 we needed to consider the current I_{14}, over R_1, which is recommended to be around 2 mA, and the current over the diode, D_1, which needs to be at least 10 mA to achieve a stable V_{ref} at 5.6 V. To achieve this and still have some margin for currents from later additions to the circuit, the current over R_3 had to be around 14 mA. Kirchoff's current law gives:

$$I_{14} = I_{R_3} - I_{D_1} => 14\text{mA} - 12\text{mA} = 2\text{mA}$$

Knowing this, the resistance was calculated with ohm's law.

$$R_3 = \frac{+15 - V_{ref}}{I_{R_3}} = \frac{15 - 5.6\text{V}}{14\text{mA}} = 671\Omega$$

With the closest resistance at 680 Ω.

After the circuit was built on a breadboard and the program loaded to the DE1-card, everything was checked to see if it worked correctly by altering the digital input from the card and measuring the analogue output. The results are as shown in the tabular below.

Digital control signal	*Calculated analog [V]*	*Measuered analog output [V]*
0	0	0,32
16	0,61	0,86
32	1,22	1,55
64	2,44	2,63
128	4,87	5,15
255	9,71	8

Calculated period time: 0,01 s, since it is supposed to count up or down 100 times/second.
Measured period time: 2,5 s, as can be seen in the picture from the oscilloscope in Figure 2 below. The difference comes from that the circuit is a lot slower than the theoretical calculations.

When measuring the slewrate with and without a capacitance in parallel with R_2, it was clear that the slew rate became a bit slower with the capacitor. Since the response to the changes from 1 to 0 in the output became better with less overshooting and the slew rate was small enough, the result was better with the capacitor.
Slew rate with capacitor: 9,26 V/us
Slew rate without capacitor: 5,43 V/us

This D/A-converter was at first unipolar but later changed into a bipolar converter by adding an resistance R_5 between V_{ref} and the negative input of the operational amplifier.

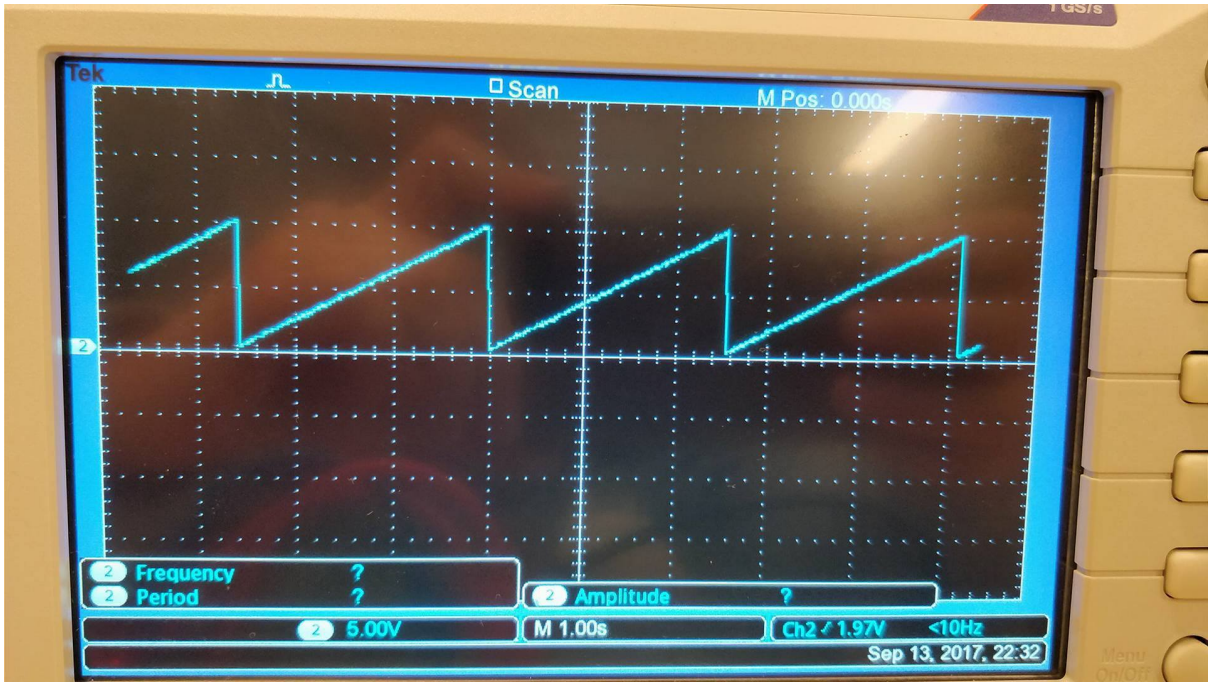

Figure 2: Results from measuring the binary counter. Here it is visualised how it counts up and then gets a overflow and starts over again.

This resistance was caluculated as below, where $I_{14} = 2\text{mA}$ and A is the value from the D/A output, where $A_{max} = 2^8 - 1$.

$$V_{DA} = R_2(I_4 - I_5) => I_5 = I_4 - \frac{V_{DA}}{R_2} = I_4 - 1.06\text{mA}$$

$$I_4 = I_{14}\frac{A_{max}}{2^8} = 1.99\text{mA} => I_5 = 1.93\text{mA}$$

$$R_5 = \frac{V_{ref}}{I_5} = 6021\Omega$$

In order to use a resistance from the E-12 series 6021 is rounded to 5600 Ω.
Now the D/A-converter is bipolar in the range ± 5V.

3.3 A/D converter

The A/D-converter is going to convert the voltage from the microphone into a digital signal Q.

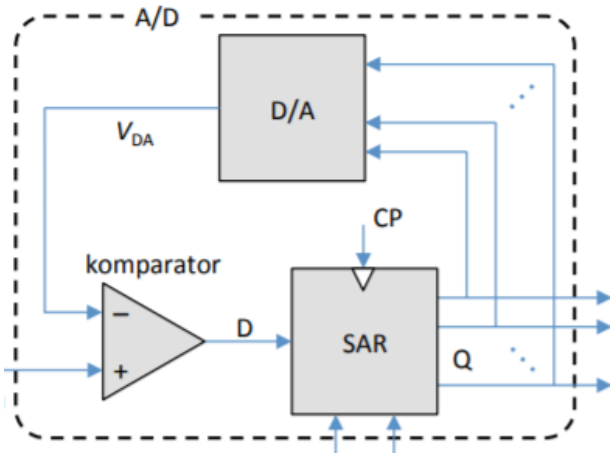

Figure 3: A/D-Converter

It is built by three different parts that is attached together, shown in Figure 3. The D/A-converter were used as a base for the A/D-converter, since they use the same functions only for different reasons. In order for it to work, it was expanded with an comparator (LM 311), and a successive approximation register (SAR), shown in figure 5 . The comparator gives an output of 0 V or 3.3 V depending on if the input is higher or lower than the signal from the D/A-converter. The SAR is coded in VHDL, according to the state machine in figure 4. It begins with guessing half the value of V_{max} and then successively approaches the actually value. If the value is higher than the guess, it will send out the number 1 and if it is lower it will send out number 0, it does this prcess eight times. The output will then send out eight bits in parallel.

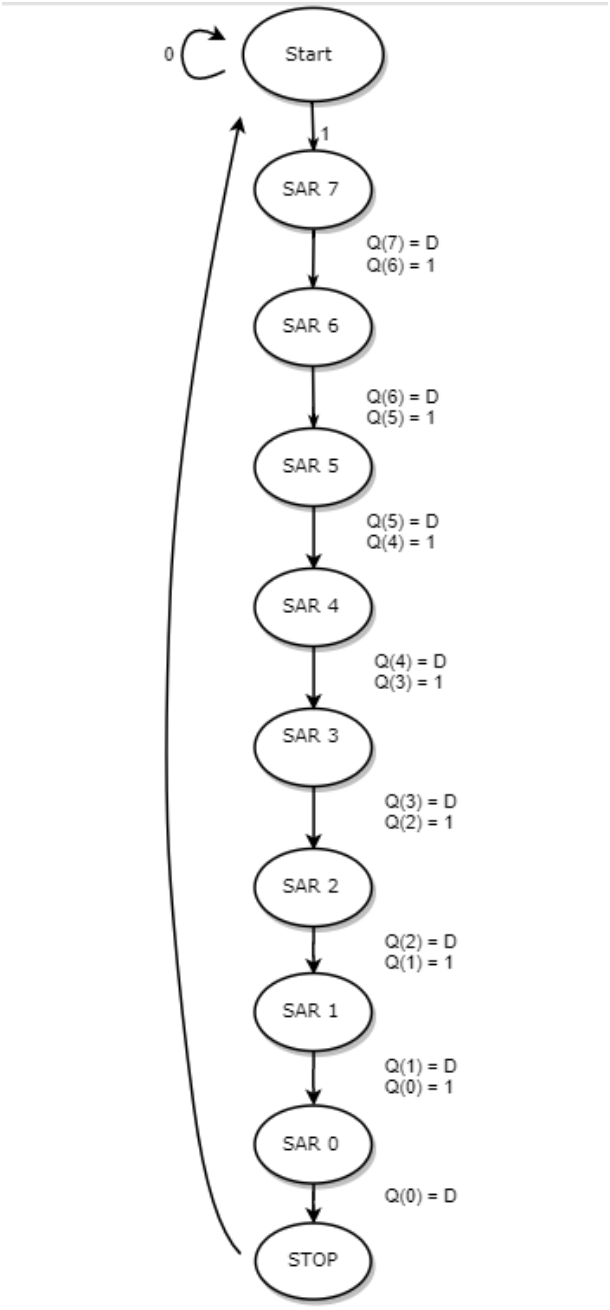

Figure 4: State machine for the successive approximation register

When we measured the time delay in the real circuit from Q to D it was 0,248 micro seconds.
To verify that the A/D-converter is working we changed the analog input to 5 V to see the digital output. The theoretical value is 1000 0000 and the result on the DE1 board showed 1000 0100.
The delay in the circuit was 0,41 mikroseconds, which makes the bitrate 2,44 Mbit/s.

3.4 Sample and hold

In order to achieve a correct A/D-conversion, the input to the converter needs to be constant while the comparator is working on the next bit. This is done by adding a sample and hold (S/R) circuit triggered by the signal T_s, before the comparator. The S/R can keep the value in to the comparator constant by loading a capacitor and thus "saving" the value. A screen shot of the measurements are shown in Figure 5. Here it is visualised how the sample and hold works. The trigger from T_s is calculated so that the sampling is synced with the minimum and maximum values of the input. As is shown in Figure 5, the output remains the same as the sampled input until the next trigger.

Figure 5: Screen shot of the input to and output from the sample and hold. Here, the input is yellow and the output is blue.

3.5 Serial transmitter

To be able to send the information that is coming from the microphone to the receiver it needs to be converted from parallel to serial data, which happens after the A/D-conversion. To make it into parallel-sequences we used the program Modelsim and programmed it in VHDL. Each time the trigger Tb is high the register shifts one step to the right and then takes the least significant bit and makes it into the serial output, see figure 6.

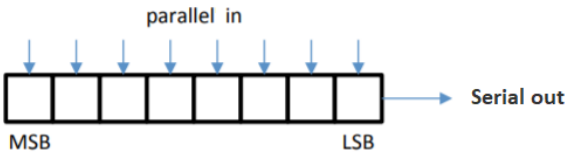

Figure 6: Serial to parallel

Before adding the rest of the serial transmitter we tested that this part in the code worked as it should by connecting the oscilloscope to see that our input matched the output, which it did.

The next step was to add a start and a stop bit into the serial output, which is used later for the synchronization in the serial receiver. Number 0 represent the start and number 1 represent the end of the sequence.

The second test with the A/D-converter we use a DB9 connector to communicate between a computer and the DE1 Board. By using a computer program (ds30 loader GUI) we can transmit our bit sequences and the program will show the value of the voltage in binary or in ascii. When the analog signal was 5 voltage the output in the program showed the binary code 1000 0100 and the received ascii text was 0x84.

3.6 Serial receiver

In this part it is very importent to get the synronizing right, the clock needs to be the same in the receiver and the transmitter. This is done by using start and stop bits. By doing this the program knows which eight bits it is supposed to convert from serial to parallel. The program follows the state machine that is shown in figure 7. When the program comes to the state "STOP" it is waiting for the start bit and doesn't continue until it has received the number 0. By doing this the receiver compensates even if the time isn't precisely matched.

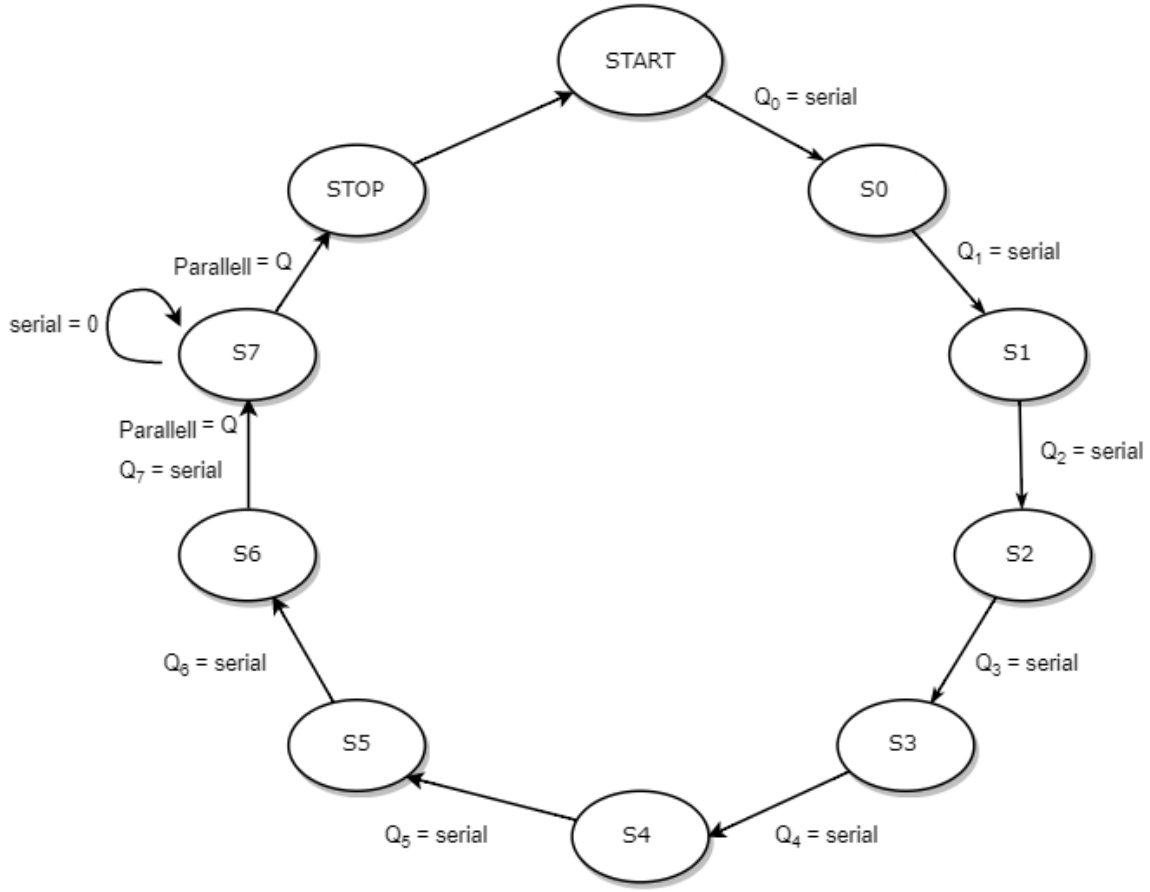

Figure 7: State machine for the serial to parallel converter

In this part the D/A-converter is modified from an unipolar into a bipolar, which is decribed in section 3.2. The calculated and the measured ouput from the A/D-converter is shown in the figur below. By using the the lamps on the DE1-board we can see the digital output, the result from different analog inputs is seen below.

Analog input DC (V)	*Calculated digital output*	*Measured digital output*
-5	00000000	00000000
-2	01001101	01000110
-1	01100110	01100000
0	10000000	01110100
1	10011010	10010101
2	10110011	10101111
3	11111111	11111110

To test the receiver we sent Ascii text trough a RS232 communication program from the computer to see if we got the right binary code. The result is below.

ASCII text DC	*Received bit sequence*	*Measured analog output (V)*
P	01010000	-1,6
?	0011111	-2,26
å	11100101	-2,26
!	00100001	-3,38
=	00111101	-2,32
A	01000001	-2,17

3.7 Audio amplifier

The audio amplifier is divided in two parts of the circuit, an amplifier for the microphone and a power amplifier for the effect output to the headphones. The microphone amplifier is needed to be able to A/D-convert the signal with a good quality and the power amplifier is used to increase the current to the headphones, since it is needed to achieve the sound pressure. To be able to transmit and receive data in a frequency that works in the system with its limitations, T_S and T_B where recalculated.

$$T_S = f_{sampel} = 2 \times f_{max} = 2 \times 12000 = 24kHz$$

$$Counter\_s = \frac{50MHz}{24kHz} = 2083$$

$$T_B = 10 \times T_S = 240kHz$$

$$Counter\_b = \frac{50MHz}{240kHz} = 208$$

3.7.1 Microphone amplifier

The inimpedance the microphone sees towards the amplifier was calculated by using a small signal schematic which simplified the calculations. Then C_1 is overlooked and R_1 and R_2 can be seen as parallel connected. Z_{in} is then calculated by

$$Z_{in} = \frac{R_1 \times R_2}{R_1 + R_2} = 9.1k\Omega$$

By having this impedance and knowing the sensitivity for the microphone under certain conditions, the sensitivity in this circuit could also be calculated.
The sensitivity of the microphones is -39 dBV/Pa @ 2kΩ Z_{in}, according to the datasheet.

$$-39dBV = 20log\frac{U_{eff}}{1V} => U_{eff} = 10^{\frac{-39}{20}} = 11.2mV_{eff}$$

$$I = \frac{U_{eff}}{R_{in}} = \frac{0.0112}{2000} = 5.6\mu A$$

Here, R_{in} is the resistance from the datasheet. It is used to find the current through the microphone during the sensitivity from the datasheet.

$$U = Z_{in} \times A = 9.1k\Omega \times 5.6\mu A = 50.96mV_{eff} = -25.9dBV$$

This means that the sensitivity is a bit higher with the use of a bigger resistance in the input.

The A/D converter works in the span ± 5 V which means that the maximum sound pressure the microphone can send without the converter being saturated is 114.85 dB SPL, according to the calculations below.

$$V_{AD} = \frac{5V}{\sqrt{2}} = 3.53V_{eff}$$

$$V_{mic} = \frac{3.53}{A} = \frac{3.53}{\frac{R_4 + R_3}{R_4}} = 0.32V_{eff} = -4.95dBV$$

The sensitivity is -25.8 dBV as calculated earlier.

$$-25.8 - (-4.95) = -20.85$$

1 Pa = 94 dB SPL according to the data sheet and 94 + 20.85 = 114.85 dB SPL which is the maximum sound pressure accepted from the microphone.

The amplifier was then built in LTspice and simulated to find the theoretical lower cut off frequency and the resting voltage over the microphone input.

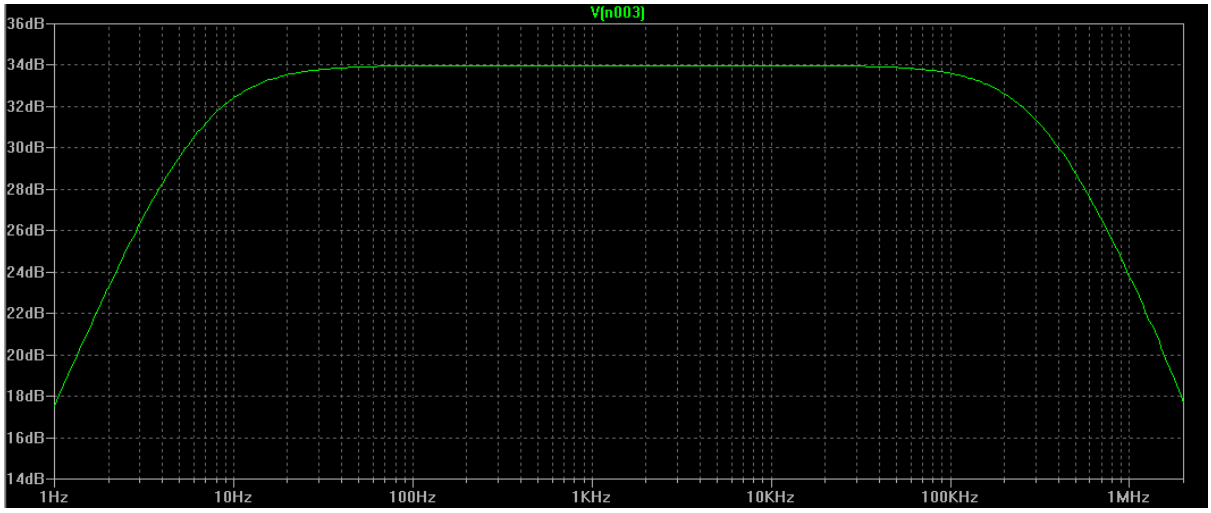

Figure 8: Simulation result of the microphone amplifier

The lower cut off frequency, f_{lower} is 6.6 Hz as can be seen in Figure 9 below. This is achieved by looking at the magnitude of the stable part and subtracting 3 dB. In this case it is about 34dB - 3 = 31dB. Here, the frequency is 6.6 Hz. This figure is also good for finding the amplification as it is the magnitude at the stable part, 34 dB.

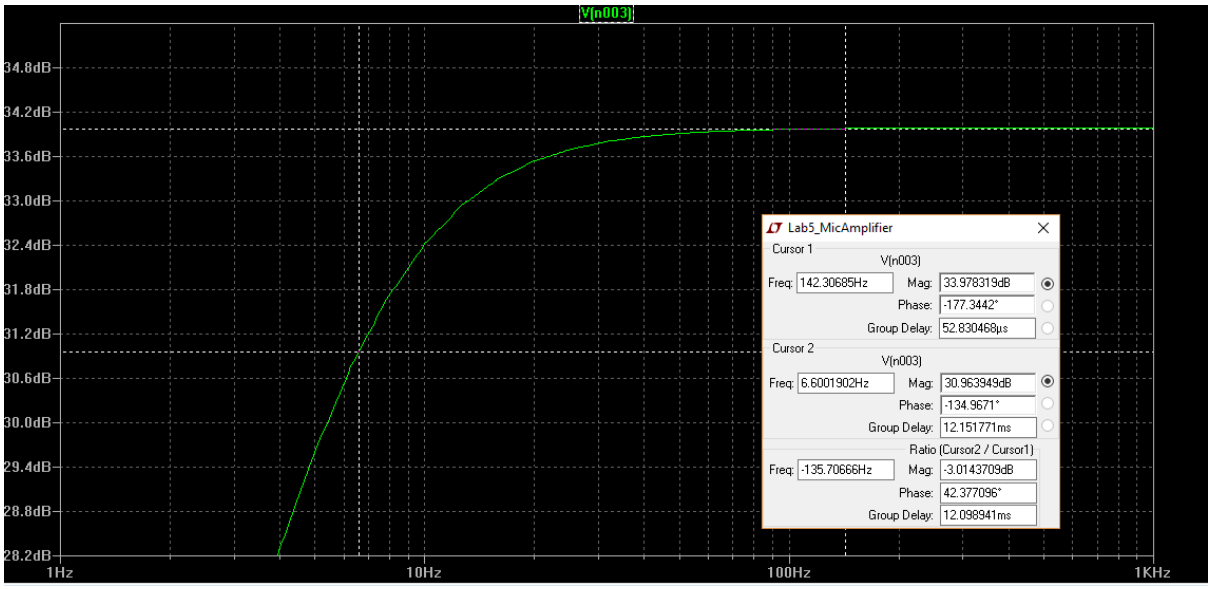

Figure 9: The lower cut off frequency of the microphone amplifier is found by finding the frequency at 3 dB less than the stable amplification.

The resting voltage over the input from the microphone was calculated with a voltage division over R_1 and R_2 to find the voltage drop over R_1. This was then subtracted from V_{ref} to find V_{mic}.

$$V_{mic} = \frac{R_1}{R_1 + R_2} \times V_{ref} = 0.51V$$

$$V_{mic} = V_{ref} - 0.51 = 5.1V$$

3.7.2 Power amplifier

To calculate the theoretical amplification of the power amplifier the circuit was divided into three parts, a voltage division over the first resistances, amplification over the operational amplifier and another voltage division over the last resistances. V_1 and V_2 are shown in the schematic in Figure 10 below.

$$A_1 = \frac{V_1}{V_{LP2}} = \frac{R_3//R_2}{R_1 + R_3//R_2} = 0.089$$

$$A_{OP} = \frac{V_2}{V_1} = \frac{R_5 + R_4}{R_5}$$

$$A_2 = \frac{V_2}{V_{OP}} = \frac{R_8}{R_7 + R_8} = 0.299$$

$$A_{tot} = 0.089 \times 4.5 \times 0.299 = 0.11 = -18.8dB$$

The amplification seems low but it is not supposed to amplify the voltage. It is the current to the headphones that need amplification.

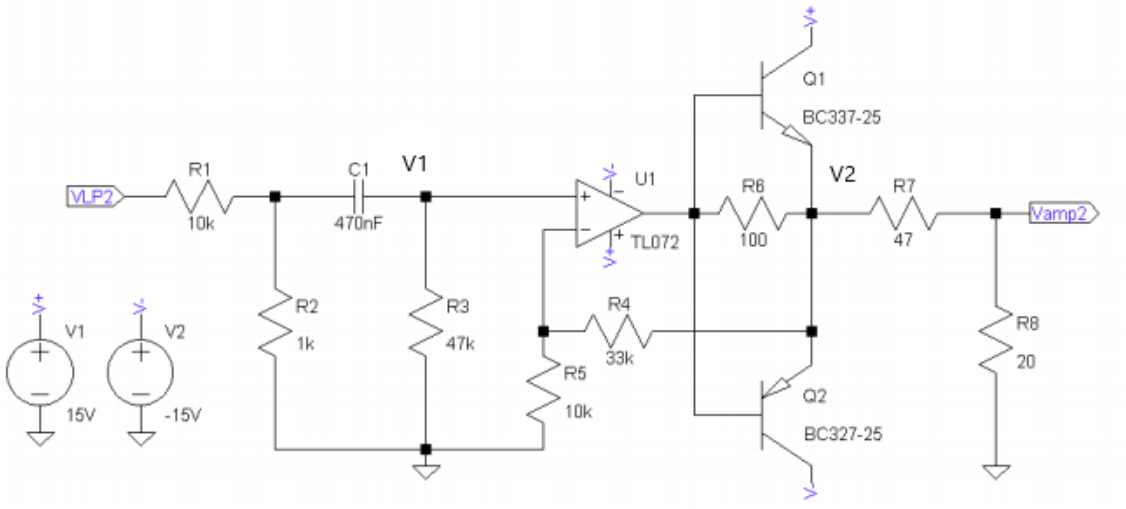

Figure 10: Schematic of the power amplifier

The maximum sound pressure which can be achieved from the power amplifier is 108.78 dB SPL and 7.56 mW, as shown in the equations below.

$$V_{amp,eff} = \frac{V_{LP2} \times A}{\sqrt{2}} = \frac{5 \times 0.11}{\sqrt{2}} = 0.389V_{eff}$$

$$P = \frac{V_{amp,eff}^2}{R_8} = \frac{0.151}{20} = 0.00756W = 7.56mW$$

$$10log(7.56) = 8.78dBm$$

$$100\text{dB SPL @ } 1\text{mW} => 1\text{mW} = 0\text{dBm} =>$$

$$100 + 8.78 = 108.78dBSPL$$

Then these calculations were controlled by using LTspice for a simulation. The results are shown in Figure 11 below, with the total bandwidth och the amplifier. The lower

cut off frequency is 7 Hz and the upper is 258 kHz, as shown in Figures 12 and 13. In Figure 14 the amplification is 0.1 which is close to the calculated amplification.

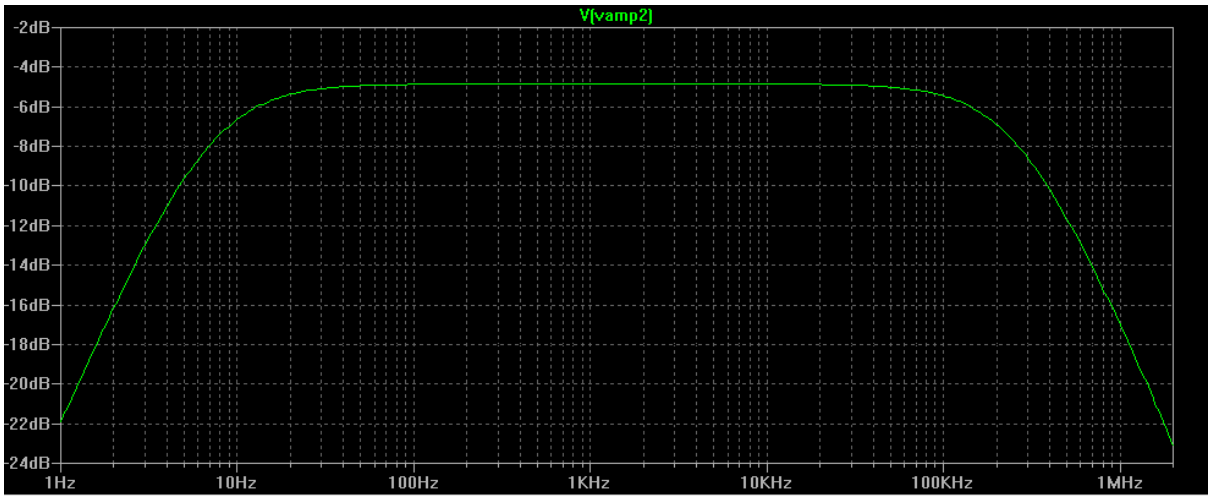

Figure 11: Result from the simulation of the power amplifier with frequencies and bandwidth

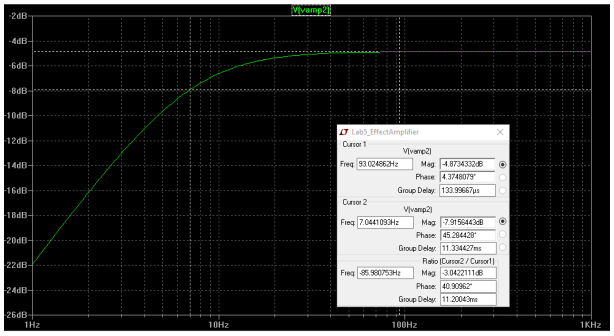

Figure 12: The lower cut off frequency of the power amplifier.

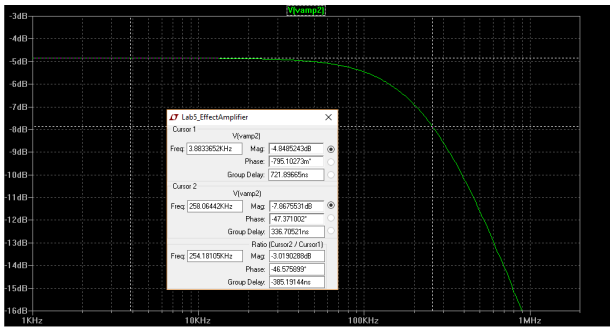

Figure 13: The upper cut off frequency of the power amplifier.

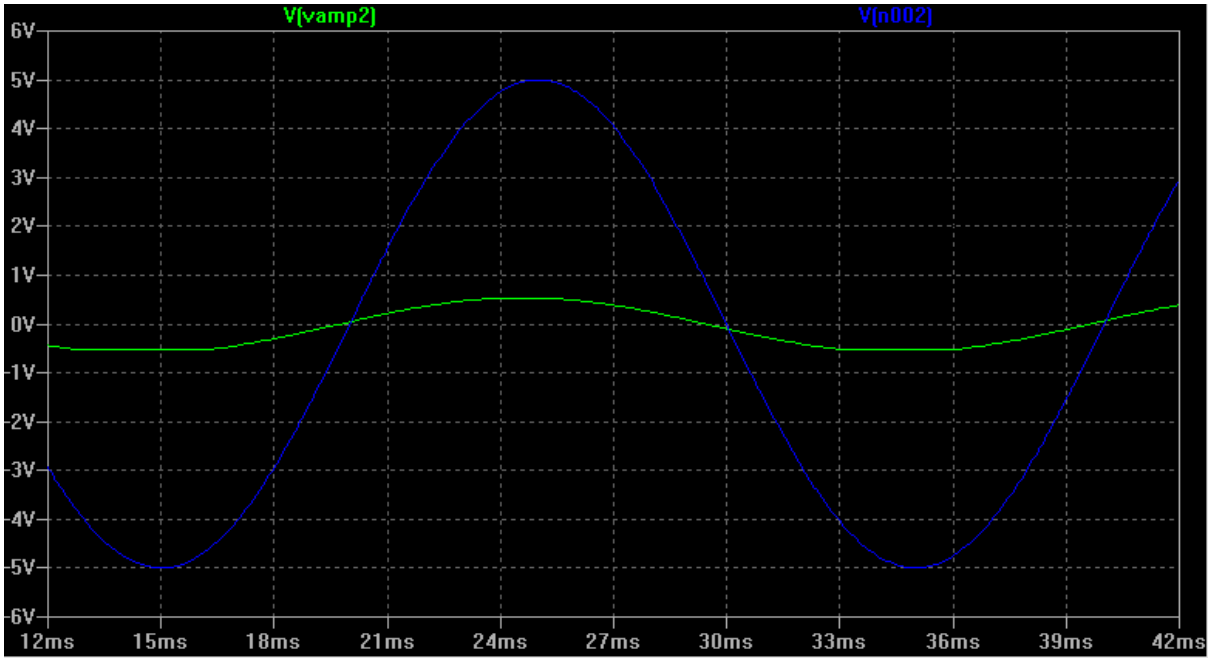

Figure 14: Result from the simulation of the power amplifier, over time. The blue curve is the input, V_{LP2}, and the green curve is the output, V_{amp2}.

When the circuit was built on the breadboard it was time to take some measurements to see that it worked as specified. In Figure 15 and the tabular below are the results from these measurements. Shown here are the input and output of the power amplifier and the amplification.

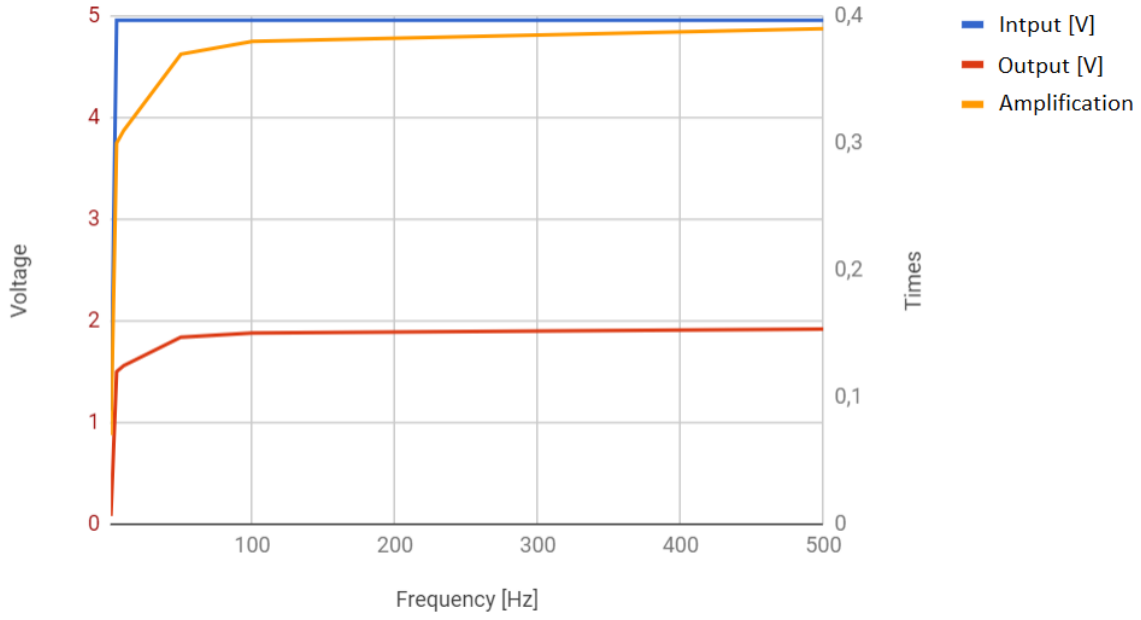

Figure 15: Here are the input (blue), output (red) and amplification (orange) of the power amplifier. Note that the amplification is scaled with times, on the right side of the diagram.

Frequency [Hz]	*Input [V]*	*Output [V]*	*Amplification [V]*
1	1.12	0.08	0.07
5	4.96	1.5	0.30
10	4.96	1.56	0.31
50	4.96	1.84	0.37
100	4.96	1.88	0.38
500	4.96	1.92	0.39
1 000	4.96	1.92	0.39
5 000	4.96	1.92	0.39
10 000	4.96	1.92	0.39
15 000	4.96	1.92	0.39
20 000	4.96	1.92	0.39

3.8 LP filter

The last part in this system is to build a fourth grade low pass filter for the transmitter and the receiver. It's for avoiding aliasing to the sample and hold circuit and —NÅGOT FÖR D/A-SKITEN–. To build these we have dimensioned resistors and capacitors needed for the Sallen-Key filter.

Since it's a fourth grade filter we need the Butterworth polynom which is:

$$H(s) = (1 + 0,765a + a^2)(1 + 1,848a + a^2)$$

The low pass formula with the Butterworth filter:

$$H_{LP}(s) = \frac{1}{P \times (s/W_{\ddot{o})}} = \frac{1}{(1 + 0,765 \times (s/W_{\ddot{o}}) + (s/\omega_{\ddot{o}})^2)(1 + 1,848 \times (s/\omega_{\ddot{o}}) + (s/\omega_{\ddot{o}})^2)}$$

The transfer function for the Sallen-key filters is:

$$H(s) = \frac{1}{1 + s(R_1 + R_2)C_2 + (s^2 R_1 R_2 C_1 C_2)} \times \frac{1}{1 + s(R_3 + R_4)C_4 + (s^2 R_3 R_4 C_3 C_4)}$$

With these equations we can dimension the components. If $C_1 = 1$ nF the calculated value for R_1 and R_2 will be:

$$(R_1 + R_2)C_2 = 0,765 \times \frac{1}{\omega_{\ddot{o}}} = 0,765 \times \frac{1}{2\pi 12000} = 10,15\,\mu s$$

$$(R_1 + R_2) = \frac{10,15\mu s}{1 \times 10^{-9}} = 10146\,\Omega$$

The closest value from the E-12 series is to build a serial circuit with two resistors with the values 4,7 kΩ and 5,6 kΩ.
The second capacitor is calculated with the following equation:

$$R_1 R_2 C_1 C_2 = \frac{1}{\omega_{\ddot{o}}^2} = 1,759 \times 10^{-10}$$

$$C_1 = 7nF$$

For this one we chose 6,8 nF from the E6 serie.

To find the values for the resistors and capacitator for the second sallen-key filter we used the same equations but changed the value for the first capacitator C_4 to 10 nF and calculated with 1,848 for the Butterworth polynom. With these values we got:
$R_3 = R_4$=1,2 kΩ
C_3 = 12,2 nF (10 + 2,2)
C_4 = 10

The figure 16 below shows how the circuit is built together and the next figure 17 shows the simulated cutoff frequency on 12 kHz in LT-spice with our calculated values.

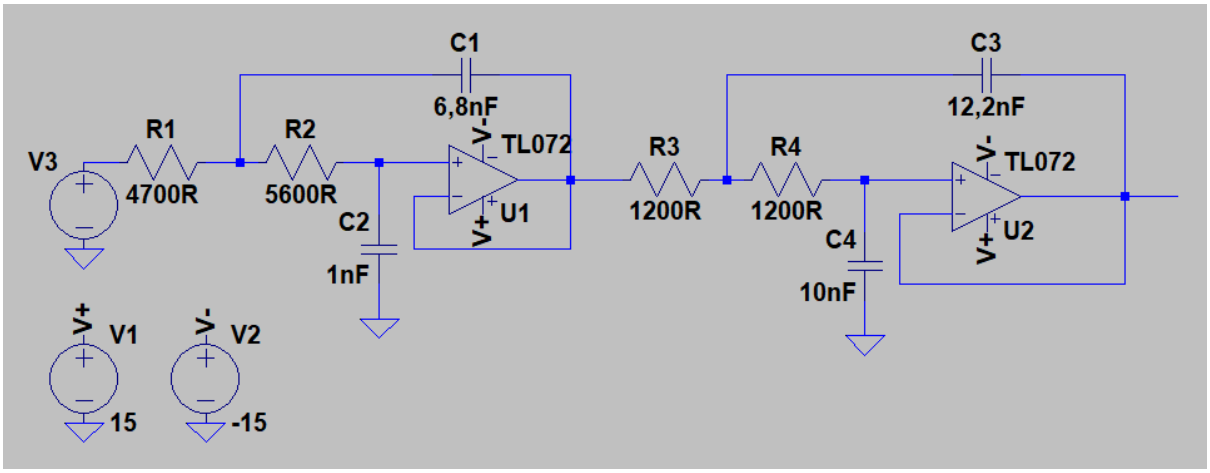

Figure 16: Sallen-key circuit

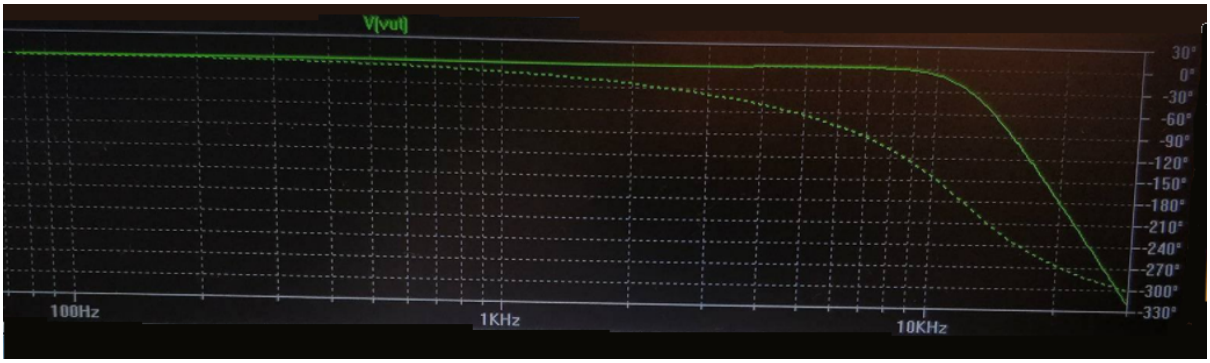

Figure 17: Cuttoff frequency

To examine how the damping ratio changes depending on the frequency we have measured the frequency between 1 KHz and 30 kHz. The result is showed in Figure 18.

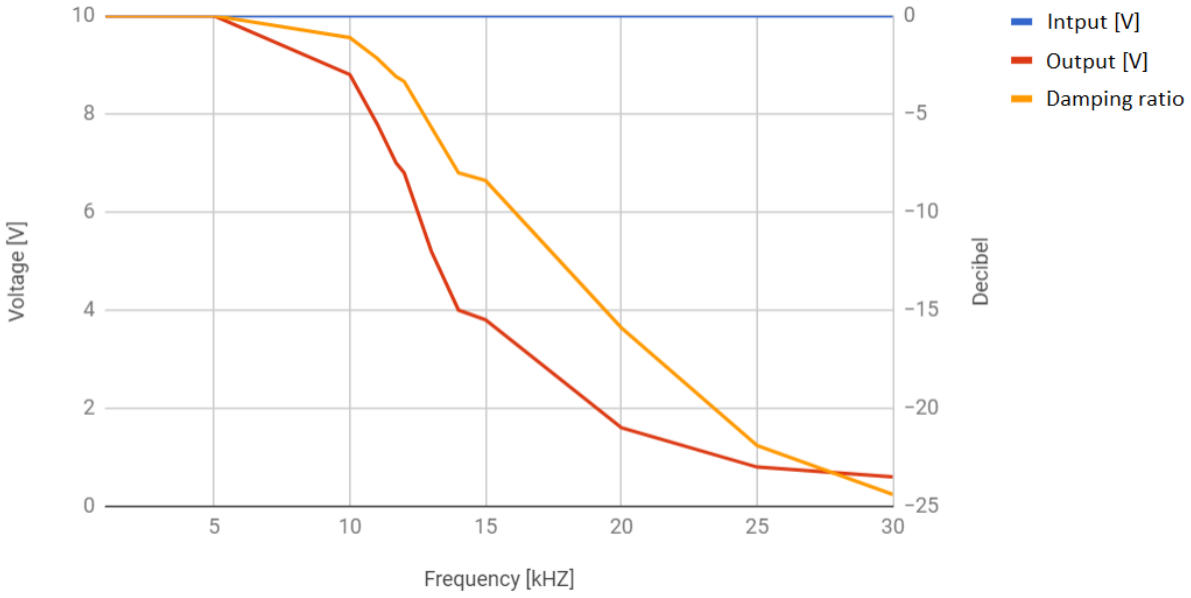

Figure 18: Sallen-key Low pass filter

Measured cutoff frequency: 11,7 kHz

4 Test and verification

To verify that the system is working we worked together with another group to see if one group could talk in the microphone and the other receive the sound. We tested our receiver, therefore we used the code for serial to parallel conversion and made sure of that we used the same rate as the transmitter used.
To see if the two systems was connected right the transmitter sent direct voltage and trough the oscilloscope and the lamps on the DE1-board we saw that we received the same voltage as the transmitter had sent After this we tested the system with different input values on the voltage.

Maximum input without distortion: 8 V_{pp}
Minimum input : 400 mV_{pp}
Transfer dynamics: 26 dB
Bandwidth: 6,8 Hz-20 kHz

The last test was to see if we actually got any sound trough the microphone into the headset. Everything worked as it should, you could hear the sound trough the system. The quality of the sound wasn't very good and there was a lot of noise, but there was no problem to hear what the other person said. Even when we took away three of the least significant bits you could still hear what the person on the other end said.

5 Discussion

It was a bit of a struggle to get the system working. We did not have one lab without searching for wrong connections, faulty parts or errors in the code. There was a lot of

small problems and mistakes that could have been easily avoided. We did our best to build the circuit in a way that makes it easy to debug and in the beginning it was quite easy. After a while it became a bit crowded on the breadboard and time was not on our side. The third lab, building the A/D-converter and serial transmitter, took about 10 hours in the lab-room to get working. The problem there was some small details in our code that made the counters give an positive trigger in the end of a pulse instead o the beginning. This resulted in some misunderstandings in the synchronisation. There were also some faulty connections, and we had a operational amplifier from the box of comparators. Note to self, always check that you have the right components.

In the test where we connected a phone to the sender circuit and listened to it through the headphones in the receiver part of the circuit, the sound was at least decent. Not very good but not to bad either. We could hear the music clearly and in another context it might have been a bad pair of headphones. Later, when we connected our receiver circuit with another groups sender, the sound was terrible. The filter couldn't remove the all of noise and we just got a headache from the sound. But we were able to hear what the other group said in their microphone without and problems so it worked. There might have been a few things we could do to the circuit to improve the quality but, as stated, it worked.

6 Reflection

This project has worked well overall in the end. There was enough time to prepare for the laborations but it took quit a while in the lab to get everything working. Some parts was more difficult than others, like how to simulate in LT-spice to get the answer and the preparations for lab 5 took longer time than expected.

At the lab station most of the laborations went smoothly, but there was a few times we got really stuck at some problem that took very long time so solve, especially in lab 3. It was very good that we could use the two extra scheduled laborations, by doing this we could actually understand the theory and not just collecting data. Both the lecture and the lab assistants was very helpful when you had got stuck on some problem. The lab equipment worked well, there was some breadboard that didn't work but that was the only thing that didn't work as expected.

7 Appendix

7.1 Code from Lab 1, counter and parallel-to-serial convertion

Preparation task 1.1
This is the code from the first assignment, which connects the switches to the led lights.

```
library ieee;
use ieee.std_logic_1164.all;

entity Lab1sharp is
port (
        LEDR: out std_logic_vector(9 downto 0);
        SW: in std_logic_vector(9 downto 0));
end entity;

architecture arch of Lab1sharp is
begin
process(SW)
begin

LEDR(9 downto 0) <= SW(9 downto 0);

end process;
end architecture;
```

Praparation task 1.2
This code takes a high frequency clock pulse and returns a lower frequency puls.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Uppg2 is
port(
clk: in std_logic;
reset: in std_logic;
squarewave_s: out std_logic;
squarewave_b: out std_logic;
trig_s: buffer std_logic;
trig_b: buffer std_logic);
end entity;

architecture arch of Uppg2 is
signal counter_s: std_logic_vector(16 downto 0);
signal counter_b: std_logic_vector(16 downto 0);

begin
```

```
process(clk, reset) --squarewave_s, trigger
begin

if reset='0' then
                counter_s <= (others => '0');
                trig_s <= '0';
else

                if rising_edge(clk) then
                counter_s <= counter_s+1;
                        if counter_s=49479 then   --49479
                                trig_s <= '1';
                        end if;
                        if counter_s=52083 then
                                trig_s <= '0'; --52083
                                counter_s <= (others => '0');
                        end if;
                end if;
end if;

end process;

process(clk, reset) --squarewave_b, trigger bit
begin
if reset='0' then
                counter_b <= (others => '0');
                trig_b <= '0';
else

                if rising_edge(clk) then
                        counter_b <= counter_b+1;
                        if counter_b=5207 then   --5207
                                trig_b <= '1';
                        end if;
                        if counter_b=5208 then
                                trig_b <= '0'; --5208
                                counter_b <= (others => '0');
                        end if;

                end if;
end if;

end process;

squarewave_s <= trig_s;
squarewave_b <= trig_b;

end architecture;
```

Preparation task 1.4

This code is a parallel- to-serial converter. The input is eight switches and the led lights are the output. The lights are supposed to shift one bit to the right by every trigger from the pulse from task 1.2 above.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;



entity Uppg3 is
port(
clk: in std_logic;   --50MHz clock
reset: in std_logic;
SW: in std_logic_vector(7 downto 0);
squarewave_s: out std_logic;
squarewave_b: out std_logic;
Dtx: out std_logic;
LEDR: out std_logic_vector(7 downto 0);
trig_s: buffer std_logic;
trig_b: buffer std_logic);
end entity;

architecture arch of Uppg3 is
signal counter_s: std_logic_vector(16 downto 0);
signal counter_b: std_logic_vector(16 downto 0);
--temp memory for shift of pin value
signal vector: std_logic_vector(7 downto 0);
begin
process(clk, reset) --squarewave_s, trigger Ts
begin

if reset='0' then
                counter_s <= (others => '0');
                trig_s <= '0';
                else

                if rising_edge(clk) then
                                counter_s <= counter_s+1;
                        if counter_s=49479 then
                        --49479 for 5 percent of 960Hz
                                trig_s <= '1';
                                counter_s <= counter_s+1;
                        end if;
                        if counter_s=52083 then
                        --52083 for 960Hz
                                trig_s <= '0';
```

```
                                counter_s <= (others => '0');
                                --reset of counter
                        end if;
                end if;
end if;

end process;

process(clk, reset) --squarewave_s, Trigger Tb
begin
if reset='0' then
                counter_b <= (others => '0');
                trig_b <= '0';
        else

                if rising_edge(clk) then
                        counter_b <= counter_b+1;
                        if counter_b=5207 then   --5206
                                trig_b <= '1';
                        end if;
                        if counter_b=5208 then  --5208 for 9600Hz
                                trig_b <= '0';
                                counter_b <= (others => '0');
                        end if;
                end if;
end if;

end process;

--shift pin values, output Dtx
process(trig_s, trig_b, reset, SW, clk)
begin
if reset='0' then
        vector <= SW;
else
  if rising_edge(clk) then
        if trig_b='1' then
                if trig_s='1' then
                        vector<= SW;
        ELSE

                vector(6 downto 0) <= vector(7 downto 1);

        end if;
end if;
end if;
end if;
end process;
```

```
LEDR <= vector;
Dtx <= vector(0);
squarewave_s <= trig_s;
squarewave_b <= trig_b;


end architecture;
```

7.2 Code from Lab 2, bit counter of the D/A-converter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity BitCounter is
port (
        LEDR: out std_logic_vector(7 downto 0);
        SW: in std_logic;
        --SW: in std_logic_vector(9 downto 0);
        --PINout: out std_logic_vector(9 downto 0);
        clk: in std_logic;   --50MHz clock
        reset: in std_logic;
        Tb: buffer std_logic;
        Ta: buffer std_logic);
end entity;


architecture arch of BitCounter is
signal counter_100Hz: std_logic_vector(18 downto 0);
signal counter_8bit: std_logic_vector(7 downto 0);
signal squarewave_100Hz: std_Logic;

begin
process(clk, reset) --squarewave_100Hz, trigger bit Tb
begin
if reset='0' then
                counter_100Hz <= (others => '0');
                counter_8bit <= (others => '0');
                Tb <= '0';
else
                if rising_edge(clk) then
                        counter_100Hz <= counter_100Hz+1;
                        if counter_100Hz=9 then   --499 999
                                Tb <= '1';
                        end if;
                        if counter_100Hz=11 then  --500 000
                                Tb <= '0';
                                counter_100Hz <= (others => '0');
                        end if;
                end if;
end if;

if rising_edge(clk) then
        if Tb='1' then
                if SW='1' then    --rakna uppat
```

```
                        counter_8bit <= counter_8bit+1;
                                if counter_8bit = 3 then
                                        Ta <= '1';
                                end if;
                                if counter_8bit = 5 then
                                        Ta <= '0';
                                        counter_8bit <= (others => '0');
                                end if;
        else Ta <= '0';

                --if SW(9)='0' then    --rakna nedat
--                                      counter_8bit <= counter_8bit-1;
--                      end if;
        end if;
end if;
end if;
end process;
squarewave_100Hz <= Tb;
end architecture;
--LEDR(7 downto 0) <= counter_8bit;
```

7.3 Code from Lab 3, A/D-converter and serial sender

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Lab3_3 is port(
clk: in std_logic;
reset: in std_logic;
D: in std_logic; -- input from comparator: higher or lower?
LEDR: out std_logic_vector(7 downto 0);
Q: buffer std_logic_vector(7 downto 0); -- parallel output
DRS:out std_logic;
Ts, Tb: buffer std_logic);
end entity;

architecture state_machine of Lab3_3 is
type StateType is (IDLE,SAR7,SAR6,SAR5,SAR4,SAR3,SAR2,SAR1,SAR0,STOP);
-- list of states
signal state: StateType;
signal vector: std_logic_vector(9 downto 0);
signal counter_s: std_logic_vector(16 downto 0);
signal counter_b: std_logic_vector(16 downto 0);

begin
process(clk, reset) --trigger Tb, 9600 Hz
begin

if reset='0' then
        counter_b <= (others => '0');
        --Tb <='0';

else
        if rising_edge(clk) then

                if counter_b=5207 then  --5207
                        Tb <= '1';
                  counter_b <= (others => '0');
                else
                        Tb <= '0';
                        counter_b <= counter_b+1;
                end if;
        end if;
        end if;
end process;
```

```
process(clk, reset) --trigger Ts, 960 Hz
begin

if reset='0' then
        counter_s <= (others => '0');
        --Ts<='0';
else
        if rising_edge(clk) then

                if counter_s=2603 then    -- 49479
                        Ts <= '0';
                        counter_s <= counter_s+1;
                elsif counter_s=52079 then  --52083
                        Ts <= '1';
                        counter_s <= (others => '0');  --reset of counter
                else
                  counter_s <= counter_s+1;
                end if;
        end if;
end if;
end process;


process(clk,reset, Tb, Ts)
begin

if reset='0' then
        state <= IDLE; --Samma

elsif rising_edge(clk) then
        if Tb = '1' then
                case state is
        when IDLE =>
                if Ts = '1' then
                Q <= "10000000"; -- half of the maximum value
                state <= SAR7;
                else
                state <= IDLE;
                end if;
        when SAR7 =>
                Q(7) <= D; -- MSB
                Q(6) <= '1';
                state <= SAR6;
        when SAR6 =>
                Q(6) <= D;
                Q(5) <= '1';
                state <= SAR5;
        when SAR5 =>
```

```
                Q(5) <= D;
                Q(4) <= '1';
                state <= SAR4;
        when SAR4 =>
                Q(4) <= D;
                Q(3) <= '1';
                state <= SAR3;
        when SAR3 =>
                Q(3) <= D;
                Q(2) <= '1';
                state <= SAR2;
        when SAR2 =>
                Q(2) <= D;
                Q(1) <= '1';
                state <= SAR1;
        when SAR1 =>
                Q(1) <= D;
                Q(0) <= '1';
                state <= SAR0;
        when SAR0 =>
                Q(0) <= D;
                state <= STOP;
        when STOP =>
                state <= IDLE;
                LEDR <= Q(7 downto 0);
        when others =>
                state <= IDLE;
end case;
--LEDR <= Q;
end if;
end if;
--end if;
end process;

--LEDR <= Q(7 downto 0);

process(Ts, Tb, reset, clk) --shift pin values, output Dtx
begin

        if rising_edge(clk) then
        if Tb='1' then
        if Ts='1' then
                vector(9)<='1';
                vector(8 downto 1) <= Q(7 downto 0);
                vector(0)<='0';
        else
                vector(8 downto 0) <= vector(9 downto 1);
        end if;
```

```
end if;
end if;
end process;
DRS<= vector(0);

end architecture;
```

7.4 Code from Lab 4, serial transmission

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Lab4 is port(
clk: in std_logic;
reset: in std_logic;
--D: buffer std_logic; -- input from comparator: higher or lower?
LEDR: out std_logic_vector(7 downto 0);
Q: buffer std_logic_vector(7 downto 0); -- parallel output
DRS,Dtx:out std_logic;
Ts, Tb: buffer std_logic;
D_async: in std_logic);
end entity;

architecture state_machine of Lab4 is
type StateType is (START,S7,S6,S5,S4,S3,S2,S1,S0,STOP); -- list of states
signal state, next_state: StateType;
--signal vector: std_logic_vector(9 downto 0);
--signal counter_s: std_logic_vector(16 downto 0);
signal counter: std_logic_vector(16 downto 0);
signal D_semisync: std_logic;
signal serial: std_logic;
signal next_Q, next_parallel, parallel: std_logic_vector(7 downto 0);

begin


----------------------------makes the input signal 'serial' synkronized-------

process(clk)
begin
if rising_edge(clk) then
D_semisync <= D_async;
serial <= D_semisync;
end if;
end process;
```

```
---------- bit clock, sync'd to start bit ----------

process(clk,reset)
begin
if reset='0' then
        state <= STOP;
        Q <= (others => '0');
        counter <= (others => '0');
elsif rising_edge(clk) then
        counter <= counter+1;
        if state=STOP and serial='1' then
                counter <= (others => '0');
        elsif counter=2603 then -- half-period 2604 clock cycles
                state <= next_state;
                Q <= next_Q;
                parallel <= next_parallel;
        elsif counter=5207 then -- bit clock 50MHz/5620 = 9600 Hz ------
                counter <= (others => '0');
        end if;
end if;
end process; -- bit clock

---------- update state and outputs ----------

process(state,serial,Q)
begin

next_Q <= Q;

case state is
when STOP =>
        next_state <= START;
when START =>
        next_state <= S0;
        next_Q(0) <= serial;
when S0 =>
        next_state <= S1;
        next_Q(1) <= serial;
when S1 =>
        next_state <= S2;
        next_Q(2) <= serial;
when S2 =>
        next_state <= S3;
        next_Q(3) <= serial;
when S3 =>
        next_state <= S4;
        next_Q(4) <= serial;
when S4 =>
```

```
            next_state <= S5;
            next_Q(5) <= serial;
when S5 =>
            next_state <= S6;
            next_Q(6) <= serial;
when S6 =>
            next_state <= S7;
            next_Q(7) <= serial;
when S7 =>
            next_parallel <= Q;
            if serial='0' then
                    next_state <= S0;
            else
                    next_state <= STOP;
            end if;
when others => -- undefined state
            next_state <= STOP;
end case;
end process; -- update state

LEDR <= Q;


end architecture;
```