Lab report SSY010 Electric Systems

2017-10-20

Gabriel Lindeby gabriel.lindeby@student.chalmers.se
Rikard Hellberg rikhell@student.chalmers.se

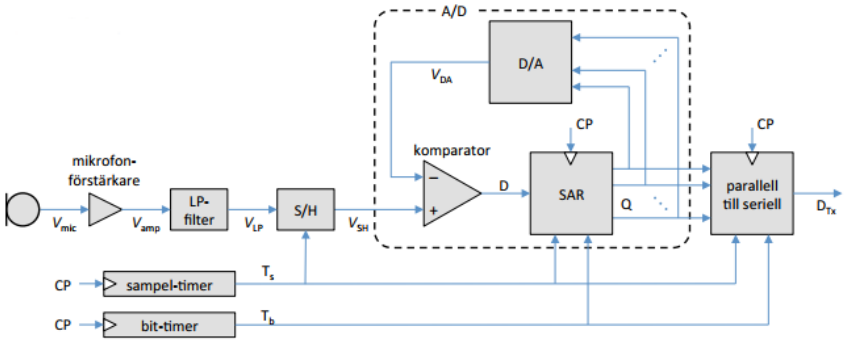
Abstract

This report describes a audio transmission system implemented using analog components and a fpga circuit. Furthermore, the report contains calculation, simulation and implementation of a microphone amplifier, analog to digital converter, digital to analog converter, low pass filter and a power amplifier. The report also covers sending and transmitting data serially over the RS-232 standard.

Introduction

The purpose of this project was to build and design a fully functional audio transmission system using the cyclone 2 fpga circuit together with analog components. The system should be able to sample an audio signals within the frequency of 20Hz to 12kHz, then transmit a digital representation serially to a receiver using the RS-232 standard. The receiver should then be able to playback the audio signal.

Transmitter

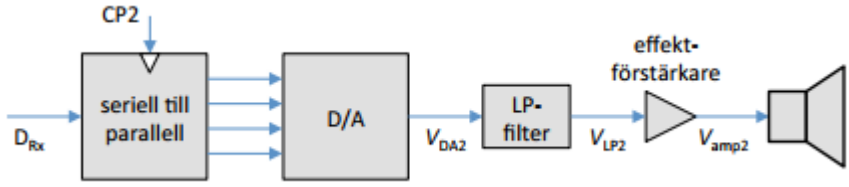


The transmitting part of the system consists of :

- Microphone and microphone amplifier.
- Low pass filter with an uppercut frequency of 12kHz.
- A/D converter consisting of an A/D converter, comparator and SAR.
- Parallel to serial converter.

The SAR and parallel to serial circuit is synthesised on the cyclone 2 fpga.

RECEIVER



The receiving part of the system consists of:

- D/A converter
- Low pass filter with an uppercut frequency of 12kHz
- Power amplifier with an power amplification of 7.17
- Headphone with an impedance of 20Ω sensitivity of 100 dB SPL/mW.

Subsystems

Counter

In advanced digital electronics systems a clock pulse often used to trigger system functionality regularly. This is also the case in this audio transmission system. The DE1 development board is equipped with a crystal oscillator that is producing a square wave with a frequency of 50 MHz. The signal from this oscillator will be referred to as clk50. 50 MHz clock pulse is too fast for the electronics in this system and different frequencies are needed for sample-timer and bit-timer.

This is solved by implementing two counters, one for sample-timer T_s and one for bit-timer T_b. A counter has a variable that is increased by one every time clk50 goes from a low 0 to a high 1. When a desired value of the variable i reached the output signal of the counter is toggled and the process repeats, generating a squarewave with a variable frequency that can be used as a clock.

T_b should have a frequency of 9600 Hz.
Iterations to achieve a frequency of 9600 Hz: $\frac{50K}{9.6K} = 5208$. T_b signal is low when the counter is between 0 and 5207. When it reaches 5207 T_b signal is high, then it starts from 0 again.

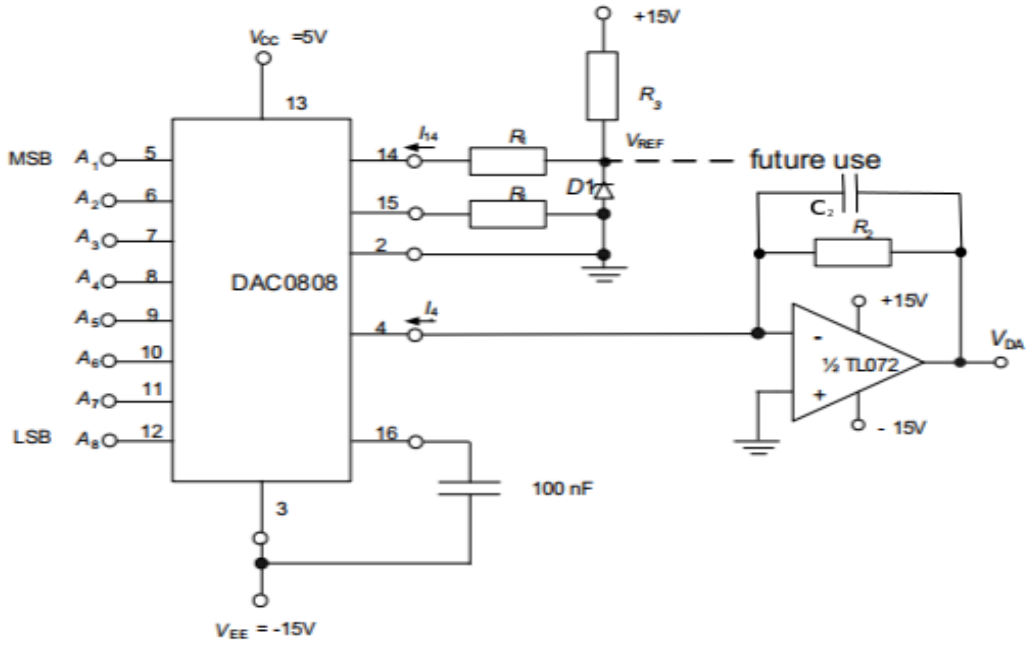
T_s should have a frequency of 960 Hz and a duty cycle of 5%.
Iterations to achieve a frequency of 960 Hz: $\frac{50K}{960} = 52083$, this value has to be 10 times T_B. Therefore this value was rounded to 52080. Duty cycle of 5%: $52083 \times 0.05 = 2604$. T_s is high between 0 and 2603 then set to low. When the counter reaches 52079 T_s is set to high and starts from 0 again.

Measurements from oscilloscope:

Signal	T_s	T_b
Amplitud	3,8 V	900 mV
Period time	1.04 ms	104 µs
Frequency	961.5 Hz	9.6 kHz
Pulse width (half max)	50 µs	36 ns

D/A converter

In this audio transmission system there are two digital to analog converters. One is part of the analog to digital converter and the other one before the LP filter. A D/A converter takes a digital input value and outputs a voltage depending on the input value. In this system the digital input consists of 8 bits where the binary value of 255 will result in the highest voltage output and 0 the lowest.



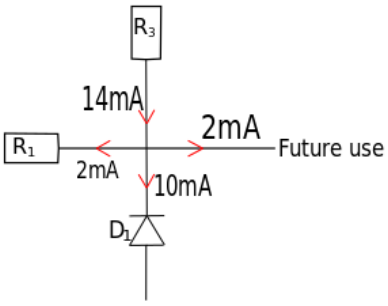
Specifikations

- Current through I_{14} should be 2mA
- V_{DA} between 0 - 10V
- Current through R_3 not ower 20mA
- V_{Ref} = 5,6V, current through D_1 should be a minimum of 10mA
- Components connected to future use will draw a maximum of 2mA
- 0V = logic 0, 3.3V = logic 1

$$R_1 = \frac{V_{Ref}}{I_{14}} = \frac{5.6V}{2mA} = 2.8k\Omega \; from \; E12 \gg 2.7k\Omega$$

$$R_2 = \frac{V_{DA} \cdot R_1 \cdot 256}{V_{Ref} \cdot A} = \frac{10V \cdot 2.7k\Omega \cdot 256}{5.6V \cdot 255} = 4840\Omega \; from \; E12 \gg 4.7k\Omega$$

$$R_3 = \frac{15V - V_{Ref}}{I_{R3}} = \frac{15V - 5.6V}{14mA} = 671\Omega \; from \; E12 \gg 560\Omega$$



Relation between input signal A and V_{DA}:

$$V_{DA} = \frac{V_{Ref} \cdot R_2}{R_1} \cdot \frac{A}{256}$$

These are the measured and calculated output values of V_{DA} given digital input signal. The vhdl code SW0_9_to_LEDR0_9 was used to do these measurements.

Digital input signal	Calculated output value (V)	Measured output value (V)
0	0	0.003
16	0.69	0.607
32	1.219	1.215
64	2.45	2.427
128	4.87	4.49
255	9.71	9.66

The vhdl code eight_bit_coutner.vhd is designed to output an 8 bit value to the D/A. The value starts from 0 and increases to 255 and starts over or from 255 down to 0 depending on the state of switch 9. This creates a sawtooth wave with a frequency of 100Hz.

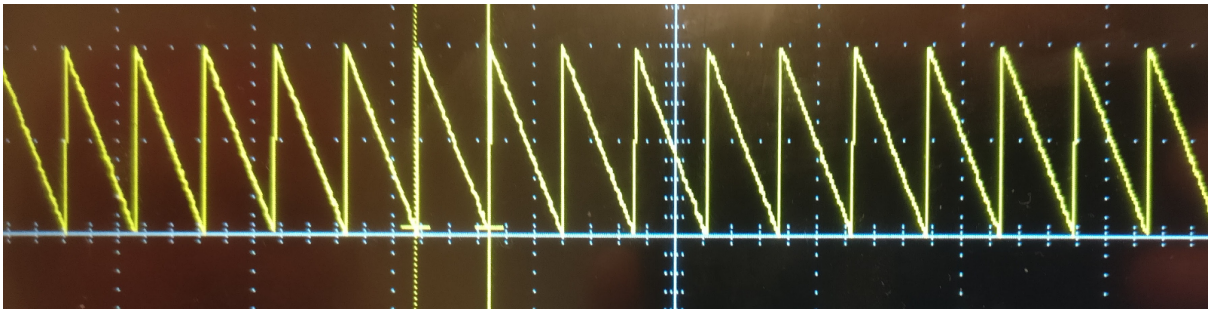
Calculation of time period for the sawtooth wave:

$\frac{1}{50MHz} = 20ns$

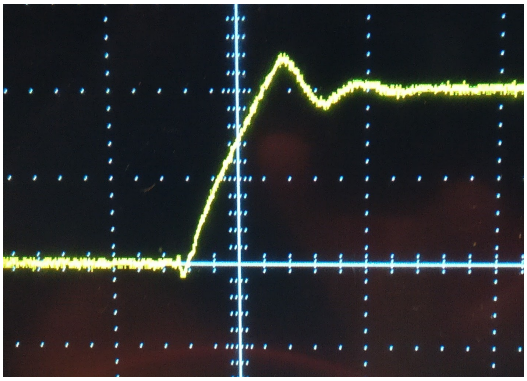
$\frac{50MHz}{100Hz} = 500,000$

$256 \cdot 500,000 \cdot 20ns = 2.6 \cdot 10^3 ms$

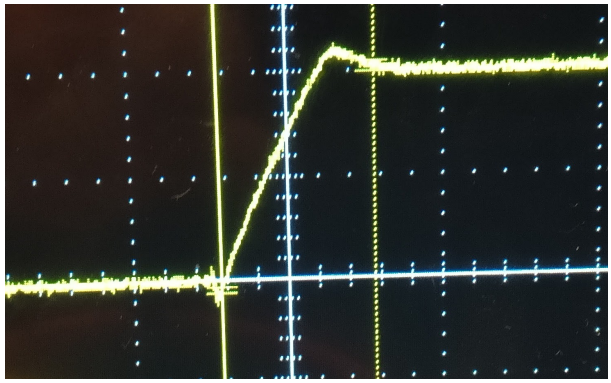
Measured time on oscilloscope was $2.56 \cdot 10^3 ms$



When increasing the frequency to 9.6kHz an overshoot becomes noticeable when going from 0 to 255. To fix this a capacitor is C_2 is placed in parallel with R_2. The value of C_2 is 33pF.



Without C_2

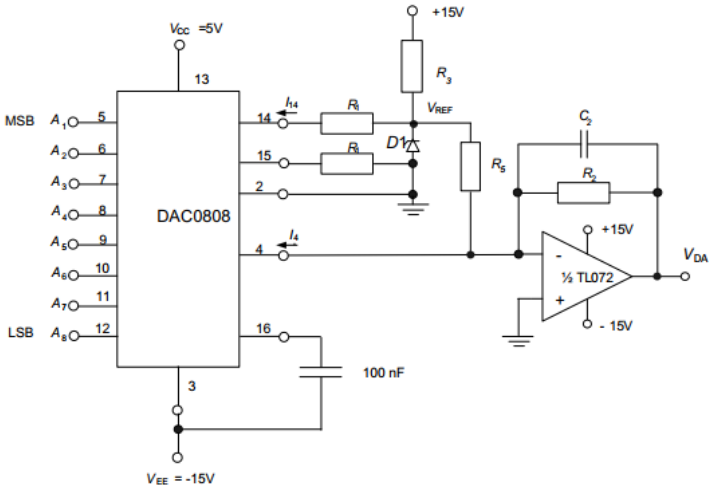


With C_2

According to the datasheet of TL072 the slew rate is 13 V/µS. When measured without C_2 the slew rate was 7.1 V/µS and with C_2 the slew rate was 10.8 V/µS.

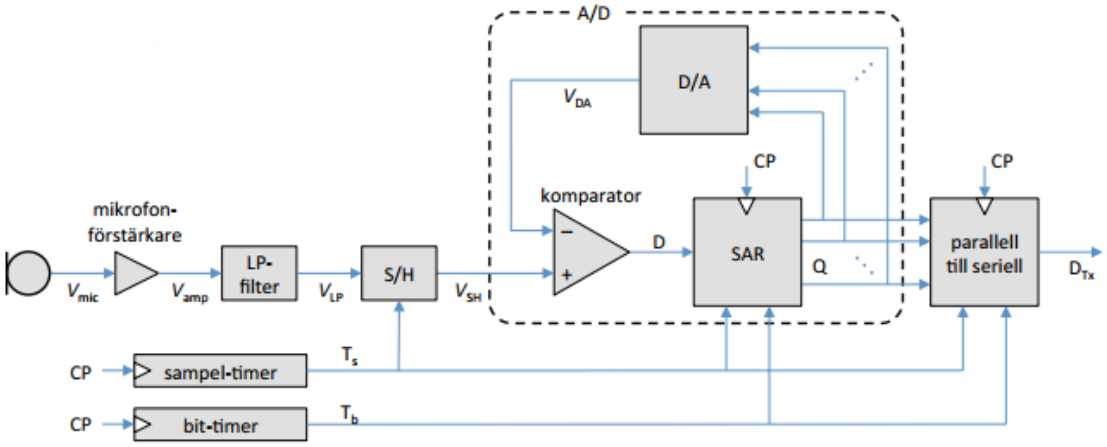
When converting AC-signals it is a good idea to have the input voltage symmetrical around 0V. To do this whe have to modify the D/A converter and make it bipolar. This is done by introducing a resistance R_5. When choosing R_5, I_5 should be $\frac{I_4}{2}$. The range of V_{DA} will then be from $-R_2\frac{I_{4Max}}{2}$ to $R_2\frac{I_{4Max}}{2}$.

$$R_5 = \frac{V_{ref}}{I_4/2} = \frac{5.6V}{1mA} = 5.6k\Omega$$



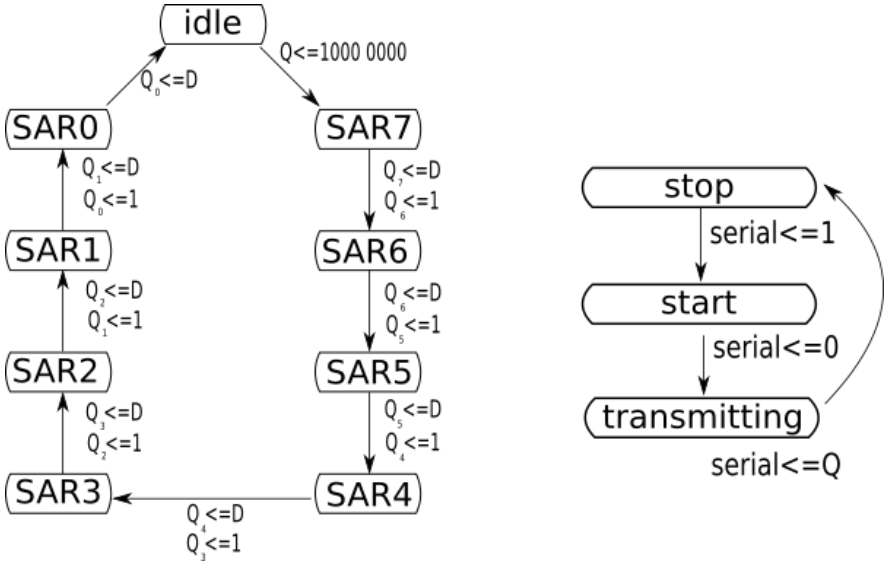
A/D converter

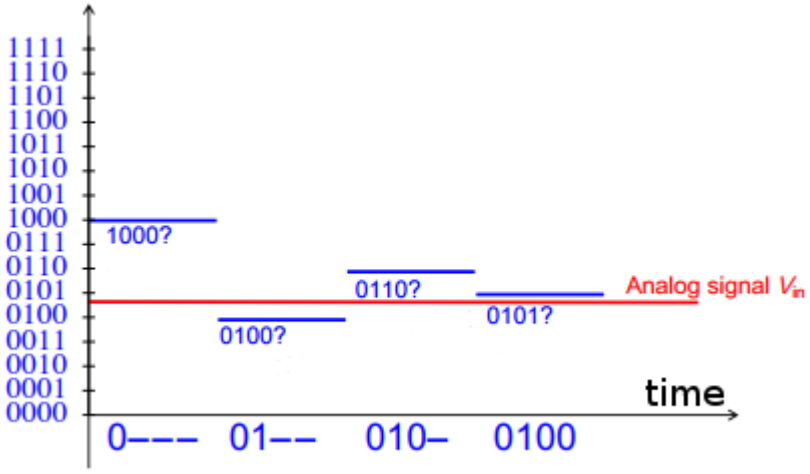
The A/D converter converts an analog voltage to a digital representation. This is done using the D/A converter that is described in previous section together with a SAR approximation.



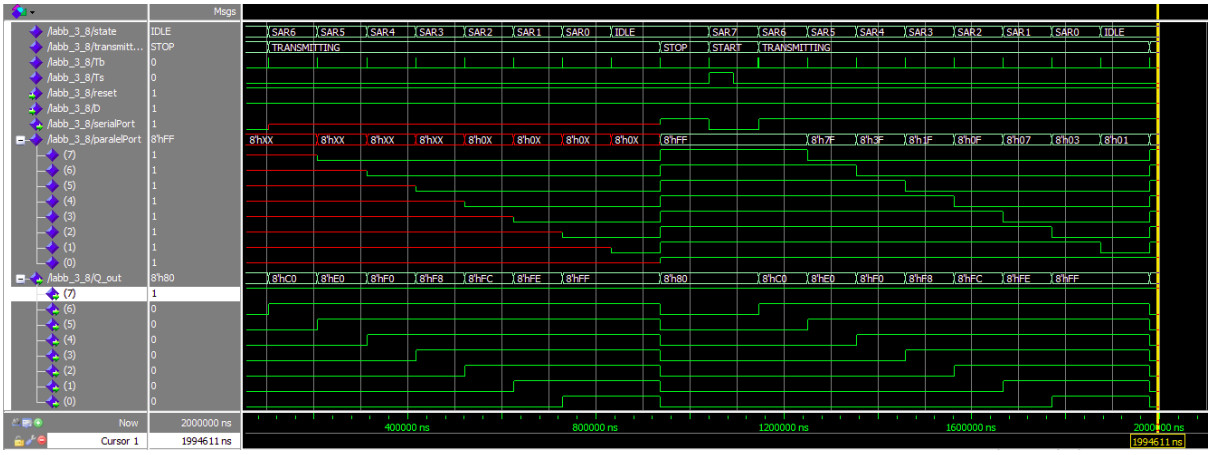
The approximation is done by comparing V_{DA} with the input voltage V_{SH} that is going to be represented. Starting of the SAR will give the D/A 1000… representing $\frac{V_{DA\,max}}{2}$. If this value is less than V_{SH} the feedback D will be 0 or 1 if it is greater than V_{SH}. For the next iteration the output of the SAR will be $D100...$ This process is then repeated n times where n is the number of bits that is going to represent the analog value.

The state machine for the SAR and parallel to serial are two different state machines. The parallel to serial state machine takes the previous Q value from the SAR and bit shifts it out on the serial port. A counter in the transmitting state makes sure that all bits are shifted out on the serial port before going to a stop state. This is implemented in state_machine.vhd.



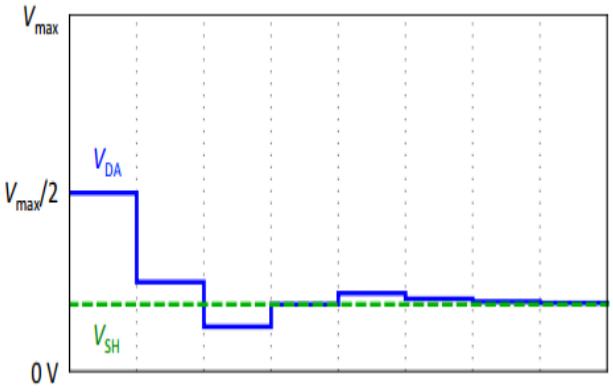


To test out the SAR and parallel to serial in modelsim the feedback D was set to a constant 1.

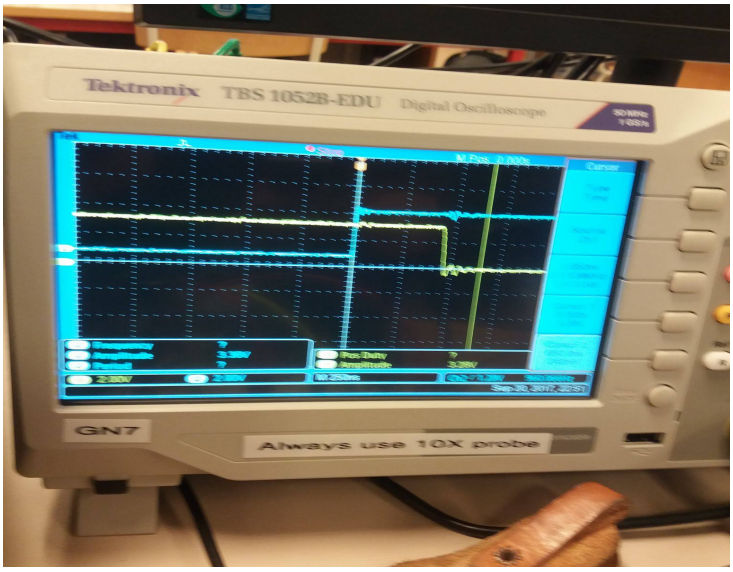


To test out the A/D converter a function generator was attached to V_{SH}. When generating a 5V DC voltage the digital representation from the SAR was $1000\ 0011$ and the theoretical calculated value was $\frac{256 \cdot 5v}{V_{DA\,max}} = \frac{256 \cdot 5v}{9.66V} = 132 \Rightarrow 1000\ 0100$. The difference between the measured value and the calculate value is only one bit.

The result of trying to mimic the approximation to the right below. The analog input signal was 1.9V and the digital representation from the SAR was 00110000. The correlation relation between the analog value and the digital representation is $A = \frac{256 \cdot V_{DA}}{V_{DA\,max}}$.

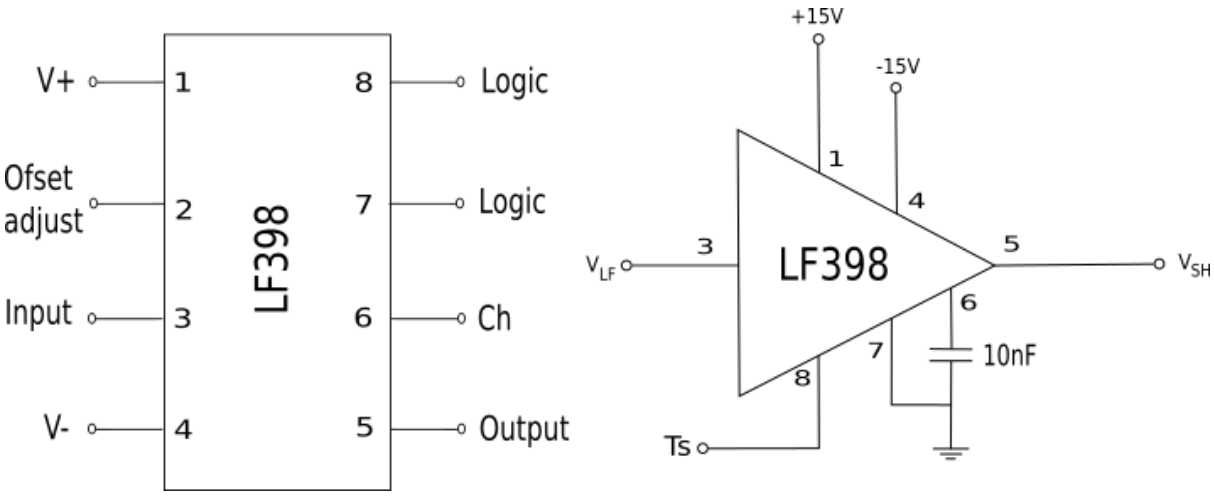


The measured delay from Q to V_{DA} when $V_{SH} = 3V$ was 350ns. When V_{SH} gets closer to $\frac{V_{DA\,max}}{2}$ the delay is increased. This is becauses it takes longer time for the comparator to compare two values that is close to each other.



Sample And Hold

The type of A/D converter in this system requires a constant analog input signal while determining the digital representation with the sar algorithm. Therefore a sample and hold circuit LF398 is placed before the A/D converter.

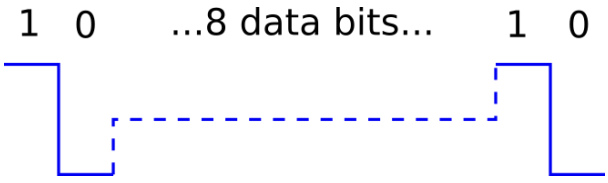


The circuit is sampling when $V_8 - V_7 > 1.4V$ and holding when $V_8 - V_7 < 1.4V$. When sampling, V_{SH} is equal V_{LF} and the voltage of V_{LF} is stored in the capacitor. When holding, V_{SH} is equal the voltage of the capacitor which is the previous value of V_{LF}.

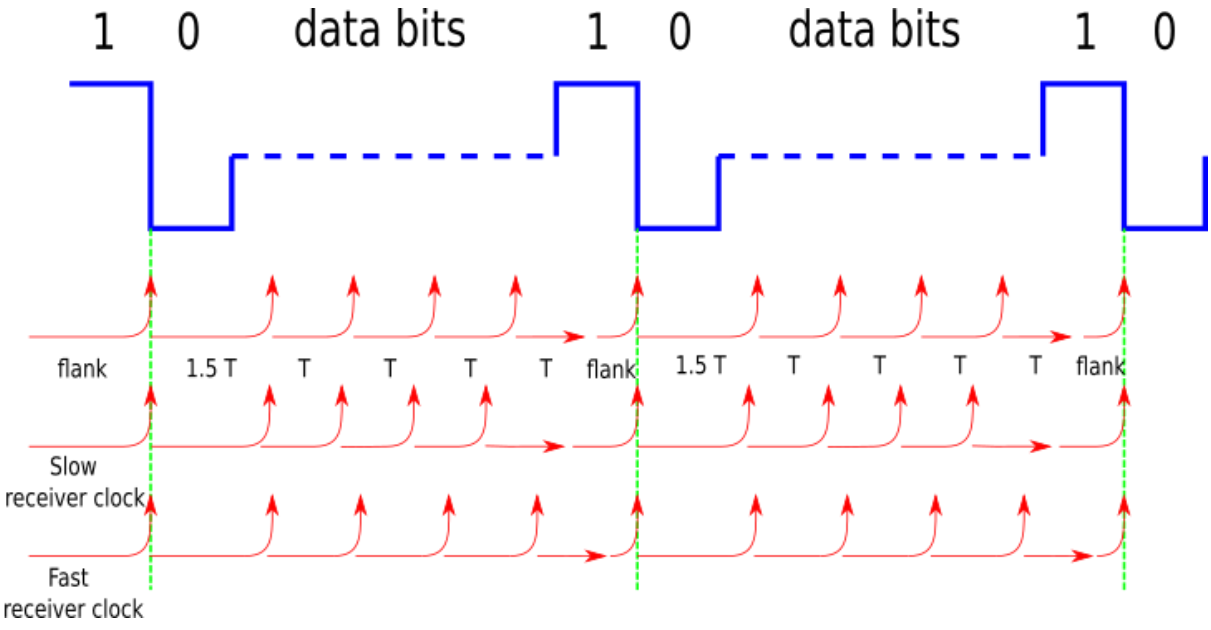
Serial receiver

The serial receiver is a state machine that receives 8 bit words and outputs the word parallel to the D/A. The transmission is done over the RS-232 port.

A message consists of 10 bits where the first bit is a start bit (0), then 8 data bits and last a stop bit (1). The state machine is initially in a stop state and waits for a start bit 0 before it begins to sample the 8 bit data word. After the 8 data bits the state machine returns to a stop state if it receives a stop bit 1.

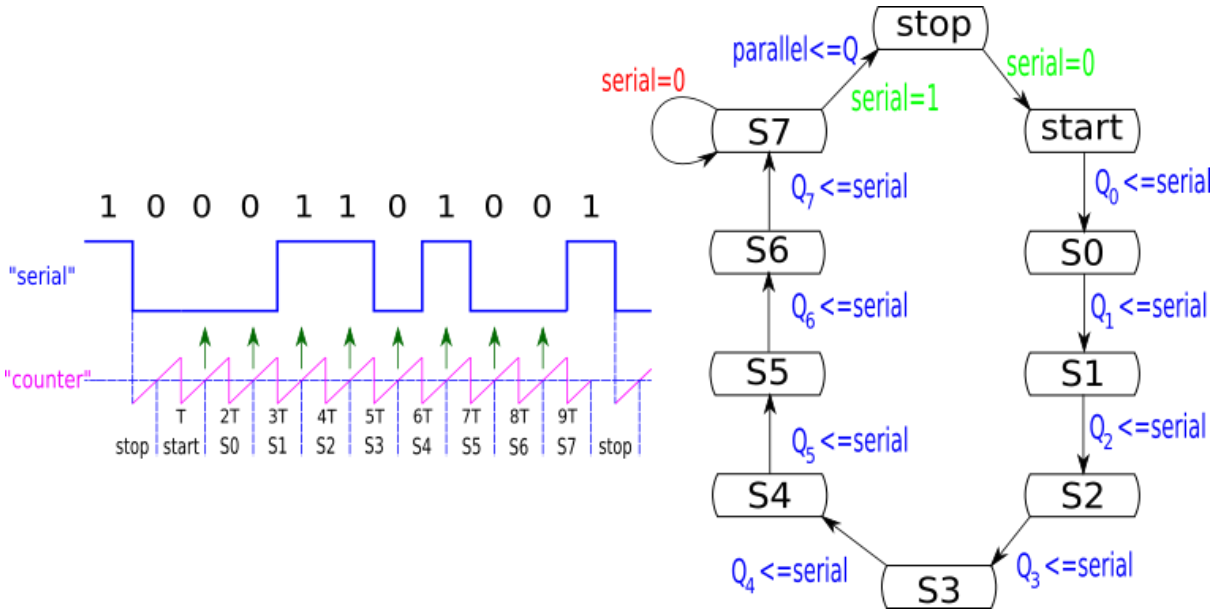


When receiving data serial there are multiple challenges with synchronisation between sender and receiver. The first one is to ensure that the sender and receiver keeps up with the same bitrate. Even though crystal oscillators are accurate they will eventually drift, then sender and receiver will no longer be in sync. This is solved by symbol synchronization, this means that the sender's clock signal is recycled from the received signal.

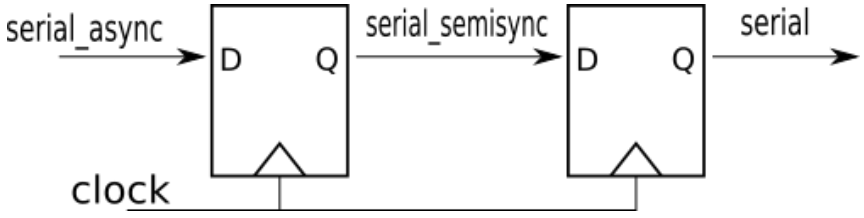


Every Time a flank is detected a counter is reset, because of this the receiver clock can be a bit to slow or to fast without drifting relative to the sender. The state machine waits 1.5 T from a flank before starting to sample with a period of T. This is to avoid sampling in a transition between two symbols.

There is also a problem if the state machine detects a false flank, then the data bits will not be blocked correctly. This is solved by block synchronization. The state machine is not allowed to go from S7 to stop without receiving (1) on serial. If the state machine is in state S7 and does not receive (1), this means that it started on a false flank and has to wait until it receives (1). Even if it receives (1) in state S7, there are no guarantees that this is the stop bit. Eventually after a few data words has been transmitted the sender and receiver will be in sync. Block synchronization is only a problem in the beginning of a transmission.



When dealing with asynchronous input signals there is a problem with metastability. This can occur when the serial in signal changes close to a clock pulse. Then the state of Q is neither 1 or 0 and the outcome is random. The risk of metastability can not be completely eliminated but it can be reduced if the asynchronous signal is clocked.



To test out the receiver it was connected to a computer that sent out 8 bit words with a baudrate of 9600.

ASCII-character	Received bit sequence	Analog output D/A (V)
P	0101 0000	-1.581
?	0011 1111	-2.225
!	0010 0001	-3.357
=	0011 1101	-2.298
A	0100 0001	-2.148

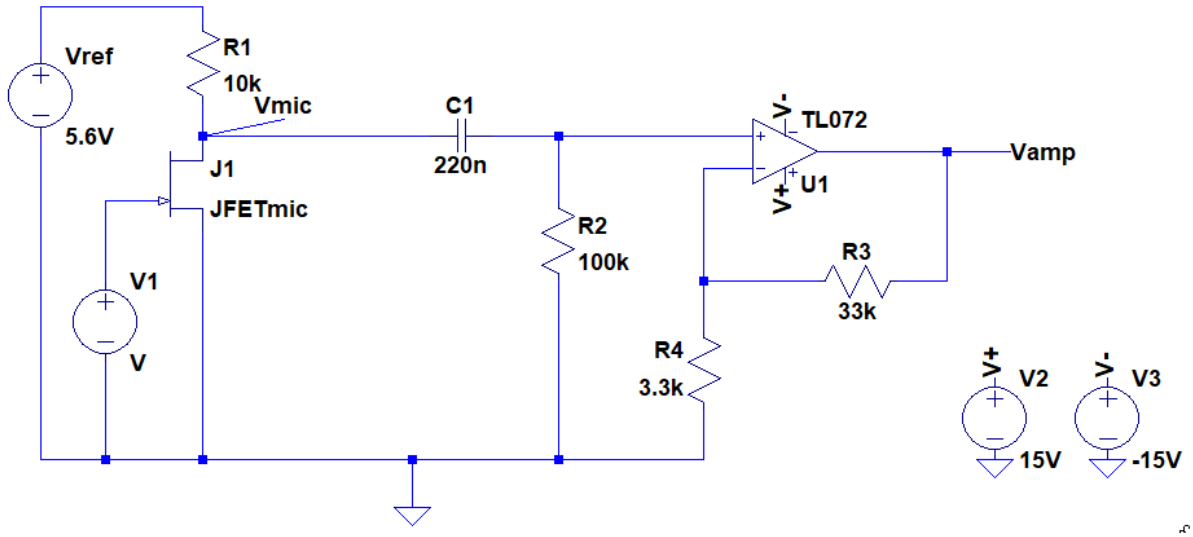
The audio transmission system should be able to handle audio signals in the 20-12000Hz interval. According to Nyquist-Shannon sampling theorem the sample rate should be double the frequency that is going to be sampled. The sampling frequency T_s will then be $12k \cdot 2 = 24kHz$

Every sample is represented by 8 data bits and 2 signal bits. This means that T_b hast to be $(8 + 2) \cdot T_s = 240kHz$

Audio amplifier

Microphone amplifier

The microphone amplifier should make full use of the A/D converter. This means that the output amplitude should be ∓5V or 11dBV. At an impedance of 2kΩ the sensitivity of the microphone is -39dB. The idle current of the microphone is 0.25 mA. The microphone is simulated with a jfet transistor and a voltage supply. The reference pressure is 20µPa which translates to 94dB.



Calculation of the input impedance for the microphone amplifier:

$10k\Omega \,//\, 100k\Omega = \frac{10k \cdot 100k}{10k+100k} = 9090\Omega \approx 9k\Omega$

Calculation of the maximum sound pressure level without exceeding the limit of the D/A:

$-39dBV \text{ translates to } 11mV$

$I = \frac{11mA}{2k\Omega} = 5.5\mu A$

$The\ sensitivity\ of\ the\ microphone\ is\ 5.5\mu A\ @\ 1Pa$

$V_{mic\ idle} = 9k\Omega \cdot 5.5\mu A = 49mV$

$A_u = \frac{R_1+R_2}{R_4} = \frac{3.3k+33k}{3.3k} = 11$

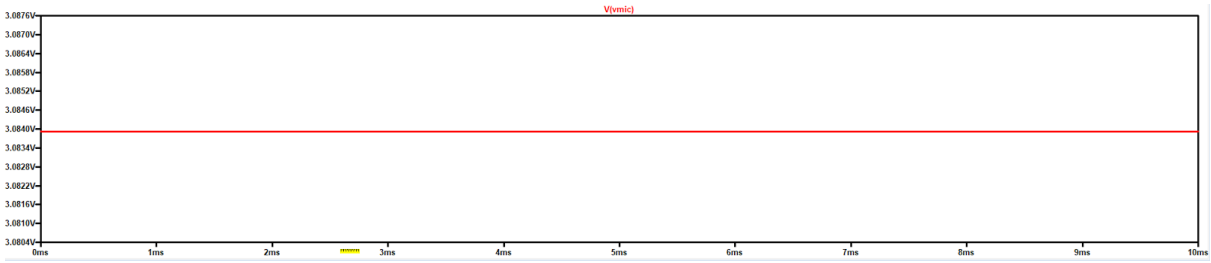
$V_{mic\ max} = \frac{5}{11} = 454mV$

$RMS\ of\ V_{mic\ max} = \frac{454mV}{\sqrt{2}} = 321mV$

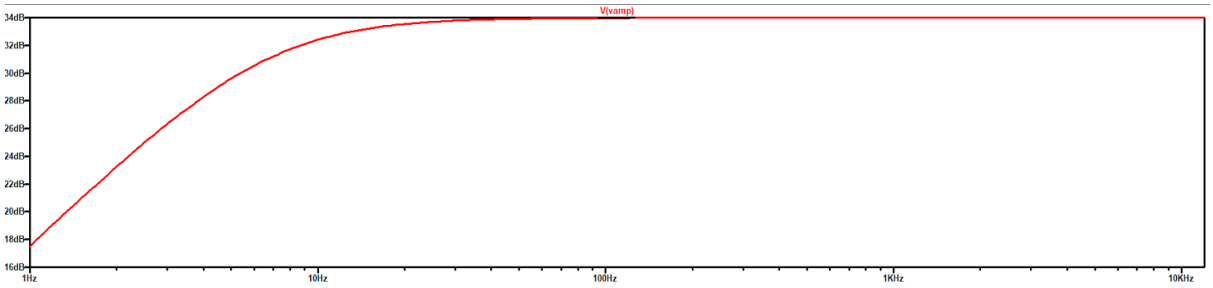
$change\ in\ dB\ between\ V_{mic\ idle}\ and\ V_{mic\ max} = 20\ log\left(\frac{321mV}{49mV}\right) = 16.33dB$

$The\ maximum\ sound\ pressure\ level\ will\ then\ be = 94dB + 16.33dB = 110.3dB\ SPL$

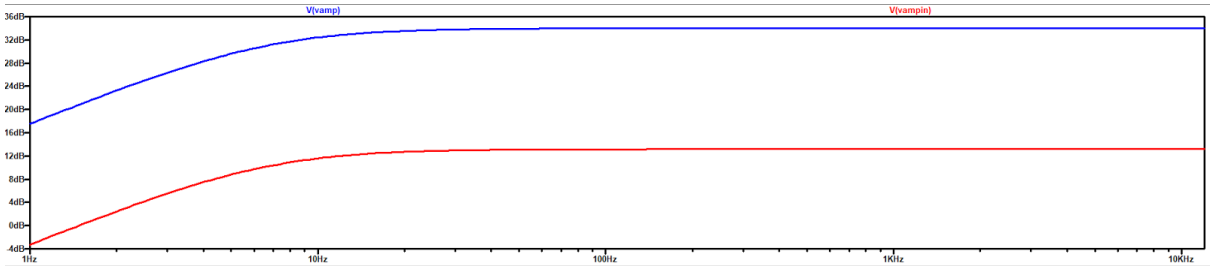
When simulating the the microphone amplifier the Idle voltage of Vmic is 3V



The lower cut frequency is of Vamp is 6.7Hz

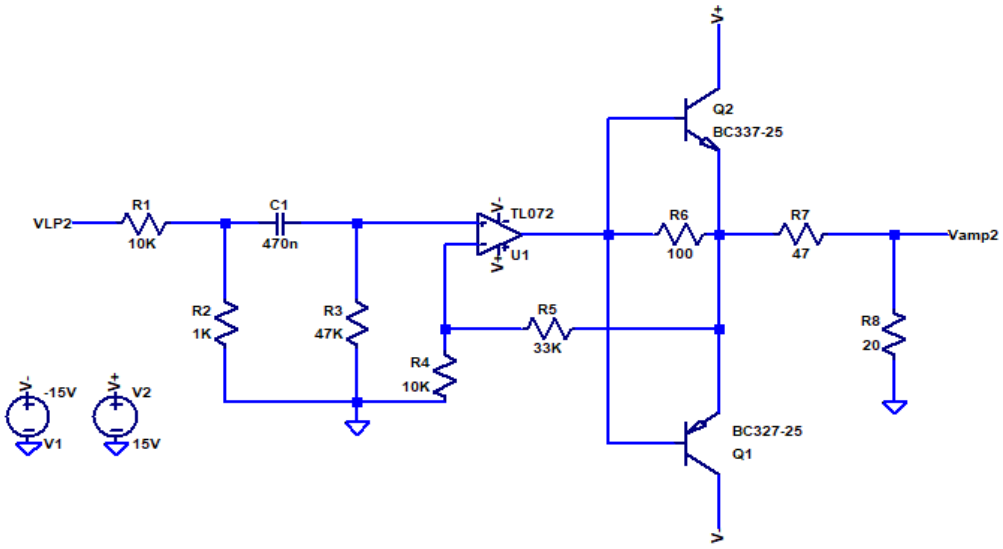


The amplification is 21dB which translates to a amplification factor of 11.2



Power amplifier

The power amplifier is a push-pull type class B amplifier. The amplifier is designed to provide enough power to supply a headphone with an impedance of 20Ω. This means that the amplifier at least has to deliver an output of $35mA_{rms}$. A normal headphone has a sensitivity of 100dB SPL/mW.



Calculation of voltage amplification:

$1k \,//\, 10k = \frac{1k \cdot 47k}{1l+47k} = 979\Omega$

$Voltage\ deviation\ 1 = \frac{979}{10k+979}$

$Op\ amplification = \frac{10k+33k}{10k}$

$Voltage\ deviation\ 2 = \frac{20}{47+20}$

$amplification = \frac{979}{10k+979} \cdot \frac{10k+33k}{10k} \cdot \frac{20}{47+20} = 0.1145$

Calculation of the maximum sound pressure the power amplifier can achieve:

$u_{top} = A \cdot 5V = 0.1145 \cdot 5V = 0.57V$

$u_{rms} = 0.40V$

$power\ of\ headphone = \frac{u_{rms}^2}{R} = \frac{0.40^2}{\sqrt{2}} = 8mW$

$8mW = 10\ log\left(\frac{8mW}{1mW}\right) = 9\ dBM$

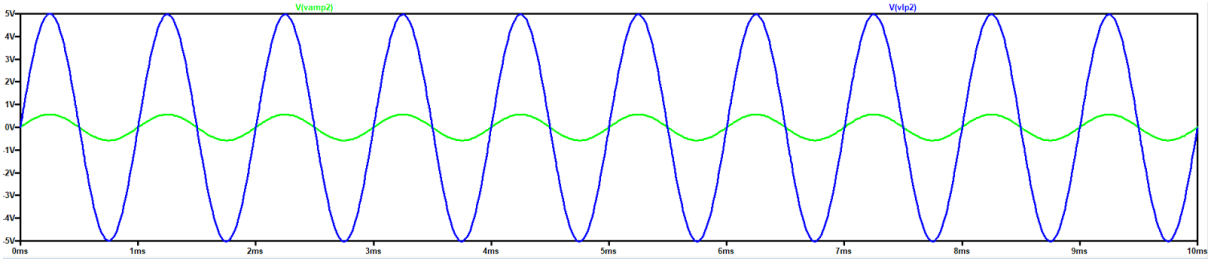
$1mW = 0\ dBM \Rightarrow 100\ dB\ SPL$

$9\ dBM \Rightarrow 100 + 9 = 109\ dB\ SPL$

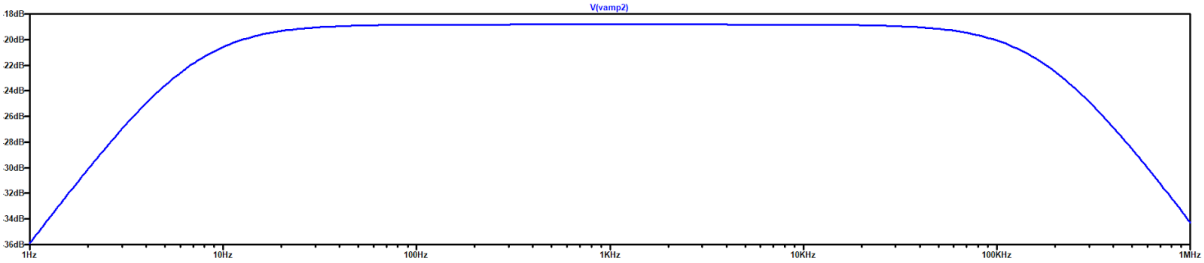
The maximum sound pressure is 109 dB SPL

Given an input of 5V the output of the power amplifier is 568mV

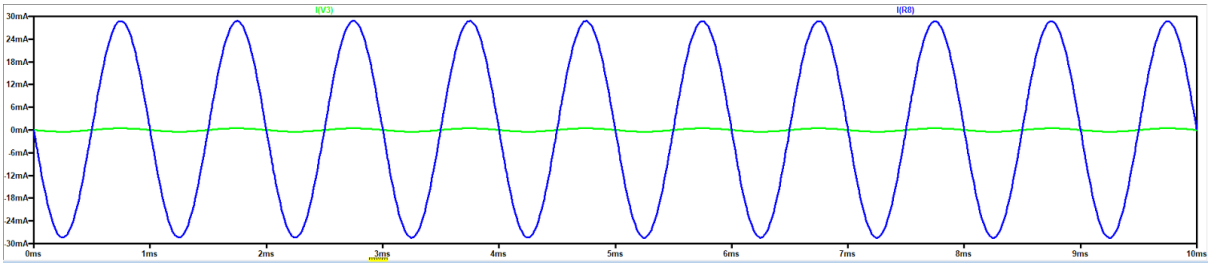
$A = \frac{568mV}{5V} = 0.1136$ which is very close to the previous calculation.



The lower cut frequency is 7.1Hz and the upper cut frequency is 172.9kHz.



Given a input current of 454.6µA the output current is 28.7mA



$P_{in} = 5V \cdot 453.6\mu A = 2.273mW$

$P_{out} = 568mV \cdot 28.7mA = 16.3mW$

The power amplification from V_{LP2} to V_{amp2} is $\frac{16.3mW}{2.273mW} = 7.17$

LP filter

The transmitted audio signal is sampled at 24kHz, when played back frequencies over 12kHz will only be unwanted distortion and noise. To remove frequencies over 12kHz a Lowpass filter is needed. The lowpass filter will be a fourth degree active butterworth filter consisting of two sallen key links using the TL072. Calculated values for resistors and capacitors will be rounded to closest value in E12 for resistors and E6 for capacitors. To achieve smaller rounding errors two capacitors are used in parallel in some of the calculations.

$$P(a) = (1 + 0.765a + a^2)(1 + 1.848a + a^2)$$

$$H(s) = \frac{1}{1+\frac{0.765s}{\omega_u}+\frac{s^2}{\omega_u}} \cdot \frac{1}{1+\frac{1.848s}{\omega_u}+\frac{s^2}{\omega_u}}$$

$$\omega_u = 2\pi \cdot 12k$$

$$H(s)_{sallen\ key} = \frac{1}{1+s(R_1+R_2)C_2+S^2R_1R_2C_1C_2}$$

Sallen key link 1:

$$Choosing\ C_2 = 3.3nF$$

$$\frac{0.765}{\omega_u} = C_2(R_1 + R_2)$$

$$R_1 + R_2 = 3075\Omega$$

$$R_1 = R_2 = 1538\Omega \approx 1.5k\Omega$$

$$\frac{1}{\omega_u^2} = R_1R_2C_1C_2$$

$$C_1 = \frac{\frac{1}{\omega_u^2}}{R_1R_2C_2} = 24nF \approx (22nF + 2.2nF)$$

Sallen key link 2:

$$Choosing\ C_4 = 11nF$$

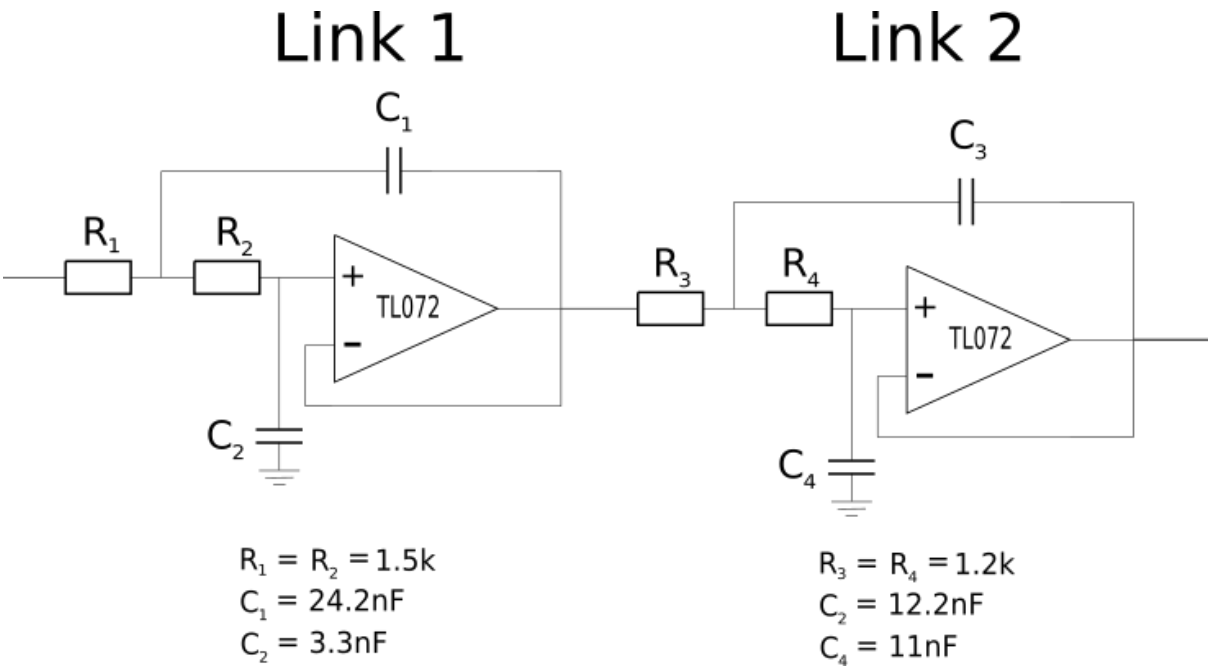
$$\frac{1.848}{\omega_u} = C_4(R_3 + R_4)$$

$$R_3 + R_4 = 2228\Omega$$

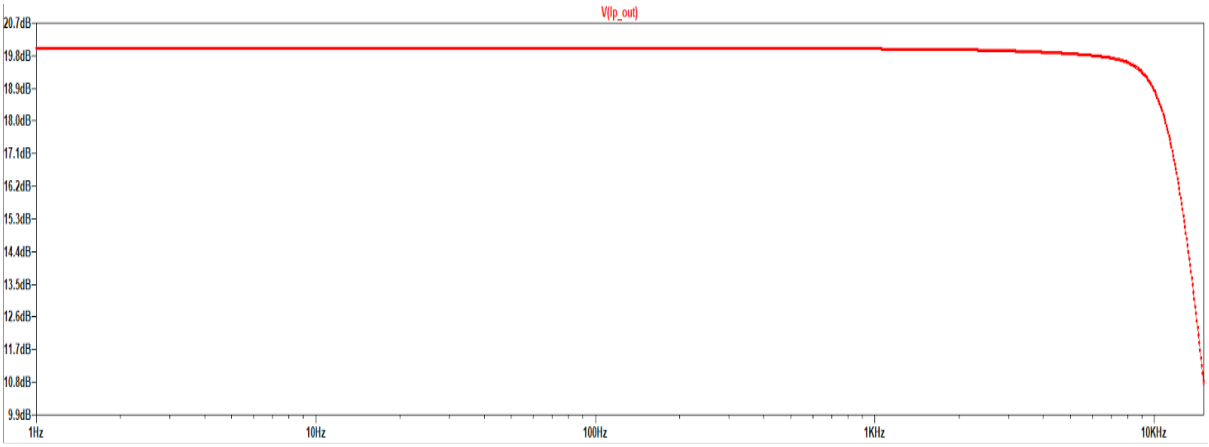
$$R_3 = R_4 = 1114\Omega \approx 1.2k\Omega$$

$$\frac{1}{\omega_u^2} = R_3R_4C_3C_4$$

$$C_3 = \frac{\frac{1}{\omega_u^2}}{R_3R_4C_4} = 12nF \approx 12.2nF$$



The LT-spice simulation of the low pass filter resulted in a upper cut frequency of 11.7kHz.



To test out the LP-filter a function generator was connected with a sinusoidal output of 10Vpp and varying frequencies. Results in table below.

Frequency (kHz)	Input (V)	Output (V)	Damping (dB)
1 kHz	5 V	5 V	0
5 kHz	5 V	5 V	0
8 kHz	5 V	4.96 V	0.07
10 kHz	5 V	4.5 V	0.09
11 kHz	5 V	4.08 V	1.73

12 kHz	5 V	7 V	3.09

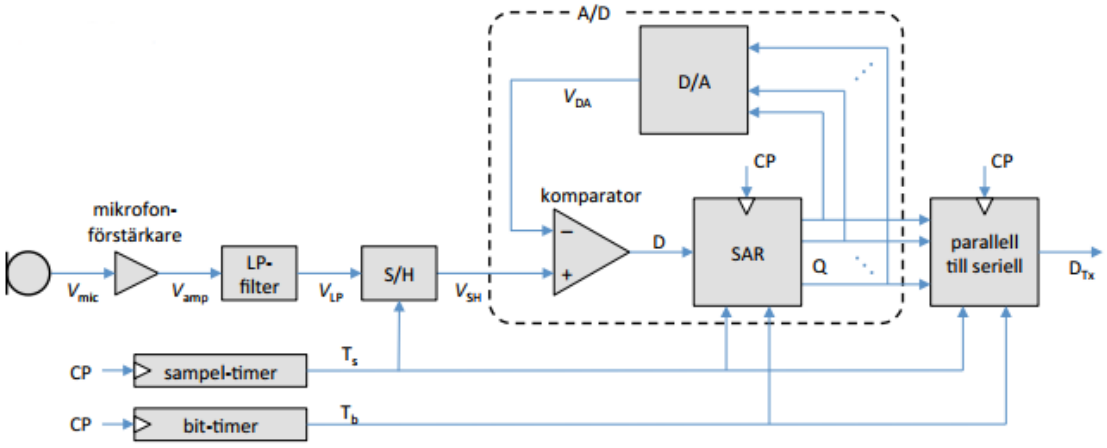
The LP-filter resulted in a upper cut frequency of 12kHz.

When sound was transmitted it sounded good, even with some of the bits shorted out. With only the 3-4 most significant bits it sounded good and with just 2 it was understandable.

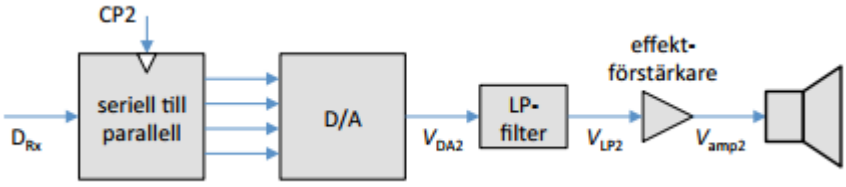
Test and verification

When testing the system one labsystem was assigned a sender roll and the other one handled the receiving of the audio signals.

The sender system consists of a microphone, microphone amplifier, A/D converter and parallel to serial transmitter.



The receiver system consists of a serial to parallel receiver, low pass filter, power amplifier and a pair of headphones.



The sender and receiver were then connected with each other using the RS232 cable. To synchronize the systems different DC voltages was then given to the sender using a function generator. The output was analysed on the receivers leds until the two systems were synchronized.

The output of the function generator was then changed to AC providing a sinusoidal output of 1kHz. The least measurable output value of the receiver was $256mV_{pp}$. The peaks of the output signal gets distorted when the transmitter sends signals over $9.7V_{pp}$. The transfer dynamic is $\frac{9.7V}{256mV} = 37.9\ dB$

At the max distortion free value value $256mV_{pp}$ the bandwidth is between (1hz-6.5kHz).

A microphone was connected to the transmitter and a headphone to the receiver. It was then possible to transmit audio signals. The sound quality was decent and you could hear another person talking in the microphone. Even when shorting out 7 of the 8 bits on the D/A it was possible to distinguish words.

Discussion

Lot of time has been invested in debugging and troubleshooting during the design and building phase of this audio transmission system. There has been multiple errors both in the analog circuits and the code implementation on the fpga circuit. Following are some of the problems that we have encountered and implementations that should have been done differently.

A/D-converter
The SAR and the parallel to serial transmitter are two separate state machines. This design worked fine in the lab session and there were never any problems with transmitting the data. However when looking back at the code you notice that it is hard to understand what is going on and keep track of all the signals. It would have made much more sense to design it as a signal state machine. This was taken in consideration when designing the D/A-converter.

D/A-converter
The D/A-converter and serial to parallel receiver worked fine at the first test when a computer acted as the transmitter. However when the audio transmitting system was sending data the receiving system were not able receive the same data. The cause of this problem could be metastability together with a faster baud rate. This was solved by clocking the asynchronous serial signal.

Lowpass filter
The designing process of the lowpass filter was very time consuming and many recalculations were made. Even with correct calculations the simulated upper cut frequency had big deviations from the calculated. This was the result of rounding errors when choosing resistors from the E12 series and capacitors from the E6 series. To solve this two capacitors were used in parallel to achieve a smaller rounding error.

After the final lab session the system performed as in the specifications and it was possible to transmit audio from a transmitting system to a receiving system. The audio quality was beyond expectations considering the D / A converter only has a resolution of 8 bits.

Reflection

This was the first time the two of us worked in a project together. We had some different ways of solving problems like this. The pre assignments for the first four lab sessions worked out great and we had a good collaboration. The both of us comes from the computer engineering program and we are used to write code. It was easy to share and contribute using a git repository. The pre assignments for lab session 5 and 6 was more about analog components and calculation. There was a great knowledge difference between us when it comes to analog components and it was not as easy to share the calculations and understanding behind them. This resulted in only one of us solving the pre assignments and we missed out on the opportunity to learn from each other.

Since we are not from the electrical engineering program there was a big learning curve when it came to using the lab equipment. We spent lot of time trying to understand how to get a good representation of the signals on the oscilloscope before we were able to to the measurements. It would have been nice to have more of an introduction in using the lab equipment.

The assignments involving VHDL and modlesim was not that hard in the early state of the lab series. In lab session 3 when the state machine was introduced the number of signals escalated quickly. The time and effort debugging the code drastically increased at this point.

It was really hard to start working with lt-spice, and to know which simulation command you should use to measure the results. The third handin involving the mosfet transistors helped with getting to know lt-spice.

The supervisors had lot of knowledge and they were an important asset when debugging the analog components. During some of the lab sessions the time with a supervisor was very limited and major parts of the session was wasted waiting for help. It would have been nice if the supervisors had taken more time discussing the result and making sure that we understand them before moving on to the next part.

Final thoughts, it was interesting to see the final result coming together and be able to understand how transmitting data and soundsystem works since this is something we take for granted in everyday life.

Apendex

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

entity eight_bit_counter is port (
	clk50: in std_logic;
	sw : in std_logic_vector (7 downto 0);
	reset: in std_logic;
	SW9: in std_logic;
	ldr: out std_logic_vector (7 downto 0);
	DAC_A : out std_logic_vector (7 downto 0);
	Tb_out: out std_logic);

end entity;
architecture arch of eight_bit_counter is
	signal timer: std_logic_vector(18 downto 0);
	signal counter : std_logic_vector(7 downto 0);
	signal Tb: std_logic;
begin
	Tb_out <= Tb;
	ldr <= counter;
	DAC_A <= counter;
	process(clk50, reset)
	begin
		if reset='0' then
			timer <= (others => '0');

		elsif rising_edge(clk50) then
			if timer=499999 then
				Tb <= '1';
				timer <= (others => '0');
			else
				timer <= timer +1;
				tb <='0';
			end if;
		end if;

	end process;
	process(clk50, Tb, reset)
	begin
		if reset ='0' then
			counter <= (others => '0');

		elsif rising_edge(clk50) then
			if Tb = '1' then
				if SW9 = '1' then
					counter <= counter + 1;
				elsif SW9 = '0' then
```

```
                                counter <= counter -1;
                            end if;
                        end if;
                end if;
        end process;

end architecture;



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

entity SAR is port (
        clk50: in std_logic;
        reset: in std_logic;
        serialPort: out std_logic;
        serialShadow: out std_logic;
        LDR: out std_logic_vector(7 downto 0);
  D: in std_logic;
  Q_out: out std_logic_vector(7 downto 0));
end entity;

architecture arch of SAR is
        signal Tb: std_logic;
  signal Ts: std_logic;
  signal timer_Ts : std_logic_vector(15 downto 0);
  signal timer_Tb : std_logic_vector(12 downto 0);
        signal Q: std_logic_vector(7 downto 0);
        signal prev_Q: std_logic_vector(7 downto 0);
        signal bitsTransmitted: std_logic_vector(4 downto 0);
        type TransmittingType is (U,STOP,START,TRANSMITTING);
        signal transmittingState: TransmittingType;
  type StateType is (U,IDLE,SAR7,SAR6,SAR5,SAR4,SAR3,SAR2,SAR1,SAR0);
  signal state: StateType;
        signal paralelPort : std_logic_vector(7 downto 0);
begin
  Q_out <= Q;

        process(clk50, reset)
  begin
    if reset='0' then
                        timer_Ts <= (others => '0');
                        timer_Tb <= (others => '0');
                        Ts <= '1';
                        Tb <='1';
                elsif rising_edge(clk50) then
                        --period 52083 pulses
                        if timer_Ts=103 then
                                Ts <= '0';
                                timer_Ts <= timer_Ts +1;
                        elsif timer_Ts=2079 then
                                Ts <= '1';
```

```
                        timer_Ts <= (others => '0');
                    else
                        timer_Ts <= timer_Ts +1;
                    end if;

                    if timer_Tb=207 then
                        Tb <='1';
                        timer_Tb <= (others => '0');
                    else
                        timer_Tb <= timer_Tb +1;
                        Tb <='0';
                    end if;

            end if;
    end process;


        process(Tb, Ts, clk50)
        begin
            if reset='0' then
                    bitsTransmitted  <= (others => '0');
                    transmittingState <= STOP;
              elsif rising_edge(clk50) then

                    case transmittingState is

                        when STOP =>
                            serialPort <= '1';
                            serialShadow <= '1';
                            --paralelPort <= Q;
                            paralelPort <= prev_Q;
                            LDR <= prev_Q;
                            if (Ts = '1') and (Tb = '1') then
                                transmittingState <= START;
                            end if;

                        when START =>
                            serialPort <= '0';
                            serialShadow <= '0';
                            if (Tb = '1') then
                                transmittingState <= TRANSMITTING;
                            end if;

                        when TRANSMITTING =>
                            serialPort <= paralelPort(0);
                            serialShadow <= paralelPort(0);
                            if (Tb = '1') and (bitsTransmitted = 7) then
                                bitsTransmitted  <= (others => '0');
                                transmittingState <= STOP;
                            elsif (Tb = '1') then
                                paralelPort(7 downto 0) <= '0' & paralelPort(7
downto 1);
                                bitsTransmitted <= bitsTransmitted + 1;
                            end if;
```

```
                        when others =>

                    end case;
                end if;
            end process;

process(clk50, reset)
begin
  if reset='0' then
    state <= IDLE;
  elsif rising_edge(clk50) then
    if Tb = '1' then
      case state is
        when IDLE =>

          Q <= "10000000";
          if Ts = '1' then
            state <= SAR7;
          end if;
        when SAR7 =>
          Q(7) <= D;
          Q(6) <= '1';
          state <= SAR6;
        when SAR6 =>
          Q(6) <= D;
          Q(5) <= D;
          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;
          prev_Q(7 downto 1) <= Q(7 downto 1);
          prev_Q(0) <= D;
          state <= IDLE;
        when others =>
```

```
            --nothing
        end case;
      end if;
   end if;
  end process;

end architecture;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity receiver is port(
  reset, clk50: in std_logic;
  serial_async: in std_logic;
  parallel_out: out std_logic_vector(7 downto 0);
  parallel: out std_logic_vector(7 downto 0));

end entity;

architecture state_machine of receiver is
  type StateType is (U,STOP,START,S0,S1,S2,S3,S4,S5,S6,S7);
  signal state, next_state: StateType;
  signal serial_semisync, serial : std_logic;
  signal Q, next_Q, next_parallel: std_logic_vector(7 downto 0);
  signal counter: std_logic_vector(12 downto 0);
  begin

  process (clk50)
  begin
       if rising_edge(clk50) then
              serial_semisync <= serial_async;
              serial <= serial_semisync;
       end if;
  end process;

  process(clk50,reset)
  begin
    if reset='0' then
      state <= STOP;
      Q <= (others => '0');
      counter <= (others => '0');
    elsif rising_edge(clk50) then
      counter <= counter+1;
      if state=STOP and serial='1' then
        counter <= (others => '0');
      elsif counter=103 then -- half-period
        state <= next_state;
        Q <= next_Q;
        parallel_out <= next_parallel;
      elsif counter=207 then -- bit clock
        counter <= (others => '0');
      end if;
    end if;
  end process; -- bit clock
```

```
  process(state,serial,Q) -- update state and outputs
  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 <= S7;
        else
        next_state <= STOP;
        end if;
      when others => -- undefined state
      next_state <= STOP;
    end case;
  end process; -- update state
  parallel <= next_parallel;
end architecture;


library ieee;
use ieee.std_logic_1164.all;
entity SW0_9_to_LEDR0_9 is port (
        sw : in std_logic_vector (9 downto 0);
        ldr : out std_logic_vector (9 downto 0);
        DAC_A : out std_logic_vector (7 downto 0));

end entity;
```

```
architecture arch of SW0_9_to_LEDR0_9 is
begin

                ldr <= sw;
                DAC_A <= sw(7 downto 0);

end architecture;
```