Lab Report
Audio transmission system

Elektriska system SSY011

Oskar Hellqvist & Pontus Karlsson

Summary

The purpose of this project is to construct a audio transmission system using the RS232 standard. The specifications for the system involves a frequency range between 20Hz-12000kHz and a resolution of 8 bits.

The resulting system is fully functional and fills the specifications. Both a transmitter and receiver was constructed. As both the transmitter and receiver use some of the same subcomponents a system cannot be configured as both a transmitter and receiver.

1. Introduktion

The purpose of the project is to construct a sound transmission system consisting of both analog and digital equipment. The specifications for the transmission system involves a frequency range 20-12000 Hz and a resolution of 8 bits. The system should be able to communicate with the RS232 standard. The goal of project is to get the system working and be able to communicate with equivalently constructed systems.

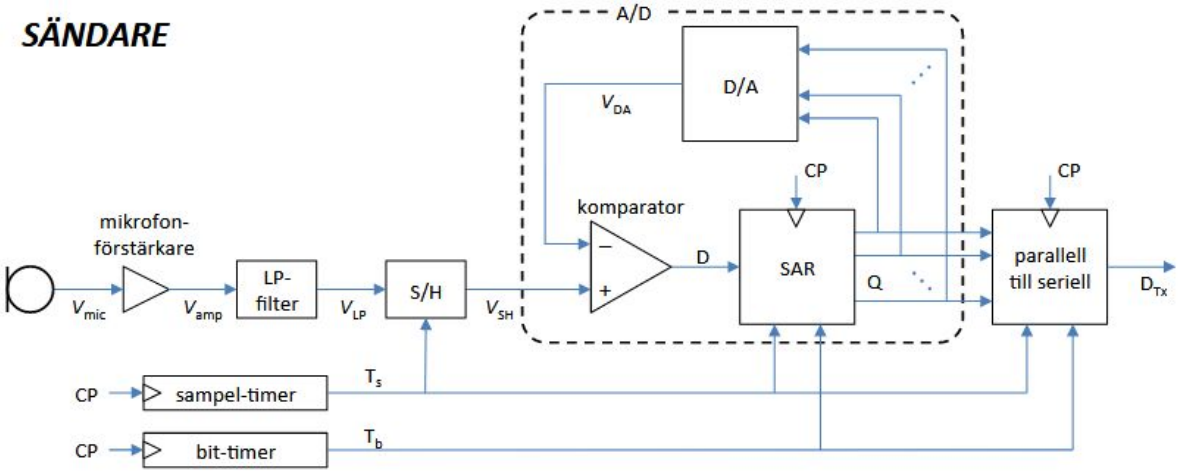

Figure 1: Transmitter

The sound transmission system can be divided into two main parts. Transmitter as illustrated in figure 1 above and receiver which is illustrated in figure 2 below. The transmitter consists of especially digital and digitally dependent subsystems. A transmission system of this construction cannot not be setup as a transmitter and a receiver at the same time since it utilizes some of the same subsystems in both versions of the system.

The A/D-converter is the biggest subsystem of the transmitter which in turn can be split into three further subsystems: a successive approximation register (SAR), a D/A-converter and a comparator. The main principle of the A/D-converter can be described by it comparing an analog in signal with a generated digital signal produced by the A/D-converter itself. When the generated signal has been fully approximated the converter sends the signal forth as a parallel output. Inside the A/D-converter the analog signal is compared several times with the generated signal in the comparator which result each time is used by the SAR to approximate and tweak the generated signal in order for the generated signal to correspond to the output. The D/A converter is used inside the A/D converter since the output from the SAR is a digital parallel signal which

has to be converted to analog form in order to make it comparable with the analog input. When the conversion is done the generated signal is sent from the SAR in parallel digital form. In order to follow the specifications the signal has to be converted from parallel output to a serial output which is done in the digitally constructed subsystem serial transmitter. From this subsystem the signal is sent serially to the receiver through a RS232-cable.

The transmitter's different subsystems is synchronized using modifications of the generated trig signals of the local oscillator.

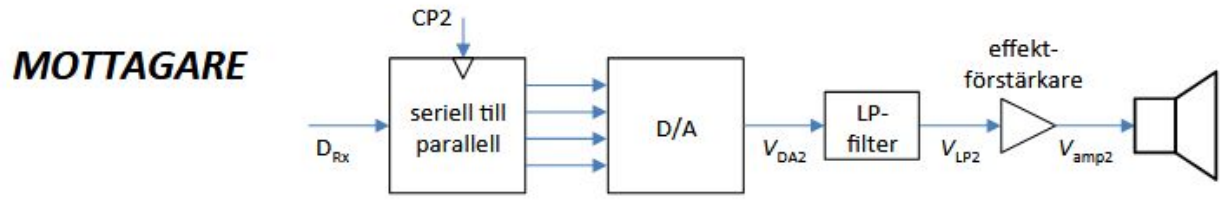

Figure 2: Receiver

The receiver obtains the sent signal in serial form which is converted to parallel form in the digitally constructed subsystem serial receiver in order for it to be converted to analog form in D/A-converter. When the signal is finished converting it is sent through a LP-filter in order to filter unwanted frequencies and static. A power amplifier is implemented after the LP-filter in order to give the signal the current required for driving the speaker.

2. Subsystems

2.1 Counter

There are two signals reliant on counters in the system. T_b and T_s. These are digital signals based on the 50Mhz internal clock of the Altera DE1 board. The T_b and T_s signals are achieved by using a counter to divide the frequency of the 50Mhz clock into 9600Hz and 960Hz respectively. This was achieved using the Altera DE1 board and VHDL code. T_B and T_S are used for synchronising the systems different subsystems.

To test these signals a parallel to series converter was made in VHDL using the signals. The input to the parallel to series converter was controlled with a set of switches.

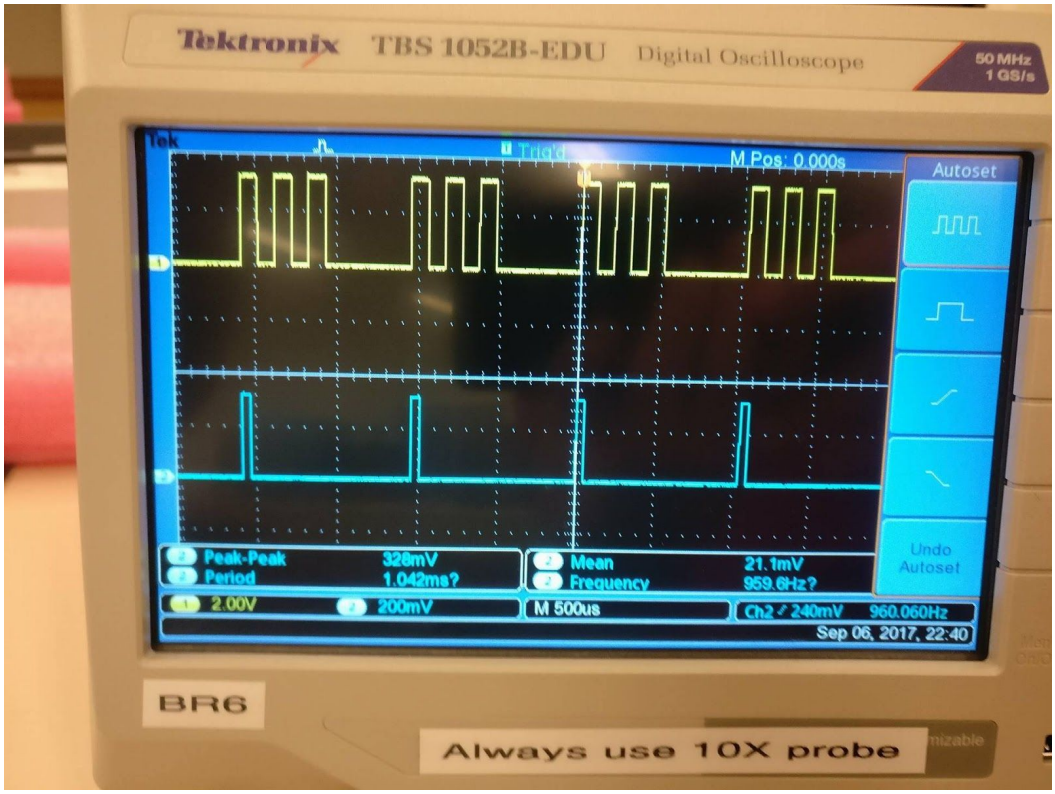

Figure 3: The upper pulse train is based on T_b, the lower pulse train is T_s. The word sent is 10101000

Signal	T_s	T_b
Amplitude (V)	3.3	4.1
Period time (ms)	1.042	0.1042
Frequency (Hz)	959.8	9598
Pulse width (µs)	52	19.4

Table 1: Measurements of pulses from T_b and T_s

See appendix 1 for VHDL code.

2.2 D/A converter

A D/A converting circuit is created to convert an 8-bit digital signal to an analog output. To achieve this a DAC0808 digital-to-analog converter is used for the initial conversion along with a TL072 OP amplifier to amplify the output. The input to the DAC0808 is created using the DE1 FPGA with VHDL code. The input to the DAC as previously mentioned consists of a 8-bit digital binary signal, which means that the resulting signal inside the DAC can be given a value ranging from 0-255 using 256 different levels. The output of the DAC is sent out from pin 4 on the DAC which is amplified in the TL072 circuit to obtain a value between 0-10V corresponding linearly to the the value 0-255 produced by the DAC dependent on which input bits A_1-A_8 are active.

The resistor R_5 is implemented in a later construction of the system in order to obtain bipolar signals. This works due to the current through the resistor R_5 corresponds to half the max current on pin 4 (I_4) with opposite polarity. Therefore the new value on V_{DA} can range from -5V to +5V.

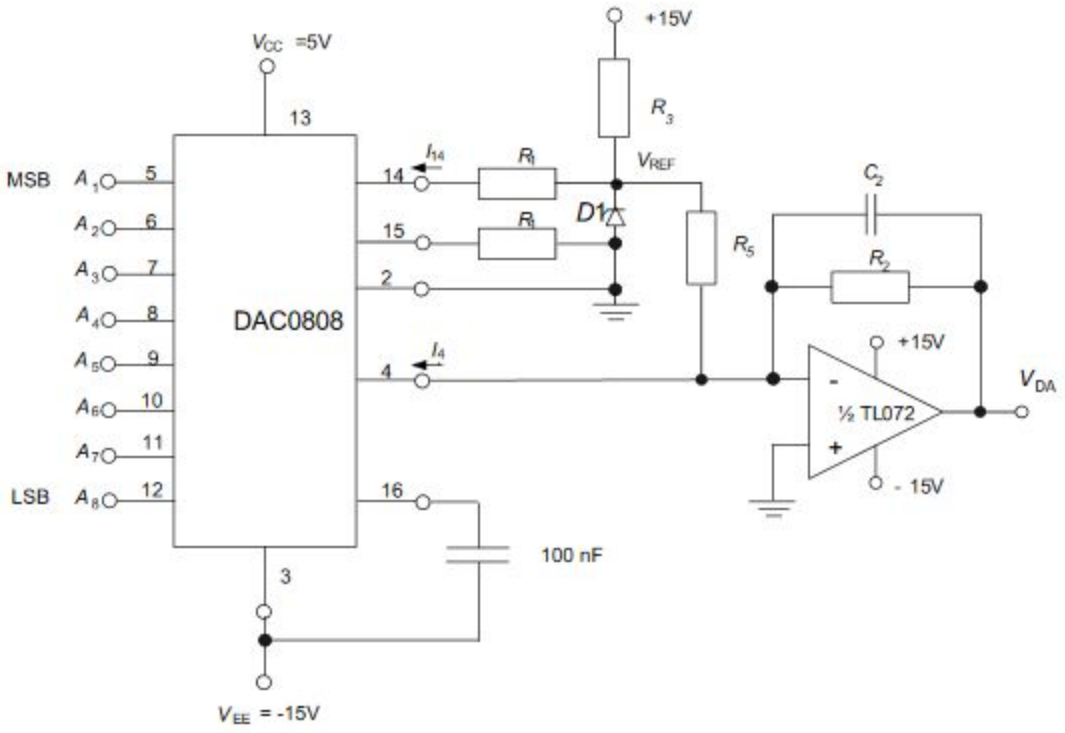

Figure 4: D/A Converting circuit (Bipolar-signals)

The circuit has the following specifications:
R1 = 2800 Ohm
R2 = 5000 Ohm
R3 = 680 Ohm

Tests were ran to verify the integrity of the circuit.

Digital Input (decimal)	Expected analog output (V)	Real analog output (V)
0	0	0.004
16	0.625	0.656
32	1.25	1.311
64	2.5	2.618
128	5	5.248
255	10	10.43

Table 2: Testing results of the D/A Converter.

Testing was also performed on the circuit to measure the slew rate

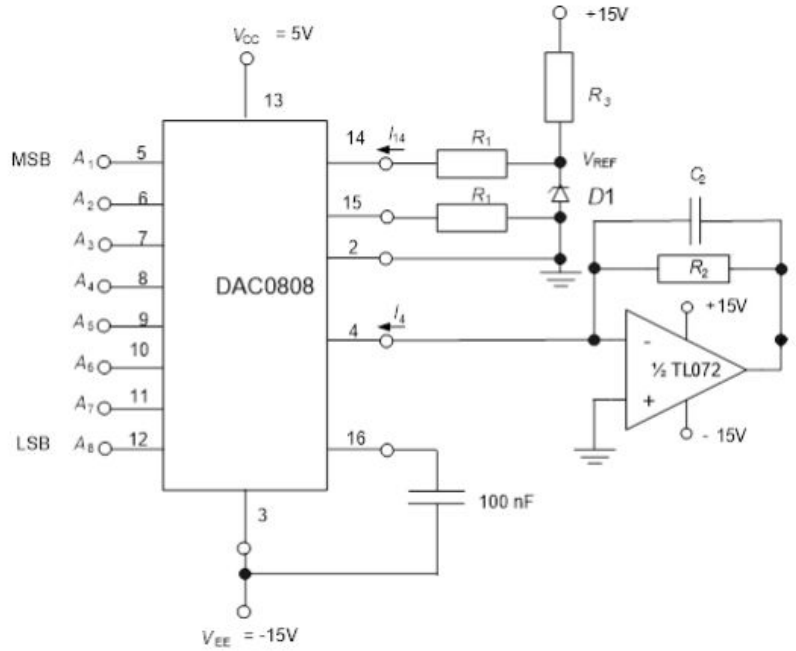

Figure 5: A 33pF (C2) capacitor is added to the circuit

The slew rate of the TL072[1] is 13 v/us. However the circuit has a slew rate of 2.89v/us. When adding a 33pF capacitor parallel to R2 the slew rate significantly improved to 8.89v/us.

A new counter was made to test the 8 bit input. This counter increased the input incrementally from 0-255 with a frequency of 100Hz. As expected the output ranged from 0V to 10V in a sawblade pattern.

Figure 6: Analog output for the counter-test

See appendix 2 for VHDL code.

[1] http://www.ti.com/lit/ds/symlink/tl072a.pdf 2017-10-20, Page 19.

2.3 A/D converter

The purpose of the AD Converter is to convert an analog input into an 8 bit digital signal. This is achieved using a Successive Approximation Register (SAR) which attempts to create a digital signal as close to the analog input as possible. This is achieved by setting each bit of A to 1 one by one, and using the LM311 circuit to check if V_{DA} is larger or smaller than V_{SH}. The result (D) is then sent back to the SAR and becomes the permanent value of the prior bit of A.

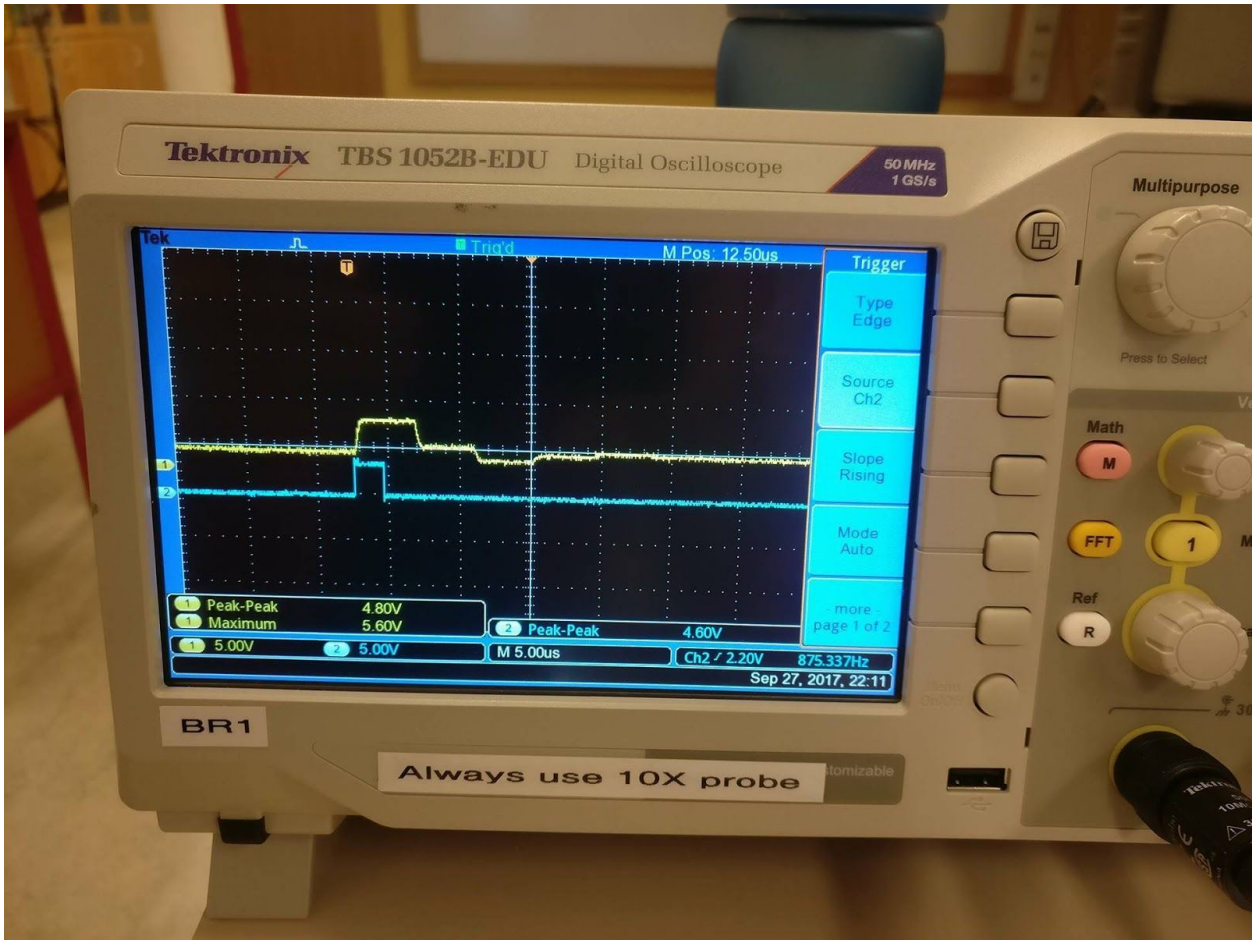

Figure 7: A single set of SAR guesses. Yellow is V_{DA}, blue is T_s

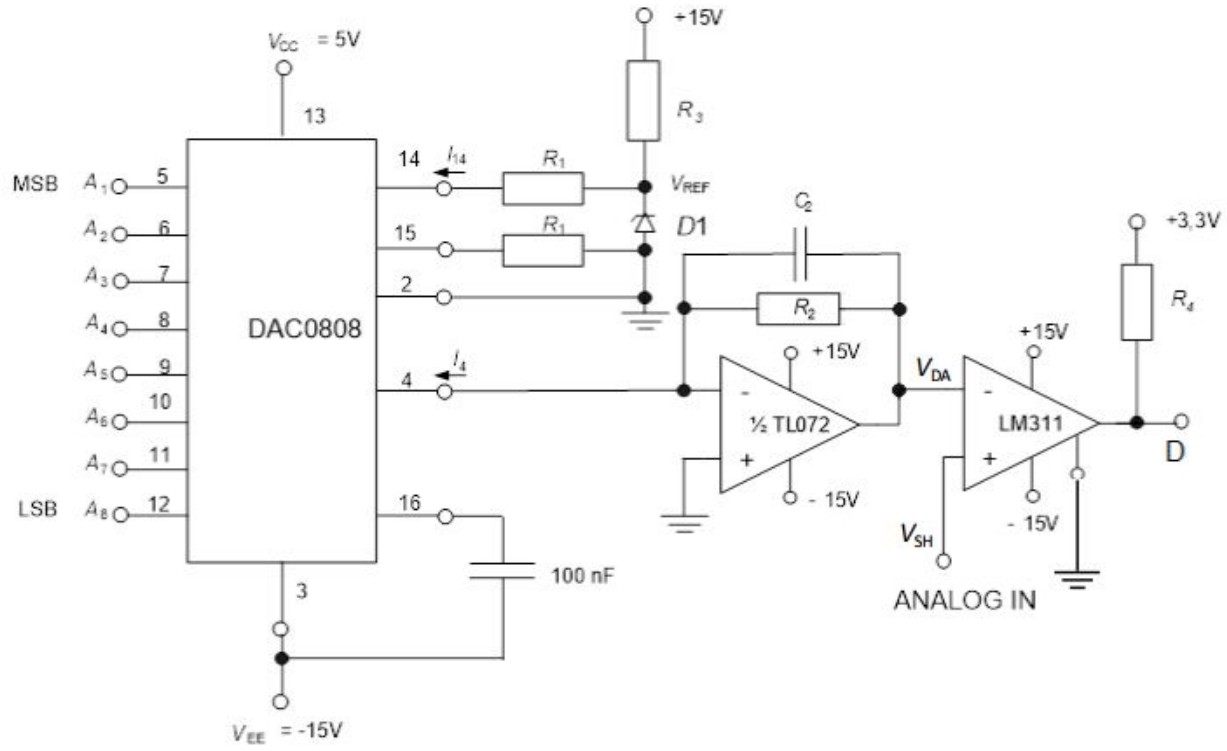

Figure 8: A/D Converting circuit. V_{SH} represents the analog input.

This circuit is rather similar to the D/A converter (see figure 4), the addition of the LM311 comparator and a change in VHDL code allows the circuit to perform the opposite operation, Analog-to-digital conversion.

A series of tests were ran on the circuit to verify correct functionality.

A 5V DC signal was added as an input resulting a digital output of 122, which represents 4.77 volts.
The circuit was found to have a larger delay of 390ns when testing an input signal close to 5V, than the 240ns delay with an input signal of 3V. This is because the LM311 comparator has a harder time comparing two similar signals than two very different signals and the first test input is always 5V.

See appendix 3 for VHDL code.

2.4 Sample and Hold

The sample and hold circuit is implemented before the A/D-converter and is used because the construction of the A/D-converter requires a constant input in order for the converter's compare function to operate as intended. This is achieved using the sample-and-hold circuit LF398 which is illustrated in figure 8 below.

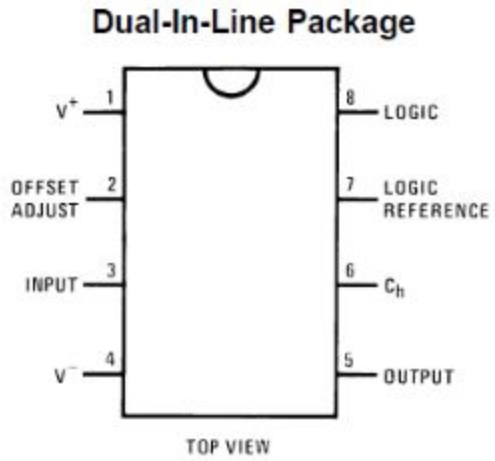

Figure 9 Sample and hold circuit LF398

The S/H-circuit samples if the difference between the pin 8 and 7 is above 1.4V and holds if the difference is below 1.4V as follows: Sample if V_8-V_7 > 1.4V and hold if V_8-V_7 < 1.4V. This can be utilized through assigning the trig signal T_S to pin 8 and assigning ground to pin 7. At the start of each conversion there is a pulse sent on T_S which causes the circuit to sample the input and then holding it while waiting for a new pulse from T_S. While holding the sample it sends it as output on pin 5 for the A/D-converter to use. When the conversion is done the S/H-circuit receives a new pulse from T_S and the process is repeated. The circuit works by storing the sampled voltage in a capacitor used as an analog memory which can deliver the sampled voltage for a short while.

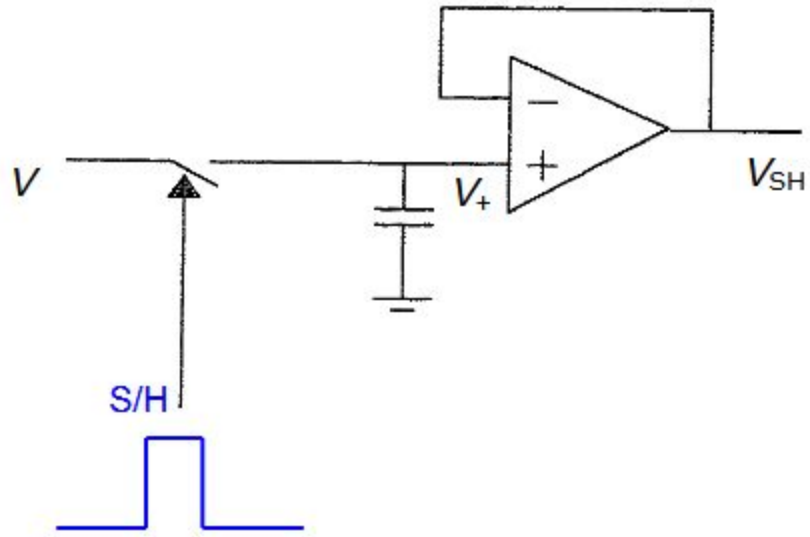

Figure 1 sample and hold circuit

Figure 10 illustrates the basic function of the sample and hold circuit. V_+ is always equal to V_{SH} and when a new pulse from T_S arrives the switch closes for a short while and the capacitor at V_+ samples the new voltage.

2.5 Serial transmitter

The serial transmitter is implemented since one of the specifications for the system is to be able to communicate serially using a RS232-cable. The input which the serial transmitter receives from the A/D-converter is in parallel form which it has to convert for serial transmission. The whole operation for the serial transmitter is achieved by coding it in VHDL. The process of the serial transmitter is illustrated in figure 11 below.

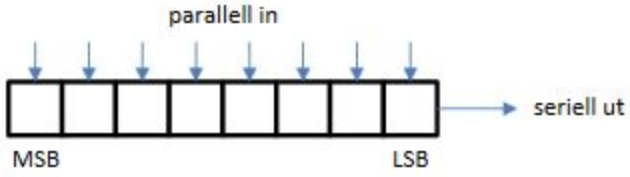

Figure 11 illustration of serial transmitter

For every conversion and pulse sent from T_S a new value is saved in the serial transmitters shift register. The trig signal from T_B starts the serial transmission by sending out the least significant byte from the shift register since that is the standard for the RS232 communication. When the least significant bit is sent the whole register shifts right a step and the process is repeated with a pulse from T_B and the second least significant bit. This process goes on till all the bits saved in the register have been sent out and a new pulse is given by T_S.

In order to satisfy the RS232-standard a start and stop bit is implemented before and after the datastream. The startbit corresponds to a 0 and the stopbit corresponds to a 1, in theory there is 10 bits sent serially but only 8 of those are the generated datastream.

See appendix 3 for code VHDL-code.

2.6 Serial receiver

Similar to the serial transmitter, the RS232 standard is used to communicate with a transmitting unit. For information about the RS232 standard see 2.5.

To transcode the serial information received into a parallel stream of information the receiver has to record each serial bit. Being in sync with the transmitter is key. The start and stop bits are what allows the system to sync with the transmitter.
Once the start of a data stream has been identified the receiver waits half of the bitrate to make sure the signal is stable, and not in the process of switching between a high or low signal.
Each state handles a single bit of information, loading the bit into a vector which is later output into the circuit. To make sure the system does not fall out of sync with the transmitter the states S7 and STOP does not progress unless the input is 1 (stop bit) or 0 (start bit) respectively.

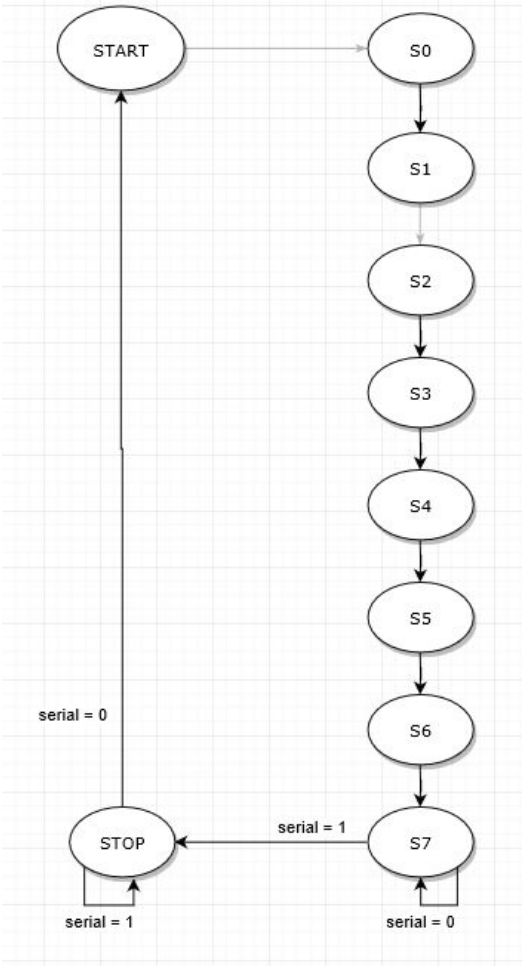

Figure 12: State diagram of the receiver.

To achieve bipolarity of the output signal the resistor R5 is added to the circuit. (see figure 4). R_5 needs to be correctly dimensioned to ensure an even bipolarity. This was calculated to be 5600 Ohms.

$V_{ref} / I_4 = R_5 = 5600\Omega$

The receiver was tested by connecting the system to a PC, and using the RS232 standard some ASCII signs were sent to the receiver. The results were the following:

ASCII-Sign	Received bit sequence (hex)	Analog output (V)
P	50	-1.82
?	3F	-2.54
å	3F	-2.54
!	21	-3.75
=	3D	-2.59
A	41	-2.41

Table 2: Results from the RS232 test with a PC.

Notably, the letter å was shown as 3F, or a question mark. This is because the system does not support extended ASCII. All results had a negative voltage. This is because the normal ASCII table only goes to 127 (decimal) and bipolarity has been added to the circuit. All inputs under 128 (decimal) result in a negative analog output.

2.7 Audio amplifier

Two sets of amplifiers are used. One for the microphone and one power amplifier for the speakers.

The microphone amplifier is used because the microphone outputs a very low power signal. If we want to reliably send it to a receiver we have to amplify it. The JFET circuit below was used.

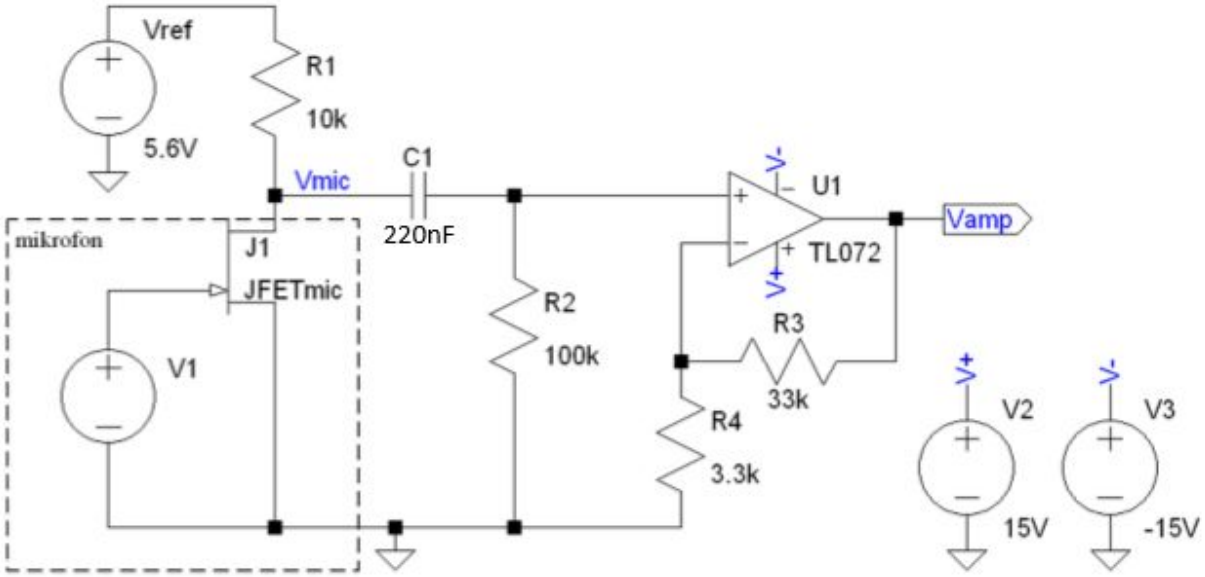

Figure 13: Circuit for the microphone amplifier.

Simulations in LTspice showed the microphone amplifier having full amplification in the range of 20Hz - 105kHz. This is more than ample enough since human hearing only reaches about 20kHz

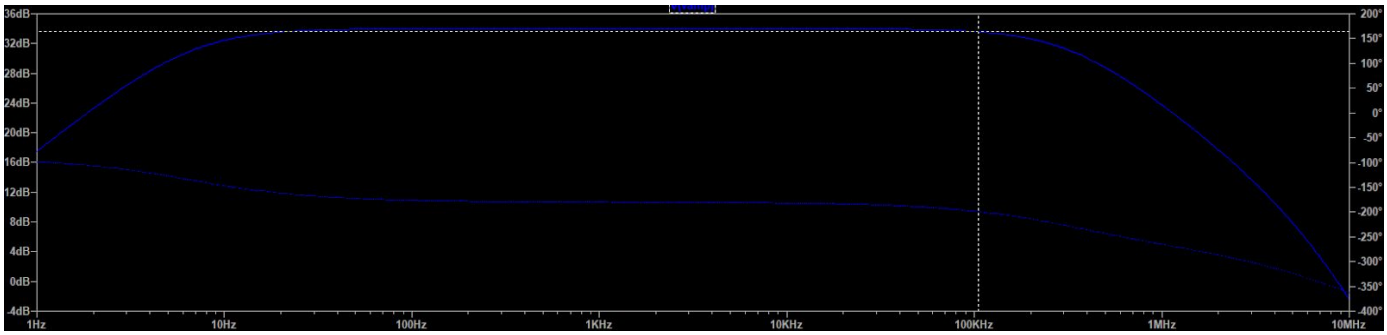

Figure 14: LTspice simulation of the cutoff frequencies for the microphone amplifier.

The working point was found to be 3.08V

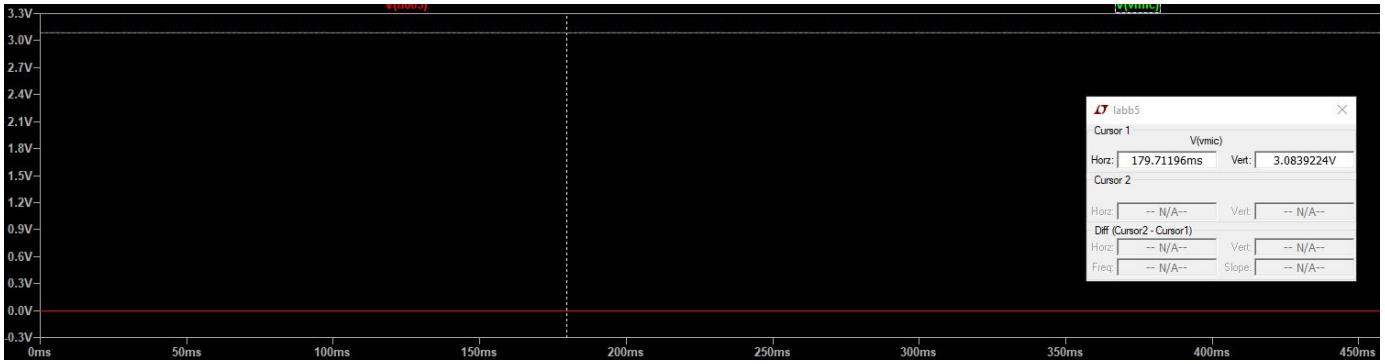

Figure 15: LTspice transient simulation.

To be able to output the sound signals to a speaker a power amplifier must be used.

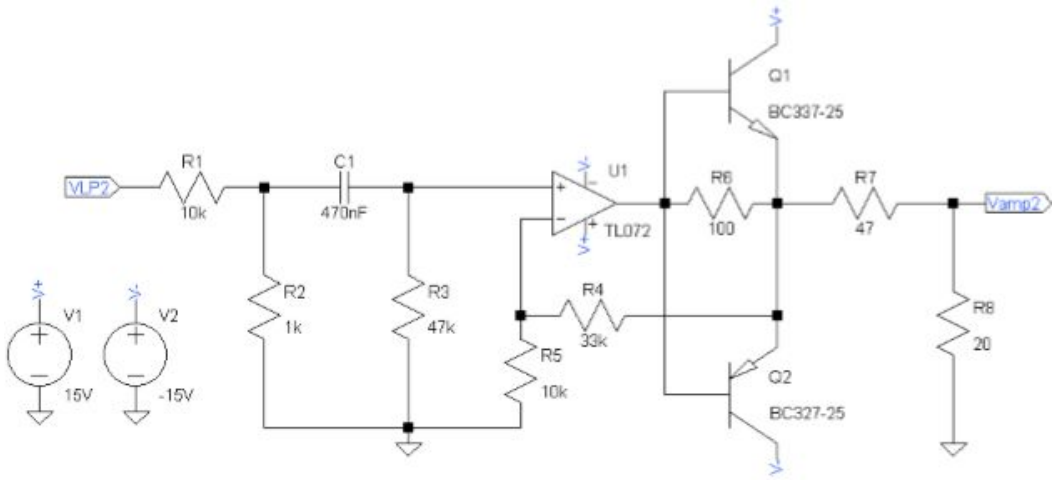

Figure 16: CMOS power amplifier.

Before use some calculations and simulations were made. The power amplifier was calculated to have a 0.11 voltage amplification. The formula for calculating the amplification can be seen below.

$V_1 / V_{LP2} = (R_3 // R_2) / ((R_1 // R_3) + R_1) = A_1$
$A_{OP} = (R_5 + R_4) / R_5$
$A_2 = R_8 / (R_7 + R_8)$
$A_{TOT} = A_1 * A_{OP} * A_2 = 0.11$

When simulated in LTSPICE the resulting voltage amplification was 0.112. See figure 16 for simulation circuit.

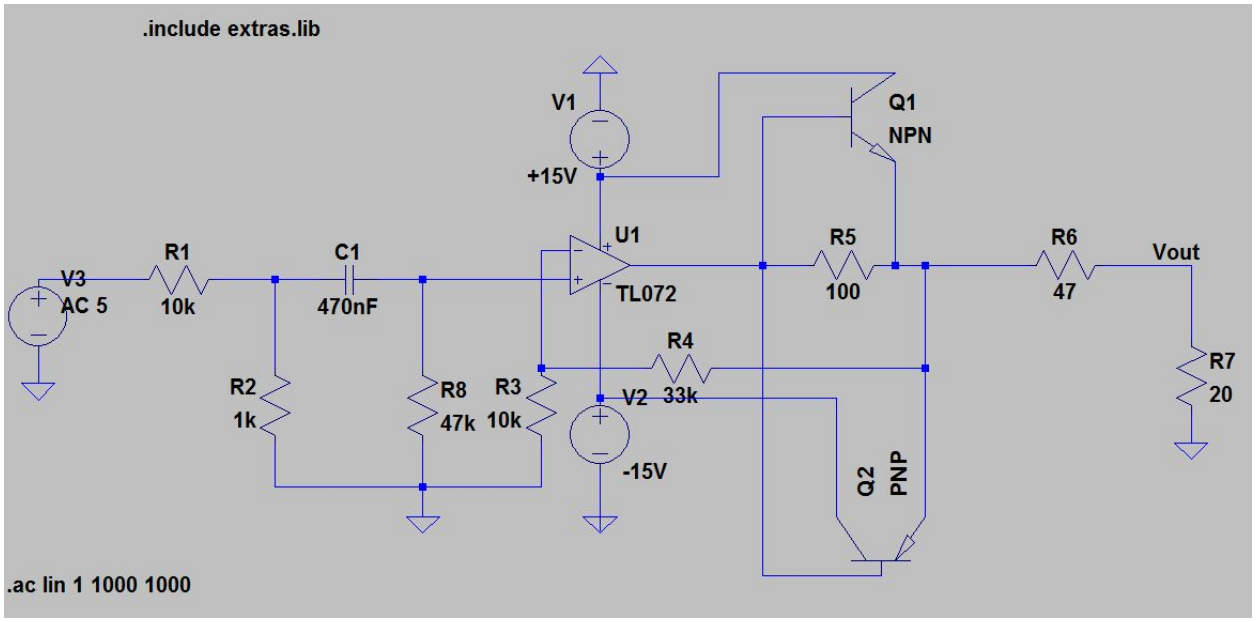

Figure 16: LTspice simulation of the CMOS amplifier.

2.8 LP filter

Since the sampling frequency is set to 24kHz a lowpass filter is needed with the cutoff frequency at half of the sampling frequency which corresponds to 12kHz which satisfies our requirements for the system. The reason why the cutoff frequency is set to half of the sampling frequency can be explained using the Nyquist-Shannon sampling theorem. In short a sampling frequency has to be at least double the signal's frequency in order to synthesize a reasonable signal. The filter constructed is a fourth order Butterworth filter using two Sallen-key links. The specifications for the filter is to have an upper cutoff frequency at 12kHz and the circuit should be designed using resistors from the E12-series and capacitors using the E6-series.

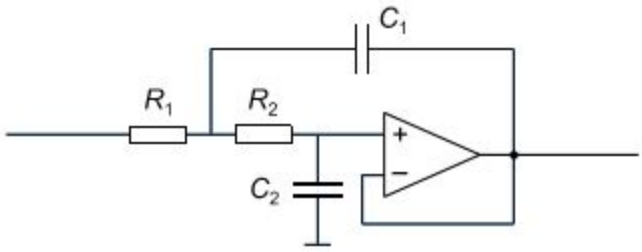

Figure 17: Sallen key filter

The transfer function for a Sallen Key filter is the following

$$H(s) = \frac{1}{1 + s\left(R_1 + R_2\right)C_2 + s^2 R_1 R_2 C_1 C_2}$$

The transfer function of a 4th grade Butterworth filter is

Ordning	Polynom $P(a)$
1	$1+a$
2	$1+1.414a+a^2$
3	$1+2a+2a^2+a^3$ $=(1+a)\left(1+a+a^2\right)$
4	$1+2.613a+3.414a^2\|+2.613a^3+a^4$ $=\left(1+0.765a+a^2\right)\left(1+1.848a+a^2\right)$

Table 3: Denominator for each transfer function of a butterworth filter, up to the 4th order

Combining these two formulas the following result are achieved.

First Sallen-Key filter:
$(R_1 + R_2) * C_2 = 0.765 / (2*12kHz * \pi)$
$R_1 * R_2 * C_1 * C_2 = (1 / (2 * 12kHz * \pi))$

Second Sallen-Key filter:
$(R_1 + R_2) * C_2 = 1.848 / (2*12kHz * \pi)$
$R_1 * R_2 * C_1 * C_2 = (1 / (2 * 12kHz * \pi))$

Through testing a satisfactory result was found with

First filter:
$R_1 = R_2 = 1500$ Ohm
$C_1 = 22nF$

Second filter:
$R_1 = R_2 = 3900$ Ohm
$C_2 = 3.3$ nF

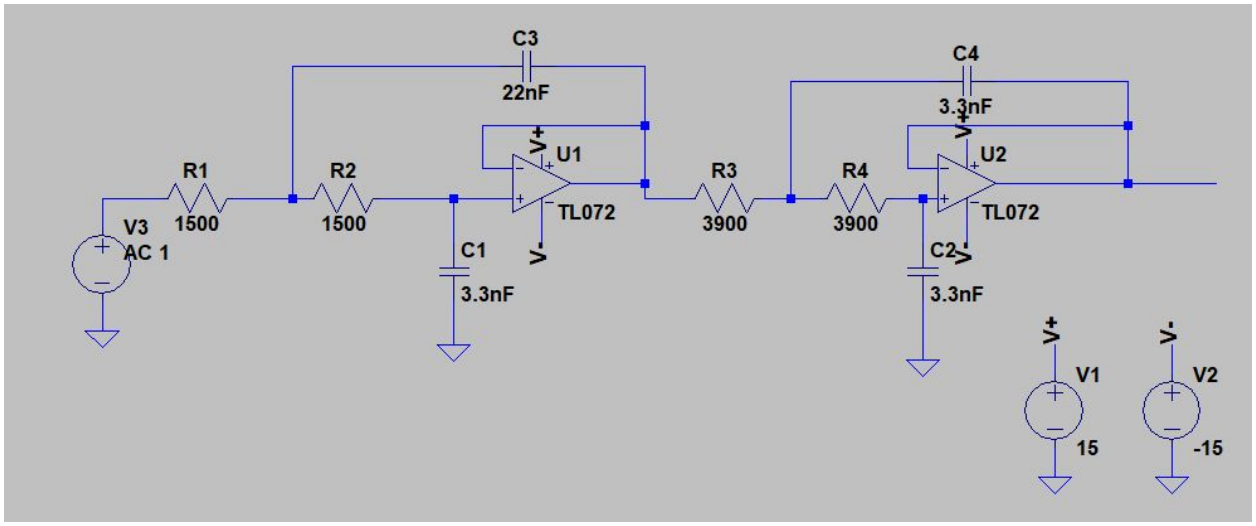

Figure 18 Schematic LP-filter

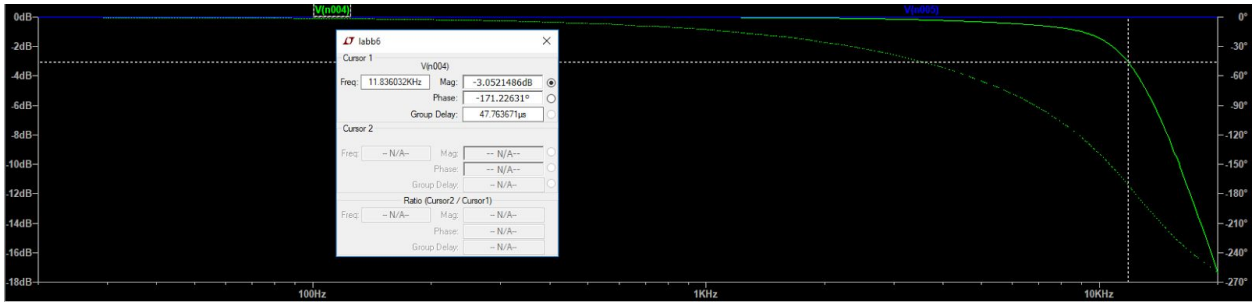

Figure 19 Simulation LTSpice LP-filter

The simulation of the LP-filter in LTSpice is how above in figure 18 & 19. The cutoff frequency for the filter is estimated to 11,8 KHz which is near the specified 12kHz.

3. Test and verification

A few tests were ran on a full system with two units, a transmitter and receiver.
The first test was finding the transfer dynamic, the difference between the max amplitude without distortion and the lowest distinguishable amplitude.
The highest amplitude without distortion was found to be $10V_{PP}$. The lowest amplitude was found to be 300mV. This gives us a transfer dynamic of 70dB.

The second test was finding the bandwidth of the system, or rather the upper cutoff frequency. Around 4.9kHz the amplitude of the receiver was half of the input from the transmitter. As such 4.9kHz is the upper frequency.
This gives a bandwidth range of 20Hz - 4.9kHz.

A more human test of the system found that the audio quality was lacking. It is reminiscent of a poor radio signal. Listening to music was a subpar but fully functional, experience.

The final test performed was testing how well the transfer system worked if some of the transfer bits were nonfunctional. It was found that the audio system worked without major reduction in audio quality with one or two bits faulty. However with less than 6 functional bits the results varied greatly. Some genres of music played without much issue, others were completely distorted.

4. Discussion

The finished product is functional and working after specifications but we are not totally satisfied with the quality of the produced audio. We believe that a big factor for this is the dimensions design of all the components and subsystems of the circuit. Since it is a complex system a small margin of error early in the system might evolve to a bigger error later in the circuit. Since most of the resistors and capacitors is taken from a specific series the actual used value for the components might not correspond exactly to the precalculated wanted values and instead a adjacent value is taken. Because the circuit is implemented on a coupling plate in the course over a few weeks there might be bad connections and imperfections emerging in the cabling and components remaining unseen which in turn adds insult to injury.

In conclusion the product works as intended but we are not completely satisfied with the quality of the produced audio. We are not completely sure about where the errors stem from but we have some theories as mentioned above.

5. Reflection

The labs worked well, were worthwhile and we enjoyed the design. The prep work was on a good level, not too easy or too hard. However we found the second portion of lab 3, the transmitter, to be quite confusing. The RS232 standard was only lightly touched upon in lectures and that portion of the lab would have been significantly easier if some examples were readily available, like the other labs. In conclusion we really appreciated this course.

6. Appendix

1 VHDL code counter

```
process(reset,clk50)            -- Tb Counter, 9600Hz
begin
        if reset='1' then
           Tb <= '0';
           counterB <= ( others => '0');
        elsif rising_edge(clk50) then
                if counterB=0 then
                     Tb <= '1';
                elsif counterB=1 then
                     Tb <= '0';
                elsif counterB=5207 then
                counterB<= (others => '0');
                end if;
                 counterB <= counterB + '1';
        end if;
end process;

process(reset,clk50)            -- Ts counter, 960Hz
begin
        if reset='1' then
           Ts <= '0';
           counterS <= ( others => '0');
        elsif rising_edge(clk50) then
                if counterS=0 then
                        Ts <= '1';
                elsif counterS= 2603 then
                        Ts <= '0';
                elsif counterS=52079 then
                        counterS<= (others => '0');
                end if;
                        counterS <= counterS + '1';
        end if;
end process;

process(Ts,Tb)                  -- Parallel to series coverter
begin
        if Ts='1' then
                bitmonster<=SW;
        elsif Tb = '1' then
                bitmonster(6 downto 0)<=bitmonster(7 downto 1);
        end if;

end process;
end architecture;
```

2 VHDL code D/A - converter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity upgl is port (
LEDR: out std_logic_vector (7 downto 0);
JP: out std_logic_vector (7 downto 0);
SW: in std_logic_vector (7 downto 0);
clk50: in std_logic;
trig: out std_logic;
upp_ner: in std_logic;
reset: in std_logic
);
end entity;

architecture arc of upgl is
signal counter: std_logic_vector(18 downto 0);
signal raknare: std_logic_vector (7 downto 0);
begin
LEDR <= raknare;
JP <= raknare;

process(reset,clk50)
begin
        if reset='1' then
           trig <= '0';
           counter <= ( others => '0');
        elsif rising_edge(clk50) then
                if counter=0 then
                     trig <= '1';
                counter <= counter + '1';
                elsif counter=1 then
                     trig <= '0';
                counter <= counter + '1';
                elsif counter=5207 then
                counter<= (others => '0');
                else
                 counter <= counter + '1';

                end if;
        end if;
end process;
process(clk50)
  begin
    if reset='1' then
      raknare<=(others => '0');
    elsif rising_edge(clk50) then
     if counter=0 then
                if upp_ner='1' then
                       raknare<=raknare+1;
                elsif upp_ner='0' then
                       raknare<=raknare-1;
       end if;
          end if;
    end if;
end process;
end architecture;
```

3 VHDL code D/A Converter, Serial transmitter

```
 process(reset ,clk50)
begin
        if reset='0' then
           clk2 <= '0';
           counter2 <= ( others => '0');
        elsif rising_edge(clk50) then

                if counter2 = 5207 then
                     clk2 <= '1';
                          counter2 <= (others => '0');

                else
                     clk2 <= '0';
                          counter2 <= counter2 + '1';
                end if;


        end if;
        Tb <= clk2;
end process;
 process(reset ,clk50)
begin
        if reset='0'  then
           state <= IDLE;

  elsif rising_edge(clk50) and clk2 = '1' then
     case state is
        when IDLE =>
        if clk = '1' then
         Q <= "10000000"; -- half of the maximum value
          state <= S7;
        END if;
        when R7 =>
         Q(7) <= D; -- MSB
         Q(6) <= '1';
          state <= S6;
        when S6 =>
         Q(6) <= D;
         Q(5) <= '1';
          state <= S5;
        when S5 =>
         Q(5) <= D;
         Q(4) <= '1';
          state <= S4;
        when S4 =>
```

Lab3

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

entity lab3 is port (
        LD: BUFFER std_logic_vector(7 downto 0);
        clk50: in std_logic;
        reset: in std_logic;
        Tb: out std_logic;
        Ts: out std_logic;
        D: in std_logic;
        Dtx: out std_logic;
        Rtx: out std_logic;
        UT: out std_logic_vector(7 downto 0));
end entity;

architecture arch of lab3 is
type StateType is (IDLE,S0,S1,S2,S3,S4,S5,S6,S7,STOP); -- States
signal state: StateType;
signal counter: std_logic_vector(28 downto 0);
signal counter2: std_logic_vector(28 downto 0);
signal clk2: std_logic;
signal clk: std_logic;
signal reg: std_logic_vector(9 downto 0);
signal Q: std_logic_vector(7 downto 0);

begin
 process(reset,clk50)
begin
        if reset='0' then
           clk <= '0';
           counter <= ( others => '0');
        elsif rising_edge(clk50) then
           counter <= counter + '1';
                if counter < 3 then
                     clk <= '1';
                elsif counter =< 85 then
                     clk <= '0';
                          counter <= (others => '0');
                end if;

                end if;
        Ts <= clk;
end process;
```

```
          Q(4) <= D;
          Q(3) <= '1';
         state <= S3;
         when S3 =>
          Q(3) <= D;
          Q(2) <= '1';
          state <= S2;
         when S2 =>
          Q(2) <= D;
          Q(1) <= '1';
          state <= S1;
         when S1 =>
          Q(1) <= D;
          Q(0) <= '1';
          state <= S0;
         when S0 =>
          Q(0) <= D; --LSB
          state <= STOP;
         when STOP =>
         LD(7 downto 0) <= Q(7 downto 0);
         state <= IDLE;
         when others =>
          state <= IDLE;
      end case;
   end if;
end process;

UT(7 downto 0) <= Q(7 downto 0); --sends the values to (JP1)


 process(clk50,reset)
begin
         if reset='0' then
            reg <= ( others => '0');
         elsif rising_edge(clk50) then
            if clk = '1' then
           reg(0) <= '0';
           reg(9) <= '1';
           reg(8 downto 1) <= LD;
            end if;
         if clk2 = '1' then
           reg(8 downto 0) <= reg(9 downto 1);


         end if;
         end if;
```

```
 Dtx <= reg(0);
 Rtx <= reg(0);
 end process;
end architecture;
```

4 Serial reciever

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

entity mottagare is port(
reset,serial_async, clk50: in std_logic;
parallel: out std_logic_vector(7 downto 0);
Ts: out std_logic;
LEDR: out std_logic_vector(7 downto 0)
);
end entity;

architecture state_machine of mottagare is
type statetype is (STOP,START,S0,S1,S2,S3,S4,S5,S6,S7);
signal state, next_state: statetype;
signal Q, next_Q, next_parallel: std_logic_vector(7 downto 0);
signal counter: std_logic_vector (12 downto 0);
signal counter2: std_logic_vector (16 downto 0);
Signal serial, serial_semisync: std_logic;

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 = 2603 then -- After half the clock then... 2603 (104)
                state <= next_state;
                Q <= next_Q;
                parallel <= next_parallel;
        elsif counter = 5207 then
                counter <= (others => '0'); -- clockrate: 50Mhz/5208 = 9601Hz  5207 (207)
        end if;
end if;
end process;

process(clk50,reset)
begin

if reset='0' then
        counter2 <= (others => '0');
elsif rising_edge(clk50) then
        counter2 <= counter2+1;
```

```
elsif rising_edge(clk50) then
        counter2 <= counter2+1;
        if (counter2 = 0) then
                Ts <= '1';
        elsif counter2 = 20 then -- After 5% of clock then... 2603
                Ts <= '0';
        elsif counter2 = 207 then
                counter2 <= (others => '0'); -- clockrate: 50Mhz/52080 = 960.1Hz 52079
        end if;
end if;
end process;


process(state, serial,Q)
begin
--next_Q <= Q;
case state is
        when STOP =>
                if (serial = '0') then
                        next_state <= START;
                end if;
        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;
                LEDR <= Q;
                if serial = '1' then
                        next_state <= STOP;
                else
                        next_state <= S7;
                end if;
        when OTHERS =>
                next_state <= STOP;
end case;
end process;
```