Carl Larsson, 920808-1910

Marcus Viken, 930221-1736

Electrical systems

-Lab report

Summary

The purpose for this series of laborations was to get a deeper understanding for how a electrical system may work. This was done by designing an audio transmitting system which contained both a transmitter and a receiver. The goal was to be able to send and receive sound which in fact the final system was able to do, but with some disturbances. Overall, the project was successful.

Introduction

The main reason for making this project was to test our theoretical and practical knowledge in digital and analog electronics, and put that knowledge into use by creating a functioning electrical system. The system built was a sound transmission system that mainly consisted of two parts, one receiver and one transmitter. Both the receiver and transmitter consisted of smaller subsystems which will be designed separately part by part during 6 laborations. Every part of the system will first be theoretically calculated and then tested in the laboratory. The different subsystems designed can be seen below and will each be presented part by part in this lab report.

Subsystems:

- Counter
- D/A converter
- A/D converter
- Sample and Hold
- Serial transmitter
- Serial receiver
- Audio amplifier
- LP filter

The sound transmission system created will meet the following conditions:

- Frequency range 20-12000 Hz
- Resolution 8-bits

A block diagram over the complete system with both transmitter and receiver can be seen in figure (1) below:

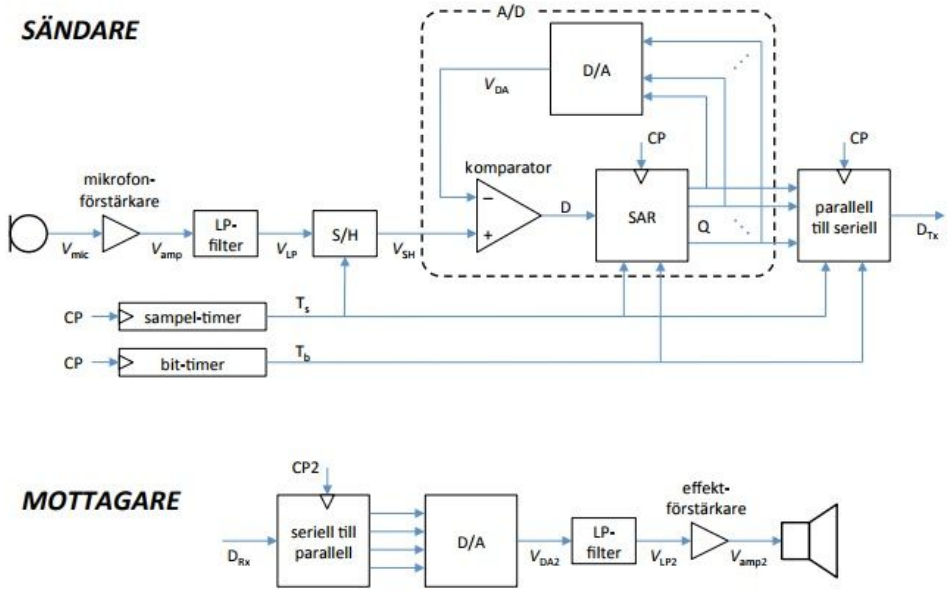

Figure (1): Block diagram for the complete audio transmitting system.

Subsystems

In this audio transmitting system, several subsystems were designed and tested separately to simplify the debugging in the case that something did not work properly. All the software was designed and simulated in a VHDL-simulator called Modelsim. The software was then downloaded on the hardware board DE1. The circuits was designed on an ordinary breadboard with a set of different components. Some circuits was first designed and simulated in ltSpice before they were implemented on the breadboard.

Counter

The counter was designed during the first laboration. There are actually two counters implemented in the system. One each for the two trigger signals, sample-timer and bit-timer. The sample-timer (t_s) will steer the sampling frequency and the bit-timer (t_b) will steer the frequency at which the sampled bit strings would be transmitted out.

Both t_s and t_b were designed using the integrated crystal oscillator with a frequency of 50 MHz. The desired frequency of t_s were 960 Hz respectively 9600 Hz for t_b. The duty cycle for t_s were supposed to be 5 % of the pulse. t_b were only supposed to have a short pulse of 20 ns before going back to zero. The following equation was used to determine the limits for the two counters:

$$\frac{Current\ frequency}{Desired\ frequency} = limit\ for\ the\ counter \qquad (1)$$

With 50 MHz as the current frequency and 960 Hz respectively 9600 Hz for the two timers, the limit for t_s was evaluated to 52080 and the limit for t_b was evaluated to 5208. To determine the duty cycle for t_s, the evaluated limit was divided with 20. The quotient from this division was evaluated to 2604. As long as the value of the counter was lower than 2604, the signal was high. When the value of the counter exceeded 2604, the signal went low. The t_b signal were only supposed to be high during one period of the inbuilt 50 MHz clock oscillator. To obtain this the t_b signal was high as long as the counter was lower than or equal to 1. When the counter exceeded this limit the signal went low. The complete VHDL-code for the the two timers can be seen in Appendix I.

Some measurements was later performed on the two timers. The results can be seen in table (1) below:

Table (1): Measurements performed on t_s and t_b.

Signal	**t_s**	**t_b**
Amplitude [V]	3,4	5,2
Periodtime [s]	1,04 m	104 µ
Frequancy [Hz]	960	9600
Pulsewidth [s]	50 µ	20 n

D/A converter

The D/A converter was designed during the second laboration. The main idea with a D/A converter is to convert a digital signal into an analog. The D/A converter consists of a DAC0808 card and a TL072 subwoofer.

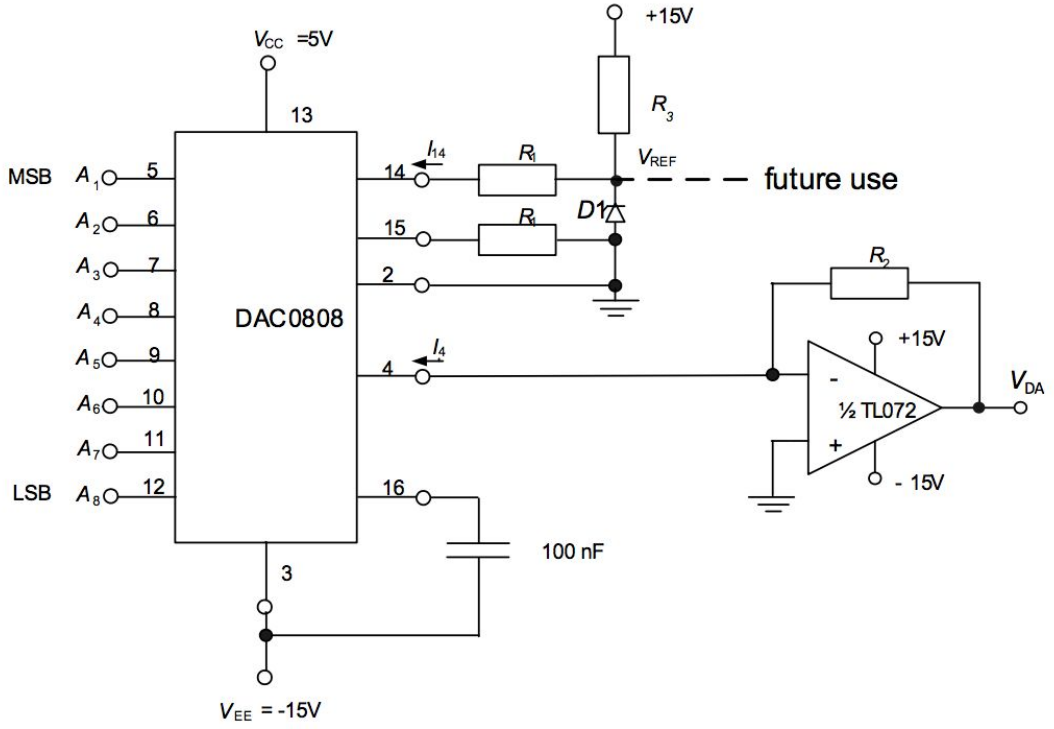

Figure (2): Circuit of D/A converter

The D/A converter should be restricted to the following specifications.

- Current I_{14} approximately 2 mA.
- V_{DA} 0-10 volt
- Current through R_3 lower than 20 mA
- Current through diode D1 higher than 10 mA and max effect 0,5 W.

The resistance R_1 was calculated to 2800 Ω with formula (2) which can be seen below. V_{REF} = 5,6 V because of the zener-diode and the the current I_{14} was specified to be 2 mA. Resistors from the E12-series was available so therefore R_1 was selected to 2700 Ω as it was the closest value.

$$R_1 = \frac{V_{REF}}{I_{14}} = \frac{5,6}{2*10^{-3}} = 2800\ \Omega \qquad (2)$$

The resistance R_2 was calculated to 4840 Ω with formula (3). V_{DA} is 0-10 volt and A is 255. In the laboratory the resistance was selected to 4700 Ω. In the formula below is N = the number of bits and MAX = maximum digital value.

$$R_2 = \frac{V_{DA} * R_1 * N}{V_{REF} * MAX} = \frac{10 * 2700 * 256}{2*10^{-3} * 255} = 4840\ \Omega \qquad (3)$$

The resistance R_3 was calculated with formula (4). The Current I_3 through R_3 was specified to be lower than 20 mA and the voltage over resistance R_3 is known by taking the potential 15 V minus the potential in V_{REF}. In the laboratory the resistance was selected to 560 Ω in the E12-series.

$$R_3 = \frac{15 - V_{REF}}{I_3} = \frac{15 - 5{,}6}{18 * 10^{-3}} = 522\ \Omega \qquad (4)$$

During laboration 2, the output from the D/A converter was measured and compared to the theoretically calculated values. In the middle column of table (2), the calculated values can be seen. In the column to the far right, the measured values can be seen.

Table (2): Calculated and measured values of analog output.

Digital control signal (decimal)	**Calculated analog output (V)**	**Measured analog output (V)**
0	0	0,001
16	0,609	0,607
32	1,218	1,216
64	2,437	2,436
128	4,874	4,868
255	9,71	9,69

In laboration 2 a binary counter was created in VHDL. The VHDL-code for the binary counter can be seen in appendix II. The binary counter should count 1 bit 100 times per second up to 256. The counter is counting up or down depending on a switch which is either high or low. The simulation of the binary counter can be seen in figure(4).

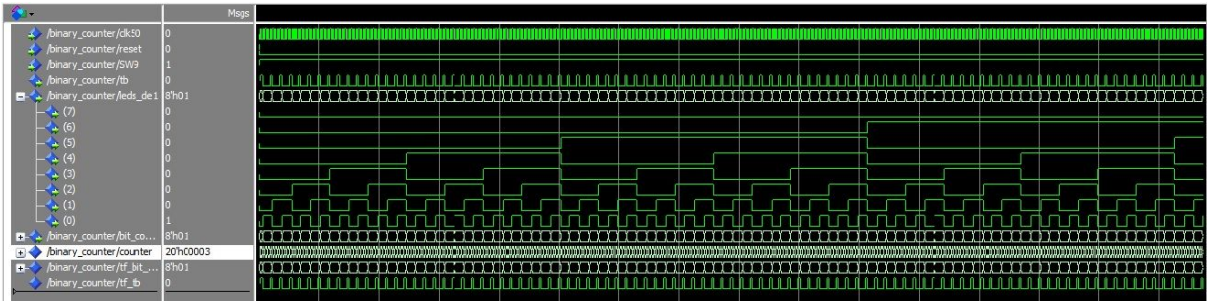

Figure (4): Simulation of the binary counter.

The VHDL-code of the binary counter was then downloaded to the DE1-board in the laboratory. A measurement was performed on the output with an oscilloscope and the signal that was detected was the sawtooth signal that can be seen in figure (5). The sawtooth's period-time was calculated to approximately 2.56 s and was later measured to 2.56 s in the laboration.

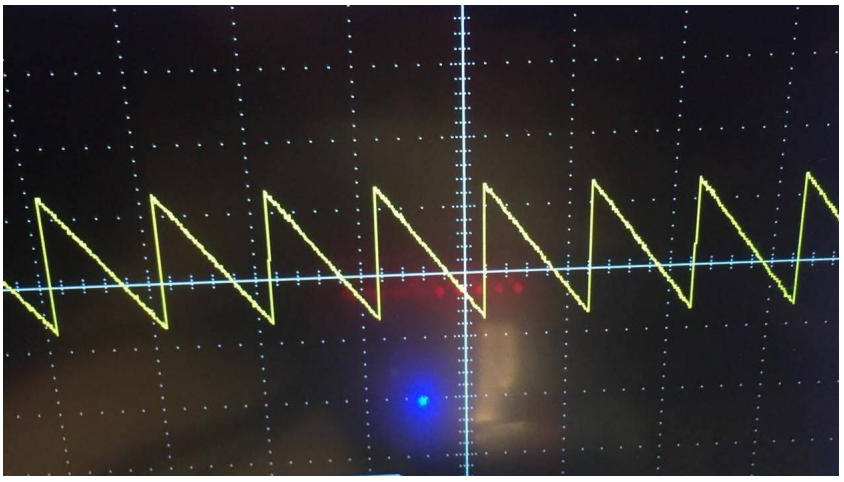

Figure (5): Sawtoothwave.

The binary counter was later speeded up and when the signal was studied a little bit closer on the oscilloscope, a overshoot was detected when the counter reset from “11111111” to “00000000”. The overshoot can be seen in figure (6).

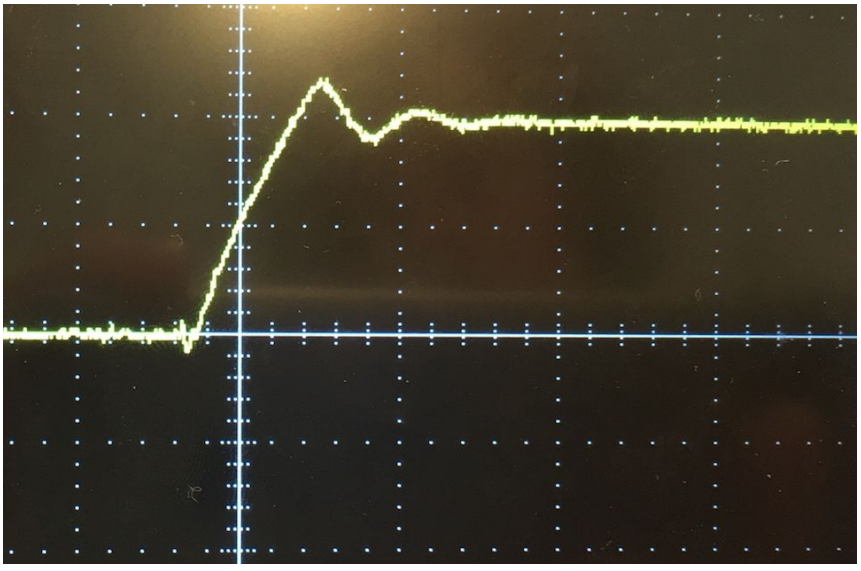

Figure (6): Overshoot of the sawtoothwave.

A capacitor C_2 with capacitance 33 pF was placed parallel with R_2 to prevent this overshoot from happening. As can be seen from figure (7), the overshoot is much smaller.

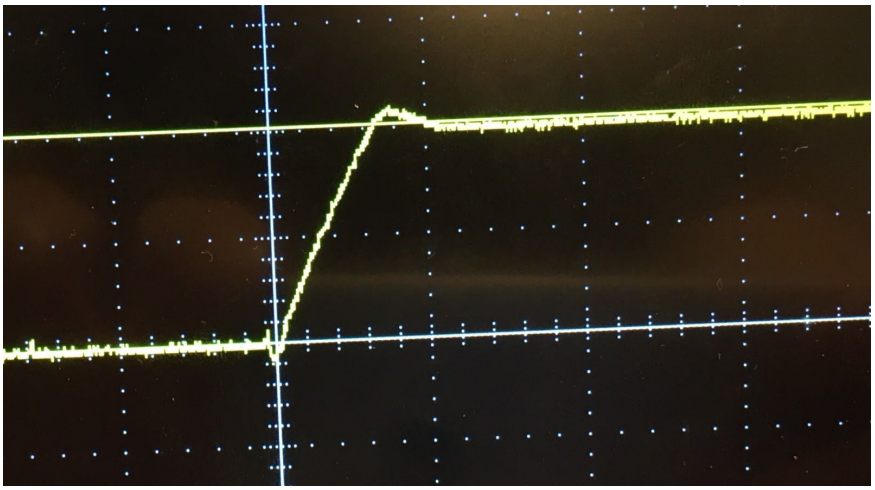

Figure (7): Overshoot with capacitor C_2 in gear.

One of the components in the D/A converter is a TL072 amplifier. The slew rate for the amplifier was calculated by measuring the time it takes for a step signal to reach from zero to its final value. According to the datasheet for the amplifier the slew rate should be approximately 13 V/µs. The slew rate without capacitor C_2 in gear was measured to 9,6 V/µs and the slew rate to 10.2 V/µs.

A/D converter

The A/D converter was designed during the third laboration. As can be seen from figure (1) it contains three main blocks, a D/A converter, a Successive Approximation Register (SAR) and a comparator. The D/A converter were designed earlier in laboration 2. The comparator LM311:s minus input was connected to the output from the D/A converter. In this laboratory the plus input on the comparator was connected to a waveform generator which generated a DC voltage that could be variated between 0 and 10 Volts. The complemented wiring diagram can be seen in figure (8) below:

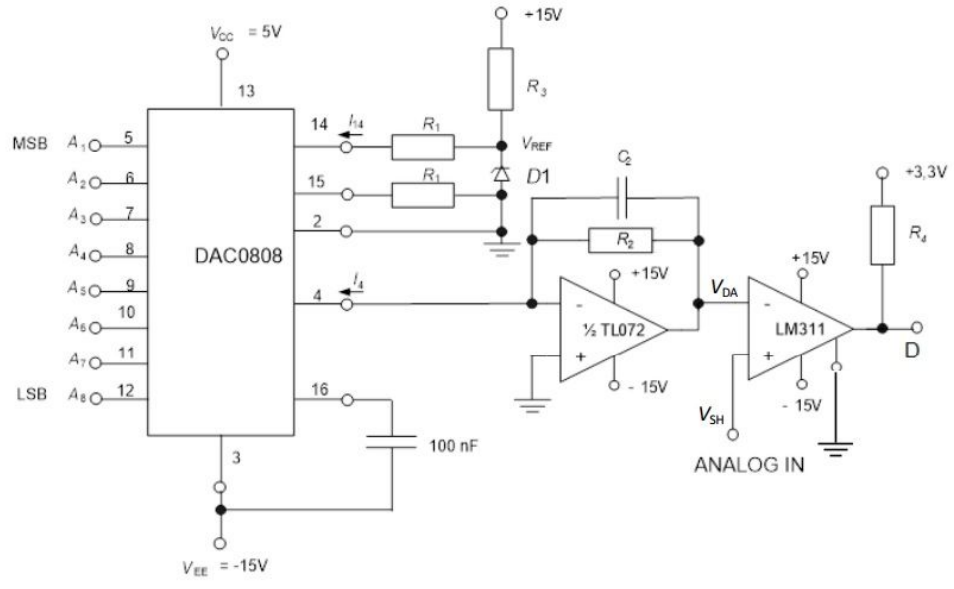

Figure (8): The D/A converters wiring diagram complemented with the comparator LM311.

The resistor R_4 was selected to 1 kΩ. The waveform generator was connected to the node V_{SH} in figure (8). The circuit from the current I_4 to the output D was then implemented into LTspice to determine the delay between these two points. The sketch can be seen in figure (9) below:

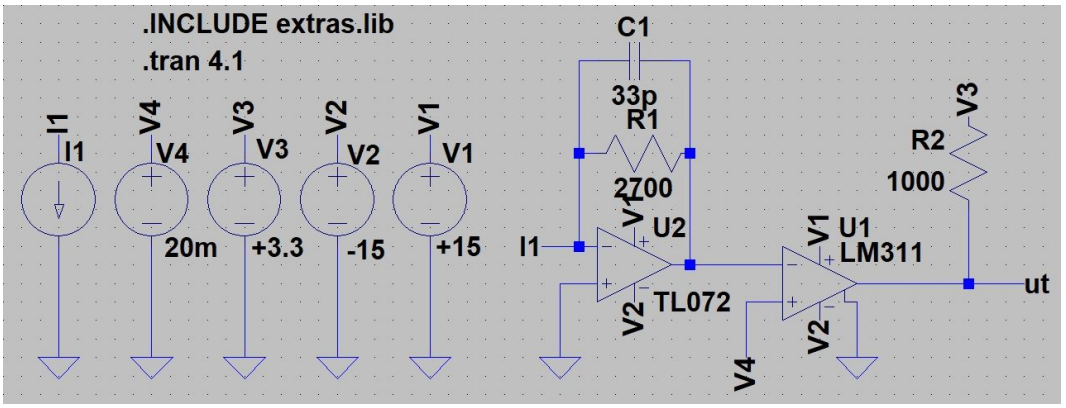

Figure (9): Circuit from I_4 to D.

The input V_{SH} was set to a DC voltage of 20 mV. The input current I_4 was set as a squarewave with a period of 2 s and a minimum value of 0 A and a maximum value of 1 mA. The circuit was simulated as a transient for 4,1 s. The input current and the output voltage was plotted in the same time graph so that the elapsed time between the two could be measured. The result from when I_4 went from 0 A to 1 mA can be seen in figure (10) below:

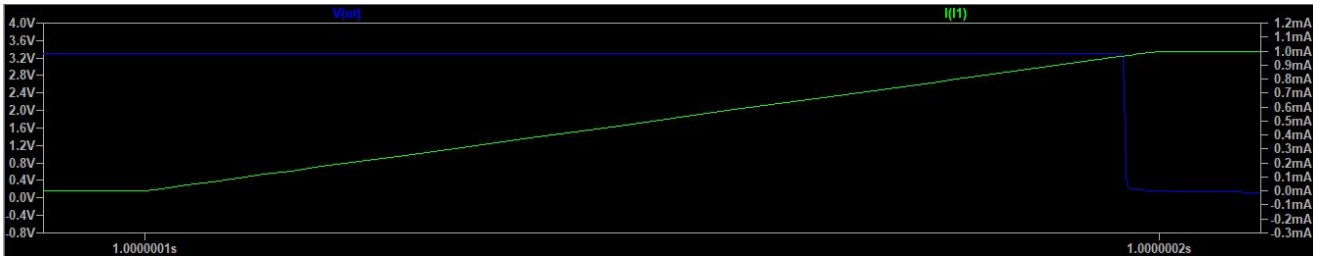

Figure (10): Graph from I_4 to D when I_4 went from 0 A to 1 mA.

As can be seen from the figure, the elapsed time from when the input I_4 starts to change until that the output has been stabilized on to it's new value, is approximately 100 ns. The other case, when I_4 went from 1 mA to 0 A can be seen in figure (11) below:

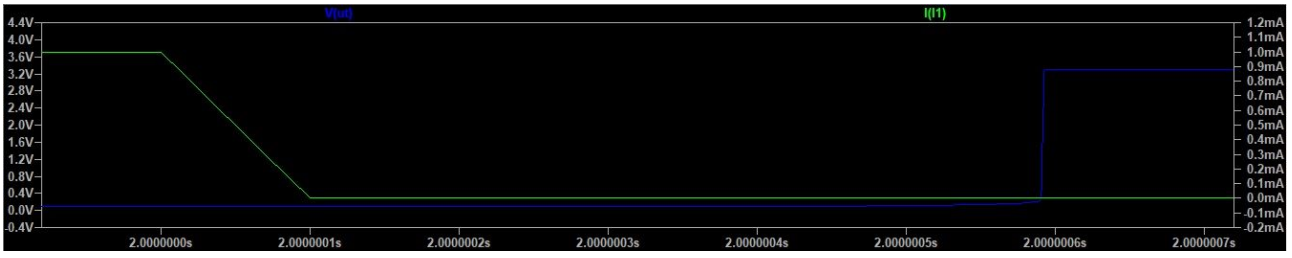

Figure (11): Graph from I_4 to D when I_4 went from 1 mA to 0 A.

In this case, the delay was longer. As can be seen from the figure, it takes approximately 600 ns for the output to stabilize onto its new value from when the input starts to change.

In the next part of the simulation, the delay from I_4 to V_{DA} in figure (8) was measured. The same circuit and simulation command as for the previous simulation was used. The result from when I_4 went from 0 A to 1 mA can be seen in figure (12) below:

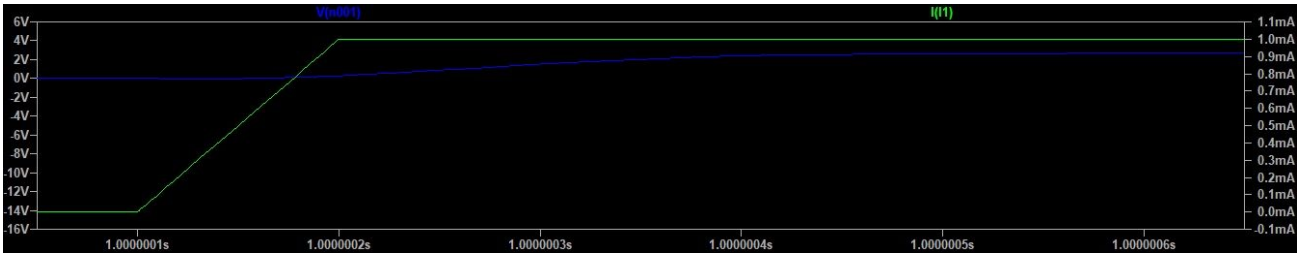

Figure (12): Graph from I_4 to V_{DA} when I_4 went from 0 A to 1 mA.

In this simulation, the time delay was approximately 350 ns as can be seen from figure (12). The result from when I_4 went from 1 mA to 0 A can be seen in figure (13) below:

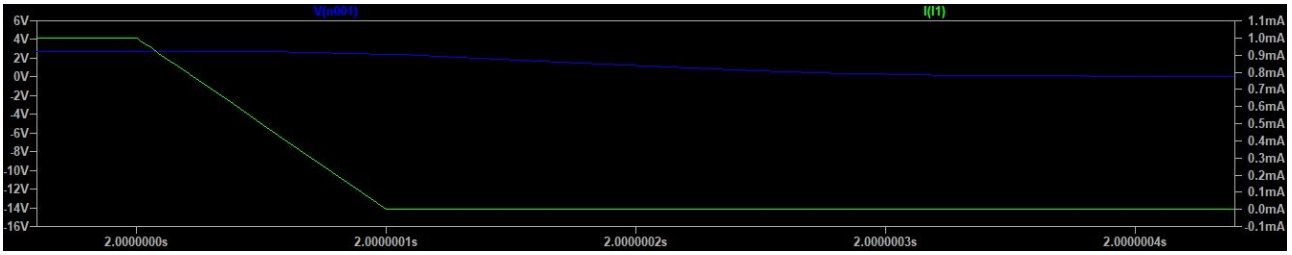

Figure (13): Graph from I_4 to V_{DA} when I_4 went from 1 mA to 0 A.

What can be determined from the simulations is that it takes longer time for the system to react when the two input voltages into the comparator are far apart. The voltage V_{SH} into the plus input on the comparator are in this simulation constantly 20 mV. When the input voltage on the minus input goes from 0 up to 2,7 V the system reacts faster than when the input goes from 2,7 to 0 V.

The next step was to design the Successive Approximation Register (SAR) which is a digital control unit. It was designed as a VHDL code in modelsim. To simplify the design process, a moore machine of the function was sketched. The sketch can be seen in figure (14) below:

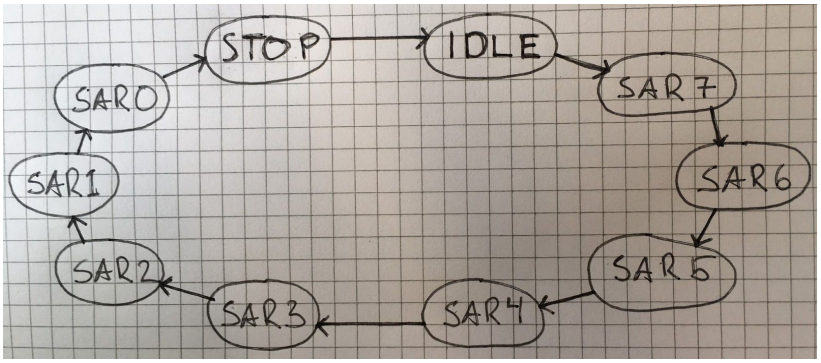

Figure (14): Graph over the state machine of type Moore.

The state diagram was implemented into code which can be seen in Appendix III. The function is split up into 10 states. Each of the states SAR7 - SAR0 determines one bit in the 8-bit word that is sent out from the A/D-converter, with SAR7 as the most significant bit and SAR0 as the least significant bit. The 8-bit word is sent out parallel from the A/D. The function is steered by the two trigger signals t_s and t_b. The function was also simulated in Modelsim before it was implemented on the DE1 board. The result from the simulation can be seen in figure (15) and (16) below:

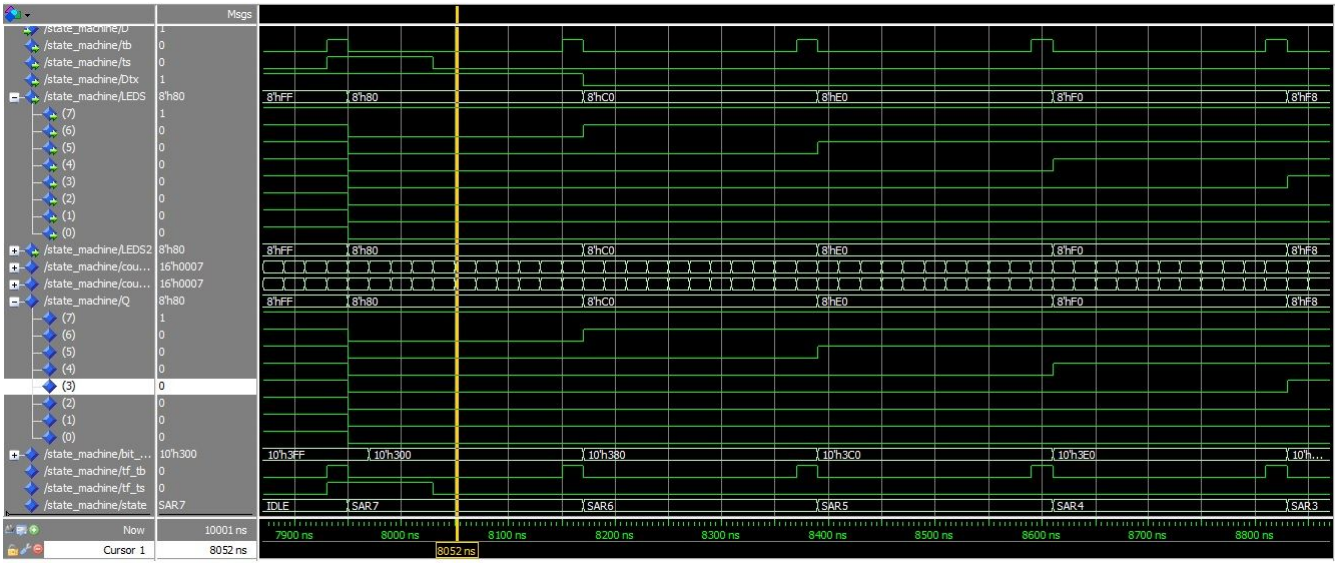

Figure (15): Timegraph from the simulation of the SAR (part 1).

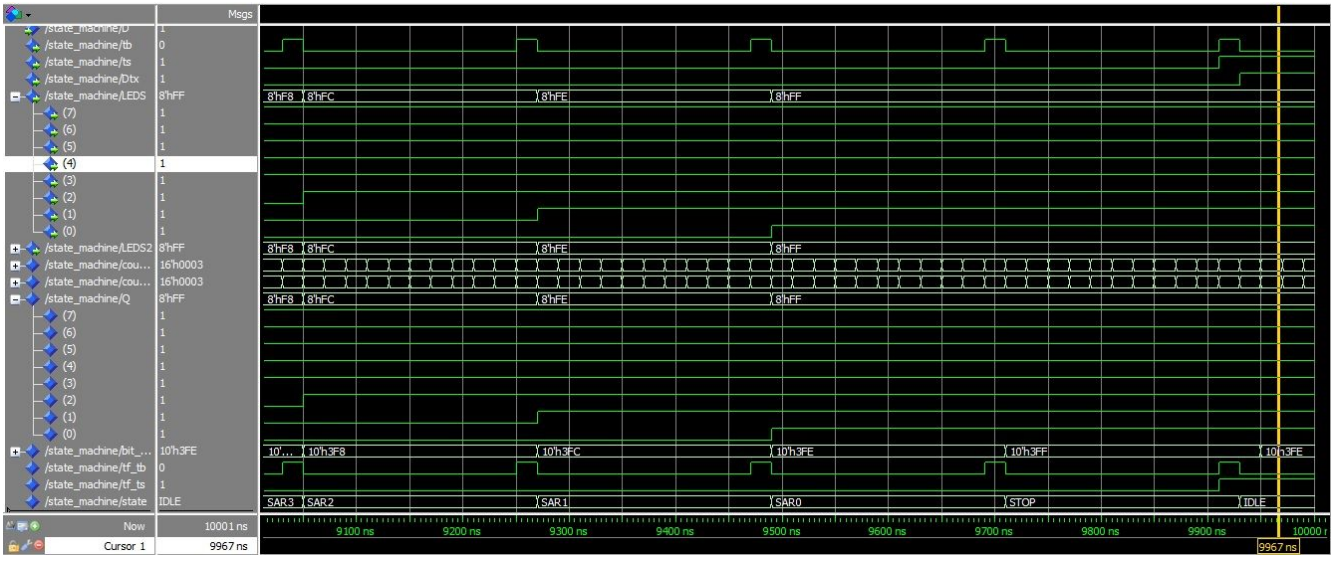

Figure (16): Timegraph from the simulation of the SAR (part 2).

In figure (15), the results from the state IDLE to SAR3 can be seen. In the figure (16), the remaining states from SAR3 back to IDLE can be seen. Together from the two figures, one complete simulation cycle from the SAR function is generated. In the simulation, the input signal that later on will come from the comparator, was set to always input a 1. This will also

result into that the 8-bit parallel output will be set to 11111111 when a complete SAR cycle is finished. This can be seen in figure (16).

When all the subsystems were designed and simulated separately it was time to realize the complete system on actual hardware. The circuit was designed on a breadboard and the VHDL-program was downloaded onto the DE1-board. A signal generator was used to simulate the analog input signal V_{SH}. It was set to generate a DC-voltage to be variated between 0 to 10 V. First the DC-voltage was set to 5 V. The 8-bit word that was generated from the A/D-converter was 1000 0100. This was represented on the LEDs on the DE1-board which can be seen in figure (17) below:

Figure (17): 8-bit word represented by the DE1-boards LEDs.

The theoretically calculated 8-bit word from a voltage of 5 V would be 1000 0000 so the result is a little bit of but still good enough.

The next part was to connect the trigger signal t_s and the V_{DA} voltage to the oscilloscope to try to verify the SAR circuits function. A 1,8 V input on V_{SH} gave the following time graph on the oscilloscope:

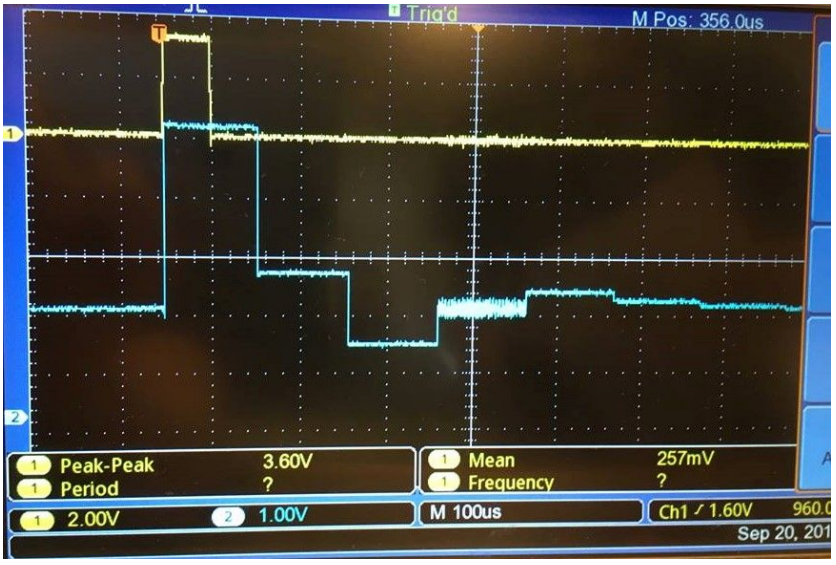

Figure (18): Time graph from when the SAR-function determines the digital word for 1,8 V.

In figure (18) the yellow curve represents the t_s signal. As can be seen from the figure, a positive flank on t_s will start the conversion cycle for the SAR-register, represented by the

blue graph. After a cycle has begun, a negative flank on the blue graph represents a 0 and a positive flank represents a 1, with the first flank as the most significant bit and so on. So for this particular cycle it can be seen from the graph the 7 most significant bits would be 0011 000. The last bit is hard to determine because of the resolution but theoretically it would be a 0.

The next part was to measure the delay from the digital input Q to the output D which can be seen in figure (1). The input Q is the digital word that is sent in on the pins A_1 to A_8 in figure (8). The signal generator was set to generate a DC voltage of 3 V. The input pin A_1 and the output D was connected to the oscilloscope, but still with triggering on t_s. The following graph was generated on the oscilloscope:

Figure (19): Delay from A_1 to D.

The delay was measured to 260 ns. If the reference voltage V_{SH} was increased from 3 V to 5 V, the delay was measured to 590 ns. The delay increases because of the slew rate of the comparator. It takes longer time for the comparator to decide which of the two values that are the biggest if the two values are close to each other. In the previous laboration, a delay of ca 100 ns was measured for the comparator. This delay is significantly lower than the delay from A_1 to D. This is actually quite logically thus that the circuit from A_1 to D includes the comparator. Compared to the simulations that were done earlier the results are quite close. In the simulation the DAC0808 were not connected to the circuit which also has some delay in it.

Sample and Hold

In laboration 4 a sample and hold circuit was implemented in the system. This type of circuit is used because the analog input need to be constant through the whole conversion circuit of the A/D converter. The sample and hold circuit stores the current and voltage in an analog memory and keeps it constant for a short period of time.

The circuit was tested in the laboration. A input signal was connected in the shape of a triangular wave with frequency 500 Hz and amplitude 5 volts. The input and output signal

was measured with the oscilloscope and the result could be seen in figure (20). The turquoise wave shows the input signal and the yellow one describes the output signal. The difference between the input and the output signal is that the S/H circuit samples a value from the input and holds it for a short period of time. The values that have been sampled and hold can be seen in the shape of short squares in the yellow signal.

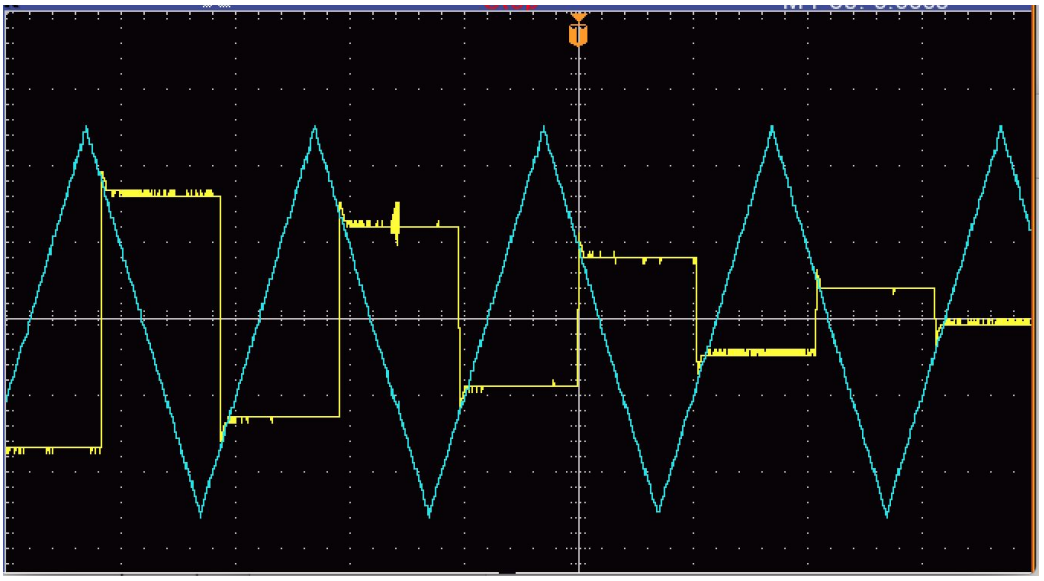

Figure (20): Input and output signal of the S/H circuit plotted on the oscilloscope

Serial transmitter

The first part of the serial transmitter was designed during the first laboration. The main idea with a serial transmitter is to transform parallel binary code into a serial binary code. To be able to send a serial binary code the transmitter in this case is triggered by the sample-timer (t_s) and the bit-timer (t_b). When (t_s) is high a parallel bit pattern is stored in a vector, whenever (t_b) is triggered the least significant bit in the vector is shifted out and assigned to an output signal.The code for this function can be seen in Appendix IV.

In the third laboration the serial transmitter was tested. The VHDL-code for the parallel to serial converter was added to the SAR-function that was designed in laboration 3. The complete code for the transmitting system can be seen in Appendix V. The A/D-converter was used as the transmitter and connected via a DB9 contact to a computer which was used as a receiver. The signal was transmitted according to the asynchronous standard RS232. The signal generator was set to deliver a DC voltage of 5 V, which is represented by the digital word 1000 0011. The word is received as the hexadecimal value 83. This can be seen in figure (21) below:

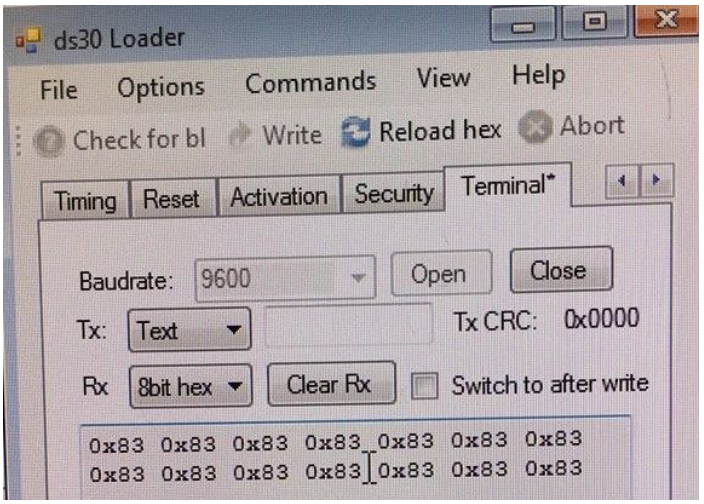

Figure (21): The received signal as a hexadecimal value.

The value can be translated into the ASCII-sign f.

Serial receiver

In laboration 4 a serial receiver was created in VHDL code. The serial receiver should transform serial binary code into parallel code. The serial binary code that is transmitted is a 10-bits number where the first bit is a start-bit(0) and the last bit is a stop-bit(1). The serial receiver is created in VHDL code as a state machine which can be seen in appendix VI. If the state machine receives a start-bit(0) it starts to transmit the incoming data into parallel code. There are 10 different states, one start and stop state and 8 states for the 8 bit number that should be transformed. In each state except the start and stop state the incoming data is sent to a separate output. When the state machine has reached the stop state it will stay there until it receives a new start-bit(0). The state machine in RS232-standard could be seen in figure(22).

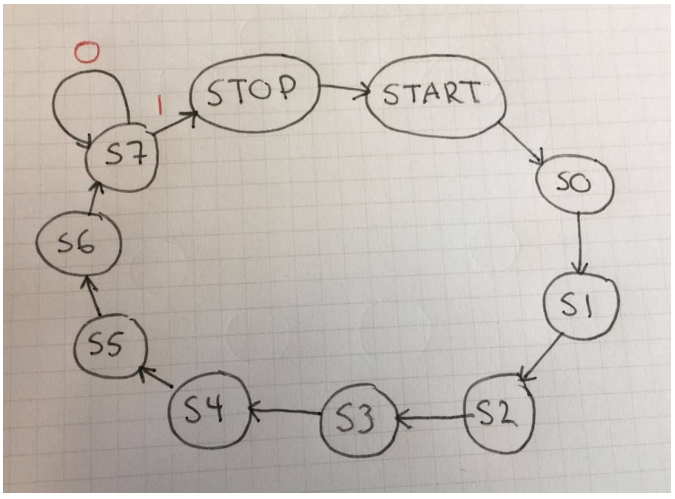

Figure (22): Graf over the state machine of type RS232.

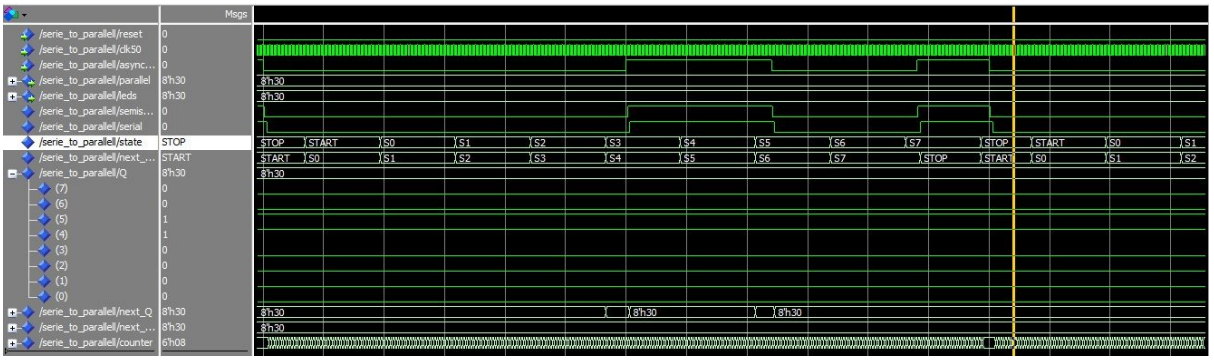

Figure (23): Simulation-plot of serial receiver.

In the simulation plot in figure(23) the bit pattern “0000011001” has been sent to the serial receiver. The simulation shows one conversion cycle and the result can be seen in vector Q where the bit pattern has been stored as a parallel output.

The A/D converter was modified in this laboration to transmit bipolar signals in the range ± 5 volts instead of the range 0-10 volt. This was made possible by putting a resistance between V_{REF} and the input to TL072 and choosing it to the right value. In figure(24) the new resistor R_5 can be seen.

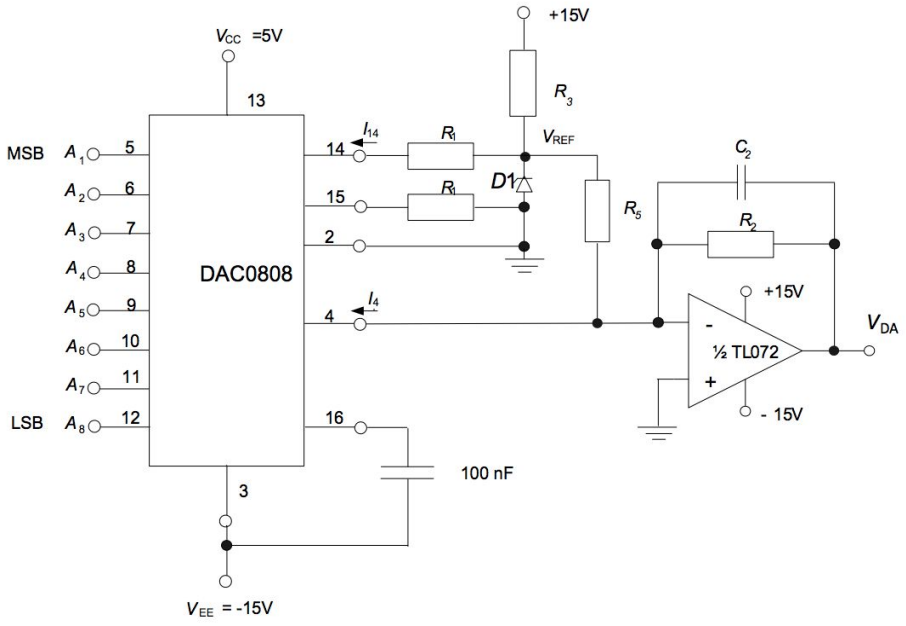

Figure (24): A/D converter modifier with resistance R_5.

The resistance R_5 was calculated to the right value by deciding which current that should go through it. We knew since earlier that the current I_4 has it maximal value 2 mA and that represents digital values in the range from 0-10. If we divide this current by 2 we will get a maximal digital value of 5 volts. We also know that the potential in the point V_{REF} = 5,6 volts. If we then divide the voltage by the current we get the resistance value 5600 Ω that represent resistance R_5.

$$I_5 = \frac{I_4 Max}{2} = \frac{0.002}{2} = 0.001\ A \qquad (6)$$

$$\frac{U_{R5}}{I_5} = 5600\ \Omega \qquad (7)$$

In the laboration the output from the A/D converter was measured at different voltages between -5 & 5 and compared to the values that was expected. The result can be seen in table(3). As can be seen the measured output did not fully quite agree with the expected values. The reason for this is that the components in the circuit is not ideal.

Table (3): Calculated and measured values of A/D converter.

Analog input DC (V)	**Calculated digital output**	**Measured digital output**
-5	00000000	00000000
-2	01001101	01000101
-1	01100110	01011111
0	10000000	01111010
1	10011010	10010100
2	10110011	10101111
5	11111111	11111111

The serial receiver was tested by sending a chosen ASCII-sign from a computer with a RS232-compatible communication-software and see what bit-pattern that was received. The analog output was also measured. The result could be seen in table(4) below.

Table (4): Received and measured values from computer.

ASCII-sign	**Received bit-pattern**	**Measured analog output (V)**
P	01010000	-1.6
?	00111111	-2.254
å	11100101	-2.253
!	00100001	-3.388
=	00111101	-2.329
A	01000001	-2.169

Audio amplifier

In this laboration the two amplifiers along with the mic and the speaker were designed. First the mic and the microphone amplifier were designed. The schematic for the mic and the amplifier can be seen in figure (25) below:

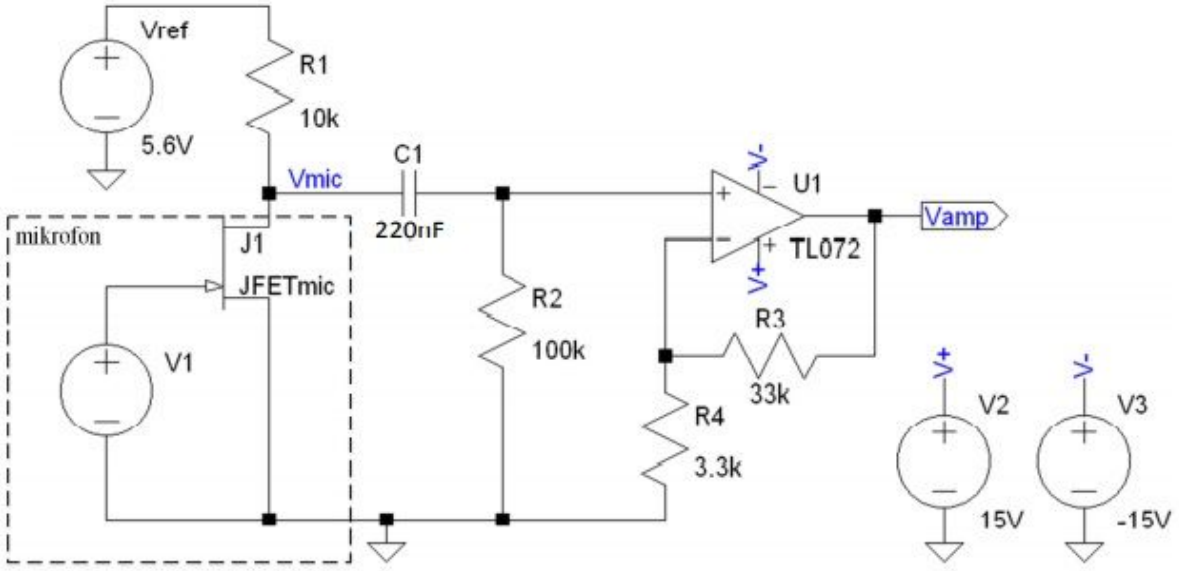

Figure (25): Circuit diagram for the mic and its amplifier.

To evaluate the microphones in-impedance, allt the power supplies and condensators were shortened. The resistors R_1 and R_2 can then be seen as a parallel coupling between each other which will result into a resistance of 9,1 kΩ. This is also the in-impedance of the microphone. In the datasheet for the microphone, its sensitivity is defined for a in-impedance of 2 kΩ. The fact that the in-impedance in this coupling is 9,1 kΩ will not affect the microphone's sensitivity.

To determine the maximum sound pressure, the fact that the sound pressure with an in-impedance of 2 kΩ was equal to -39 dB, was used from the datasheet. With and in-impedance of 9,1 kΩ, a factor of around 4,5 will be added to the reference pressure. The factor 4,5 can also be expressed as 20*log(4,5) = 13 dB. The maximum sound pressure will thus be -39 + 13 = -26 dB.

The circuit was then simulated in LTspice to establish its resting voltage (DC) on the microphone input, the amplifiers lower cut off frequency and the amplification from V_{mic} to V_{amp}. The resting voltage was established to 3,084 V, the lower cut off frequency to around 7 Hz and the amplification to ca 21 dB.

The next part was to design the power amplifier. The circuit diagram for the amplifier can be seen in figure (26) below:

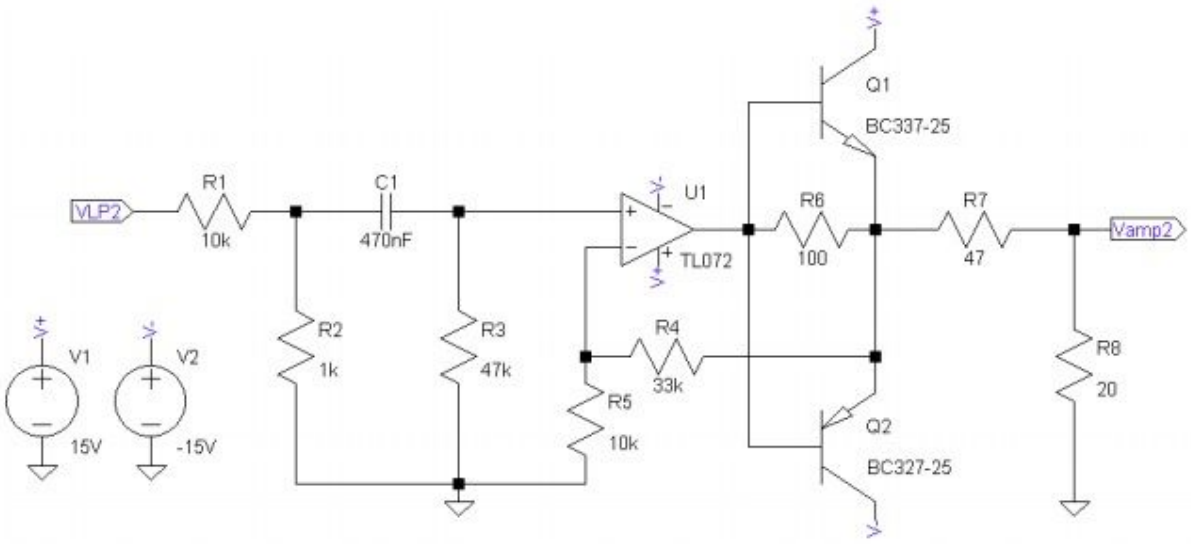

Figure (26): Schematic for the power amplifier.

To theoretically calculate the circuits voltage amplification from V_{LP2} to V_{amp2} the capacitor C_1 and the resistance R_6 were shortened. R_2 and R_3 can then be seen as a parallel coupling and the operational amplifier can be seen as a non-inverted operational amplification circuit with transfer function $(R_5 + R_4)/R_5$. The voltage amplification can then be evaluated through voltage dividing and then multiplying the separate factors with each other according to formula (8) below:

$$\frac{\frac{R_2*R_3}{R_2+R_3}}{R_1+\frac{R_2*R_3}{R_2+R_3}} * \frac{R_5+R_4}{R_5} * \frac{R_8}{R_8+R_7} \qquad (8)$$

By inserting the values of the resistors, the amplification could be evaluated to 0,115 = -19 dB. The next step was to evaluate the maximum sound pressure the power amplifier supplies on a speaker with an in-impedance of 20 Ω when the input signal has an amplitude of 5 V. This could be done by evaluating the power that generates in the speaker. To do this the following equation was used:

$$P = \frac{U_{eff}^2}{R} \qquad (9)$$

The power was evaluated to P = 0,0075625 W. This value can be written i dBm by the following equation:

$$dBm = 10 * log(\frac{0,0075625}{10^{-3}}) \approx 9$$

Speakers can typically have a sensitivity of 100 dB SPL/mW which means that the maximum sound pressure with an input of 5 V will be 100 + 9 = 109 dB SPL.

The circuit was then simulated in LTspice with an input signal with variable frequencies to establish the voltage amplification and the cut of frequencies. The bode-plot can be seen in figure (27) below:

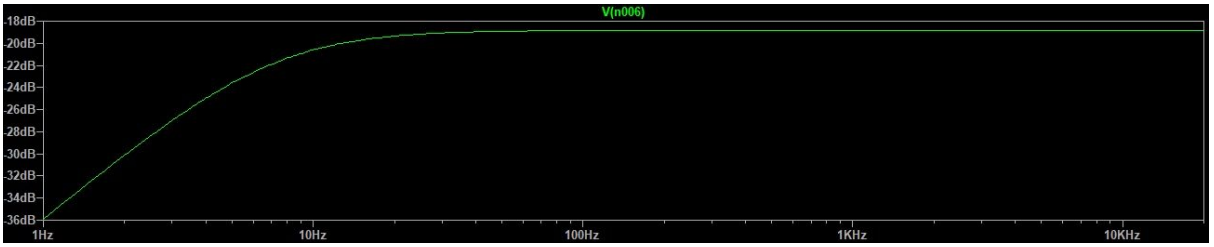

Figure (27): Bode-plot for the power amplifier.

The voltage amplification was measured to ca 0,116 = -18,7 dB which corresponds to the theoretically calculated value. The lower cut of frequency can be seen in figure () above and was established to 7 Hz. The upper cut off frequency was established to around 100 kHz.

To evaluate the power amplifiers power amplification the circuit was analysed with an input of 5 V with a frequency of 1 kHz. By multiplying the outputs and inputs currents and voltages to determine the power on the input and the output, the amplification for the system could be decided by dividing the output power with the input power. The result from the simulation can be seen in figure (28) below:

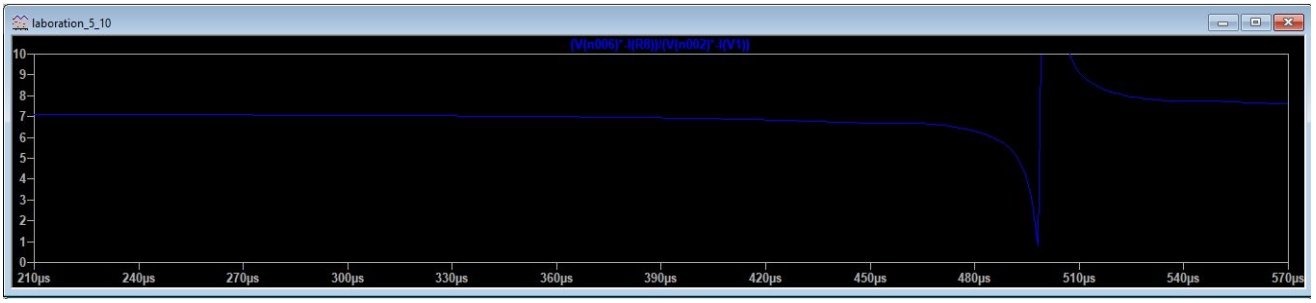

Figure (28): Power amplification for the power amplifier.

As can be seen from the figure the power amplification was established to ca 7 times = 16,9 dB.

The next step was to realize the two amplifiers in hardware. Both amplifiers were designed on the same breadboard as in previous laborations. The microphones 5,6 supply voltage was taken from the point V_{ref} in figure (2). The DC-voltage on the input of the microphone was measured to 2,97 V. The resting current through the microphone could then be evaluated to 0,297 mA by applying Ohm's law. The microphone was tested by placing it in different locations and measuring the output voltage for different sound sources.

When the testing of the microphone was done it was time to test the speaker and the power amplifier. A signal generator was connected to the input of the power amplifier. The generator was set to deliver a sinusoid with an amplitude of 5 V. The frequency was varied from 1 Hz to 20 kHz. The following table of measured values was obtained:

Table (5): Measured input and output voltages for different frequencies.

Frequency [Hz]	**Input [V]**	**Output [V]**	**Gain [dB]**
1	5	0,272	-25,3
5	5	4,04	-1,85
6	5	5,12	-0,50
7	5	5,40	0,21
8	5	5,84	0,67
10	5	6,64	1,35
20	5	6,96	2,46
50	5	6,96	2,87
100	5	6,96	2,87
500	5	6,96	2,87
1k	5	6,96	2,87
2k	5	6,96	2,87
5k	5	6,96	2,87
10k	5	7	2,92
15k	5	7	2,92
20k	5	7	2,92

From the table, to lower cut of frequency can be obtained, which is defined as when the output has sunken by -3 dB. This will happen at around a frequency of 6 Hz. The measured values can be seen in the plot in figure (29) below:

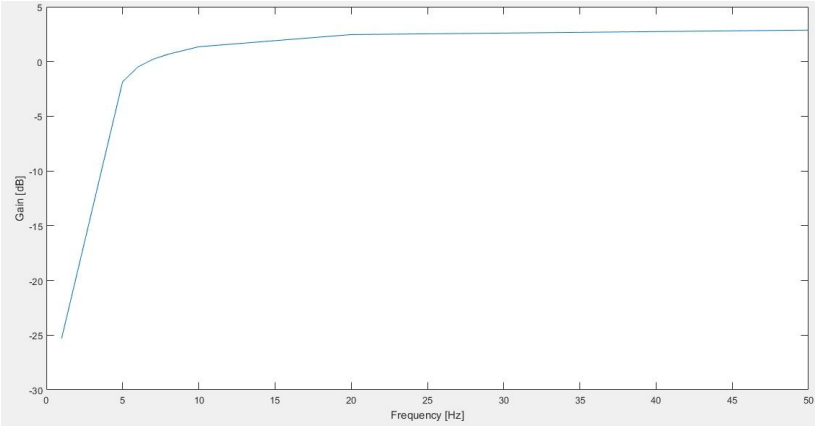

Figure (29): Measured gain as a function of frequency.

As can be seen from the figure as well, the cut of frequency is around 6 Hz. The figure can also be compared to the simulation results that was obtained in figure (27) above. In the simulation, the frequency-axis was in a logarithmic scale which makes it easier to see the characteristics over a wider range of frequencies. In figure (29) the frequency-axis was in a linear scale with a frequency of 1 Hz up to 50 Hz. In both figures, the cut of frequency can be evaluated to 6 Hz.

When all the measurements were done, it was time to test the speaker and the amplifier with a real soundsource. The sound got a little bit distorted but was surprisingly good.

LP-filter

In laboration 6 a lowpass butterworth-filter of the fourth grade was created. The lowpass filter is used in the transmitter and the receiver that has been created in earlier laborations to filter out higher frequencies in form of unwanted noise. The filter should have a cutoff frequency at 12kHz and that is because of the sampling rate of 24kHz. The filter was created with two cascaded sallen-key links of the second grade. The resistors in the filter was chosen between 1kΩ and 10kΩ and the capacitors between 1nF and 33nF.

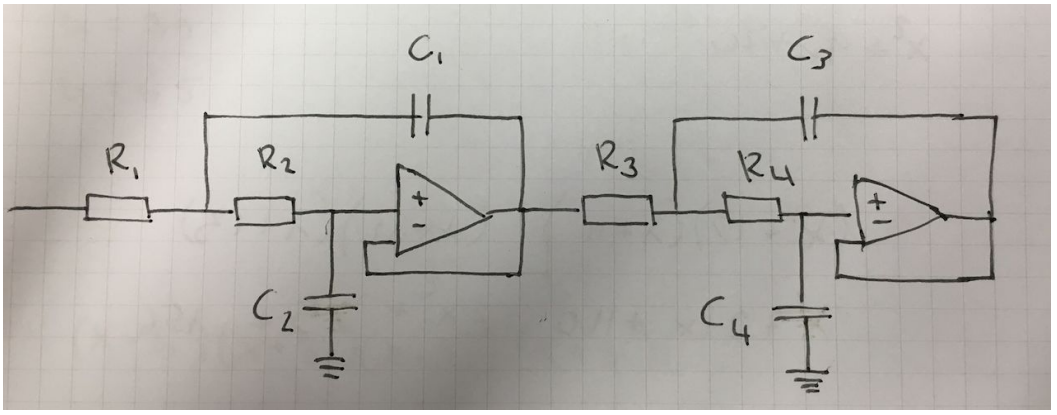

Figure (30): Cascaded sallen-key links.

In figure(30) two cascaded sallen-key links can be seen. To choose the right components the polynom of a fourth grade butterworthfilter needed to be known. The polynom was found i a table of butterworth polynomials and can be seen in equation (10).

$P(a) = (1+0{,}765a+a^2)(1+1{,}8485a + a^2)$ (10)

The polynomial was transformed into a transfer function and can be seen in equation(11) below. $w_ö$ represents the cut of frequency 12 kHz in radians $2\pi*12000 = 24000\pi$. The first second grade polynomial represents one sallen-key link and the second polynomial the second sallen-key link.

$$H(s) = \left(\frac{1}{1+\frac{0{,}765s}{w_ö}+\frac{s^2}{w_ö^2}}\right) * \left(\frac{1}{1+\frac{01{,}848s}{w_ö}+\frac{s^2}{w_ö^2}}\right) \quad (11)$$

The transfer function of a sallen-key link can be seen in figure(31) below. If the transfer function of the filter is compared to the sallen-key transfer function we see that $\frac{0,765s}{w_ö}$ should represent $(R_1 + R_2)C_2$ and so on.

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

Figure(31): Transfer function of a sallen-key link

If we put in the cut of frequency in the transfer function of the filter we can see what values the components in the salley-key links should represent.

$\frac{0,765s}{w_ö} = \frac{0,765}{24000\pi} = 10,1461\ \mu S = (R_1 + R_2)C_2$ (12)

C_2 was chosen to 10nF and the resistors R_1 and R_2 to a parallel connection between two 1000Ω resistors which is 500Ω. $(R1 + R2)C2$ = (500+500)*10nF = 10 $\mu S \approx$ 10,1461 μS. The other component was chosen in the same way for the other parts of the transfer function $\frac{s^2}{w_ö^2} = R_1R_2C_1C_2$, $\frac{01,848s}{w_ö} = (R_3 + R_4)C_4$ and $\frac{s^2}{w_ö^2} = R_3R_4C_4C_3$. All the chosen components can be seen i table(6) below.

Table (6): Table of the chosen components of the butterworthfiter.

R_1	500Ω
R_2	500Ω
R_3	1200Ω
R_4	1200Ω
C_1	10nF
C_2	66nF
C_3	22nF
C_4	10nF

In figure(32) below the complete coupling of the butterworth-filter can be seen with all the right components connected. In figure(33) a simulation of the filter in LT-spice can be seen. The cut off frequency can be seen at 12khz as wished. The low-pass filter was tested in the laboratory and the test result can be seen in table(7).

Table(7): Table of the test results of the low-pass in the laboratory.

Frequency(kHz)	**Input**	**Output**	**Attenuation(dB)**
1	10	10	0

5	10	10.8	0.67
8	10	10.6	0.506
9	10	10.2	0.172
10	10	9.2	-0.72
11	10	7.8	-2.2
12	10	6.56	-3.7
13	10	5.36	-5.4
14	10	4.24	-7.45
15	10	3.32	-9.57
20	10	1.02	-19.82
25	10	0.400	-27.95
30	10	0.184	-34.7

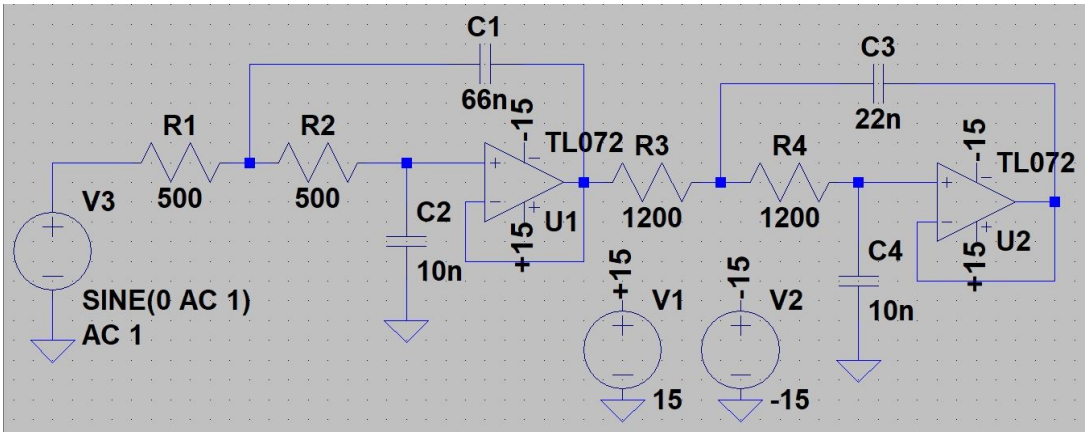

Figure 32): Diagram showing the complete coupling of the butterworth-filter in LT-spice.

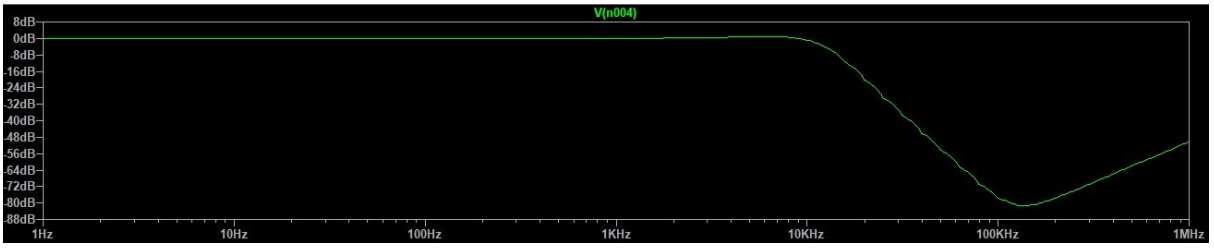

Figure (33): Diagram showing the complete coupling of the butterworth-filter in LT-spice.

Test and verification

In the final part of this series of laborations it was time to test the complete system. This was done by collaborating with another group. The other group manipulated their system to act as a transmitter while we manipulated ours to act as a receiver. In the first part of the test, the microphone and the speaker amplifier was not connected. A variable DC-voltage was transmitted. The amplitude was varied between some random values to ensure that the transmitter and receiver was synced. An oscilloscope was used to detect the signal at the receiver along with the diodes on the DE1-cards.

When the transmitter and the receiver was synced, a sinusoid was sent through the system with a frequency of 1 kHz. The amplitude was varied and the lower and upper boundaries for which voltages that could be detected was measured. The lower boundary was measured to 100 mV_{pp} and the upper to 9,1 V_{pp}. This values was used to determine the communications dynamic by dividing the maximum amplitude with the minimum. The result from this division was 39 dB.

One oscilloscope was connected at the transmitting side to measure the potential in the points V_{LP} and V_{SH} and another was connected at the receiver to measure the points V_{DA2} and V_{LP2}. The points can be seen in figure (1). The measurements can be seen in figure (34) and (35) below:

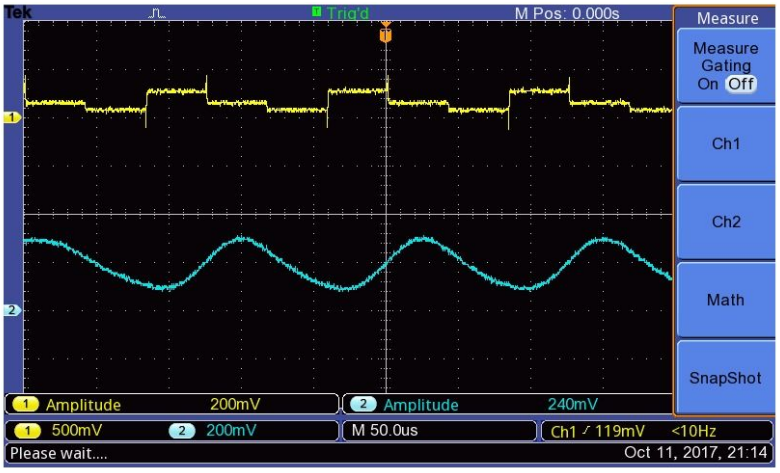

Figure (34): Yellow curve: V_{SH}, blue curve: V_{LP}.

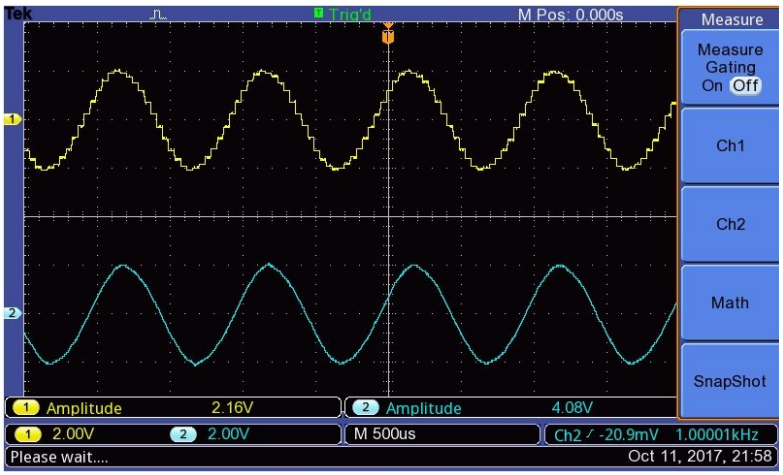

Figure (35): Yellow curve: V_{DA2}, blue curve: V_{LP2}.

In the testing, the input was changed between capturing of these images which explains the different voltages detected. Nonetheless, similarity between the two blue curves can be identified.

The inputs amplitude was set to 9 V and the frequency was varied from 3 Hz to the point when the output had sunken with 3 dB. The bandwidth of the communication was established from 3 Hz to 4,5 kHz.

In the next part of the testing, the two amplifiers were connected with a soundsource on the transmitting side and a speaker on the receiving side. The quality of the sound was okay, but with some unwanted overtones. By grounding some of the of the inputs A_1 to A_8 of the D/A converter, which can be seen in figure (2), it could be determined which the least amount of bits that was needed to be able to apprehend the sound. This was done by first grounding the least significant bit and then the second least and so on. With only two bits left (A_1 and A_2) the sound was still detectable but with many disturbances and interruptions.

Discussion

This project has been both challenging and fun. We can both say that we learned new things at the same time as old forgotten knowledge were refreshed. The biggest lesson we learned from this project was that it's very important to be meticulous when writing code and connecting components on the coupling plate. Small careless mistakes and inaccuracies may result in unnecessary time consuming troubleshooting. For instance using color coding when coupling wires on the coupling plate is a thing that saves much time when troubleshooting is needed. Also leaving comments when writing code in VHDL makes it a lot easier to look back and understand what you have done.

After completing the system we can say that i worked as it was supposed to. Maybe the system was not that perfect that it could be sold as a finished product. Although the system both could send and receive sound, the quality of the received sound was not that good. We played music through the transmitter and listened to it on the receiver. Some distortion could be heard and was not that pleasant to listen to.

The system had a resolution of 8-bits and we tested to reduce the resolution until we could not hear the music clearly any more. Surprisingly the music could still be heard when the resolution was only two bits.

One thing that is worth to mention is that the band width of the transmitter was measured to about 4.5 kHz when is was supposed to be 12 kHz. The low-pass filter that is used in both transmitter and the receiver is constructed to have a bandwidth of 12 kHz so that is a bit strange. Several other groups experienced the same phenomenon and we have a theory that i might have something to do with the S/H circuit

Reflection

The work with this project have worked well we must say. Although it worked well it has been very intensive with one laboration each week that had to be prepared. We had good cooperation in the group, both in the laboratory and outside the laboratory preparing labs and writing the lab report. We have met once a week before the laboration and collaborated to solve the preparation tasks that had to be done. The preparation tasks have been both easy and hard, sometimes the description in the tasks have been fuzzy which has created unnecessary time consuming confusion. But the majority of the tasks have been good and we have had good guidance how to solve most of them at lectures before laboratory.

In the laboratory almost everything have been working smooth. We have had some minor issues such as our code have been working just fine in the simulation but later in the laboratory it did not work at all. We encountered this type of problem a few times and it was very time consuming to solve it. We also had some minos issues with the DE1-card, some times when we downloaded the code to the card it did not work as it should. With repeated attempts downloading the code again it worked inexplicably again. The guidance from the lab-assistants have been good. When we had questions they always had a answer or could help us troubleshooting.

Appendix

Appendix I

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

entity klocka_och_triggsignal is
port (
clk50: in std_logic;
reset: in std_logic;
ts, tb: out std_logic);
end entity;
```

```
architecture arch of klocka_och_triggsignal is
signal counter: std_logic_vector(15 downto 0);
signal counter2: std_logic_vector(15 downto 0);
begin

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

		elsif rising_edge(clk50) then
		counter <= counter + 1;

			if counter < X"A2C" then
				ts <= '1';
			else
				ts <= '0';
			end if;

			if counter = X"CB73" then
				counter <= (others => '0');
			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;

			if counter2 < X"1" then
				tb <= '1';
			else
				tb <= '0';
			end if;

			if counter2 = X"1457" then
				counter2 <= (others => '0');
			end if;
		end if;
end process;
end architecture;
```

Appendix II

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

entity counter is
port (
clk50, reset, SW9: in std_logic;
tb: out std_logic;
leds_de1: out std_logic_vector(7 downto 0);
bit_counter: out std_logic_vector(7 downto 0));
end entity;

architecture arch of counter is
signal counter: std_logic_vector(19 downto 0);
signal tf_bit_counter: std_logic_vector(7 downto 0);
signal tf_tb: std_logic;
begin

process(clk50,reset)
begin

	if reset = '0' then
	counter <= (others => '0');

	elsif rising_edge(clk50) then
	counter <= counter + 1;

		if counter < X"1" then
			tb <= '1';
			tf_tb <= '1';
		else
			tb <= '0';
			tf_tb <= '0';
		end if;

		if counter = X"1457" then
			counter <= (others => '0');
		end if;
	end if;
end process;

process(clk50,reset)
begin
```

```
if reset = '0' then
tf_bit_counter <= (others => '0');

elsif rising_edge(clk50) then

	if tf_tb = '1' then

		if SW9 = '1' then
		tf_bit_counter <= tf_bit_counter + 1;

		else
		tf_bit_counter <= tf_bit_counter - 1;
		end if;
	end if;
end if;

end process;

bit_counter <= tf_bit_counter;
leds_de1 <= tf_bit_counter;
end architecture;
```

Appendix III

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

entity serie_to_parallell is port(
reset, clk50: in std_logic;
async_serial: in std_logic;
parallel, leds: out std_logic_vector(7 downto 0));
end entity;

architecture state_machine of serie_to_parallell is
type StateType is (U,STOP,START,S0,S1,S2,S3,S4,S5,S6,S7);
signal semisync_serial, serial: std_logic;
signal state, next_state: StateType;
signal Q, next_Q, next_parallel: std_logic_vector(7 downto 0);
signal counter: std_logic_vector(15 downto 0);
begin

process(clk50)
begin
```

```
		if rising_edge(clk50) then
			semisync_serial <= async_serial;
			serial <= semisync_serial;
		end if;
end process;

process(clk50,reset)
begin

		if reset = '0' then 					-- 0 i labbet...
			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
				state <= next_state;
				Q <= next_Q;
				parallel <= next_parallel;

			elsif counter = 5207 then
				counter <= (others => '0');

			end if;
		end if;
end process;

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 =>
                if serial = '1' then
                        next_state <= STOP;
                else
                        next_state <= S7;
                end if;
                next_parallel <= Q;
                leds <= Q;

                when others =>
                next_state <= STOP;

                end case;
end process;

end architecture;
```

Appendix IV

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

entity frequencyparalell is
port (
		mhz50: in std_logic;
		reset: in std_logic;
		SW0,SW1,SW2,SW3,SW4,SW5,SW6,SW7: in std_logic;
		Ts,Tb,Dtx: out std_logic);
end entity;

architecture arch of frequencyparalell is
signal counter: std_logic_vector(15 downto 0); -- 16 bitar 50mhz/960hz = X"CB73" 4 bitar per
hex
signal counter2: std_logic_vector(15 downto 0);
signal triggTs: std_logic;
signal triggTb: std_logic;
signal bitmonster : std_logic_vector(7 downto 0);
begin
process(mhz50,reset)
begin
		if reset='0' then -- Om reset är 1 alla bitar sätts till 0
			counter <=(others => '0');
		elsif rising_edge(mhz50) then -- vid positiv flank räknas counter upp
			counter <= counter + 1;
		if counter < 2603 then -- Under 5% av periodtiden är pulsen hög
			Ts <= '1';
			triggTs <= '1';
		else
			Ts <= '0'; -- Efter 5% av periodtiden är pulsen låg
			triggTs <= '0';
		end if;
if counter = 52079 then -- När clockan räknat upp till hela periodtiden nollställs counter
		counter <= (others => '0' );
		end if;
end if;
end process;

process(mhz50,reset)
begin
		if reset='0' then -- Om reset är 1 alla bitar sätts till 0
			counter2 <=(others => '0');
```

```
                elsif rising_edge(mhz50) then -- vid positiv flank räknas counter upp
                        counter2 <= counter2 + 1;
                if counter2 < X"1" then -- Under 5% av periodtiden är pulsen hög
                        Tb <= '1';
                        triggTb <= '1';
                else
                        Tb <= '0'; -- Efter 5% av periodtiden är pulsen låg
                        triggTb <= '0';
                end if;
if counter2 = 5207 then -- När clockan räknat upp till hela periodtiden nollställs counter
                counter2 <= (others => '0' );
                end if;
end if;
end process;

process(mhz50,triggTs,triggTb)
                begin
                if rising_edge(mhz50) then


                        if triggTs = '1' then
                        bitmonster(0) <= SW0;
                        bitmonster(1) <= SW1;
                        bitmonster(2) <= SW2;
                        bitmonster(3) <= SW3;
                        bitmonster(4) <= SW4;
                        bitmonster(5) <= SW5;
                        bitmonster(6) <= SW6;
                        bitmonster(7) <= SW7;
                end if;
                if triggTb = '1' then
                bitmonster(6 downto 0) <= bitmonster(7 downto 1);
                end if;


end if;

end process;
Dtx <= bitmonster(0);
end architecture;
```

Appendix V

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

```
entity state_machine is
port (
clk50, reset, D: in std_logic;
tb, ts, Dtx: out std_logic;
LEDS: out std_logic_vector(7 downto 0);
LEDS2: out std_logic_vector(7 downto 0));
end entity;

architecture arch of state_machine is
signal counter: std_logic_vector(15 downto 0);
signal counter2: std_logic_vector(15 downto 0);
signal Q: std_logic_vector(7 downto 0);
signal bit_monster: std_logic_vector(9 downto 0);
signal tf_tb, tf_ts: std_logic;
type StateType is (IDLE,SAR7,SAR6,SAR5,SAR4,SAR3,SAR2,SAR1,SAR0,STOP); -- list of
states
signal state: StateType;

begin

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

                elsif rising_edge(clk50) then
                counter <= counter + 1;

                        if counter < X"A2C" then            -- X A2C   simulering: X 5
                                ts <= '1';
                                tf_ts <= '1';
                        else
                                ts <= '0';
                                tf_ts <= '0';
                        end if;

                        if counter = X"CB6F" then           -- X CB6F  simulering: X 64

                                counter <= (others => '0');
                        end if;
                end if;
end process;

process(clk50,reset)                    -- tb
begin
```

```
		if reset = '0' then
		counter2 <= (others => '0');

		elsif rising_edge(clk50) then
		counter2 <= counter2 + 1;

			if counter2 < X"1" then
				tb <= '1';
				tf_tb <= '1';
			else
				tb <= '0';
				tf_tb <= '0';
			end if;

			if counter2 = X"1457" then		-- X 1457   simulering: X A

				counter2 <= (others => '0');
			end if;
		end if;
end process;

process(clk50,reset,tf_ts,tf_tb)			-- SAR
begin

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

elsif rising_edge(clk50) then

			case state is

			when IDLE =>
				if tf_ts = '1' then
					Q <= "10000000"; -- half of the maximum value
					state <= SAR7;
				end if;

			when SAR7 =>
			if tf_tb = '1' then
			Q(7) <= D;
			LEDS2(7) <= D;
			Q(6) <= '1';
			state <= SAR6;
			end if;
```

```
        when SAR6 =>
        if tf_tb = '1' then
        Q(6) <= D;
        LEDS2(6) <= D;
        Q(5) <= '1';
        state <= SAR5;
        end if;

        when SAR5 =>
        if tf_tb = '1' then
        Q(5) <= D;
        LEDS2(5) <= D;
        Q(4) <= '1';
        state <= SAR4;
        end if;

        when SAR4 =>
        if tf_tb = '1' then
        Q(4) <= D;
        LEDS2(4) <= D;
        Q(3) <= '1';
        state <= SAR3;
        end if;

        when SAR3 =>
        if tf_tb = '1' then
        Q(3) <= D;
LEDS2(3) <= D;
        Q(2) <= '1';
        state <= SAR2;
        end if;

        when SAR2 =>
        if tf_tb = '1' then
        Q(2) <= D;
        LEDS2(2) <= D;
        Q(1) <= '1';
        state <= SAR1;
        end if;

        when SAR1 =>
        if tf_tb = '1' then
        Q(1) <= D;
        LEDS2(1) <= D;
        Q(0) <= '1';
        state <= SAR0;
```

```
                        end if;

                        when SAR0 =>
                        if tf_tb = '1' then
                        Q(0) <= D;
                        LEDS2(0) <= D;
                        state <= STOP;
                        end if;

                        when STOP =>

                        state <= IDLE;
                        when others =>

                        end case;

end if;

end process;

LEDS(7 downto 0) <= Q(7 downto 0);

process(clk50, tf_ts, tf_tb)                    -- Parallell till serie omvandling
begin
if rising_edge(clk50) then
                if tf_tb = '1' then
                        if tf_ts = '1' then
                                bit_monster(0) <= '0';
                                bit_monster(1) <= Q(0);
                                bit_monster(2) <= Q(1);
                                bit_monster(3) <= Q(2);
                                bit_monster(4) <= Q(3);
                                bit_monster(5) <= Q(4);
                                bit_monster(6) <= Q(5);
                                bit_monster(7) <= Q(6);
                                bit_monster(8) <= Q(7);
                                bit_monster(9) <= '1';
                        else
                                bit_monster(8 downto 0) <= bit_monster(9 downto 1);
                                Dtx <= bit_monster(0);
                        end if;
                end if;
end if;
end process;

end architecture;
```

Appendix VI

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

entity serie_to_parallell is port(
reset, clk50: in std_logic;
async_serial: in std_logic;
parallel, leds: out std_logic_vector(7 downto 0));
end entity;

architecture state_machine of serie_to_parallell is
type StateType is (U,STOP,START,S0,S1,S2,S3,S4,S5,S6,S7);
signal semisync_serial, serial: std_logic;
signal state, next_state: StateType;
signal Q, next_Q, next_parallel: std_logic_vector(7 downto 0);
signal counter: std_logic_vector(15 downto 0);
begin

process(clk50)
begin

		if rising_edge(clk50) then
			semisync_serial <= async_serial;
			serial <= semisync_serial;
		end if;
end process;

process(clk50,reset)
begin

		if reset = '0' then 					-- 0 i labbet...
			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
				state <= next_state;
```

```
                        Q <= next_Q;
                        parallel <= next_parallel;

                elsif counter = 5207 then
                        counter <= (others => '0');

                end if;
        end if;
end process;

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 =>
                if serial = '1' then
                        next_state <= STOP;
                else
                        next_state <= S7;
                end if;
                next_parallel <= Q;
                leds <= Q;

                when others =>
                next_state <= STOP;

                end case;
end process;

end architecture;
```