Electronic Design 344: Ultrasonic distance sensor

Ignatius de Villiers

17502292

Declaration

I, Ignatius de Villiers, the undersigned, hereby declare that this report and the work contained therein is my own original work, except where indicated.

Signature:

Ignatius de Villiers	Date

Index

Appendices

A	Measured Results
B	Technical Specification
C	Circuit Diagrams
D	PCB design
E	Photo of Complete Circuit
F	Extra Work
G	Software

List of figures

List of Tables

1. Summary

This report provides detailed information on an ultrasonic distance sensor that can be used as an aid to park a car in a garage. The ultrasonic distance sensor provides accurate real-time analogue output proportional to the distance to a vehicle. The measurement range of the system is 30 cm (centimetre) to 3 m (metre).

A microprocessor, Arduino, is used to sample the output of the analogue circuit and communicate with a PC (Personal computer) through a serial interface, and displayed on a GUI (Graphical user interface), which also allows the user to send commands to the Arduino.

Results of data analysed, prove that this system can provide the user with accurate real-time distance measurements, which can be used as an aid to park a car in a garage.

The report finds that the ultrasonic distance sensor is a reliable system that can accurately monitor real-time analogue output proportional to the distance to a vehicle, which can be best implemented in garage, where space is limited.

Recommendations discussed include:

- Protective casing for the system.
- Enabling a Bluetooth connection with the system.
- Adding a Power on-and-off button.
- Adding a Buzzer.

The report also investigates the fact that the system may have some limitations

Limitations include:

- Cannot use in wet conditions
- Limited measurement range

2. Introduction

The system that was built, is designed to provide accurate real-time analogue output proportional to the distance to a vehicle. The distance is measured using a 40 kHz (kilo Hertz), ultrasonic transmitter, receiver pair and other stages of analogue circuitry.

A 12 V (Volt) battery is used to power the system. A virtual ground then supplies the various components with their required positive and negative supply voltages. The virtual ground allows battery use.

The measurement range of the system is from 30 cm to more than 3 m and a maximum range resolution of 10 cm. This measurement range is adequate, as the system is most likely to be used in a garage, where space is can be limited.

A microprocessor, Arduino, is used to sample the output of the analogue circuit and communicate with a PC through a serial interface. A GUI then allows the user to view the distance measurements and also allows the user to send various commands to the Arduino.

Adequate testing procedures, unit test, were also developed to ensure software efficiency.

2.1 *Problem Statement*

In a garage where space is usually limited, it is easy to incorrectly estimate distance to a wall. This is a problem as a car or other equipment in a garage can be damaged as a result of a lack of judgement.

The design of the ultrasonic distance sensor has to overcome the following problems:

- The sensor has to work with a single 12 V battery.
- The sensor has to have a measurement range of 30 cm to 3 m and a maximum range resolution of 10 cm.
- The transmitted pulses should be 40 kHz sinusoidal pulses, to ensure that the pulses are transmitted.
- The received pulses will be very small, thus it needs a lot of amplification.
- The time it takes a pulse to detect an object, needs to be converted into a DC (Direct current) Voltage level.
- The DC Voltage needs to be sent to an Arduino, which feeds the data to a PC.
- Developed software has to interpret the Voltage level and perform the necessary calculations to display the distance on a GUI

3. System Description

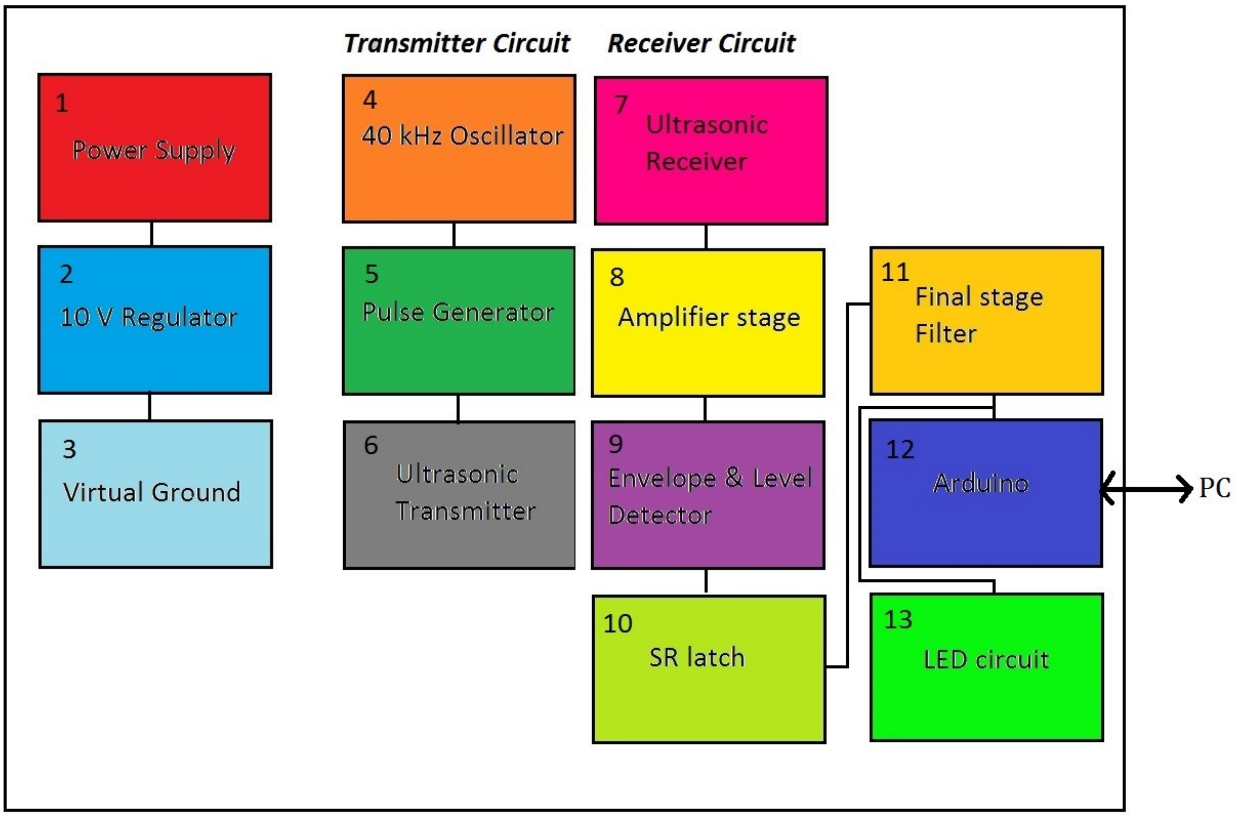


FIGURE 1: SYSTEM DESCRIPTION

See appendix C

1) A 12 V battery, supplies the entire system with power.

2) 10 V regulator used to stabilize the DC supply.

3) Allows Battery supply. Provides positive and negative supply voltages to various circuital components.

4) Generates a 5 V, 40 kHz sinusoidal signal.

5) Generates 40 kHz, sinusoidal pulses to be sent to Ultrasonic transmitter.

6) Transmit pulses, which allow distance measurement.

7) After hitting an object the pulses reflect backwards and are received by the ultrasonic receiver.

8) Amplifies the received signal.

9) Generates a digital pulse.

10) Uses the transmitted and received digital pulses, to generate output that can be filtered to a DC level, which is proportional to the distance measured.

11) Converts the latch output to a DC level, which is proportional to the distance measured.

12) Sends the DC level to the PC, which can then be interpreted and used to determine distance in cm.

13) And array of colour LED's (Light-Emitting diode) that switch on and off at certain distances.

4. Literature Study

Ultrasonic distance sensors are widely used in everyday life, in production lines to see whether an object is approaching or leaving, and how fast it is doing so. Motorists, especially truck drivers also make use of this technology to ensure they do not cause damage to either the vehicle or the good it is carrying, on delivery. It also used in some cases to measure water levels. (Senix)

There are various methods of distance sensing. Frequency modulation is one way of measuring distance, by measuring the frequency of the returned signal and comparing it to the original, the difference can easily be found, thus the frequency shift is used to measure distance. This method is often used in aircraft radar. A more complicated method, known as Pulse-Doppler signal processing, where the space between each transmit pulse is divided into a range of cells where each cell undergoes frequency filtering, the spectrum of the signal is then analyzed to determine the distance. This method is often used with weather radar. The method we will be using is known as the Transmit time method, whereas the distance is measured based on the time of flight of the transmitted pulse, until it is received again.

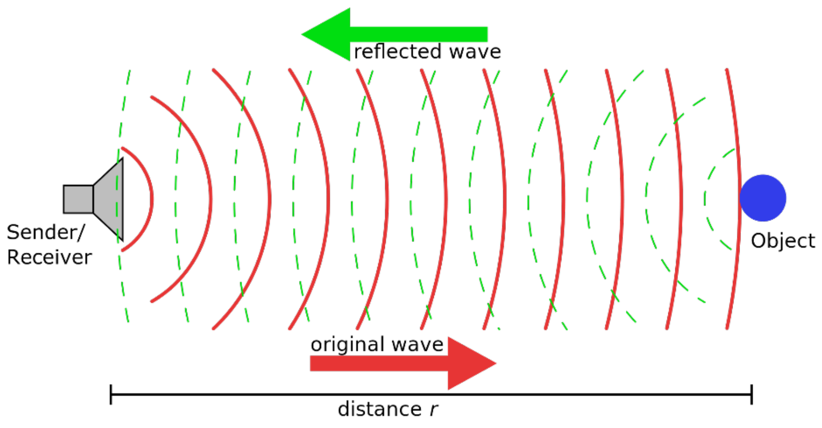


FIGURE 2: OBJECT DETECTION - TIME OF FLIGHT METHOD N.D.

In order to process build a distance sensing circuit an ultrasonic Transmitter and Receiver has to be used. Both the transmitter and the receiver will have separate supporting circuits to help achieve modulation and demodulation of the sent and received signal.

Following one example circuit, the transmitter circuit is designed to achieve maximum output power and a measurement range of up to 6 m. shown in figure 3. The output of this circuit, will produce an output voltage of at least twice the supply voltages. It is noted that if the transducers are driven at a high voltage the will heat up. An example output is shown in figure 4.

The receiver circuit, shown in figure 6, is mainly responsible for the sensitivity of the circuit. For this circuit a high gain, low noise amplifier is used and bandpass filtering to ensure optimal performance. After the gain and filter stages, envelope detection was used to demodulate the signal for further use. Output of the received signal is shown in figure 5. (Kerry D. Wong)

Design discussed in this report will follow similar concepts, but will follow a different design route, with added functionality.

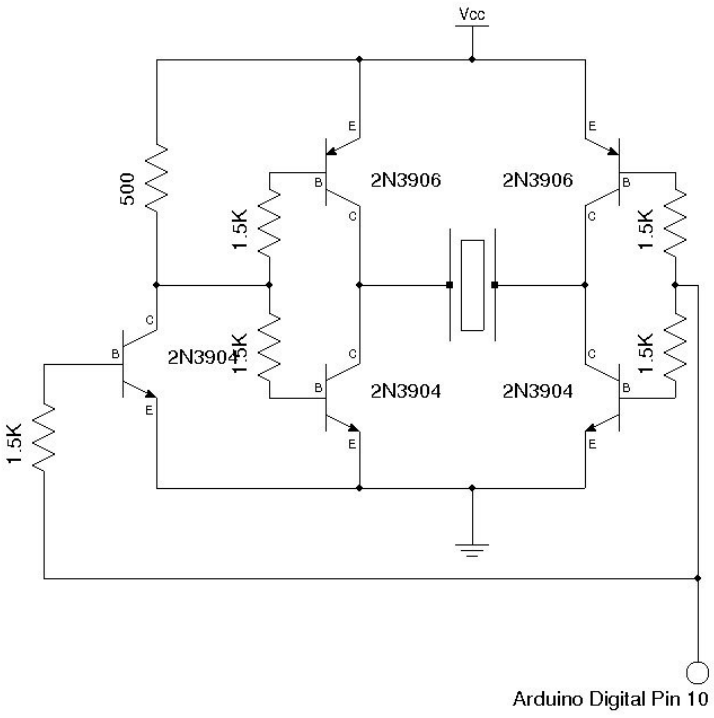


Figure 4: Concept Transmitter circuit

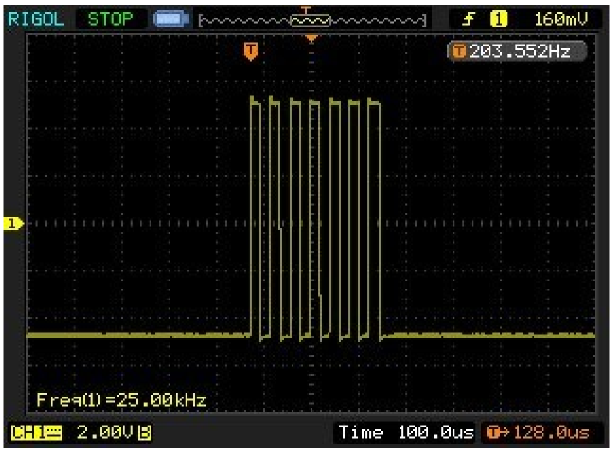


Figure 3: Concept transmitter output

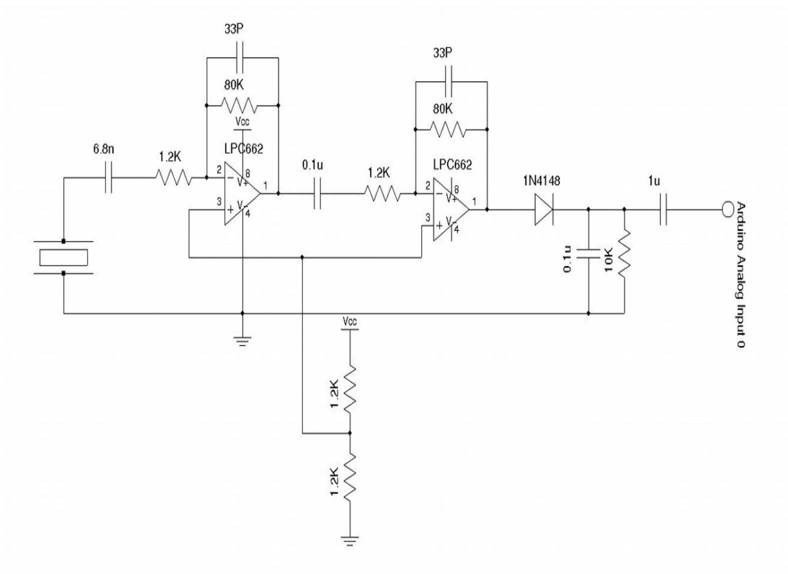


Figure 6: Concept Receiver circuit

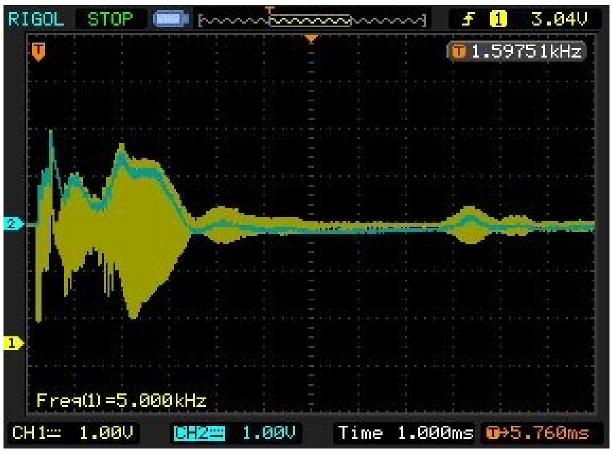


Figure 5: Concept receiver output

5. System Design

5.1 Power Supply

A specification for the design, is that the circuit must be powered by a 12 V battery. The Circuit will be designed to operate between -5 and 5 V. A UA7805 voltage regulator will be used to produce a constant 10 V output.

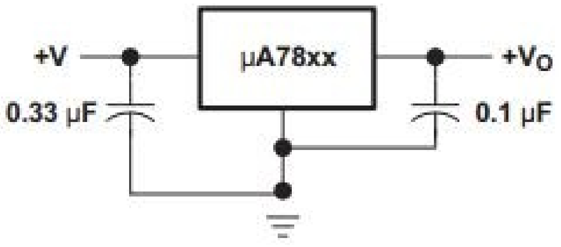


FIGURE 7: FIXED POSITIVE VOLTAGE REGULATOR

Design

From the UA7805 datasheet, figure 3, we find a circuit design, which can be used to produce a constant 10 V to the circuit. The complete circuit diagram for simulated use is shown in figure 16.

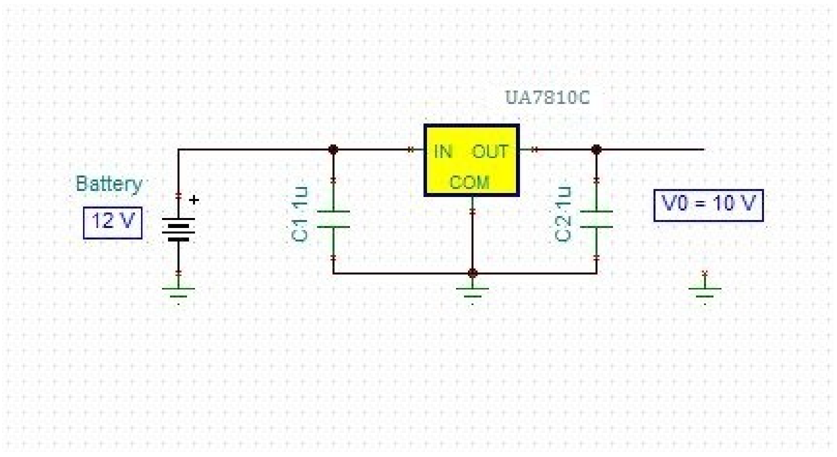


FIGURE 8: VOLTAGE REGULATOR

5.2 Virtual Ground

A 10 V power supply may however cause some problems, as operational amplifiers which are used in some circuits, require a positive and negative input voltage. A good solution is to shift the ground level, to a virtual ground level of 5 V. This will provide the Operational amplifiers with the necessary positive and negative supply voltages.

Design

No design is necessary for this circuit, it is simply required that resistor values are identical and in the range of 100 kΩ (kilo ohm), to ensure that the circuit pulls the minimum amount of current, which will in return increase battery life. A large output capacitor is chosen, to ensure a steady output voltage.

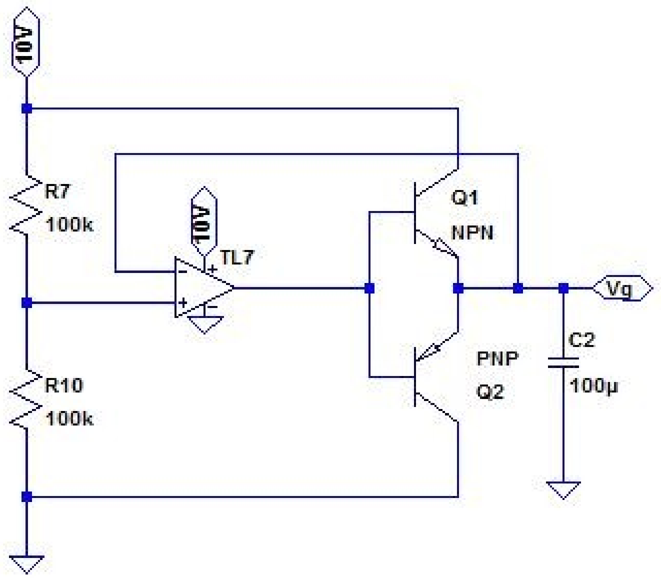


FIGURE 9: VIRTUAL GROUND

Analysis

The SPICE circuit is shown in figure 4, and in figure 5 we see that the output is a constant 5 V DC signal. This will be the virtual ground.

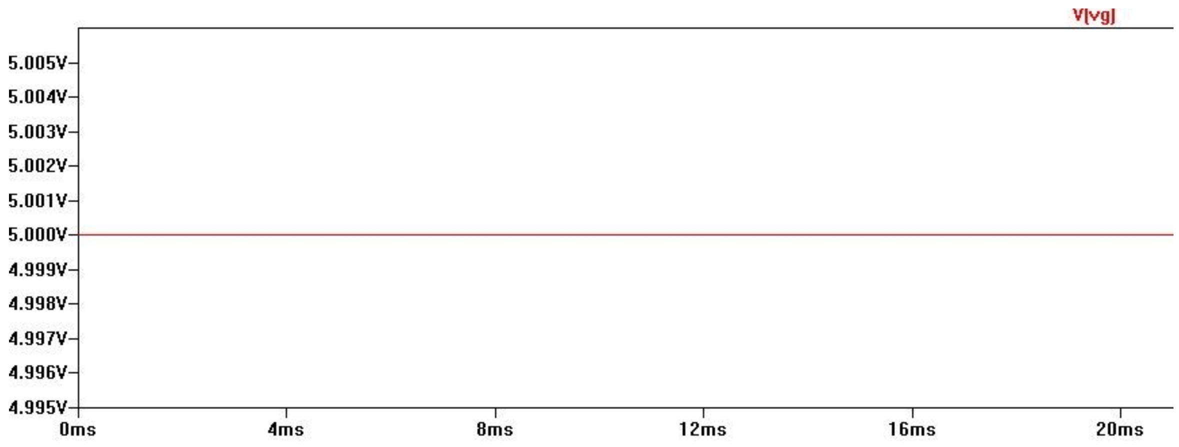


FIGURE 10: VIRTUAL GROUND OUTPUT VOLTAGE

5.3 *Ultrasonic Transmitter model*

Both the ultrasonic transmitter and receiver have a characteristic internal impedance, and as we are connecting both the transmitter and receiver to the circuit, we have to find the models of both, to ensure minimum loading effects.

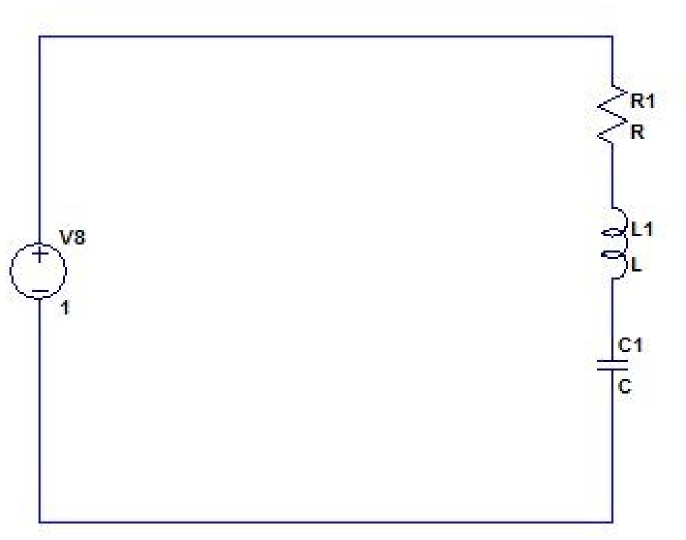


FIGURE 11: ULTRASONIC TRANSMITTER EQUIVALENT MODEL

The Equivalent mode of the Ultrasonic Transmitter is an RLC (Resistive Inductive Capacitive) circuit with all components in series

To find the equivalent Resistance we choose a known resistor,

R1 = 1 kΩ (known)

<u>Resonant Frequency:</u>

f_o = 40 – 40.2 KHz

At the resonant frequency of the circuit the components with complex numbers are equal to zero, thus the measured Voltage over R2 using an oscilloscope was,

$$Vmeas = 500\,mV$$

Leaving the equivalent resistance of the circuit as the only unknown – which is easily solved using voltage division. Choose an input, V1 = 1 V and a 1 kΩ resistor to simplify the calculations:

$$Vmeas = \left(\frac{R2}{R1+R2}\right)(V1)$$

$$0.5=\left(\frac{R2}{1000+R2}\right)(1)$$

$$R2=1000\,\Omega$$

Thus after calculations It was found that the internal

Resistance, R2 = 1KΩ

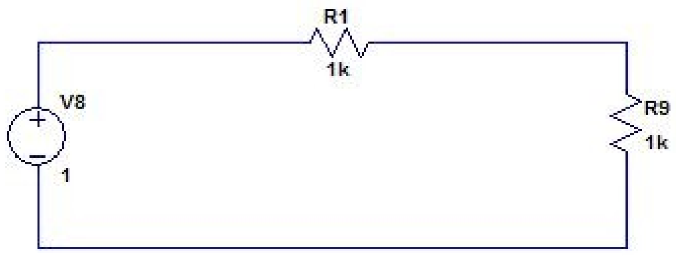


FIGURE 12: ULTRASONIC TRANSMITTER - EQUIVALENT MODEL AT RESONANCE

5.4 *Ultrasonic Receiver model*

The Ultrasonic receiver model was determined by seeing the Ultrasonic receiver as a two port network.

R1 = 1 KΩ

Calculations were as follow:

Input signal: 5 V (p-p) (volt peak-to-peak) sinusoid

Frequency: 40.2 KHz (resonant Frequency)

$$i=\frac{V_2}{R_1}=\frac{160\,mV}{1\,K\Omega}=1.6\times 10^{-4}\,A$$

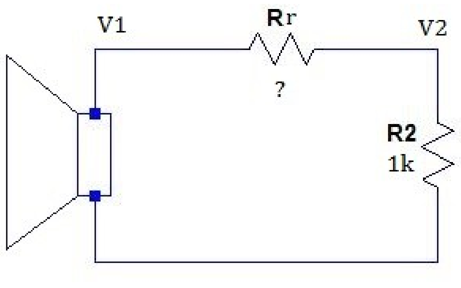
Open loop voltage of the Receiver:

$$i=\frac{V_{actual}-V_2}{R_r}=\frac{1.08\,V-160\,mV}{R_r}$$

$$R_r = 575\Omega$$

Figure 8 shows how the ultrasonic receiver model was obtained.

.



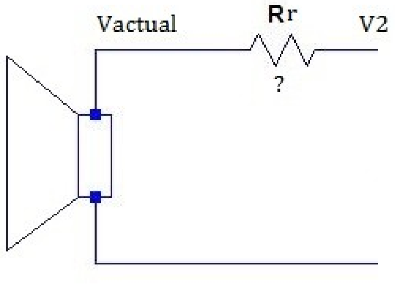


FIGURE 13: ULTRASONIC RECEIVER EQUIVALENT MODEL

5.5 40 kHz Oscillator

The ultrasonic transmitter and receiver pair only transmit and received signals with a frequency of 40 kHz, thus an oscillator circuit had to be designed to ensure that pulses that is sent to transmitter is at the correct frequency.

Design

For the oscillator design, phase shift oscillator, as seen in Figure 8, was chosen, as it is easy to build and simple to debug. All design calculations and circuit diagrams were found in the *Neaman, Microelectronics* text book.

The inverting amplifier introduces a -180 degree phase shift, which means that each RC network must provide 60 degrees of phase shift to produce the 180 degrees required of the frequency-sensitive feedback network in order to produce positive feedback. (Neaman, Donald A., 2010)

***The amplifier Resistor ratio must be*:**

$$\left(\frac{R2}{R}\right)=29$$

Choose:

$R_{1,}$ R_2, R_3 = 1200 Ω

R_4 = 42700 Ω

Decoupling capacitors: 10 nF

***Actual Resistor ratio*:**

$$\left(\frac{R2}{R}\right)=\left(\frac{42700}{1200}\right)=35.58$$

<u>Calculations:</u>

$$wo=2\pi fo=2\pi(40000)$$

$$wo=80200\pi$$

$$wo=\frac{1}{\sqrt{6}RC}$$

Thus,

C1, C2, C3 = 1.353 nF (nano Farad)

For the practical circuit, all chosen component values were based on the theoretical calculations, table 1 shows the theoretically calculated component values versus the chosen values.

Component	Theoretical	Practical
C	1.353 nF	1.22 nF
R	1.2 kΩ	1.2 kΩ
R2	42.7 kΩ	42.7 kΩ

TABLE 1: THEORETICAL AND PRACTICAL VALUES FOR 40 KHZ OSCILLATOR

In LT Spice the circuit was designed, using values, theoretically calculated in the previous steps.

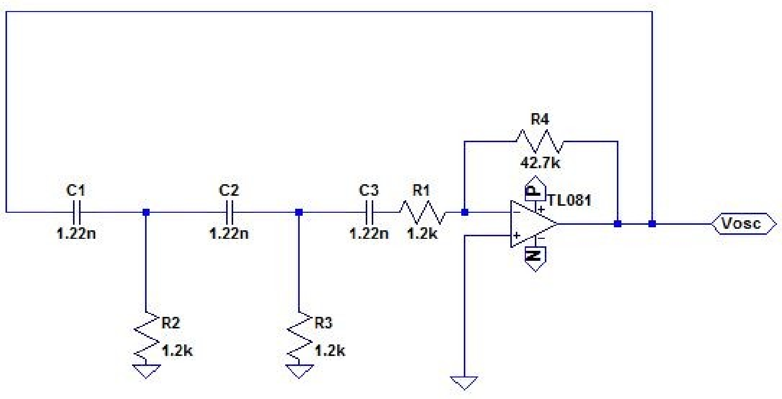


FIGURE 14: OSCILLATOR - SIMULATED CIRCUIT DIAGRAM

Analysis

The SPICE circuit is shown in figure 9, the output of the circuit is obtained during simulation and shown in figure 10. We can clearly see an oscillating output, just as we would expect. Using the FFT (fast Fourier transform) function in SPICE, from figure 11 we see that the output is oscillating at a frequency of about 40 kHz.

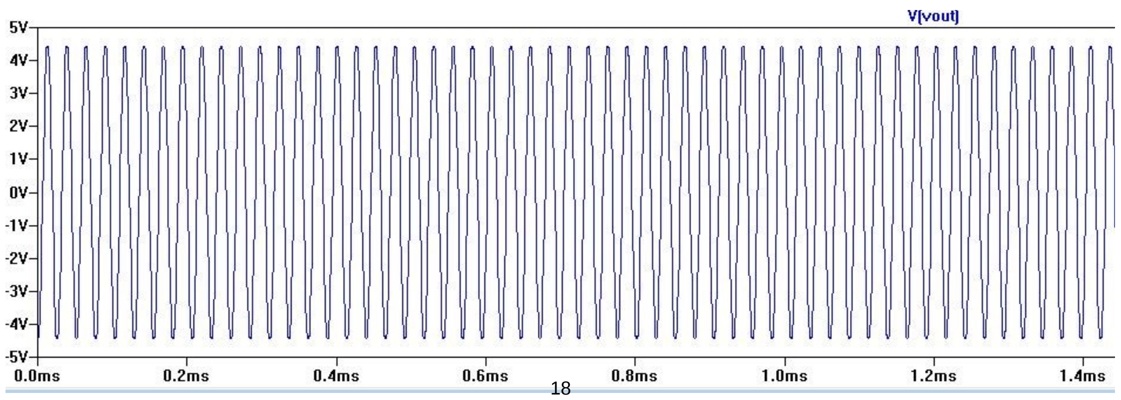


FIGURE 15: OSCILLATOR OUTPUT

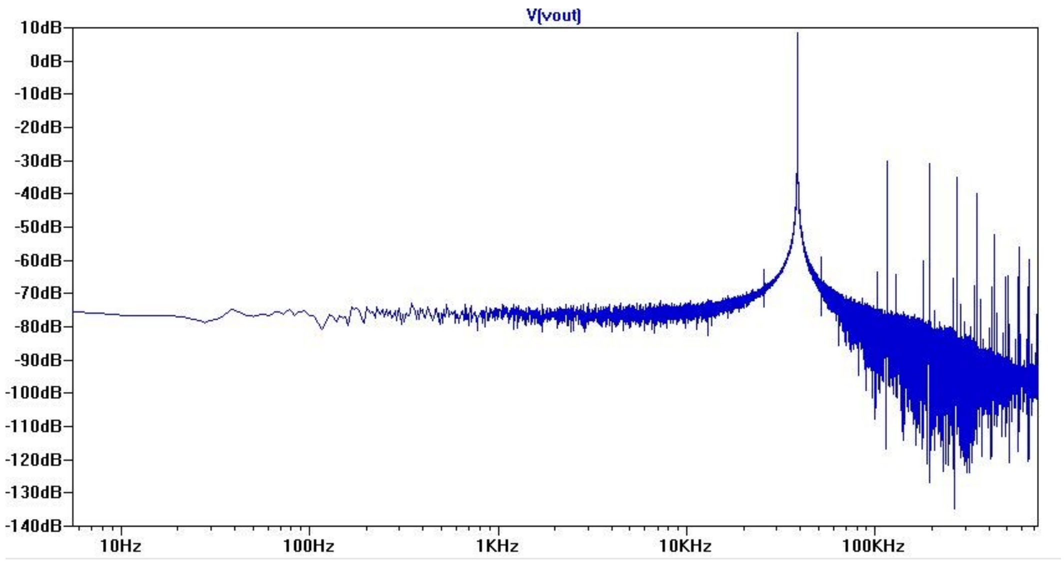
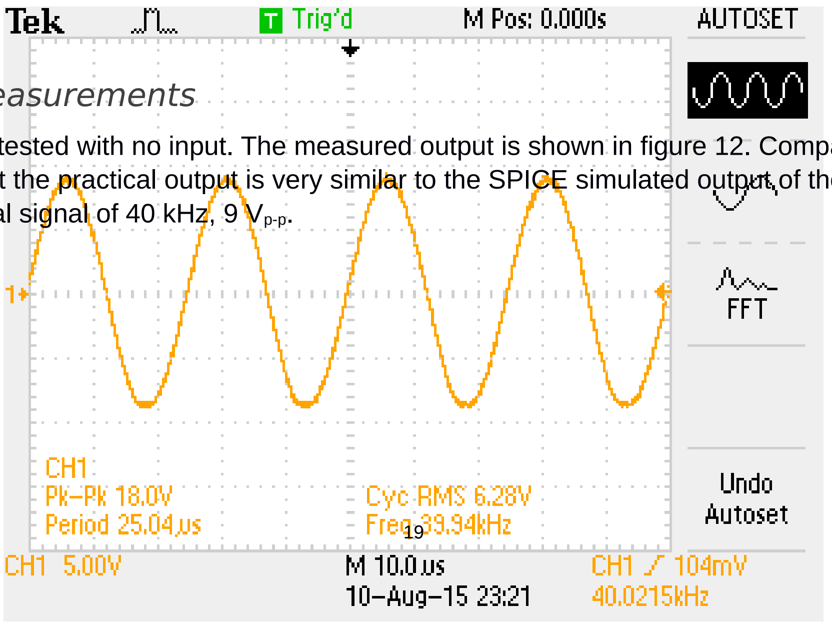


FIGURE 16: OSCILLATOR OUTPUT FFT

Practical Measurements

The circuit was tested with no input. The measured output is shown in figure 12. Comparing figure 10 and 12, it was found that the practical output is very similar to the SPICE simulated output of the oscillator with an output sinusoidal signal of 40 kHz, 9 $V_{p\text{-}p}$.



M Pos: 0.000s

CH1
Pk-Pk 18.0V
Period 25.04µs

Cyc RMS 6.28V
Freq 39.94kHz

CH1 5.00V
M 10.0µs
10-Aug-15 23:21
CH1 ⁄ 104mV
40.0215kHz

FIGURE 17: PRACTICAL OUTPUT OF OSCILLATOR CIRCUIT

It is recommended that a variable resistor is used for R4, which will improve the control of the oscillation frequency.

Conclusion and recommendations

The results for the theory, simulation and the practical circuit all yield a similar oscillation frequency of 40 kHz as shown in table 2. This design is acceptable and can thus be used as the oscillator circuit.

Oscillator Output frequency		
Theory	Simulation	Practical
40 kHz	40 kHz	40.0215 kHz

TABLE 2: THEORETICAL, SIMULATED AND PRACTICAL OSCILLATOR FREQUENCY

It is recommended that a variable resistor is used for R4, to improve control over the frequency at which the circuit oscillates.

5.6 Pulse Generator

For a pulse generator, an NE555 Astable circuit was chosen. When the circuit is connected in Astable mode, it triggers itself and free runs as a multi-vibrator.

Design

The external capacitor charges through R_1 and R_2 and discharges through R_2 only. Thus the duty cycle can be set accurately by adjusting the ratio of these two resistors.

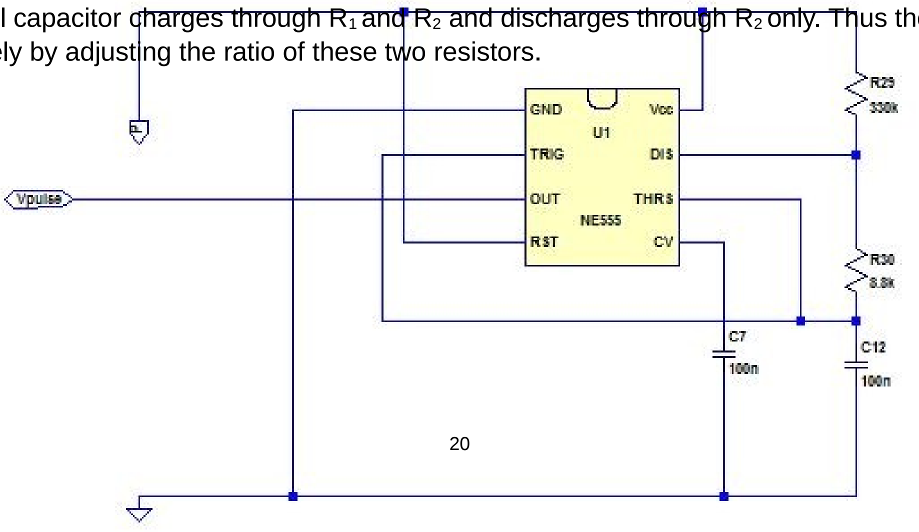


Figure 18: NE555-Astable circuit

From the NE555 data we find the necessary equations to design the pulse generator.

Charge time (output high):

$$t_1 = 0.693\left(R_1 + R_2\right) \times C_1$$

Discharge time (output low):

$$t_2 = 0.693\left(R_2\right) \times C_1$$

Total period:

$$T = t_1 + t_2 = 0.693\left(R_1 + 2R_2\right) \times C_1$$

Thus the frequency is:

$$f = \frac{1}{T} = \frac{1.44}{\left(0.693\left(R_1 + 2R_2\right) \times C_1\right)}$$

Duty cycle:

$$\frac{t_1}{\left(t_1 + t_2\right)} = \frac{\left(R_1 + R_2\right)}{\left(R_1 + 2R_2\right)} = 1 - \left[\frac{R_2}{\left(R_1 + R_2\right)}\right]$$

General notes:

- Increasing C will increase cycle time
- Increasing R_1 will increase time High t_1
- Increasing R_2 will increase time High t_1, increase time low t_2 and decrease the duty cycle
- “www.ohmslawcalculator.com”

Component	Value
R1	330 kΩ
R2	8.8 kΩ
C	100 nF

Table 3: chosen values for NE555-astable circuit

The time delay between each pulse has to be large enough for the pulse to reach the target at distance ‘d’ and travel the same distance ‘d’ back to the sensor before sending another pulse. Thus the equation below has to be satisfied in order for the circuit to function properly.

$$\frac{2d}{V_{sound}} > 2t_2 + t1$$ Assume speed of sound: $V_{sound} = 343\,m/s$

From table 3, we can see that the equation above is satisfied, thus this circuit design can be used to generate pulses for the ultrasonic transmitter, where d is the theoretical maximum measurable distance.

$$d \cong \left(\frac{1}{2}\right)(2t_2 + t1) \times V_{sound} \qquad d \cong 4.2358$$

Theoretical values	
Period	24.089 ms
Duty cycle	97.40 %
Time high t_1	23.479 ms
Time low t_2	609.840 μ s

Table 4: Theoretical pulse properties - NE555-Astable circuit

Ultrasonic signal transmission generally works with pulses, thus it is necessary to create pulses with the 40 kHz oscillating signal in order to be able to transmit the signal through the ultrasonic transmitter. To achieve this, a simple NPN, common emitter transistor circuit had to be designed. In figure 9, the pulses from the NE555-astable circuit, will turn the transistor on and off, and thus create 40 kHz sinusoidal pulses. An additional unity-gain buffer was added to prevent loading from the ultrasonic transmitter’s internal impedance.

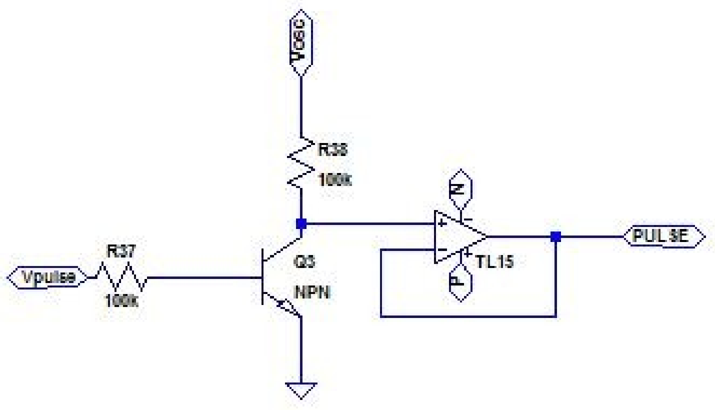


FIGURE 19: PULSE GENERATOR - COMMON EMITTER CIRCUIT

Analysis

The SPICE circuit is shown in figure 13 and 14, the output of the circuit is obtained during simulation and shown in figures 15, 16 and 17. We can clearly see from figure 11, that the signal that is to be transmitted is a 40 kHz, sinusoidal pulse train.

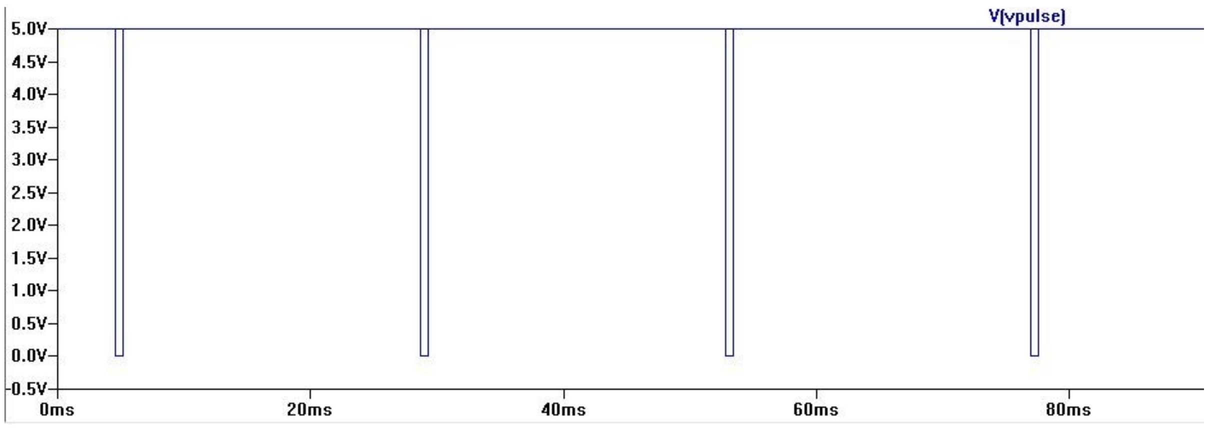


FIGURE 20: NE555-ASTABLE CIRCUIT OUTPUT

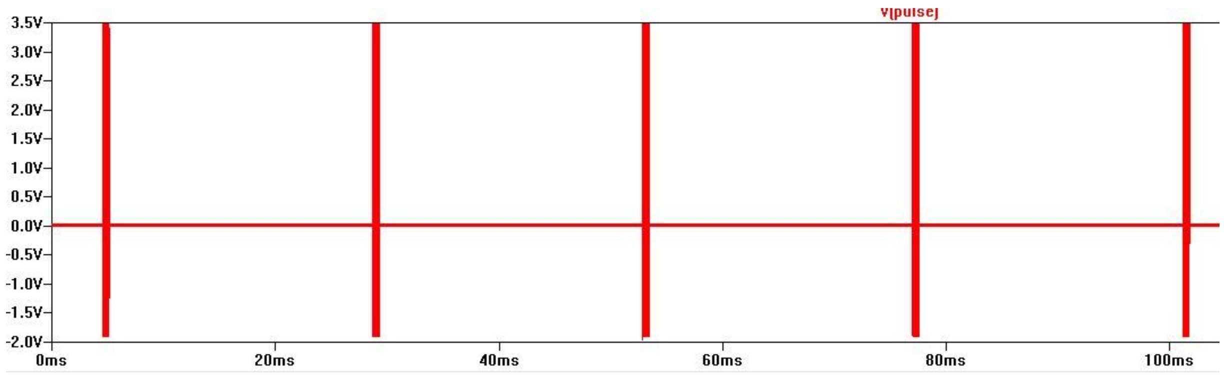


FIGURE 21: COMMON EMITTER PULSE GENERATOR OUTPUT

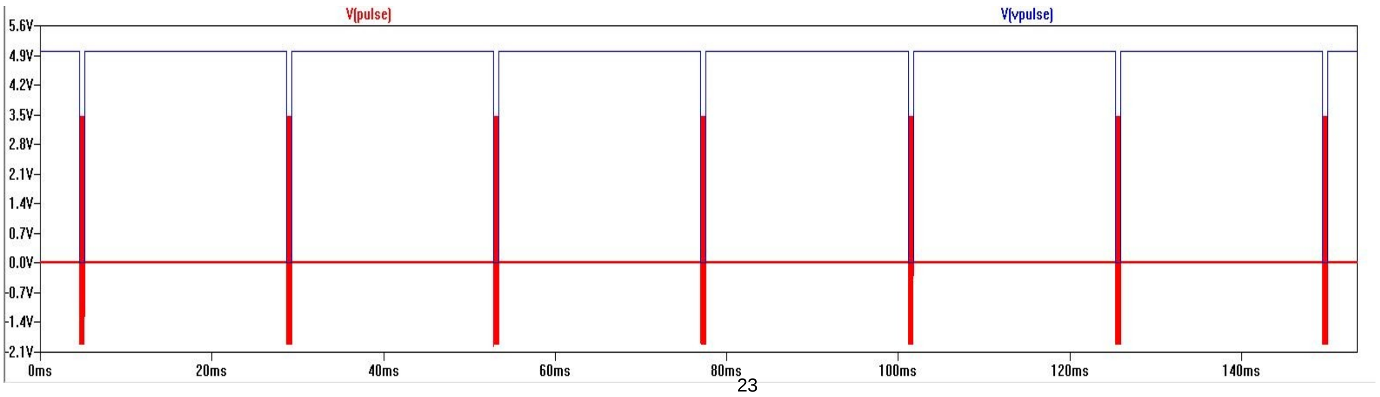


FIGURE 22: NE555-ASTABLE OUTPUT VS COMMON EMITTER OUTPUT

From table 5, we can conclude that the simulated and the theoretically calculated pulse properties are very similar.

Simulated values	
Period	24.1 ms
Duty cycle	97.5 %
Time high t_1	23.5 ms
Time low t_2	610 μ s

Table 5: Simulated pulse properties (Approximated) – NE555-Astable circuit

Practical Measurements

The NE555-astable and the transistor circuits were tested simultaneously. Figure 15 shows the output of the practical NE555-astable circuit, figure 16 shows the output of the Pulse generator circuit, which is to be transmitted.

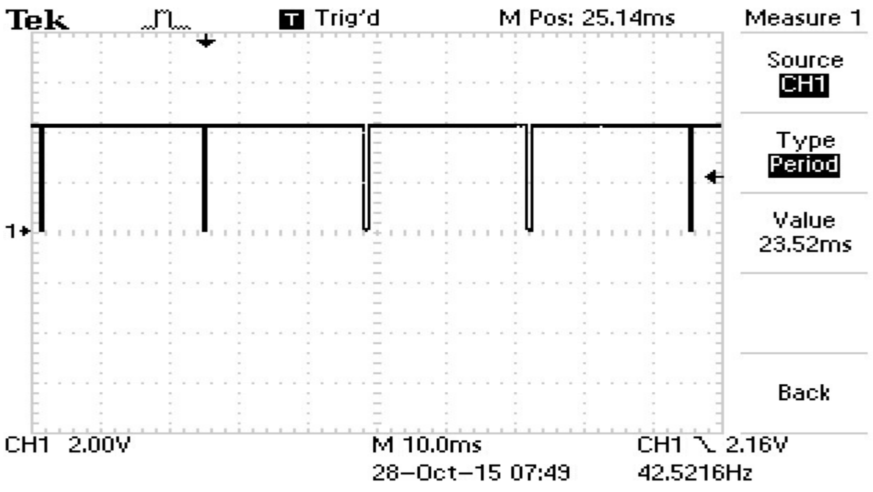


Figure 23: NE555-astable practical output

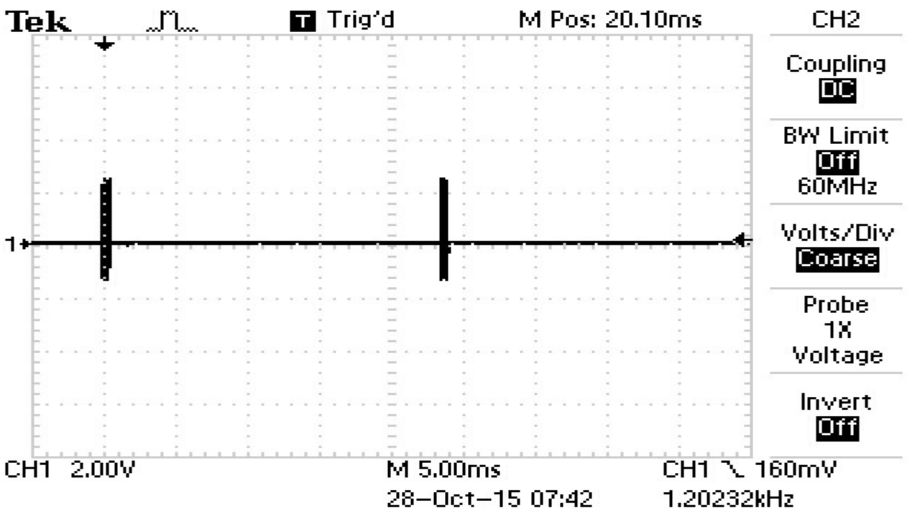


FIGURE 24: PULSE GENERATOR CIRCUIT OUTPUT (TRANSMITTED PULSE)

Conclusion and recommendations

From table 5, and the previous sections, it is clear that theoretical, simulated and actual signal properties are very similar, and can therefore be used.

Pulse generator output			
	Theoretical	Simulated	Practical
Period	24.089 ms	24.1 ms	23.52 ms
Duty cycle	97.40 %	97.5 %	97.1 %
Time high t_1	23.479 ms	23.5 ms	23 ms
Time low t_2	609.840 μ s	610 μ s	600 μ s

TABLE 6: NE555- ASTABLE PULSE GENERATOR OUTPUT

It is recommended that a variable resistor is used for R2, to improve control over the pulse width and duty cycle.

5.7 Receiver Gain

As a result of the transmitted signal travelling a distance through the air, the received signal is expected to be a very small signal, thus we have to amplify this signal significantly. This amplification will make the received signal usable in other circuits, where it will play a crucial role.

Design

A three stage amplifier will be use, instead of one, to minimize the impact of input and output loading, thus creating an amplifier with high input resistance, low output resistance and high gain stages. The gain of this circuit is around 6000. Before the final stage of amplification, a capacitor is used to ensure that there isn't a DC offset in the signal, a DC offset could vastly limit the functionality of some circuital components. De-coupling capacitors were used to ensure that the circuit does not oscillate, and to give smoother output signals – this is important, as any noise will also be amplified.

Gain:

$$G=\frac{R5}{R19}\times\frac{R6}{R18}\times\frac{R10}{R16}$$

$$¿\frac{27k}{1.2k}\times\frac{27k}{1.2k}\times\frac{560k}{47k}$$

$$¿6031.914$$

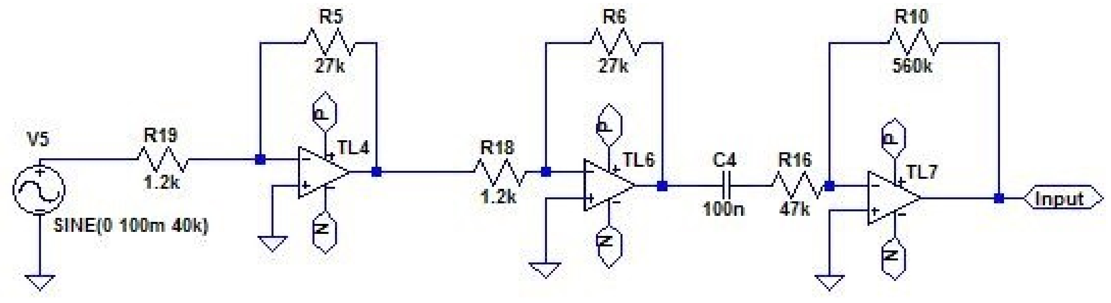


FIGURE 25: RECEIVER GAIN

Analysis

The SPICE circuit is shown in figure 20, the output of the circuit is obtained during simulation and shown in figure 21. We can clearly that the signal that will be received, is hit against the rails, this is necessary, as we want a very high gain in order to be able to see the received pulses.

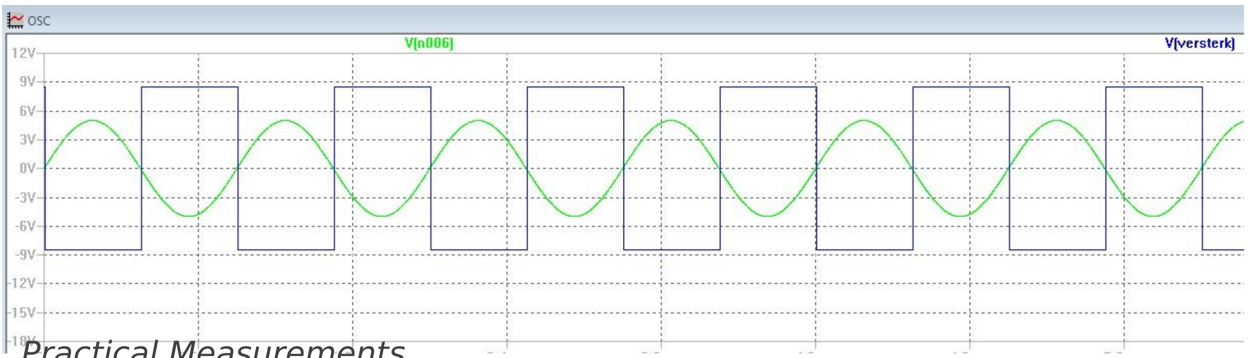


FIGURE 26: SIMULATED GAIN STAGE OUTPUT

Practical Measurements

The practical circuit was tested with a 1 V_{p-p} Sinusoidal signal input, and at supply voltages of +10 V and -10 V. The output signal from figure 22, was measured to be about 9 V_{p-p}. This circuit clearly gives high gain without any distortion or DC-offset.

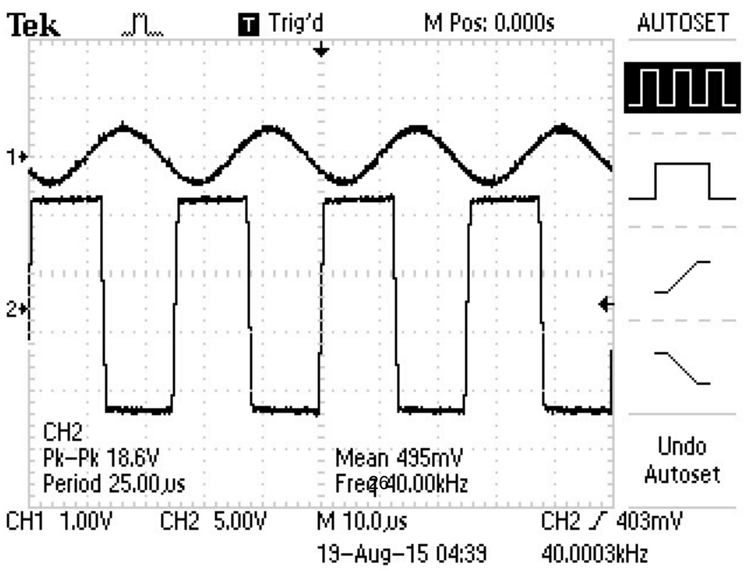


Figure 27: Practical Receiver Gain stage output

Conclusion and recommendations

Figures 21 and 22 show's that the practical circuit, which will be used for amplification at the ultrasonic receiver, has sufficiently large gain with minimum amounts of distortion and noise.

Form table 7 it can be seen that the gain stages are identical in all three cases.

Receiver Amplifier		
Theoretical Gain	Simulated Gain	Practical Gain
6000	6000	± 6000

Table 7: Receiver Amplifier Gain

It is recommended that a variable resistor is used for R10, to improve control over the total gain of the circuit.

5.8 *Envelope and Level Detector*

Envelope Detector

The next step was to design an envelope detector, to detect the envelope of the received signal. One important thing to note when designing an envelope detector is not to choose an RC time constant which is too large, or too small, thus some calculations were necessary.

Level Detector

The envelope needs a proper digital pulse shape, thus a Level detector is used to cause the envelope of the signal to hit against the rails of the operational amplifier. By giving an opamp a reference voltage, we make use of the operational amplifiers very high gain in open-loop state, whereas all values, below the reference voltage, are ignored, and all values above the reference voltage are given a very high gain, causing the signal to obtain a square form.

Design

Envelope Detector

For the envelope detector design, a diode detector was chosen, as the only calculations that has to be done is choosing a time constant.

Calculations:

Choose RC time constant to be $5RC \gg \frac{2\pi}{\omega_c}$

Thus, $RC \gg \frac{1}{\omega_c}$ is all that needs to be satisfied, where $\omega c = 2\pi(40\,kHz)$

Choose
$C = 100\,nF$

Then the resistor should be given a value of about $R = 12\,k\Omega$

(St. Andrews)

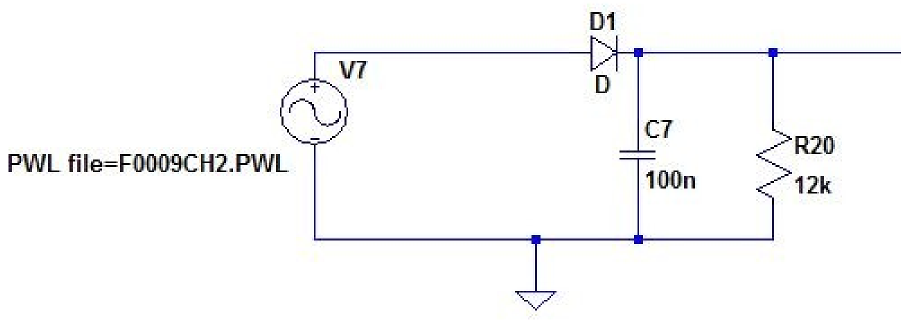


FIGURE 28: ENVELOPE DETECTOR

Level Detector

For the level detector design, a non-inverting Comparator Circuit was chosen. In this configuration the reference voltage, which will be controlled by a Voltage divider, is connected to the inverting input of the operational amplifier and the actual input signal connected to the non-inverting input.

Choose $R1 = 10\,k\Omega$

$R2 = 10\,k\Omega\ variable\ resistor$

We make $R2$ a variable resistor to control the reference voltage

(Electronic Tutorials)

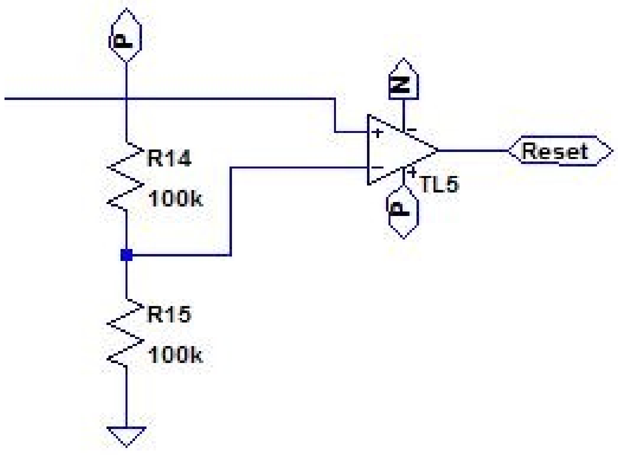


FIGURE 29: LEVEL DETECTOR

Analysis

The SPICE circuits are shown in figures 23 and 24, the output of each circuit is obtained during simulation and shown in figures 26 and 27. The output of the Envelope detector is fed into the non-inverting input of the comparator circuit. The combination of the Envelope-and level detector produces the output, seen in figure 28, which is clearly a digital signal that can be used for the reset of the latch, covered in the next section.

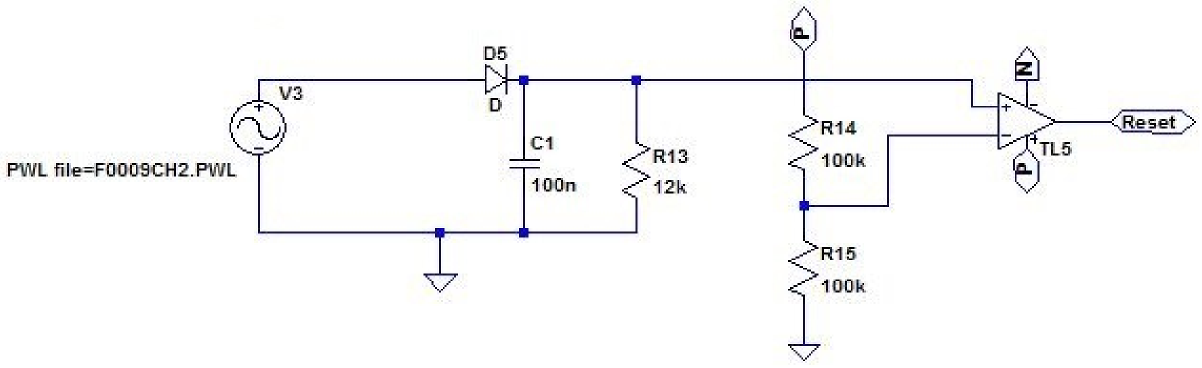


FIGURE 30: ENVELOPE AND LEVEL DETECTOR COMBINATION

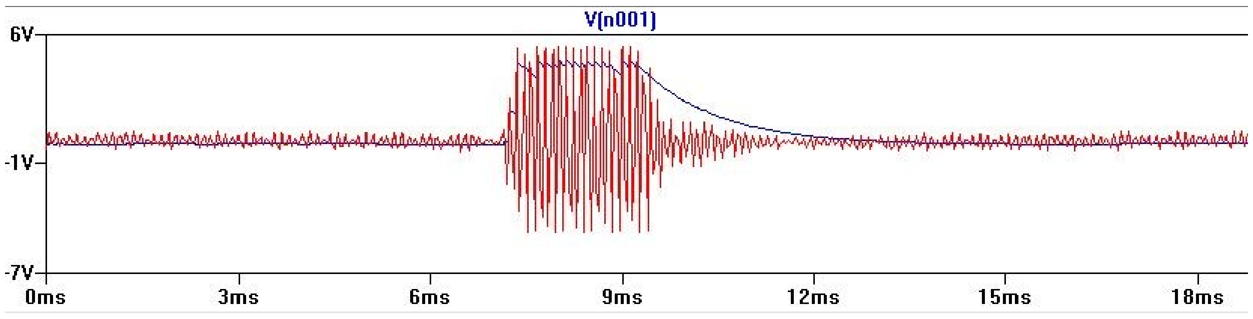


FIGURE 31: ENVELOPE DETECTOR SIMULATED OUTPUT

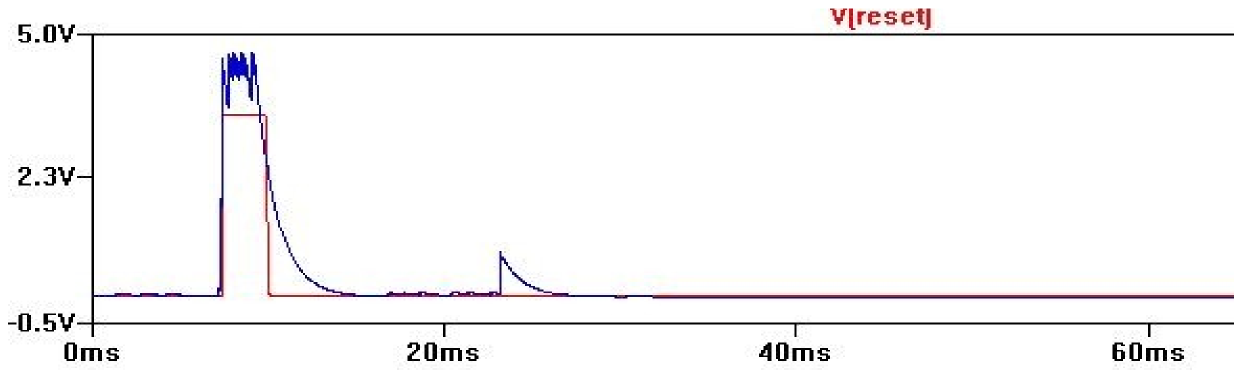


FIGURE 32: LEVEL DETECTOR SIMULATED OUTPUT

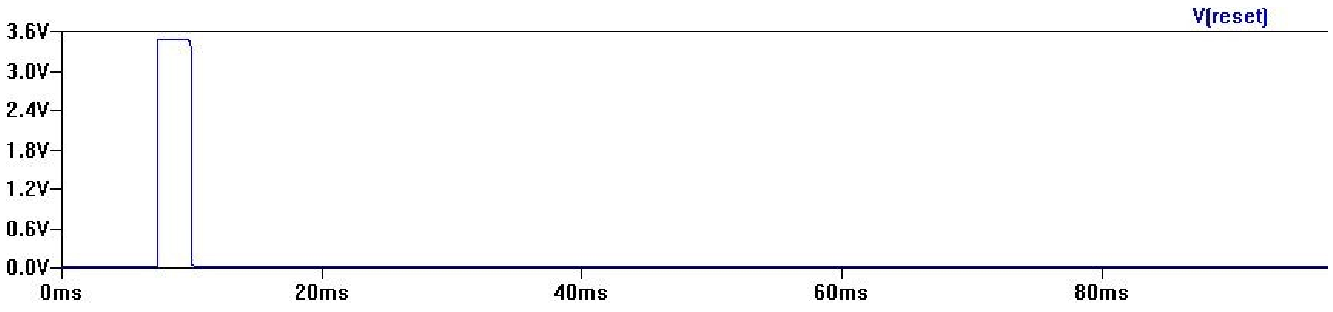
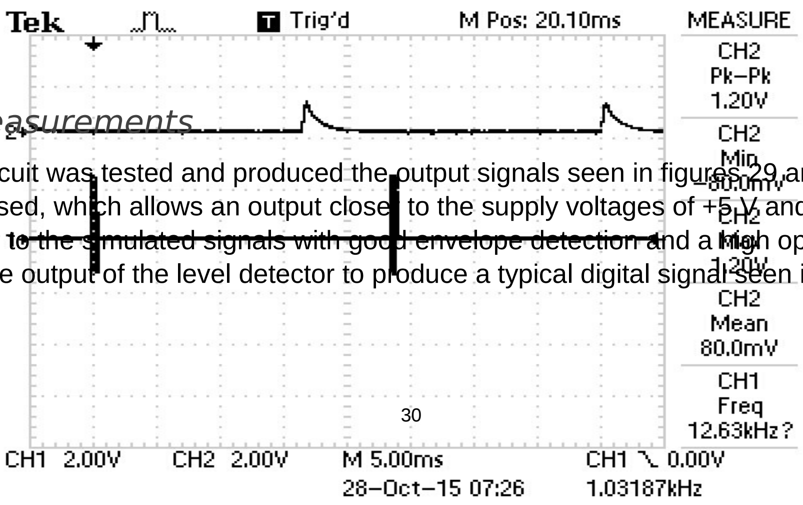


FIGURE 33: RESET - DIGITAL SIGNAL



Practical Measurements

The practical circuit was tested and produced the output signals seen in figures 29 and 30. A rail-to-rail transistor was used, which allows an output closer to the supply voltages of +5 V and 0 V. The output signals look very similar to the simulated signals with good envelope detection and a high open loop voltage gain, which causes the output of the level detector to produce a typical digital signal seen in figure 30.

FIGURE 34: EVELOPE DETECTOR PRACTICAL OUTPUT VS TRANSMITTED PULSE

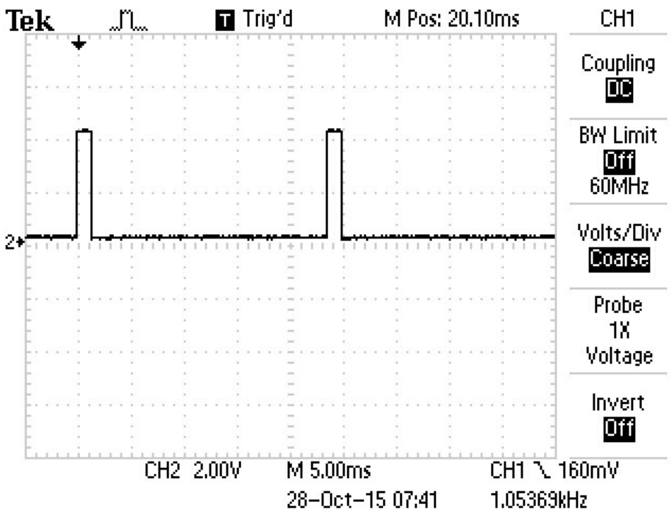


FIGURE 35: LEVEL DETECTOR PRACTICAL OUTPUT

Conclusion and recommendations

From table 2 we can see the simulated envelope and Level detection versus the practical envelope and level detection amplitudes. From this we can conclude that the envelope and level detectors work as expected and that this circuit can be used for the ultrasonic distance sensor circuit.

Envelope and Level detector			
	Simulated	Practical	
		30 cm	2 m
Envelope detector	4 V	2 V	1.88 V
Level detector	3.5V	5.04 V	5.04 V

TABLE 8: ENVELOPE AND LEVEL DETECTOR AMPLITUDES

It is recommended that a variable resistor is used for R2, in order to control the time constant.

5.9 S/R Latch

An S/R latch (set/Reset) is an asynchronous device: it works independently of control signals and relies only on the state of the S and R inputs.

Design

A latch will be created by two NOR gates with a cross- feedback loop. When a high is applied to the set line of the S/R latch, the Q output goes high. The cross-feedback loop causes the output to remain high, even when the S input goes low again. Only a high input to the Reset line will cause the Output Q to go low.

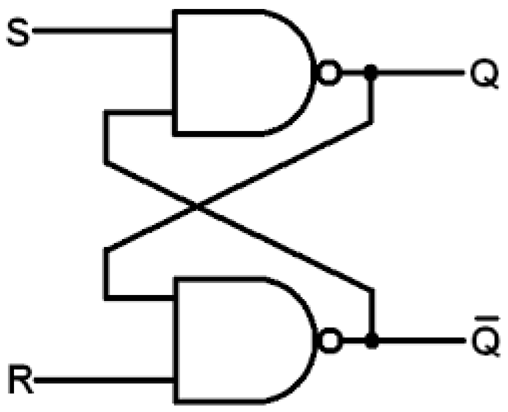


FIGURE 36: NOR GATES WITH CROSS-FEEDBACK LOOP - S/R LATCH

Analysis

The SPICE circuit is shown in figure 32, the output of the circuit is obtained during simulation and shown in figure 33. The output of the NE555-astable circuit will be used as the Set, to initially drive the Q output high and the Level detector output will be used as the Reset, to set the Q value low again. Output shown in figure 33 is only conceptual and is not a true reflection of what the output will be, lower voltages are due to incorrect transistor models used during simulation.

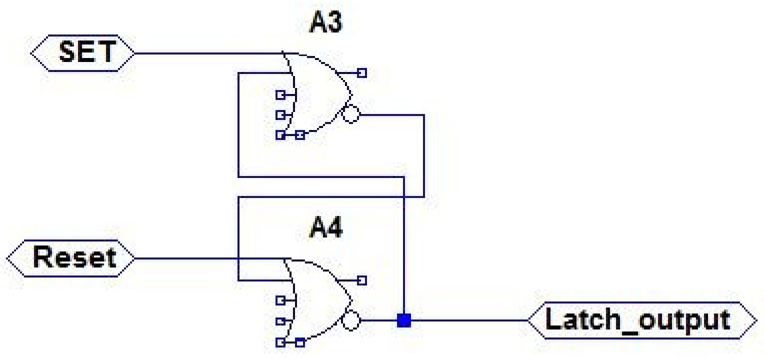


FIGURE 37: S/R LATCH

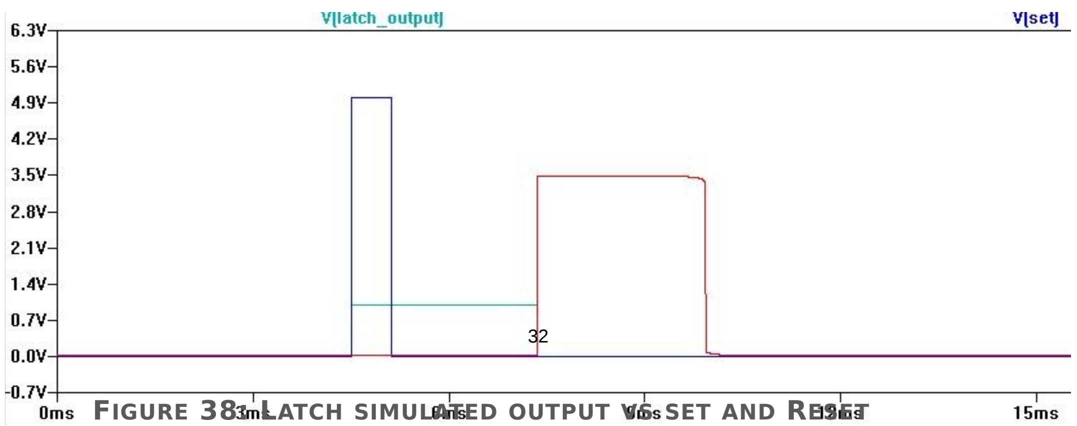


FIGURE 38: LATCH SIMULATED OUTPUT VS SET AND RESET

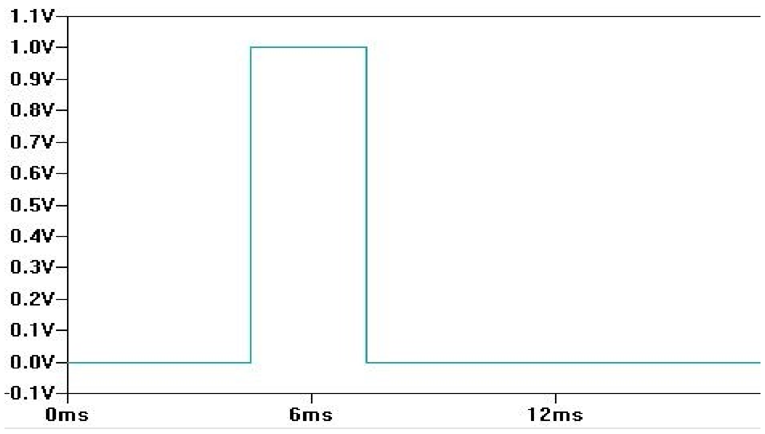


FIGURE 39: LATCH SIMULATED OUTPUT

Practical Measurements

The practical circuit was tested and produced the output signal seen in figure 35. We can see that the Latch Set the Q output to high on the rising edge of the NE555-astable and resets on the Received signal (Level detector output).

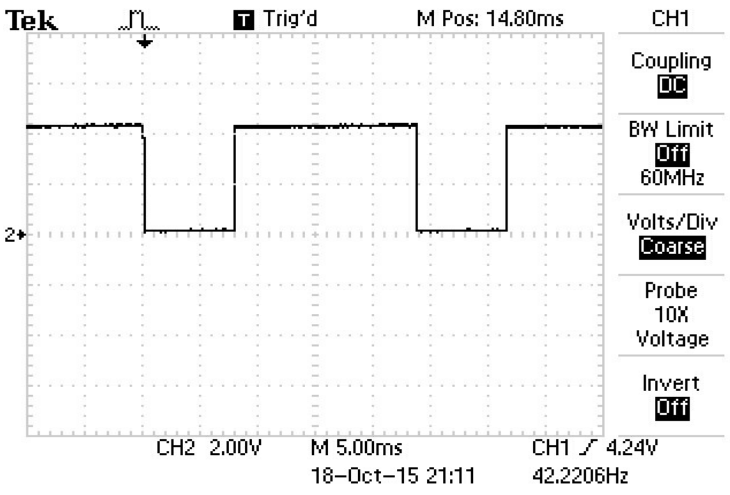


FIGURE 40: LATCH PRACTICAL OUTPUT

Conclusion and recommendations

Comparing figures 34 and 35, it is clear to see that the practical latch output looks very similar to the simulated latch output and can thus be used in the ultrasonic distance sensor circuit.

5.10 *NE555-Monostable*

One problem that was noticed during the testing of this circuit, was that there is a large amount of coupling between the ultrasonic transmitter and receiver. To rectify this problem, a NE555-monostable circuit was used to generate a slightly longer pulse than the Astable circuit, which effectively causes the latch to ignore any coupling in the received signal. (CSGNetwork.com)

Design

The monostable circuit produces one pulse of a set length in response to a trigger input, which in this case is the NE555-astable circuit. The monostable output will only become high if it was triggered. [4]

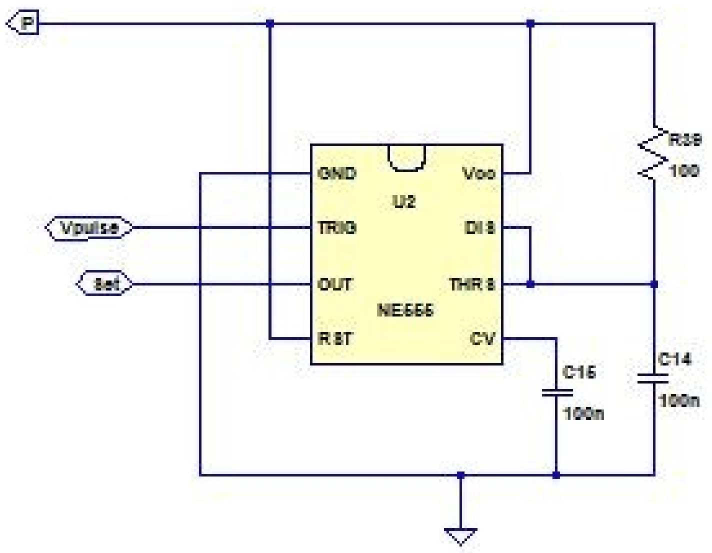
Time set high

$$t = 1.1 \times RC$$

Using this equation Resistor and Capacitor values were chosen.

$$R = 100\,k\Omega\ variable\,resistor \qquad C = 100\,nF$$

This will allow a monostable pulse almost double the width of the astable pulse.



Analysis

The SPICE circuit is shown in figure 36, the output of the circuit is obtained during simulation and shown in figure 37. The output of the NE555-monostable circuit will now replace the output of the NE555-astable to be used as the Set for the S/R latch. Figure 37 shows that the monostable is triggered by the astable circuit, and produces a slightly wider pulse.

FIGURE 41: NE555-MONOSTABLE CIRCUIT

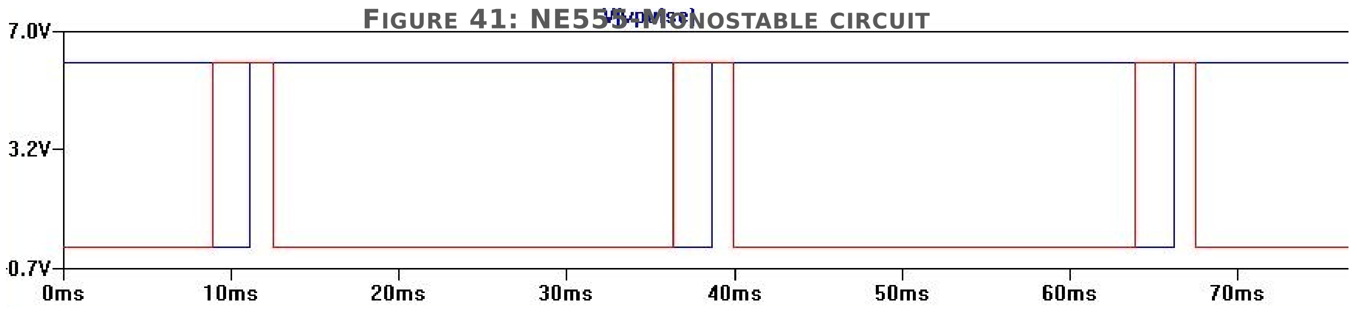


FIGURE 42: NE555-MONOSTABLE SIMULATED OUTPUT

Practical Measurements

The monostable circuit was built from the design in figure 36. After testing the circuit, output shown in figure 38 was produced. In this figure channel 1 is the monostable circuit output, and channel 2 is the astable trigger. The monostable triggers on the astable and produces pulses with a width of

$$t = 1.1 \times RC$$

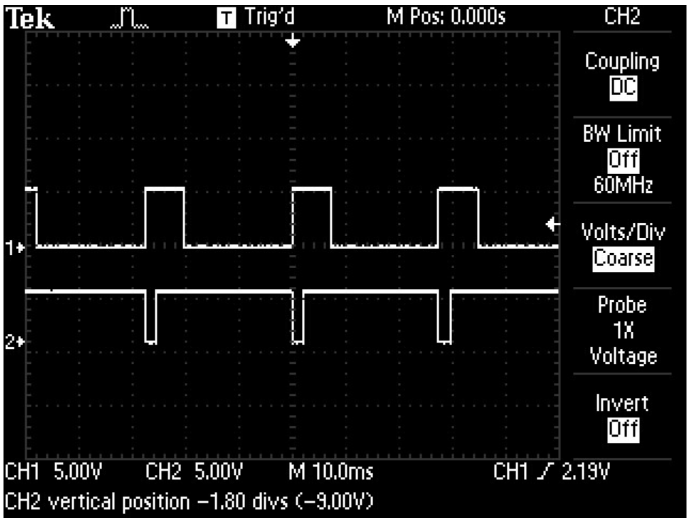

Tek Trig'd M Pos: 0.000s CH2
Coupling DC
BW Limit Off 60MHz
Volts/Div Coarse
Probe 1X Voltage
Invert Off
CH1 5.00V CH2 5.00V M 10.0ms CH1 ∕ 2.19V
CH2 vertical position −1.80 divs (−9.00V)

FIGURE 43: NE555-MONOSTABLE OUTPUT VS NE555-ASTABLE TRIGGER

Conclusion and recommendations

From table 9, it is clear that the pulses produced by the monostable circuit is wider, and can therefore be used to counteract coupling caused by the ultrasonic transmitter and receiver.

Monostable pulse width		
	Astable	Monostable
Pulse width	610 μ s	610 μ s – 1.1 ms

TABLE 9: MONOSTABLE VS ASTABLE PULSE WIDTHS

5.11 *Final Stage Filter*

In order to convert the output of the latch to a DC voltage, a filter with a low cut-off frequency and a high Quality factor, needs to be designed.

Design

To ensure a low cut-off frequency and a high quality factor a two stage sallen-key LPF (Low Pass Filter) was chosen.

Choose: $R1=R2=150\,k\Omega$

$C1=C2$

$f_c=10\,Hz$

Calculations [5]

Cut off frequency: $2\pi f_c=\frac{1}{\sqrt{R1R2C1C2}}$ $C=0.106\,\mu F$

Choose $C=100\,nF$

Then $f_c=10.6\,Hz$

Quality Factor: $Q=\frac{\sqrt{R1R2C1C2}}{C1(R1+R2)}$ ¿0.5

(Texas Instruments)

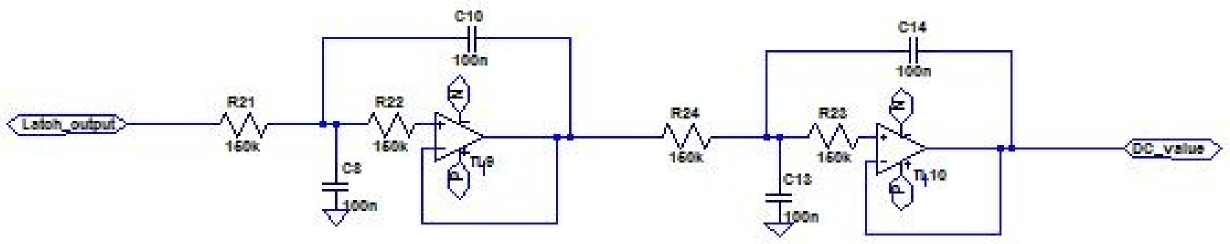


FIGURE 44: SALLEN-KEY LPF

Analysis

The SPICE circuit is shown in figure 39. After performing an AC-analysis in spice, the frequency response is obtained and shown in figure 40. From this figure it is clear that the cut-off frequency is close to 10 Hz with a high quality factor. This cut-off frequency and quality factor is highly desirable and will work extremely when converting the pulses to DC.

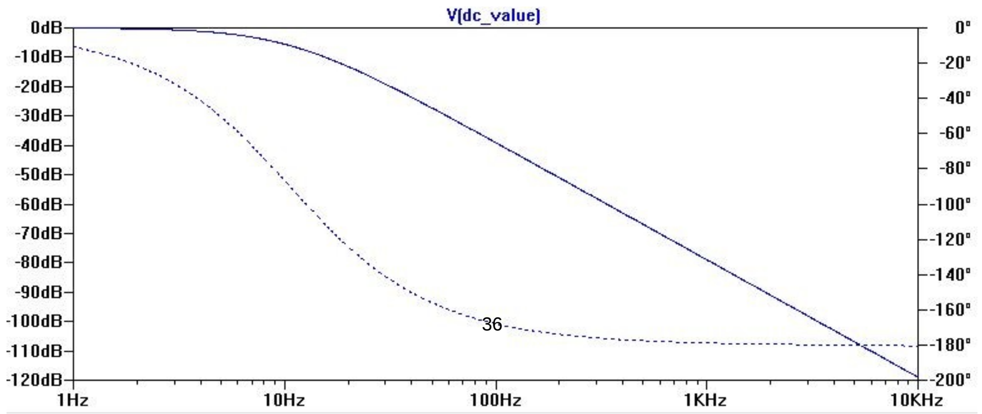


FIGURE 45: SALLEN-KEY LPF FREQUENCY RESPONSE

Practical Measurements

The final stage filter was built from the design in figure 39. After testing the circuit, output shown in figure 41 was produced. The DC voltage is at a stable level, which means that the cut-off frequency is low enough and the Quality factor is high enough to produce a stable DC voltage.

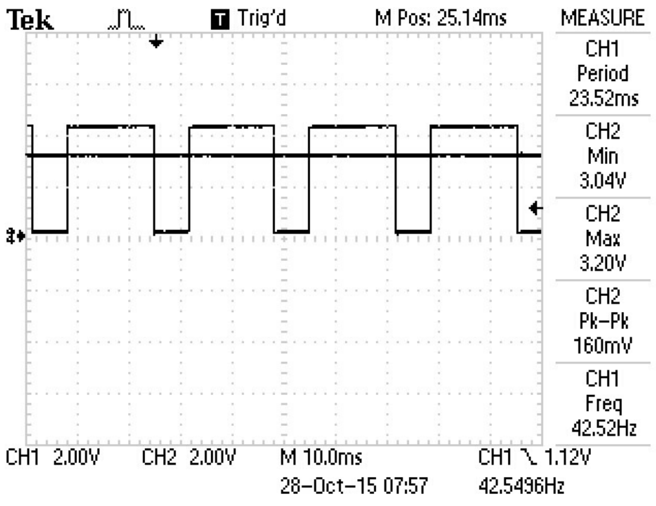


FIGURE 46: FINAL STAGE FILTER PRACTICAL OUTPUT VS LATCH OUTPUT

Conclusion and recommendations

Looking at figure 41, the conclusion can be made that the two-stage sallen-key LPF is a good design and can be used to produce the final DC voltage level.

Table 10 shows how the voltage level differs at different distances.

Sallen-key LPF output	
Distance	Voltage Level
30 cm	500 mV
3 m	1.5 V

TABLE 10: SALLEN-KEY LPF OUTPUT AT DIFFERENT DISTANCES

5.12 LED circuit

The purpose of this circuit is to visually show when an object is close or far away, switching LED's on and off at certain distances.

Design

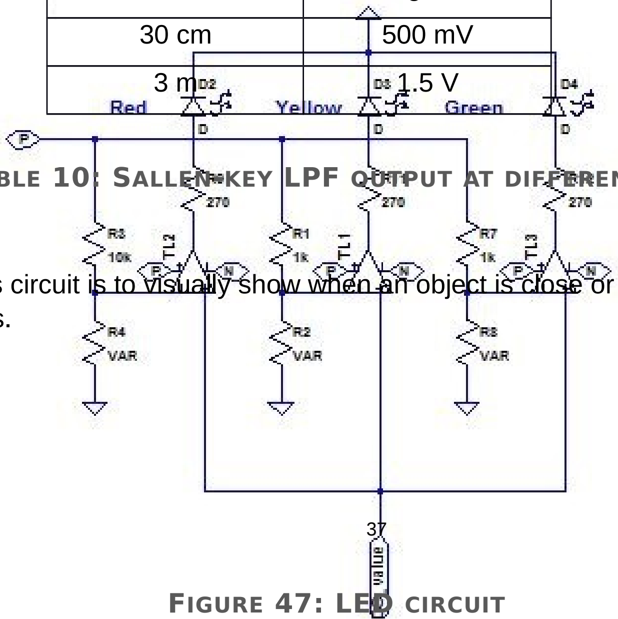


FIGURE 47: LED CIRCUIT

For practical purposes, it was decided to use voltage comparator circuits for each LED.

The DC level after Filtering is fed into the non-inverting input of each LED circuit.

For the inverting input, a voltage divider generates a reference voltage.

For each voltage divider:

$$R1 = 1\,k\Omega$$

$$R2 = 1\,k\Omega\,variable\ resistor$$

Reference voltage:	Green LED – 2 V
	Yellow LED – 1 V
	Red LED – 200 mV

Thus at each level, the output of the operational amplifier will go high and switch on the applicable LED’s, see table 11 for details.

The output of the operational amplifier is fed through a 270 Ω resistor, which limits the current, to the LED.

LED circuit Details (X indicates LED is on)			
Distance	Green LED	Yellow LED	Red LED
-30 cm			X
1 m		X	X
+ 2 m	X	X	X

TABLE 11: LED CIRCUIT DETAILS

Conclusion and recommendations

After testing the circuit, it was observed that the LED circuit works as expected, and can therefore be used to visually indicate distances.

It is recommended that an array of coloured LED’s is used instead of one of each colour, to improve the visual effect that the LED circuit has.

6. System Integration

When connecting all the sub-circuits together, one thing that has to be kept in mind is the input and output impedance of each stage. In this circuit design operational amplifiers and active filters were used, resulting in high input impedance and a low output impedance, which is highly desirable. In some cases a buffer was used to prevent inter-circuital loading, but in general there was minimal loading effects observed.

The pulses that are sent for transmission are displayed in figure 43. This output is identical to the output before system integration.

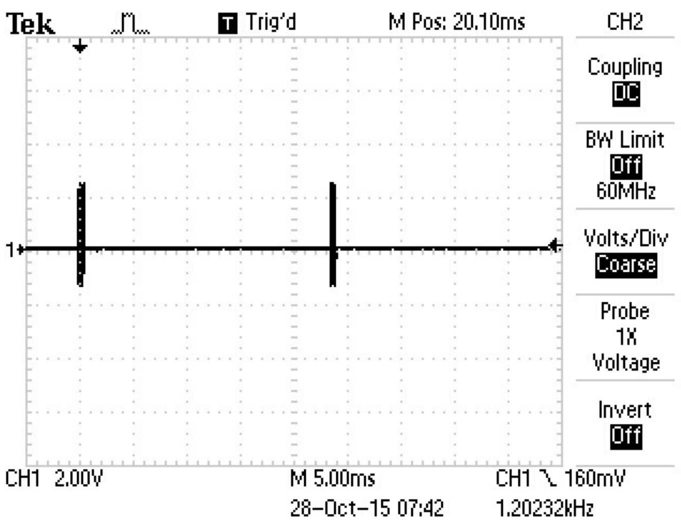


FIGURE 49: TRANSMITTED PULSE - INTEGRATED CIRCUIT

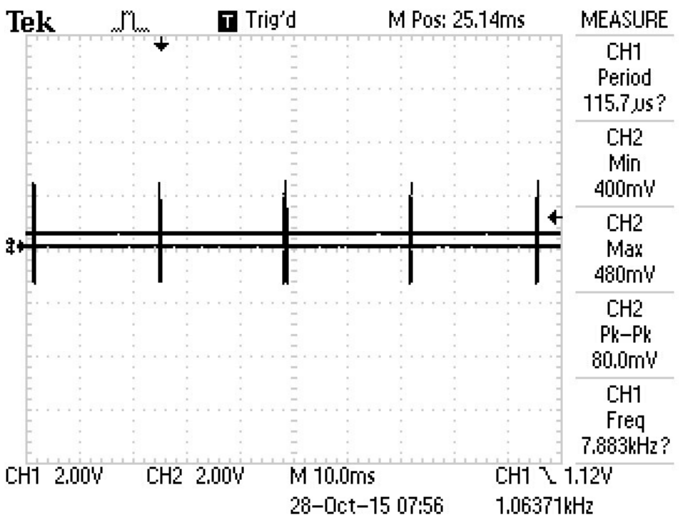


FIGURE 48: INTEGRATED SYSTEM FINAL OUTPUT VS TRANSMITTED PULSE

Figures 43 and 44 both show that the complete circuit integration did not have a negative effect on the measured output, as the circuit functions as expected.

The final system output can be viewed in figure 44. This output is identical to the output before system integration. Measured results can be found in Appendix A.

The complete circuit diagram, along with a photo of the complete circuit and PCB (Printed Circuit Board) design, can be found in Appendices C to E.

7. Software

In order for the output of the Analogue system to be converted to distance, software had to be developed to interoperate the DC level fed to the Arduino and perform the necessary calculations with a pleasant GUI. The programming language used to develop the distance sensor software, was Python. The TKinter class allowed a GUI to be created to visually present the distance to the user and to allow some control over the functionality of the Software.

Software Engineering

Below are some examples of Software engineering concepts.

Unit Testing:

This is a software engineering principle, whereas functions are written to test developed code. Unit testing hold in various advantages which include, more correct software, prevent collaborators from causing bugs in the software, when they update it. Unit testing does however hold in a few disadvantages, as writing unit tests can be time consuming, tests may contain false positives and some unit tests may be very complex and will thus be very hard to implement, and will have a long execution time.

Version control:

Version control is a system that records changes to files over a period of time by users, specifically so you can go back to previous version of that file. This is very handy as you can compare versions very easily and see all changes made over time. One will thus also always have a working copy. [6]

(Getting Started – About Version Control)

Git Server:

A Git server is basically the repository. This is where the master files are situated, and all team members have access to. It is summed up as a collaboration point for all members working on a project. [7]

(Git on the Server)

Git Client:

The Git client is the program that the user uses to access the repository and pull the master project and work parallel to team members on different parts of the project. This client program is very handy for creating branches where a user can make changes without breaking the original project.

Workflows:

Centralised workflow:

- someone creates central repository
- Everyone clones repository
- Everyone works in parallel
- Everyone pushes to origin individually

Feature branch workflow:

- Uses centralised workflow
- All features are added in a dedicated branch, rather than the master branch
- The master project is always in working order.

Forking workflow:

- Does not use centralised workflow
- Each member has his/her own central database to push to and a public repository
- Creates a copy of initial repository rather than a clone
- No other developers can push to personal repositories, but they can pull from it
- “git clone”
- This is a powerful workflow for loosely knit teams

(Git Comparing workflows)

Software Design

For the software design a Feature branch workflow was used, which means that the master project was always in working order and no that no one could accidently break the code.

The software was split into four categories, Connection, stream, Settings and unit testing. Four members could thus each work individually on either:

- Connection widget
- Stream widget
- Settings widget
- Or unit tests

This approach enabled developers to work parallel to one another, which decreased the development time vastly. Software integration was also very easy, as developers could communicate freely and share code when it was necessary.

User Instructions – Distance Sense

See figure 45 as reference.

Getting Started

- Connect Arduino Leonardo to PC
- Go to Device manager and view COM port connection for the Arduino

Software instructions

- Open the Distance Sense Program

Connection widget:

- In the “Serial Port” text field type the COM port at which the Arduino is connected
- In the “baud rate” text field type 9600
- Click the Connect button

If the connection fails, perform the following checks:

- Is the device connected
- Does the device have power (are the LED’s on the Arduino on)
- Is the correct COM port chosen
- Is the baud rate 9600

Stream widget:

- If the analogue circuit is connected and running, click the “turn stream On” button, to view distances and velocity
- Click the button again to turn the data stream off.

Settings widget:

The following commands can be typed into the “Enter command text field”

Command: **calibrate.param**

Description: Captures and saves the sensor values at reference distances, which are then used to calibrate the distance scale. The calibration requires one measurement with an object placed at the zero reference point of the distance measurement, and one measurement with the object placed one meter away from the zero reference. The measurements must be done in a straight line with the distance sensor pointing directly at the object. These settings will be remembered if the Arduino is powered off and on again. The possible values of param are:

- Zero – saves the sensor value for the zero reference
- One – saves the sensor value at one meter away from the zero reference

Command: **set.param.value**

Description: Configure the distances where the LED should be turned on (initially green when turned on at furthest distance), at what distance to change from green to yellow, and at what distance to change from yellow to red. The distances should be specified as integers. These settings will be remembered if the Arduino is powered off and on again. The possible values of param are:

- Green – configures LED to turn on and be green at value cm
- Yellow – configures LED to change from green to yellow at value cm
- Red – configures LED to change from yellow to red at value cm

Command: **led.param**

Description: Controls the LED connected to pins 3 (red) and 5 (green) of the Arduino. The possible values of param are:

- Green – stops using measurement data to update LED and turns it green
- Red – stops using measurement data to update LED and turns it red
- Yellow – stops using measurement data to update LED and turns it yellow
- Sensor – updates LED using measurement data with the configured LED colour thresholds (see command: set.param.value)
- off – stops using measurement data to update LED colour and turns it of

Command: **sensor.param**

Description: Controls the sampling of the analogue signal of the sensor. This setting will be remembered if the Arduino is powered off and on again. The possible values of param are:

- On – configures the analogue signal to be sampled every 100 milliseconds
- Off – stops sampling the analogue signal

Command: **stream.param**

Description: Controls whether the Arduino sends measurement data to the PC or not. This setting will be remembered if the Arduino is powered off and on again. The possible values of param are:

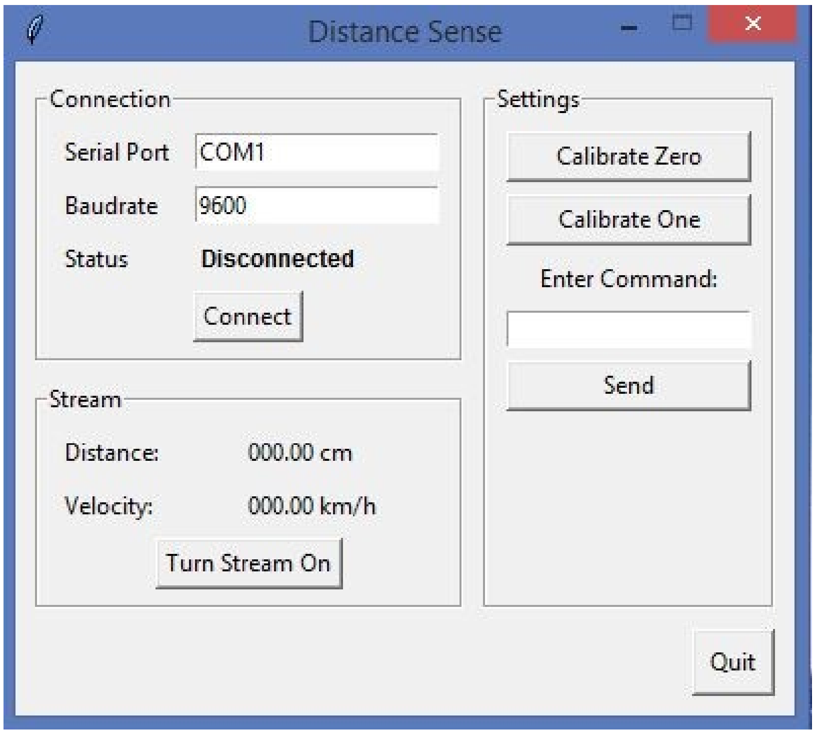
- On – configures the Arduino to send the measured distance to the PC (in centimetres)
- Off – stops sending measurement data to the PC

Ending the program

- To Exit the program simply click the quit button in the bottom right corner of the screen.

(Ontwerp/Design E344 courses webpage) all command descriptions were found in the “Arduino details” pdf.

See Appendix F for all Python Code.



Serial Port COM1
Baudrate 9600
Status **Disconnected**
Distance: 000.00 cm
Velocity: 000.00 km/h

FIGURE 50: GUI - DISTANCE SENSE

Software conclusion and Recommendation

This program is fully functional and can accurately measure distance and velocity of an object. The GUI is also very user friendly. In terms of basic function the software developed is sufficient to fulfil the task of converting a DC level to distance and velocity. The Software can however be improved by adding more functionality such as Bluetooth capability, more visual indicators of distance and velocity, Graphs of distance and Velocity over time. Another improvement will also be to rather make all the commands which the users can type in buttons; this will prevent small typing errors.

8. Conclusion

In conclusion it is clear that the ultrasonic distance sensor can accurately measure distance with a range of 30 cm to 3 m, providing a certain DC Voltage level at certain distances. This information can be further interpreted by developed software to calculate distance and velocity and display it to the user. This circuit was a good design, which produced accurate results.

This system can however be improved by adding the following:

- A protective casing for the system, to ensure normal function during wet conditions.
- Adding Bluetooth communication could mean that a user could see real time measurements and alarms produced by the system on a phone or PC.
- Users may also want to save battery life if there is no necessity for the system, thus adding an on/ off button would increase functionality.
- Adding a Buzzer to the system will aid in warning the user when an object is too close.

The report also investigates the fact that the system may have some limitations

- The system has a measurement range of 30 cm to 3 m, which means that any distance further or closer than that will not be measured accurately.
- The system measures more than one object at a time, which can cause incorrect measurements.
- As the system is completely electronic, exposure to wet conditions could cause water damage, and complete system failure.

9. List of abbreviations

cm	Centimetre
m	Metre
s	second(s)
V	Volt
$V_{p\text{-}p}$	Volt (peak-to-peak)
A	Ampere
F	Farad
Ω	Ohm
Hz	Hertz
R	Resistor
C	Capacitor
L	Inductor
FFT	Fast Fourier Transform
GUI	Graphical User Interface
PC	Personal Computer
PCB	Printed Circuit Board
USB	Universal Serial Bus
LED	Light Emitting Diode
AC	Alternating Current
DC	Direct Current
f_c	Cut-off frequency (Hertz)
n	Nano

μ	Micro
m	Milli
K	Kilo
LPF	Low Pass Filter

1. References

[1] St. Andrews n.d., *The Envelope detector,* viewed 27 October 2015 https://www.st-andrews.ac.uk/~www_pa/Scots_Guide/RadCom/part9/page2.html

[2] Electronic Tutorials n.d., *Op-amp Comparator,* viewed 27 October 2015 http://www.electronics-tutorials.ws/opamp/op-amp-comparator.html

[3] CSGNetwork.com n.d, *Monostable 555 timeout calculator,* viewed 27 October 2015 http://www.csgnetwork.com/ne555timer1calc.html

[4] 555 Timer Circuits n.d., *555 Timer Operating Modes,* viewed 25 October 2015 http://www.555-timer-circuits.com/operating-modes.html

[5] Texas Instruments n.d., *Analysis of the Sallen-Key Architecture,* viewed 14 October 2015 http://www.ti.com/lit/an/sloa024b/sloa024b.pdf

[6] Git n.d., *Chapter 1 Getting Started,* viewed 22 September 2015 https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[7] Git n.d., *Chapter 4 Git on the Server,* viewed 22 September 2015 https://git-scm.com/book/en/v1/Git-on-the-Server

[8] Git workflows and tutorials n.d. *Comparing Workflows,* viewed 22 September 2015 https://www.atlassian.com/git/tutorials/comparing-workflows/

[9] Ontwerp/Design E344, *Arduino Technical details,* viewed 27 October 2015 http://courses.ee.sun.ac.za/Design_E_344/weeklytasks/Arduino.pdf

[10] Senix n.d., *Common ultrasonic sensor applications,* viewed 28 October 2015 http://senix.com/ultrasonic-sensor-applications/

[11] Kerry D. Wong n.d., *A sensitive DIY Ultrasonic Range Sensor,* viewed 28 October 2015 http://www.kerrywong.com/2011/01/22/a-sensitive-diy-ultrasonic-range-sensor/

[12] *Time of flight method*, n.d picture, viewed 28 October 2015
https://en.wikipedia.org/wiki/Radar

[13] Neaman, Donald A., 2010, *Microelectronics, Circuit Analysis and Design,* McGraw-Hill Education, Asia

Appendix A

Measured Results

Measurement	**Distance**					
	30 cm			**3 m**		
	Max	*Min*	*pk-pk*	*Max*	*Min*	*Pk-pk*
1. Transmitted signal	3.2 V	-1.2 V	-	3.2 V	-1.2 V	-
2. After amplifier stage	4.7 V	-2 V	-	4.1 V	-2 V	-
3. Envelope detector	2 V	-0.08 V	2.08 V	1.88 V	-0.08 V	1.96 V
4. Level Detector	5.04 V	0 V	5.04 V	5.04 V	0 V	5.04 V
5. S/R latch	4.24 V	0.08 V	4.16 V	4.32 V	0.08 V	4.24 V
6. Output DC Voltage	480 mV	320 mV	160 mV	3.20 V	3.04 V	160 mV

Appendix B

Technical Specification

Power

The system requires a 12 V battery supply. A Voltage regulator ensures a steady 10 V DC supply. A Virtual ground Supplies the operational amplifiers with the required positive and negative supply voltages of $+5V \wedge -5V$

Serial Bit Rate

Baud rate – 9600 baud

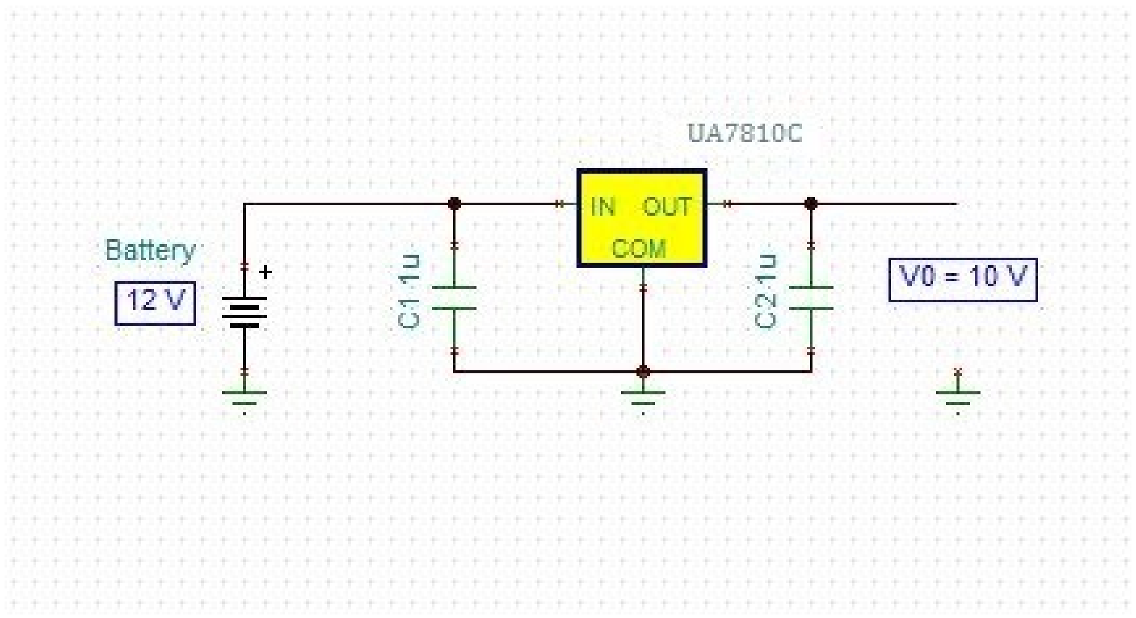
External Connections

USB – mini B cable for Arduino Leonardo microcontroller

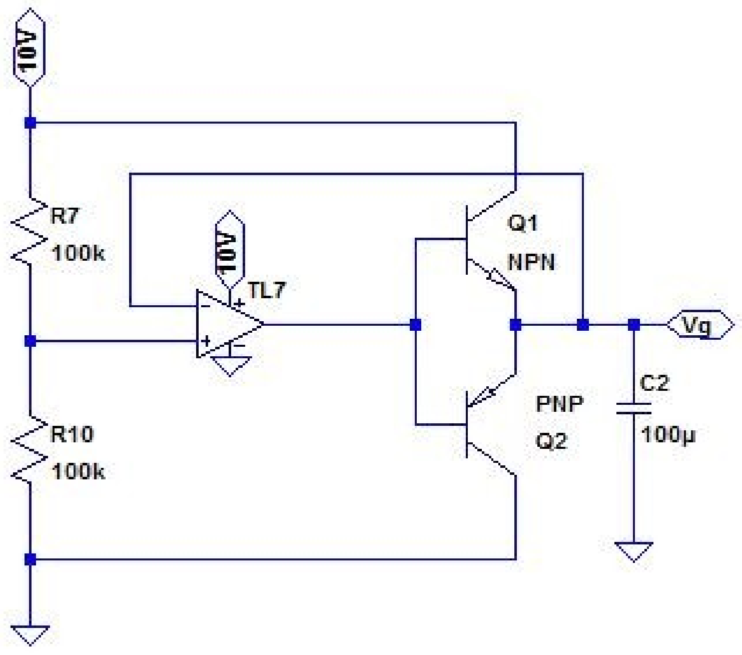
Appendix C

Circuit Diagrams

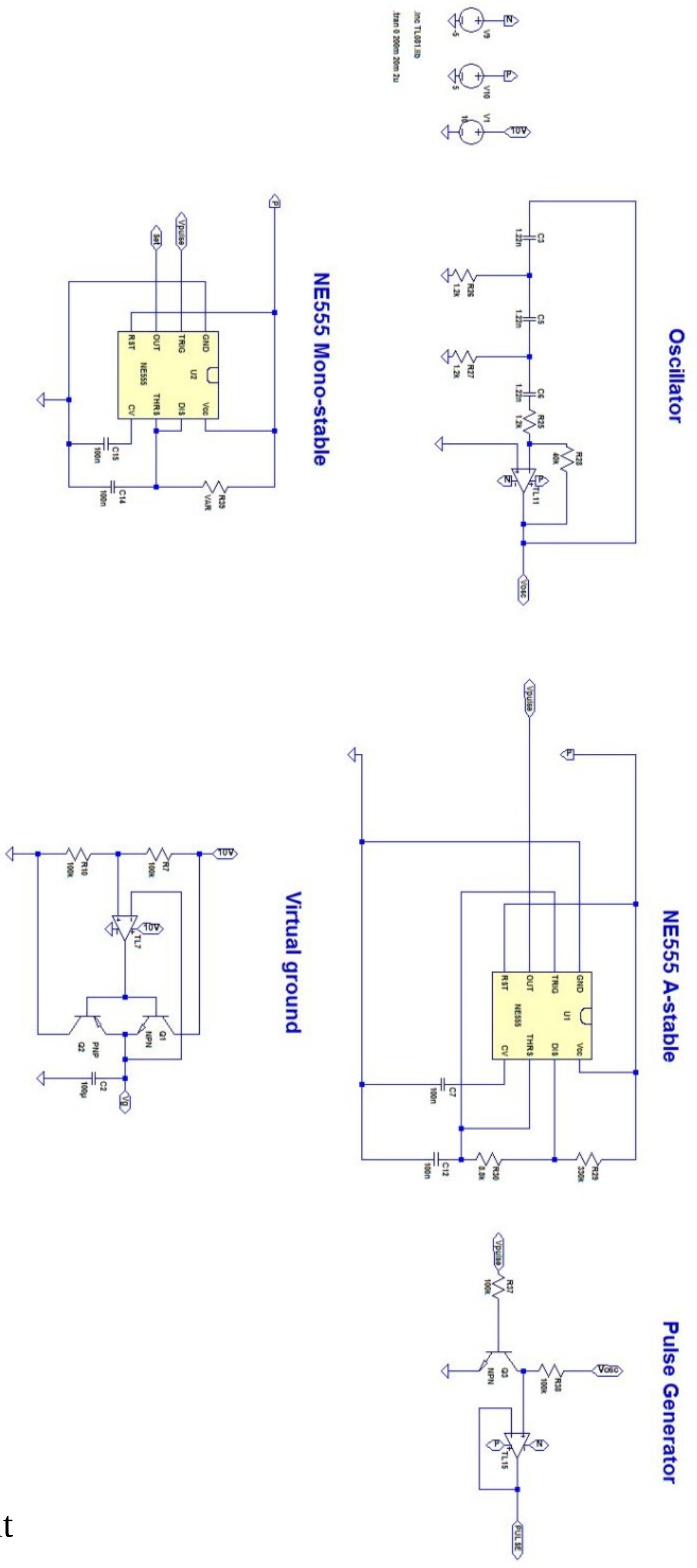
Power Supply



Virtual ground



Transmitter circuit



Receiver circuit

FIGURE 51: COMPLETE TRANSMITTER CIRCUIT

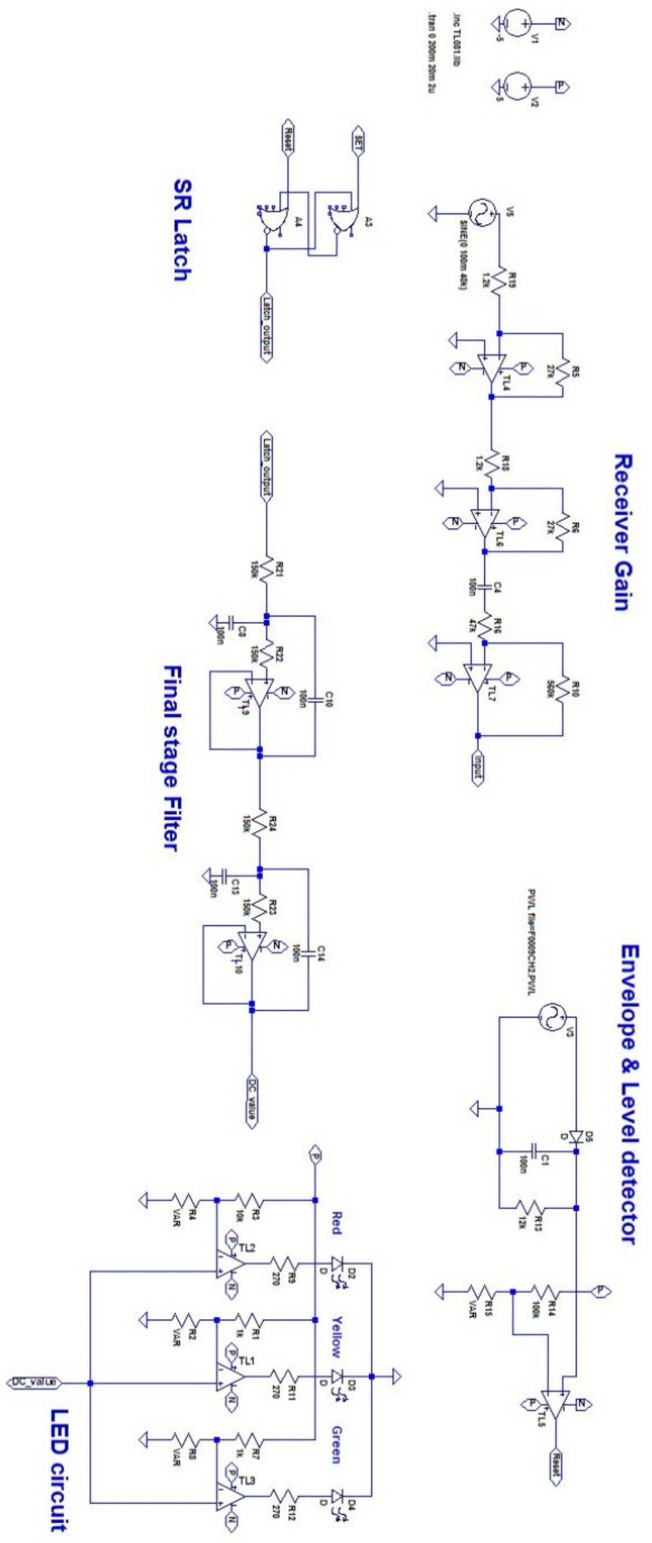


FIGURE 52: COMPLETE RECEIVER CIRCUIT

Appendix D

PCB Design

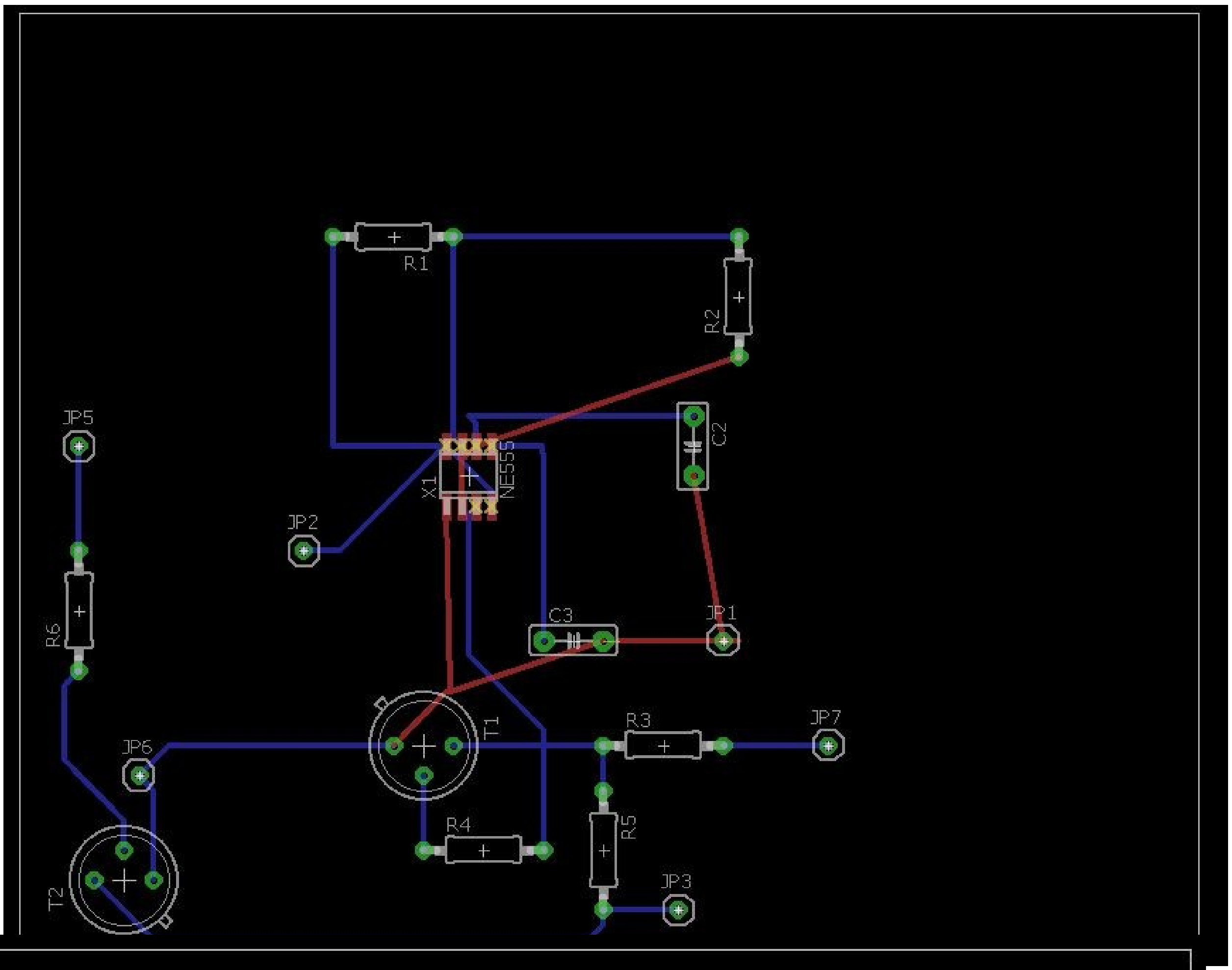


Figure 53: PCB layout - Pulse Generator

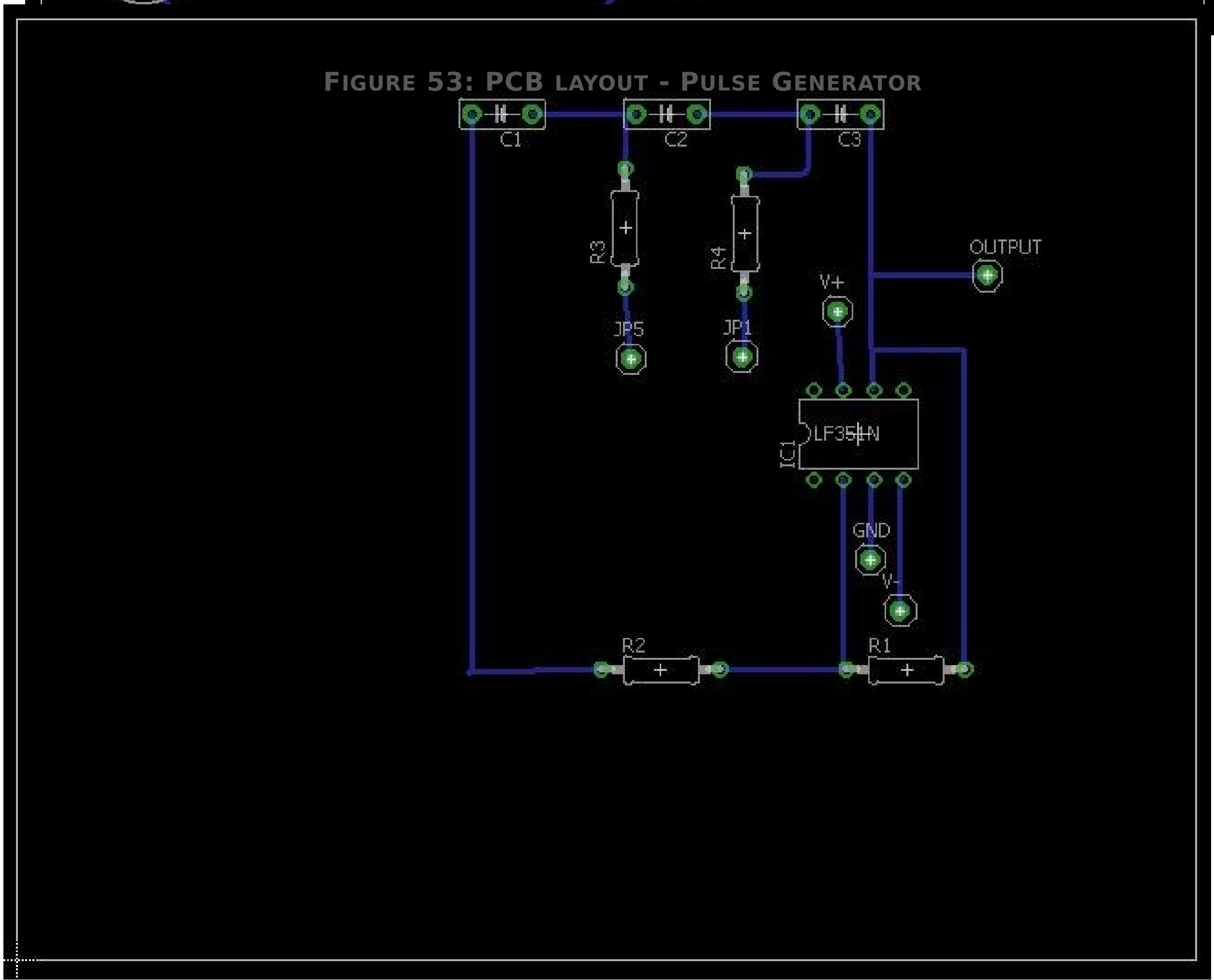


Figure 54: PCB layout - Oscillator

Appendix E

Photo of Complete Circuit

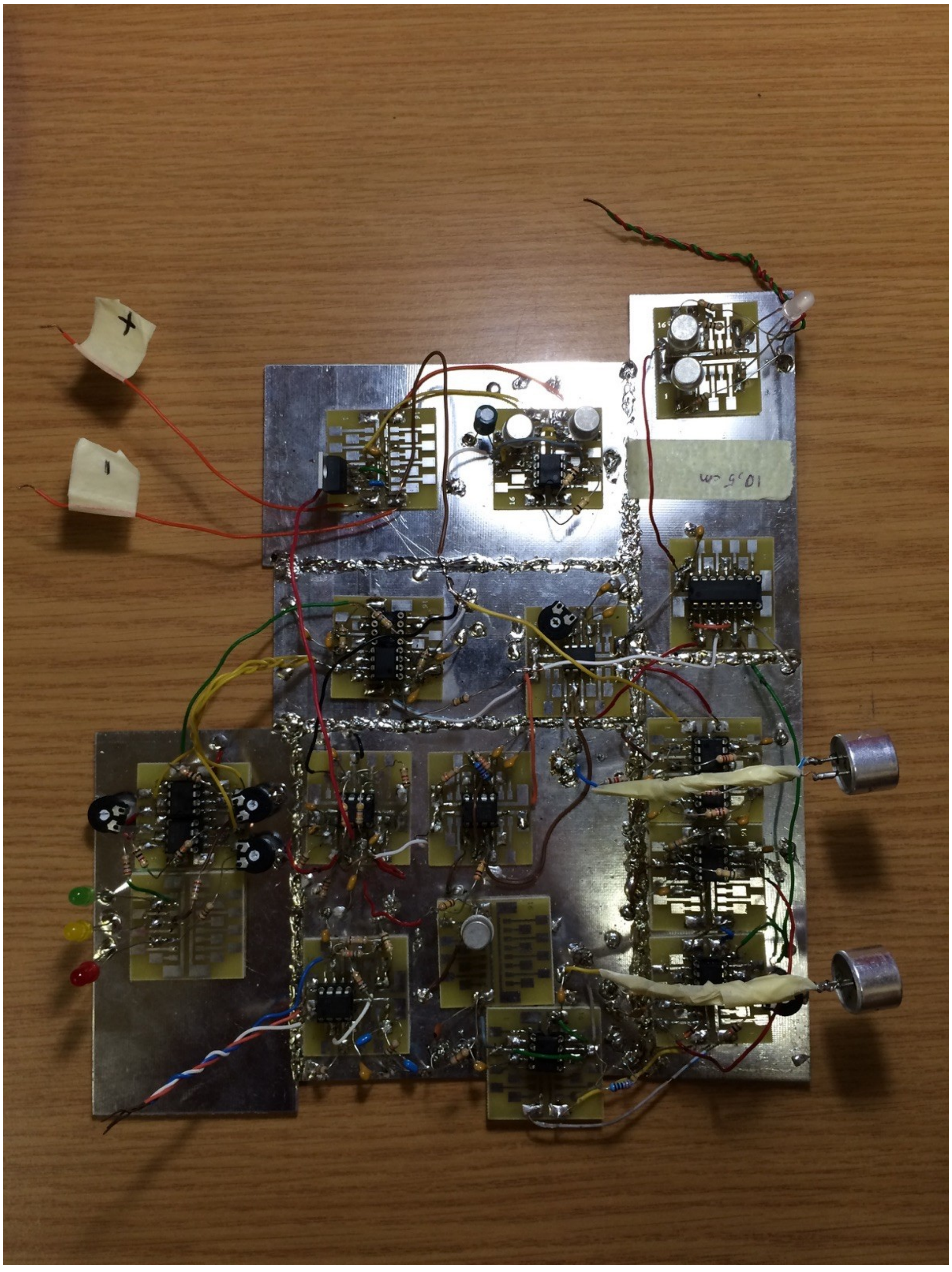


FIGURE 55: COMPLETED CIRCUIT

Appendix F

Extra work

See the Following Sections for information on all extra work:

5.2 Virtual Ground

5.12 LED circuit

Appendix G

Software

Python.py

```
#Imports
from SerialComms import SerialComms
from tkinter import *
from util import *

#Globals
tk = Tk()
com = SerialComms('COM1', 9600)

#GUI Class
class DistanceSense(Frame):
    #Attributes
    title = "Distance Sense"
    master = None

    p=5

    wSettings=None
    wConnection=None
    wStream=None

    bStream=None

    bQuit=None

    #Constructor
```

```
def __init__(self, master):
    self.master = master
    Frame.__init__(self, self.master)

    self.master.title(self.title)
    self.master.resizable(width=False, height=False)

    self.wSettings = LabelFrame(self, text="Settings", padx=self.p, pady=self.p)
    self.wSettings.grid(row=0, column=1, rowspan=2, sticky=W+E+N+S, padx=self.p, pady=self.p)

    self.wConnection = LabelFrame(self, text="Connection", padx=self.p, pady=self.p)
    self.wConnection.grid(row=0, column=0, sticky=W+E+N+S, padx=self.p, pady=self.p)

    self.wStream = LabelFrame(self, text="Stream", padx=self.p, pady=self.p)
    self.wStream.grid(row=1, column=0, sticky=W+E+N+S, padx=self.p, pady=self.p)

    self.createSettingsWidget(self.wSettings)
    self.createConnectionWidget(self.wConnection)
    self.createStreamWidget(self.wStream)

    bQuit = Button(self, text="Quit",command= master.destroy, padx=self.p, pady=self.p)
    bQuit.grid(row=2, column=1, sticky=E+S, padx=self.p, pady=self.p)

    self.pack(padx=self.p, pady=self.p)

#Create Settings Widget
def createSettingsWidget(self, widget):
    def _send(msg, lbl="Command sent"):
        updateLabel(lbl)
        com.send(msg)
        Clear_tf()

    def updateLabel(lbl="Command sent"):
        sStatus.set(lbl)
        clearStatus()

    def sendString():
        message = T.get()

        if not com.isOpen:
            updateLabel("Not connected")
        elif message in ['led.green','led.red','led.yellow','led.sensor','led.off',
                    'sensor.on','sensor.off']:
            _send(message)
        elif message == 'stream.on' :
            if(self.bStream['text'] == "Turn Stream On"):
                self.updateStream()
                updateLabel()
            else:
                _send(message)
    elif message == 'stream.off':
        if(self.bStream['text'] == "Turn Stream Off"):
            self.updateStream()
            updateLabel()
        else:
            _send(message)
    elif 'set.green' in message or 'set.red' in message or 'set.yellow' in message:
        _send(message)
    else:
        #Error
        updateLabel("Invalid command")
        pass

    Clear_tf()
    return

def Clear_tf():
    T.delete(0, END)
    T.insert(0, "")
    return

def clearStatus():
    nonlocal iCounter
    iCounter = (iCounter + 1) % 3e12

    def _clearStatus(myCounter):
        if(myCounter == iCounter):
            sStatus.set("")

    widget.after(3000, _clearStatus, iCounter)

def cal_zero():
    if com.isOpen:
        _send("calibrate.zero", "Calibarting zero")
    else:
        updateLabel("Not connected")
    return

def cal_one():
    if com.isOpen:
        _send("calibrate.one", "Calibarting one")
    else:
        updateLabel("Not connected")
    return

sStatus = StringVar("")
iCounter = 0

Calibrate_zero = Button(self.wSettings, text = "Calibrate Zero",command = cal_zero)
Calibrate_one = Button(self.wSettings, text = "Calibrate One",command = cal_one)
```

```
    C = Label(widget, text= "Enter Command:")
    T = Entry(self.wSettings)
    Send = Button(self.wSettings, text = "Send", command = sendString)
    lStatus = Label(widget, textvar=sStatus)

    Calibrate_zero.pack(fill=X, padx=self.p, pady=self.p/2)
    Calibrate_one.pack(fill=X, padx=self.p, pady=self.p/2)
    C.pack(fill=X,padx=self.p, pady=self.p/2)
    T.pack(fill=X,padx=self.p, pady=self.p/2)
    Send.pack(fill=X, padx=self.p ,pady=self.p/2)
    lStatus.pack(fill=X, padx=self.p, pady=self.p/2)
    return

#Create Connection Widget
def createConnectionWidget(self, widget):
    #Connect on button click
    def fButtonClick():
        #Setup SerialComms
        com.setBaudrate(int(eBaud.get()))
        com.setCOMPort(ePort.get())

        #Connect/Disconnect
        if com.isOpen:
            com.close()
        else:
            com.open()

    def eBaudValidate(text):
        for i in text:
            if not i.isdigit():
                return False
        return True

    def setStatusColor(c):
        lStatusMain.config(fg=c)

    lPort = Label(widget, text="Serial Port")
    lPort.grid(row=0, column=0, sticky=W, padx=self.p, pady=self.p/2)

    ePort = Entry(widget)
    ePort.insert(END, com.COMPort)
    ePort.grid(row=0, column=1, sticky=W, padx=self.p, pady=self.p/2)

    lBaud = Label(widget, text="Baudrate")
    lBaud.grid(row=1, column=0, sticky=W, padx=self.p, pady=self.p/2)

    eBaud = Entry(widget, validate='key',
              validatecommand=(widget.register(eBaudValidate), '%S') )
    eBaud.insert(END, com.baudrate)
    eBaud.grid(row=1, column=1, sticky=W, padx=self.p, pady=self.p/2)
    lStatus1 = Label(widget, text="Status")
    lStatus1.grid(row=2, column=0, sticky=W, padx=self.p, pady=self.p/2)

    fStatus = Frame(widget)
    fStatus.grid(row=2, column=1, sticky=W, padx=self.p, pady=self.p/2)

    sStatusMain = StringVar(widget, "Disconnected")
    sStatusMessage = StringVar(widget, "")
    sConnect = StringVar(widget, "Connect")

    lStatusMain = Label(fStatus, textvariable=sStatusMain, font="-size 9 -weight bold")
    lStatusMain.pack(side=LEFT)
    lStatusMessage = Label(fStatus, textvariable=sStatusMessage)
    lStatusMessage.pack(side=LEFT)

    com.registerConnectionWidget(sStatusMain, sStatusMessage, sConnect, setStatusColor)

    fConnect = Frame(widget)
    fConnect.grid(row=3, column=0, columnspan=2, padx=self.p, pady=self.p/2)

    bConnect = Button(fConnect, command=fButtonClick, textvariable=sConnect,
                      )
    bConnect.pack()

    return

#Create Stream Widget
def createStreamWidget(self, widget):
    distanceCur=0
    distancePrev=0
    velocity=0
    missingData = 0

    #Get distance measurement from Arduino
    def getDistance():
        nonlocal missingData

        recArray = com.receive()
        if len(recArray) > 0:
            distanceArduino = float(recArray[len(recArray) - 1])
            missingData = 0
        else:
            distanceArduino = -1
            missingData = missingData + 1
        return distanceArduino

    #Update the distance
    def updateDistance():
        nonlocal distancePrev
    nonlocal distanceCur

    distanceArduino = getDistance()
    if distanceArduino != -1:
        distancePrev = distanceCur
        distanceCur = calcAverage(distancePrev, distanceArduino)
        lDistanceValue['text'] = str("{0:06.2f}".format(distanceCur) + " cm")
    return

#Update the velocity
def updateVelocity():
    nonlocal velocity
    lVelocityValue['text'] = str("{0:06.2f}".format(velocity) + " km/h")
    return

#Update the measurements
def updateMeasurements(call):
    nonlocal distanceCur

    if com.isOpen and bStream['text'] == "Turn Stream Off":
        if call == 1:
            distanceArduino = getDistance()
            if distanceArduino != -1:
                distanceCur = distanceArduino
                widget.after(100, updateMeasurements, 0)
            else:
                widget.after(100, updateMeasurements, 1)
        else:
            updateDistance()
            updateVelocity()
            widget.after(100, updateMeasurements, 0)
    return

#Turn stream on or off
def updateStream():
    if com.isOpen:
        if bStream['text'] == "Turn Stream On":
            bStream['text'] = "Turn Stream Off"
            com.send('stream.on')
            widget.after(150, updateMeasurements, 1)
        else:
            com.send('stream.off')
            bStream['text'] = "Turn Stream On"
    return

widget.grid_columnconfigure(0, weight = 1)
widget.grid_columnconfigure(1, weight = 1)

lDistance = Label(widget, text= "Distance:")
lDistance.grid(row=0, column=0, sticky=W, padx=self.p, pady=self.p/2)
```

```
        lDistanceValue = Label(widget, text= str("{0:06.2f}".format(distanceCur) + " cm"))
        lDistanceValue.grid(row=0, column=1, sticky=W, padx=self.p, pady=self.p/2)

        lVelocity = Label(widget, text= "Velocity:")
        lVelocity.grid(row=1, column=0, sticky=W, padx=self.p, pady=self.p/2)

        lVelocityValue = Label(widget, text= str("{0:06.2f}".format(velocity) + " km/h"))
        lVelocityValue.grid(row=1, column=1, sticky=W, padx=self.p, pady=self.p/2)

        fStream = Frame(widget)
        fStream.grid(columnspan=2, padx=self.p, pady=self.p/2)

        bStream = Button(fStream, text= "Turn Stream On", command=updateStream)
        bStream.pack()
        self.bStream = bStream
        self.updateStream = updateStream

        return

app = DistanceSense(tk)
app.mainloop()
```

Util.py

```
def calcAverage(prev, cur):
    return 0.7 * prev + 0.3 * cur

def calcVelocity(prev, cur, deltaT):
    if deltaT > 0:
        return (cur - prev) / deltaT * 0.036
    else:
        return 0
```

Utiltests.py

```
import unittest
from util import *

class utilTests(unittest.TestCase):
    def test_average_higher(self):
        self.assertEqual(calcAverage(20,100), 44)
    def test_average_lower(self):
        self.assertEqual(calcAverage(20,10), 17)
    def test_average_negative(self):
        self.assertEqual(calcAverage(20,-20), 8)
    def test_average_same(self):
        self.assertEqual(calcAverage(20,20), 20)

    def test_velocity_positive(self):
        self.assertEqual(calcVelocity(30,40,5), 0.072)
    def test_velocity_negative(self):
        self.assertEqual(calcVelocity(40,30,5), -0.072)
    def test_velocity_zero(self):
        self.assertEqual(calcVelocity(40,40,5), 0)
    def test_velocity_time_zero(self):
        self.assertEqual(calcVelocity(40,50,0), 0)
    def test_velocity_time_negative(self):
        self.assertEqual(calcVelocity(40,50,-10), 0)

if __name__ == '__main__':
    unittest.main()
```