

Design of sound speed profiler

Water Parameter Sensor

Anwar Nazih Shaban

Master thesis



Department of Physics

Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

April / 2017

© Anwar Nazih Shaban

2017

Design Sound Speed profiler (WPS)

Department of physics, University of Oslo

<http://www.duo.uio.no/>

Trykk: Reprosentralen, Universitetet i Oslo

Acknowledgements

I can now say at this journey is coming to reach the last station. It was a long, exhausting and difficult journey. But it was also exciting and rich of knowledge. I have to admit that I was not lucky to choose the most suitable thesis for my abilities. Many things were very new to me and I hadn't any experience with the most parts of this work. Therefore I used more time than expected trying to teach myself many things step by step. This does not mean that I am regret, at the contrary, I had the chance to learn many useful things and do very exciting work.

This work would not be possible without the support and help of many people who was beside me and helped in every possible ways.

First of all, Praise be to God, Lord of the worlds, for guidance, inspiration, patience and success.

I am grateful for my supervisor Helge Balk for his support and encouragement. I would like to thank Ketil Røed and Professor Ørjan Martinsen for finding time answering my questions and helping me.

Thanks to Stein Nielsen and his crew at the Electronics lab, especially Erlend Bårdsen who was more than helpful and patient to answer my questions, provide useful suggestions and help in PCB production. I'm very grateful for you Erlend, I really learned very much from you.

I should not forget to thank my colleagues Asbjørn Vinje and Bendik Søvegjarto who didn't hesitate to help solving and discussing some problems or coming with helpful suggestions.

To my Parents, my grandparents, my family and my friends everywhere many thanks for your support, prayers, faith of me and warm thoughts that I really needed in this period.

My great father, Dr. Nazih Shaban, There is no word can describe my grateful for your love, encouragement and support. You were always worried about me and how I'm doing. I really appreciate that and I hope I can make you proud of me as much as I'm proud to be your daughter.

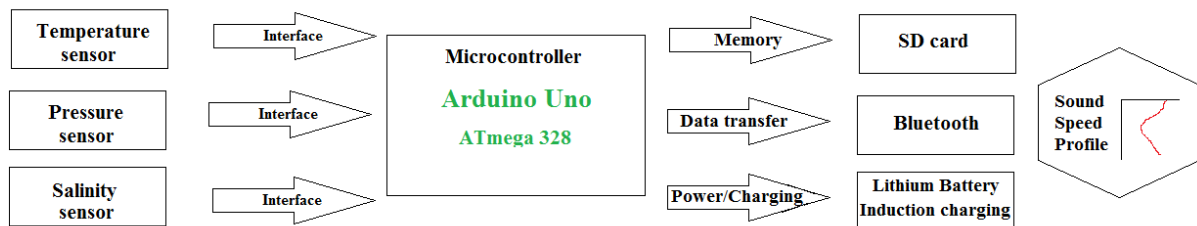
At last but not least, my sincerely and deepest thanks to the sunshine of my life my lovely soulful husband Mahmoud Shaban and to my wonderful, amazing daughters Limar and Basmala. It was very hard period for all of us. There is no word can describe my thanks for your patience and love. My dear husband, you was always there supporting me with your endless love and your warm prayer, encouraging me, and helping me in all the possible ways. I appreciate your standing with me every station of this long journey and always raising my spirits up and overwhelm me with your love. You are a fabulous father and a gorgeous husband. We are lucky of you. Nothing could be managed without you Thank you.

ABSTRACT

This thesis describes designing and building of a low cost, wireless Water Profile Sensor (WPS). Development of the WPS is part of the Autonomous Mobile Hydroacoustic Assessment project (AMHA) run by the Sonar group at the department of Physics. The idea is that an automatic survey vessel should be able to stop and launce the sensor automatically to acquire water profile data. The data is needed by the echosounder to correct for refraction and absorption of the sound beam along the survey. The WPS has been designed to measure temperature, pressure and conductivity at intervals of 1 sec on the way down through the waterbody. When deployed, the unit will start to measure and save data to a flash memory card. After each run, the data will be sent to the main computer on board the vessel through Bluetooth. Between each deployment the sensor will be charged inductively.

The current construction uses three different sensors, DS18B20 for temperature, MS5803 for depth and one for conductivity. The conductivity sensor has been designed from scratch while the pressure and temp sensors are on the shelf sensors.

An Arduino Uno development board is applied to control the whole WPS and to take care of its individual tasks.



Continents

- 1.INTRODUCTION..... 10**

- 2.BACKGROUND AND RELATED WORK 11**

- 3. PROPERTIES OF THE SEAWATER 15**
 - 3.1 SALINITY 16
 - 3.2 TEMPERATURE 16
 - 3.3 PRESSURE..... 17
 - 3.4 SOUND SPEED PROFILE IN THE SEA WATER [24, P.117]..... 18
 - 3.5 SNELL’S LAW..... 19

- 4.ELECTRICAL CONDUCTIVITY (EC) SENSOR:..... 21**
 - 4.1 INTRODUCTION..... 21
 - 4.2 CONDUCTIVE SOLUTIONS: 21
 - 4.3 CONDUCTIVITY AND TEMPERATURE: 23
 - 4.5 TEMPERATURE COMPENSATION: 24
 - 4.6 MEASURING THE ELECTRICAL CONDUCTIVITY 25
 - 4.7 4-ELECTRODES CONDUCTIVITY CELL: 28
 - 4.8 SALINITY..... 28
 - 4.9 TOTAL DISSOLVED SOLID (TDS): 28
 - 4.10 DESIGNING 4-ELECTRODE EC CONDUCTIVITY METER:..... 29
 - 4.11 PERIPHERAL CIRCUIT: 30
 - 4.12 MEASURING CONDUCTIVITY EXPERIMENT: 33
 - 4.12.1 MATERIALS AND TOOLS USED IN THE EXPERIMENT: 33
 - 4.12.2 PROBE DESIGN AND BUILDING: 34
 - 4.12.3 DIFFERENTIAL AMPLIFIER: 36
 - 4.12.4 CURRENT TO VOLTAGE AMPLIFIER: 38
 - 4.12.5 TESTING THE PROBES 39
 - 4.12.6 TESTING THE PROBES WITH THE PERIPHERAL CIRCUITS: 40
 - 4.12.7 GENERATING SINE WAVE INPUT SIGNAL FROM THE ARDUINO UNO: 43
 - 4.12.8 TESTING THE CIRCUIT: 48
 - 4.13 SUMMERY: 50

- 5. WATER PARAMETERS SENSORS 51**
 - 5.1 TEMPERATURE SENSORS: 51
 - 5.1.1 THERMOCOUPLE: 51
 - 5.1.2 (RTD)RESISTIVE TEMPERATURE DETECTOR 51
 - 5.1.3 THERMISTORS 52
 - 5.1.4 SILICON JUNCTION TEMPERATURE SENSORS (PN JUNCTION TEMPERATURE SENSORS)..... 52
 - 5.1.5 DS18B20: 1-WIRE DIGITAL TEMPERATURE SENSOR [44] 53
 - 5.1.6 ONE-WIRE INTERFACE 55

5.1.7 DS18B20 WITH ARDUINO UNO TO MAKE A THERMOMETER.....	58
5.2 PRESSURE.....	59
5.2.1 BACKGROUND	59
5.2.2 PRESSURE SENSORS [42]	60
5.2.2.1 Piezoresistive effect and strain sensitivity	60
5.2.2.2 Piezoresistive Pressure sensors.....	61
5.2.3 MS5805-14BA PRESSURE SENSOR [55]	62
5.2.4 I ² C INTERFACE:	64
5.2.5 MS5803 WITH ARDUINO UNO TO MEASURE THE PRESSURE AND THE DEPTH UNDERWATER.....	67
5.3 SALINITY SENSORS	68
5.4 MEMORY:	69
<u>6. WIRELESS DATA TRANSFER:.....</u>	<u>70</u>
6.1 BLUETOOTH.....	70
6.2 HOW BLUETOOTH WORKS?	71
6.3 TESTING THE BLUETOOTH BLUESMIRF MODULE:	73
6.4 SOFTWARE	74
<u>7 SYSTEM DESIGN</u>	<u>75</u>
7.1 PCB:	75
7.2 THE PRODUCTION OF PCB BOARD IN ELAB:	76
7.4 POWER.....	78
7.4.1 HOW TO POWER THE PROJECT?	78
7.5 WIRELESS INDUCTION CHARGING:.....	82
7.5.1 SIMPLE WIRELESS POWER EXPERIMENT	82
7.5.2 WIRELESS CHARGER	85
7.5.3 TESTING THE INDUCTIVE CHARGER:	87
7.6 PACKAGING TO GET WATERPROOF INSTRUMENT	88
7.8 IMPLEMENTATION OF THE ALGORITHM'S	92
<u>8. TESTING THE WPS</u>	<u>94</u>
TEST OF THE DEPTH SENSOR	94
TESTING THE TEMPERATURE PROFILER.....	95
<u>9. SUMMERY AND CONCLUSIONS</u>	<u>100</u>
SUGGESTED IMPROVEMENTS AND FUTURE WORK	ERROR! BOOKMARK NOT DEFINED.
<u>10. APPENDIX.....</u>	<u>102</u>
10.1 ARDUINO UNO SOURCE CODE	102
10.2 REFERENCES	119

1.INTRODUCTION

Sound travels through the seawater better than the other forms of radiation. Thus, the underwater sound has many applications in the exploration and studying of the sea. The study and the application of sound in the water known as Hydroacoustics. The applications of the underwater sound have its origin deep in the past, it may return to before the World War I and later huge progress was made in the instruments and devices.

Hydroacoustics methods are frequently applied for monitoring fish, plankton, and aquatic vegetation in water bodies. The method is fast, cheap and invasive compared with alternative methods based on sampling. The monitoring is commonly carried out with one or more echo sounders or sonars mounted on a vessel that cruises the water body according to a defined survey plan.

The hydroacoustic method needs additional key parameters from the water body such as the absorption coefficient and the sound speed. These parameters vary with other parameters such as salinity, temperature, depth and Ph.

At the sea where vertical applications dominate, it is currently considered to be sufficient with average measurements for the absorption and sound speed.

For shallow lakes, however, horizontal beaming is needed to cover the. In horizontal sound propagation, the vertical distribution of the sound speed is important. Horizontal beaming is strongly influenced by refraction caused by the sound speed profile. In practice the refraction phenomena leads to misinterpretation of the estimates from the echo sounder. A common error is that the sound beam will bend down into deeper water and fish deep down will be observed as fish in the surface layer. To avoid this, one need to know the sound speed profile. An accurate information about the sound speed in the water as a function of temperature, pressure and salinity is important, because resulting refraction may operate either to channel the sound energy and thus, increase the transmission range or to send this energy into the ocean depth. This motivates the building of sound speed profiler.

Sound speed profilers are well known and can be bought as on the shelf equipment. Most of the profilers are, however, too expensive for many inland research institutes, too large for many boats used on inland water bodies. This thesis presents a project of developing of a low-cost, wireless, multi-sensor system that can be applied, to collect the sound speed profile along the vessels survey and easily transfer the data to the echo sounder system.

The project system consists of three sensors: temperature sensor (DS18B20 1-wire digital thermometer), MS5803-14BA high-resolution pressure sensors with SPI and I2C bus interface, and 4-electrode conductivity sensor was designed and developed for this project. The conductivity gives an indication of the amount of salt in the water, the more the salt, the higher the conductivity. All the data from the sensors are processed and analysed by Arduino Uno, and transmitted by a Bluetooth to a receiver.

Section 2 reviews the related works that were considered important to this project.

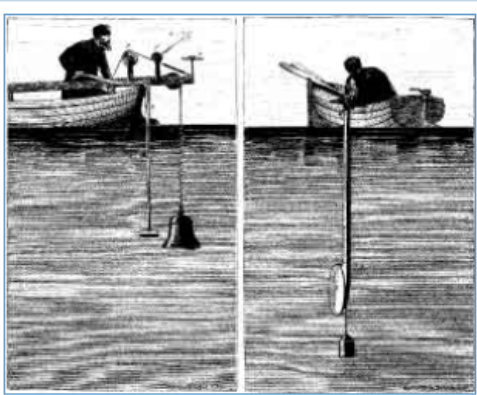
2. Background and related work

The subject of sound is one of the important sections of physics. The researchers have been investigated many phases of the sound and used it in many applications in different branches. An especial example of this is the underwater sound. The sound in water travels faster and with less energy absorption than in the air. The ease of transmission of the sound through the seawater allows the researchers of using the underwater sound in exploring the sea, and developing a wide range of submarine acoustic applications with benefit for navigation.

In many scientific applications, the accuracy of the sound speed is required. The speed of sound propagation through the water or any medium is not constant. It varies by a small amount affected by the density and the elasticity. As in Newton equation for the speed of sound in a given medium [3]:

$$\text{Sound speed (c)} = \sqrt{\frac{\text{elasticity of the medium}}{\text{density of the medium}}} \quad (2.1)$$

The elasticity and the density of the medium are influenced by pressure, temperature and salinity. Cold water is denser than warm water; salty water is denser than fresh water; pressure increases the density of water. This leads to at the sound speed also is influenced by these parameters. The sound speed in seawater varies by a few percent from place to place, season-to-season, morning to evening, and with water depth. Although the variations in the speed of sound are not large, they have important effects on how sound travels in the seawater.



In 1826 on Lake Geneva, Switzerland, Jean-Daniel Colladon, a physicist, and Charles-Francois Sturm, a mathematician, made the first recorded attempt to determine the speed of sound in water. In their experiment, the underwater bell was struck simultaneously with ignition of gunpowder on the first boat. The sound of the bell and flash from the gunpowder were observed 10 miles away on the second boat. The time between the gunpowder flash and the sound reaching the second boat was used to calculate the speed of sound in water. Colladon and Sturm were able to determine the speed of sound in water fairly accurately with this method. J. D. Colladon, Souvenirs et Memoires, Albert-Schuchardt, Geneva, 1893.

Figure 1: measuring the speed of sound in 1826 [19]

The history of efforts to measure the underwater sound speed has its origin deep in the past. In 1827 a Swiss Physicist, Daniel Colladon, and a French mathematician, Charles Sturm, worked together to measure the sound velocity in Lake Geneva in Switzerland. They struck a submerged bell in the lake and simultaneously set off a charge of powder in the air, as quoted by Urik [1, p.2, 113]. By timing the interval between a flash of light and the striking of a bell underwater across the lake. They obtained the velocity of sound in water. The result was 1435m/s at 8.1°C with a surprisingly degree of accuracy.

Later the interest of underwater sound was not allowed to die, Subsequent of investigators worked to determine the sound speed by different ways and experiments. The United States Army had a series of investigations to determine the velocity of the sound in seawater. They achieved very accurate determination of the speed of sound along the surface in certain locations. The physicist, E.B Stephenson was associated in

this work and had discussed the experiments and the results in his paper [2]. The experiments were done in January 1922 by measuring the time required for the sound produced by the explosion of 5.kg bomb of TNT to travel through the water ,of Block Island Sound, New York, a known distance of 15500m to each of the five hydrophones established at widely separated points. The result was found to be 1453.3 m/s at -0.3°C and salinity of 33.4 ppt (parts per thousand = g/L).

The British Navy during the same period has been at work on acoustic methods for estimating the depth of the water and determining the sound velocity. They could obtain the velocity of sound along the surface. An experimental formula based on their result showed the velocity as a function of temperature and salinity of the water.

Many other statements have studied the velocity of the sound along the surface and made on the basis of the latest available published information, like the French hydrographic office and the German hydrographic office [3].

From November 17 to December 29, 1923 the Coast and Geodetic Survey steamer Guide was engaged on the oceanographic cruise. In this tour of investigations, they could obtain the velocities of sound by simultaneous depth and time determination and a rational basis for applying theoretical velocities had been developed. They could found that the velocity increased with the depth although the temperature decreased and the salinity was almost the same. This fact suggested that the sound velocity is a function of not only the temperature and the salinity but also of the pressure. Investigations based on reliable theoretical ground, was begun to find the relation between the velocity, temperature, salinity and pressure [3].

Many other investigations was done later to find the sound velocity not only by timing the arrival of sound, but also tried to utilize the water parameters like salinity, temperature and depth. A relationship between the sound speed and the water parameters can be found based on either theory method, using certain basic properties of water such as its specific heat or of specific volume, or by laboratory measurements of sound speed over a range of temperatures, salinities and pressures.

Tables of sound speed, based on theoretical method, were prepared by Matthews [4], Heck and Service [3], and Kuwahara [5]. For about 20 years, the tables of Kuwahara are presented as the standard tables of sound speed, and used by the Naval Research Laboratory (NRL). Kuwahara's tables shows, for example, that the sound speed in surface water of salinity 35ppt at 30°C is 1543 m/s, while it is 1510 m/s for distilled water. At the same conditions, the speed increases about 1 m/s for each ppt increase, and 2.2 m/s for each degree increase of temperature, and 1.8 m/s for the increase of pressure caused by 100 m of depth [5-6].

These theoretical tables use compressibility and determination made over forty years ago and assumes the temperature dependence of specific heat for distilled water to the same as for sea water [7]. The tables based on the following equation [6]:

$$c = \sqrt{\frac{\gamma}{\rho k}} \quad (2.2)$$

c : The velocity of sound.

ρ : The density.

k : The isothermal compressibility

γ : The ratio of specific heat.

The more modern experimental method based on the laboratory techniques for direct measurements of the sound velocity at a different water parameters under certain conditions. For examples, the measurement of Wiessler and Del Grosso [6], Del Grosso[7], and Wilson [8].

Wiessler and Grosso have made laboratory measurements in sea water samples from Caribbean and Middle Atlantic by means of a three-megacycle ultrasonic interferometer. For each sample, they determined the density, sound velocity and the concentration of chloride at 20⁰ and 30°C. They compared there results with Kuwahara's. There results at various salinity and temperature were 3-4 m/s lower than Kuwahara's results. They also found that the variation of dissolved air content have a negligible effect on the speed of sound and absorption [6].

Wilson [8] prepared tables of sound speed in sea water based on an empirical formula of sound speed as a function of temperature, Pressure and salinity. The formula was derived to fit the resulted data of measuring sound speed that he obtained over: the temperature range - 3°C to 30°C, the pressure range 1.033kg/cm² to 1000 kg/cm², and the salinity range 33ppt to 37 ppt. At those ranges, 581 sound speeds were measured and this information was used to found the coefficients of equation (2.3):

$$V \text{ (sound speed)} = 1449.22 + \Delta V_T + \Delta V_P + \Delta V_S + \Delta V_{STP}, \quad (2.3)$$

Where:

$$\Delta V_T = 4.6233T - 5.4585 \times 10^{-2}T^2 + 2.822 \times 10^{-4}T^3 - 5.07 \times 10^{-7}T^4,$$

$$\Delta V_P = 1.60518 \times 10^{-1}P + 1.0279 \times 10^{-5}P^2 + 3.451 \times 10^{-9}P^3 - 3.503 \times 10^{-12}P^4,$$

$$\Delta V_S = 1.391(S-35) - 7.8 \times 10^{-2}(S - 35)^2,$$

$$\Delta V_{STP} = (S-35)(-1.197 \times 10^{-2}T + 2.61 \times 10^{-4}P - 1.96 \times 10^{-7}P^2 - 2.09 \times 10^{-6}PT) + P(-2.796 \times 10^{-4}T + 1.3302 \times 10^{-5}T^2 - 6.644 \times 10^{-8}T^3) + P^2(-2.391 \times 10^{-7}T + 9.286 \times 10^{-10}T^2) - 1.745 \times 10^{-10}P^3T.$$

Both theoretical and experimental methods obtained the sound speed in terms of the three basic quantities: temperature, salinity, and pressure (depth). However, the dependence of velocity on these physical parameters is by no means a simple one. An empirical relationship giving the sound speed, as a function of those three parameters is very complicated and suitable for computer programming. Del Grosso [12] gives a complicated and long equation of 19 terms. Chen and Millero [13] gave another example of such complicated formula. There model is endorsed by the international standard algorithm, often known as the UNESCO algorithm [14] and used as the standardized reference model.

Some simpler expressions may be enough for practical work, when it is not very important to get about 0.5 m/s error. Such expressions have appeared in the literature and listed in Table 1.

Table1: Expressions for sound speed [m/s] in term of Temperature, salinity, and Depth. Urik [1, p.113]

Expression	Limits	Reference
$c = 1492.9 + 3(T-10) - 6 \times 10^{-3}(T - 10)^2 - 4 \times 10^{-2}(T - 18)^2 - 1.2(S-35) - 10^{-2}(T - 18)(S-35) + D/61$	$-2^\circ \leq T \leq 24.5^\circ$ · Temperature [°C] $30 \leq S \leq 42$, Salinity [ppt] $0 \leq D \leq 1,000$ depth [m]	Leroy
$c = 1449.2 + 4.6T + 5.5 \times 10^{-2}T^2 - 2.9 \times 10^{-4}T^3 + (1.34 - 10^{-2}T)(S-35) + 1.6 \times 10^{-2}D$	$0^\circ \leq T \leq 35^\circ$ $0 \leq S \leq 45$ $0 \leq D \leq 1,000$	Medwin
$c = 14498.96 + 4.519T - 5.304 \times 10^{-2}T^2 + 2.374 \times 10^{-4}T^3 + 1.340(S-35) + 1.630 \times 10^{-2}D + 1.675 \times 10^{-7}D^2 - 1.025 \times 10^{-2}T(S - 35) - 7.139 \times 10^{-13}TD^3$	$0^\circ \leq T \leq 30^\circ$ $30 \leq S \leq 40$ $0 \leq D \leq 8,000$	Mackenzie

c: is the speed of sound in water as a function of temperature(T), salinity (S) and depth(D). The speed of the sound in seawater is proportional with T, S and D.

The **Expendable Bathythermograph (XBT)** and the **velocimeter** are two device commonly used for finding the velocity of sound in terms of depth in the sea. The velocimeter measures sound speed directly in terms of the travel time of sound over constant fixed path. Many modern submarines are often equipped with velocimeters. The XBT is the most widely used tools. It may be lunched from submarines, surface ships and even aircrafts. The device sinks at a known rate into the sea and measures temperature as a function of depth. Neglecting the salinity, the sound speed profile can be estimated in terms of temperature and pressure, which has the most effect on the speed of sound in seawater. It is simple and nowadays, expendable.

Nowadays there is a wide range of **oceanographic instruments** available to measure the speed of sound in seawater with high accuracy. For example, STD/CTD sound speed profiler, model SD204, from SAIV A/S [15] environmental sensors and systems. This instrument measures, calculates and records seawater conductivity, salinity, temperature, depth (pressure), sound velocity and water density.



Figure 2 : STD/CTD model SD 204 sound speed profiler [15].

Valeport [16], one of the UK's leading manufacturers of Hydrographic and Oceanographic Equipment, produce a huge number of sound speed profiler and sensors with high accuracy.

For example, **MIDAS SVX2 Combined CTD/SVP** a great hybrid device offering the Superior Sound Velocity Data combined with leading edge Salinity and Density measurement, this unit offers the best of both worlds. It also has a 0.01% pressure sensor as standard and utilizes synchronized sampling to ensure perfect profiles.



Figure 3: MIDAS SVX2 combined CTD/SVP (the latest version of Valeport's unique instrument)[16]

These instruments are somewhat expensive and not everyone can obtain them. The rest of the thesis will explain the work have been done to design a simple non-expensive sound speed profiler.

3. Properties of the seawater

The temperature and the salinity are two of the most important characteristics of the seawater. They together control the density of the seawater, which is an important factor that the sound speed in the seawater depends on. The mean density of seawater is, $1.035 \times 10^3 \text{ kg/m}^3$. This density varies a little, by just a few percent, but these variations have noteworthy effects. Density depends on temperature T , salinity S , and pressure p . Cold water is denser than warm

water; salty water is denser than fresh water; pressure increases the density of water. In general, temperature usually decreases with depth, salinity can either increase or decrease with depth, and pressure always increases with depth. The range of temperature and salinity in the open ocean is 0-30°C and 33-36 ‰. Thus, the density is more influenced by the temperature than the salinity and so do the speed of sound [18].

3.1 Salinity

Salinity is a measure of the amount of salt dissolved in the water. Table 2 shows the major salts of the seawater. These salts, tabulated in table2, come largely from weathering of continents, which is very slow input compared to the mixing rate of the whole ocean. Thus, there is no much variation of salinity from place to place. The most variations of the salinity occurs at the surface of the seawater. The evaporation of the seawater causes increasing in the salinity of the sea surface while the rainfall decrease the salinity, because of the change in the mix of pure water and dissolved salts..-

Table 2: Major constituent of seawater [18].

Salinity	‰ ppt (g/kg)
Chloride	18.98
Sodium	10.56
Sulphate	2.65
Magnesium	1.27
Calcium	0.40
Potassium	0.38
Bicarbonate	0.14
Others	0.11
Overall salinity	34.48

Modern salinity measurements are dimensionless; the Practical Salinity Scale defines salinity in terms of a conductivity ratio very nearly equal to g/kg or, equivalently, ‰ (ppt, parts per thousand). Seawater has typically a salinity of 34.5 psu (practical salinity units). This means that 1 Kg of seawater contains 34.5 g of dissolved salts, although salinity can vary locally based on hydrological conditions. Closed seas have a greater difference in their salinity influenced by for example, evaporation or fresh water inputs from precipitation or rivers inflow or melting ice. Salinity varies very little with depth; it is almost constant salinity in the deep water.

3.2 Temperature

The seawater temperature typically decreases with depth, but there are many local variations depend on many conditions. The most variations with time and depth occurs in the shallow layers that are exposed the most to the solar heating, water currents, seasonal variation , the

time of the day, and surface mixing. At deep water, the temperature becomes constant with depth of about 2 - 4°C. (See. Fig.4).

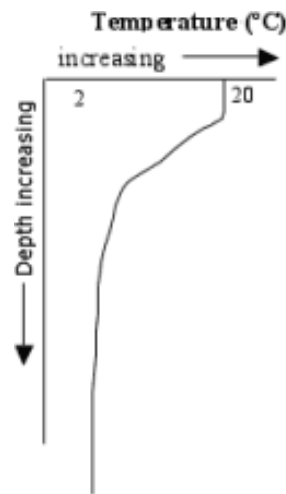
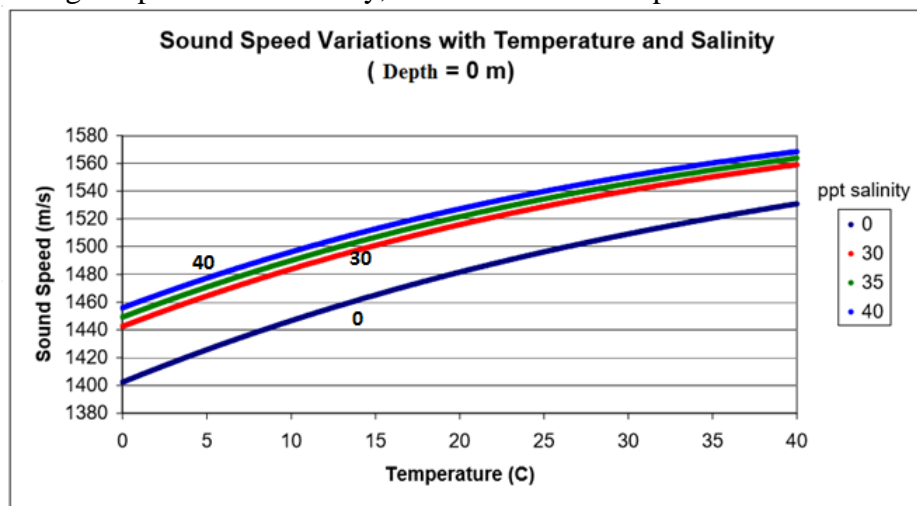


Figure 4 Temperature profile. [19]

The sound speed in the seawater is proportional with temperature and salinity. As shown in Fig.5, increasing temperature or salinity, increase the sound speed.



(From: Lurton, X. An Introduction to Underwater Acoustics, 1st ed. London, Praxis Publishing LTD, 2002, p37)

Figure 5: Sound speed variation with temperature and salinity at depth 0 m. [2]

3.3 Pressure

The seawater pressure increases by large amount with depth according to the following equation:

$$P = P_o + \rho gh \tag{3.4}$$

P = pressure in seawater (N/m², Pa).

P_o = atmosphere pressure (1.013 x 10⁵ pa).

ρ = density of seawater (Kg/m³).

g = is the acceleration of gravity (9.8 m/s²).

h = the depth of the seawater.

The increasing of pressure will increase the speed of the sound in the seawater, especially at the deep water where the temperature and the salinity are almost constant.

3.4 Sound speed profile in the sea water [24, p.117]

Sound speed profile is the variation of sound speed with depth. It is difficult to measure the locally sound speed, but easy to measure the parameters that depends on. The devices that described previously can be used to found the speed profile in the deep sea or by hydrographic observation of salinity, temperature and depth. The speed of sound in water increases with increasing of temperature, salinity and pressure. Knowing the behaviours of each of these properties, as they was discussed previously, helps in finding the sound profile in seawater. Fig.6 shows sound speed profiler in deep sea divided into several layers that have different conditions and characteristics depending in the variety of seawater properties.

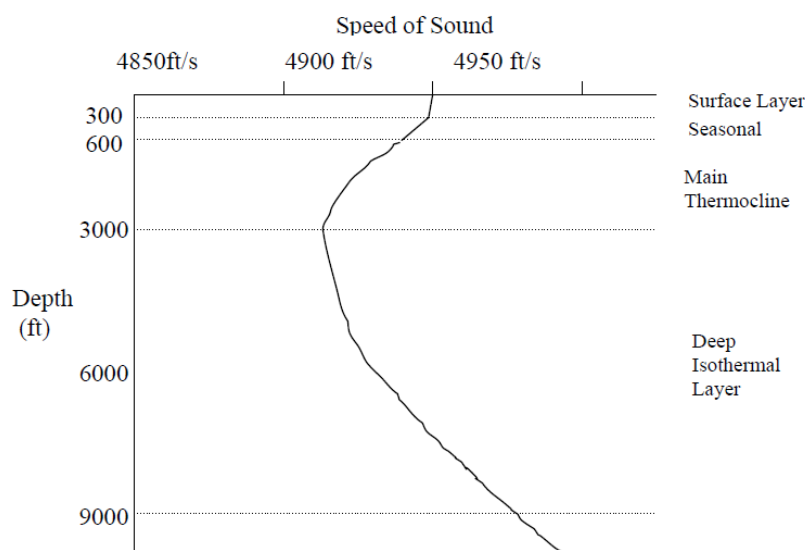


Figure 6: Deep-sea sound velocity profile divided into layers. [17]

The layers are:

- 1) Surface layer: It is the most variable layer. The sound speed is liable to local change of cooling, heating and wind actions.
- 2) Seasonal thermocline: the temperature or sound speed decrease with depth and varies with the seasons, time of day, current, etc.
- 3) The main thermocline: affected slightly with the seasonal changes. Negative sound speed gradient effected by the decreasing of temperature.
- 4) Deep isothermal layer: nearly constant temperature $\approx 2 - 4^{\circ}\text{C}$. The speed of sound increase with depth due to increasing of pressure.

The surface and seasonal layers are the most interesting because they vary the most. During the summer, there is a sharp negative gradient of sound speed. This is because the temperature is high from the sun warming of the upper layers of the sea. While in the winter, the cooled air keeps the surface water cold. The warmer water from below rises up to the colder in the surface leading to more mixing in the upper layers. Moreover, the strong winter storms and large waves create more mixing of the surface layer. As the depth increase, the temperature decrease until it reaches an almost constant value. Since

the temperature is constant, the pressure of water has large influence on the sound speed by increasing with depth.

Between the negative speed gradient at layer 3 and the positive gradient at layer 4 there is a minimum speed toward which sound travelling at great depths tend to be bent or focused by refraction.

The salinity have a smaller effect on sound speed than temperature and pressure because the salinity varies slightly especially on the open sea. However, in the places where the salinity varies strongly like in the rivers inflow or near shore the influence on the speed can be greater. In the shallow water, the velocity profile is irregular and unpredictable. It is very sensitive to the surface heating and cooling, salinity change and water current.

As we saw before the speed sound as a function of T, S and P, is very complicated. The approximate of the dependence of the speed of sound on each property is [21]:

Temperature (T) increases $1^{\circ}\text{C} = 4.0 \text{ m/s}$ increase in speed.

Salinity(S) increases $1\text{PSU} = 1.3 \text{ m/s}$ increase in speed.

Depth (pressure) increases $100\text{m} = 1.7 \text{ m/s}$ increase in speed.

3.5 Snell's law

We can use the same techniques that are used in tracing of the reflection and refraction of the light rays, to predict the path of sound in seawater. Snell's law describe the relationship between the angles of incident and refraction sound waves passing through the boundary of two different mediums or variable velocity medium. Here we look at grazing angles θ , referenced to the horizontal and used when looking at refraction, (see Fig. 7). When the ray cross the boundary between two different media, the ray is refracted. The angle of refraction depends on ratio of the speed of the in the two media. This is valid for sound ray propagation as for the light ray. This is Snell's law.

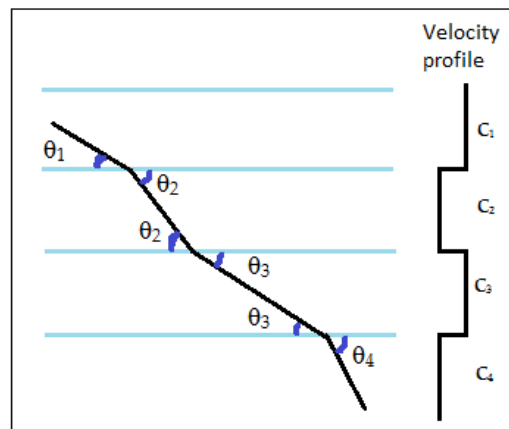


Figure 7: Refraction of the sound ray in layered medium

Snell's law states that in a medium of constant velocity layers, the grazing angles $\theta_1, \theta_2, \dots$ of a ray at the layer boundaries are related to the sound velocity c_1, c_2, \dots of the layers by:

$$\frac{\cos \theta_1}{c_1} = \frac{\cos \theta_2}{c_2} = \frac{\cos \theta_3}{c_3} = \dots = a \text{ constant for any one ray} \quad (3.5)$$

From equation if $c_1 > c_2 > c_3 \dots$, then $\theta_1 < \theta_2 < \theta_3 \dots$. This means that the ray travelling from high speed to lower speed layer, the ray angle will be larger and bends toward the layer of lower sound speed away from the horizontal. When the sound speed increases, the ray angle tends to decrease.

Snell's law may be restated for acoustic rays: "sound rays passing from one medium to another bend toward the medium with the slower wave speed" [22].

4. Electrical Conductivity (EC) sensor:

4.1 Introduction

Electrical conductivity is the reciprocal of the electrical resistivity, and it is an intrinsic property of a material revealing its ability to conduct an electric current, the materials with this property are ranging from materials with a very high conductivity like metals to very non-conductive materials like plastics or glass. The aqueous solutions, such as seawater, are about halfway between the two extremes in conductivity.

$$\sigma = \frac{1}{\rho} = n * e * \mu \quad (4.1)$$

σ : The electrical conductivity [S/m]

n : number of “free” or conduction electrons per unit volume

μ : The mobility: a measure of how easily the electron moves in response to an electric field [$\text{m}^2\text{V}^{-1}\text{s}^{-1}$]

e : electron charge [V].

Electrons in metals like silver and copper carry the electrical current, while it is carried by charged ions in liquids and solutions like water. In both cases, as shown in equation (4.1), the electrical conductivity depends on the number of charge carriers, the mobility of them, the amount of charge each one carries and the material temperature. Thus, the higher the concentration of dissolved salts in water, which will lead to more ions in the solution, the higher the conductivity of the solution. Therefore the conductivity can indicate the amount of substances dissolved in the water and so for example, the salinity and the density of the seawater can be derived from its conductivity. Metals are extremely conductive because electrons move almost with the speed of light, while the conductivity of the water is much lower, that the ions move much slower

The SI units of the conductivity is Siemens per metre(S/m) which is identical to the older unit of mho/m. Conductivity is usually expressed in the unit of micro Siemens/cm ($\mu\text{S}/\text{cm}$) or milliSiemens/cm (mS/cm) [25].

4.2 Conductive solutions:

The conductivity is measured primarily in aqueous solutions of electrolytes. The electrolytes are compounds contain mobile ions that conduct the electric current when the compound is in an aqueous solution or melted. The ions are charged particles formed by the dissociation of acid, basis, salts and certain gases such as carbon dioxide, hydrogen chloride and ammonia. The Ions conduct electricity due to their positive and negative charges. When electrolytes dissolve in water, they split into positively charged (cation) and negatively charged (anion) particles as shown in Fig. 1. Positive ions migrate to negative electrode (cathode), while negative ions move to positive electrode (anode) and conduct current. Pure water itself can conduct the electricity because it dissociates to form Hydrogen ions (H^+) and Hydroxyl ions (OH^-).

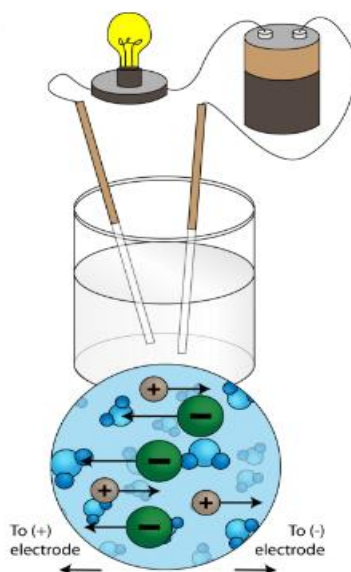


Figure 1: A water solution of an ionic compound conducts electricity well. The positive and negative ions make up the basis of conductivity in water.[29]

The conductivity of the solution depends on the concentration of ions in the liquid. The conductivity increases with increasing density of ions until a certain ion concentration. With further increase the ions movement will be restricted and the conductivity starts to drop off. This gives room for ambiguity in conductivity measurements. This can result in two different concentrations of a salt having the same conductivity, as we can see in Table 1.

Table 3: Conductivity values of typical samples at 25°C

Sample at 25°C	Conductivity $\mu\text{S}/\text{cm}$
Ultrapure water	0.055
Distilled water	1.0
Drinking water	50.0
Seawater	50000
5% NaOH	223000
50%NaOH	150000
10% HCl	700000
32% HCl	700000
31% HNO ₃ (Highest known)	865000

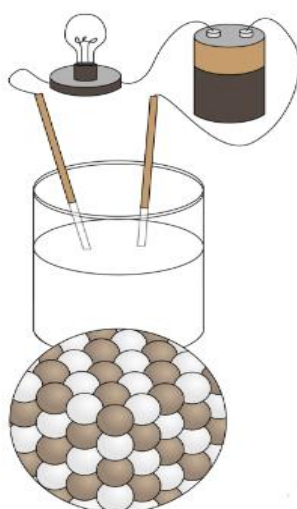


Figure 2: Positive and negative ions fixed in a solid don't conduct a current [29]

Some aqueous solutions are more conductive than the other depending on their degree of ionization in water. Each acid, base, or salt has its own characteristic curve for the concentration vs. conductivity.

Distilled or deionized water have very low conductivity value. Seawater, on the other hand, has a very high conductivity due to the ions that come from dissolved salts.

4.3 Conductivity and temperature:

The conductivity of a solution increases with increasing the temperature, because viscosity of aqueous solution decreases and so the mobility of the ions increases in the solution. The number of ions in the solution increase also by increasing the temperature due to dissociation of molecules. A solution's change in conductivity with temperature depends on the electrolyte's identity and concentration.

The rate at which a conductivity of solution changes with an increase of temperature called The Temperature Coefficient of Variation and is expressed as the percentage increase conductivity for a temperature change of 1°C [%/°C]. The conductivity varies about 1–3% per degree C, and this temperature coefficient may itself vary with concentration and temperature. Table 2 shows the temperature coefficient of conductivity variation with temperature for some solutions.

Table 4: Temperature Coefficient of Variation for Common Solutions

Solution	Temperature Coefficient of Variation %/ °C at 25 °C
Ultrapure Water	5.50
NaOH 5%	2.01
NaOH 30%	4.50
HCl 5%	1.58

HCl 30%	1.52
KCl 5%	2.01
KCl 30%	1.68
Fresh Water	Ca.2.0

Several values of *temperature coefficient of conductivity variation* are commonly cited in the standard literature. For example, Clesceri *et al.* (1998, pp. 2–47) recommended $a = 0.0191$, based on the EC-temperature relation of 0.01 M KCl solution. Groundwater textbooks frequently cite ‘2% increase of EC per 1 °C increase of temperature’ (Matthess, 1982, p. 71; Hem, 1985, p. 67),

Which translates into $a = 0.02$, while geophysicists commonly use $a = 0.025$ (Keller and Frischknecht, 1966, p. 31). The use of an arbitrarily chosen value of a may be justified, considering a large degree of uncertainty in salinity- conductivity relation. [28]

As it can be seen in table 2, the temperature has a significant effect on the measurement of the conductivity.

4.5 Temperature Compensation:

Temperature control is impractical in many applications so for comparison reasons a common reference temperature is used. The measured samples conductivity values are corrected to values that correspond to conductivity measurements at a standard temperature usually measurements refer to 25° C, this is known as temperature compensation. This means if a measurement is performed at a different temperature; it will then be corrected to 25° C by taking into the consideration the temperature coefficient. The conductivity meters measure the actual conductivity and temperature and, by using a compensation function, the conductivity value will be converted to a value at the reference temperature. The temperature must be always connected to the conductivity results.

The same temperature must be used for both calibration and measurement to obtain high accuracy.

Temperature compensation can be done by different ways:

- 1) Non-linear temperature compensation:
- 2) Linear temperature compensation.
- 3) Manual temperature compensation.

Linear temperature compensation:

An experimental correction was proposed assuming a linear increase of conductivity versus temperature based on the same temperature coefficient. It is used to convert the conductivity at the measured temperature to the conductivity at 25°C or another reference temperature by using the following linear equation:

$$EC_{25} = EC_T / [1 + \alpha (T - 25)] \quad (4.2)$$

EC_T : electrical conductivity at temperature T [°C]

EC_{25} : electrical conductivity at 25 °C [S/m] .

T : Sample temperature [°C].

α : temperature coefficient of conductivity variation [%/°C⁻¹] .

The temperature coefficient can be determined by measuring the conductivity at two different temperatures and by solving equation (1) for α , we can calculate α :

$$\alpha_{T,25} [\% / ^\circ\text{C}] = (\text{EC}_T - \text{EC}_{25}) / \text{EC}_{25}(T-25) \quad (4.3)$$

Table 3 shows the temperature coefficient of some electrolytes.

Table 1: Temperature coefficient of some electrolytes.

Electrolyte	α (%/°C)
Acids	1.0 - 1.6
Bases	1.8 - 2.2
Salts	2.2 - 3.0
Drinking water	2.0
Ultrapure water	5.2

Linear Temperature compensation for salt solutions is normally done by assuming a linear increase of conductivity versus temperature of typically 2% per degree Celsius.

Non-Linear temperature compensation:

When the assumption of linear compensation is not valid, a polynomial function can be used to get results that are more precise.

The non-linear correction for different solutions is available, for example ISO/DIN7888 standard, natural water temperature correction, which is applicable for measurements between 0 and 35.9°C.

Manual temperature compensation:

Temperature compensation can also be done manually by taking a series of conductivity measurements at different temperatures and find the linear or polynomial function that fits the results to find the best equation for the temperature compensation at the wanted reference temperature.

One might notice that in both linear and non-linear compensation the results are not accurate as measuring the conductivity of the samples directly at the reference temperature.

Moreover, the greater the difference between the samples temperature and the reference temperature the higher the error.

4.6 Measuring the electrical conductivity

The electrical conductivity can be deduced by measuring the flow of electrical current through the solution, and how many salts are in the aqueous solution can be estimated from this measurement. Conductivity measurements are very common and useful methods. They are used often for controlling the concentration of salts in the solutions like drinking water control, process water quality and estimation of the total number of ions in the solution or the salinity of the seawater as it is used in this project.

The simplest way to measure the EC. is to apply an alternating voltage (AC) to two flat plates ,of area = A and separated by fixed distance=L, immersed in the solution, e.g. NaCl solution, as shown in Fig. 3, and find the resistance of the solution between these two plates by measuring the current (I) through this system.

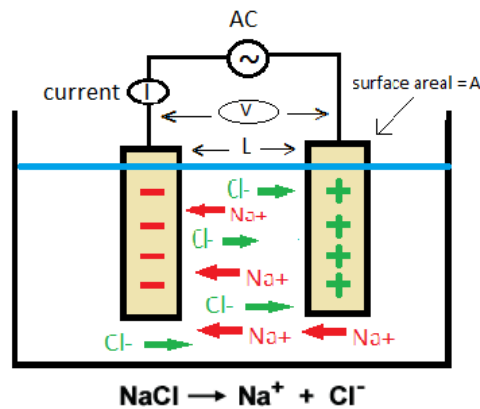


Figure 8 principle of measuring the electrical conductivity by using two electrodes immersed in NaCl solution.

Figure3 shows two electrodes in NaCl solution. The NaCl solution is dissociated into positive Na^+ ions (cation) and negative Cl^- ions (anion), the positive ions move to the cathode, and the negative ions move to the anode, and then the solution conduct the electric current. The current (I) passing through a given body of solution is proportional to the applied potential difference (V) on the two electrodes, by measuring the resulting current between those two electrodes, the resistance of the solution can be calculated using Ohm's law:

$$R = V / I \quad (4.4)$$

V : voltage [volts V].

I : current [amperes A].

R : resistance of the solution [ohms Ω].

AC voltage is used and not DC voltage to avoid the polarization of the electrodes due to the accumulation of the ions near the surface of the electrodes and chemical reaction at the surface. This leads to polarization resistance on the electrode surfaces and incorrect results of the solutions resistance.

Ohm's law plays a basic rule in conductivity measurements, where the electrolytes solutions obey ohm's law exactly as metals do. Since solution's resistance hinders the current flow, its reciprocal provides an indication of how easily current flows between the two electrodes through the solution due to ions movement. The reciprocal of the electrical resistance of a solution between the electrodes is called the conductance (G) of the solution.

$$G = R^{-1} = I / V \quad (4.5)$$

G : Solution's conductance in Siemens(S)= $\text{ohm}^{-1}[\Omega^{-1}]$.

R : solution's resistance [ohms]

As in conductors the resistance R is proportional to the length of the conductor and inversely proportional to its cross-sectional area the same is valid on the electrolyte solutions, that the resistance of the solution is proportional to the distance (L) between the electrodes and inversely proportional to the electrodes cross sectional area (A).

$$R = \rho * L / A \quad (4.6)$$

L : The distance between the electrodes [m].

A : The surface area of the electrode [m²].

ρ : The resistivity [Ω /m].

The proportional factor ρ is called the resistivity; it is a fundamental physical property of the conductor and the solution that measures the resistance of the conductor or the solution for the flow of a current over a distance ($\rho = R * A / L$). Resistivity decreases as the ionic concentration of the solution increases. Since the resistance is inverse of the conductance, then the electrical conductivity is the conductance to the flow of a current over a distance. From equation (4.6) the electric conductivity is:

$$EC = \rho^{-1} = R^{-1} * L / A = G * (L / A) \quad (4.7)$$

EC : the electric conductivity [S/m].

From equation (4.7) the conductivity depends on the surface area of the electrodes and the distance between them. In practical, it is not always easy to have well defined dimensions and uniform surface area; therefore, for any cell, there exist a cell constant (K) that must be determined by calibration with a standard solution of known electrical conductivity (EC).

From equation (4.7) the cell constant (K) is:

$$K[m^{-1}] = L[m] / A[m^2] = EC * R \quad (4.8)$$

By measuring the resistance of of solution cell with a known EC, the cell constant (K) can be easily determined using equation (4.8). After determining the cell constant (K) the electrical conductivity of any solution can be calculated by measuring its resistance according to equation (4.7).

As it was mentioned previously the electrical conductivity depends on the concentration of the ions, the ion nature and the solution temperature, which has a significant effect on the conductivity. Thus, it is important when measuring the conductivity to measure the temperature at the same time.

Cell constant, measured temperature, temperature coefficient and reference temperature are important factors that control in conductivity measurements.

The conductivity meters use the cell constant and the conductance to display the conductivity at the sample temperature.

$$\text{Electrical conductivity}(EC) = \text{Cell constant}(K) \times \text{Conductance}(G=R^{-1}) \quad (4.9).$$

4.7 4-electrodes conductivity cell:

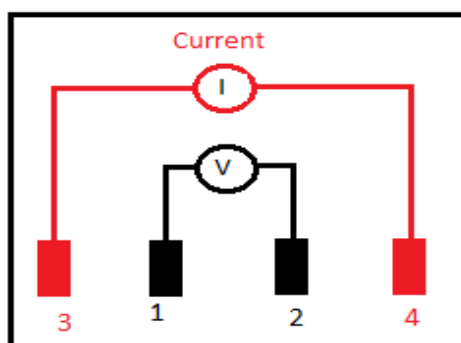


Figure4: 4-electrodes conductivity cell.

This method uses two pairs of electrodes which effectively isolate the measurement circuit. The current is only applied to the outer electrodes (3 and 4), as shown in Fig. 4. The inner electrodes (1 and 2) can measure the resulting voltage. This method is very useful to avoid the electrodes polarization. In the contrary of the 2-electrodes cell, where the current carrying electrodes and the voltage measurements take place at the same electrode , the voltage measurements take place at the inner electrodes where there is no current applied.

Using 4-electrodes cell is Ideal for measurement in high conductivity solutions. Here the calculated conductivity is directly proportional to the applied current.

4.8 Salinity

Salinity is the total concentration of dissolved salts in the water. When the salts dissolved in the water they produce positive and negative ions. These ions play a basic rule in solution`s conductance as it was explained previously. Thus, the salinity is proportional to the conductivity, until a certain ion concentration, and so can be derived from the conductivity of the solution instead of measuring it by difficult and time consumption chemical methods.

The salinity units vary depending on the applications and the reporting methods. Modern salinity measurements are dimensionless; the Practical Salinity Scale (psu) defines salinity in terms of a conductivity ratio very nearly equal to g/kg or, equivalently, ‰(parts per thousand, ppt.) [42, p.165].

The salinity of the sea water is 34.5 psu (practical salinity units), there are many kind of salts contribute in this salinity. The major ions in the sea water are chloride, sodium, magnesium, sulfate, calcium, potassium, bicarbonate and bromine. About 85% of sea water salinity is chloride and sodium.

4.9 Total Dissolved Solid (TDS):

Total dissolved solid is a measure of all the inorganic and organic substances contained in a water. In clean water the TDS is almost equal the salinity.

The TDS is proportional with the conductivity and can be found by multiplying the conductivity a TDS factor (a) that depends on the kind of solids that dissolved in the water and varies between 0.55 to 0.8 most conductivity meters will use a common approximated constant about 0.65

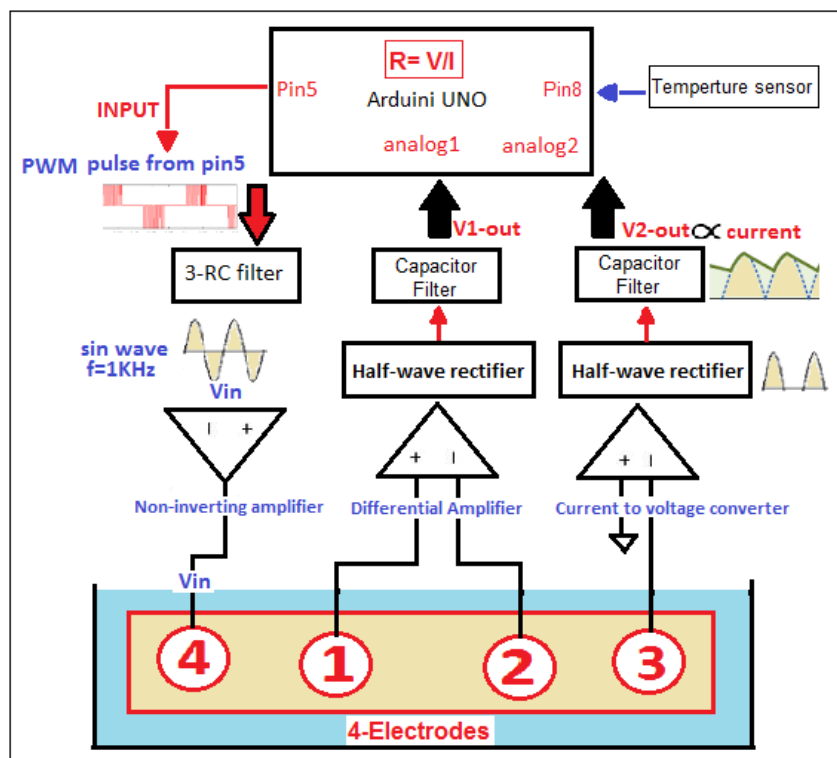
$$\text{TDS} = a * \text{EC} \tag{4.10}$$

TDS : total dissolved solid [mg/L].
 EC : electrical conductivity [$\mu\text{S}/\text{cm}$].
 a : TDS constant (0.55-0.8).

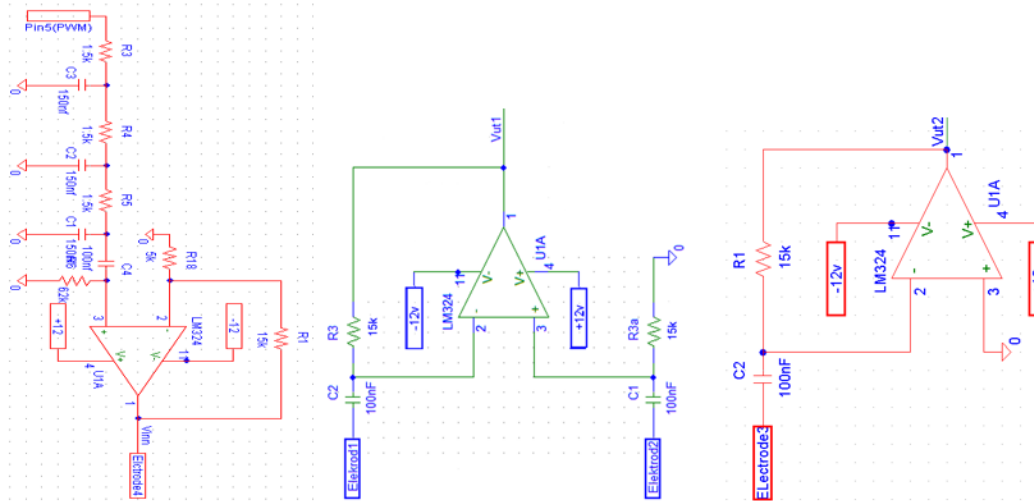
4.10 Designing 4-electrode EC conductivity meter:

This section explains the work that have been done in designing and testing 4-electrodes conductivity sensor based on Arduino Uno.

As shown in figure 5, the 4-electrode EC measurement circuit is composed of Arduino Uno control board. The Arduino both sends out the input alternating voltage to the electrodes and receives the measured signals from the electrodes in the water. It uses the measured results to find out the resistance and to determine the conductivity and the salinity. In addition to Arduino Uno the brain of the system, there is 3RC filter, differential amplifier, current to voltage amplifier, half wave rectifiers, AC to DC circuits and temperature sensor.



Figur 5: The structure of the 4-electrode electrical conductivity meter.



Figur 6: 4-electrodes Electricals Circuits

Figure 6 shows the circuit of each electrode. The inner pairs of electrodes are connected to a differential amplifier to measure the resulted voltage in the solution. There is no current applied to the inner electrodes and so there is no polarization problem to worry about. A 1 KHz alternating current is applied to the outer electrode (4). Electrode (3) is used to measure the current in the solution by using current to voltage amplifier whose output voltage is proportional with the current in the solution. The resistance of the solution can be determined Using Ohms low on both the measured voltage and measured current from the mentioned electrodes.

By using the previous explained theory and equations (4.2- 4.10), we can determine the conductivity of the solution.

$$R = V_{out1}/V_{out2} \quad ; \quad V_{out 2} \text{ are proportional with the current (I)}$$

$$\rho = R \cdot A/L .$$

$$EC = \rho^{-1} = R^{-1} \cdot L/A = G \cdot (L/A) .$$

$L/A = K$, Cell constant which can be determined by calibration as mentioned previously.

$$EC = R^{-1} \cdot K.$$

Arduino Uno

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.

4.11 Peripheral Circuit:

In order to receive the analog output signals from the amplifiers (electrodes) to the analog inputs pins of the Arduino Uno a peripheral circuits had to be used. The Arduino Uno takes analog input between 0 and 5 volts, which means that the peak value must be less than 5V and so the analog inputs have to be stepped down to be positive and less than 5 volts. Many methods can be used to do that for example, we can use potential transformer, voltage divider, difference amplifier or rectifier.

Potential amplifier is the best to be use for a huge voltage more than 400V, while it is more economically to use the difference amplifier for voltage less than 400V. In this project, the

rectifier is used to keep the voltage between 0 and 5 because we deal with small voltage which is already less than 5V so we suppose that the simplest method is the rectifier.

As shown in Fig.(7), Two steps was done before sending the signal into the analogue input pins of the Arduino:

- 1) Half wave rectifier (precision rectifier circuit).
- 2) Capacitor Filter to smooth out the ripple to measure the peak voltage of an AC signal.

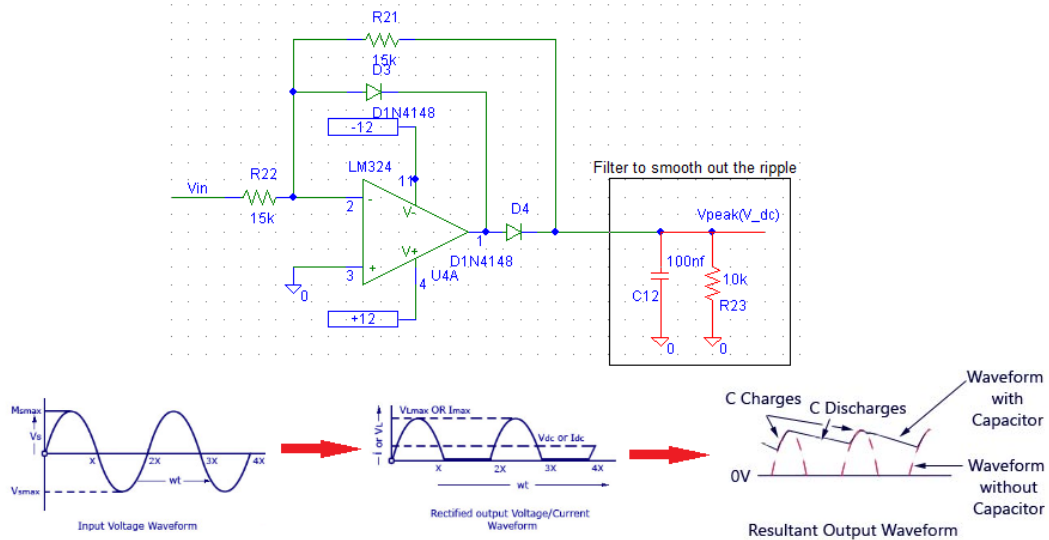


Figure 7: Half wave rectifier and filter to smooth the ripple before the output goes to the analogue pin of the Arduino. [34]

Precision half-wave rectifier (The Super diode)

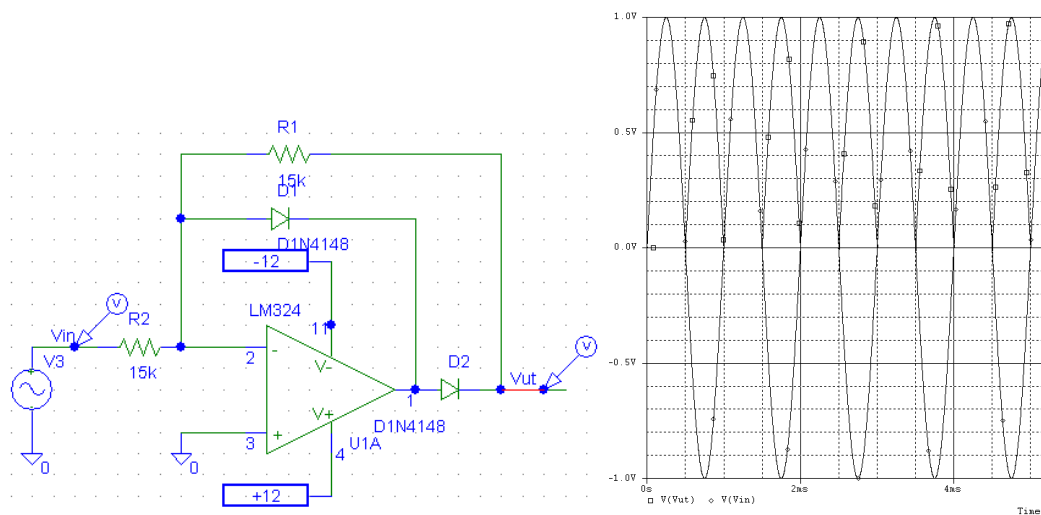


Figure 8: Precision Half wave Rectifier Circuit (Super diode) and it is simulating.

The rectifier circuits are used to convert the amplitudes of AC signals to DC values to be more easily measured. The simple rectifier circuit build with diodes can't be used to rectify signals less than a diode drop so they can't be used to measure small amplitudes, and the output voltage equals the peak value of the input minus the forward voltage bias drop of the

diode. Op-amp can be used in conjunction with diodes in order to get high degree of precision and to can deal with small signals.

As shown in Figure8 an ideal half wave rectifier can be made from an inverting op-amp with two diodes.” For the negative half of the input swing, the first diode D1 is reverse biased, the second diode D2 is forward biased, and the circuit operates as a common inverter amplifier with gain $-R1/R2$. For the positive half of the input swing, D1 is forward biased closing the feedback around the amplifier. D2 is reverse biased disconnecting the output from the amplifier. The output will be at the virtual ground potential ($-$ input terminal) through resistor $R2$ ” [37].

As we see in figure 8 (right), the peak of the output is equal to the peak of the input and this is the aim of the super diode.

Capacitor Filter

In order to measure the peak of the AC signal a capacitor is added in parallel (as shown in Fig.7). The property of a capacitor is that it allows AC component and blocks the DC component. The operation of the capacitor is to smooth out the rectified signal before the output goes to the Arduino analogue inputs pin. During the rise voltage cycle of the input, the capacitor stores up the charge and discharges slightly through the load resistor during the fall voltage cycle.

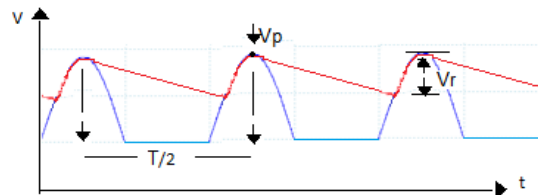


Figure 9 Resultant output waveform after the capacitor filter.

The ripple V_r depends on how far the capacitor discharges through the load resistance R_L before the next charging cycle.

The discharge of the capacitor is exponential, but the initial portion is almost linear . Thus, we can do the following approximation of the discharging [35]:

$$V = V_p e^{-t/R_L C} \approx V_p (1 - t/R_L C)$$

$$\frac{dV}{dt} = -\frac{V_p}{R_L C}$$

$$V_r = \frac{T}{2} \frac{dV}{dt} = \frac{V_p}{2R_L C f} \quad (4.11)$$

Where:

V_p : Peak voltage [v].

R_L : Load resistance connected in parallel with the capacitor filter [Ω]

C: the capacitance of the capacitor filter [F: farad]

T : period time [s] = $1/f$

f : the AC signal frequency [Hz]

V_r : ripple peak tp peak [v]

The dc output voltage V_{dc} is the peak voltage minus the average ripple, $V_p/2$,with neglecting the short charging portion.










$$V_{dc} = V_p - \frac{V_r}{2} = V_p - \frac{V_p}{4 R_L C f} \quad (4.12)$$

As shown in equation(4.11) , the larger the capacitor and the load resistor the smoother the ripple. Thus, the capacitor have to be large enough and match the load to get better dc output voltage with low possible ripple $\approx V_p$.

4.12 Measuring conductivity experiment:

In this experiment a simple 4-conductivity probes was constructed and integrated into two different circuits to test its behaviour in the salt water and tap water, to find the resistance and the conductivity in both salt water and tap water in order to design a 4-electrodes conductivity meter.

4.12.1 Materials and tools used in the experiment:

- Picoscope with PC  or oscilloscope  
- Isolated copper wires. 
- (4) 5mm-stainless steel screws, nuts and washers. 
- Cylinder of Plexiglas cylinder  and CD cover to lock the bottom of the cylinder
- Super Glue 
- (6) 100nf capacitors
- (3) 1.5K Ω resistors
- (6) 15 K Ω resistors
- (1)62 K Ω resistors
- () Ω resistors
- (4) diodes
- Breadboard
- Jumpers
- Power supply
- (3) Amplifiers TL072CP  
- Arduino UNO
- Wire stripper

4.12.2 Probe Design and building:

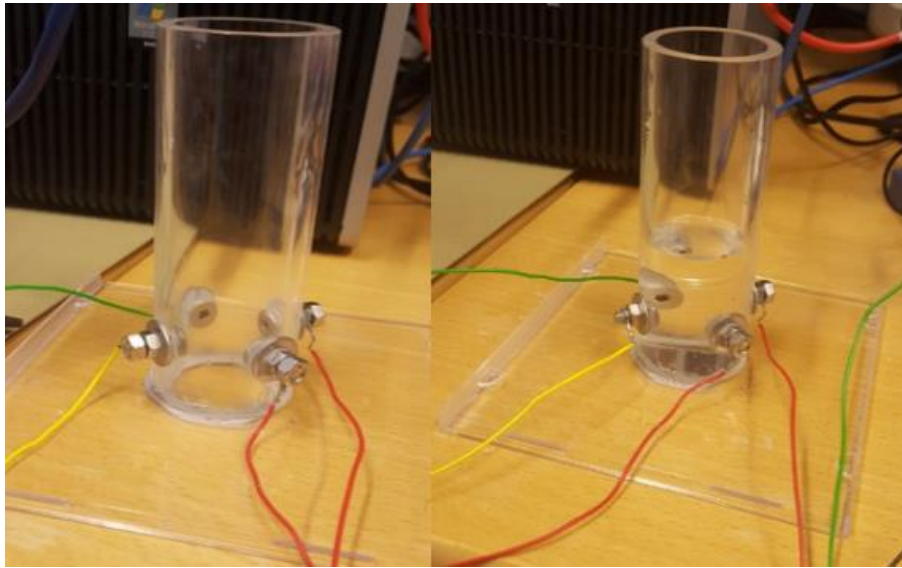


Figure 10: The design of the 4-probes

The probes give a significance whether the solution is able to carry the current or not or if some solutions are more conductive than others are. The measuring circuit contains a gap filled by a solution samples between the four electrodes, which measure the resistance in the solution. This can be imaged like adding a resistor in series at the gap to connect the circuit together.

Fig. 10 shows the construction of the probes cell, which contains a cylinder of Plexiglas with four stainless steel screws was fixed through the cylinder wall in opposite to each other, as shown in Fig. 11, and connected to copper wires from the outside of the cylinder. Some glue was used between the screw and the cylinder wall to avoid the water leakage. At the bottom of the cylinder, a CD cover was glued to keep the solution in the cylinder.

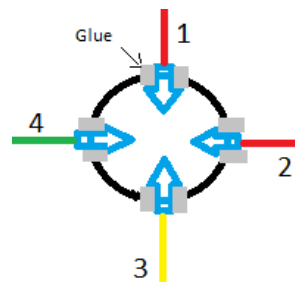


Figure 11: A simple sketch of the cross section of the cylinder and the electrodes, which are made from steel screws.

An AC voltage was applied via the green wire to electrode 4 in the solution. Electrode 3 was connected via the yellow wire to a current to voltage amplifier circuit, which was built on the breadboard, and electrode 1 and 2 through the two red wires was connected to a differential amplifier circuit on the breadboard. By this connection, The electric field causes the ions in the solution to move back and forth producing a current, the red probes works to measure the differential voltage and the yellow probe measures the current in the solution then by using ohm's law the resistance of the solution can be estimated. The lower the conductivity of the solution the less ions in the solution and so the higher the resistance.

The electrodes was connected to the circuits shown in Fig. (12). Those circuits was created on the breadboard as shown in Fig. (13).

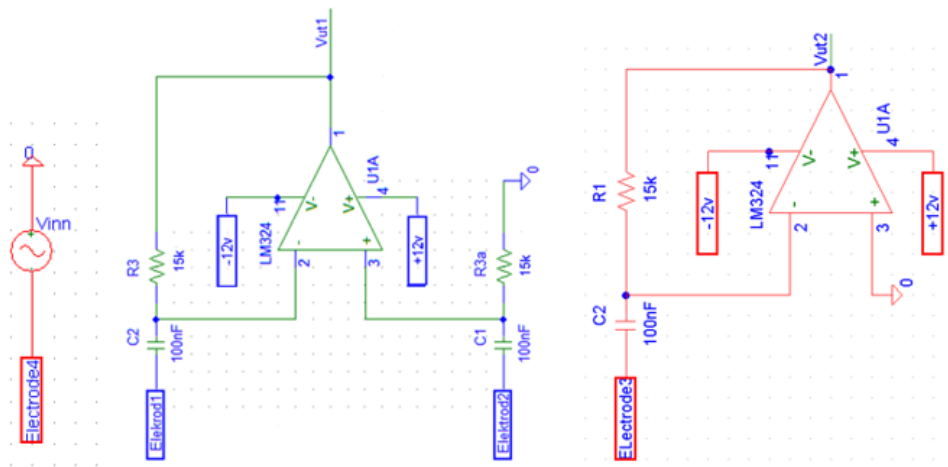


Figure 12 : The electronic structure of the probes circuits; the current is applied to electrode4, differential amplifier circuit is connected to electrode1 &2 and electrode4 is connected to current to voltage amplifier.

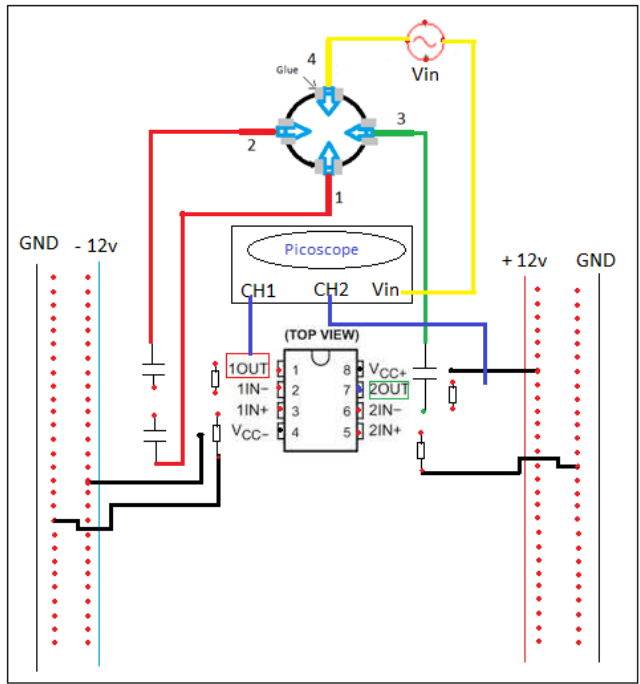


Figure 13: A simple sketch of the electrodes circuits on the breadboard.

4.12.3 Differential Amplifier:

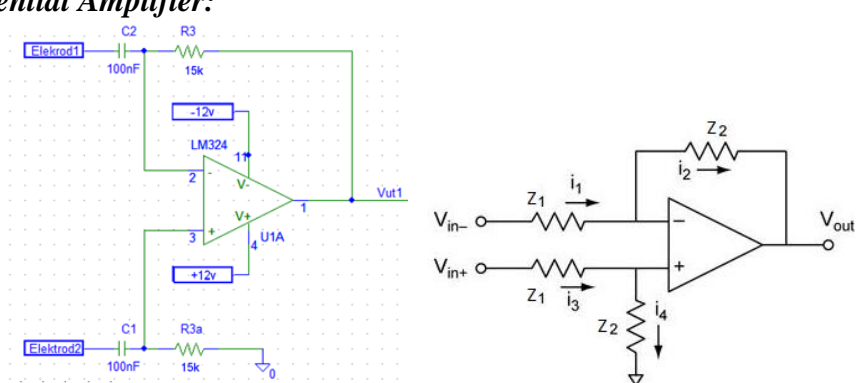


Figure 14: Differential Amplifier

The output of the differential circuit depends on the difference of the two input voltages that comes from the electrodes in the solution which depends on the resistance of the solution self; the higher the resistance the higher the voltage difference the higher the output voltage

For differential amplifier:

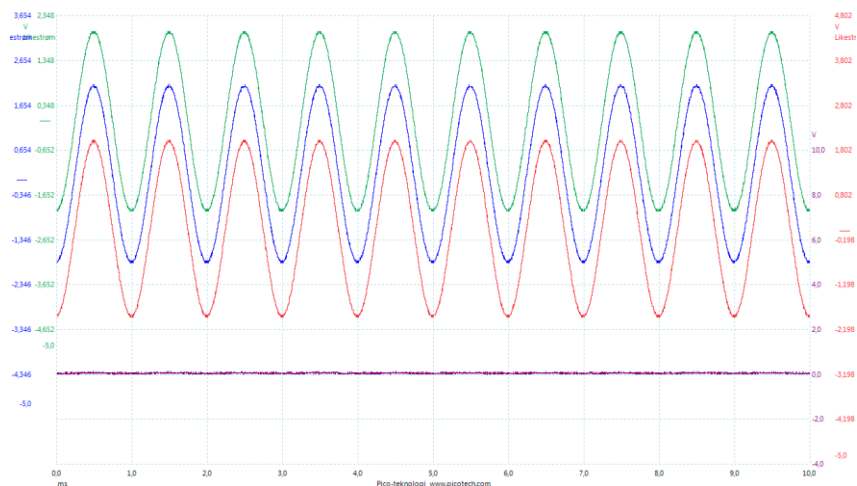
$$V_{out} = \frac{Z_2}{Z_1} (V_{+in} - V_{-in}) \quad (4.13).$$

In our amplifier circuit, two equal capacitors was used after each input to remove the DC term from the input signal and so the output voltage is:

$$V_{out} = \frac{R}{X_C} (V_{+in} - V_{-in}) = j\omega RC (V_{+in} - V_{-in}) = j2\pi f RC (V_{+in} - V_{-in}) \quad (4.14).$$

Equation 4.14 shows that the output voltage of the circuit depends on the frequency.

In testing the differential circuit and the effect of a solution of 35g/L salt water , an alternating current of amplitude 2v and frequency 1Khz was sent into the solution via electrode4, and by using the PicoScope the voltage at electrode 1 and 2 was measures directly out of the solution and before the amplifier. The results are in the following figure.



Kanal	Navn	Verdi	Min.	Maks.	Gjennomsnitt	σ	Overføringstall	Spenn
C	Topp til topp	4.007 V	4.007 V	4.087 V	4.055 V	24.71 mV	20	Hele spor
A	Topp til topp	3.977 V	3.977 V	4.018 V	3.992 V	19.64 mV	20	Hele spor
B	Topp til topp	3.946 V	3.946 V	3.986 V	3.948 V	8.941 mV	20	Hele spor
A - B	Topp til topp	143.9 mV	143.9 mV	143.9 mV	143.9 mV	0 V	1	Hele spor

Figure 15: The PicoScope viewing of the signals coming from the electrodes in saltwater. Up(C):Vin, middle(A): V1 from electrode1, down(B):V2 from electrode2 and purple is V1-V2.

Saltwater is a good conductive due to the free ions that carry the electric current. Therefore, as we see in Fig.15, the difference between V1 and V2 is very little which means that the resistance of the salt solution is very low.

To study the sensitivity of the electrodes and the effect of the solution, frequency response (amplitude vs. frequency) was plotted using the PicoScope sweep tool for the frequency from 1Hz to 300 KHz for both saltwater and tap water. As shown in Fig. (16), there is no effect of the frequency on the amplitude at low frequencies. The saltwater works as a low pass filter with break point at about 12 KHz.

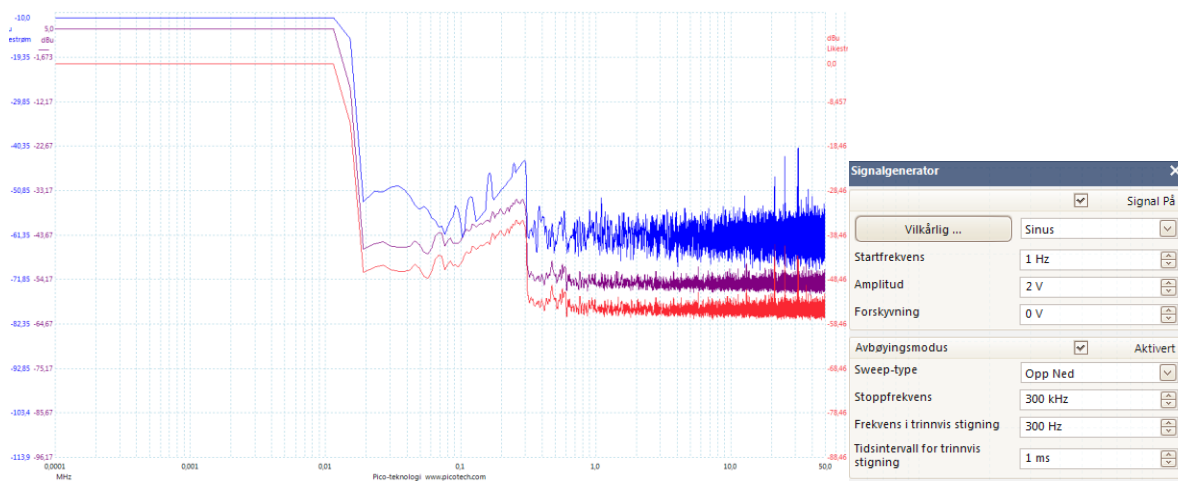


Figure 16: Frequency response of the saltwater using the sweep tools of the Picoscope. up = V1, down: V2, middle: V1-V2

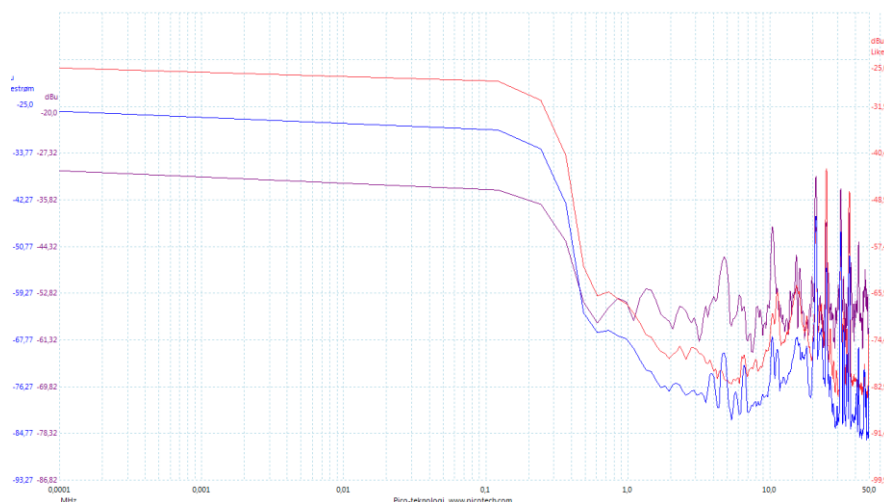


Figure 17: Frequency response of the tap water using the sweep tools of the Picoscope. middle = V1, up: V2, down: V1-V2

The tap water works also as a low pass filter with break point at about 100 KHz.

The frequency response was also found manually by measuring the amplitude at the frequencies from 1Hz to 300 KHz and was plotted using Excel. The following figure shows

the plot of frequency response at electrode1 and 2 given by V1 and V1 and the difference between them in both salt and tap water.

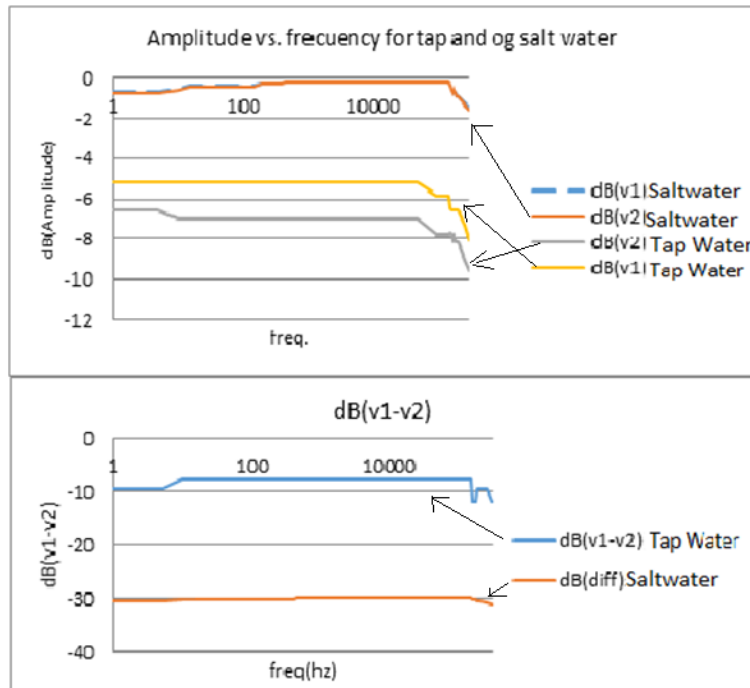


Figure 18: frequency response for saltwater and tap water.

From the previous figures, we can see that the best frequency the sensor can be sensitive at is almost between 300Hz and 100 KHz.

The plot of the difference between the voltage on electrode 1 and 2 shows that in the higher the salt in the water the lower the difference between V1 and V2 because the salt dissociated in water into positive and negative free ions that allow the electric current to flow through the solution.

4.12.4 Current to voltage amplifier:

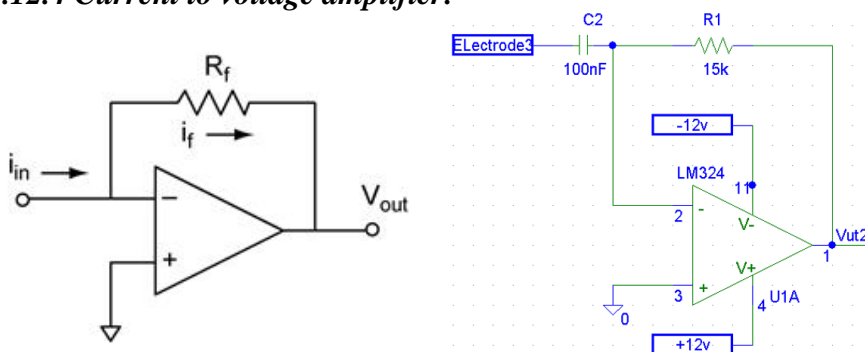


Figure 19: Current to voltage amplifier.

The circuit takes the input current (I_{in}) from the solution through electrode3 and convert it into voltage output (V_{out}). The relationship between I_{in} and V_{out} can be found using the op-amp rules Ohm's law and Kirchoff's voltage law.

From op-amp's rules, we know that $V_- = V_+$ and so $V_- = 0$ because V_+ are connected to ground. Since the input impedance of the op-amp + and - inputs is infinite and there is no current flows between them, so $I_{in} = I_f$.

$$I_{in}=I_f = \frac{(V_- - V_{out})}{R_f} \quad , \text{ but } V_- = V_+ = 0 .$$

$$I_{in} = \frac{(0 - V_{out})}{R_f} = \frac{(-V_{out})}{R_f}$$

$$V_{out} = -R_f I_{in} \tag{4.15}$$

In our circuit, we have $V_{out2} = -R_1 I_{in}$. The output voltage is directly proportional with the input current. Which means the lower resistance solutions, the better the current flows and so the higher the output voltage of the amplifier. This can be proved in the next section of testing the probes.

Again, the capacitor in the circuit is used to drop the dc offset from the input signal but it also effect the output. By using the same rules and laws we can find the relationship between V_{in} , the capacitor impedance and V_{out2} .

$$I_{in}=I_f$$

$$\frac{(V_{in} - V_-)}{X_c + R_s} = \frac{(V_- - V_{out})}{R_f}$$

$$V_{out} = -V_{in} \frac{R_f}{X_c} = -V_{in} j\omega c R_f = -j2\pi f V_{in} C R_f \tag{4.16}$$

One can notice that the output depends on the frequency which means that we actually going to measure the impedance of the solution.

4.12.5 Testing the probes

After building the electrodes and integrating them into the amplifiers circuits on the breadboard, they was tested in saltwater solution of 35g/l salt and in tap water. The output of each amplifier was visualized using the PicoScope. Differential amplifier output is called V_{out1} , and the current to voltage output is called V_{out2} . A signal of 1V amplitude and 1KHz frequency was applied to the saltwater solution and tap water. Fig.20,21 shows the input and output signals for saltwater and tap water. The green signal is the input, the blue is V_{out1} and the red is V_{out2} . The figures show that the more the ions in the solution the better the current flow. Thus V_{out2} in salt water is greater than V_{out2} in tap water where there is higher resistance to the current flow. While V_{out1} in saltwater is less than V_{out1} in tap water due to the lower difference between V1 and V2 because of lower resistance between the two electrodes in the saltwater than in tap water .

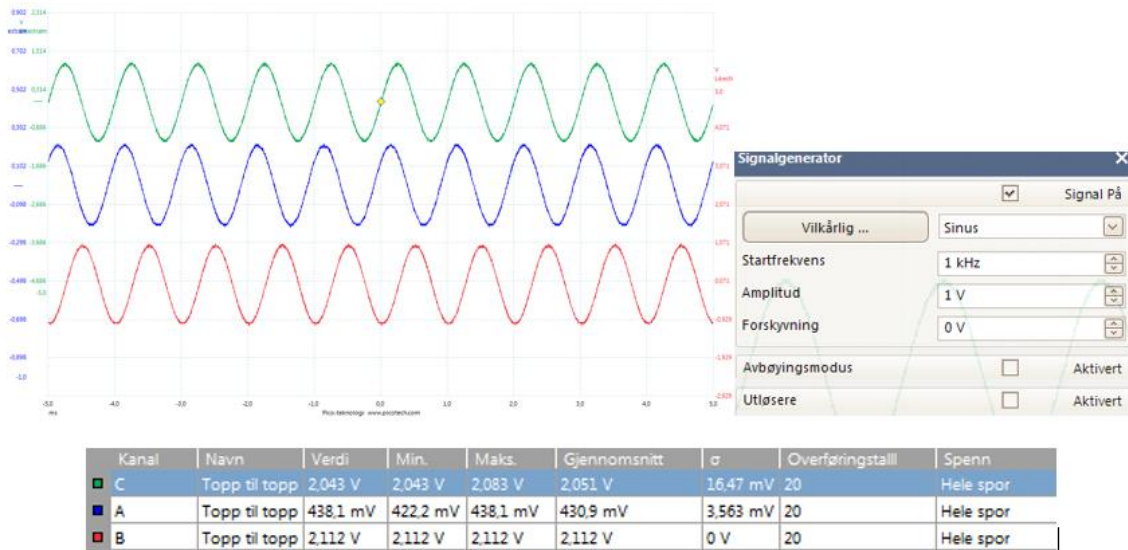


Figure 20: The input signal and the output signals of the conductivity probes in saltwater. $C(\text{up})=V_{in}$, $A(\text{middle})=V_{out1}$, $B(\text{down})=V_{out2}$

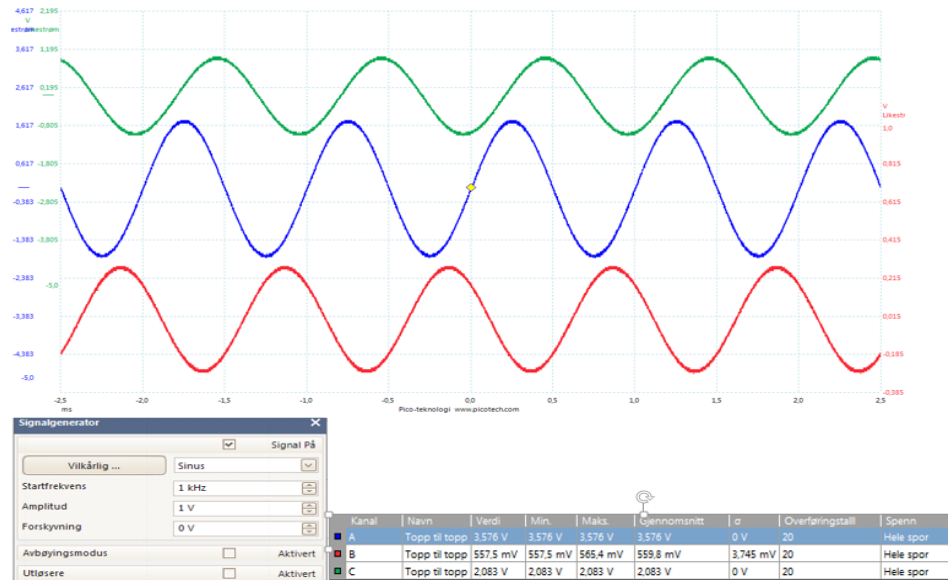


Figure 21: The input signal and the output signals of the conductivity probes in tap water. $C(\text{up})=V_{in}$, $A(\text{middle})=V_{out1}$, $B(\text{down})=V_{out2}$

The next step is to prepare the circuit to be connected to the Arduino and then using the Arduino to measure the voltage and the current and to estimate the conductivity. As it was discussed before, precision half-wave rectifier and capacitor filter was used to step down the analogue input of the Arduino analogue input pins, which take analogue inputs between 0V and 5V.

4.12.6 Testing the probes with the peripheral circuits:

The precision half-wave rectifier and the capacitor filter was built on the breadboard for both the output of the differential amplifier and the current to voltage amplifier then the whole circuit was tested again in both saltwater and tap water using Picoscope. The following figures shows the results of the test.

The first test was done by using a simple peak detector composed of a simple diode to detect the positive peaks of the signal, with a 100 nF capacitor and 1MΩ resistor to smooth out the ripple. Fig.22 shows the output voltage of the differential amplifier after applying Ac voltage, of 1V amplitude and 2 KHz frequency, to the electrodes in tap water.

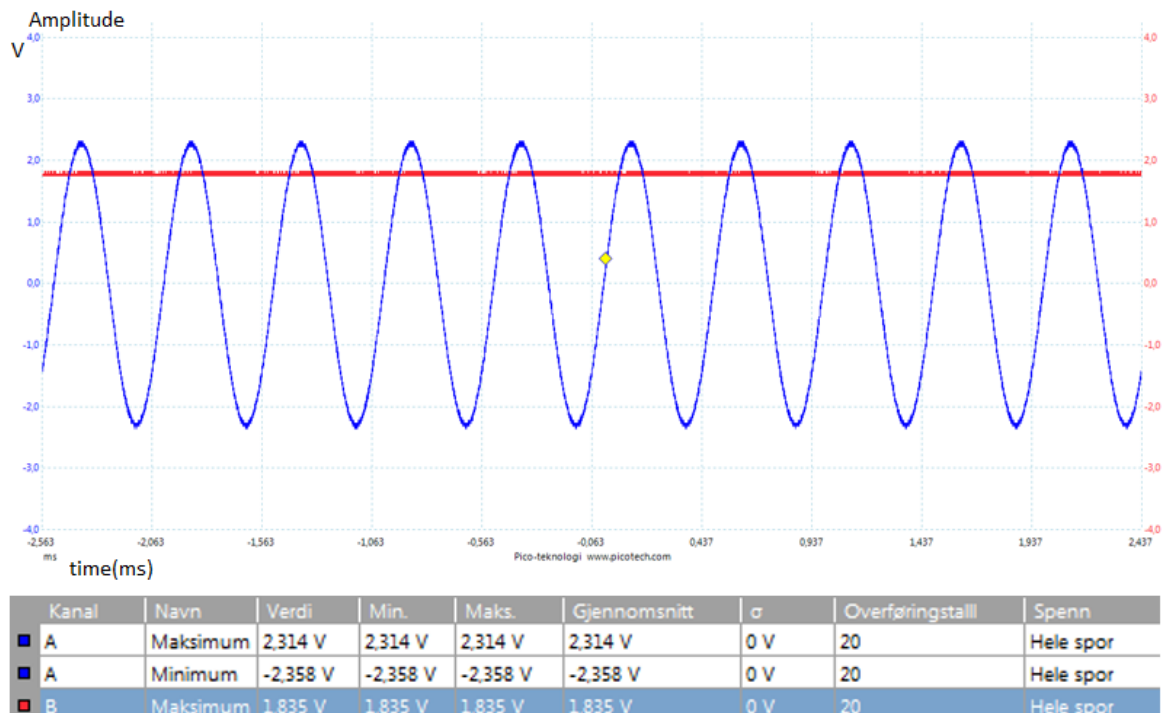


Figure 22: V_{out1} in tap water before the simple detector (down) and after it (up)

Fig.23 shows the output of the current to voltage amplifier in tap water before and after simple peak detector.

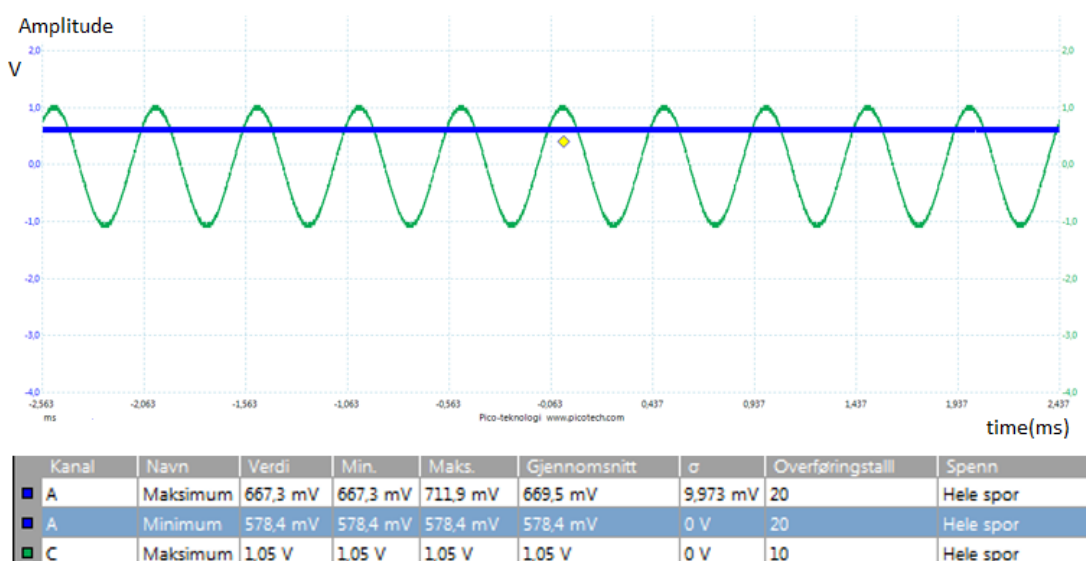


Figure 23: V_{out2} in tap water before the diode (down) and after the filter (up)

As shown in the previous figure simple peak detector of a simple diode doesn't actually give the true peak value of the signal because of the diode drop voltage. This voltage drop can be avoided by using the precision half wave rectifier (the super diode) which was shown in Fig.8 and discussed in previous section.

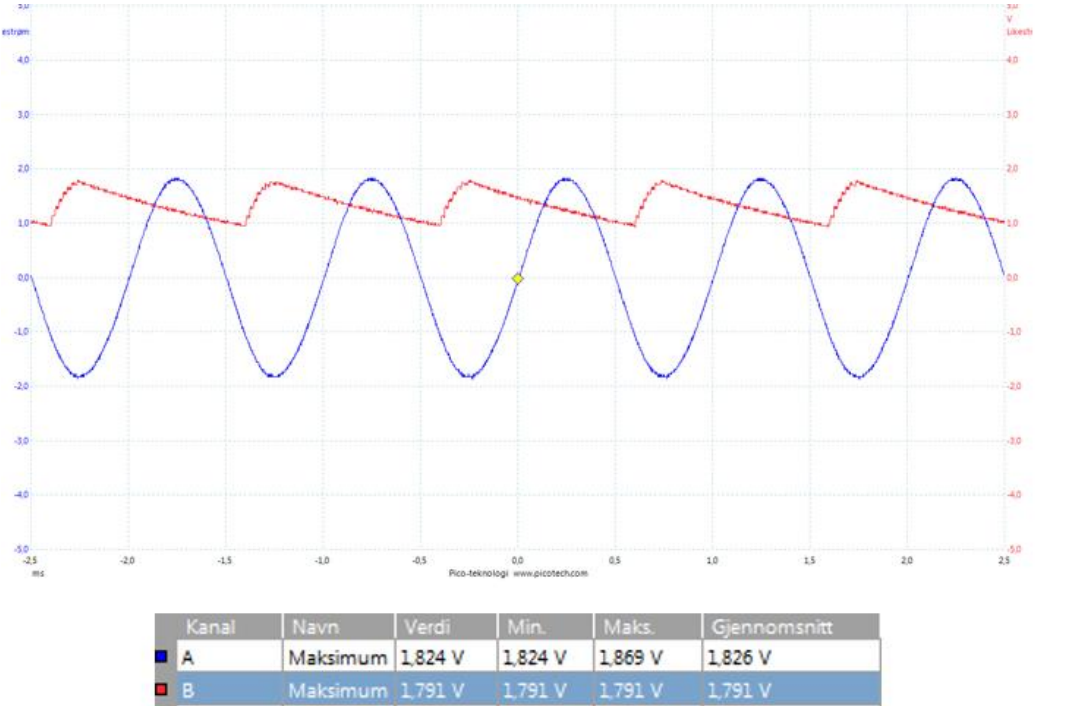


Figure24: V_{out1} in tap water before and after the super diode.

Fig.24 and 25 show how the super diode is precision to get very close peak value. Here we don't get a constant output as in the previous test because the resistor in this test is much smaller than that was used in the previous test.

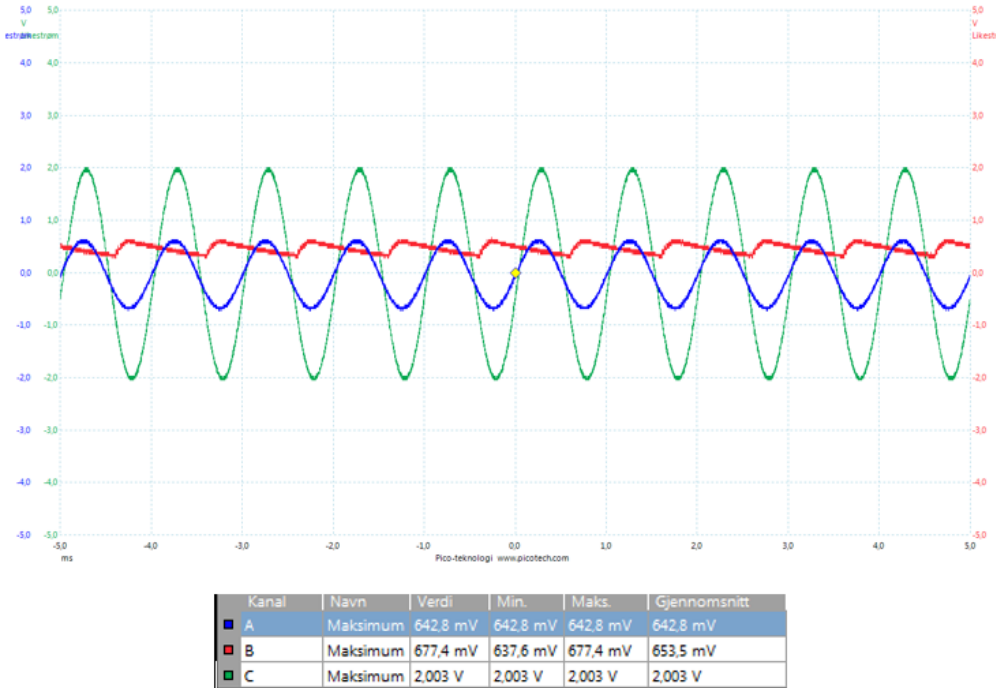


Figure 25: V_{out2} in tap water before and after the super diode and the 2V input signal (green).

Now the signals can be sent to the Arduino analogue pins. The input signal must be less than 2V amplitude and with 1KHz frequency to avoid getting any output greater than 5V.

4.12.7 Generating sine wave input signal from the Arduino Uno:

This part of the project owes much to the tutorial example of generating sine wave from Arduino, given in Eprojectszone [14]. The example shows how to generate a sine wave modulated from different PWM (Pulse Width Modulation) signals. PWM is a technique for generating analog results with digital control, which creates a square wave signals that switch between on and off. PWM can be defined by two main components: a frequency and a duty cycle. The duty cycle is the fraction of one period in which a signal is active (on).

Some Arduino pins can generate PWM signal, we can adapt this signal for sign equation. The used frequency (f) here is 1 KHz, so the time period is $T = 1/f = 1\text{ms}$, and the half cycle period is 0.5ms. In this half period will be many pulses with different duty cycles starting with small duty cycle to the maximum duty cycle at the middle of the signal and back to the small duty cycle at the end of the signal.

Pin5 and Pin 6 will be used to generate the sine wave one for the positive half cycle and one for negative half cycle. **Phase correct PWM** at a frequency **31372 Hz will be used** for a smooth signal.

How to calculate the necessary duty cycle for each pulse to produce sine wave of 1 KHz frequency(full cycle period=1ms)?

$$f = 31372\text{Hz}$$

The period for each pulse $T = 1/f = 31.8\text{ us}$

The number of pulses for a half cycle $N = \frac{\text{half cycle}}{\text{period for each pulse}} = \frac{0.5\text{ms}}{31.8\text{us}} = 15.723$ pulses.

Now we can use the function $Y = \sin x$ to calculate the duty cycles for each pulse.

In degrees, half cycle has 180 degree for 15 pulses, so for each pulse $\frac{180}{15} = 12^\circ/\text{pulse}$. This means to move forward 12° every pulse.

The following program will find the duty cycle for each pulse between (0-180°): [14]

```
float x=0;
float y=0;
const float pi=3.14;
int z=0;
float v=0;
int w=0;
boolean OK=true;
void setup() {
  Serial.begin(9600);
}
// the loop function runs over and over again forever
void loop() {
  if (w==0){
    v=x*pi/180; // making deg in radians
    y=sin(v); //calculate sinus
    z=y*250; // calculate duty cycle(250 not 255 because will help to turn off transistors)
    delay(100);
    if(OK==true){
```

```

x=x+12;// increase the angle
}
}
if (x>90){
OK=false;
}
if (OK==false){
x=x-12; //decreasing the angle for the other half
}
Serial.println(z);// on the serial monitor will appear duty cycles between 0 and 180 deg
}
After finding the duty cycles for each pulse, we put them in vector:
sinPWM[]={ 1,51,101,146,185,216,237,248,248,237,216,185,146,101,51,1 }

```

we will use Timer0 with pin5 and pin6 to write the duty cycles and we will use Timer1 in CTC (**C**lear **T**ime on **C**ompare) mode with interrupts to alternate the pins for each half cycle and make a variable duty cycle for each pulse. A signal of frequency 31372 Hz will be generated, the same frequency as pin5 and pin6 frequency, then the duty cycle of the vector will be changed after each pulse with an interrupt. The pins will be changed at the end of the vector and start over.

```

/*
 * In ISR function we set the OCR0A and OCR0B with duty cycle values
 * and change this values according to vector,
 * also at the finish of each crossing of vector we change the enabled pin.
 * Before sine wave we must see if it is everything ok so we have alternate the pins
 * at a stable duty cycle OCR0A and OCR0B equals with 128.
 */
int i=0;
int x=0;
int OK=0;
int sinPWM[]={ 1,51,101,146,185,216,237,248,248,237,216,185,146,101,51,1
};

void setup() {
Serial.begin(9600);

pinMode(5, OUTPUT);
pinMode(6,OUTPUT);

cli();// stop interrupts
TCCR0A=0;//reset the value
TCCR0B=0;//reset the value
TCNT0=0;//reset the value
//0b allow me to write bits in binary
TCCR0A=0b10100001;//phase correct pwm mode
TCCR0B=0b00000001; //no prescaler
TCCR1A=0;//reset the value
TCCR1B=0;//reset the value
TCNT1=0;//reset the value
OCR1A=509;// compare match value
TCCR1B=0b00001001; //WGM12 bit is 1 and no prescaler

TIMSK1 |=(1 << OCIE1A);// enable interrupts

sei();//stop interrupts
}
ISR(TIMER1_COMPA_vect){// interrupt when timer 1 match with OCR1A value
if(i>14 && OK==0){// final value from vector for pin 6
i=0;// go to first value of vector

```

```

OK=1;//enable pin 5
}
if(i>14 && OK==1){// final value from vector for pin 5
i=0;//go to first value of vector
OK=0;//enable pin 6
}
x=sinPWM[i];// x take the value from vector corresponding to position i(i is zero indexed)
i=i+1;// go to the next position
if(OK==0){
OCR0B=0;//make pin 5 0
OCR0A=128;//enable pin 6 to corresponding duty cycle
}
if(OK==1){
OCR0A=0;//make pin 6 0
OCR0B=x;//enable pin 5 to corresponding duty cycle
}
}
}
void loop() {
}

```

OCR1A=509, the compare value can be found by using the following formula:

$$f_{OC1A} = \frac{f_{clk}}{2 * N(1 + OCR1A)}$$

f_{OC1A} : the output frequency.

f_{clk} : the clock frequency of the chip (16MHz).

N : the counter divider (prescaler).

$OCR1A$: the counter value.

Rewriting the formula to find the value of the counter when $N=1$ (no prescaler) and

$f_{OC1A}=31372\text{Hz}$.

$OCR1A = \frac{f_{clk}}{Nf_{OC1A}} - 1 = \frac{16000000}{1*31372} - 1 = 509$, we drop 2 from the formula because the signal must be high and low to have a frequency to the signal.

The results on the Picoscope:

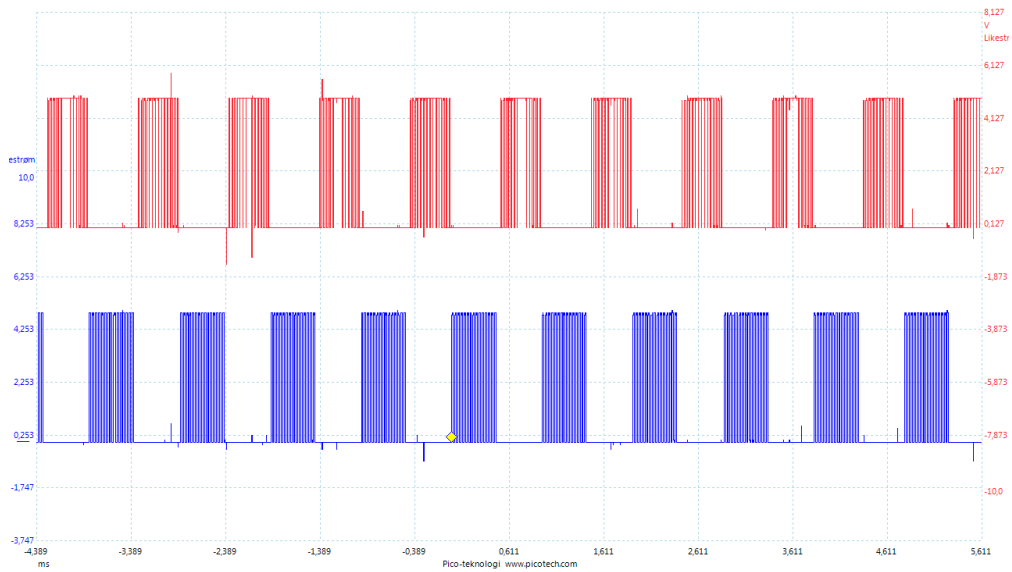


Figure 26: The PWM signal with variable duty cycles on pin5(up) and pin6(down).

As shown in Fig.26 the output signal is PWM signal with variable duty cycle. This signal can be adapted to half sine wave by using a low pass filter, and a minus operation can be done between the two signals to get a full sine wave. Instead of that, a square to sine wave converter of 3-RC circuits, shown in Fig.28, was used to convert the square PWM signal to a sine wave. The result is, as shown in Fig.27, two sine waves on both pin 5 and 6 instead of having half sine wave on each pin.

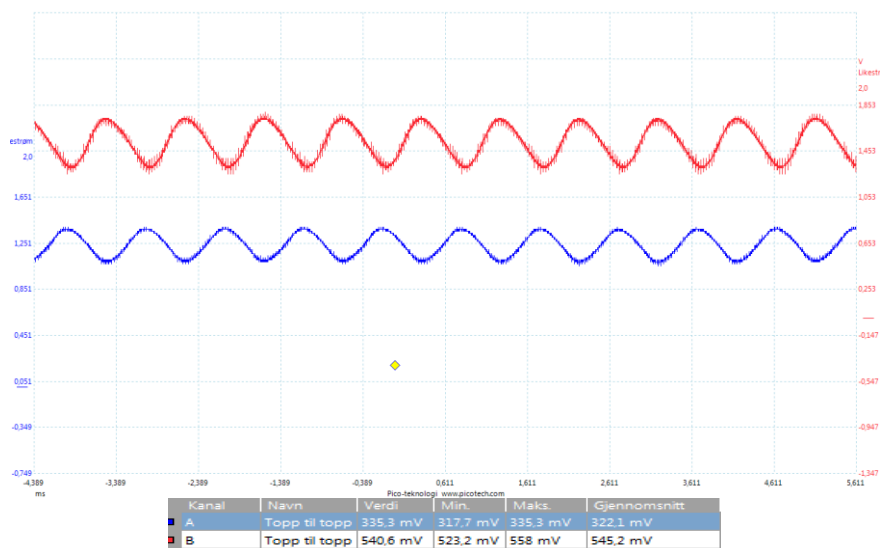


Figure 27: PWM signals converted to sine wave signals on pin5 (up) and pin (down).

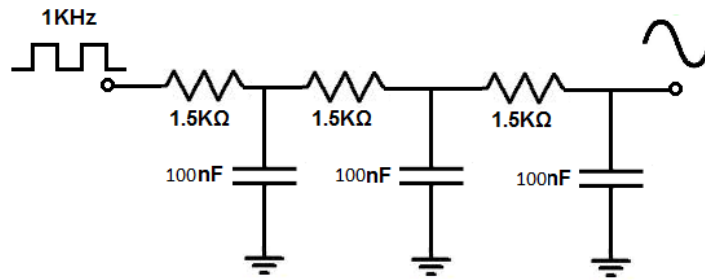


Figure 28: square to sine wave converter.

As shown in Fig.27 the resulted signal is small, so it must be amplified to the wanted voltage value. A follower with gain was used to amplify the signal and avoid any current load. Fig.29 shows the schematic of the 3-RC and amplifier circuit. Fig.30 shows the result of amplifying the filtered signal from pin5.

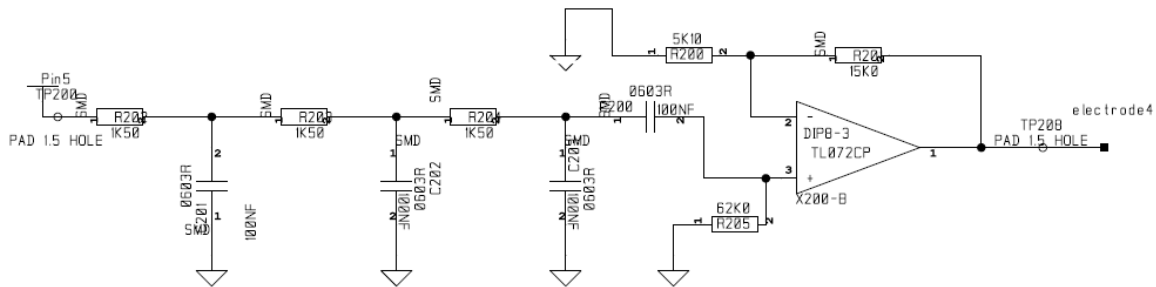
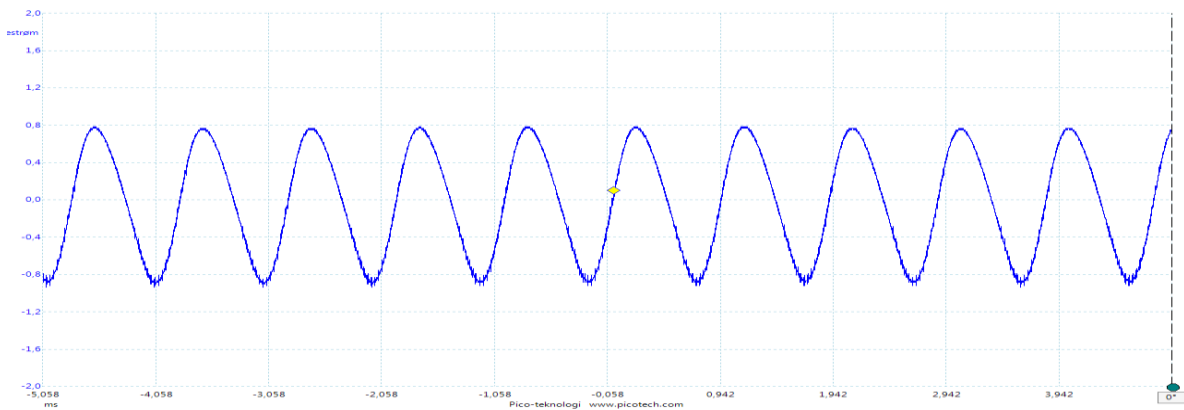


Figure 29: Schematic of the circuit from Pin5 to electrode4.



Kanal	Navn	Verdi	Min.	Maks.	Gjennomsnitt
A	Topp til topp	1.725 V	1.725 V	2.85 V	2.004 V
A	Frekvens	1.044 kHz	968.4 Hz	1.046 kHz	1.031 kHz

Figure 29: The amplified sine wave signal generated at pin5.

The amplified signal was sent to the solution via electrode4, and the output signals was received by the Arduino analogue pins A1 and A2 via the peripheral circuits that was described earlier. The complete connection of the electrodes, breadboard circuits and the Arduino is shown in Fig.31.

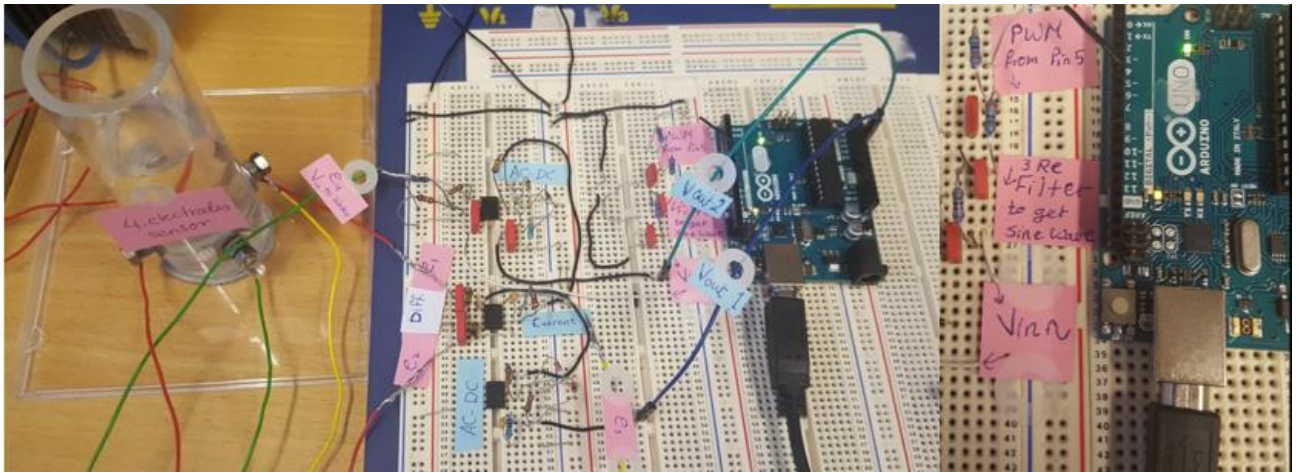


Figure 31: the structure of the experiments electrodes and the breadboard circuits with input signals generated by Arduino.

4.12.8 Testing the circuit:

After sending the generated signal ,of 1KHz and about 1.25V amplitude, to the saltwater, Fig.32 and 33 show the output V_{out1} and V_{out2} before and after the peripheral circuits.

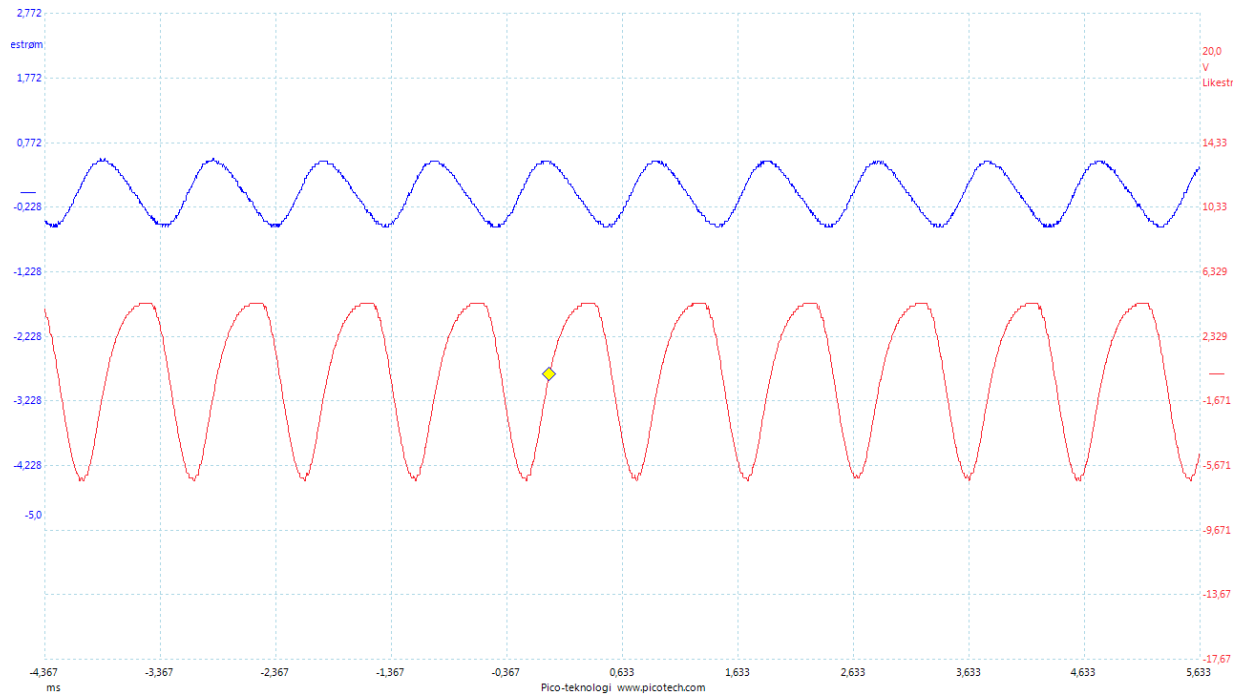


Figure 32 : V_{out1} (up) and V_{out2} (down) signals before the peripheral circuits

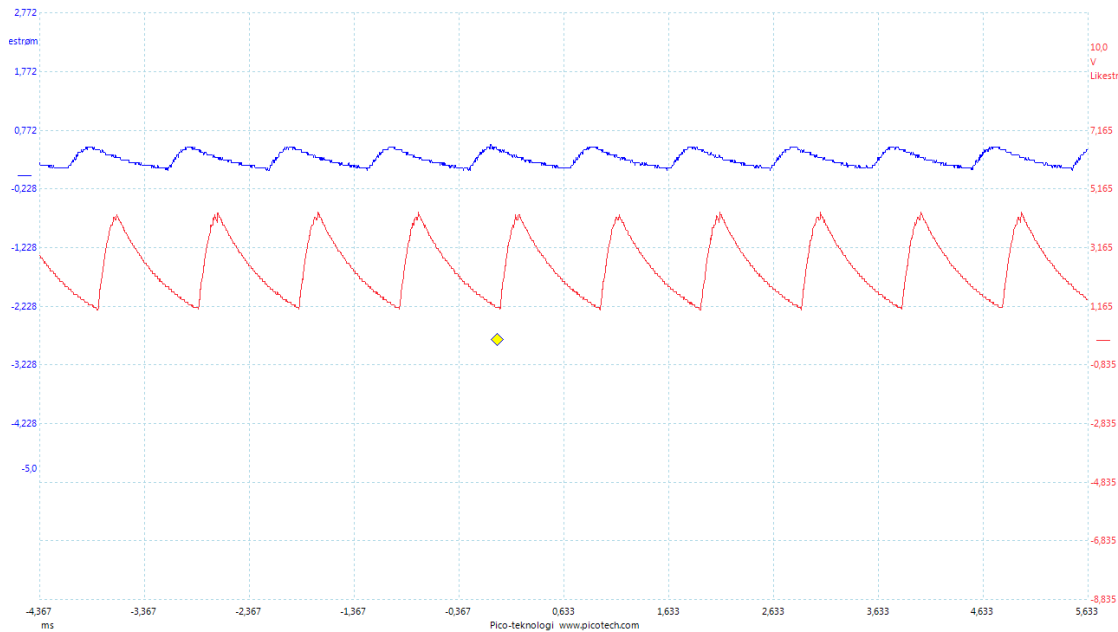


Figure 33: V_{out1} (up) and V_{out2} (down) signals after the peripheral circuits (the peak value)

The Arduino will pick up the peak value of the analogue input signals and by using the cell constant, which was determined by the calibration in a solution with known conductivity of 48,4 mS/cm , we can find the conductivity of any other solutions.

By testing in a saltwater solution of 35g/l salt the measured conductivity is :

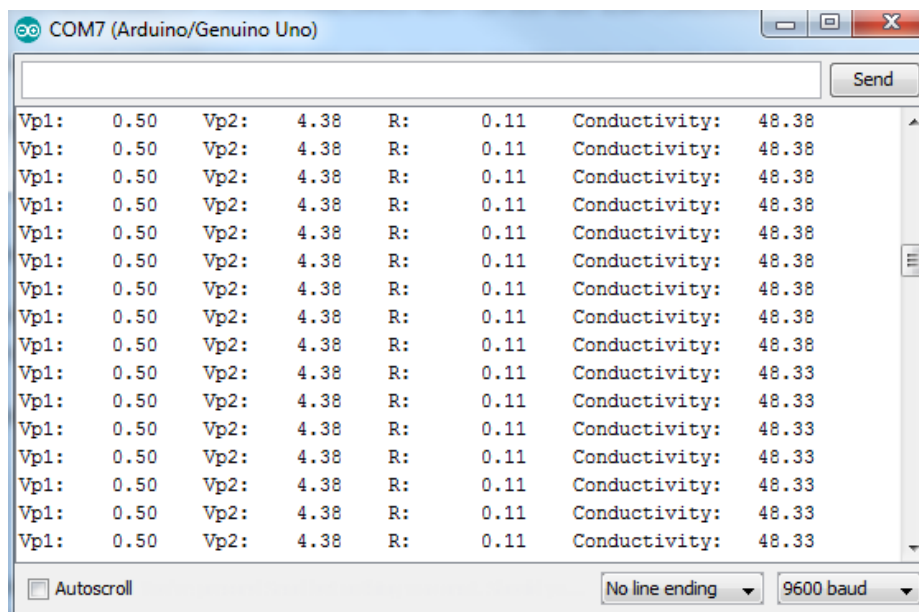


Figure34: The result of measuring the conductivity 35g/l saltwater solution. Vp1 and Vp2 are the peak value of Vout1 and Vout2.

The aim of this experiment was to check if the system that was described work well and can be used in our WPS. All of those tests shows at the system is somewhat useful to be used, so a printed circuit board (PCB) was designed for the breadboard circuits in order to have smaller area and can be easily used in our sensor. This will be discussed later.

4.13 Summery:

A 4-electrode conductivity system based on Arduino Uno, was built on breadboard and tested in saltwater and tap water to be the base of designing and building a conductivity meter. From the experiment and the measurements, it was clear that the conductivity in the saltwater is high, while it is very low in tap water. We saw that V_{out1} in tap water was greater than in saltwater where the voltage difference between electrode 1 and 2 is very small because of the ions that carry the electric current between the electrodes. While V_{out2} in the contrary was lower in tap water than saltwater because it is directly proportional with the current, which flows easily through the ionic solutions like saltwater. Fig.35 shows the 4-electrode conductivity meter based on Arduino Uno. The circuit is designed on printed circuit board (PCB) made for this project.

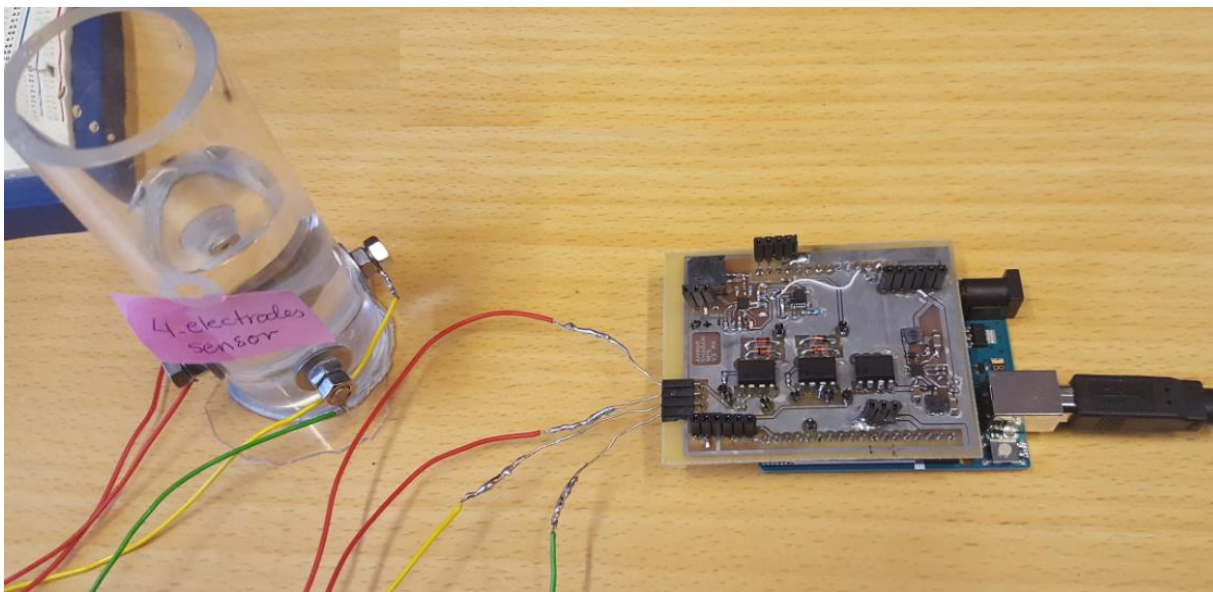


Figure 35: 4-electrodes conductivity meter.

5. Water parameters sensors

Sensors:

Nowadays sensors are widely used. They are used in everyday objects such as the touch screen of your mobile phone, in electric light control, smoke detectors and many other applications. In the broadest definition, a sensor is a device that receive a stimulus and responds with an electrical signal. The sensor is a special kind of transducers converts any type of energy into electrical energy. A thermopile, which converts heat into an electrical signal, is an example of a temperature sensor.

In this project, three sensors was used to measure the water parameters that are important to estimate the sound speed profile. These parameters are temperature, pressure (depth) and salinity (conductivity).

5.1 Temperature sensors:

There is many temperature sensors exist that can be connected to microcontroller to measure the temperature. All those sensors have individual characteristics like maximum and minimum temperature they can measure, accuracy, linearity, price and ease of use. When checking a temperature sensor for a project those characteristics must be considered. One should know what kind of sensors exists and what the differences are between those sensor.

The most popular temperature sensors that can be connected to microcontroller:

- Thermocouples
- RTDs (Resistive Temperature Detector)
- Thermistors
- Silicon junction temperature sensors

5.1.1 Thermocouple:

Thermocouple is the most widely used type of temperature sensor because of :

- Wide range temperature measurement from -220°C to 1800°C .
- Huge variety of their manufactured forms.
- Rugged
- Cheap.

Thermocouple is basically a junction of dissimilar metals. When two dissimilar metals junctions are held at steady but different temperatures, they produce a temperature dependent voltage. The higher the temperature the higher the voltage. This voltage can be interpreted to measure the temperature. Thermocouples usually generate voltage from one to few dozens of millivolts so an amplifier is needed to use the thermocouple sensors.

5.1.2 (RTD)Resistive temperature Detector

- Wide temperature range (-200°C to 800°C).
- High accuracy.
- Good long term- stability; they will not change their properties over time.
- Linear.
- Expensive.

RTD is a wire wound or a thin film of temperature dependent resistor. It is often made from platinum and therefore it is expensive. Because of the linear characteristic, they can easily find the temperature by multiplying their resistance by temperature coefficient. There are plenty of RTD Packages but they have smaller variety of packages than thermocouples.

5.1.3 Thermistors

Thermistors are thermal resistance like the RTDs. They can measure temperature from -100°C to 150°C. Unlike RTDs, they are non-linear. They have very close exponential characteristic. They can calculate the temperature from their resistance using a special formula or using resistance to temperature look up table.

There are two types of thermistors:

- **NTC** (Negative Temperature Coefficient): the resistance decreases with increasing temperature. This is the most widely used type of the thermistors because they are very cheap, come in small packages, very easy to connect to microcontroller. It is hard to get accurate temperature reading because of their non-linearity. Usually it is used for over temperature circuits protection.
- **PTC** (Positive Temperature Coefficient): the resistance increases with decreasing the temperature.

5.1.4 Silicon junction temperature sensors (PN junction temperature sensors)

They measure temperature using PN junction properties of transistors and diodes and they have amplifier built in so they can output signals with very usable amplitude.

- They have the smallest temperature range (-55°C to 155°C)
- High accuracy.
- Very easy to connect to microcontroller.
- Cheap.

There are two types of silicon junction sensors exist analog, like LM33 and LM62, and digital, like DS18B20 and LM70. Unlike analog sensors, digital sensors do not need to have on board ADC to be connected to microcontroller. They can be connected directly to digital pins and can be communicated with them by one of the serial protocols like I2c, SPI or one wire. Most of the digital temperature sensors can be connected in parallel so the temperature of multiple spots can be read with a just few pins of the microcontroller or even just one pin in case of using one wire interface.

In this project, we are using a digital silicon junction temperature sensor with one wire protocol. It is one of the most popular sensors of Maxim, It is DS18B20. This sensor is chosen because it is cheap, has high accuracy, simple, can be found manufactured as waterproof form and very easy to connect to Arduino microcontroller.

5.1.5 DS18B20: 1-Wire Digital Temperature Sensor [44]

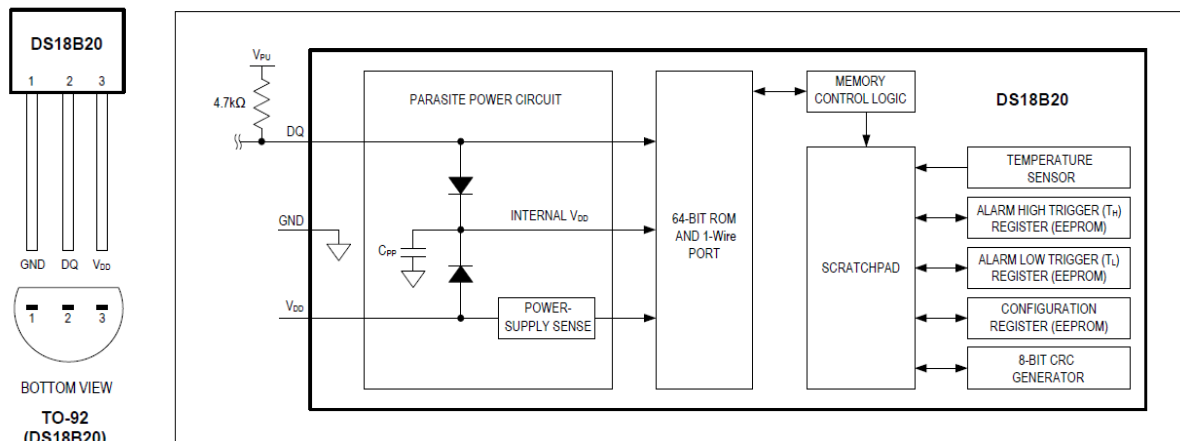


Figure 1: DS18B20 Pin configurations (left) and Block diagram (right).

The DS18B20 is a 1-Wire digital temperature sensor from Maxim IC. It is a very small simple device to measure the temperature between -55°C to 125°C . It provides 9 to 12-bit (configurable) temperature readings over a 1-Wire interface, so that only one wire (and ground) is needed for the communication. Multiple DS18B20s can exist on the same 1-Wire bus, because each DS18B20 sensor contains a unique silicon serial number. This is useful in measuring temperature in many different places. This feature is important in many applications like sensing temperature inside building or equipment, and process monitoring and control.

Features

- One -Wire Interface: only one data line and ground are needed to communicate with the microcontroller.
- Reduce Component Count with Integrated Temperature Sensor and EEPROM
 - Measures temperatures from -55°C to $+125^{\circ}\text{C}$ (-67°F to $+257^{\circ}\text{F}$)
 - $\pm 0.5^{\circ}\text{C}$ Accuracy from -10°C to $+85^{\circ}\text{C}$
 - Programmable Resolution from 9 Bits to 12 Bits
 - Converts 12-bit temperature to digital word in 750 ms (max.)
 - Requires no External Components.
- Parasitic Power Mode Requires Only 2 Pins for Operation (DQ and GND)
- Can be powered from data line. Power supply range is 3.0V to 5.5V.
- Simplifies Distributed Temperature-Sensing Applications with Multi-drop Capability.
- Each Device Has a Unique 64-Bit Serial Code Stored in On-Board ROM.
- Flexible User-Definable Non-volatile (NV) Alarm Settings with Alarm Search Command Identifies Devices with Temperatures Outside Programmed Limits
- Available in 8-Pin SO (150 mils), 8-Pin μSOP , and 3-Pin TO-92 Packages
- Supply voltage $V_{\text{DD}} = 3.0\text{V}$ to 5.5V .
- The DS18B20-PAR does not need an external power supply because it derives power directly from the data line (“parasite power”).
- operating current of the DS18B20 is up to 1.5 mA.
- Applications include thermostatic controls, industrial systems, consumer products, thermometers or any thermally sensitive system.

PIN DESCRIPTION (TO92):

GND - Ground

DQ - Data Input/output

VDD - No Connect. Power supply voltage

Overview

Fig.1 shows a block diagram and pin configuration of the DS18B20. The DS18B20 has five main data components: 1) 64-bit ROM, 2) the scratchpad memory, 3) temperature sensor, 4) non-volatile temperature alarm triggers TH and TL, and 5) a configuration register.

- The 64-bit ROM : stores the device's unique serial code.
- The scratchpad memory:
 - Contains the 2-byte temperature register that stores the digital output from the temperature sensor.
 - Provides access to the 1-byte upper and lower alarm trigger registers (TH and TL) and the 1-byte configuration register.
- The configuration register allows the user to set the resolution of the temperature-to digital conversion to 9, 10, 11, or 12 bits.
- The TH, TL, and configuration registers are non-volatile (EEPROM), so they will retain data when the device is powered down.

The DS18B20 uses Maxim's exclusive 1-Wire bus protocol. The 1-Wire protocol uses one control signal to implement bus communication. A weak pull-up resistor is needed at the control line since all devices are linked to the bus via the DQ pin. In this bus system, the microprocessor (the master device) uses device's unique 64-bit code to identify and address devices on the bus. Unlimited number of devices can be addressed on the bus because each one has a unique address code.

The master must provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. These commands operate on the 64-bit ROM portion of each device and can single out a specific device if many are present on the 1-Wire line as well as indicate to the bus master how many and what types of devices are present. After a ROM function sequence has been successfully executed, the memory and control functions are accessible and the master may then provide any one of the six memory and control function commands [45, p. 2].

One control function command instructs the DS18B20 to perform a temperature measurement. The result of this measurement will be placed in the DS18B20's scratch-pad memory, and may be read by issuing a memory function command which reads the contents of the scratchpad memory. The temperature alarm triggers TH and TL consist of 1 byte EEPROM each. If the alarm search command is not applied to the DS18B20, these registers may be used as general-purpose user memory. The scratchpad also contains a configuration byte to set the desired resolution of the temperature to digital conversion. Writing TH, TL, And the configuration byte is done using a memory function command. Read access to these registers is through the scratchpad. All data is read and written least significant bit first [45, p.2].

5.1.6 One-Wire interface

1-Wire is a communication system designed by Dallas Semiconductor Corp to interface simple sensors and devices with a single wire interface for both signalling and power. 1-Wire is similar in concept to I²C, but with lower data rates and longer range. One or several devices (**slaves**) can be connected to the 1-Wire bus at the same time. The DS18B20 is the slave. Only one **master** should be connected to the bus. The communication between the master and the slaves on the bus is initiated and controlled by the master. Transfer of data is only possible between master and slaves, so data cannot be transferred between slaves. Each 1-Wire slave device has a unique **64-bit ID** (identification number), which serves as device address on the 1-Wire bus. This address is factory programmed and unchangeable. The least significant 8 bits are a 1-Wire family code (DS18B20 code is 28h), a subset of the 64-bit ID, identifies the device type and functionality. The next least 48 bits give the serial number of the device. The most significant 8 bits give the CRC generated from the least 56 bits.

Typically, 1-Wire slave devices operate over the voltage range of 2.8V (min) to 5.25V (max). Most 1-Wire devices have no pin for power supply; they take their energy from the 1-Wire bus (parasitic powered). 1-Wire devices include an 800 pF capacitor to store charge, and to power the device during periods when the data line is active. When required, an extra wire can also be used to power up the slave devices (External powered).

Each device attached to the 1-Wire bus must have open drain or 3-state outputs. The 1-Wire port of the DS18B20 (DQ pin) is open drain with an internal circuit as shown in Fig. 2. The idle state for the 1-Wire bus is high. The 1-Wire bus requires a pull-up resistor of approximately 5 k Ω so that the bus is driven low if at least one device drives the bus low. This protocol enables the transfer of data between two devices on the bus while the other devices are in the idle state. An internal oscillator synchronized to the falling edge of the bus clocks each slave. Thus, the clock is not required for this protocol. In transferring, the least significant bit is transferred first.

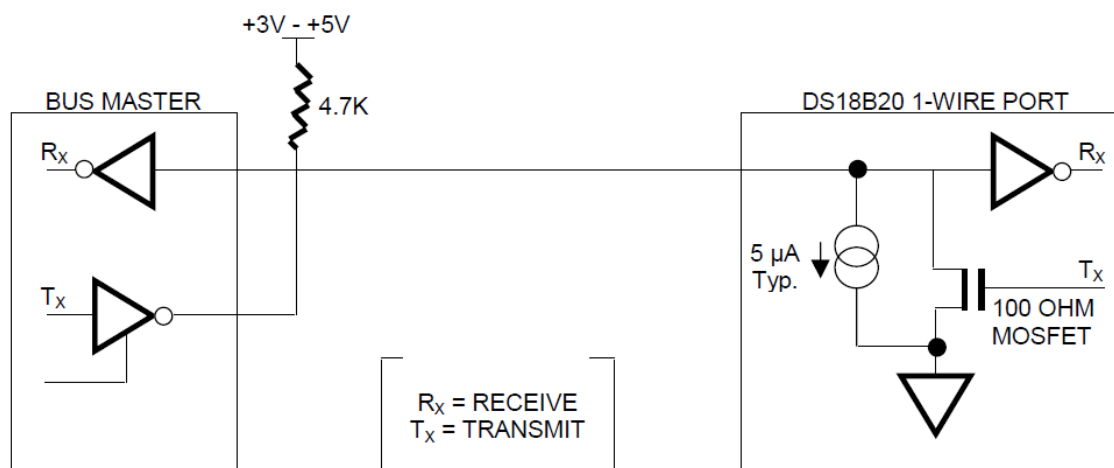


Figure 2: The hardware configuration of the 1-Wire bus system.[4]

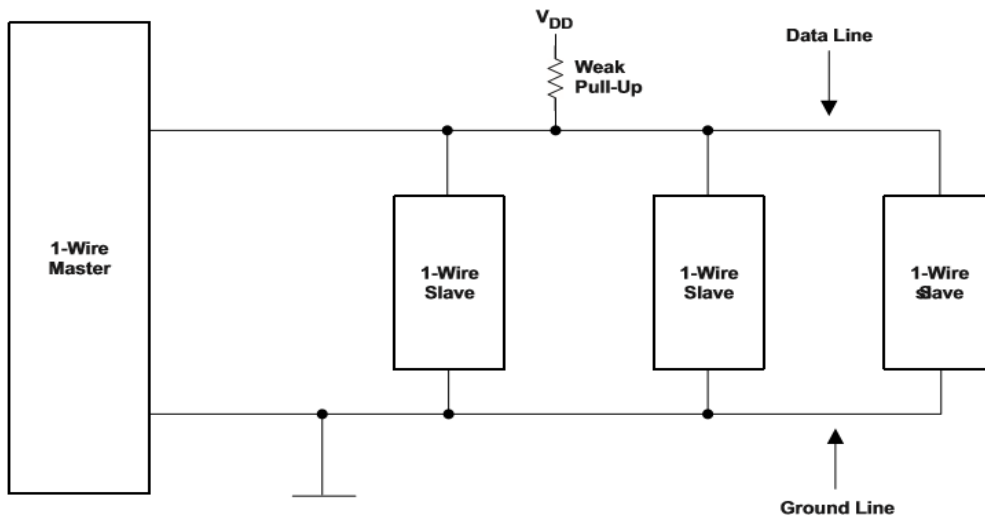


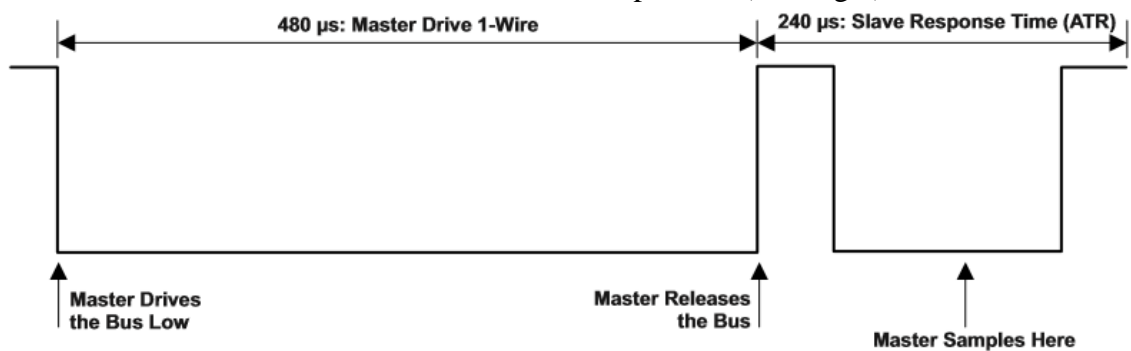
Figure 3: Bus Topology [8]

Signalling on 1-Wire [8]:

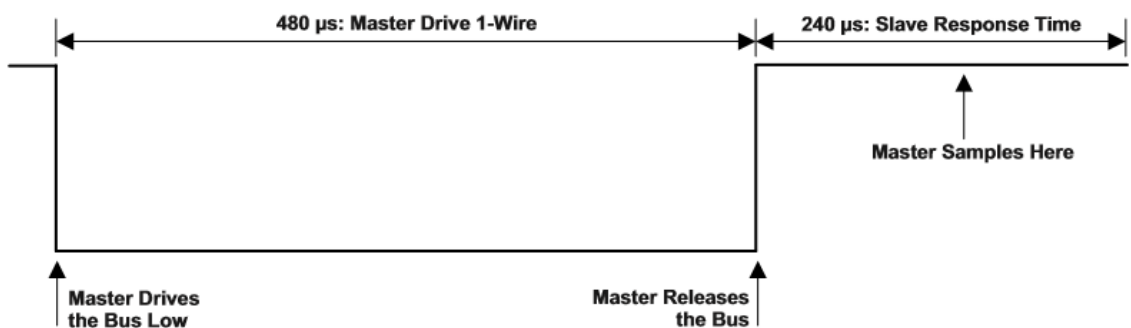
On data line, there are four types of signalling:

- 1) Reset Sequence with Reset Pulse and Answer to Reset (ATR):

The master drives the bus low for 480 μ s to reset all the slaves. The slaves answer to the reset (ATR) in the next 240 μ s by holding the line low. The master samples the bus, if the bus reads low, then at least one slave is present. (See Fig.4)



Reset Sequence Bus Timing When There is at Least 1 Slave on the Bus



Reset Sequence Bus Timing When There are no Slaves on the Bus

Figure 4: Reset sequence with and without ART. [8]

2) **Write 0 bit onto the bus:**

Send 0 bit to the 1-Wire slaves by driving the bus low for 60 us.

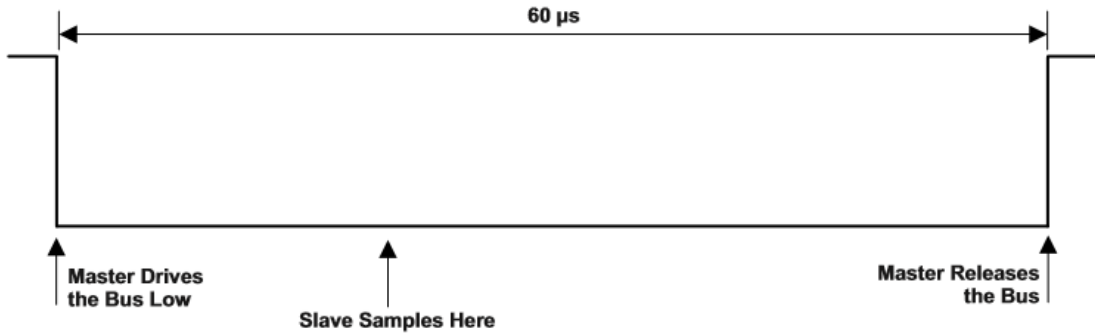


Figure 5: Write 0 Bus Timing

3) **Write 1 bit onto the bus:**

Send 1 bit to the 1-Wire slaves by driving the bus low for 15 us or less. Release the bus until 60 μs after the falling edge. The typically time is 60 us.

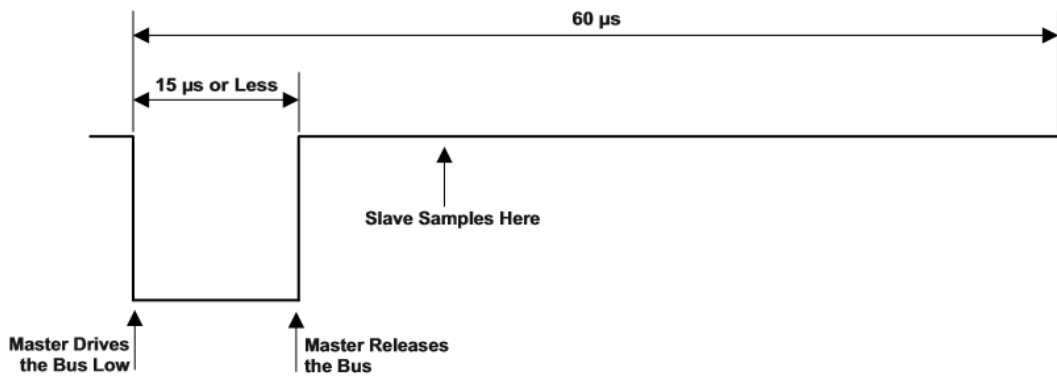


Figure 6: Write 1 Bus Timing

4) **Read bit :**

Read one bit from the slave. It is the same as write 1 bit, but here the master reads instead of writes. The master drives the bus low for 15 us or less and samples the bus after that to read the bit from the slave.

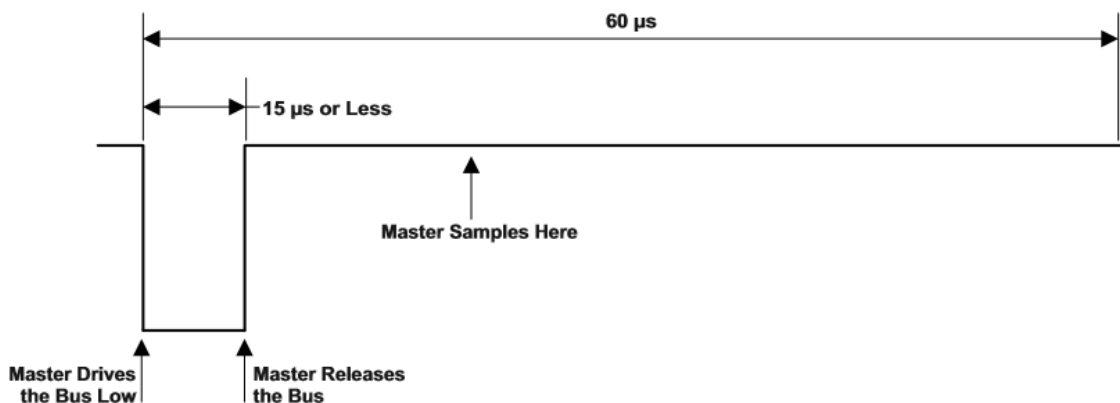


Figure 7: Read one Bus Timing

Typical communication flow on the 1-Wire bus [8]

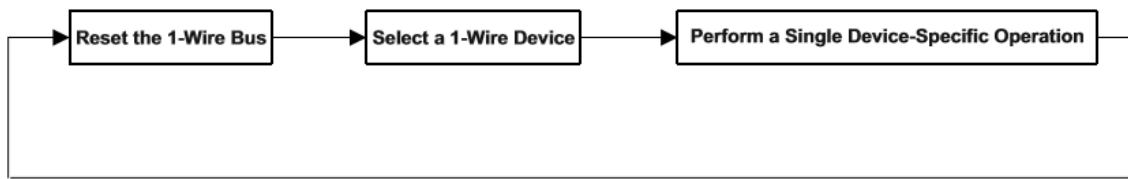


Figure 3: Typical communication flow on the 1-Wire bus.

As shown in Fig.8, the typical communication flow on the 1-Wire bus is summarized in the following steps:

- 1) Reset Sequence.
- 2) The master searches to detect the ROM number of the wanted slave.
- 3) Before performing any operation on the slave, it must be configured and/or selected using ROM commands.(1-Read ROM, 2- Match ROM, 3-Search ROM, 4- Skip ROM, or 5- Alarm Search.). All the ROM commands are 8 bit long.
- 4) After selecting the wanted device, a device specific operation can be performed
- 5) A reset pulse is sent after each operation.

5.1.7 DS18B20 with Arduino Uno to make a thermometer

To make a thermometer that can measure the temperature underwater we need:

- Waterproof DS18B20 sensor (see Fig.9- left).
- Arduino Uno board.
- One 4.7K Ω resistor.
- Breadboard and jumpers.
- Building the circuit as shown in Fig.9.

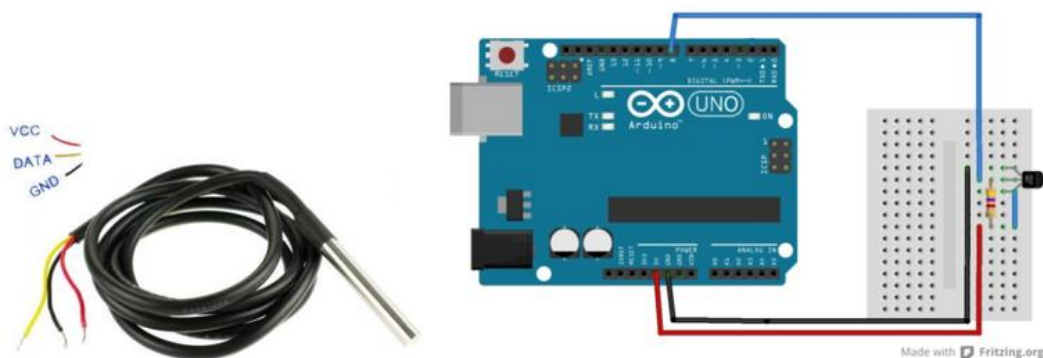


Figure 9: Waterproof DS18B20 (left). Building the circuit of the thermometer [53] (right)

To simplify the work, two libraries can be downloaded into the Arduino libraries folder. Libraries are collections of software functions geared towards a single purpose, such as communicating with a specific device. Arduino comes with a number of built-in libraries that help you do advanced tasks. The libraries we use to communicate with BS18B20 are :**1-wire**

bus [51], and Dallas Temperature [50] to do the required calculations. These two libraries must be included at the header of the code. The data pin (DQ pin) must be defined. Using the code in appendix A., we can measure the temperature.

5.2 Pressure

5.2.1 Background

The concept of pressure has its roots back to 1643. When Evangelista Torricelli, during his experiment with mercury-filled dishes, realized that, the atmosphere exerts pressure on Earth. In 1643, Blaise Pascal and his brother in law, Perrier, performed an experiment at the base and the top of the mountain Puy de Dome. He found that the pressure on the column of mercury depends on height. Pascal named mercury in vacuum instrument, which was used in the experiment, the barometer. The famous relationship: “*The product of the measures of pressure and volume is constant for a given mass of air at fixed temperature*”. Was stated by Robert Boyle in 1660. In 1738, Daniel Bernoulli developed an impact theory of gas pressure to the point where Boyle’s law could be deduced analytically. Moreover, Bernoulli predicted the Charles Gay Lussac Law when he stated that, at a constant volume, the pressure is increased by heating gas [42, p.375].

The pressure P is the force F acting perpendicularly on a unit area A of a boundary surface:

$$P = \frac{F}{A} \quad (5.1)$$

The SI unit of pressure is the Pascal (Pa). It is equivalent to one newton per m², (1 Pa = 1 N/m² = Kg.m⁻¹.s⁻²). That is one newton force is uniformly distributed over 1 m of surface. Another common used units like bar, ponds per square inch and hecto Pascal (hPa), atmosphere (atm).

$$1 \text{ bar} = 10^5 \text{ Pa}$$

$$1 \text{ atm} = 101325 \text{ Pa}$$

$$1 \text{ hPa} = 100 \text{ Pa}$$

The pressure in a static fluid comes from the weight of the fluid on a unit of area:

$$P = \frac{F}{A} = \frac{Mg}{A} = \frac{\rho Vg}{A} = \frac{\rho hgA}{A} = \rho hg \quad (5.2)$$

M: the mass of the fluid [Kg].

g : the acceleration of gravity [m/s²].

ρ : the density of the fluid [Kg/m³].

V: the volume of the fluid [m³].

h : the depth of the fluid [m].

From equation (5.2), the pressure exerted by a static fluid depends only upon the depth of the fluid (h), the density of the fluid (ρ), and the acceleration of gravity (g).

5.2.2 Pressure sensors [42]

A pressure sensor is a complex sensor. It requires more than one-step of the energy conversion to convert the pressure into electrical signal. Therefore, many pressure sensors operating system is based on converting a result of the applied pressure on a sensitive element with a defined surface area. Consequently, the element is deformed or displaced. Thus, the measuring of a displacement or deformation can be converted into electrical signal representing the pressure values. In pressure sensors, this deformable element is a mechanical device that changes its structure or form under pressure. Such device can be a wall of pressure chamber that is coupled to a strain sensor. The strain sensor converts the deformation of the chamber into electrical signals by means of the piezoresistivity. This is the basis of many pressure sensors. Nowadays, most of the pressure sensors are fabricated with silicon membranes by using micro-machining technology.

5.2.2.1 Piezoresistive effect and strain sensitivity

When the material is mechanically deformed, its electrical resistance changes. A mechanical deformation changes either the specific resistivity of or the geometric factor. This sensitivity to the strain is called the **piezoresistive effect**, and it is the principle of the strain sensor, which is basis of many pressure sensors. The applied stress, σ , is related to force as in the following equation:

$$\sigma = \frac{F}{A} = E \frac{dl}{l} \quad (5.3)$$

E : is Young's modulus of the material.

F : is the applied force on cross sectional area A .

$\frac{dl}{l} = e$: is called the strain: the normalized deformation of the material.

If, for example, a cylinder of volume V and length l is exposed to a force F , the volume of the material will stay constant, while the length l and the cross sectional area A will be changed. As a result the resistance of the cylinder, which depends on material resistivity, ρ , l and A , will be changed.

$$R = \rho \frac{l}{A} \quad , \quad A = \frac{V}{l} \quad \rightarrow \quad R = \frac{\rho}{V} l^2 \quad (5.4)$$

$$\frac{dR}{dl} = 2 \frac{\rho}{V} l \quad (5.5)$$

From equation (5.5) the sensitivity becomes higher for longer and thinner cylinder with high resistivity.

Rewriting equation (5.5) using $A = \frac{V}{l}$ we get:

$$\frac{dR}{R} = 2 \frac{dl}{l} = S_e e \quad (5.6)$$

Normalized incremental resistance of the strained wire is a linear function of strain, $e = \frac{dl}{l}$, as shown in equation (6). S_e is known as the gauge factor of resistivity of the strain gauge element. For metallic wires S_e ranges from 2 to 6. For semiconductor S_e is much higher. It is between 40 and 200, because in the semiconductors, the geometry factor plays much smaller role than change in the specific resistivity due to deformation of the crystalline structure of the material [42, p.84].

5.2.2.2 Piezoresistive Pressure sensors

To make a pressure sensor, two important component are needed. The components are: a membrane of known area (A) and a detector that responds to applied force like a piezo resistive gauge resistor. These components can be fabricated of silicon.

When pressure is applied to semiconductor resistor having initial resistance R, the resistance will be changed by ΔR . We discussed this piezoresistive effect in the preceding section.

$$\frac{\Delta R}{R} = \pi_l \sigma_l + \pi_t \sigma_t \quad (5.7)$$

π_l = the piezoresistive coefficient in the longitudinal direction.

π_t = the piezoresistive coefficient in the transverse direction.

σ_l = the stress coefficient in the longitudinal direction.

σ_t = the stress coefficient in the transverse direction.

The π -coefficients depend on the orientation of resistors on silicon crystal. [1, p. 381]. A change in resistivity is proportional to applied stress and subsequently to applied pressure. The silicon resistors exhibit quite strong temperature sensitivity. Thus, temperature compensation is necessary.

Rather than piezoresistive pressure sensors, there are many other sensors based on applied force into an area, for example:

- **Capacitive sensors:**

A variable capacitor is created by using a diaphragm and pressure cavity. This will help in detecting strain caused by applied pressure, capacitance decreasing as pressure deforms the diaphragm.

- **Piezoelectric sensors**

Uses the piezoelectric effect in certain materials such as quartz to measure the strain upon the sensing mechanism due to pressure. This technology is commonly employed for the measurement of highly dynamic pressures. The piezoelectric effect is generating of electric charge by subjecting certain materials to stress.

- In addition, many other types of sensors like electromagnetic sensors, optical sensors, etc.

5.2.3 MS5805-14BA Pressure Sensor [55]

The MS5803-14BA is a new generation of high-resolution pressure sensor with SPI and I2C bus interface. It is perfect for depth measurement systems with a water depth resolution of 1cm and below. The sensor module is based on MEMS technology and includes a high linear pressure sensor and an ultra-low power 24 bit $\Delta\Sigma$ ADC with internal factory calibrated coefficients. It provides a precise digital 24 Bit pressure and temperature value. This sensor can be interfaced to any microcontroller. The gel protection and antimagnetic stainless steel cap protects against 30-bar overpressure water-resistant.

Features

- High-resolution module, 0.2 mbar.
- Fast conversion down to 1 ms.
- Low power, 1 μA (standby $< 0.15 \mu\text{A}$)
- Integrated digital pressure sensor (24 bit $\Delta\Sigma$ ADC)
- Supply voltage 1.8 to 3.6 V
- Operating range: 0 to 14 bar, -40 to +85 °C
- Max response time about 8.22 ms
- I²C and SPI interface (Mode 0,3)
- No external components (Internal oscillator)
- Excellent long term stability
- Hermetically sealable for outdoor devices
- Applications include mobile water depth measurements system, Diving computers and adventure or multi-modes watches.

Overview

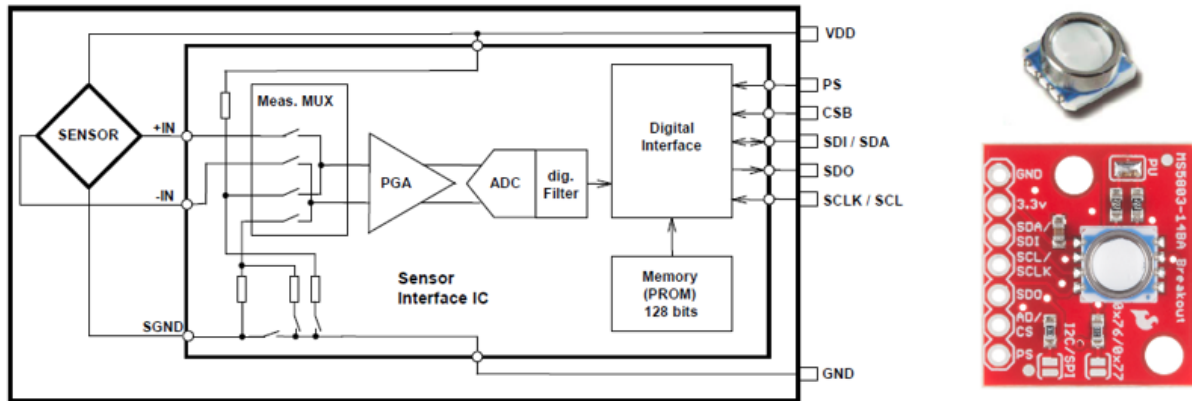


Figure 4: MS5803-14BA and its Block diagram.

Fig. 10 shows the block diagram of the MS5803-14BA. It consists of a piezoresistive pressure sensor and IC sensor interface. The MS5803-14BA converts the uncompensated analogue output voltage from the piezoresistive pressure sensor to a 24 bit digital value, and provides 24 bit digital value of the temperature of the sensor.

The 123-bit PROM memory is used to store the 6 calculated coefficients (W_1 to W_6) necessary to compensate for process variations and temperature variations of each module. Every module is individually calibrated at two temperature and two pressures.

The sensor has two types of serial interfaces I^2C and SPI. High PS Pin activates I^2C protocol. Low PS Pin activates SPI protocol. The user can choose which protocol to use. The pins needed for I^2C protocol are SDA (Serial Data), SCL (Serial Clock) and CSB (Chip Select). The pins needs for SPI protocol are SDI (Serial Data In), SDO (Serial Data Out), SCLK (Serial Clock) and CSB (Chip Select).

Table 6 : The interface modes of the MS5803 sensor.

	SPI MODE	I^2C MODE
Pin PS	Low	High
Pins used	SDI, SDO, SCLK, CSB	SDA, SCL, CSB
MCU clocks in the data through:	SCLK, SDI (input)	SCLK, SDA
The sensor response on:	SDO (output)	SDA
CSB	Is used to enable/disable the interface, so that other devices can talk on the same SPI bus.	The complement of the pin CSB represents the LSB of the I^2C address. Must be connected to VDD or GND.

COMMANDS

The MS5803-14BA has only five basic commands of one-byte size each:

1. Reset (0x1E)
2. Read PROM (128 bit of calibration words) (0xA0 to 0xAE)
3. D1 conversion
4. D2 conversion
5. Read ADC result (24 bit pressure / temperature) (0x00)

After ADC read commands, the device will return 24 bit result and after the PROM read 16bit result. The address of the PROM is embedded inside of the PROM read command using the a2, a1 and a0 bits.

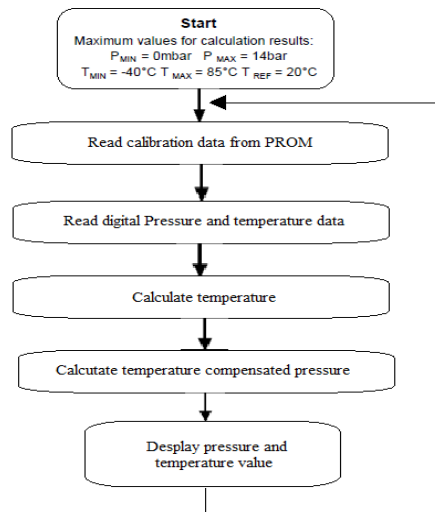


Figure 11: Flow chart for pressure and temperature reading and software compensation.

5.2.4 I²C Interface:

Since we use the I²C interface in this project to communicate with our MCU, we will talk more about it. The Inter-integrated Circuit (I²C) Protocol is a protocol intended to allow multiple “slave” digital integrated circuits (“chips”) to communicate with one or more “master” chips. It only requires two signal wires (SDA and SCL) to exchange information, while SPI needs 4 wires and each additional slave requires one additional CS pin on the master. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Unlike the SPI, the I²C can allow multiple masters. Most I²C devices can communicate at 100 kHz or 400 kHz. The clock signal (SCL) is always generated by the current bus master; some slave devices may force the clock low at times to delay the master sending more data.

Unlike UART or SPI connections, the I²C bus drivers are “open drain”, meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the

system. As shown in Fig.12, each signal line has a pull-up resistor on it, to restore the signal to high when no device is asserting it low [56].

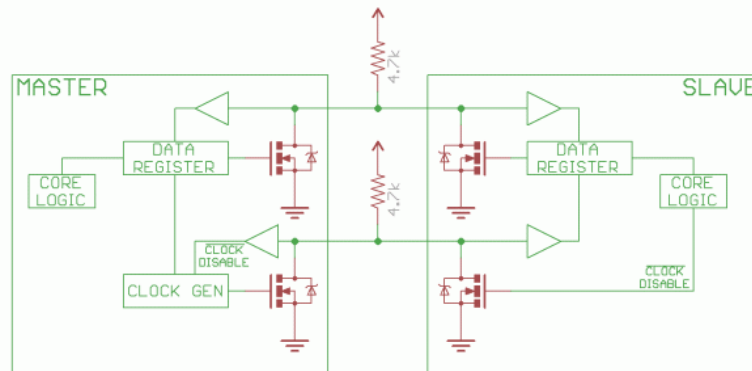


Figure 5: I2C Protocol communication [15]

Communication via I²C is more complex than with a UART or SPI solution. The signaling must adhere to a certain protocol for the devices on the bus to recognize it as valid I²C communications.

There are two types of message frame in I²C Protocol:

- **Address frame:** where the master chooses the slave to which the message is being sent. It is always first in any new communication sequence. For a 7-bit address, the address is clocked out most significant bit (MSB) first, followed by a R/W bit indicating whether this is a read (1) or write (0) operation. The 9th bit of the frame is the NACK/ACK (Non-Acknowledge/Acknowledge) bit. The receiving slave must pull the SDA low before the 9th pulse to indicate that it has received the data or the exchange halts.

The MS5803-14BA address is 111011Cx, where C is the complementary value of the pin CSB.

- **Data frame:** It is 8-bit data message from the master to the slave or from the slave to the master. The data can be transmitted after the address frame has been sent. The master will simply continue generating clock pulses at a regular interval, and either the master or the slave, depending on whether the R/W bit, will place the data on SDA. After SCL goes low, the data is placed on the SDA. After the SCL goes high, the data is sampled.

Each I²C message starts with **start condition** and ends with **stop condition**. In the start condition, the master leaves the SCL high and pulls the SDA low to initiate the address frame. This will notify all the slaves that the transmission is going to start. If there is more than one master device wants to use the bus at the same time, the one that pulls the SDA first can control the bus. The master generates the stop condition when all the data have been sent. The stop condition finds its place, when low to high transition on SCL followed by low to high transition on SDA, with SCL staying high.

The MS5803 I²C commands [55, p.11]:

- **Reset sequence:**
The reset can be sent any time when the power is on.

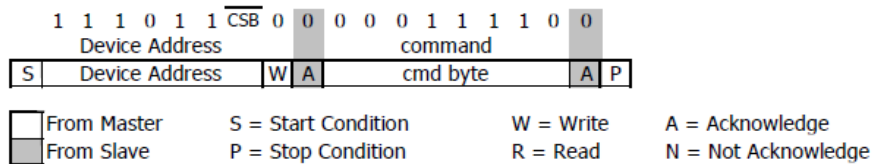
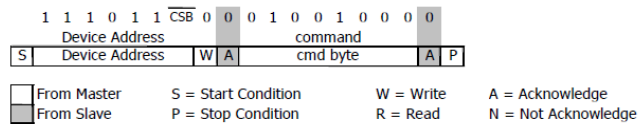


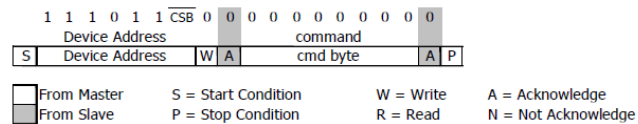
Figure 6: I²C Reset command [14, p.11]

- Conversion Sequence :**

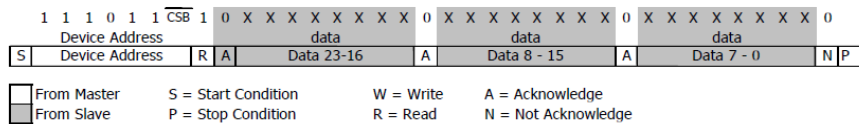
The conversion command is used to initiate uncompensated pressure (D1) or uncompensated temperature (D2) conversion. A conversion can be started by sending the command to MS5803-14BA. When command is sent to the system, it stays busy until conversion is done. When conversion is finished the data can be accessed by sending a Read command, when an acknowledge appears from the MS5803-14BA, you may then send 24 SCLK cycles to get all result bits. Every 8 bit the system waits for an acknowledge signal.



I²C Command to initiate a pressure conversion (OSR=4096, typ=D1)



I²C ADC read sequence



I²C pressure response (D1) on 24 bit from MS5803-14BA

Figure 7: Conversion Sequence

- PROM Read Sequence**

It consists of two parts. 1) sets up the system into PROM read mode. 2) Get the data from the system.

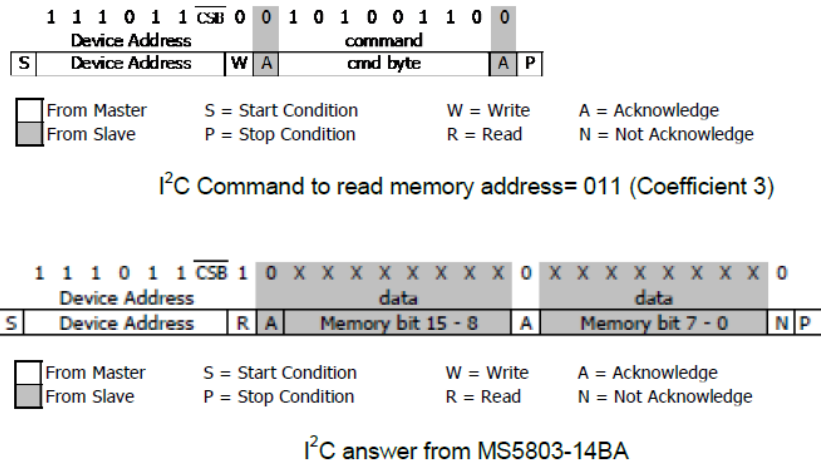


Figure 8: PROM Read Sequence

5.2.5 MS5803 with Arduino Uno to measure the pressure and the depth underwater.

To measure the pressure and depth underwater, we used the MS5803-14BA pressure sensor breakout from SparkFun with I²C interface to Arduino Uno. They are connected together as shown in Fig.16. (SDA→A₄, SCL → A₅, VCC→ 3.3v, GND→GND)

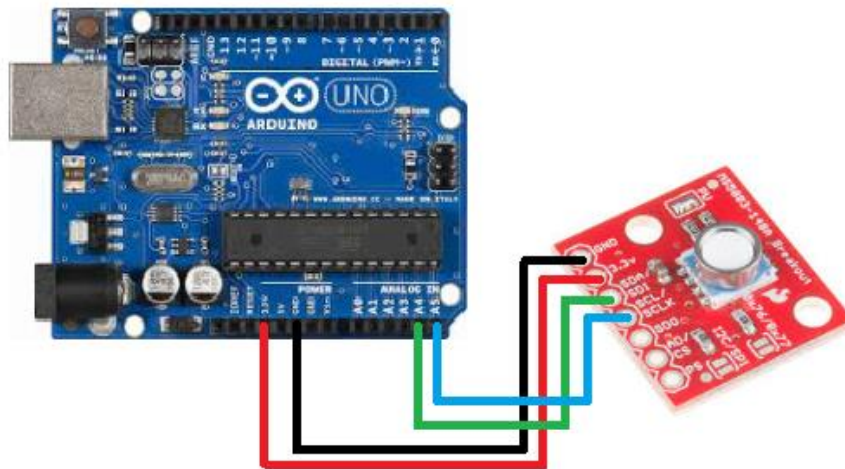


Figure 9: Connecting the MS5803- 14BA board to the Arduino.

An Arduino library called SFE_MS5803_14BA_I2C has been written by Sparkfun to allow us to easily talk to the MS5803 sensor. This library was installed to the Arduino libraries folder, from [16]. Moreover, the built-in Arduino library, Wire.h, is required to complete the I²C transaction. Using an example sketch, we could read the pressure in a bowl of water at different depths.

The Pressure/depth sensor was tested in a tank at the lab from 0 to 0.6 meter and in situ at a depth of 15 meter. According to the sensors specification its accuracy is less than +/-20mBar

for pressures down to 6 Bar. Measured pressure was converted into water depth by rewriting equation (5.2) after adding the atmosphere pressure. This gives us the depth

$$Depth = \frac{P - P_0}{\rho g} \quad 5.8$$

where $P_0=101325$ Pa is the atmospheric pressure at the surface, $\rho=1030\text{kg/m}^3$ the water density and $g=9.8$ m/s the gravity. The following data was measured:

Depth(m) (Ruler)	P (measured)(Pa)	Depth(m) Estimated	Depth(m) difference.	Estimated P
0.00	101100	-0.02	0.02	101325
0.035	101470	0.01	0.02	101678,3
0.06	101600	0.03	0.03	101930,6
15.00	252990	15.03	-0.03	252735

Table 7: Measured and estimated depths and pressure from the MS5803 pressure sensor.

The measurements show that the pressure sensor predicts the depth with an accuracy of +/- 0.03m relative to the depth measured by the rulers. The pressure was also estimated from the depth and we get pressure accuracy of about +/- 3mbar.

5.3 Salinity sensors

Salinity is the total concentration of dissolved salts in the water. There are two main methods of determining the salt content of water: **Total Dissolved Salts** (or Solids) and **Electrical Conductivity (EC)**. Total Dissolved Salts (TDS) is measured by evaporating a known volume of water to dryness, then weighing the solid residue remaining. This method is tedious and cannot be carried out in the field. The EC is explained and discussed in the previous chapter. EC measurement is much quicker and simpler and is very useful for field measurement. Measurements of EC can be used to give an estimate of TDS.

A **Refractometer** can be used to measure the salinity. The refractometer measures how much light bends or refracts, when it enters the liquid. The more salts (and other material) dissolved in the water, the more resistance the light will meet and the more it will bend.

The salinity can be measured using a **hydrometer**. The hydrometer measures the specific gravity of water, or its density compared to pure H₂O. Because almost all types of salt are denser than water, a hydrometer reading can tell you how much salt is present [58].

In this project, we have designed 4-electrodes conductivity meter based on Arduino Uno to estimate the salinity, as it is discussed in the previous chapter.

Atlas Scientific [18] makes high-quality sensors for environmental monitoring available to everyday hackers and makers. All of their kits are easy to calibrate and connect to your microcontroller-based project. The Atlas Scientific EZO Conductivity Kit [60] is a small footprint computer system that is specifically designed to be used in robotics applications where the embedded systems engineer requires accurate and precise measurements of

Electrical Conductivity (EC), Total Dissolved Solids (TDS), Salinity and Specific Gravity (SG) of water solvents. This is just an example for more information see the data sheet [61].

5.4 Memory:

The microcontroller ATmega328, used in Arduino Uno board, has three pools of memory:

- Flash memory (program space), 32K bytes, is where the Arduino sketch is stored.
- SRAM (static random access memory), 2K byte, is where the sketch creates and manipulates variables when it runs.
- EEPROM, 1K byte, is memory space that programmers can use to store long-term information.

The data that measured by the sensors under the water must be stored somewhere. Arduino Uno has no enough memory to store the data. Therefore, an external SD-card is required to save the data until the instrument is out from the water and so send the data to PC, any other receiver, or even reading the data directly from the SD-card. **SparkFun OpenLog** seems to be useful to our project. It is an open source data logger that works over a simple serial connection and support SD-card up to 64 GB.

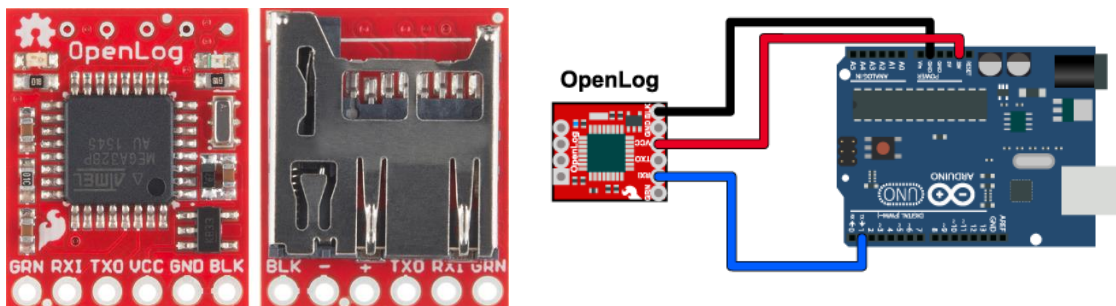


Figure 10: SparkFun Openlog

The OpenLog can log all the serial data that our project generates to a microSD-card.

The SparkFun OpenLog runs off an onboard ATmega328, running at 16MHz thanks to the onboard crystal. The OpenLog average current draws is 5mA. All data logged by the OpenLog is stored on the microSD card that involves the features of 64MB to 64GB capacity and FAT16 or FAT32 file type.

Features:

- VCC Input: 3.3V-12V (Recommended 3.3V-5V)
- Log to low-cost microSD FAT16/32 cards up to 64GB
- Simple command interface
- Configurable baud rates (up to 115200bps)
- Preprogrammed ATmega328 and bootloader
- Four SPI pogo pins
- Two LEDs indicate writing status

- 2mA idle, 6mA at maximum recording rate

Two questions we have to find answer to before deciding to use the OpenLog:

- 1) If we are, for example, taking measurements every 2 seconds, can we write data to the SD-card that quickly?
- 2) If we are making a log every second, will we have enough space on the memory to capture data for the entire flight?

First, the right speed:

The data is added to the card at speed of 9600 bits/s. every 8 bits equal one byte. This means at $9600/8 = 1200$ bytes/s is added to the SD-card. Is this enough to our use?

Let us say that, for example, we measure the water parameters every 20 cm until 30m in 2 min. This means the sensors reads data 150 times in 2 min, in rate one read per 0.8 s. Is this enough speed to make the measurements? The characters, numbers, commas and periods are translated into bytes (1 characters = 1 byte). As long as we type < 1200 character/s, it is OK. An approximation of how many bytes do we have in each measurement gives 36 byte per 0.8s which is much lower speed than 1200 byte/s.

Second, how many log entries can be stored at the SD-card?

1 Log entry = 40 bytes. If the SD-card has 8 GB store space, and we make one log/s, it will take $8GB/40 \text{ byte} = 200$ million log entries. Which means we need 200 million seconds to fill the SD-card = 6.3 years.

6. Wireless data transfer:

Wireless communication with Arduino can be achieved using:

- WiFi
- ZigBee
- Wireless inventors Shield
- 433 MHz RF transmitter and receiver module
- XBee
- Bluetooth

6.1 Bluetooth

In this project, we use Bluetooth to transfer data. **Bluetooth** is a standardized protocol for sending and receiving data via a 2.4GHz wireless link. It is a secure protocol, and it is perfect for short-range, low-cost, low power, wireless transmissions between electronic devices. It is perfectly suited as a wireless replacement for serial communication interfaces.

There are many types Bluetooth modules can be use. For example, HC-05, HC-06, BlueSMiRF, JY-MCU and Blue Fruit EZ-Link.

Bluetooth has a short range but it is quite enough in our application. There are many applications on the smartphones that able the Arduino to communicate with the phone via Bluetooth. This means that we can use the smartphone as a receiver to the data from our instrument after coming out of the water. This can be very helpful at the cases where it is difficult to use the PC.

The Bluetooth modules that is used here is the BlueSMiRF Silver. It is the latest Bluetooth wireless serial cable replacement from SparkFun Electronics. BlueSMiRF can both send and receive data.

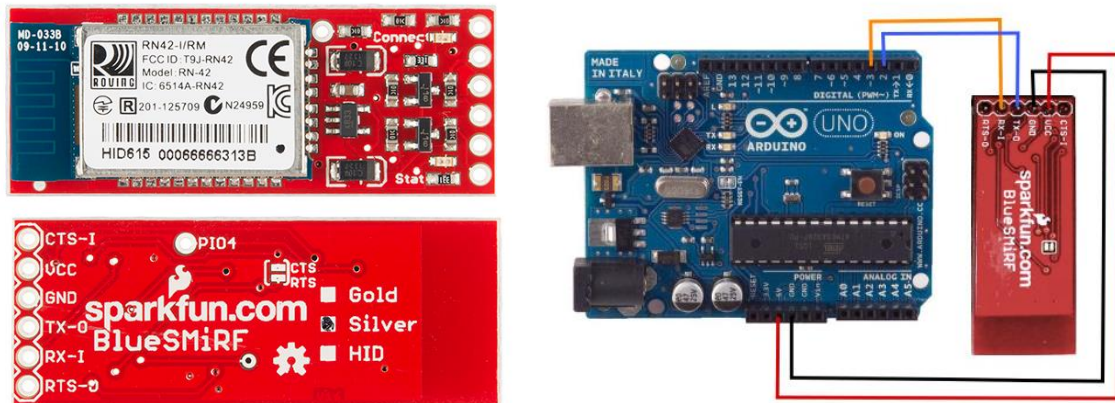


Figure 18: BlueSMiRF silver.

Specifications:

- v6.15 Firmware
- FCC Approved Class 2 Bluetooth Radio Modem
- Extremely small radio - 0.15x0.6x1.9"
- Very robust link both in integrity and transmission distance (18m)
- Hardy frequency hopping scheme - operates in harsh RF environments like WiFi, 802.11g, and ZigBee
- Encrypted connection
- Frequency: 2.402~2.480 GHz
- Operating Voltage: 3.3V-6V
- Serial communications: 2400-115200bps
- Operating Temperature: -40 ~ +70C
- Built-in antenna

6.2 How Bluetooth works?

When any two devices want to communicate with each other, they have to agree number of points before starting the conversation. They have to agree for example, how much data will be sent at a time, how they will speak to each other. A set of commands and responses must be developed. By other words, they need a protocol to communicate with each other.

Bluetooth devices use radio waves instead of wires or cables to communicate with each other. The Bluetooth protocol operates at 2.4GHz in the same unlicensed ISM (Industrial, Scientific and Medical) frequency band where RF (Radio Frequency) protocols like ZigBee and WiFi also exist. Bluetooth is like a RF version of serial communication.

When two Bluetooth devices want to talk to each other, they need to pair. Communication between Bluetooth devices happens over short-range, forming networks known as **piconets**. A piconet is a network of devices connected using Bluetooth technology. Piconets use master and slave model to control the communication between devices. A single master device can be connected to maximum seven slave devices. The slave device can be only connected to a single master. The master can send data to its slaves and request data from them. The slaves can just send to and receive data from the master.

Every single Bluetooth device has a unique 48-bit address. It can also have a user name given by the user to identify the devices.

There are two modes for the Bluetooth module: command mode and data mode (default). In data mode the module operates as a pipe. The data that are received by the module, it strips the Bluetooth header and trailer and passes the user data to the UART port. The data is written to the UART, and then the module constructs the Bluetooth packets and sends it out over the Bluetooth wireless connection. The command mode is used to configure the Bluetooth characteristics like the device name, baud rate, PIN code and data rate. See Fig. 19.

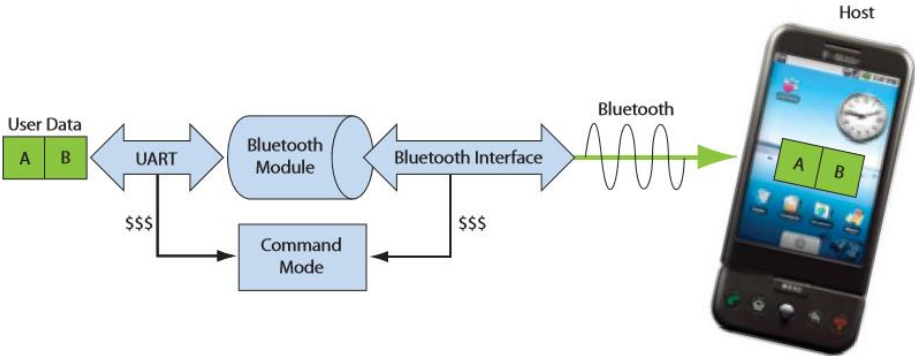


Figure 19: Bluetooth data and command modes.[24]

6.3 Testing the Bluetooth BlueSMiRF module:

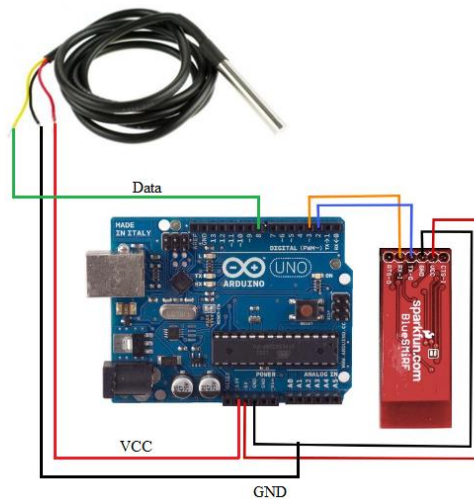


Figure 20: BLueSMiRF connected to Arduino to send temperature data

As shown in Fig. 20, the Arduino is connected to DS18B20 temperature sensor and Bluetooth module to test wireless sending of temperature data. The receiver here is a smartphone with special application called Bluetooth terminal. This application allows the smartphone to receive data from Arduino via Bluetooth. It can also send the received data as e-mail to the PC or share them by another ways. Fig. 21 shows an example of received data by smartphone from the Arduino using BlueSMiRF. This data was received and stored in Dropbox by the help of Bluetooth terminal application.

BtTemp - Notisblokk				
File	Rediger	Format	Vis	Hjelp
[12:10:58.523]	>>	23.81		
[12:10:59.310]	>>	23.87		
[12:11:00.091]	>>	23.87		
[12:11:00.871]	>>	23.87		
[12:11:01.667]	>>	23.87		
[12:11:02.435]	>>	23.87		
[12:11:03.246]	>>	23.94		
[12:11:04.013]	>>	23.94		
[12:11:04.825]	>>	23.94		
[12:11:05.606]	>>	23.94		
[12:11:06.407]	>>	23.94		
[12:11:07.173]	>>	23.94		
[12:11:07.983]	>>	24.00		
[12:11:08.773]	>>	24.00		
[12:11:09.586]	>>	24.00		
[12:11:10.330]	>>	24.00		
[12:11:11.115]	>>	24.00		
[12:11:11.926]	>>	24.00		
[12:11:12.705]	>>	24.06		
[12:11:13.503]	>>	24.06		
[12:11:14.302]	>>	24.06		

Figure 11: Temperature data received by smartphone via Bluetooth.

6.4 Software

Tera Term and RealTerm

Tera Term is an open-source, free, software implemented, and popular windows terminal program. It is simple to use and one of the best terminal options.

Realterm is an engineers terminal program specially designed for capturing, controlling and debugging binary and other difficult data streams. It is the best tool for debugging comms.

We use this software to communicate with Arduino board via Bluetooth and read sensors.

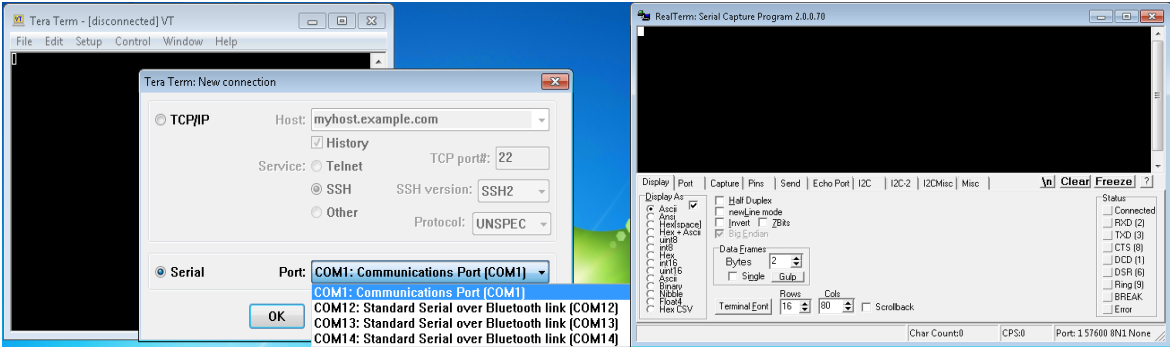


Figure 12: (left) Tera Term, (right) RealTerm windows

7 System Design

The goal of this project is to build a sound speed profiler which we called WPS. By measuring T, P and S in the water can WPS estimate the sound speed profile in the water. After choosing and preparing the required sensors and components, we can now design the system of the WPS. As shown in Fig. 1, the WPS consists of three sensors that requires interface protocol in order to communicate with the microcontroller. The microcontroller receives the data from the sensors, analyses it and saves it in the SD-card or sends it through Bluetooth module to the receiver.

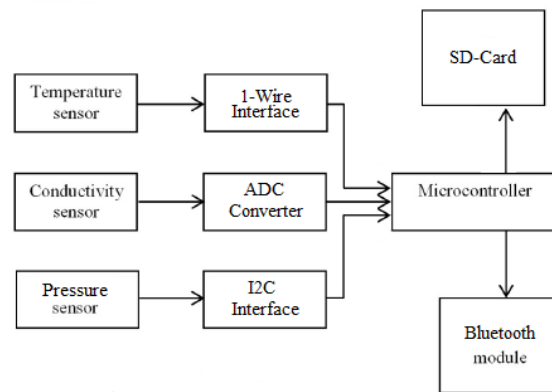


Figure 1: The block diagram of the system.

Previously, we talked in details about each sensor and its interface to Arduino Uno board. The main reason to choose the Arduino Uno board is that it is simple and popular board. It is the best board to get started with when one has poor background of microcontroller programming as I do. Very easy to find tutorials and videos about almost anything you wonder about.

The idea is to design a printed circuit board (PCB) that contains all the required electrical components and contacts pins. This PCB works as an Arduino Shield that it has contact pins in the bottom to be connected to the Arduino Board.

7.1 PCB:

The printed circuit board was created using CAD Star, and manufactured by Elab in physics institute. The production of the PCB was also done in the Elab. The components were soldered on the board by hand. Arrays of pin were soldered on the board to make it easier to connect the sensors or other small boards like the Bluetooth module and the SD-Card reader. Fig. 2 shows the WPS board, that was made, connected to Arduino Uno.

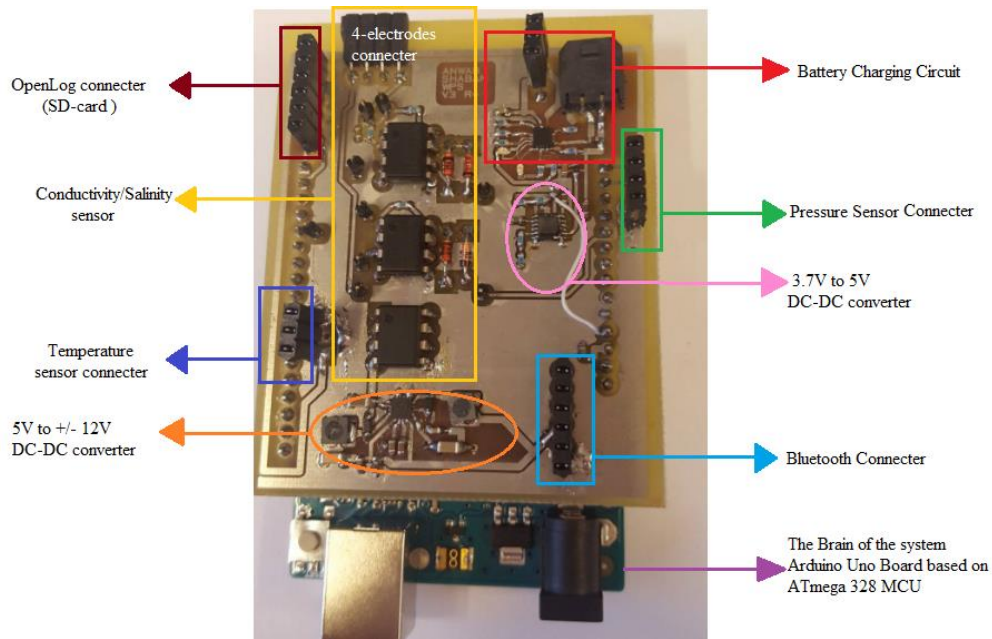


Figure 13: The PCB of the Water Profile Sensor

7.2 The production of PCB board in Elab:

I was somewhat curious to see how they produce the PCB board in the Elab. Thanks to Erlend he let me be with him while he was producing the PCB of my sensor.

To produce the PCB they use FR4 PCB laminate coated with positive photoresist. The following steps summarize the production of the board:

- 1) The outlay is printed in 1:1 format on transparent plastic foil.
- 2) The plastic foil is laid over the PCB laminate.
- 3) The photoresist not covered by the pattern on plastic foil is UV illuminated for 1 minute and 40 seconds.
- 4) Developer is used to remove the photoresist that have been rayed.
- 5) In etching machine the PCB is showered with sodium persulfate until the unprotected copper is etched away.
- 6) A stripper is used to remove the photoresist which were not UV investigated and that has protected the copper will be kept through the etching process.
- 7) Immersion tin is poured over the card to get a surface that does not oxidize, and that makes the card easier to solder on.
- 8) Finally drilling up the holes.

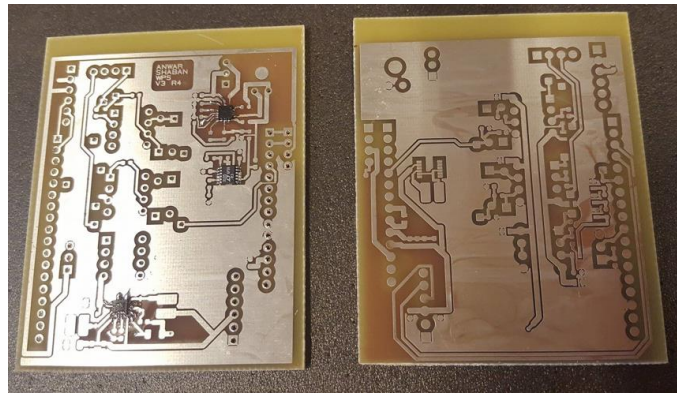


Figure 3: The produced board (both sides)

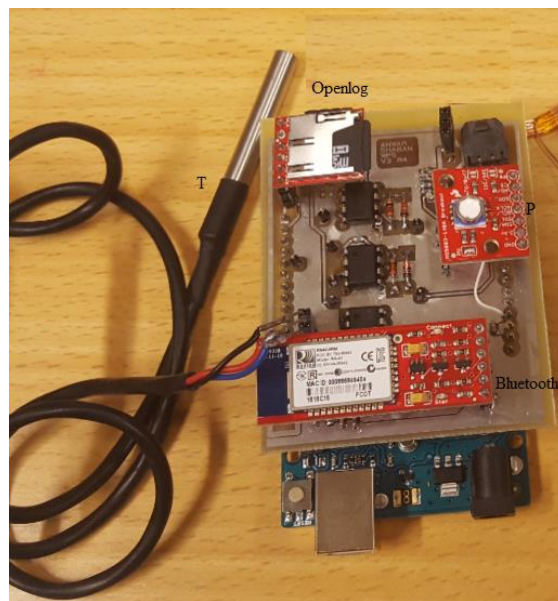


Figure 4: Water Parameter sensor

Fig.4 shows WPS connected to the PCB board. We have now everything in one board, which make it easier to minimize the size of the device

7.4 Power

7.4.1 How to power the Project?

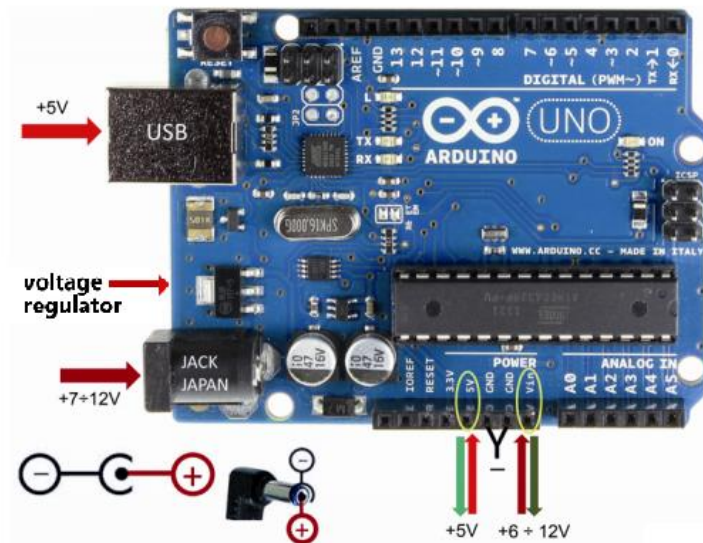


Figure 5: Arduino Uno powering inputs.

As shown in Fig.5, Arduino Uno has 4 locations for applying power: the USB port, the 2.5mm DC socket, the Vin pin on the shield header (which connects back to the DC socket) and the 5V pin. The USB port is designed to take in a regulated 5-volt DC voltage. The 5V pin can take in just 5V.

The Vin pin and the 2.5mm DC socket is more flexible. It will take a DC voltage over a 7-12-volt range. This is because Arduino board has a linear 5VDC three-terminal regulator. The regulator converts that voltage range back into a regulated 5VDC for the ATMEGA controller chip. The regulator can only maintain that voltage at no more than around 800mA of current. If the project needs more than 800mA, the regulator will be overheated and eventually fail.

It is important to know the amount of the current consumption. This depends on the Arduino board self, the sensors and the circuits that are connected to the board. If the chosen power supply cannot give the system the amount of current it needs, the circuit may start acting in a strange, unpredictable way. This is also known as a brown-out.

Two ways were used to find how much current the WPS might draw; experimentally and theoretically. Experimentally, by using a digital multimeter to measure the current going to the circuit while it is running. The result was about 200mA is drawn by the system.

Theoretically, by founding the current budget of the system as shown in table 8. The table shows the max current each component or circuit may use.

Table 8: The system currant budget.

Component	Current Draw
Arduino Uno board	50mA

DS18B20 Temperature sensor	3mA
MS5803-14BA Pressure sensor	1 μ A
3.7Vdc-5Vdc regulator	24 μ A
SparkFun Openlog	6mA
Bluetooth module	50mA
3x OP-AMP TL072cp	3x2.5mA
5VDC-12DC LTC3582-12	325 μ A
Current drawn by the water $V_{out2}/R_{I \rightarrow V}$	1mA

From table.1 the current that drawn by the system is about 100mA. This is half the measured current. The reason of high difference may related to the other small components on the board, and the used LEDs. Another reason can be the connecting of the water parameter sensor board to the Arduino Board. This may cause extra current drawing by the Arduino.

Now we know how much voltage and current our system needs. We have to find a suitable power supply. For a mobile project, which can be sent to the water, we need a mobile power. Arduino is a popular choice for battery-powered project. All we need is a battery, but what kind of battery can be used? Fig.6 views the most commonly battery types.



Figure 6: the most commonly battery types

The first common solution is to use a 9V battery-clip to DC connector adapter, as shown in Fig.3. The 9-volt PP3 batteries have very low capacity and aren't designed to supply more than around 40mA, which may be not enough.

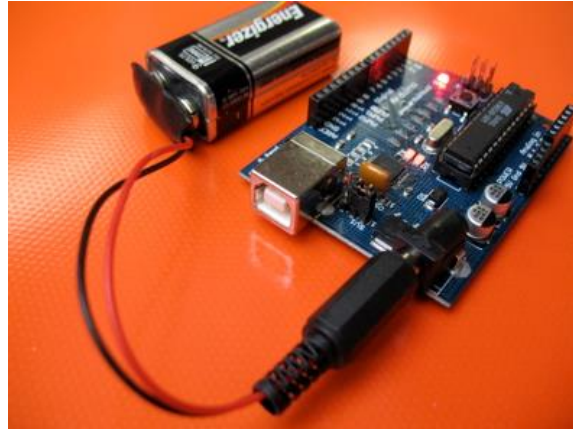


Figure 7: powering Arduino by 9V-battery

The second solution is to use a pack that holds four 1.5V AA or AAA batteries as in Fig. 7. However, using 4 batteries is somewhat too much and not desired in our project.



Figure 8: Powering Arduino by AA batteries.

The solution that is used here is a **3.7V Lithium Ion rechargeable battery** of capacity 1160 mAh. To charge this battery we have **BQ24074 charging** circuit built on the PCB board. Figure.9 shows the schematic of the charging circuit.

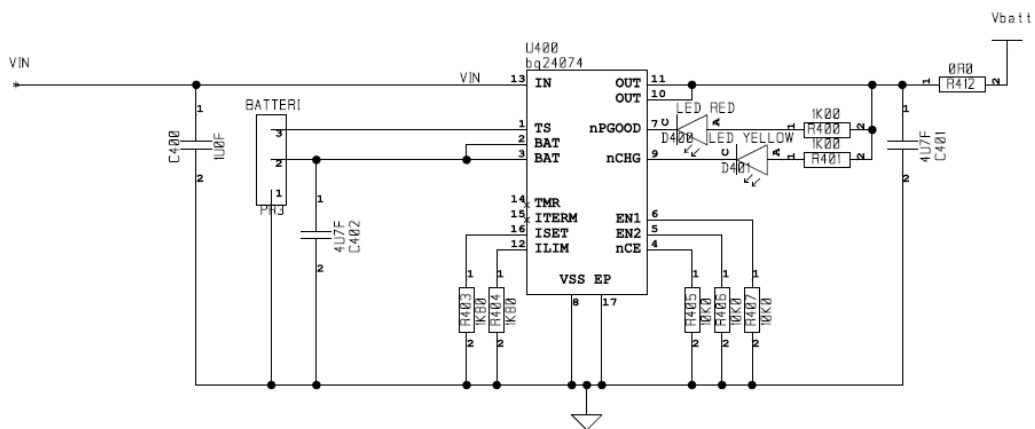


Figure 9: The Schematic of the charging circuit

Of course, no one of the Arduino powering input (Fig. 1) can take in 3.7V. This means, that we have to use a DC-DC converter to step up the 3.7V to 5V and fed it in pin 5V. In order to do this we use **LTC3105 400mA step-up DC/DC Converter** with maximum power point control and 250mV start-up. The schematic of the converter is shown in Fig.10.

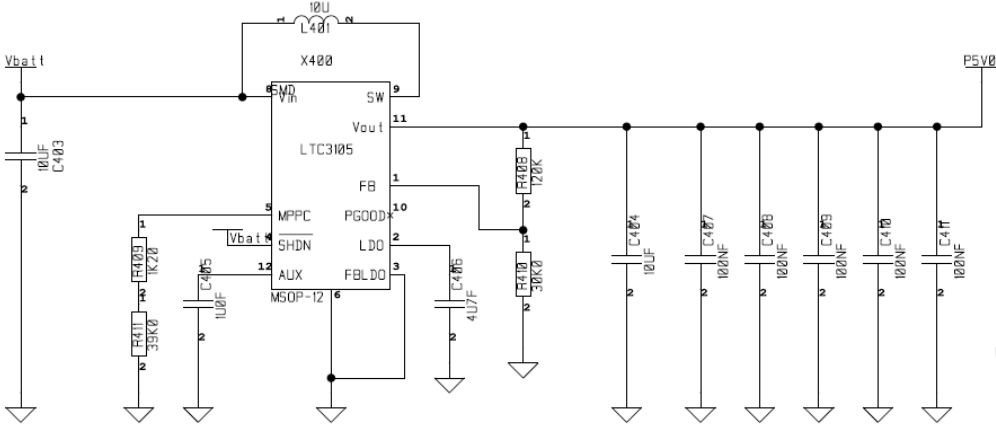
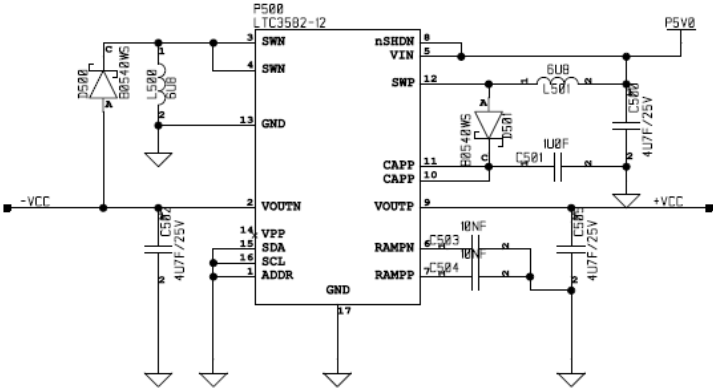


Figure 10 The Schematic of the LTC3105 DC-DC converter circuit

The Battery powers the Arduino Uno, which in turn send the required power to the sensors on the shield board. The Arduino can provide either 5V or 3.3V, but the LTC072cp amplifiers, which we use in the conductivity sensor, need at least $\pm 9V$ to power the $\mp VCC$. This is fixed by using **LTC3582-12 DC-DC converter**. This converter converts the 5V from the Arduino pin to $\mp 12V$. The $\mp 12V$ is fed on the $\mp VCC$ pins of the LTC072cp amplifiers.

Switcher circuit 5V -> +/-12V



Figur 11: 5V to $\mp 12V$ DC-DC converter

Every part of the system gets the required power and everyone is happy. What will happen when the system uses all the battery and it is empty? It needs charging but our system is always working under water. This means waterproof charging way or wireless charging must be used to protect the electronics from the water!

7.5 Wireless Induction Charging:

In 1831, Michel Faraday and Josef Henry discovered one of the most fundamental effects of electromagnetism: a varying magnetic field is able to induce electric current in a wire. The effect is the same if the field is produced by a permanent magnet or by solenoid. As long as the magnet field changes, electric current is generated. This effect helps us very much in wireless charging of the system battery.

Faraday's law in induction states: the inductive voltage, or electromotive force (e.m.f), is equal to the rate at which the magnetic flux (Φ) through the circuit changes. If varying magnetic flux is applied to a solenoidal, e.m.f. appears in every turn and all these turn must be added. If each turn of the solenoid has the same cross-sectional area (A), the flux through each turn will be the same, and then the induced voltage is:

$$V = -N \frac{d\Phi}{dt} = -N \frac{d(BA)}{dt} \quad (7.1)$$

N is the number of turns, Φ is the magnetic field, B is the magnetic field and A is the cross sectional area of each turn. The minus sign is an indication of the direction of the induced e.m.f.

From equation (7.1), either changing amplitude of B or area of the circuit can produce the voltage in a pick-up circuit. Therefore, the induced voltage in the pick-up circuit depends on:

- Moving the source of the magnetic field.
- Varying the current in the coil or wire, which produces the magnetic field.
- Change the orientation of the magnetic source with respect to the pick-up circuit.
- Changing the geometry of a pick-up circuit or changing the number of turns in a coil.

According to Faraday's, if an electric current passes through a coil, which is situated in close proximity with another coil, e.m.f, will appears in the second coil. This is known as magnetic inductive coupling. The coupling can be increased by more turns in the coils and placing them close together on a common axis. This was tested in a simple experiment as shown in the next section.

7.5.1 Simple Wireless Power Experiment

In this experiment, we make a simple circuit to wirelessly transfer the electricity using the principle of magnetic inductive coupling. The material needed in this experiment:

- Magnetic wire (copper wire)
- LED light
- 2N2222 NPN-Type transistor.
- Battery (power supply).

- Soldering iron.
- Electrical tape.

First, the inducer and receiver coils are made in the same manner with 30 turns, but the inducer coil needs a center tap at turn 15. Then the receiver is connected to the LED light and the inducer is connected to the transistor and the power, as shown in Fig.8.

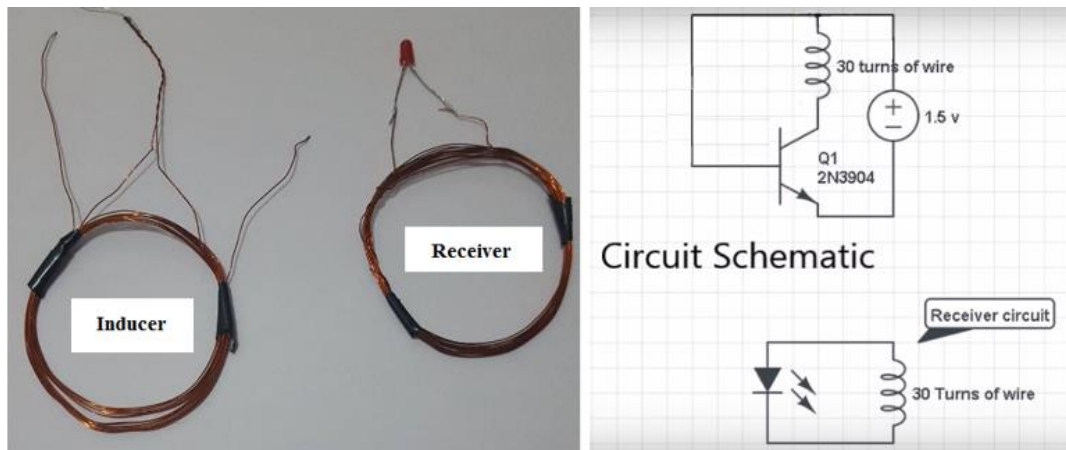


Figure 12: Inducer and receiver coils and circuit schematic.

The transistor is the brain of this operation. It is the cause of creating the changing magnetic field in the inducer coil by connecting and disconnecting the power at rapid pace. Fig.13 views the signal that created by the transistor.

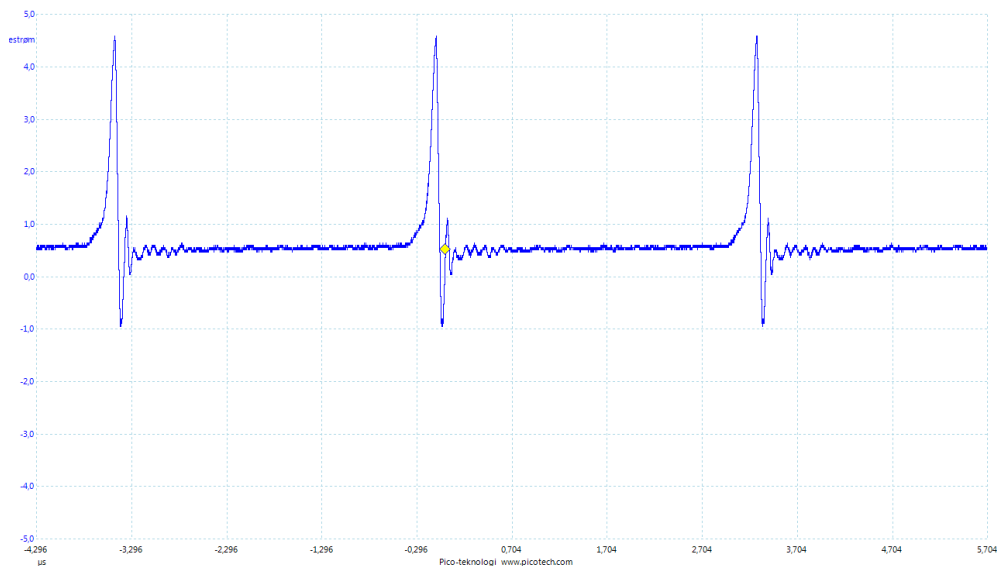


Figure 13 signal created by the transistor

Fig.14 shows the received signal by the receiver coil. And Fig.11 shows how the LED is powered.

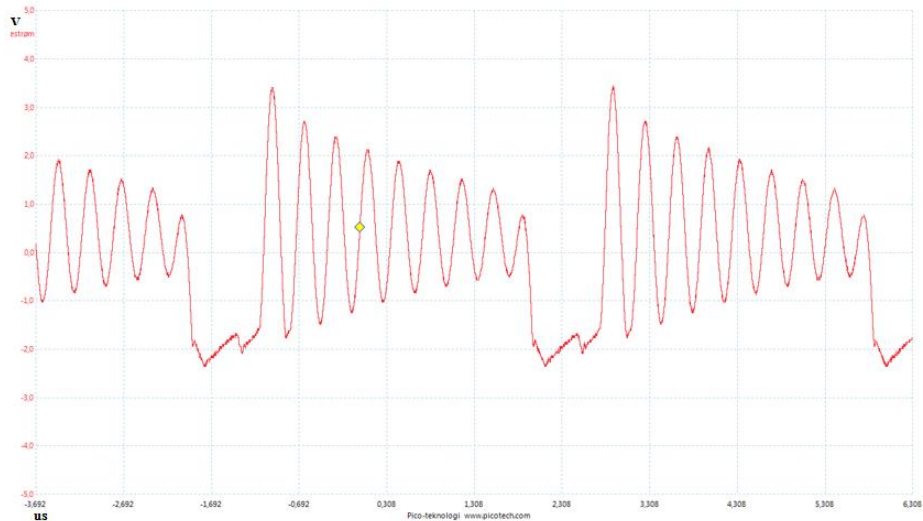


Figure 14 : The wireless created signal at the receiver coil

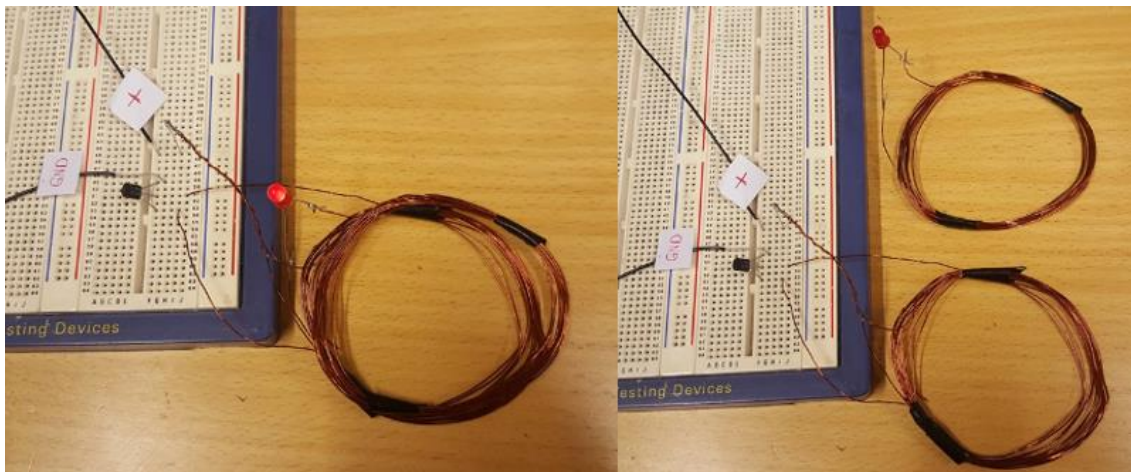


Figure 15 Wireless Electric transfer based on Faraday's law

The LED is powered when the receiver coil is moved around the magnetic field without any wires. As we know the created voltage in the receiver coil depends on the cross sectional area (A) of the coils. Thus, when we move the coil to the side the light become weaker and weaker. When $A = 0$ the light turns off.

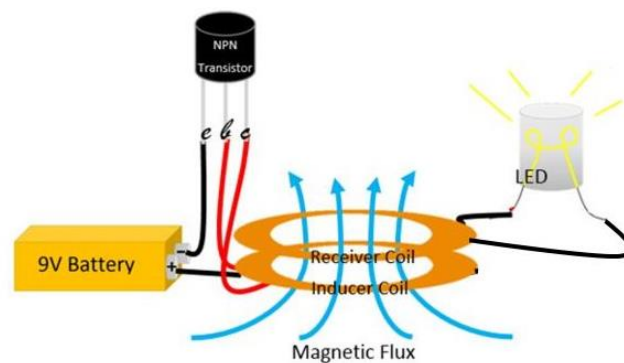


Figure 16 Wireless power

Summary of the work principle of the circuit:

In the inducer circuit (transmitter), the transistor works as an oscillator and creates an alternating current in the inducer coil. The alternating currents flowing creates magnetic field. The magnetic field extends to the receiver coil. An e.m.f is created and so a current flows in the receiver coil and so the LED turns on. This what happens in the Induction charger also.

This circuit is too simple to be used in our system and it can't transfer enough energy to our system. But with some modification it can be useful. We need at least 5V to charge the 3,7V lithium battery. The transmitter part need an oscillator to generate sinewave signal.

7.5.2 Wireless Charger

Using an electromagnetic field to transfer energy between two objects through electromagnetic induction is becoming more and more common in wireless charging of new electronic devices like smart phones, laptops and toothbrushes. The main parts of these wireless charges are two coils with alternating current in one of them.

To charge our battery wireless, we use inductive charging set from adafruit (Fig.17). It is a basic charge set consists of transmitter (Tx) and receiver (Rx) coils. An induction coil creates an alternating electromagnetic field from within the transmitter circuit powered with 9V-12V. The receiver coil takes power from the electromagnetic field and converts it back into electrical current to the receiver circuit that outputs 5V – max 500mA (based on distance). The distance between the Tx and Rx ranges between (1mm-20mm). as distance increases current capacity of receiver decreases.

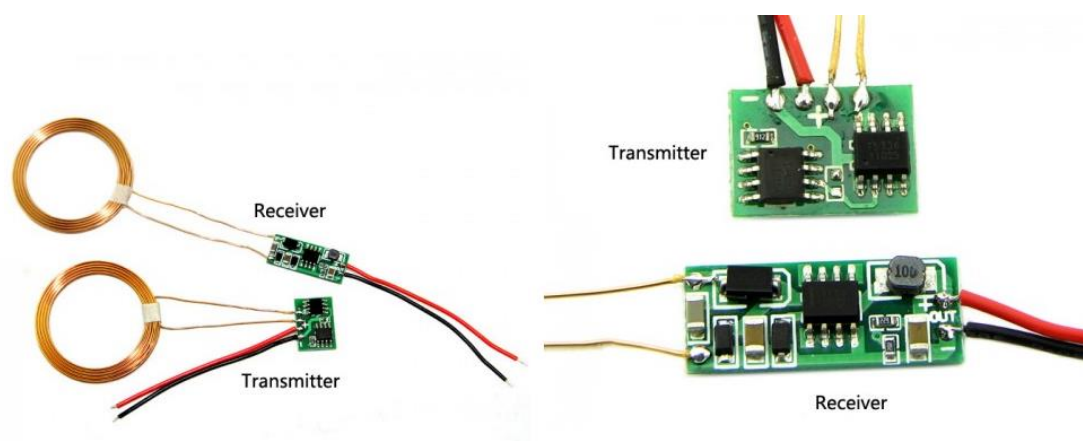


Figure 17: Inductive charging set.

Transmitter Board has two ICs,

- 1) XKT-408A / 1215A - It Generates sinewave signals
- 2) T5336 from Elcoteq - Seems like Mosfet Driver in SOIC8 package to drive coils up to 60V peaks on sinewave.

Receiver Board has one IC

- 1) T3168 from Elcoteq

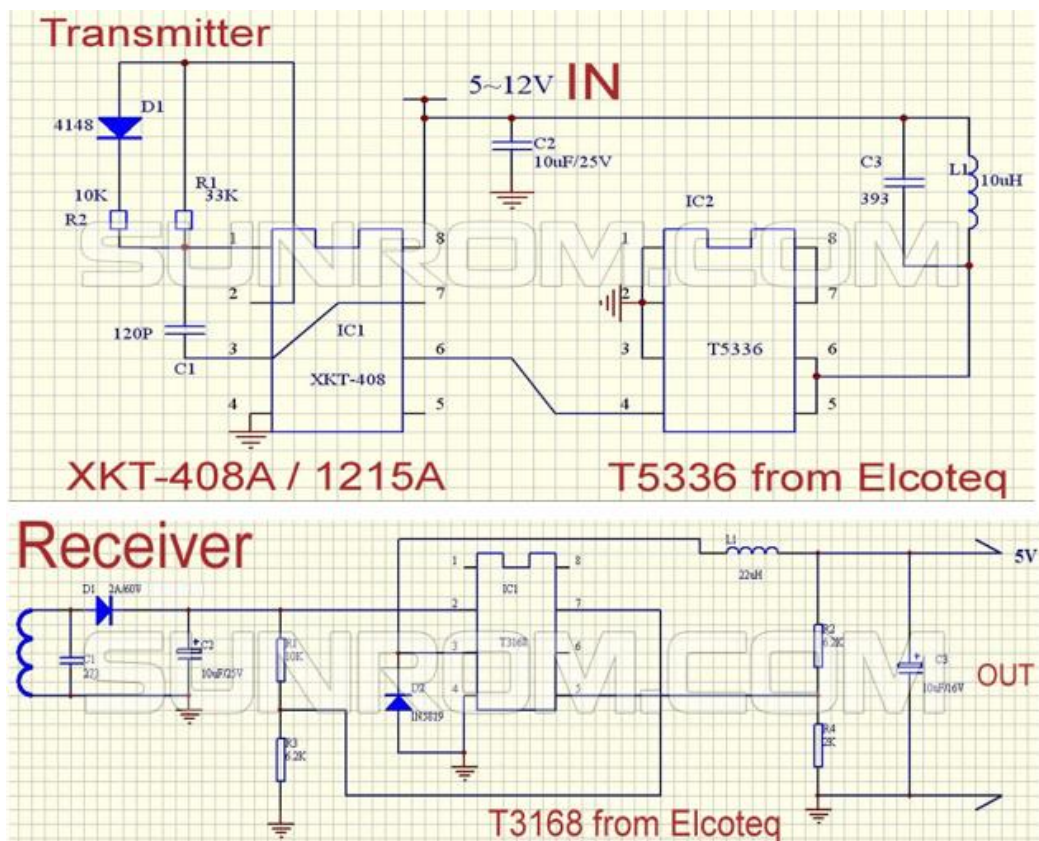


Figure 18: Schematics of the transmitter and receiver [72].

7.5.3 Testing the inductive charger:

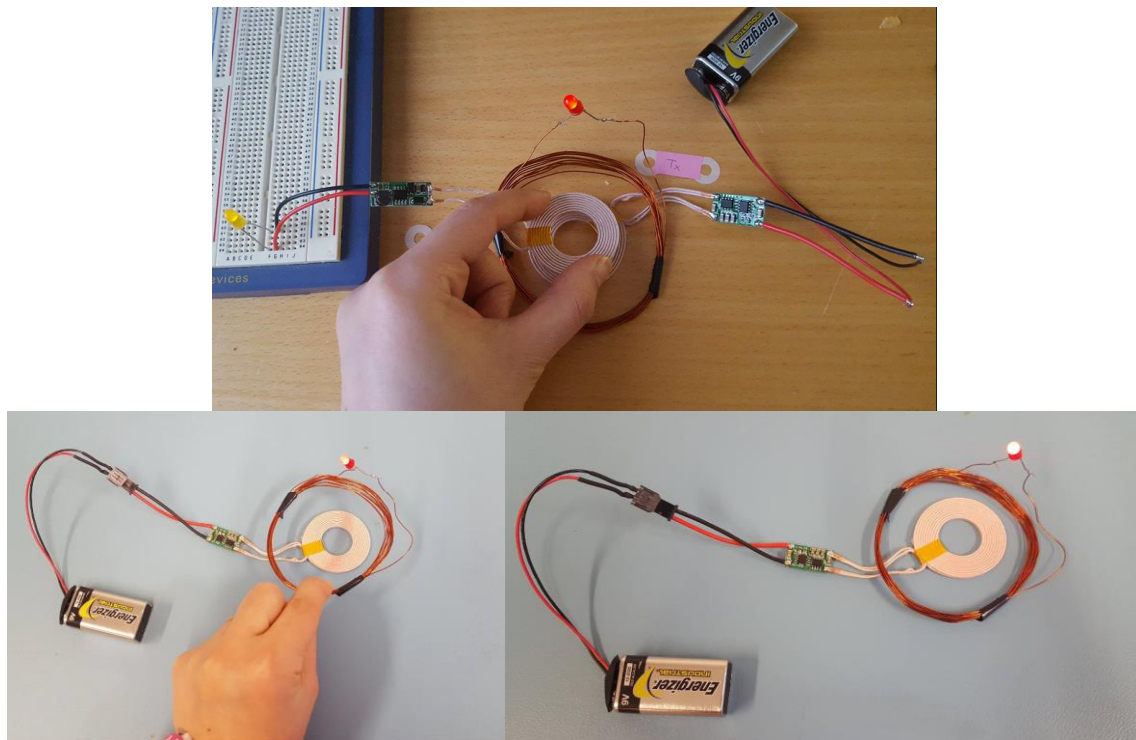


Figure 19 testing the wireless power transfer.

In Fig.19 if a rechargeable battery is connected to the receiver part instead of the yellow LED, It will be wireless charged. This we do to charge our system as shown in Fig.19.

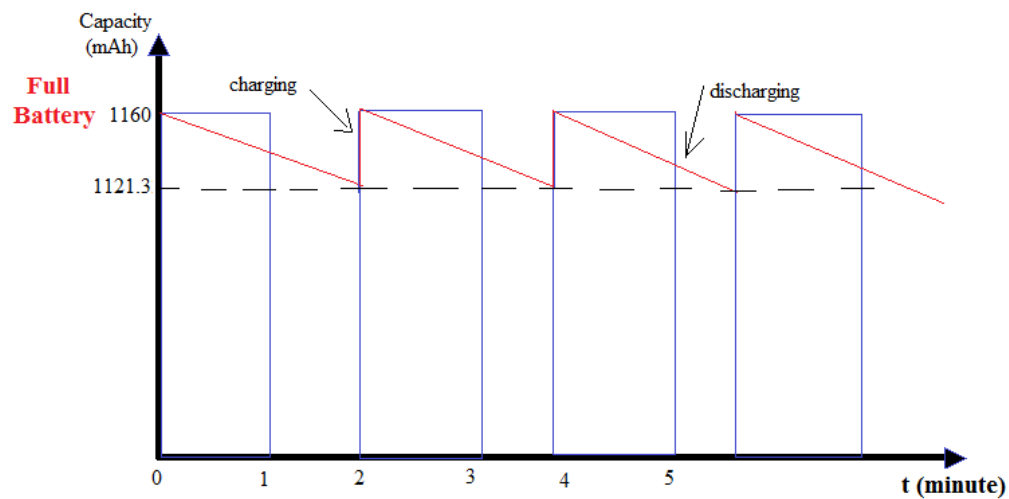


Figure 20: Battery charging and discharging

The system battery capacity is 1160mAh and our system draws about 200mA. This means the battery can hold on 5 hours. If the system is measuring in the water, for example, in two minutes, the battery will discharge from 1160 mAh to about:

$$1160 - \frac{1160 \times 2}{60} = 1160 - 38.67 = 1121.33$$

After getting the instrument out of the water, it will be placed on the induction charger until the next measurement, and so on, as shown in Fig.20.

7.6 Packaging to get waterproof instrument

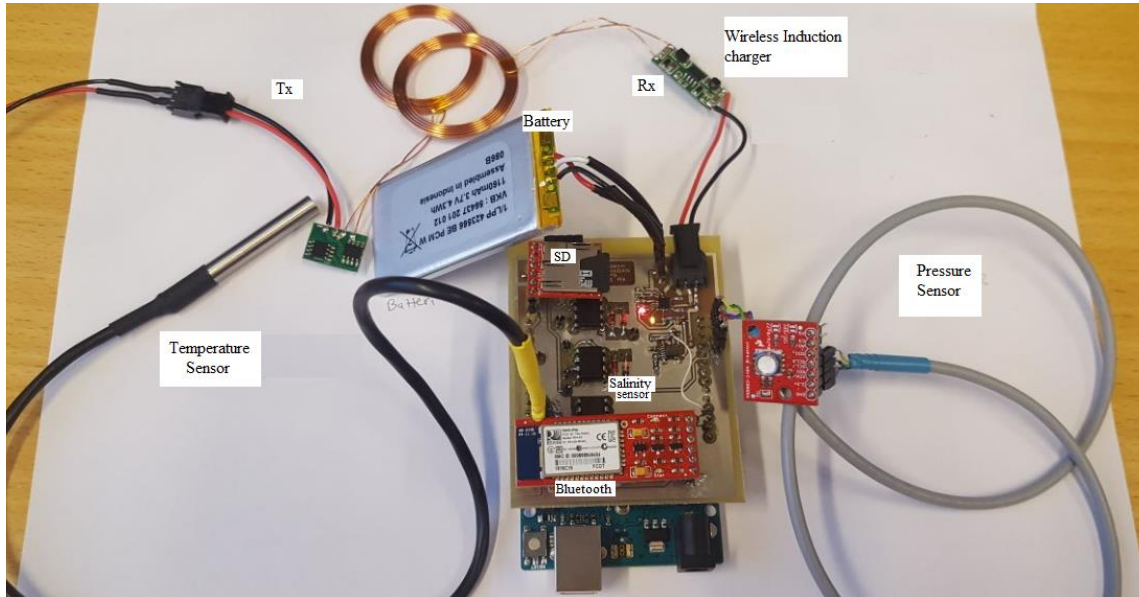


Figure21: The whole system of the projec

After getting every thing in its place, the system is complete. Now we have to think about packaging. Fig.21 shows our complete system containing the three sensors, Bluetooth, openlog, battery and wireless charger. All of those components attached to the water parameter sensor board. The board is connected to Arduino Uno. This system have to be waterproof to can be tested under water. The sensors have to be in contact with the water to measure water parameters and the rest of the electronic have to be isolated.

The idea is to make a waterproof package of plexiglass that is strong enough to withstand the water pressure at 30m depth. The best geometric form to resist the water pressure is the sphere. The water applies the same pressure on all of the sides of the sphere. But it is somewhat difficult to package the electronics in the sphere. It is easier to use cylinder, the next strong geometric form. The pressure is equal on the sides of the cylinder so it can not exposed to any deformation. At the bottom and the top the pressure can be different so we have to take care of, and ensure to be strong enough . Figure 22 translates our thoughts by primary drawing of the cylinder that can be used to protect our system.

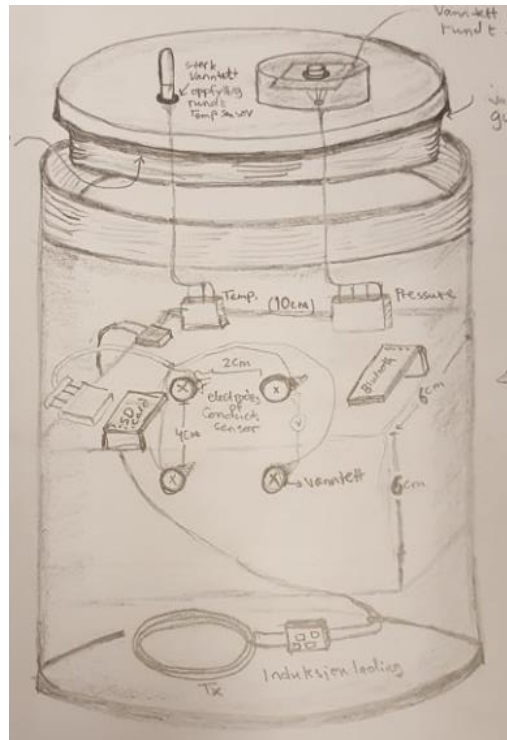


Figure 22: The primary drawing of the package.

A cylinder of Plexiglas, 5mm thickness and 90mm diameter. The three sensors get out from the lock through waterproof cable. All of the electronic that can be in contact with water is isolated by epoxy. The lock is seal by O-ring and supported by brackets at the sides. The brackets supports the bottom and top, holds the 4-electrodes of salinity sensor, and has holes to tie a rope used to lower the instrument in the water. Fig.23 shows the final design of the waterproof package. Thanks to the workshop, especially Jan Kristiansen, for helping in building the package of the system. The following pictures shows how we packaged the system to be ready for diving and testing in the water.

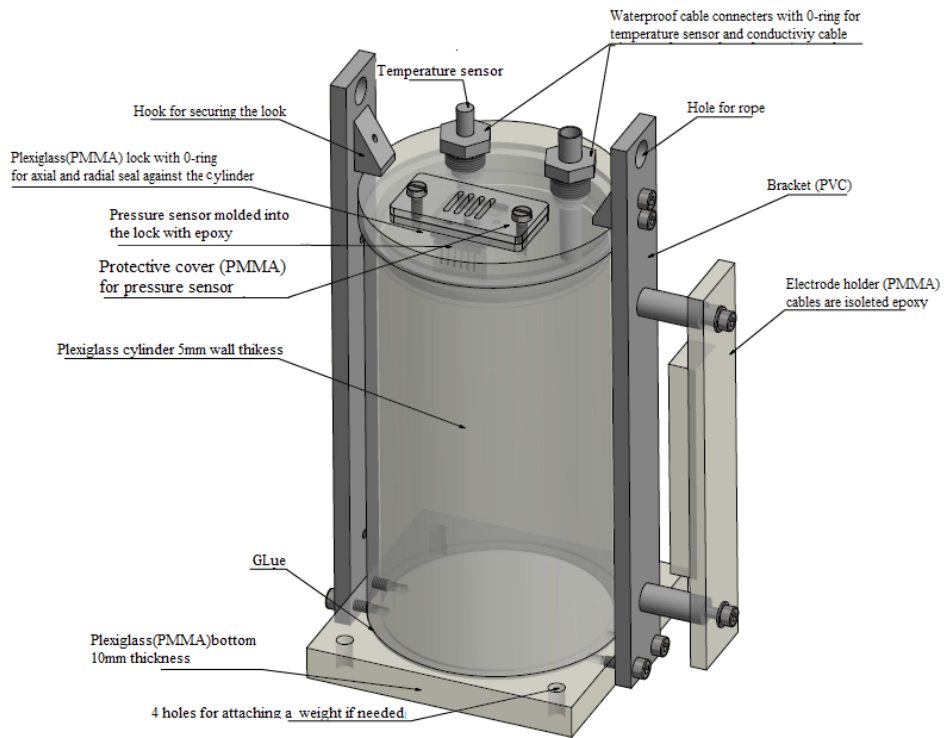
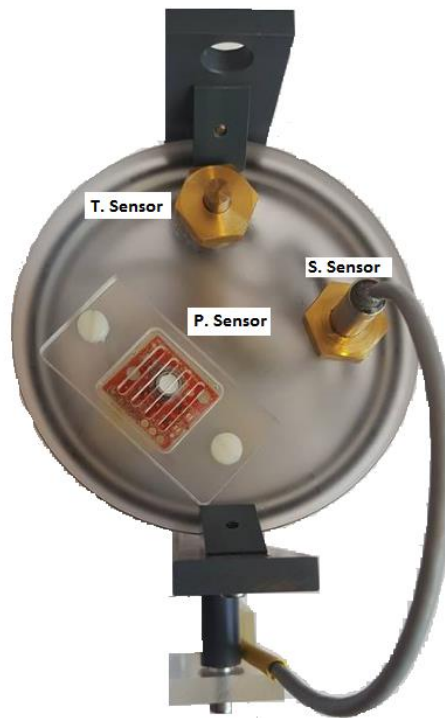


Figure 23: The final design of the WPS package.



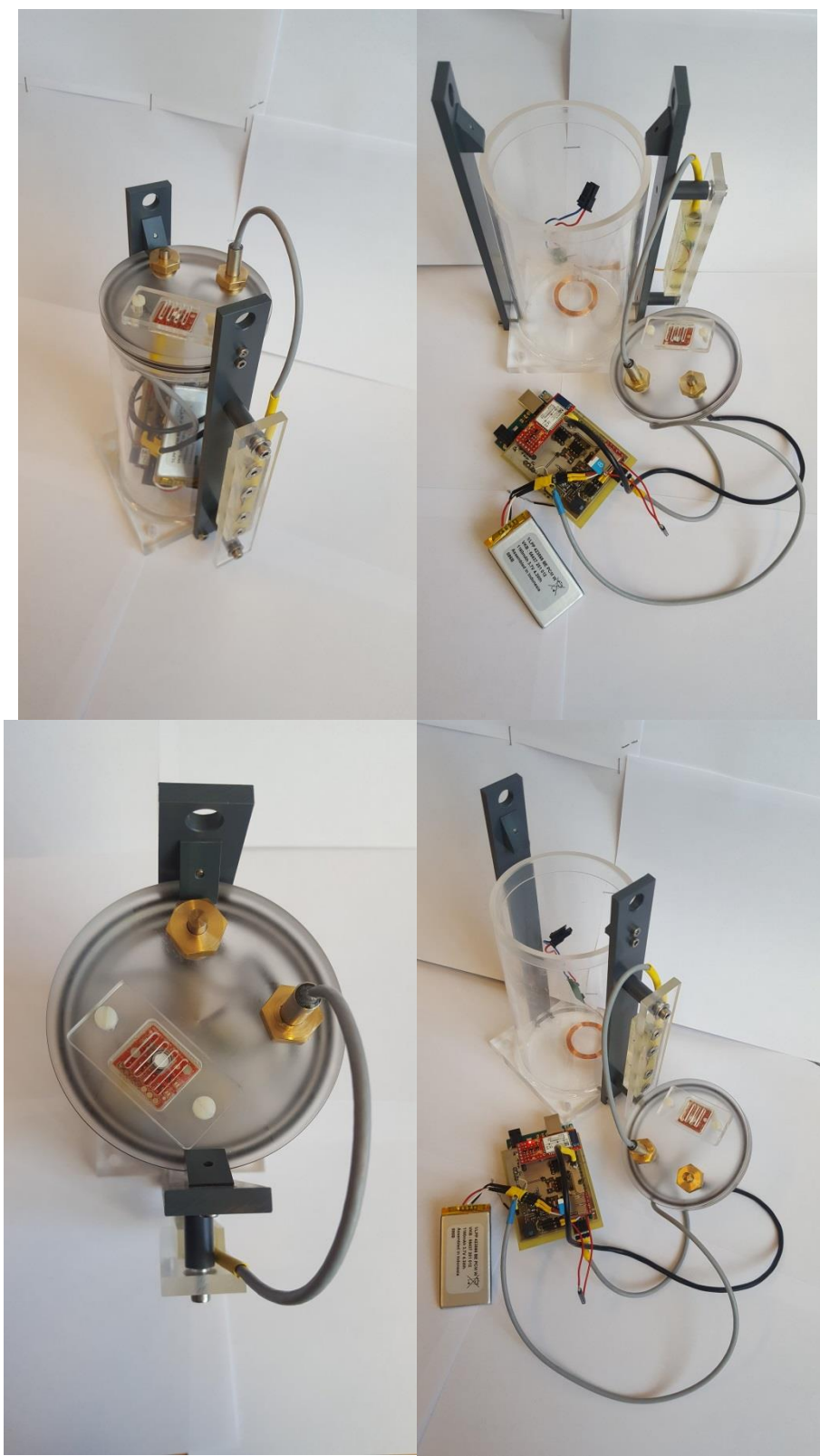


Figure 24: Pictures of the system and the package. Ready to dive in the water 😊

7.8 Implementation of the algorithm's

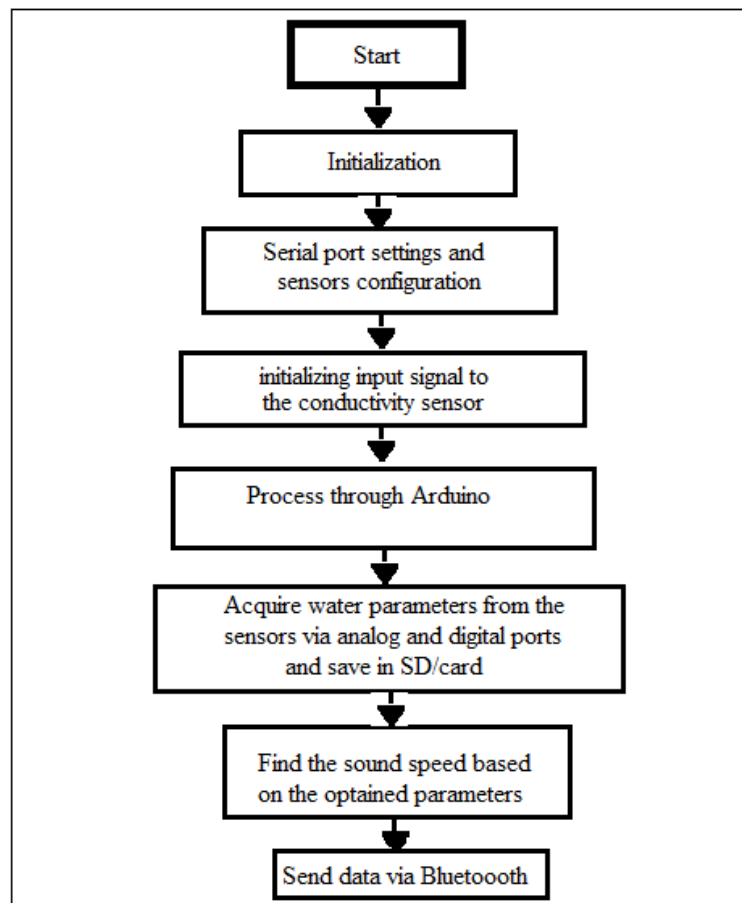


Figure 25: The flow of the code

The program code is based on some modules used together with some modifications to run the entire system. It initialize and configure the sensors and the input signal preparing the system to receive the data from the water. The received parameters data are used to find the water parameters and sound speed using Medwin's expression and store the results in the SD-Card.

Unfortunately, the code doesn't work well because of possible conflicts between the modules libraries. The generated input PMW signal of the conductivity sensor at Pin-5 became unstable, as shown in Fig.26, and the pressure sensor doesn't read correct values. As shown in Fig.26 some pulses don't come at regular time which causes unstable sine wave and then incorrect outputs.

The most possible reason to the instability is the using of the timers. It is possible that the system uses the same timers in two different places, which may cause disorder in the program that leads to faults in the results. In the conductivity sensor code, we use timer 0 and timer 1. There is a big chance that the pressure and temperature sensors libraries use one of these timers or both. The result of that, troubles in the PMW signal and errors in the results of both pressure and conductivity sensors.

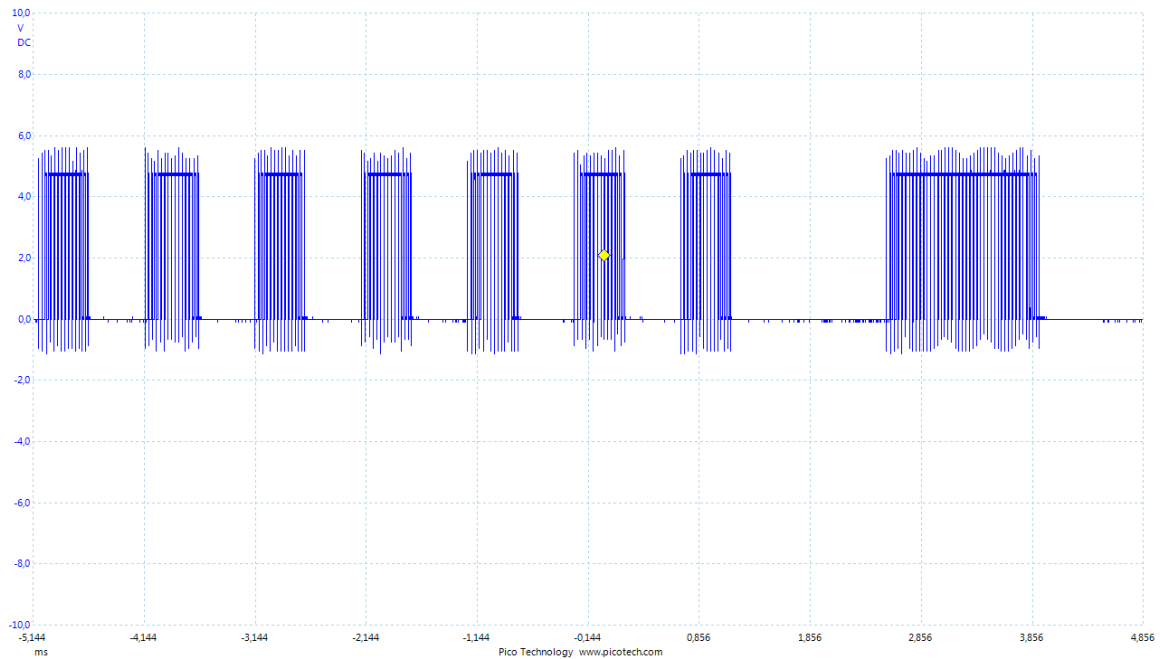
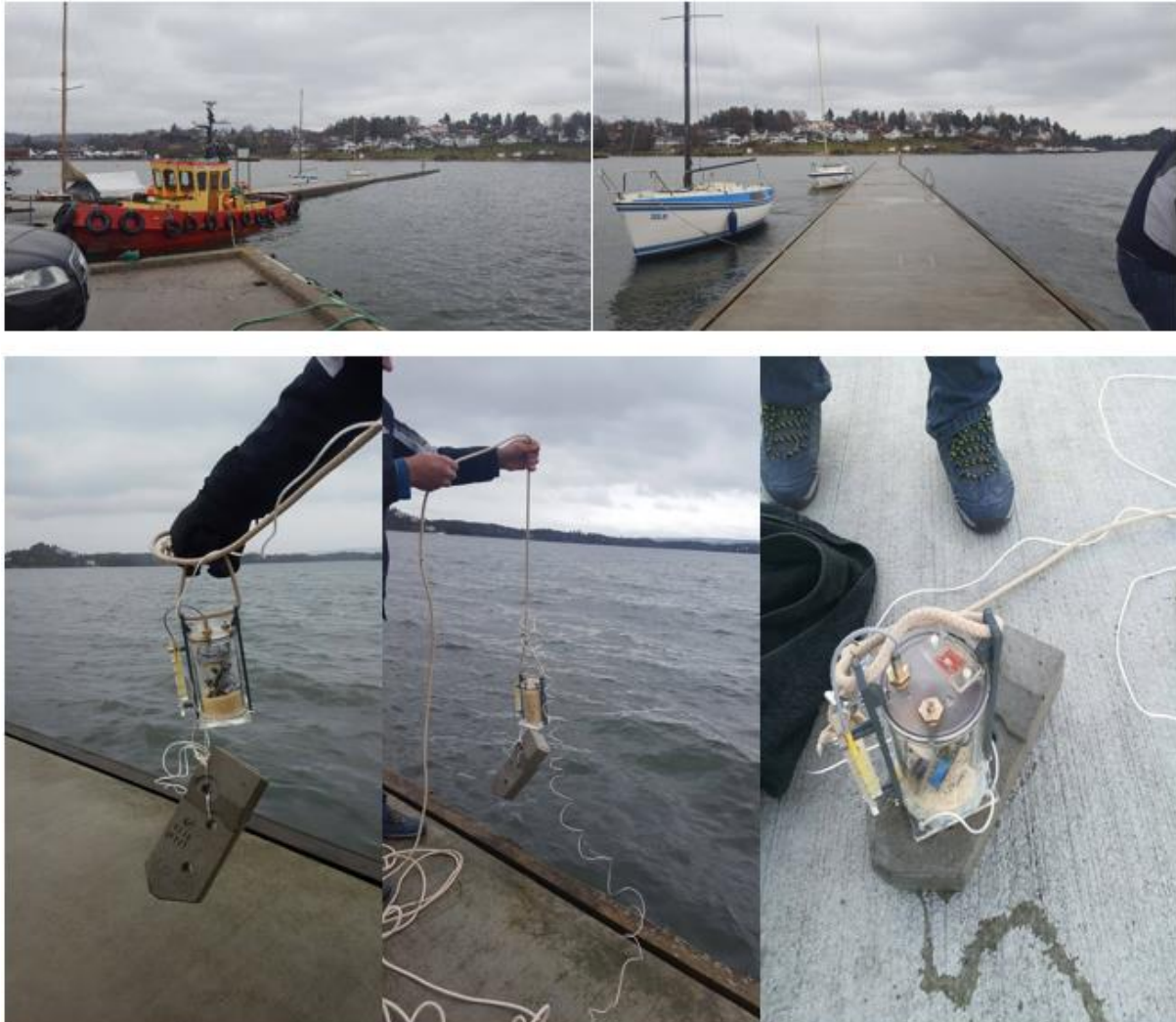


Figure 26: We see here the unstable in the PWM signal from the ATMEGA328 (pin 5) for producing the input sinus signal to the conductivity sensor. The un-stability is seen as jitter in the timing. This instability occur when the ATMEGA run the entire code for all sensors.

To solve the problem, we tried to use timer_2 instead of timer_1 but it didn't work. This means that the libraries are using timer_2 also. Another try was to not use timer_1 at all. Timer 1 is used to alternate between pin_5 and 6 for each half cycle and make a variable duty cycle for each pulse. Since we just use the signal on pin_5, we assumed that we could drop this part of alternating the pins. Unfortunately it was not possible to produce a sine wave from the resulting pulse because of the jitter in the output from the pulses was not regular as required.

The good solution of this problem is to use an external oscillator to produce the input sine wave signal. An external oscillator instead of producing the input signal from the Arduino self may solve the problem and make more table system. This need some changes in the PCB board that we don't have enough time to, so we have to accept the system as it is now and separate the code into two parts. The first part will estimate the pressure, depth and the temperature. The second part will estimate the conductivity. This what we have done in the testing.

8. Testing the WPS



The WPS was tested in the lab and in situ. The main lab tests were done in various water tanks where we could control depth, salinity and temperature. The in situ test was conducted in Holmen fjord 16 April 2017 at Holmen marina. The plan for the *in situ* testing was to lower the WPS unit together with an industrial and well calibrated profiler STD/CTD SD204. Unfortunately this device had to be delivered back to the owner few days before we managed to do the test. As an alternative we had to use a ruler rope to control the depths.

Test of the depth sensor

The depth sensor was tested in a tank at the lab from 0 to 0.6 meter and in situ at a depth of 15 meter. According to the sensors specification its accuracy is less than ± 20 mBar for pressures down to 6 Bar. Measured pressure were converted into water depth according to equation

$$(5.8) \text{ Depth} = \frac{P - P_0}{\rho g}$$

where $P_0 = 101325$ Pa is the atmospheric pressure at the surface, ρ the water density and g the gravity. The following data was measured:

Table 9 Measured and estimated depths from the MS5803 pressure sensor.

Depth(m) (Ruler)	P (measured)(Pa)	Depth(m) Estimated	Depth(m) difference.
0.00	101100	-0.02	0.02
0.035	101470	0.01	0.02
0.06	101600	0.03	0.03
15.00	252990	15.03	-0.03

The measurements show that the pressure sensor predicts the depth with an accuracy of +/- 0.03m relative to the depth measured by the rulers.

Testing the temperature profiler

Relying on the depths estimated by the depth sensor we measured the temperature profiles presented in the figures below. The sensors was lowered with an average speed of 12 cm per second and lifted with an average speed of 18 cm per seconds. The speed was controlled manually when the rope was lowered and lifted. We tempted to keep a stable velocity, but Fig1 show that we did not succeeded as well as intended. This influence on the temperature sensors ability to adapt to the correct temperature at each depth, and the profiles below must be regarded according to this.

We see from Fig.2 and Fig.3 that there is a hysteresis between the lowering and rising of the sensors. At 7 meter we see a difference of 2°C. for the DS18B20 and 1.5°C for the MS5803. The hysteresis indicates that the sensor needs more time than allowed in the test to adapt to the correct water temperature. The reason why we did not use longer time in testing in the water that we found at some water was leaking to inside the package of the sensor and it was not waterproof. So we had to use some rice around the electronics and draw the rope quickly to avoid the contact with the water.

The temperature sensors ability to adapt to changes in temperature was tested in the lab moving the sensor between two water containers with different temperature. The data is shown in the following Fig.4. Here we see that the rate of change in the sensor output is high when there is a high difference in the temperature. When the difference between measured and correct temperature is small we see that the rate of change also become small. This means that longer and longer time is needed to approach more and more correct readings.

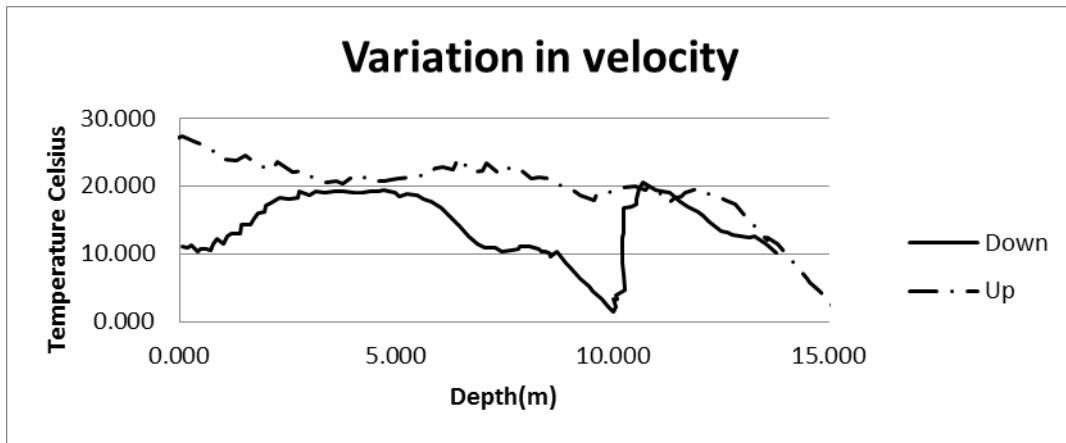


Figure 1: the variation of velocity of lowering and lifting the temperature sensor.

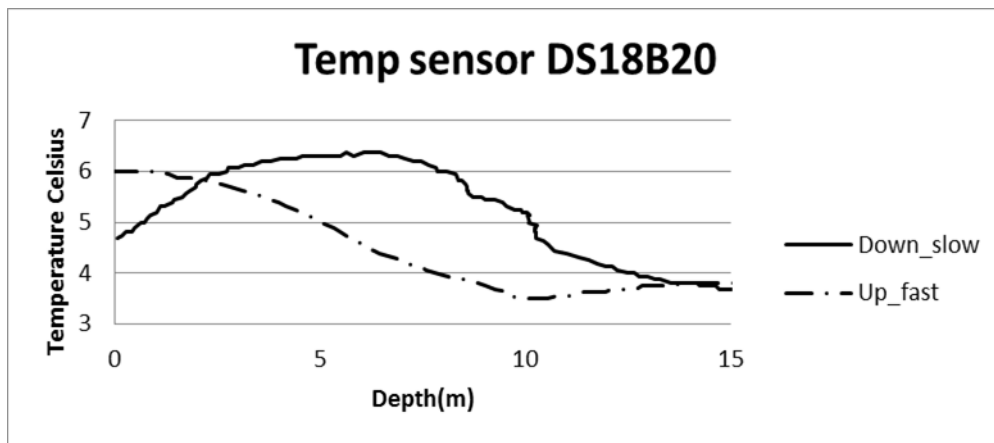


Figure 2: Temperature sensor DS18B20 lowered to 15 m and up. Down speed was 12cm/sec while up speed was 18cm/sec

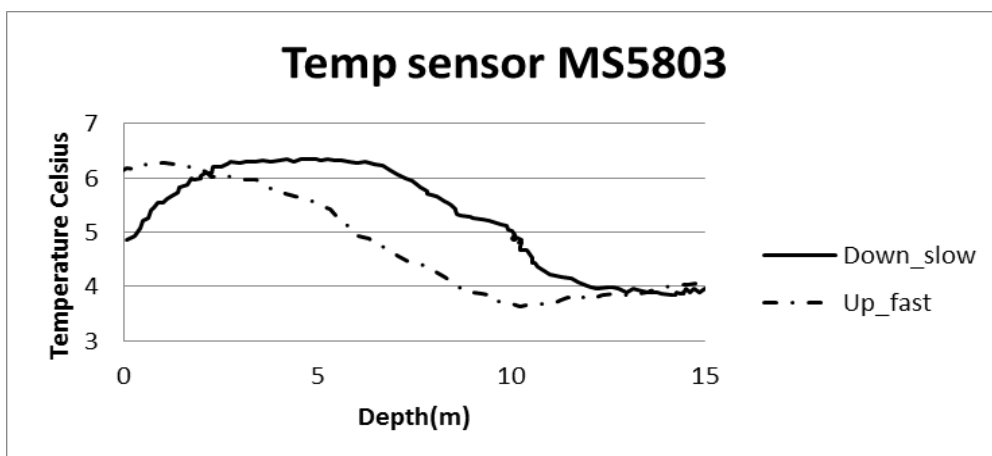


Figure 3: Temperature sensor MS5803 lowered to 15 m and up. Down speed was 12cm/sec while up speed was 18cm/sec as for the previous figure

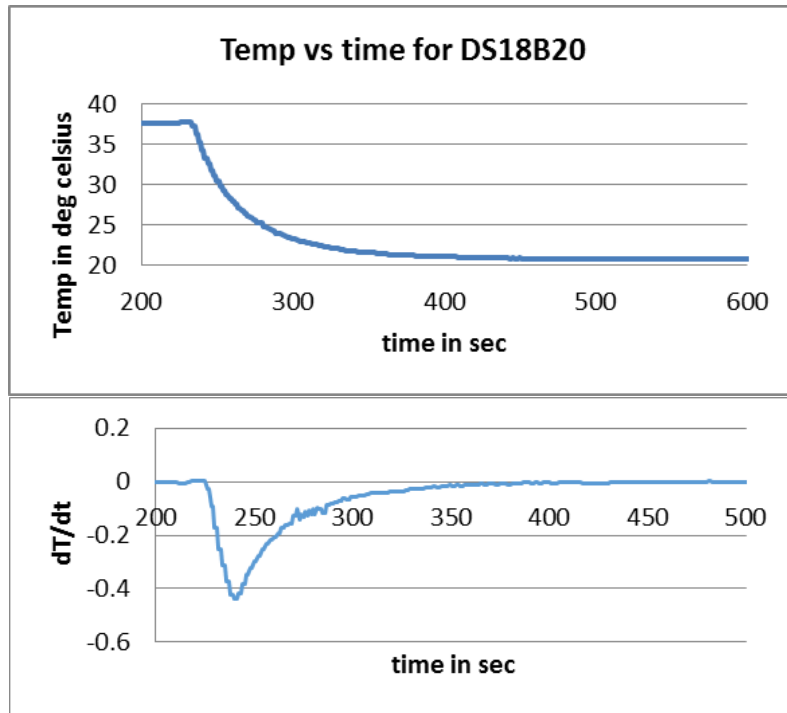


Figure 4 Upper temp change as function of time. Lower: differential change of temp versus time

We have tested the instrument in Holmen fjord on 16 April 2017. We did two separate tests, one for the pressure and temperature, and one for the salinity. Fig.24 shows the temperature profile measured with the temperature sensor DS18B20 and Fig.25 shows the temperature measured by the pressure and temperature sensor MS5803. We see how the temperature changes with the depth. It increases slightly in the first 6m and then started to decrease until the bottom at depth 15m. We used a ruler rope to lower the instrument in the water. The rope had marks for each meter and we lowered about 15m of the rope in to the water. We lowered the sensor. The sensor recorded the pressure. The pressure was recorded on the memory card. Using Equation (5.8) the measured pressure was converted to depth.

The salinity sensor of the WPS was first calibrated in a solution of 30 g salt per liter of water. The calibration constant was $K = 0.15$. After calibration, the sensor was tested in Holmen fjord. The results are shown in Fig.27. Unfortunately we couldn't get another salinity meter to compare our result with. Therefore we can't verify the data we obtained from our sensor. But we know that the salinity of Oslo fjord is between 20 to 30 g/L [73] and Holmen fjord is part of Holmen fjord so its salinity can be also in this range. Since our data are not out of this range, so it can be acceptable. More tests needed to be done at the future and the results must be compared with the results of another professional instrument to be sure that our sensor works well.

At the lab we have tested the salinity sensor by measuring a solutions of known salinity and we get a reasonable results that vary about ± 2 g/L from the original salinity.

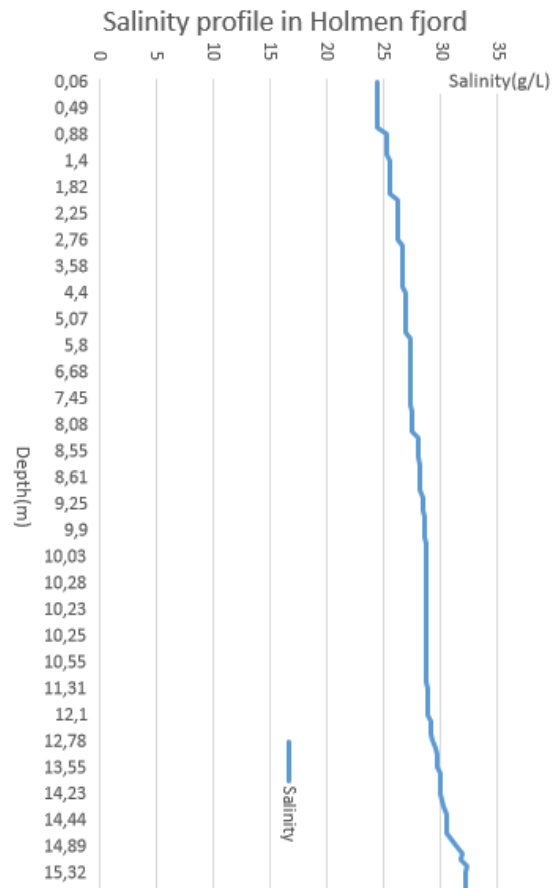


Figure 5: Salinity profile of Holmen fjord down to 15m depth

The following figure shows the relation between salinity and temperature.

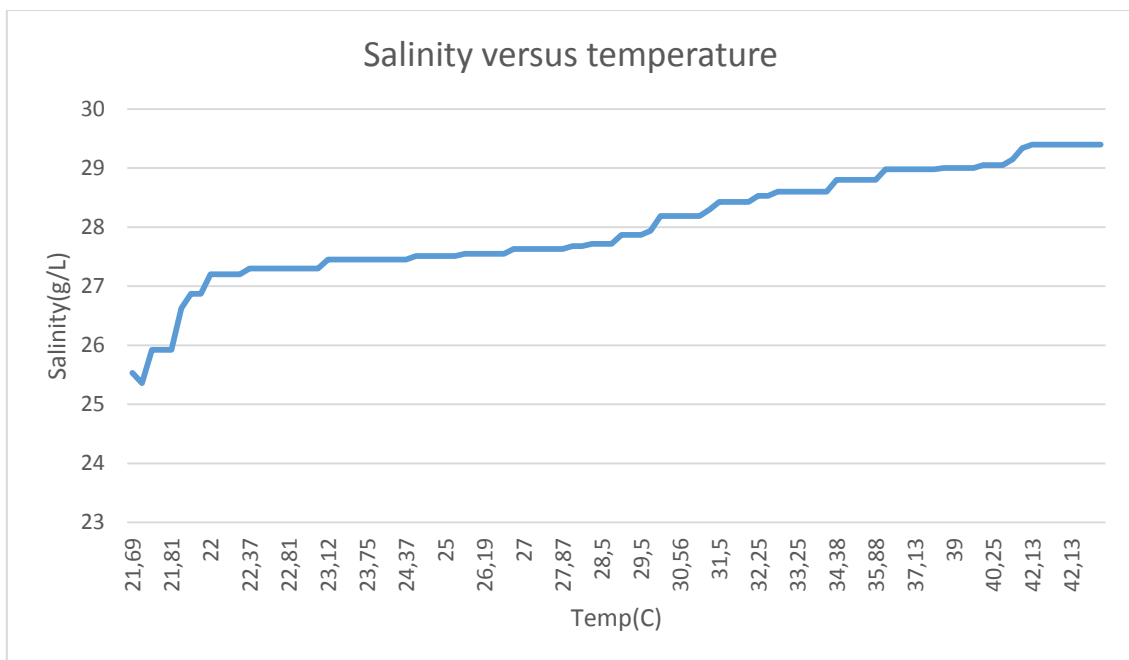


Figure 6: The relation between temperature on Salinity.

Using the data of the two tests, we can estimate the profile of the sound speed in the Holmen fjord. Medwin's expression is used here to find the sound speed using the temperature(T), depth(D) and salinity(S) of the water. As shown in Fig.7.

$$c = 1449.2 + 4.6T + 5.5 \times 10^{-2}T^2 - 2.9 \times 10^{-4}T^3 + (1.34 - 10^{-2}T)(S-35) + 1.6 \times 10^{-2}D$$

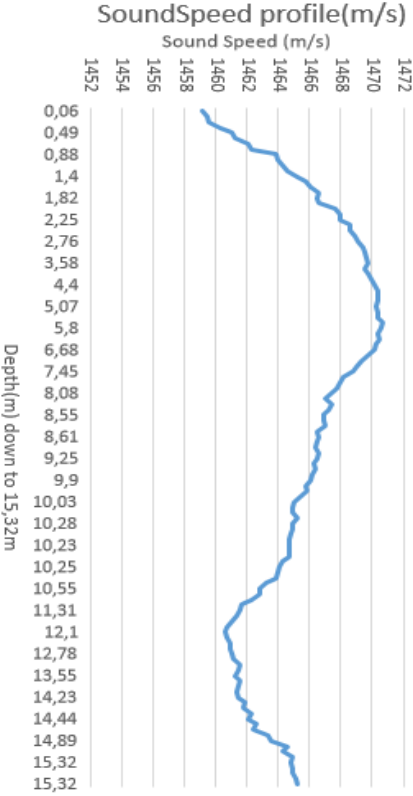


Figure 7: Sound speed profile

Notice that we are testing on the surface layer or what is called mixing layer. This layer influenced strongly by the weather and the day and night changings. For example, the rain will decrease the salinity at the surface because of increasing the fresh water. When we tested the device it was rainy and cold and this may explain the profiles that we get in the figures.

NB: Unfortunately we couldn't test the induction charging in the situ, because something wrong has happened to the Lithium battery and it was died can't be charged or give any power. It was Easter vacation and difficult to order another battery. We had to use 9v alkaline battery and connect it to 2.5mm DC socket.

9. Summery and conclusions

The goal of this project was to build a system that can measure the water parameter sensors, which are necessary to estimate the sound speed profile along the vessels survey. The sound speed is an important parameter in the hydroacostic methods. An accurate information about the sound speed in the water as a function of temperature, pressure and salinity is important. This because horizontal beaming is strongly influenced by refraction caused by the sound speed profile. Resulting refraction cause misinterpretation of the estimates from the echo sounder.

We have built a water proof wireless device that contains temperature, pressure and salinity sensors. The sensors are controlled by Arduino Uno board based on Atmega328 MCU.

The system has a reasonable price. The total cost of the parts is about \$150 distributed as shown in the following table. The most expensive part is the pressure sensor board. We can use a single sensor cost about \$25, and build the board direct on our PCB. This will lower the price about \$35.

Tabel 9. The cost of the system parts.

Parts	Price \$
Arduino Uno	24,95
Temp. DS18b20	10
P. MS5803_14ba	59,95
Wireless Charge	10
Openlog(SD_card reader)	14,95
Blusmirf silver (Bluetooth)	24,95
Lithium Battery	10

In the field test at Holmen fjord, we have tested the system in two rounds. The first one to measure the pressure and the temperature. The second to measure the salinity. We could lower the instrument down to 15,3m depth. We had to also by another professional instrument to compare the results, but we had no available instrument. It was Easter vacation and difficult to borrow any instrument.

Conclusion

We have built a wireless and low price WPS that can estimate the sound speed profile in the water by measuring the temperature(T), depth(D) and salinity(S) using Medwin's formula:

$$c = 1449.2 + 4.6T + 5.5 \times 10^{-2}T^2 - 2.9 \times 10^{-4}T^3 + (1.34 - 10^{-2}T)(S-35) + 1.6 \times 10^{-2}D$$

4-electrode salinity meter was also built in this project and used together with the other sensors to find the sound speed profile.

Every sensor worked well but we faced a problem when the MCU Atmega328 had to run the entire code for whole system. The salinity sensor needs an alternating input signal. We decided to generate this signal from the Atmega328 by generating PWM signals at pin 5, but running the entire code caused instability in the input signal that is seen as a jitter in the timing. This problem caused incorrect results from the sensors so we had to separate the code into two parts to avoid the timing errors that we have no enough time to solve this problem by another ways. The solution is to use an oscillator to generate the wanted input signal by this way the salinity sensor timing will not affect the other sensors and we can run the entire code of the system but this required some modifications in the PCB and more time.

More tests have to be done and the results have to be compared with a professional and trustable profiler to find how well does the WPS work.

Improvements and future work

The accuracy of the system can be improved. An external oscillator can be used to generate the input signal of the salinity sensor instead of direct generating it by the microcontroller. This will help avoiding timing jitter and so we can run the entire code of the whole system at the same time. This will cause more accuracy in the resulted sound speed profile and the system will be easier to use and we don't have to do the tests in two rounds. By this way it will be possible to get all the wanted parameters at the same test and get better results.

More tests needed to be done and a professional instrument must be used at the same time to compare the results and to know how to improve our instrument and its accuracy.

The system can be built to be smaller by building all the parts including the microcontroller on the same board that have to be as small as possible. An automatic lift can be designed to lower the WPS in the water and lift it at a constant speed and avoid the hysteresis and or any error.

More sensors can be added to the system. For example, to measure the oxygen dissolved in the water Atlas Scientific Dissolved Oxygen Kit can be used [74]. If it is need to measure the PH in the water PH sensor kit [75] can be added to the system. The good in those two kits that you have everything you need to measure the Oxygen and the PH but they are kind of expensive.

10. Appendix

10.1 Arduino Uno source code

10.1.1 The complete code HB: For what?

```
#include <math.h>

//Bluetooth
#include <SoftwareSerial.h>
int bluetoothTx = 2; // TX-O pin of bluetooth mate, Arduino D2
int bluetoothRx = 3; // RX-I pin of bluetooth mate, Arduino D3
SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);

/*libraries for Temperature sensor*/
#include <OneWire.h>
#include <DallasTemperature.h>
//Temp sensor
OneWire oneWire(8) ; // pin (8)
DallasTemperature sensor1(&oneWire);
float Temp, V1, V2, conductivity, EC25, TDS25, TDS ;

/*libraries for pressure sensor*/
#include <Wire.h>
#include <SparkFun_MS5803_I2C.h>
void header();
// Begin class with selected address
// ADDRESS_HIGH = 0x76
// ADDRESS_LOW = 0x77

MS5803 sensor(ADDRESS_HIGH);
// Variables to store results
float temperature_c;
double depth;
double pressure_abs, pressure_baseline;

//EC sensor
// Converting EC to ppm
float TDSconversion = 0.7; //or 0.64
//Compensating for temperature
/**assuming a linear increase of conductivity versus
temperature of typically 2% per degree Celsius.
this changes depending on what chemical we are measuring*/
float TempCoef = 0.02; //

void sound_speed(float T, float S, float D);
double SoundSpeed;

// to produce a sine wave from arduino pin5 and 6 :
int i = 0;
```

```

int x = 0;
int OK = 0;
//the duty cycle for each pulse to produce 1 Khz sine wave
int sinPWM[] = {1, 51, 101, 146, 185, 216, 237, 248, 248, 237, 216, 185, 146, 101, 51, 1
    };

void setup() {
    // setting the serial port.
    Serial.begin(9600);

    // setting the bluetooth
    bluetooth.begin(115200);
    bluetooth.print("$$$");
    delay(100);
    bluetooth.println("U,9600,N");
    bluetooth.begin(9600);

//printing the header of the data with comma separate
    Serial.println("\n");
    Serial.print("D(m)");    //Depth in meter
    Serial.print(",");      // Comma separate variable
    Serial.print("T1(C)");  // Temperature from pressure sensor
    Serial.print(",");
    Serial.print("P(mbar)"); // Pressure millimbar
    Serial.print(",");
    Serial.print("T2(C)");  // temperature from DS18B20 sensor
    Serial.print(",");
    Serial.print("V1(v)");  //Output of differential amplifier
    Serial.print(",");
    Serial.print("V2(v)");  // Output of current to voltage amplifier
    Serial.print(",");
    Serial.print("EC(mS/cm)");
    Serial.print(",");
    Serial.print("EC25(mS/cm)");
    Serial.print(",");
    Serial.print("TDS(g/L)");
    Serial.print(",");
    Serial.println("TDS25(g/L)");
//BT
    bluetooth.println("\n");
    bluetooth.print("D(m)");    //Depth in meter
    bluetooth.print(",");      // Comma separate variable
    bluetooth.print("T1(C)");  // Temperature from pressure sensor
    bluetooth.print(",");
    bluetooth.print("P(mbar)"); // Pressure millimbar
    bluetooth.print(",");
    bluetooth.print("T2(C)");  // temperature from DS18B20 sensor
    bluetooth.print(",");
    bluetooth.print("V1(v)");  //Output of differential amplifier

```

```

bluetooth.print(",");
bluetooth.print("V2(v)"); // Output of current to voltage amplifier
bluetooth.print(",");
bluetooth.print("EC(mS/cm)");
bluetooth.print(",");
bluetooth.print("EC25(mS/cm)");
bluetooth.print(",");
bluetooth.print("TDS(g/L)");
bluetooth.print(",");
bluetooth.println("TDS25(g/L)");

//Start the sensors
sensor.reset();
sensor.begin(); //start pressure sensor
sensor1.begin(); // start the temperature sensor
//header(); // call the function the prints the header og the data table.

// initializing the Input signal of the conductivity sensor
pinMode(5, OUTPUT);
pinMode(6, OUTPUT);
cli(); // stop interrupts
TCCR0A = 0; //reset the value
TCCR0B = 0; //reset the value
TCNT0 = 0; //reset the value

TCCR0A = 0b10100001; //phase correct pwm mode
TCCR0B = 0b00000001; //no prescaler
OCR0A=128;

TCCR1A = 0; //reset the value
TCCR1B = 0; //reset the value
TCNT1 = 0; //reset the value
OCR1A = 509; // compare match value
TCCR1B = 0b00001001; //WGM12 bit is 1 and no prescaler

TIMSK1 |= (1 << OCIE1A); // enable interrupts

sei(); //start interrupts
}

ISR(TIMER1_COMPA_vect) { // interrupt when timer 1 match with OCR1A value
cli();
if (i > 14 && OK == 0) { // final value from vector for pin 6
i = 0; // go to first value of vector
OK = 1; //enable pin 5
}
if (i > 14 && OK == 1) { // final value from vector for pin 5
i = 0; //go to firs value of vector
}
}

```



```

    OK = 0; //enable pin 6
}
x = sinPWM[i]; // x take the value from vector corresponding to position i(i is zero indexed)
i = i + 1; // go to the next position
if (OK == 0) {
    OCR0B = 0; //make pin 5 0
    OCR0A = 509; //enable pin 6 to corresponding duty cycle
}
if (OK == 1) {
    OCR0A = 0; //make pin 6 0
    OCR0B = x; //enable pin 5 to corresponding duty cycle
}
sei();
}

```

```

void loop() {

```

```

    Depth(); // calling the function that finds the pressure and the depth
    EC(); // calling the function that finds the conductivity, salinity and temperature.
    //SoundSpeed = sound_speed(float T, float S, float D)
    SoundSpeed = sound_speed( temperature_c, TDS, depth);

```

```

}

```

```

//Function to find the depth of sea water based on pressure sensor MS5803

```

```

void Depth()

```

```

{
    temperature_c = sensor.getTemperature(CELSIUS, ADC_512); // measuring the temp
    pressure_abs = sensor.getPressure(ADC_4096); // measuring the pressure

```

```

//calculating the Depth

```

```

float P_atmosphere = 101060 ; //101325; // paskal
float g = 9.8 ; //m/s^2
float r = 1030; // kg/m^3 density of the sea water
//P = Patm.+ D*g*r ; D: depth, g: gravity, r: density of the seawater
depth = (pressure_abs * 100 - P_atmosphere) / (g * r); //m
Serial.print(depth);
Serial.print(",");
Serial.print(temperature_c);
Serial.print(",");
Serial.print(pressure_abs);

```

```

//BT

```

```

bluetooth.print(depth);
bluetooth.print(",");
bluetooth.print(temperature_c);
bluetooth.print(",");
bluetooth.print(pressure_abs);

```

```

//delay(1000);
/*
  if(blueetooth.available() // If the bluetooth sent any characters
  {
    // Send any characters the bluetooth prints to the serial monitor
    Serial.write((char)bluetooth.read());
  }
*/

if (Serial.available() // If stuff was typed in the serial monitor
{
  // Send any characters the Serial monitor prints to the bluetooth
  bluetooth.print((char)Serial.read());
}

//result from D= 3cm under the water:
//Pressure(mbar)= 1013.70 Depth(m)= 0.03 Temp_C = 25.00
//over the water D=0
//Pressure(mbar)= 1010.40 Depth(m)= -0.00 Temp_C = 24.00
}

// function to find the electric conductivity and TDS(salinity)
// Arrays to save the data
int samplings = 100;
int readV1[100];
int readV2[100];
int index = 0;

void EC()
{
  // Conductivity MEASUREMENTS
  // every iteration of loop makes one additional measurement,
  // the first 10 readV1 will display too low average
  int value1 = analogRead(A1); // input voltage from the electrodes at analog pins A1,A2
  int value2 = analogRead(A2);

  readV1[index] = value1;
  readV2[index] = value2;
  index++;
  if (index >= samplings) index = 0;
  //Min, Max
  int sensorMax1 = 0;
  int sensorMin1 = 1023;
  int sensorMax2 = 0;
  int sensorMin2 = 1023;
  for (int i = 0; i < samplings; i++)
  {
    if (readV1[i] > sensorMax1) sensorMax1 = readV1[i]; // the max value of the received
    data at A1
    //if (readV1[i] < sensorMin1) sensorMin1 = readV1[i];
  }
}

```

```

    if (readV2[i] > sensorMax2) sensorMax2 = readV2[i];
    //if (readV2[i] < sensorMin2) sensorMin2 = readV2[i]; // the max value of the received
data at A2
}

float V1 = sensorMax1 * 5.0 / 1023.0;    // Max output from differetial opamp
float V2 = sensorMax2 * 5.0 / 1023.0;    // Max output from current to voltage opamp
float Resistance = V1 / V2;              // Ohm's low V/I Resistance of liquid
float K = 0.15 ;                        // cell constant from the calibration of 30g/L saltwater
float resistivity = Resistance * K;      //ρ (rho)(Ω·m) ( =R.A/L)... A/L=K cell constant.
float conductivity = 1.0 / resistivity;  //mS/cm EC: Conductivity, σ, is defined as the
inverse of resistivity

/**Temperature Compensation ***/
sensor1.requestTemperatures();          // request the temp from sensor DS18B20.
Temp = sensor1.getTempCByIndex(0);

float EC25 = conductivity / (1 + TempCoef * (Temp - 25.0)); // mS/cm, compensation EC
float TDS25 = (EC25) * (TDSconversion); // g/L salinity at 25C
TDS = (conductivity) * (TDSconversion); // g/L at sampling temperature

//ptinting to the serial port and the SD card with comma separate
Serial.print(",");
Serial.print(Temp);
Serial.print(",");
Serial.print(V1);
Serial.print(",");
Serial.print(V2);
Serial.print(",");
Serial.print(conductivity);
Serial.print(",");
Serial.print(EC25);
Serial.print(",");
Serial.print(TDS);
Serial.print(",");
Serial.println(TDS25);

//Bluetooyh (BT)
bluetooth.print(",");
bluetooth.print(Temp);
bluetooth.print(",");
bluetooth.print(V1);
bluetooth.print(",");
bluetooth.print(V2);
bluetooth.print(",");
bluetooth.print(conductivity);
bluetooth.print(",");
bluetooth.print(EC25);

```

```

bluetooth.print(",");
bluetooth.print(TDS);
bluetooth.print(",");
bluetooth.println(TDS25);
//delay(200);

if (bluetooth.available()) // If the bluetooth sent any characters
{
  // Send any characters the bluetooth prints to the serial monitor
  Serial.write((char)bluetooth.read());

}
/*
if(Serial.available()) // If stuff was typed in the serial monitor
{
  // Send any characters the Serial monitor prints to the bluetooth
  bluetooth.print((char)Serial.read());

  if (bluetooth.available())
  Serial.write(blueetooth.read());
  if (Serial.available())
  bluetooth.write(Serial.read());

}
*/
}

void header()
{
  /*function to print the header of the data table,
  will be called just once from the setup loop */
  Serial.print("D(m)"); //Depth in meter
  Serial.print(","); // Comma separate variable
  Serial.print("T1(C)"); // Temperature from pressure sensor
  Serial.print(",");
  Serial.print("P(mbar)"); // Pressure millimbar
  Serial.print(",");
  Serial.print("T2(C)"); // temperature from DS18b20 sensor
  Serial.print(",");
  Serial.print("V1(v)"); //Output of differential amplifier
  Serial.print(",");
  Serial.print("V2(v)"); // Output of current to voltage amplifier
  Serial.print(",");
  Serial.print("EC(mS/cm)");
  Serial.print(",");
  Serial.print("EC25(mS/cm)");
  Serial.print(",");
  Serial.print("TDS(g/L)");
  Serial.print(",");
}

```

```

Serial.print("TDS25(g/L)");

}

// Function to find the sound speed in the water based on the water parameters
//using Medwi's expression

void sound_speed(float T, float S, float D)
{
  // using Medwi's sound speed expression
  double c ;
  c = 1449.2 + 4.6 * T + 5.5 * pow(10, -2) * pow(T, 2) - 2.9 * pow(10, -4) * pow(T, 3) + (1.34
- pow(10, -2) * T) * (S - 35.0) + 1.6 * pow(10, -2) * D;
}

```

9.1.2 Measuring temperature by Ds18B29 and sending the data via Bluetooth

```

//List of libraries
#include <OneWire.h>
#include <DallasTemperature.h>
#include <SoftwareSerial.h>

int bluetoothTx = 2;    // TX-O pin of bluetooth mate, Arduino D2
int bluetoothRx = 3;    // RX-I pin of bluetooth mate, Arduino D3

SoftwareSerial BT(bluetoothTx, bluetoothRx);

//Temp sensor
OneWire oneWire(8);    // data line on pin (8)
DallasTemperature sensor(&oneWire);    // instantiate the dallas temp. with one wire as a
//parameter

void setup() {

  sensor.begin();    // start the sensor
  Serial.begin(9600);    //setup serial port.
  BT.begin(115200);    // The Bluetooth Mate defaults to 115200bps
  BT.print("$$$");

  delay(100);    // Short delay, wait for the Mate to send back CMD
  BT.println("U,9600,N");    // Temporarily Change the baudrate to 9600, no parity

  BT.begin(9600);    // Start bluetooth serial at 9600
  BT.begin(9600);
}
void loop() {

```

```

sensor.requestTemperatures(); // request the temp.
float Temp;
Temp = sensor.getTempCByIndex(0);

Serial.println(Temp,2); // print contents of variable current temp 2
if(BT.available()) // If the bluetooth sent any characters
{
// Send any characters the bluetooth prints to the serial monitor
Serial.print((char)BT.read());
BT.print((char)BT.println(currentTemp,2));
BT.println(currentTemp,2);
}
if(Serial.available() // If stuff was typed in the serial monitor
{
// Send any characters the Serial monitor prints to the bluetooth
//BT.print((char)Serial.read());
BT.print((char)Serial.println(Temp,2));
BT.println(Temp,2);
}
}

```

9.1.3 Measuring the conductivity and the temperature

// This code measures the conductivity/salinity and the temperature

```

#include <OneWire.h>
#include <DallasTemperature.h>
//Temp sensor
OneWire oneWire(8) ; // pin (8)
DallasTemperature sensor(&oneWire); // instantiate the dallas temp. with one wire as a
parameter
unsigned long time;
/*
* In ISR function we set the OCR0A and OCR0B with duty cycle values
* and change this values according to vector,
* also at the finish of each crossing of vector we change the enabled pin.
* Before sine wave we must see if it is everything ok so we have alternate the pins
* at a stable duty cycle OCR0A and OCR0B equals with 128.
*/
int samplings = 100;
int readV1[100];
int readV2[100];
int index = 0;

int i=0;
int x=0;
int OK=0;
int sinPWM[]={ 1,51,101,146,185,216,237,248,248,237,216,185,146,101,51,1
};

void setup() {

```

```

Serial.begin(9600);
sensor.begin(); // staert the Temp. sensor

Serial.println("\n");
Serial.print("Time"); //time
Serial.print(","); // Comma separate variable
Serial.print("T(C)");
Serial.print(",");
Serial.print("V1(v)"); //Output of differential amplifier
Serial.print(",");
Serial.print("V2(v)"); // Output of current to voltage amplifier
Serial.print(",");
Serial.print("R");
Serial.print(",");
Serial.print("EC(mS/cm)");
Serial.print(",");
Serial.print("TDS(g/L)");
Serial.print(",");

Serial.print("EC25(mS/cm)");
Serial.print(",");
Serial.println("TDS25(g/L)");
pinMode(5, OUTPUT);
pinMode(6,OUTPUT);

cli();// stop interrupts
TCCR0A=0;//reset the value
TCCR0B=0;//reset the value
TCNT0=0;//reset the value
//0b allow me to write bits in binary
TCCR0A=0b10100001;//phase correct pwm mode
TCCR0B=0b00000001; //no prescaler
TCCR1A=0;//reset the value
TCCR1B=0;//reset the value
TCNT1=0;//reset the value
OCR1A=509;// compare match value
TCCR1B=0b00001001; //WGM12 bit is 1 and no prescaler

TIMSK1 |= (1 << OCIE1A);// enable interrupts

sei();//stop interrupts
}
ISR(TIMER1_COMPA_vect){// interrupt when timer 1 match with OCR1A value
if(i>14 && OK==0){// final value from vector for pin 6
i=0;// go to first value of vector
OK=1;//enable pin 5
}
if(i>14 && OK==1){// final value from vector for pin 5
i=0;//go to firs value of vector
}
}

```

```

OK=0;//enable pin 6
}
x=sinPWM[i];// x take the value from vector corresponding to position i(i is zero indexed)
i=i+1;// go to the next position
if(OK==0){
OCR0B=0;//make pin 5 0
OCR0A=x;//enable pin 6 to corresponding duty cycle
}
if(OK==1){
OCR0A=0;//make pin 6 0
OCR0B=x;//enable pin 5 to corresponding duty cycle
}
}
}
void loop()
{

time = millis();
//prints time since program started

// MEASUREMENTS
// every iteration of loop makes one additional measurement,
// so the first 10 readV1 will display too low average
int value1 = analogRead(A1);
int value2 = analogRead(A2);
readV1[index] = value1;
readV2[index] = value2;
index++;
if (index >= samplings) index = 0;

//running average, min and max
float total1 = 0;
float total2 = 0;
int sensorMax1 = 0;
int sensorMin1 = 1023;
int sensorMax2 = 0;
int sensorMin2 = 1023;
for (int i = 0; i< samplings; i++)
{
total1 += readV1[i];
total2 += readV2[i];
if (readV1[i] > sensorMax1) sensorMax1 = readV1[i];
//if (readV1[i] < sensorMin1) sensorMin1 = readV1[i];
if (readV2[i] > sensorMax2) sensorMax2 = readV2[i];
//if (readV2[i] < sensorMin2) sensorMin2 = readV2[i];
}
float average1 = total1 /samplings;
float average2 = total2 /samplings;
float Vp1= sensorMax1*5.0/1023.0;
float Vp2 = sensorMax2*5.0/1023.0;
float Resistance = Vp1/Vp2;          // Ohm's low V/I

```



```

float K= 0.15 ;           //temp 13. cell constant from the calibration
float resistivity = Resistance*K; //ρ (rho)(Ω·m) (=R.A/L)...>A/L=K cell constant.
float conductivity = 1.0/resistivity; //Conductivity(S/m), σ, is defined as the inverse of
resistivity
float TDS = 0.7* conductivity ; // 1000ppm=g/L--->

//Serial.print("\tAVG2:\t");
//Serial.println(average2*5.0/1023, 1);
sensor.requestTemperatures(); // request the temp.
float Temp;
Temp = sensor.getTempCByIndex(0);
float EC25 = conductivity/ (1+ 0.02*(Temp-25.0)); //mS/cm
float TDS25=(EC25)*(0.7); // g/L

// OUTPUT TO SERIAL
Serial.print(time);
Serial.print(",");
Serial.print(Temp,2); // print contents of variablr current temp 2
Serial.print(",");

Serial.print(Vp1);

Serial.print(",");
Serial.print(Vp2);
Serial.print(",");
Serial.print(Resistance);
Serial.print(",");
Serial.print(conductivity);
Serial.print(",");
Serial.print(TDS);
Serial.print(",");

Serial.print(EC25);
Serial.print(",");
Serial.println(TDS25);
delay(200);
}

```

9.1.3.2 Calibration

// Calibrating the conductivity sensor

```

#include <OneWire.h>
#include <DallasTemperature.h>
//Temp sensor
OneWire oneWire(8) ; // pin (8)
DallasTemperature sensor(&oneWire); // instantiate the dallas temp. whith one wire as a
parameter
unsigned long time;
/*
* In ISR function we set the OCR0A and OCR0B with duty cycle values
* and change this values according to vector,

```

* also at the finish of each crossing of vector we change the enabled pin.
 * Before sine wave we must see if it is everything ok so we have alternate the pins
 * at a stable duty cycle OCR0A and OCR0B equals with 128.
 */

```
int samplings = 100;
int readV1[100];
int readV2[100];
int index = 0;

int i=0;
int x=0;
int OK=0;
int sinPWM[]={ 1,51,101,146,185,216,237,248,248,237,216,185,146,101,51,1
};
```

```
void setup() {
  Serial.begin(9600);
  sensor.begin(); // staert the Temp. sensor
```

```
  Serial.println("\n");
  Serial.print("Time"); //time
  Serial.print(","); // Comma separate variable
  Serial.print("T(C)");
  Serial.print(",");
  Serial.print("V1(v)"); //Output of differential amplifier
  Serial.print(",");
  Serial.print("V2(v)"); // Output of current to voltage amplifier
  Serial.print(",");
  Serial.print("R");
  Serial.print(",");
  Serial.print("K");
  Serial.print(",");
  Serial.print("TDS(g/L)");
  Serial.print(",");
```

```
  Serial.print("EC25(mS/cm)");
  Serial.print(",");
  Serial.println("TDS25(g/L)");
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
```

```
cli();// stop interrupts
TCCR0A=0;//reset the value
TCCR0B=0;//reset the value
TCNT0=0;//reset the value
//0b allow me to write bits in binary
TCCR0A=0b10100001;//phase correct pwm mode
TCCR0B=0b00000001; //no prescaler
TCCR1A=0;//reset the value
```

```

TCCR1B=0;//reset the value
TCNT1=0;//reset the value
OCR1A=509;// compare match value
TCCR1B=0b00001001; //WGM12 bit is 1 and no prescaler

TIMSK1 |= (1 << OCIE1A);// enable interrupts

sei();//stop interrupts
}
ISR(TIMER1_COMPA_vect){// interrupt when timer 1 match with OCR1A value
if(i>14 && OK==0){// final value from vector for pin 6
i=0;// go to first value of vector
OK=1;//enable pin 5
}
if(i>14 && OK==1){// final value from vector for pin 5
i=0;//go to first value of vector
OK=0;//enable pin 6
}
x=sinPWM[i];// x take the value from vector corresponding to position i(i is zero indexed)
i=i+1;// go to the next position
if(OK==0){
OCR0B=0;//make pin 5 0
OCR0A=x;//enable pin 6 to corresponding duty cycle
}
if(OK==1){
OCR0A=0;//make pin 6 0
OCR0B=x;//enable pin 5 to corresponding duty cycle
}
}
}
void loop()
{

time = millis();
//prints time since program started

// MEASUREMENTS
// every iteration of loop makes one additional measurement,
// so the first 10 readV1 will display too low average
int value1 = analogRead(A1);
int value2 = analogRead(A2);
readV1[index] = value1;
readV2[index] = value2;
index++;
if (index >= samplings) index = 0;

//running average, min and max
float total1 = 0;
float total2 = 0;
int sensorMax1 = 0;
int sensorMin1 = 1023;

```

```

int sensorMax2 = 0;
int sensorMin2 = 1023;
for (int i = 0; i< samplings; i++)
{
  total1 += readV1[i];
  total2 += readV2[i];
  if (readV1[i] > sensorMax1) sensorMax1 = readV1[i];
  //if (readV1[i] < sensorMin1) sensorMin1 = readV1[i];
  if (readV2[i] > sensorMax2) sensorMax2 = readV2[i];
  //if (readV2[i] < sensorMin2) sensorMin2 = readV2[i];
}
float average1 = total1 /samplings;
float average2 = total2 /samplings;
float Vp1= sensorMax1*5.0/1023.0;
float Vp2 = sensorMax2*5.0/1023.0;
float Resistance = Vp1/Vp2;          // Ohm's law V/I
float TDS = 30 ;          // 1000ppm=g/L--->
float conductivity = TDS/0.7 ; //Conductivity(S/m),  $\sigma$ , is defined as the inverse of
resistivity
float resistivity = 1.0/ conductivity ;
float K= resistivity/Resistance ;          //temp 13. cell constant from the calibration
//float resistivity = Resistance*K;      // $\rho$  (rho)( $\Omega \cdot m$ ) ( =R.A/L)....>A/L=K cell constant.

//Serial.print("\tAVG2:\t");
//Serial.println(average2*5.0/1023, 1);
sensor.requestTemperatures(); // request the temp.
float Temp;
Temp = sensor.getTempCByIndex(0);
float EC25 = conductivity/ (1+ 0.02*(Temp-25.0)); //mS/cm
float TDS25=(EC25)*(0.7); // g/L

// OUTPUT TO SERIAL
Serial.print(time);
Serial.print(",");
  Serial.print(Temp,2); // print contents of variabl current temp 2
Serial.print(",");

Serial.print(Vp1);

Serial.print(",");
Serial.print(Vp2);
Serial.print(",");
Serial.print(Resistance);
Serial.print(",");
Serial.print(conductivity);
Serial.print(",");
Serial.print(TDS);
Serial.print(",");

Serial.print(EC25);

```

```

Serial.print(",");
Serial.println(TDS25);
delay(200);
}

```

9.1.4 Measuring the pressure/depth and the temperature:

// Measuring the pressure/depth and the temperature

```
#include <SparkFun_MS5803_I2C.h>
```

```
#include <Wire.h>
```

```
#include <OneWire.h>
```

```
#include <DallasTemperature.h>
```

```
//temperature sensor ds18b20
```

```
//Temp sensor
```

```
OneWire oneWire(8); // pin (8)
```

```
DallasTemperature sensor1(&oneWire); // instantiate the dallas temp. with one wire as a
parameter
```

```
unsigned long time;
```

```
//pressure sensor:
```

```
// ADDRESS_HIGH = 0x76
```

```
// ADDRESS_LOW = 0x77
```

```
MS5803 sensor(ADDRESS_HIGH);
```

```
//Create variables to store results
```

```
float temp_C, temp_F;
```

```
double pressure_abs;
```

```
float P_atmosphere = 101325; //101325; 101060 // paskal
```

```
float g = 9.8; //m/s^2
```

```
float r = 1030; // kg/m^3 density of the sea water
```

```
double depth;
```

```
void setup() {
```

```
  Serial.begin(9600);
```

```
  //Ds18b20 temp.sensor
```

```
  sensor1.begin(); // start the temp.sensor
```

```
  //Retrieve calibration constants for conversion math.
```

```
  sensor.reset(); //trykk sensor
```

```
  sensor.begin();
```

```
  //Header
```

```
  Serial.println("\n");
```

```
  Serial.print("Time"); //Depth in meter
```

```
  Serial.print(","); // Comma separate variable
```

```
  Serial.print("D(m)"); // Temperature from pressure sensor
```

```
  Serial.print(",");
```

```
  Serial.print("P(mbar)"); // Pressure millimbar
```

```
  Serial.print(",");
```

```

Serial.print("T1(C)"); // temperature from DS18b20 sensor
Serial.print(",");
Serial.print("T2(C)"); //Output of differential amplifier
Serial.print(",");
Serial.println("T2(F)");
}
void loop() {
//Serial.print("Time: ");
time = millis();
//prints time since program started

// Read temperature from the sensor in deg C. This operation takes about
temp_C = sensor.getTemperature(CELSIUS, ADC_512);

// Read temperature from the sensor in deg F. Converting
// to Fahrenheit is not internal to the sensor.
// Additional math is done to convert a Celsius reading.
temp_F = sensor.getTemperature(FAHRENHEIT, ADC_512);

// Read pressure from the sensor in mbar.
pressure_abs = sensor.getPressure(ADC_4096);

//P = Patm.+ D*g*r ; D: depth, g: gravity, r: density of the seawater
depth = (pressure_abs * 100 - P_atmosphere) / (g * r); //m

// temp DS18B20
sensor1.requestTemperatures(); // request the temp.
float currentTemp;
currentTemp = sensor1.getTempCByIndex(0);
Serial.print(time);

Serial.print(",");
Serial.print(depth); // m
Serial.print(",");
Serial.print(pressure_abs); // mbar
Serial.print(",");
Serial.print(currentTemp,2); // Temperature measured by DS18B20 in °C
Serial.print(",");
Serial.print(temp_C); // Temperature measured by MS5803 in °C
Serial.print(",");
Serial.println(temp_F); // Temperature measured by MS5803 in F

delay(200);
}

```

10.2 References

- [1] Urik, Principles of underwater sound. 3rd. edition.
- [2] Stephenson, E. B.: Velocity of sound in sea water, Phys. Rev. 21:181 (1923).
- [3] Heck N. H., and Jerry Service: Velocity of Sound in Sea Water. U. S. Coast and Geodetic survey. Special Publication No.108 1924
- [4] Matthew, D.J.: Tables of the velocity of sound in pure water and sea water for use in echo sounding and sound ranging. Admiralty Hydrog. Dep. Publ. 282, 1939.
- [5] Kuwahara, S.: Velocity of sound in sea water and calculation of the velocity of use in sonic sounding. Hydrogr. Rev. 16:123, 1939.
- [6] Alfred Weissler and Vincent A. Del Grosso. The Velocity Of Sound in Sea Water. Naval Research Laboratory, Washington, D.C. J. Acoust. Soc. Am. , 23:219 (1951).
- [7] Del Grosso, V. A. Velocity of Sound in Se Water at Zero Depth, U. S. Nav. Res. Lab. Rep. 4002, 1952.
- [8] Wilson, W. D. : Speed of Sound in Sea Water as a Function of Temperature, Pressure and Salinity, J. Acoust. Soc. Am. 32:641 (1960).
- [9] Leroy, C. C. : Development of Simple Equations for Accurate Ans More Realistisk Calculation of the Speed of Sound in Sea Water, J. Acoust. Soc. Am., 46:216 (1969).
- [10] Medwin, H.: Speed of Sound in Water For Realistic Parameters, J. Acoust. Soc. Am., 58:1316(1975).
- [11] Mackenzie, K. V.: Nine-term Equation for Sound Speed in the Ocean, J. Acoust. Soc. Am., 70:807(1981).
- [12] Del Grosso, V. A. New equation for th sound speed of sound in Natural Water with comparison to other equations. J. Acoust. Soc. Am. 56:1084 (1974).
- [13] C-T. Chen and F.J. Millero, Speed of sound in seawater at high pressures J. Acoust. Soc. Am. 62(5) pp 1129-1135. (1977).
- [14] UNESCO (1983): Algorithms for computation of fundamental properties of seawater. UNESCO technical papers in marine science 44:1-55.
- [15] SAIV A/S, STD/CTD model SD204 , <http://www.saivas.no/visartikkel.asp?art=2>
- [16] Valeport, <http://www.valeport.co.uk/blog/measurement-of-the-speed-of-sound-in-seawater>
- [17] Region of sound velocity profiler. <https://www.usna.edu/Users/physics/ejtuchol/documents/SP411/Chapter5.pdf>
- [18] John Marshall and R. Alan Plumb. ATMOSPHERE, OCEAN, AND CLIMATE DYNAMICS: AN INTRODUCTORY TEXT.
- [19] Discovery of sound in the sea. Retrieved from: <http://www.dosits.org/science/soundmovement/speedofsound/>
- [20] Sound of speed in the sea. Retrieved from : <https://www.usna.edu/Users/physics/ejtuchol/documents/SP411/Chapter4.pdf>
- [21] Principles of Naval Weapons Systems, Edited by Joseph B. Hall, CDR, USN, Dubuque, IA: Kendall/Hunt Publishing Co, 2000, p.179
- [22] Underwater and sound velocity propagation, retrieved from : <http://www.arc.id.au/UWAcoustics.html>
- [23] Orion conductivity theory. http://www.fondriest.com/pdf/thermo_cond_theory.pdf
- [24] Gray, James R. (2004). "*Conductivity Analyzers and Their Application*". In Down, R.D; Lehr, J.H. Environmental Instrumentation and Analysis Handbook. Wiley. pp. 491–510. ISBN 978-0-471-46354-2.
- [25] John J. Barron & Colin Ashton. *The Effect of Temperature on Conductivity Measurement* Technical Services Department, Reagecon Diagnostics Ltd, Shannon Free Zone, County Clare, Ireland

[26] MASAKI HAYASHI (TEMPERATURE-ELECTRICAL CONDUCTIVITY RELATION OF WATER FOR ENVIRONMENTAL MONITORING AND GEOPHYSICAL DATA INVERSION) *Department of Geology and Geophysics, University of Calgary, Calgary, Alberta, Canada*

[27] Figure 1. <http://www.ck12.org/user:cnhdz251ckbuzzjlmxmi5tby51cw../book/NEVC-Physical-Science-for-NGSS-9th-Grade/section/42.0/>

[28] Conductivity theory and practice .

http://www.analytical-chemistry.uoc.gr/files/items/6/618/agwgimometria_2.pdf
(11.02.2017)

[29] Conductivity, Salinity & Total Dissolved Solids (Fundamentals of environmental measurements)

<http://www.fondriest.com/environmental-measurements/parameters/water-quality/conductivity-salinity-tds/#cond3>

[30] The Basis Of conductivity in the LAQUA Horiba scientific. Retrieved from:

<http://www.horiba.com/application/material-property-characterization/water-analysis/water-quality-electrochemistry-instrumentation/the-story-of-ph-and-water-quality/the-basis-of-conductivity/the-concept-of-conductivity/>

[31] Miller, R. L., Bradford, W. L., & Peters, N. E. (1988). Specific Conductance: Theoretical Considerations and Application to Analytical Quality Control. In U.S. Geological Survey Water-Supply Paper. Retrieved from <http://pubs.usgs.gov/wsp/2311/report.pdf>.

[32] Engineering , Conductivity electrodes measurements. Retrieved from

http://www.globalspec.com/learnmore/sensors_transducers_detectors/analytical_sensors/electrodes_conductivity

[34] Half wave rectifier waves figure retrieved from : <http://www.circuitstoday.com/half-wave-rectifiers>

[35] Richard J. Higinns. Electronics With Digital and Analog Integrated Circuits.

[36] Wiki, Analog Devices. CH17 Diode Application Topics. Retrieved from:

<https://wiki.analog.com/university/courses/electronics/text/chapter-7>

[37] Electronics projects and Arduino programming, Eprojectszone. Retrieved from :

<http://www.eprojectszone.com/2016/08/21/how-to-generate-a-sine-wave-from-arduino-or-atmega-328/>

[38] *Timothy Hirzel*, PWM. Arduino . Retrieved from: <https://www.arduino.cc/en/Tutorial/PWM>

[39] Sverre Grimnes, Ørjan G. Martinsen. Bioimpedence.

http://www.mn.uio.no/fysikk/english/research/projects/bioimpedance/publications/papers/encyc_bme.pdf

[40] Sverre Grimnes, Ørjan G. Martinsen. Sources of error in tetrapolar impedance measurements on biomaterials and other ionic conductors. Published 15 December 2006.

<http://www.mn.uio.no/fysikk/english/research/projects/bioimpedance/publications/papers/fouelec.pdf>

[41] John Marshall and R. Alan Plumb ATMOSPHERE, OCEAN, AND CLIMATE DYNAMICS: INTRODUCTORY TEXT.

AN

[42] Jacob Fraden. Handbooks of Modern Sensors. Fourth edition, 2010

[43] Phil Levhenko, JumperOneTv, <https://www.youtube.com/watch?v=hijgA8A-O64>.

[44] DS18B20 Datasheet, <http://datasheets.maxim-ic.com/en/ds/DS18B20.pdf>

[45] DS18B20, Dallas Semiconductor <http://cdn.sparkfun.com/datasheets/Sensors/Temp/DS18B20.pdf>

[46] Overview of 1-Wire Technology and Its Use. *Toturila1796*. Maxim integrated. Retrieved from: <https://www.maximintegrated.com/en/app-notes/index.mvp/id/1796>

[47] AVR318: Dallas 1-Wire Master on tinyAVR and megaAVR, Atmel. Retrieved from:

http://www.atmel.com/images/Atmel-2579-Dallas-1Wire-Master-on-tinyAVR-and-megaAVR_ApplicationNote_AVR318.pdf

[48] Guidelines for Reliable Long Line 1-Wire Networks, Maxim integrated. Retrieved from:

<https://www.maximintegrated.com/en/app-notes/index.mvp/id/148>

- [49] 1-Wire Enumeration, Texas instrument. Retrieved from: <http://www.ti.com/lit/an/spma057b/spma057b.pdf>
- [50] Dallas temperature library. <https://github.com/milesburton/Arduino-Temperature-Control-Library>
- [51] one-wire library, https://www.pjrc.com/teensy/td_libs_OneWire.html
- [52] Arduino thermometer using DS18B20. Arduino hub. <https://create.arduino.cc/projecthub/TheGadgetBoy/ds18b20-digital-temperature-sensor-and-arduino-9cc806>
- [53] Arduino and Ds18B20. Get Micros <http://www.getmicros.net/arduino-ds18b20-example.php>
- [54] Pressure sensor, https://en.wikipedia.org/wiki/Pressure_sensor
- [55] MS5803-14BA Miniature 14 bar Module. Data Sheet. https://cdn.sparkfun.com/datasheets/Sensors/Weather/ms5803_14ba.pdf
- [56] I2C, Sparkfun tutorial. <https://learn.sparkfun.com/tutorials/i2c>
- [57] https://github.com/sparkfun/SparkFun_MS5803-14BA_Breakout_Arduino_Library/tree/V_1.1.0
- [58] How to measure salinity: <http://www.wikihow.com/Measure-Salinity>
- [59] Atlas Scientific. https://www.atlas-scientific.com/product_pages/kits/ec_k1_0_kit.html
- [60] Electrical conductivity Kit, Sparkfun. <https://www.sparkfun.com/products/12908>
- [61] EZO Conductivity circuit. https://cdn.sparkfun.com/datasheets/Sensors/Biometric/EC_EZO_Datasheet.pdf
- [62] Memory, Arduino. <https://www.arduino.cc/en/tutorial/memory>
- [63] SparkFun Openlog. <https://www.sparkfun.com/products/13712>
- [64] Bluetooth, SparkFun. https://learn.sparkfun.com/tutorials/bluetoothbasics?_ga=1.201359510.668403741.1489761568
- [65] Bluetooth data module..., user Guide. http://cdn.sparkfun.com/datasheets/Wireless/Bluetooth/bluetooth_cr_UG-v1.0r.pdf
- [66] How to power your Arduino project, apc. <http://apcmag.com/how-to-power-your-arduino-projects.htm/>
- [67] How to power a project. <https://learn.sparkfun.com/tutorials/how-to-power-a-project>
- [68] Feeding power to Arduino: the ultimate guide, OpenElectronics. <https://www.open-electronics.org/the-power-of-arduino-this-unknown/>
- [69] Simple Wireless Power. <http://www.instructables.com/id/Simple-Wireless-Power/>
- [70] Wireless power transfer modules. Sunrom, <http://www.sunrom.com/p/wireless-power-transfer-modules>
- [71] http://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwjlp-O2j7TTAhVEXiwKHVPGBs0QFggsMAE&url=http%3A%2F%2Fwww.ewa-online.eu%2Fid-9th-ewa-brussels-conference.html%3Ffile%3Dtl_files%2F_media%2Fcontent%2Fdocuments_pdf%2FPublications%2FProceedings%2FBrussels-Conference%2F2013_9th%2520Brussels%2520Conference%2FEWA_9th_Brussels_Conference_D2_4-Thaulow.pdf&usq=AFQjCNHz6PBJApvuL7S3PkCiYhv_QqDrmg&sig2=hxE9WA5siZw_HZTpytS3Dg
- [72] Dissolved Oxygen Kit. <https://www.sparkfun.com/products/11194>
- [73] PH sensor Kit. <https://www.sparkfun.com/products/10972>