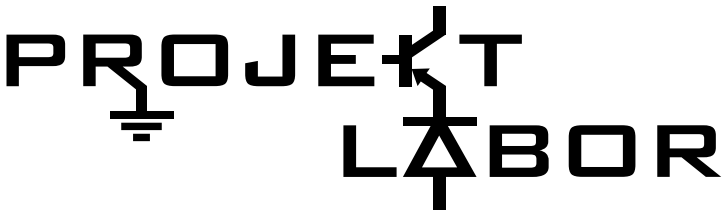
Abschlussbericht

Green-Lab

Gruppe 5: Nährstoffe

SS 2018

Projekt B



16. Juli 2018

Fachgebiet Mikroelektronik-Aufbau- und Verbindungstechniken
Leitung (kommissarisch): Prof. Dr.-Ing.Dr.sc.techn. Klaus-Dieter Lang
Fachgebiet Elektronik und medizinische Signalverarbeitung
Leitung: Prof. Dr.-Ing. Reinhold Orglmeister

Inhaltsverzeichnis

1 Gruppe 5 - Nährstoffe

1.0.1 Einleitung

Um eine geregelte Nährstoffversorgung der Pflanzen zu sichern müssen wir (wie auch im Blockschaltbild zu erkennen ist) den Säure- und Nährstoffgehalt im Wasser überwachen und regulieren. Die Messung dieser wird von zwei Gruppen mit jeweils 2 Mitgliedern und ihren selbst modellierten Bauteilen durchgeführt. Auch der Entwurf einer Pumpe und eines Füllstandmessers sind von großer Bedeutung, da die Pumpe unter keinen Umständen Luft ziehen sollte. All diese Prozesse werden an die Microcontroller-Gruppe (ebenfalls zwei Mitglieder) weitergegeben, die ihre Pflanzenspezifischen Nährstoff- und Wasserwerte von der Interface-Gruppe erhählt. Der Microcontroller regelt alle Prozesse, anhand der Vorgegebenen Werte für eine optimale dem Gewächs entsprechende Versorgung.

Hendrik Lottermoser

1.1 Team Nährstoffmessung

1.1.1 Einleitung

Zur optimalen Nährstoffversorgung der Pflanzen soll der Nährstoffgehalt im Haupttank geregelt werden. Dafür wird mittels eines selbst gebauten EC-Sensor's die Leitfähigkeit des Wassers bestimmt, um über diesen Wert auf den Nährstoffgehalt zurück zu schließen.

1.2 Blockschaltbild

Funktionsweise

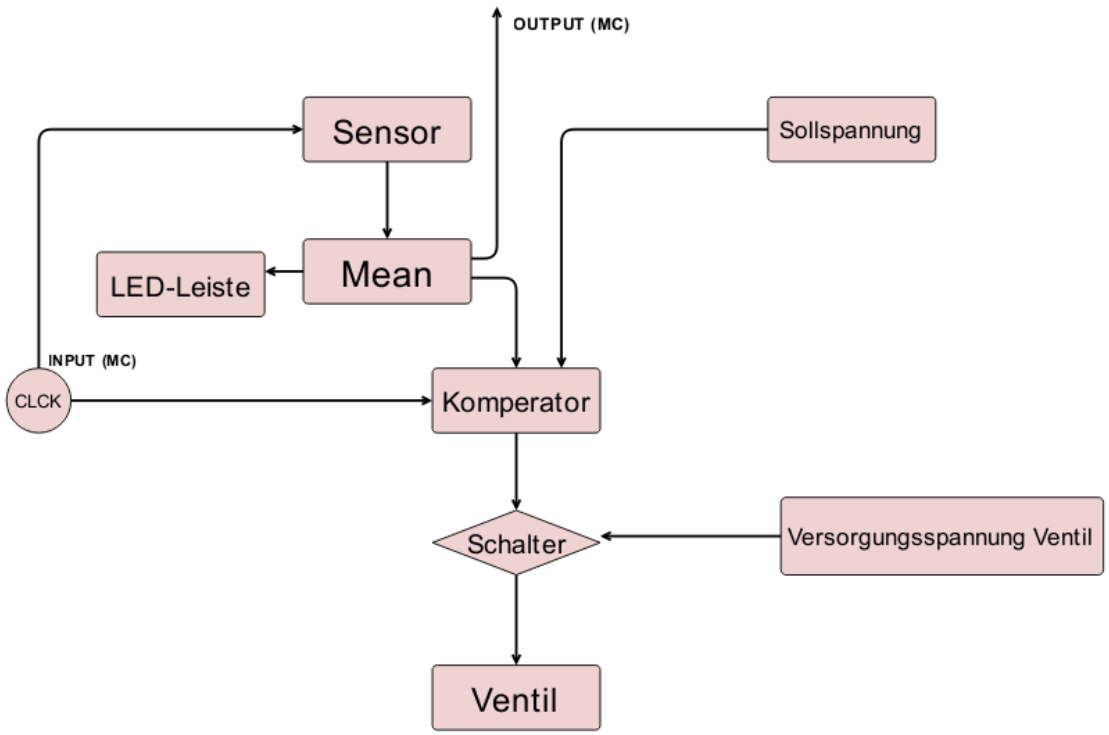


Abbildung 1: Blockschaltbild für den Nährwert-Regelkreis

Die bereitstellung eines optmialen Nährstofgehaltes erfolgt über eine Leitfähigkeitsmessung mit analoger Regelung. Messung und Regelung des Nährstoffgehaltes darf nicht zeitgleich mit der Regelung des Ph-Wertes stattfinden. Daher wird der EC-Sensor durch ein clock-Signal des Mikrocontrollers gesteuert. Ist dieses high misst der Sensor. Zunächst werden die Werte Tiefpassgefilter um zu schnelle Wertänderung herauszufiltern und damit Fehler zu minimieren. Dannach vergleicht ein Komperator den Ist-Wert (zwischen -0,5V und 5V) mit dem von der Mikrokonrtoller-Gruppe bereitgestellten Sollwert. Ist der Ist-Wert zu niedig öffnet ein Schlater das Ventil zum Nähstofftank. Dies wird durch eine grüne LED visualisiert. Bleibt das Ventil geschlossen leuchtet eine rote LED.

Jonas Mader

1.2.1 Schaltplan

Günstiger weise haben wir eine Seite gefunden, auf der dokumentiert wurde wie ein EC-Sensor selbst gebaut wurde. Zunächst haben wir diesen Schaltplan analysiert und simuliert.
Die Schaltung des Sensor besteht aus mehreren Teile. Der erste ist ein Wien-Robinson-Oszillator. Diese Baugruppe wird benötigt um eine oszillierende Spannungskurve zu erzeugen, die anschließend an den einen Kontakt der Probe angelegt wird, um zu messen wie viel der Spannung über die Lösung abfällt. Daraus lässt sich schließen welchen Leitwert die Lösung hat.
Eine wechselnde Spannung ist dabei nötig damit die sich nicht die Moleküle mit der Zeit an den Probenkontakten ansammeln und das Messergebnis verfälschen.
Im Prinzip ist die Brücke eine Kombination aus Bandpassfilter und Spannungsteiler, die an einen Operationsverstärker angeschlossen sind. Der nicht invertierende Eingang ist der Ausgang, der über einen Hoch und Tiefpass (RC-Glied parallel und in Reihe) rückgekoppelt wird. An dem Operationsverstärker gilt $V_{out} = V_{in} \frac{Z_2}{Z_1+Z_2}$ Dabei soll keine Phasenverschiebung auftreten, deswegen muss der Imaginärteil der Gesamtimpedanz Null sein. Aus dieser Bedingung lässt sich die Frequenzen des Erzeugten Sinussignals ableiten.
Damit hat der Rest der Impedanz einen Rein Ohmschen Anteil. Ermittelt man diesen, lassen sich die Parameter für den Spannungsteiler ermitteln, über den der Ausgang des Verstärkers auf den invertierenden Eingang rückgekoppelt wird. Denn die gesamte Verstärkung der Schaltung soll 1 sein.

Das Signal der Leitung in der Probe wird verstärkt und gleichgerichtet. Dadurch entsteht eine charakteristische Rechteckspannung am Ausgang dieser Schaltung. Diese Tiefpassfiltern wir mit einem einfachen RC-Glied um eine charakteristischen Gleichspannungswert zu erhalten. Dieser spiegelt den aktuellen EC-Wert des Tanks wieder.

Hannes Lorkowski

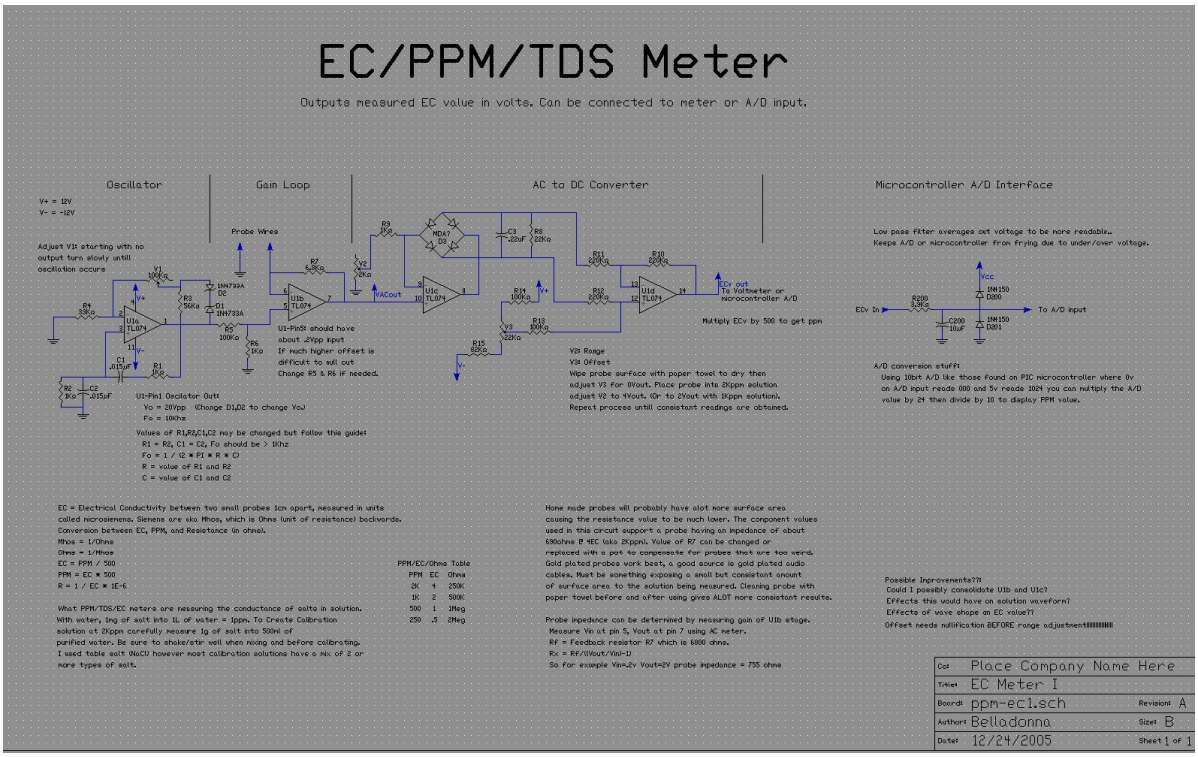


Abbildung 2: Schaltplan für den EC-Sensor. Quelle: `http://www.octiva.net/projects/ppm/`

Abbildung 3: Eigener Schaltplan des EC-Sensors

Das Ausgangssignal des EC-Sensors wird an den Eingang eines invertierenden Schmitt-Triggers geleitet, welcher den Wert mit dem von der Mikrocontroller-Gruppe bereitgestellten Referenz vergleicht. Je Höher der Spannungswert ist, desto höher ist auch der Leitwert im Tank. Daher soll sich das Ventil öffnen wenn die Eingasspannung die Referenz unterschreitet. Die Widerstände haben wir so dimensioniert, dass die Referenz mit einer Hysterese von $100mV$ in beide Richtungen beaufschlagt wird. Diesen Wert haben wir durch Tests für Sinnvoll erachtete. Das Ausgangssignal des Schmitt-Triggers wird zum einen an das Gate eines N-Mosfet's gelegt, und zum anderen dient es als Eingangssignal eines invertierenden Komperators. An dem Drainpotential des Mosfet's hängt das Ventil. Leitet dieses Öffnet sich unser Ventil und es fließen Nährstoffe in den Tank. Außerdem fließt ein Strom durch unsere grüne LED. Hierbei war zu beachten eine Diode parallel zu unserem Ventil zu schalten. Das Ventil

funktioniert durch eine Induktivität. Schließt sich der Transistor würde der Induktionsstrom des Ventils anderenfalls die LED kaputt machen.
Wie erwähnt liegt der Ausgang des Schmitt-Triggers an einem invertierenden Komperator. Der Ausgang des Komperators liegt am Gate-Anschluss es N-Mosfets. Leitet dieser fließt so ein Strom durch unsere Rote LED. Als Referenz nutzen wir das Refferenzsignal des EC-Wertes, da es zwischen der High- und der Lowspannung des Schmitt-Triggers liegt.

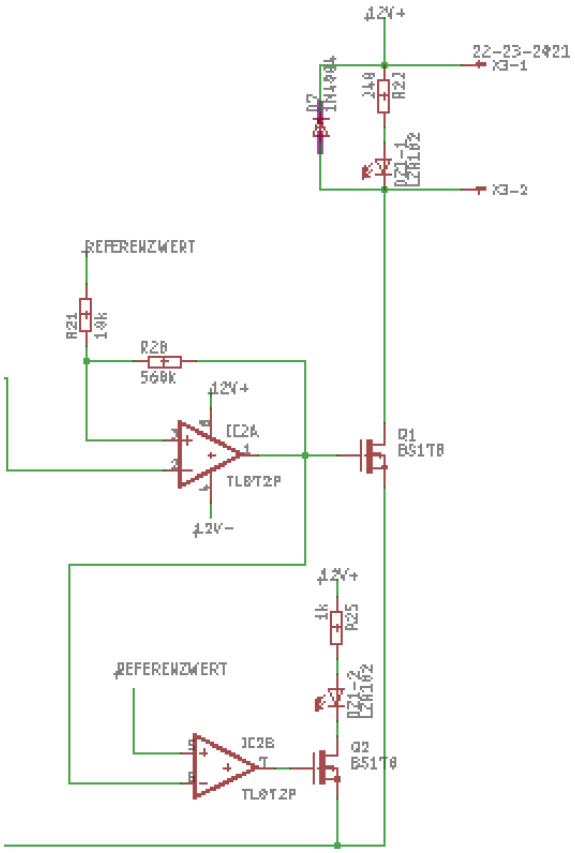


Abbildung 4: Schaltplan der Regelung

Wie eingangs erwähnt dürfen wir nicht den EC-Wert messen während die Ph-Gruppe misst, da der Stromfluss die Ph-Messung beeinflussen würde. Daher Bekommen wir ein Taktsignal von unserer internen MC-Gruppe. Dieses triggert ein Relais, welches unsere Schaltung von der Spannungsversorgung trennt.

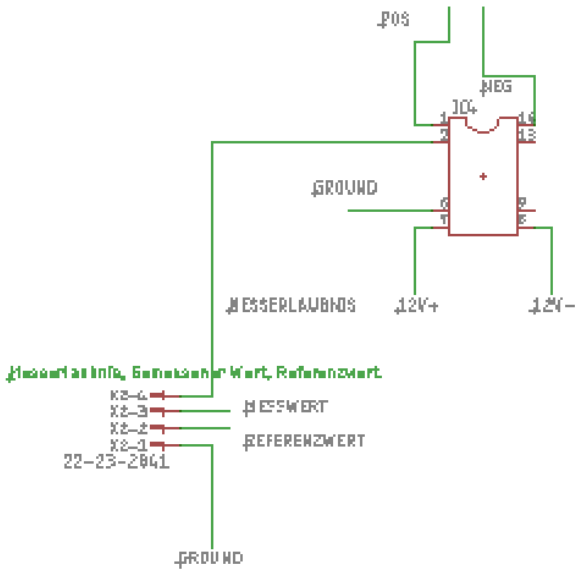


Abbildung 5: Schaltplan der Messerlaubnis

Jonas Mader

1.2.2 Simulation

Der Schaltplan für den EC-Sensor wurde in Ltspice übernommen. Leider war der Schaltplan den wir im Internet gefunden haben nicht in dem selben Programm erstellt. Einige spezifische Bauteile, nämlich Zenerdioden und Operationsverstärker konnten wir in nicht in der vorliegenden Form nicht in der Bibliothek unseres Programms finden. Potentiometer sind zusätzlich nicht in Ltspice verfügbar. Deswegen mussten wir diese Bauteile durch andere Modelle und im Falle der Potentiometer durch einfach Widerstände substituieren.
Das stellte uns jedoch vor Probleme, die Schaltung funktionierte nicht. Daraufhin fanden wir einen Schaltkreis im Internet, der nur die Wein-Brücke darstellte. Dieser war in Ltspice erstellt worden, und funktionierte auch bei uns. Der Unterschied zu dem vorherigen von uns erstellten Schaltplan auf Basis des von Isabella war, dass der Operationsverstärker ein anderer War, und statt Zener-Diode zwei normale parallel in Gegenrichtung geschaltete Dioden benutzt wurden. Nach übertragen in den Schaltplan mit dem Rest des Sensors, funktionierte dieser. Das Problem stellten vermutlich die Zenerdioden dar.

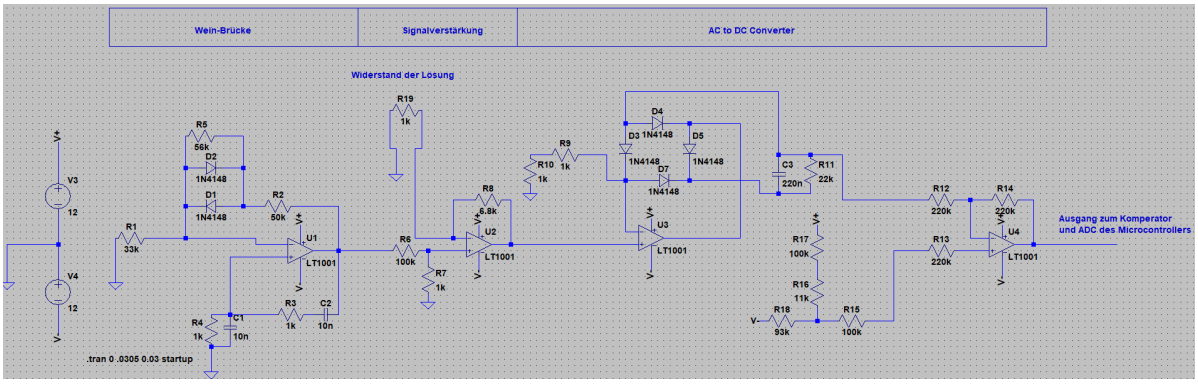


Abbildung 6: Schaltplan der Simmulation

Durch Simulation sind folgende Dinge deutlich geworden:
Je höher der Wert des Widerstandes des Potetiometer bei der Diodenbrücke, desto geringer ist der Potentialabstand zwischen dem positiven und negativen Potential der Brücke. Im Falle von Isabella stellte sie diesen Abstand auf 2V ein.
Sobald der invertierende Eingang des Operationsverstärkers der Wein-Brücke nicht mehr verbunden war, stellte sich am Ausgang dieses ein Rechtecksignal dar. Dies könnte von nutzen sein Falls uns die Sinuskurve vor Probleme stellt. Dabei war die Taktrate des Signals geringer, je höher der Widerstand des RC-Kreises der den Ausgang rückkoppelt war.

Hannes Lorkowski

1.2.3 Schnittstellen

Zur Spannungsversorgung benötigen wir wie im Schaltplan angegeben -12V und 12V Potentiale. Diese Potentiale nutzen wir gleichzeitig als Betriebsspannung unserer Komperatoren für die Regelung.
Von der Untergruppe die für den Microcontroller zuständig ist benötigen wir eine Spannung von 0-5V als Referenzwert für unsere Regelung. Zusätzlich geben wir an diese Gruppe den tatsächlichen EC-Wert, den wir messen. von dieser Gruppe bekommen wir auch eine Taktung, sprich sie koordiniert, wann wir messen und Nährlösung nachregeln. Die Übertragung läuft über einen internen Bus.
Zur Kalibrierung unserer Schaltung sind die 3 Potentiometer eingebaut. Das Potentiometer R7 ist dafür dar den Wien-Robbnison-Oszillator zum schwingen zu bringen, die Sinuswelle zu dimensionieren und die Frequenz einzustellen. Bei unserer Einstellung ergab sich eine Fre-

quenz von 26 kHz, die schnell genug ist, um eine Ansammlung von Mineralien an einer der beiden Elektroden des Sensors zu verhindern, was den Leitwert verfälschen würde.

1.2.4 Kalibrierung

Die Amplitude unseres Signals hat reicht ca von -9 bis $+9$ Volt.
Zur Kalibrierung des Messwertes geht man wie folgt vor:
Man misst den Ausgang des Operationsverstärkers an dem das Rechteck-Signal anliegt. Anschließend passt man den Widerstand des Potentiometers V3 (R26), mit einer sich in der Luft befindenden Probe, so lange an, bis die Ausgangsspannung 0 V beträgt. Dann bereitet man eine auf 2000 ppm (4 EC) kalibrierte Salzlösung (2 g Salz pro Liter) vor, und versenkt die Probe in dieser. Nun ist das Potentiometer V2 (R10) so anzupassen das der Ausgang 4 V (entspricht 4 EC) erreicht. Anschließend nimmt man die Probe aus der Lösung trocknet sie ab, und wiederholt die Anpassung von V3 auf 0 Volt. diese Schritte widerholt man mehrere male (3-4), bis sich ohne Anpassung die Werte am Ausgang einstellen. Sollten sich die gewünschten Werte nur schwer einstellen lassen, sollte man möglicherweise den Widerstand R8 austauschen, weil dann die Probe zu sehr von der den Isabelle bei ihrer Schaltung verwendet hat abweicht. Der Messwert sollte auf ca 0.05 Volt, bzw EC genau sein. Die Anleitung zur Kallibrierung wurde so von Isabelle vorgeschlagen.

Hannes Lorkowski

1.2.5 Herausforderung

Wir haben Ursprünglich erwartet das wir immer positive Spannungen am Ausgang des EC-Sensors erhalten. Daher haben wir unseren Schmitt-Trigger mit Null und Zwölf Volt betrieben. Die Tests haben gezeigt, dass die Spannung für Leitungswasser bei circa $-0.4V$ liegt. Der Schmitt-Trigger, welcher mit einem OPV aufgebaut ist funktioniert nicht gut für Eingangsspannungen außerhalb der Betriebsspannung. Dadurch mussten wir das negative Potential des OPV's von der Groundfläche trennen und mit einem Seperaten Kabel mit einer $-12V$ Leitung verbinden.
Weiterhin ist das Ventil relativ dick, wodurch auch bei kurzer Öffnungszeit eine relativ große Menge Nähstofflösung in den Tank fließt. Das macht eine genaue Regelung sehr schwierig. Wir können das Problem aber leicht vermeiden in dem wir einen dünneren Schlauch an das Ventil hängen oder die Nähstofflösung stark verdünnen.

1.2.6 Fazit

Die anfänglich gestellten Forderungen haben wir mit unser Platine erfüllt. Wir haben einen funktionierenden EC-Sensor und eine funktionierende Regelung für den Nährstoffgehalt.
Bis zur Beendigung des Projektes wollen wir noch eine Kennlinie aufnehmen, sodass die MC-Gruppe mittels der Spannung einen EC-Wert ausgeben kann.

1.2.7 Platinenlayout

Die Platine wurde mit Eagle designed. Im folgenden sind Abbildungen von den Leiterbahnen der Oberseite (Rot) und der Unterseite (Blau) dargestellt. Dabei sind der Übersicht halber die Darstellung der Masseflächen ausgeschalten worden.

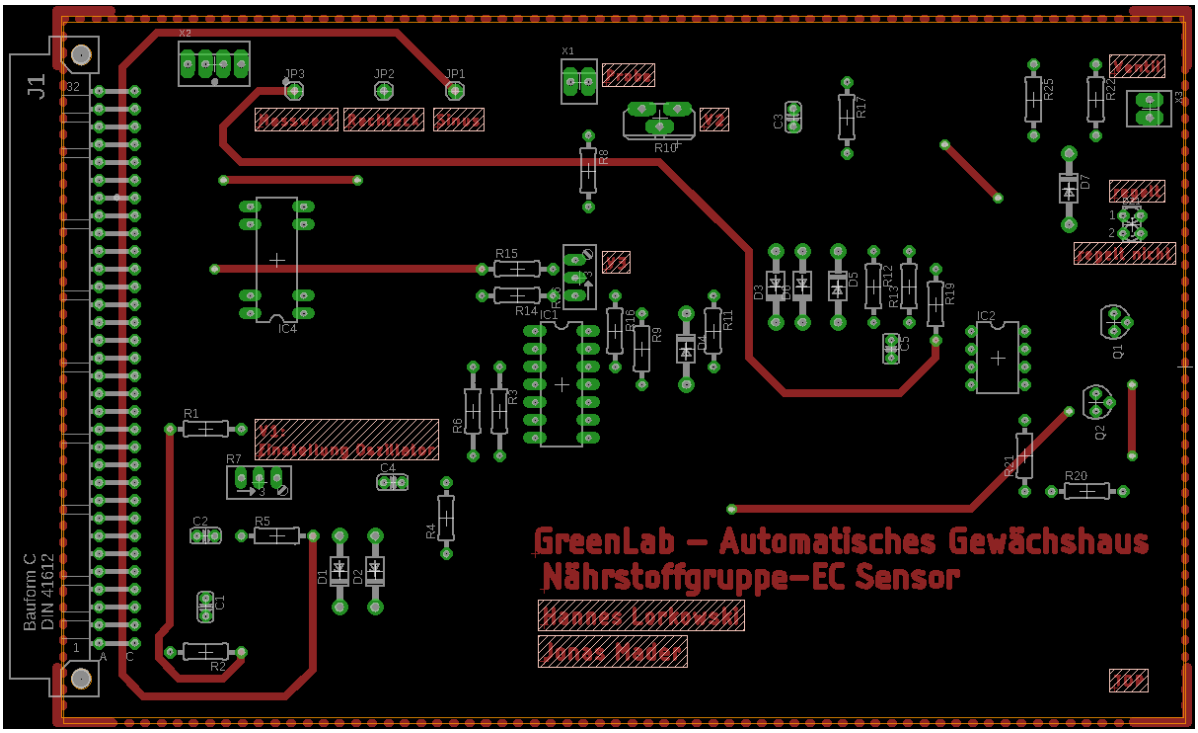


Abbildung 7: Abbildung des Top-Layers der Platine

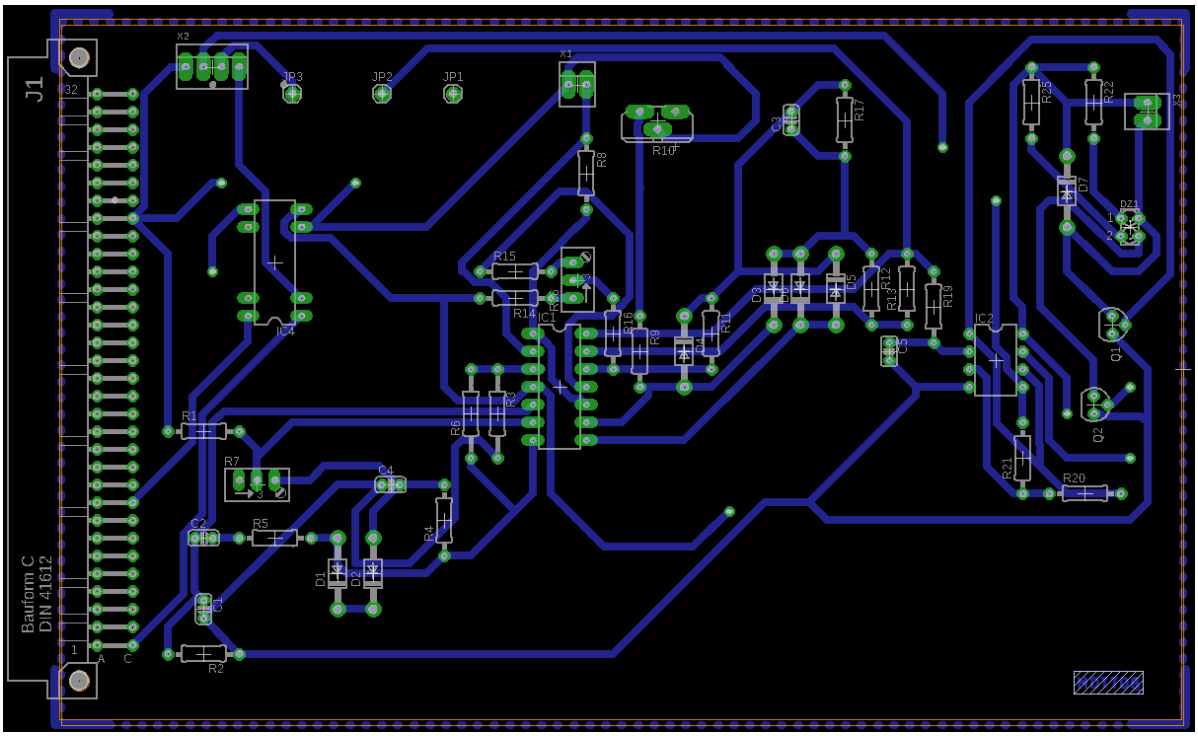


Abbildung 8: Abbildung des Bottom-Layers der Platine

1.2.8 Bautteilliste

In der folgende Liste sind die in Eagle verwendeten Bauteile zu finden. Teilweise wurden andere Bauteile beim Platinenaufbau verwendet, da die gelisteten nicht vorhanden waren. Zusätzlich wurde ein Magnetventil, ein Relais und ein EC-Sensor verwendet.

Tabelle 1: Eagle-Bauteilliste

Part	Value	Device	Package	Library	Sheet
C1	15n	C-EU025-024X044	C025-024X044	resistor	1
C2	15n	C-EU025-024X044	C025-024X044	resistor	1
C3	220n	C-EU025-024X044	C025-024X044	resistor	1
C4	2.2n	C-EU025-024X044	C025-024X044	resistor	1
C5	470n	C-EU025-024X044	C025-024X044	resistor	1
D1	1N4728	1N4728	DO41Z10	diode	1
D2	1N4728	1N4728	DO41Z10	diode	1
D3	1N4004	1N4004	DO41-10	diode	1
D4	1N4004	1N4004	DO41-10	diode	1
D5	1N4004	1N4004	DO41-10	diode	1
D6	1N4004	1N4004	DO41-10	diode	1
D7	1N4004	1N4004	DO41-10	diode	1
DZ1	LZR182	LZR182	LZR182	led	1
IC1	TL074P	TL074P	DIL14	linear	1
IC2	TL072P	TL072P	DIL08	linear	1
IC4	DIL14/8	DIL14/8	ic-package		1
J1	VG64	VG64	19inch		1
JP1	PINHD-1X1	1X01	pinhead		1
JP2	PINHD-1X1	1X01	pinhead		1
JP3	PINHD-1X1	1X01	pinhead		1
Q1	BS170	BS170	SOT54E	transistor-small-signal	1
Q2	BS170	BS170	SOT54E	transistor-small-signal	1
R1	33k	R-EU_0207/10	0207/10	resistor	1
R2	1K	R-EU_0207/10	0207/10	resistor	1
R3	1k	R-EU_0207/12	0207/12	resistor	1
R4	56k	R-EU_0207/10	0207/10	resistor	1
R5	1k	R-EU_0207/10	0207/10	resistor	1
R6	100k	R-EU_0207/12	0207/12	resistor	1
R7	100k	TRIM_EU-RJ9W	RJ9W	pot	1
R8	6.8k	R-EU_0207/10	0207/10	resistor	1
R9	1k	R-EU_0207/10	0207/10	resistor	1
R10	2k	TRIM_EU-PT10S	PT-10S	pot	1
R11	220k	R-EU_0207/10	0207/10	rcl	1
R12	220k	R-EU_0207/10	0207/10	rcl	1
R13	220k	R-EU_0207/10	0207/10	rcl	1
R14	100k	R-EU_0207/10	0207/10	rcl	1
R15	82k	R-EU_0207/10	0207/10	rcl	1
R16	100k	R-EU_0207/10	0207/10	rcl	1
R17	22k	R-EU_0207/10	0207/10	rcl	1
R19	10k	R-EU_0207/10	0207/10	rcl	1
R20	560k	R-EU_0207/10	0207/10	rcl	1
R21	10k	R-EU_0207/10	0207/10	rcl	1
R22	240	R-EU_0207/10	0207/10	rcl	1

Tabelle 2: Eagle-Bauteilliste

Part	Value	Device	Package	Library	Sheet
R25	1k	R-EU_0207/10	0207/10	rcl	1
R26	100k	TRIM_EU-RJ9W	RJ9W	pot	1
X1	22-23-2021	22-23-2021	22-23-2021	con-molex	1
X2	22-23-2041	22-23-2041	22-23-2041	con-molex	1
X3	22-23-2021	22-23-2021	22-23-2021	con-molex	1

Tabelle 3: Zusätzliche Bauteile

Part	Voltage	Power	Preis in Euro
Magnetventil für Wasser - normally off	12 V	0,48 W	7,43
Reed-Relais (Comus 3572-1220-051)	5 V	0,125 W	4,75
EC-Sensor für Adwa Messgeräte (AD33-36)			19,90

Jonas Mader

1.3 Team Microcontroller

1.3.1 Motivation Microcontroller ATMega32

Ziel der Teilgruppe Nährstoffe ist es ein, zum größten teil, unabhängiges System zu konstruieren. Deswegen haben wir uns entschieden einen eigenen Mikrocontroller in das System integriert. Dieser dient zur Kontrolle/Kommunikation der einzelnen Komponenten. Zudem müssen sowohl der Füllstand als auch die EC Messung und die Ph Messung getrennt voneinander stattfinden, da mehrere Spannungen im Wasser die Ergebnisse beeinflussen würden. Außerdem sollte die Interface-Gruppe diese Werte digital erhalten, sodass ein uC schon für die Datenverarbeitung nötig gewesen wäre. Gewählt haben wir den Microcontroller ATMega32 dieser gehört zur Microchip AVR 8-Bit-Microcontroller-Familie. Die Controller sind durch ihren einfachen Aufbau und ihrer leichten Programmierberkeit bekannt.

Phillip Rybski

1.3.2 Programmierung

Um den Microcontroller ATMega32 Programmieren zu können muss dieser erst mal mit Strom versorgt werden dazu wird der Controller mit einem Netzteil verbunden. Zur Pro-

grammierung benötigt man zusätzlich einen ISP-Programmer, ISP bedeutet In-System Programmer. Der ISP wird mit dem Controller und mit einer USB-Schnittstelle mit dem PC verbunden. Im weiteren Verlauf werden nun einzelne Programmteile beschrieben un erklärt und letztendlich das zusammenhängende Programm beschrieben.

Da das Herzstück des uC der Timer ist, haben wir angefangen Diesen zu programmieren. Die Clock (ebenfalls ein Begriff für den von uns programmierten Timer), wird über einen externen Taktgebenen Schwingquarz gesteuert , welcher an die Pins Xtal1 und Xtal angechlossen wird (Xtal ist dabei die englische Abküzung für Chrystal). Dieser Schwingquarz ist meist in eine Oszillator Schaltung eingebunden und erzeugt mit einem Schmitt-Trigger eine konstante Rechteckspannung mit einer im Mhz liegenden Frequenz. Diese Frequenz wird mithilfe eines Prescalers geteilt (durch 1024 in unserem Fall), damit einfacher gerechnet und weniger Taktfrequenzen nötig sind. Jedes im Prescaler liegende Rechtecksignal führt zur Inkrementierung eines Registers mit 8-Bit Größe. Wird nun der im Compare-Register(OCR0=0xFF) einegegebene Wert erreicht, wird ein Interrupt ausgelöst, welcher wiederum die Aufgabe erfüllen soll die Anzahl der Interrupts pro Sekunde in einen sich jede Sekunde verändernde Variable zu speichern. Wie viele Interrupts wir haben zeigt die folgende Rechnung:

$$\frac{(16 \cdot 10^6)}{1024 \cdot 256} = 61,0351$$

Demnach haben wir ca. 61 Interrupts pro Sekunde. Da wir nur ganze Zahlen benutzen können müssen wir diese Ungenauigkeit hinnehmen oder einen Uhrenquarz verwenden, welcher präzise mit 2^{15} Hz arbeitet und so genauer zur Berechnung einer Sekunde helfen kann. Für unsere Anwendung reicht allerdings diese Genauigkeit.

Hendrik Lottermoser

```
1  #ifndef F_CPU
2  #define F_CPU 16000000UL
3  #endif
4  #include <avr/io.h>
5  #include <avr/interrupt.h>
6  uint8_t count = 0x00 ;
7  uint16_t sek =0x00 ;
8  uint16_t min =0x00;
9  int main(void)
10 {
11 DDRA |= (1<< PA1);          //PD5 als output
12 TCCR0 |= (1<<CS02) | (1<<CS00); // durch 1024
13 OCR0 = 0xFF ;
```

```
14 TIMSK |= (1<<OCIE0);
15 sei();
16   while (1)
17   {
18 .....
19
20   }
21
22   ISR(TIMER0_COMP_vect)
23 {
24     if (count==0x3D)
25       {
26         sek ++ ;
27         if (sek == 61)
28           {
29             min++ ;
30             sek = 0 ;
31                           }
32       count=0x00;
33       }
34   count++ ;
35 }
```

Da wir zusätzlich beliebige Spannungen zwischen 0 und 5V ausgeben wollen wäre eine Pulsweitenmodulation nötig. Bei Pulsweitenmodulation spricht man von einem Rechtecksignal, welches in einer Periode eine beliebig lange Zeit(nicht länger als die Periode selbst) auf Low oder High sein kann. In unserem Aufbau bietet es sich an die Länge des High Signals anhand der Länge einer Periode in Prozent zu messen. Ziel unserer Pulsweitenmodulation (auch Pwm) ist es nämlich eine konstante Gleichspannung zu erzeugen mithilfe eines RC-Tiefpasses. Dieser RC-Tiefpass sollte den Mittelwert der PWM ausgeben, wenn er eingeschwungen ist. Diesen Mittelwert können wir beeinflussen, indem wir die Pulsbreite in einer Periode vergrößern oder verkürzen. Die Gleichspannung gibt nun einen Wert an der dem Soll-Wert für Ph- und EC-Wert entspricht und mit einem Komperator zusammengeschaltet eine Richtwert in den analogen Schaltungen gibt.
Um ein PWM-Signal zu implementieren müssen wir OCBX Pins als Output definieren (line 11). Hier shiften wir eine eins um PD4 nach links und verknüpfen DDRD mit einem OR, sodass alle Pins aktiviert bleiben, die aktiviert waren und PD4 als Output gesetzt wird, falls er es nicht war. Nun definieren wird die Register die zur PWM nötig sind. Mit COM1A1/COM1A0 (COM1A1/COM1A0 für den PD5 Pin) werden nun final die Pins aktiviert und bestimmt ob

sie invertierende PWM-Signale ausgeben sollen oder nicht. Am wichtigsten ist allerdings die Einstellung der Frequenz mithilfe des Prescalers und der Auflösung. Will man die Frequenz berechnen, muss man folgende Formel benutzen:

$$f_{PWM} = \frac{f_{clk}}{2 \cdot \textit{Auflösung} \cdot Prescaler} \tag{1}$$

$$f_{PWM} \approx 975Hz \tag{2}$$

Die Auflösung wird dabei durch die WGM-Bits eingestellt und der Prescaler durch die CS-Bits. Nun muss nur noch die Puls breite bestimmt werden mithilfe der Register ICR1 und OCR1A(/B). Dabei wird der angesteuerte Pin solange auf dem logischen High sein, bis es den Wert von OCR1X erreicht und schließlich auf Low bleiben bis ICR1. Unser Puls im Beispielcode hat also eine Breite von 25% der Periode. Dieser Wert wird in einer Funktion dem Sollwert der Interface-Gruppe angepasst (siehe Gesamter Code → *WaitAndMeasurePH* Funktion).

Hendrik Lottermoser

```
1
2
3
4  #include <avr/io.h>
5  #include <avr/interrupt.h>
6  #include <stdio.h>
7
8
9  int main(void)
10 {
11   DDRD |= (1 << PD4) | (1 << PD5) ;
12
13
14   TCCR1A = (1<<COM1A1) | (0<<COM1A0) | (1<<COM1B1) | (0<<COM1B0)
         | (1<<WGM11) | (0<<WGM10);
15   TCCR1B = (1<<WGM13) | (1<<WGM12) | (1<<CS10) | (0<<CS12) |
         (0<<CS11);
16
17
18   ICR1 = 20000;
19
20   OCR1A = 5000;
21   OCR1B = 5000;
```

```
22    while (1) {}
23
24 }
```

Da wir nun in der Lage waren unsere Programm zu Takten und Gleichspannungen auszugeben mussten wir nur noch einen ADC implementieren um vollständig alle Prozesse regeln und messen zu können. Der AD-Wandler sollte dabei in der Lage sein den Füllstand, den PH-Wert und den EC-Wert auszulesen. Die Gemessenen Spannung müssten dann nur noch interpretiert werden, sodass sie auf einem verständlich Wert übertragen werden können und schließlich an die Interface-Gruppe weitergegeben werden können. Es müssen demnach drei AD-Wandler implementiert werden. Diese haben in einem Atmega32 eine Auflösung von 10-Bit welche einen Spannungsbereich von 0 bis 5 V messen sollen. Daher sind die Stufen ca. 4,88 mV breit, was für unsere Anwendung mehr als ausreichend ist. Im beigefügten Code ist eine Standardmäßige Implementierung des AD-Wandler zu sehen, die noch auf unsere Anwendung zugeschnitten werden muss (siehe Quellcode des gesamten Programms).

Phillip Rybski

```
1
2
3
4  void ADC_Init(void)
5  {
6
7     ADMUX = (0<<REFS1) | (0<<REFS0);
8
9    ADCSRA = (0<<ADPS2) | (1<<ADPS0) | (1<<ADPS1) ;
10   ADCSRA |= (1<<ADEN);
11
12
13
14   ADCSRA |= (1<<ADSC);
15   while (ADCSRA & (1<<ADSC) ) {
16   }
17
18   (void) ADCW;
19 }
20
21
22
23
```

```
24
25 uint16_t ADC_Read( uint8_t channel )
26 {
27   // Kanal waehlen, ohne andere Bits zu beeinflu ß en
28   ADMUX = (ADMUX & ~(0x1F)) | (channel & 0x1F);
29   //ADMUX = (1<<REFS1) | (1<<REFS0) | (0<<MUX4) | (0<<MUX3) |
       (0<<MUX2)| (1<<MUX1) | (1<<MUX0) | (0<<ADLAR);
30   ADCSRA |= (1<<ADSC);              // eine Wandlung "single
       conversion"
31   while (ADCSRA & (1<<ADSC) ) {     // auf Abschluss der
       Konvertierung warten
32   }
33   return ADCW;                      // ADC auslesen und zurückgeben
34 }
```

1.3.3 Schaltplan

In der folgenden Abbildung 9 kann man den Schaltplan für den Microcontroller sehen den wir entworfen haben. Zu sehen ist auf der rechten Seite der 32-Bit Datenbus. Von ihm bekommen wir die Versorgungsspannung für unseren Microcontroller so wie die Masse GND. Zusätzlich bekommt der Microcontroller noch 4 weitere Leitungen die für die SPI-Kommunikation notwendig sind. Um bei Komplikationen das System neu herzustellen haben wir einen Reset Knopf mit eingebaut. Damit der Microcontroller mit unseren Zeit-Informationen genauer Arbeiten kann haben wir uns entschieden einen Externen Uhrenquarz ein zu bauen. Dieser braucht zum funktionieren noch zwei weitere Kondensatoren. Da wir mit PWM/Rechteck-Signalen arbeiten die geglättet werden müssen, um eine bestimmte Spannung zu bekommen, fügten wir an den Ausgängen jeweils Kondensatoren mit einem Widerstand hinzu. Zur besseren Übersicht und genaueren/schnelleren Arbeit haben wir die Datenübertragung unterteilt in 3 x 4 Leitungen, sodass die einzelnen Untergruppen jeweils ein Bus bekommen mit den Leitungen die sie brauchen für ihre Schaltungen.

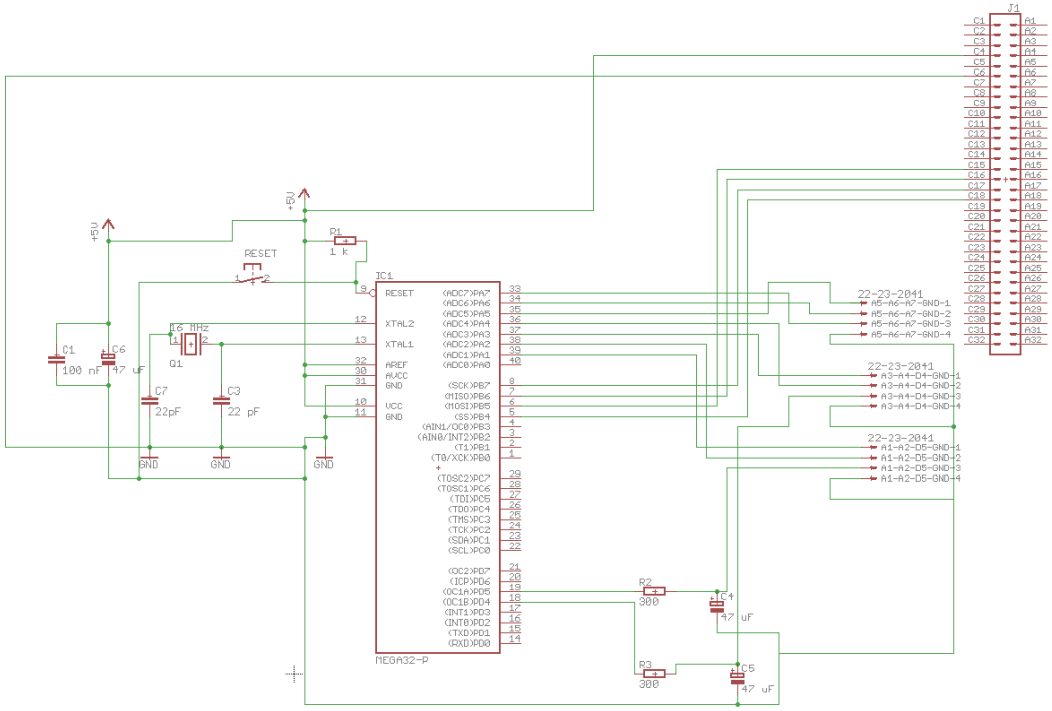


Abbildung 9: Schaltplan vom Microcontroller

Philipp Rybski

1.3.4 Simulation

Wir als Microcontroller-Gruppe haben keine direkte Simulation mit hilfe von LTspice durchgeführt. Die Hauptaufgabe in unserer Gruppe war den Microcontroller für die Anwendungen richtig zu Programmieren. Wir Programmierten den Microcontroller auf einer fertigen Platine, so konnten wir untersuchen ob die Programmierung die erwartenden Werte an den Leitern ausgibt.

Philipp Rybski

1.3.5 Platinen Layout

In der Abbildung 10 sieht man das aus dem Schaltplan 9 erzeugte Platinen Layout für unseren Microcontroller Platine. Um das anlöten zu vereinfachen haben wir versucht alle möglichen Leitungen auf der Unterseite vom Board zu platzieren, sodass es einfach wird die Bauteile

zu montieren. Beim Uhrenquarz war es wichtig diesen so nah wie möglich am Microcontrolle zu plazieren da mit jedem weiteren milimeter an Leitung das Quarz ungenauer wird. Rechts von unserer PLatine liegt der 32 Bit Datenleiter und Links die 3 x 4 Datenleiter, für die Untergruppen.

Philipp Rybski

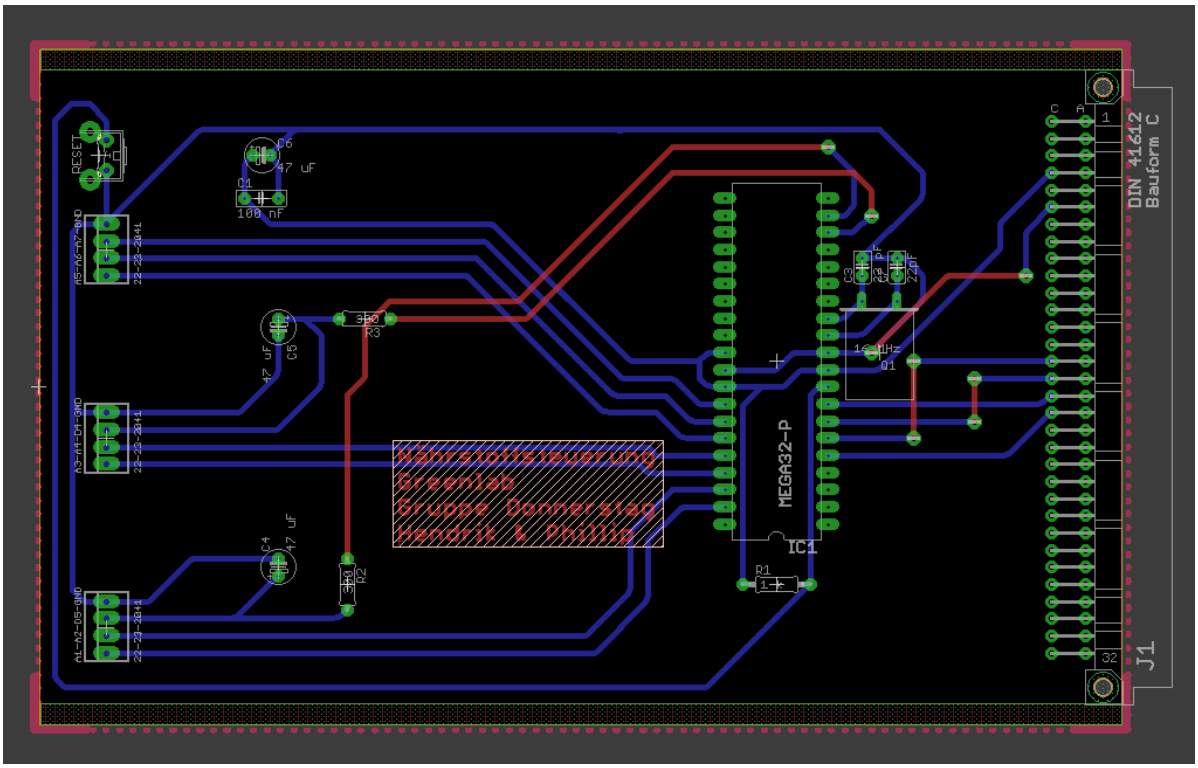


Abbildung 10: Platinen Layout vom Microcontroller

1.3.6 Vollständiger Quellcode

```
1 #ifndef F_CPU
2 #define F_CPU 16000000UL
3 #endif
4
5
6 #include <avr/io.h>
7 #include <util/delay.h>
8 #include <avr/interrupt.h>
9 #include <stdio.h>
```

```
10
11 int muell = 0 ;
12 uint16_t tmess = 5;     // Wie lange gemessen werden soll
13 uint16_t tpump = 2 ;  // Zeit in Sekunden wie lange gepumpt wird
14
15 uint16_t pump  = 2 ;  // Diese Variable gibt an in welchem Takt
       die Pumpe Pumpen soll in Sekunden (Variable da Main MC die
       Variable bestimmt)
16 uint8_t ph_soll  = 70; //PH Variable die der Main Controller
       bestimmt
17 uint8_t ec_soll = 25; //EC Varibale die der Main Controller
       bestimmt
18
19 uint16_t ph_rueckgabe = 0;   //Umwandlung PH_mess in einen Int
       von 0-140
20 uint16_t ec_rueckgabe = 0; //Umwandlung EC_mess in einen Int von
       0-50
21 uint16_t fuell_rueckgabe = 0; //Umwandlung fuell_mess
22
23
24
25 uint8_t count  = 0x00 ;  // Counter fuer TIMER
26 uint16_t sek   = 0x00 ;  // Variable fuer Sekunde
27
28 uint16_t xsek  = 0x00 ; // Variable zum Speichern einer
       beliebigen Zeit
29
30 uint16_t ph_pass  = 0; //Hochskalierte PH Wert
31 uint16_t ph_mess = 0;  //Gemessener PH Wert
32
33
34 uint16_t ec_pass = 0; //Hochskalierter EC Wert
35 uint16_t ec_mess = 0; //Gemessener EC Wert
36
37
38 uint16_t fuell_mess = 0; //Varibale fuer den Fuellstand
39
40
41 void SPI_SlaveInit(void)
42 {
43   /* Set MISO output, all others input */
44   DDRB = (1<<DDB6);
```

```
45    /* Enable SPI */
46    SPCR = (1<<SPE);
47  }
48
49
50  int SPI_tranceiver (int data)
51  {
52    // Werte ins Regiser schreiben
53    SPDR = data;
54    //Wait until transmission complete
55    while(!(SPSR & (1<<SPIF)));   // Return received data
56    //data = SPDR; // dummy byte empfangen
57    return(SPDR);
58
59  }
60
61
62  ISR(TIMER0_COMP_vect) //TIMER zaehlt im Hintergrund in Sekunden
      bis 24h
63  {
64    if (count==0x3D)
65    {
66    sek ++ ;
67      if (sek >= 21600) // Verhinderung das beim zurücksetzen die
          WHileschleife nicht aufgerufen ist.
68        {
69        sek = 0;
70        }
71    count=0x00;
72
73      }
74    count++ ;
75  }
76
77
78  void ADC_Init(void)
79  {
80    // die Versorgungsspannung AVcc als Referenz wählen:
81    //ADMUX = (1<<REFS0);
82    // oder interne Referenzspannung als Referenz für den ADC
        wählen:
83     ADMUX = (0<<REFS1) | (0<<REFS0);
```

```
84
85
86    // Bit ADFR ("free running") in ADCSRA steht beim Einschalten
87    // schon auf 0, also single conversion
88    ADCSRA = (0<<ADPS2) | (1<<ADPS0) | (1<<ADPS1) ;     //
          Frequenzvorteiler
89    ADCSRA |= (1<<ADEN);                    // ADC aktivieren
90
91    /* nach Aktivieren des ADC wird ein "Dummy-Readout" empfohlen,
          man liest
92       also einen Wert und verwirft diesen, um den ADC "warmlaufen␣
             zu␣lassen" */
93
94    ADCSRA |= (1<<ADSC);                    // eine ADC-Wandlung
95    while (ADCSRA & (1<<ADSC) ) {           // auf Abschluss der
          Konvertierung warten
96    }
97    /* ADCW muss einmal gelesen werden, sonst wird Ergebnis der
          nächsten
98       Wandlung nicht übernommen. */
99
100   (void) ADCW;
101 }
102
103 uint16_t ADC_Read( uint8_t channel )
104 {
105 // Kanal waehlen, ohne andere Bits zu beeinflu ß en
106   ADMUX = (ADMUX & ~(0x1F)) | (channel & 0x1F);
107
108   ADCSRA |= (1<<ADSC);              // eine Wandlung "single
          conversion"
109   while (ADCSRA & (1<<ADSC) ) {     // auf Abschluss der
          Konvertierung warten
110   }
111 return ADCW;                      // ADC auslesen und zurückgeben
112 }
113
114 uint16_t umwandlung_ph( uint16_t a)
115 {
116   float b ;
117   b = (float) a * 0.1367 ;
118   a =(uint16_t) b ;
```

```
119 return a;
120 }
121
122 uint16_t umwandlung_ec( uint16_t a)
123 {
124   float b ;
125   b = (float) a * 0.048828 ; //Anpassung von der gemessenen
        Spannung(0-1024/0-5V) auf 0-50 Int
126   a =(uint16_t) b ;
127 return a;
128 }
129
130 uint16_t umwandlung_fuell( uint16_t a)
131 {
132   float b ;
133   b = (float) a * 0.09765625 ; //Anpassung von der gemessenen
        Spannung(0-1024/0-5V) auf 0-100 Int
134   a =(uint16_t) b ;
135 return a;
136 }
137
138 void WaitAndMeasurePH (ph_soll){
139
140   ph_pass = ph_soll *143 ; //soll wert für ph Prozentual
        anpassen (auf den Bereich von 0-20000)
141   OCR1A = ph_pass ; // PWM auch prozentual anpassen
142   PORTA |= (1<<PA1); // Spannung damit PH-schaltung mit
        regulierung und MEssung beginenn kann
143   xsek = sek + tmess ;
144   while (sek < xsek){
145     ph_mess = ADC_Read(4); //ph Wert wird gemessen
146   }
147   PORTA &= ~(1<<PA1); // Spannung für ph-Schaltung aus
148   OCR1A = 0;   //PWM wird auf 0 gesetzt
149 }
150
151 void WaitAndMeasureEC (ec_soll){
152
153   ec_pass = ec_soll *400 ; //soll wert für ec Prozentual
        anpassen (auf den Bereich von 0-20000)
154   OCR1B = ec_pass ; // PWM auch prozentual anpassen
```

```
155   PORTA |= (1<<PA2); // Spannung damit EC-schaltung mit
          regulierung und MEssung beginenn kann
156   xsek = sek + tmess ;
157   while (sek < xsek)
158   {
159      ec_mess = ADC_Read(5); //ec Wert wird gemessen
160   }
161   PORTA &= ~(1<<PA2); // Spannung für ec-Schaltung aus
162   OCR1B = 0;    //PWM wird auf 0 gesetzt
163 }
164
165 void pumpen (){
166
167    PORTA |= (1<<PA3) | (1<<PA7); //Pumpe Pumpt für t_pump
          Sekunden
168    xsek = sek + tpump ;
169    while(sek < xsek)
170    {
171       fuell_mess = ADC_Read(6); //Abmessung vom Fuellstand
172    }
173   PORTA &= ~(1<<PA7);
174   PORTA &= ~(1<<PA3);
175 }
176
177 int main(void)
178 {
179 ADC_Init();
180   DDRD |= (1 << PD4) | (1 << PD5) ;
181   DDRA |= (1 << PA1) | (1 << PA2) | (1<< PA3) | (1<< PA0);
182
183
184   TCCR1A = (1<<COM1A1) | (0<<COM1A0) | (1<<COM1B1) | (0<<COM1B0)
          | (1<<WGM11) | (0<<WGM10);
185   TCCR1B = (1<<WGM13) | (1<<WGM12) | (1<<CS10) | (0<<CS12) |
          (0<<CS11);
186
187
188   ICR1 = 20000;
189
190
191 TCCR0 |= (1<<CS02) | (1<<CS00); // durch 1024
192
```

```
193 OCR0 = 0xFF ;
194 TIMSK |= (1<<OCIE0);
195 sei();
196 SPI_SlaveInit();
197
198
199
200
201 while (1)
202 {
203 if ((1<<PB4) == 1)
204 {
205   ec_soll = SPI_tranceiver(ec_rueckgabe);
206   ph_soll = SPI_tranceiver(ph_rueckgabe);
207   muell = SPI_tranceiver(fuell_rueckgabe);
208 }
209
210
211 if (sek>21000)
212 {
213   sek = 0;
214 }
215  while((sek%pump) == 0) //pump-2*tmess da er erst pumpen soll
       nachdem gemessen wurde es wird 2 mal gemessen
216  {
217
218   WaitAndMeasurePH(ph_soll);
219   ph_rueckgabe = umwandlung_ph(ph_mess);
220
221   WaitAndMeasureEC(ec_soll);
222   ec_rueckgabe = umwandlung_ec(ec_mess);
223
224   pumpen();
225   fuell_rueckgabe = umwandlung_fuell(fuell_mess);
226  }
227 }
228
229 }
```

[H]

Literatur

[Timer] https://www.mikrocontroller.net/articles/AVR-GCC-Tutorial/Die_Timer_und_Z%C3%A4hler_des_AVR

[AD-Wandler und PWM] https://www.mikrocontroller.net/articles/AVR-GCC-Tutorial/Analoge_Ein-_und_Ausgabe

[Allgemeine Code Beispiele] https://www.mikrocontroller.net/articles/AVR-GCC-Tutorial

1.4 Team pH-Sensor

Im Folgenden werden die Zwischenergebnisse und der aktuelle Stand vom Team pH-Sensor näher erläutert. Dafür haben wir zunächst ein allgemeines Flussdiagramm zur Regelung der pH Konzentration erarbeitet (Abbildung 11) und anschließend ein erstes Konzept zur Realisierung in LT Spice erstellt.

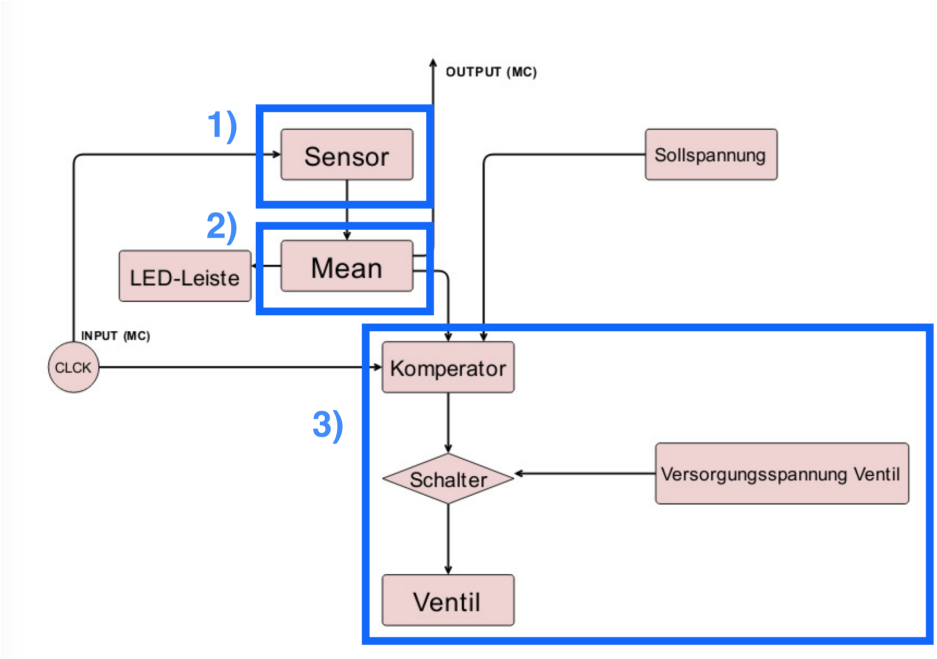


Abbildung 11: allgemeines Blockschaltbild

Zur besseren Übersicht und strukturellen Vorgehensweise haben wir das Blockschaltbild in insgesamt drei Bereiche unterteilt. Der erste Bereich beschäftigt sich mit dem "Sensor", der zweite Bereich mit dem "Mean" und der dritte Bereich umfasst die Ventilsteuerung.

Yannick Klose

1.4.1 Blockschaltbild

1.4.2 Teil 1: Sensor

Da der Sensor ein sehr komplexes Bauteil ist und dieser schwer selber zu realisieren ist, kaufen wir einen standard pH Sensor. Der Sensor liefert eine dabei eine sehr geringe Spannung, die wir vorerst verstärken müssen. Anschließend können wir durch eine Umrechnungstabelle, ähnlich

wie in Abbildung 12) zu sehen, den unterschiedlichen Spannungsstufen den entsprechenden pH-Wert zuordnen. Die Umrechnung sowie die Angabe eines Sollwertes wird vom Team Mikrocontroller übernommen.

Tim Aaron Schaub

VOLTAGE (mV)	pH value	VOLTAGE (mV)	pH value
414.12	0.00	-414.12	14.00
354.96	1.00	-354.96	13.00
295.80	2.00	-295.80	12.00
236.64	3.00	-236.64	11.00
177.48	4.00	-177.48	10.00
118.32	5.00	-118.32	9.00
59.16	6.00	-59.16	8.00
0.00	7.00	0.00	7.00

Abbildung 12: Vergleichstabelle - [Vergleichstabelle]

1.4.3 Teil 2: Mean

Hier bilden wir einen Mittelwert aus den aufgenommenen Messungen, um somit den Fehler zu minimieren. Aktuell überlegen wir uns wie dies realisierbar ist. Erste Überlegungen ergaben, dass sich ein Tiefpassfilter durchaus zum Glätten der Messwerte eignen könnte.

Tim Aaron Schaub

1.4.4 Teil 3: Ventilsteuerung

Ziel ist es durch einen Komperator den Ist-Wert mit dem Soll-Wert zu vergleichen. Wenn dabei eine festgelegte Schranke überschritten wird, soll dieser ein Ausgangssignal von 5V o.ä. liefern und automatisch eine Art "Trigger" gesetzt werden, welcher einen gewissen Bereich festlegt (Schmitt-Trigger). Dieses Ausgangssignal soll anschließend genutzt werden, um ein Magnetventil zu öffnen. (Quelle: [Ventilsteuerung])

Yannick Klose

1.4.5 Simulation

Nach ersten groben Überlegungen, haben wir eine Schaltung entworfen, die zunächst ein Signal verstärkt und einen Offset (14) erzeugt, sodass unser Ursprüngliches Signal des pH-

Sensors auf einen Bereich von 0V bis 5V abbildet. Diese Abbildung verläuft linear, wobei 0V einem pH-Wert von 0 entspricht und 5V einem pH-Wert von 14. Dieses Signal kann anschließend Mithilfe eines Komparators (15) , welcher in Form eines SchmittTriggers realisiert wurde, mit einem von der Microcontrollergruppe festgelegten Sollwert vergleicht. Wenn dabei eine bestimmte Schranke überschritten wird, löst dieser aus und der Operationsverstärker liefert ein Ausgangssignal von 12V. In Abbildung 13 ist eine Simulation von LT Spice gezeigt. Diese ist aufgeteilt in zwei Verstärkungsstufen (nichtinvertierten und invertierten), einer Offset-Schaltung und einem Komparator.

Tim Aaron Schaub

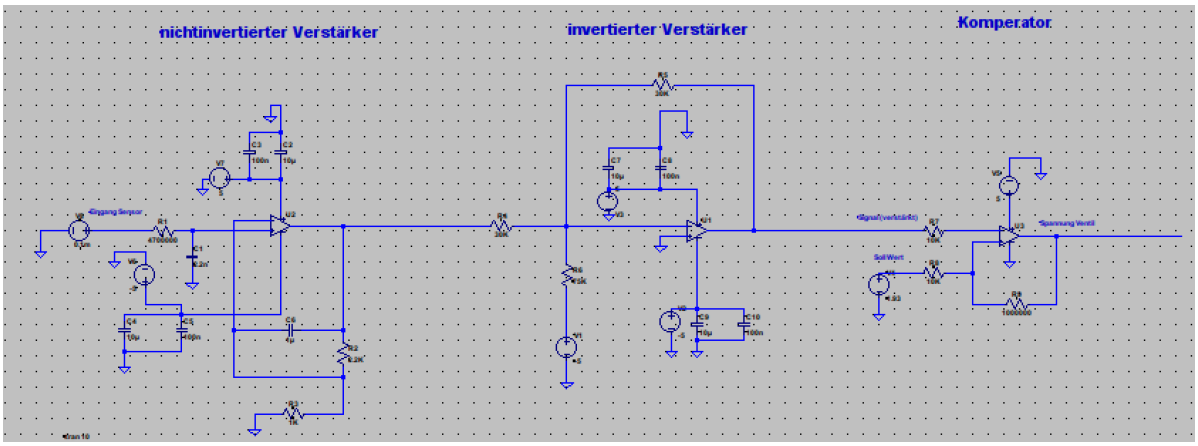


Abbildung 13: Simulation in LTspice

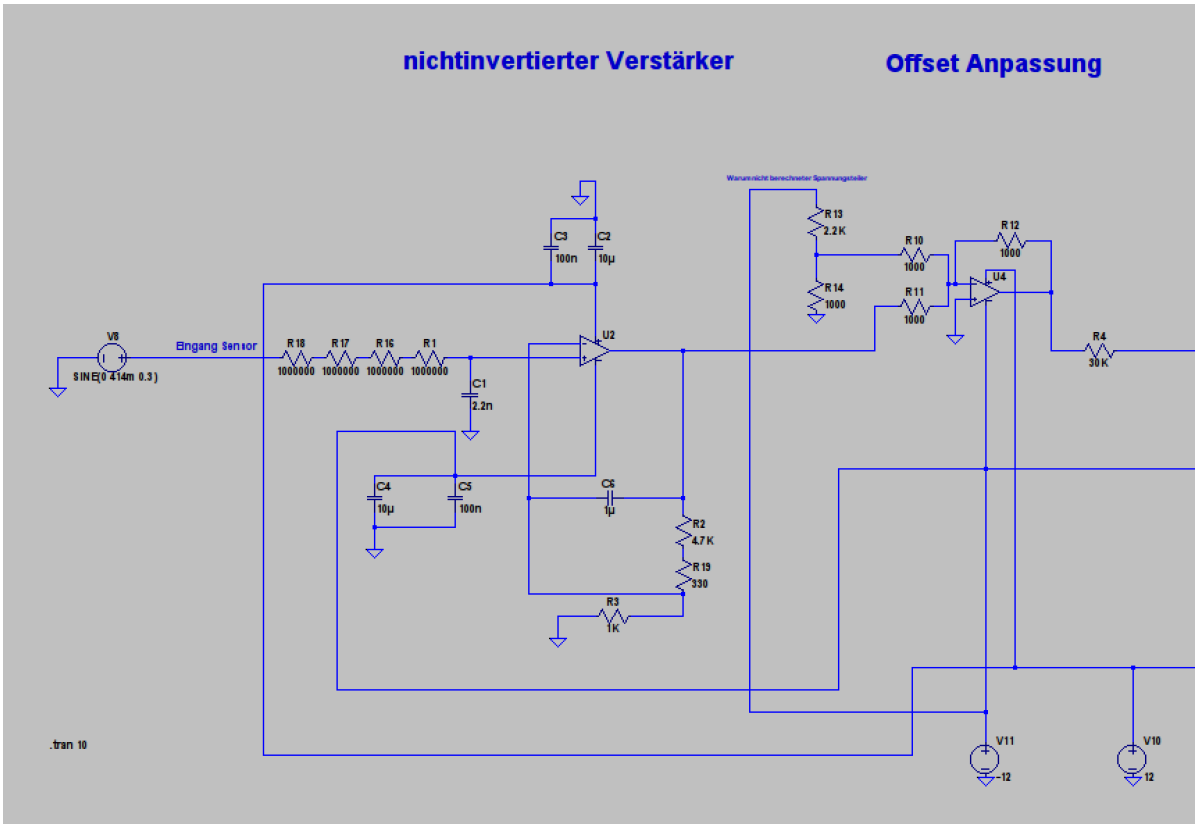


Abbildung 14: Vergrößerte Schaltung Teil1

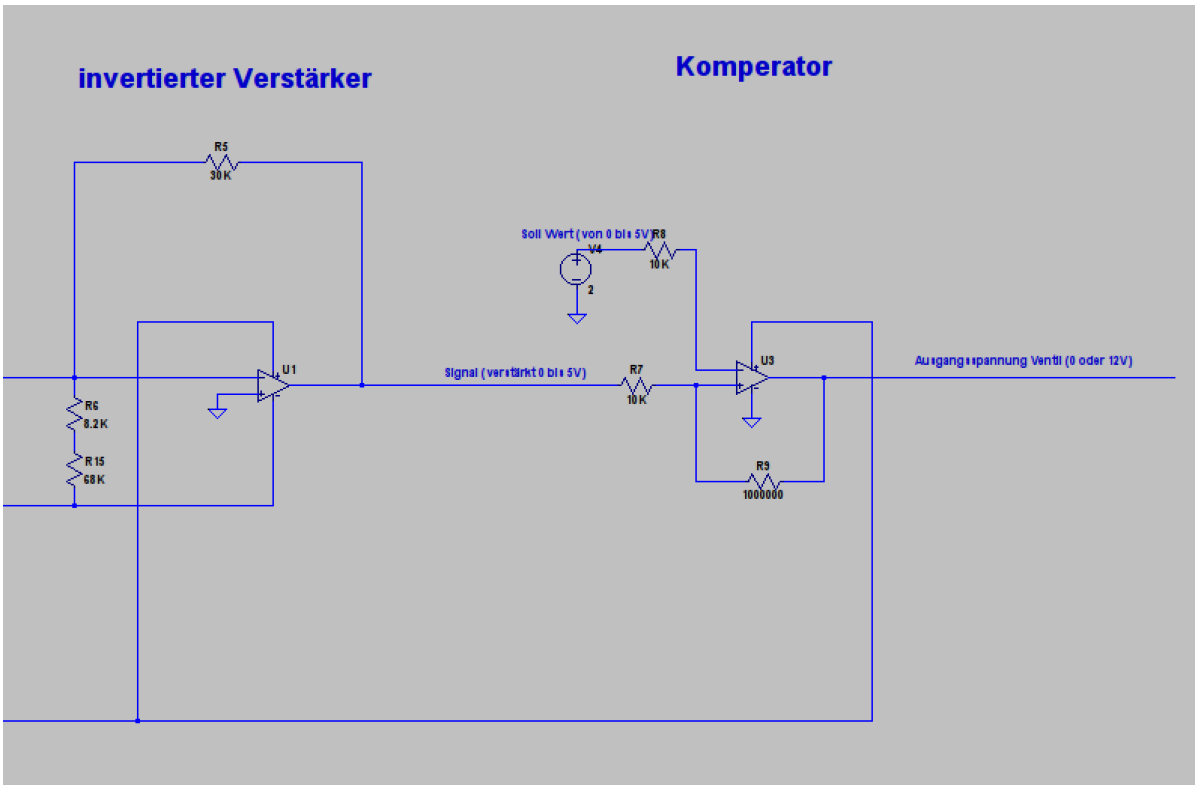


Abbildung 15: Vergrößerte Schaltung Teil2

1.4.6 Technische Umsetzung

Nachdem die Bauteile in unserer Schaltung dimensioniert und auf dem Steckbrett aufgebaut sind, haben wir zudem einen LED-Anzeige mit Transistoren aufgebaut. Eine gelbe LED leuchtet, falls der pH-Wert den Sollwert überschreitet, ansonsten leuchtet eine Gelbe LED. Außerdem haben wir für das Magnetventil eine kleine Schaltung entworfen, welche ebenfalls ansteuert falls der pH-Wert zu hoch ist. Dabei war zu beachten, dass die Spannung aus dem OPV nur als Steuerung genutzt wird und nicht als Spannungsquelle, da der Strom den der OPV liefern kann zu gering ist und die Spannung zusammen brechen würde. Des Weiteren haben wir eine LED an dem Ventil angebracht um zu zeigen wenn diese durschaltet. Auf der fertigen Platine (6) haben wir zudem ein Relais angeschlossen welches durch ein Signal "Messfreigabe"die Betriebsspannung zur Platine durchschaltet. Diese Signal soll uns von der Microcontrollergruppe bereitgestellt werden.

Yannick Klose

[H]

Literatur

[Vergleichstabelle] https://www.dfrobot.com/wiki/index.php/PH_meter(SKU:_SEN0161)

[Ventilsteuerung] https://rn-wissen.de/wiki/index.php?title=CNY70

[SensorKaufLink] https://www.amazon.de/PH-Elektrode-Sonde-PH-Messger\protect\unhbox\voidb@x\bgroup\U@D1ex{\setbox\z@\hbox{\char127}\dimen@-.45ex\advance\dimen@\ht\z@\fontdimen5\font\dimen@}\accent127\fontdimen5\font\U@Da\egroupt-Kalibrierfl\protect\unhbox\voidb@x\bgroup\U@D1ex{\setbox\z@\hbox{\char127}\dimen@-.45ex\advance\dimen@\ht\z@\fontdimen5\font\dimen@}\accent127\fontdimen5\font\U@Du\egroupssigkeitrezepte-wartungsfrei/dp/B00BLHGCCQ

1.5 Team Füllstand

Das Füllstands-Team sollte ursprünglich, wie der Name schon sagt, die Füllstandmessung des Haupt-Wasserbeckens, den Mixer, der für die ausreichende Vermengung des Nähstoff-Wasseranteils sorgen soll, sowie das Pumpsystem mit dazugehöriger Regulierung.

Im Verlauf des Semesters änderten wir unsere Planung, aufgrund der Gegebenheiten, bezüglich des Streiks und der Ausfälle, sodass wir den Teil mit der Pumpe an die Gehäusegruppe übergaben und den Mixer ausließen.

Konzentrieren wir uns deshalb auf die Füllstandsmessung und betrachten zunächst das folgende, dafür vorgesehene, Blockschaltbild:

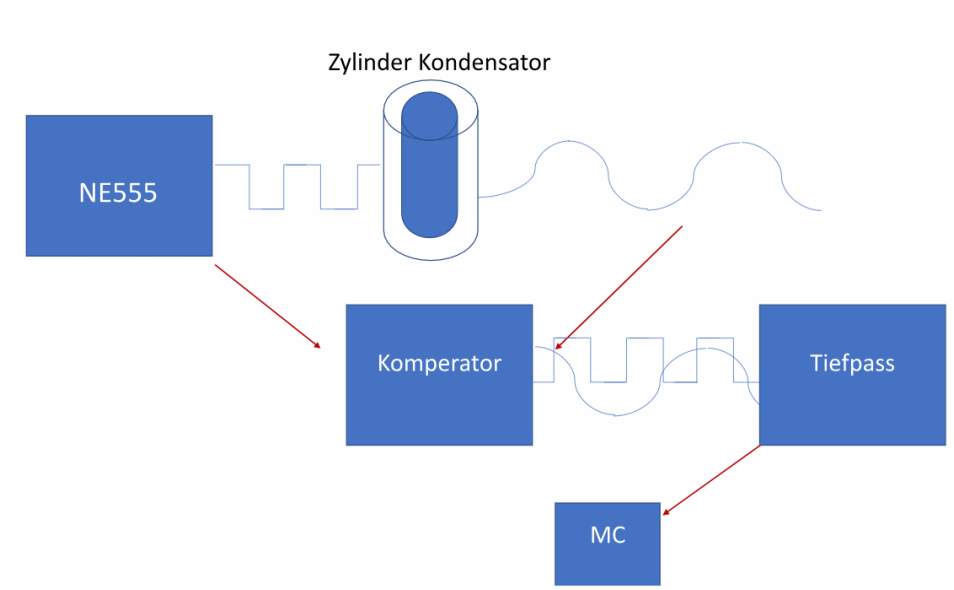


Abbildung 16: Blockschaltbild zur Füllstandsmessung

Mohamed Ali

Frederic Eidenmüller] Mit dem Timer-Bauteil NE555 erzeugen wir eine Rechteckspannung, welche den Zylinderkondensator lädt, welcher im Wasserbehälter eingetaucht werden soll. Die ausgehende Spannung, entspricht einer Ladekurve des Kondensators (Blauer Graph im Blockschaltbild). Diese wird mit einer Gleichspannung, mit dem Komperator verglichen und anschließend ein Mittelwert dieses Ausgangs mit einem RC-Tiefpass ausgegeben. Somit erhalten wir je nachdem wie sich die Kapazität des Kondensators verändert, einen entsprechenden Gleichspannungswert.

[

1.5.1 Sensor

Der Sensor, also der Zylinderkondensator besteht aus einem Eisenrohr von 25mm Durchmesser, sowie einer darin zentral fixierten, mit Schrumpfschlauch isolierten 6mm Schraube.

Abbildung 17: Blockschaltbild zur Füllstandsmessung

Um zu testen, ob wir wirklich aussagekräftige Veränderungen der Kapazität bekommen, wenn wir den Füllstand verändern, haben wir die Konstruktion mit Wasser aufgefüllt und die Kapazität mittels eines Multimeters gemessen. Dabei erhielten wir folgende Wertetabelle:

Mohamed Ali

Füllhöhe in c	Kapazität in nF
0	0,066
5	0,114
10	0,204
15	0,293
20	0,326
25	0,341
30	0,412

Abbildung 18: Blockschaltbild zur Füllstandsmessung

1.5.2 Schaltung

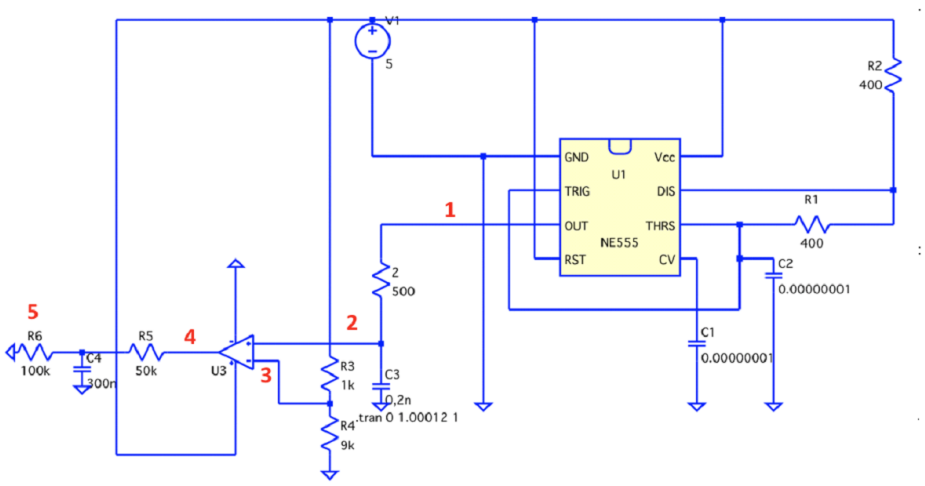


Abbildung 19: Vollständige Schaltung

1. Output des NE555 generiert eine Rechteckspannung

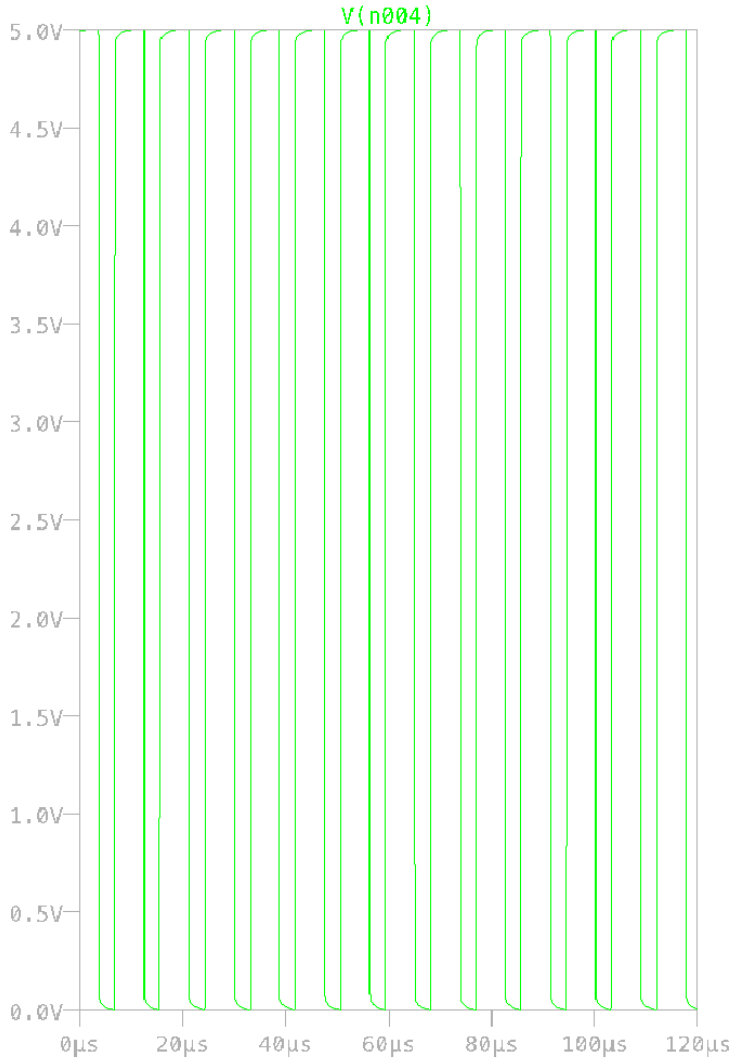
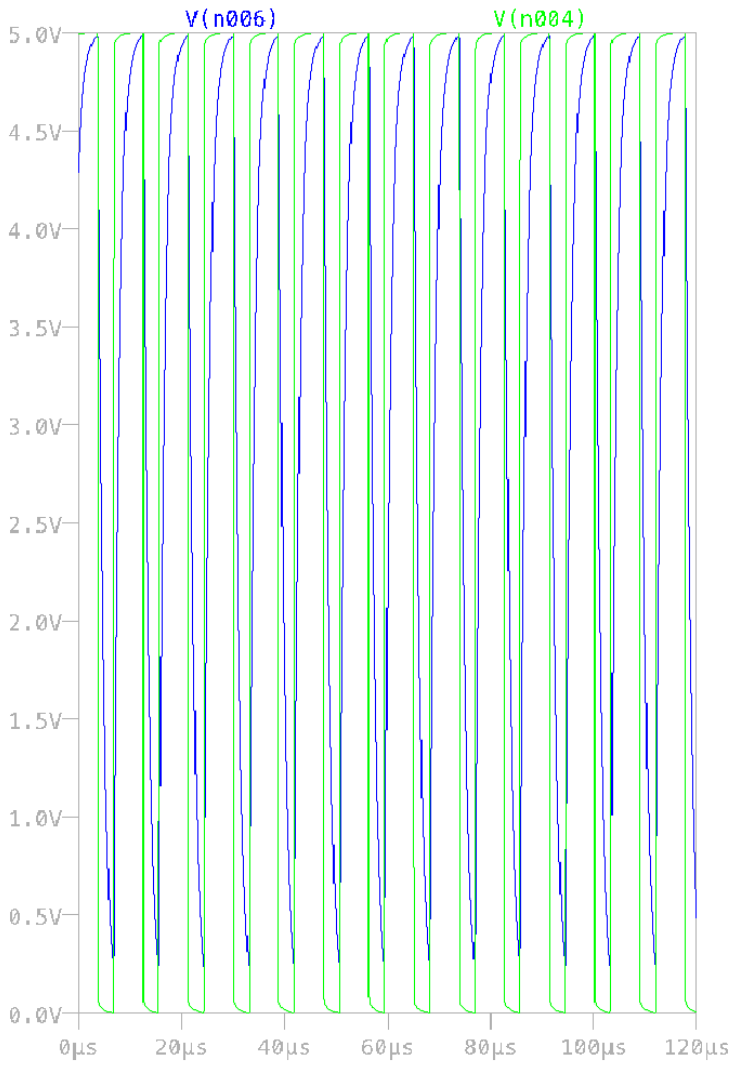
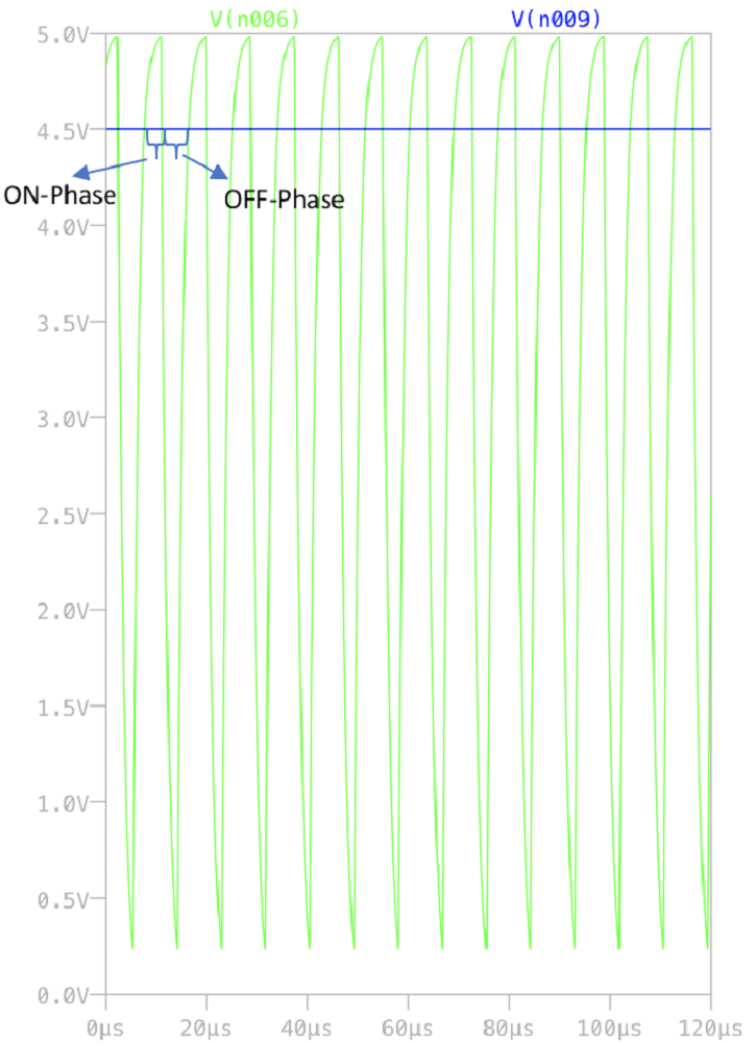


Abbildung 20: Output des NE555 generiert eine Rechteckspannung

2.Die Spannung wird durch den Kondensator C3 (unser Sensor/Zylinderkondensator) geglättet (Blaue Kennlinie).



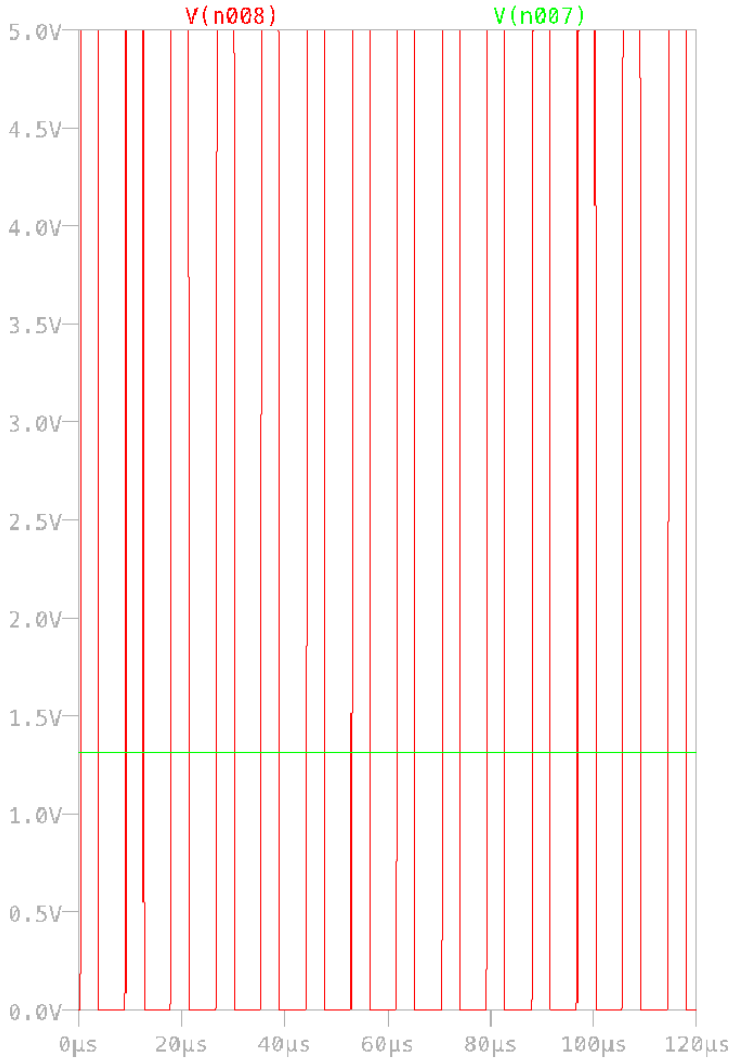
3.Es wird in den Komparator eine Vergleichs-Gleichsspannung von 4,5 Volt, mithilfe des Spannungsteilers von R3 und R4 angelegt



4. Im obigen Bild sieht man, dass durch die Gleichspannung eine Pulsweitenmodulation entstehen würde, wenn man dieses Signal durch einen Komparator leiten würde. So bekommen wir mit der Formel : (ON-Phase Zeit/OFF-Phase Zeit)*(5V) und einer Glättung mit dem RC-Tiefpass bei 5. eine Gleichspannung von 0,6V-1,7V, die wir direkt der Microcontroller Gruppe weitergeben können.

In diesem Fall als Beispiel mit einer C3 Kapazität von 0,2nF. In Rot das Signal nach dem Komparator und in grün das Ausgangssignal nach dem TP.

Frederic Eidenmüller



Die Interpretation dieser Signale erfolgt durch den MC mit folgender Tabelle:

Kapazität vom C3 in nF	Ausgangsspannung in V
0,1	1,7
0,2	1,3
0,3	0,9
0,4	0,6

1.5.3 Realisierung durch Eagle und Dimensionierung

Frederic Eidenmüller] Beim Eaglen gab es viele Probleme, die uns begegneten, da wir zum ersten Mal mit diesem Programm gearbeitet haben. Die Platine wurde fehlerhaft geätzt, da wir falsche Deklarationen im Layout hatte, was ein gleichmäßiges Potetial des Grounds angeht. Es kamen diverse weitere Schaltungsbereiche dazu. Beispielsweise haben wir diverse Kondensatoren hinzugefügt, die als Sicherheit gegen Spannungsschwankungen eingesetzt wurden. Außerdem wurden die oben erwähnten Spannungsteiler durch Potentiometer ersetzt, um eine gewisse Flexibilität beizubehalten. Die Dimensionierung unserer Schaltung fing mit dem Versuch an, zu erkennen, um was für Kapazitäten sich es bei unserem Zylinderkondensator handelt. Dadurch konnten wir C3 eingrenzen. Ansonsten haben wir viel herumprobiert um dann ein passables Ergebnis zu bekommen. Wir hätten allerdings das gesamte Potenzial des ADU's von 5 Volt ausschöpfen müssen, die war aber aufgrund der bereits geätzten Platine nicht mehr möglich. Dafür wäre nämlich eine Verstärkung mithilfe eines OPV's nach dem Tiefpass nötig gewesen. Damit hätten wir eine ordentlichere Auflösung bekommen.

[