# Projekt: Smartes Gewächshaus
# SBIO (Smart BIO)


## Projekt Entwickler:
Paul Kaupp
Leon Kempter
Maximilian Hoffmann
David Dreyer

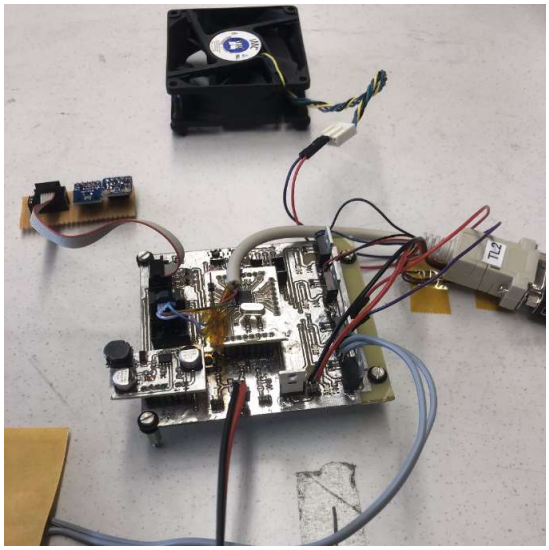
## Firma:
BMK-Gro
Werner-von-Siemens-Str. 6
86159 Augsburg

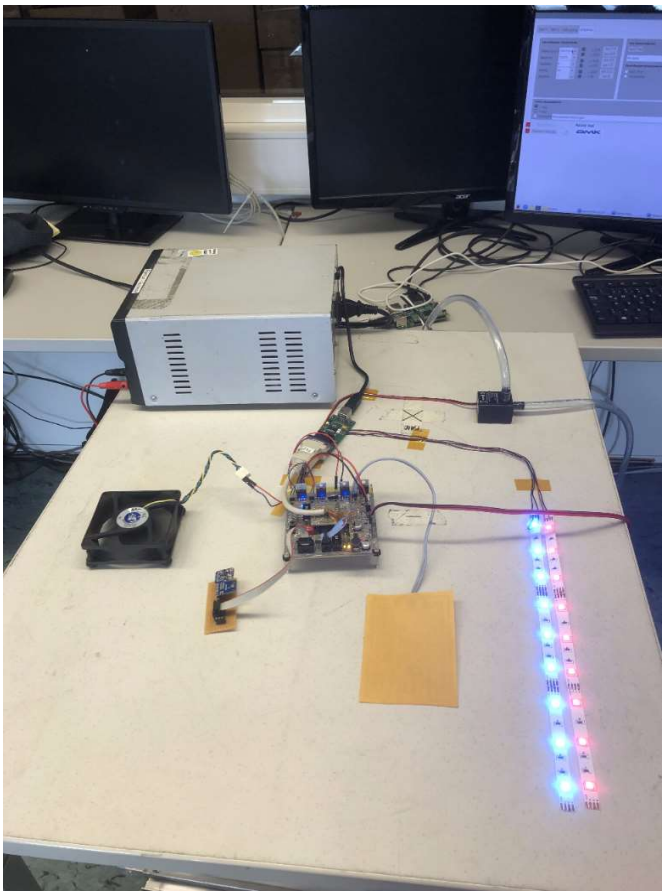

## Inhalte


1. Idee
2. Projektdurchführung
3. Beschreibung
    1. Allgemeine Funktion
    2. Elektronik
        1. Fertigung der Platine
        2. Schaltplan Steuerplatine
        3. Schaltplan Spannungsversorgung
        4. Schaltplan Dokumentation
        5. Layout
    3. Software (Steuerplatine)
    4. Software (Raspberry)
    5. Mechanischer Aufbau
4. Erfahrungen und Fehler
5. Vorstellung des Team‘s

## Idee

# Projektdurchführung



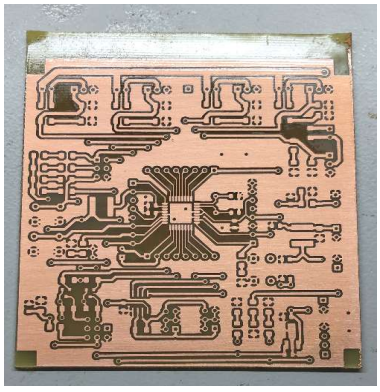


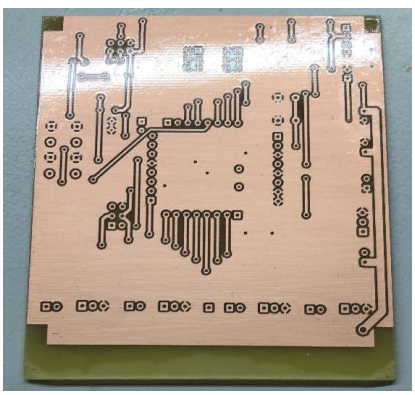
## Fertigung der Platine

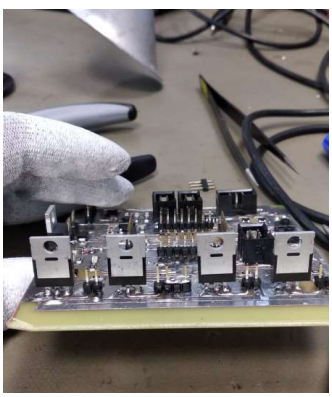
Die Steuereinheit wurde mit einer selbst geätzten Platine realisiert. Diese wurde privat zusammen mit Teamkollegen gefertigt.
Diesen Vorgang wollen wir ganz kurz beschreiben:

1. Anfertigen von Filmen unserer Leiterplatte. Hierbei wurden für jede Seite drei bedruckte Overheadfolien übereinander gelegt, um eine Verringerung der Lichtdurchlässigkeit der schwarzen Flächen zu erreichen.
2. Diese Folien wurden auf der Fotoplatine angebracht und 2:30 Min mit UV-Strahlung beidseitig belichtet.
3. Die nun belichtete Leiterplatte wurde anschließend in einen Glasbehälter mit Fotoentwickler gelegt.
4. Durch das Einlegen in das Ätzbad wurden alle Bereiche der Leiterplatte, welche beim Belichten nicht durch den schwarzen Teil der Folie geschützt waren und somit beim Entwickeln freigelegt wurden, weggeätzt.
5. Nach ca. 10 min im Ätzbad (Natriumpersulfad) wurde die Platine gründlich mit Alkohol gereinigt
6. Zuletzt folgte das Verzinnen aller Kupferflächen um der Oxidation entgegenzuwirken.

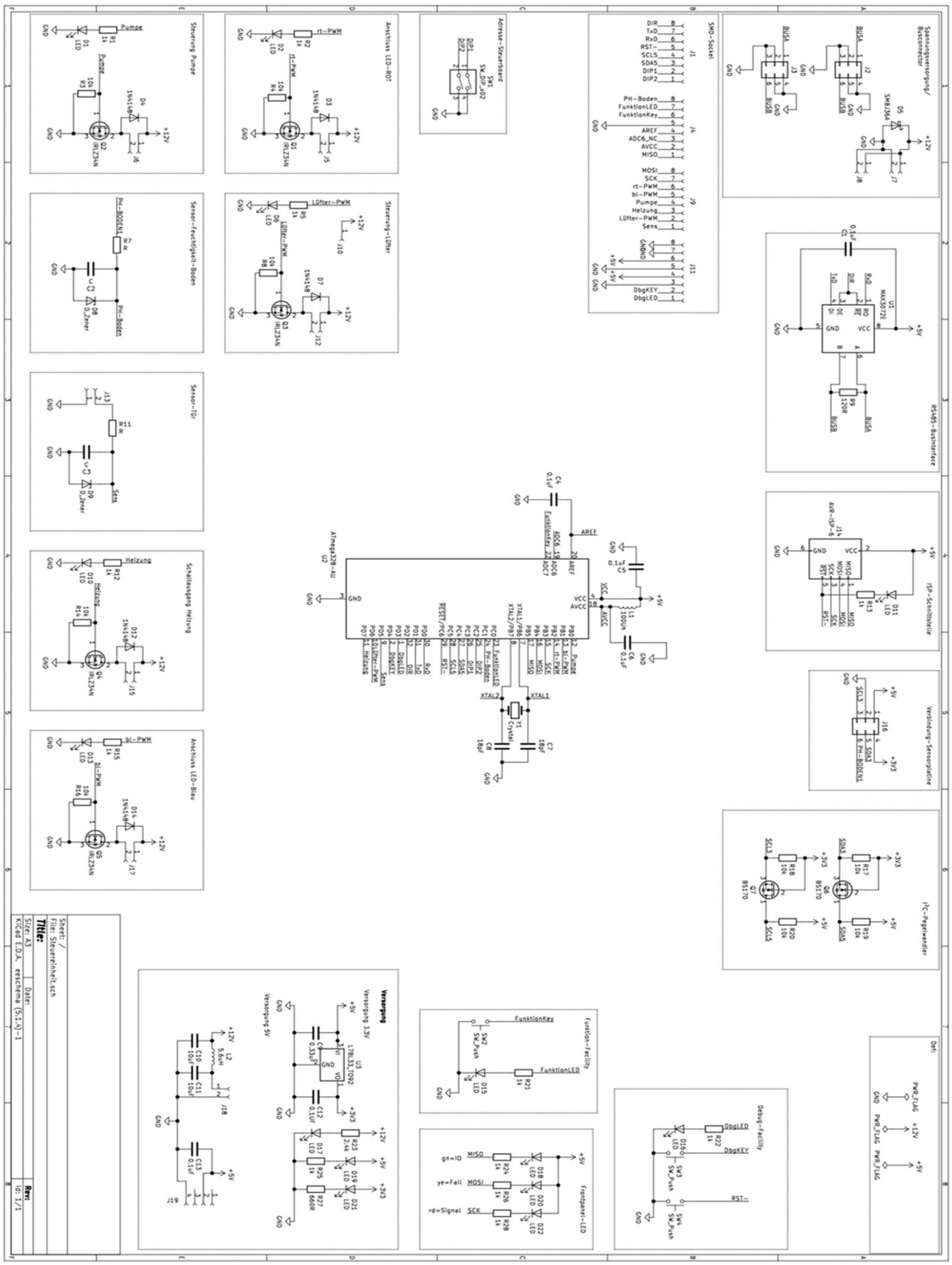
Die fertige Platine, wurde daraufhin bestückt. Wir begonnen zuerst mit allen Durchkontaktierungen. Diese wurden mit Fädeldraht auf der unteren und oberen Seite mit der Leiterbahn verbunden. Danach folgten die SMD-Bauteile. Zuletzt wurden THT-Bauteile verbaut. Diese wurden auf der Ober- und Unterseite verlötet.



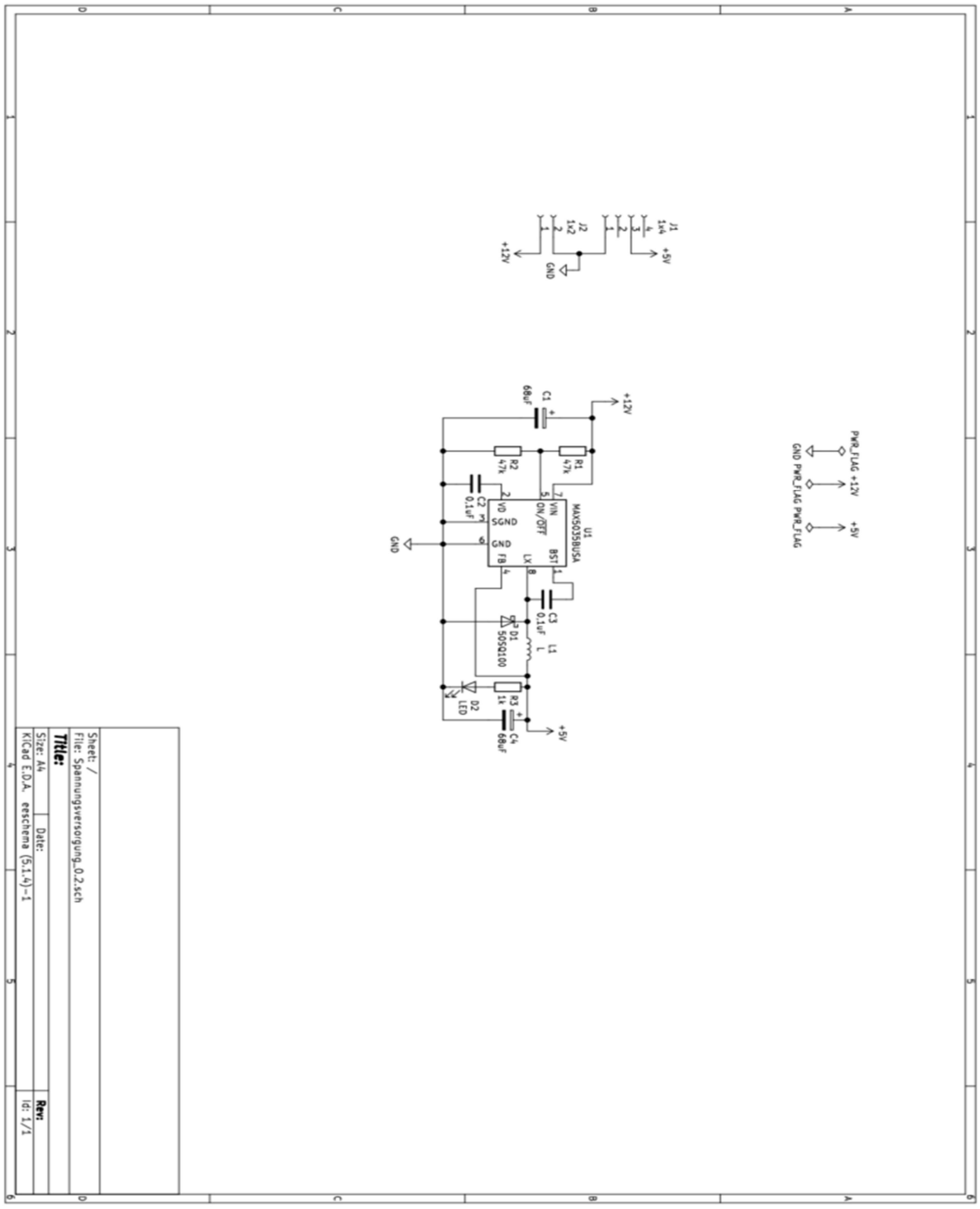




# Schaltplan Steuerplatine



## Schaltplan Spannungsversorgung



# Schaltplan Dokumentation

Im Folgenden werden nun einzelne Abschnitte der Schaltung anhand des zuvor abgebildeten Schaltplanes erklärt.

**Micro-Controller-Beschaltung:**

- Die Eingänge **VCC** und **AREF** (Versorgung für CPU und Spannungsreferenz des AD-Wandlers) sind mit 100nF Blockkondensatoren beschalten.
- Der Spannungsversorgungseingang **AVCC** ist mit einem RC-Glied beschalten. Die Versorgung für den AD-Wandler ist somit durch die RC-Glied-Siebung sehr konstant und gegen Störungen resistent.
- Da der Interne Oszillator als **Taktgeber** sehr ungenau ist, haben wir uns für einen externen **Quarz** entschieden. Zwei Keramikkondensatoren sind für das Anlaufen des Quarzes verschalten.

**Spannungsversorgung:**

- Wir haben uns für eine **Hauptspannungsversorgung** von **12V** entschieden, da alle unsere angeschlossenen Geräte mit dieser Spannung arbeiten. Zum Schutz gegen Störungen haben wir am 12V Eingang eine Supressionsdiode vorgesehen. Diese ist im Schaltplan als Shotkey-Diode dargestellt (D5)
- Die interne Spannungsversorgung für den Micro-Controller liefert ein Schaltregler (StepDown-Controller), welcher auf einer Aufsteckplatine liegt. Der Grund, warum wir nicht einfach auf einen Festspannungsregler z. B. den 7805 gesetzt haben, ist die Verlustleistung am Spannungsregler. Ein Wandler ist dabei viel effizienter und den Aufpreis wert.
- Bei der Spannungsversorgung auf 3,3V haben wir uns bezüglich wärme keine Sorgen gemacht, deshalb haben wir hier auch einen Standard Spannungsregler verbaut. Die Stromaufnahme und die abfallende Spannung am Regler ist als klein einzustufen.

**RS485 und I²C – Businterface:**

- Die Kommunikation der einzelnen Steuerplatine mit unserer Rasberry Pi erfolgt über den RS485 BUS. Dazu wurden die Ausgänge RxD(Recieve), TxD(Transmit) und Dir(Direction) am Mikrocontroller genutzt.
- Der Datenaustausch über I²C erfolgt über die dafür vorgesehen Ausgänge SCL (Clock) und SDA (Data). Da unsere I²C-Module eine Versorgung von 3,3V benötigten und unser MC mit 5V arbeite, muss der Pegel angepasst werden. Mit einem Pegelwandler, welcher von 5V auf 3,3V und umgekehrt wandelt haben wir dieses Problem behoben.

**Ausgänge:**

- Um die Ausgänge des MC zu schützen, werden in der Schaltung auf MOS-FET's verwendet. Diese eignen sich sehr gut, da sie im geschalteten Zustand eine sehr niedrige DS-Spannung haben und allgemein dank ihrer kapazitiven Charakteristik, leistungslos gesteuert werden können. Die 10kΩ Widerstände zwischen Gate und Source, ziehen das Gate während der RESET-Phase des MC sicher auf GND. Die LED's parallel dazu, sind für die Entwicklungsarbeit (Software) vorgesehen.

**Debug-Facility:**

- Da unser Mikrocontroller keine JTAG-Schnittstelle besitzt, war uns klar, dass das Debugen auf diese Art und Weiße nicht funktioniert. Um aber Funktionen austesten zu können haben wir von Anfang an zwei LED's und zwei Taster vorgesehen. Bis auf den Taster „Funktionkey" waren diese auf Digitale D/O Pins gelegt. Der Taster „Funktionkey" liegt auf einen Analogen Eingang und wird über den Gemessenen Spannungswert ins Programm eingebunden.

**Adresseinstellung:**

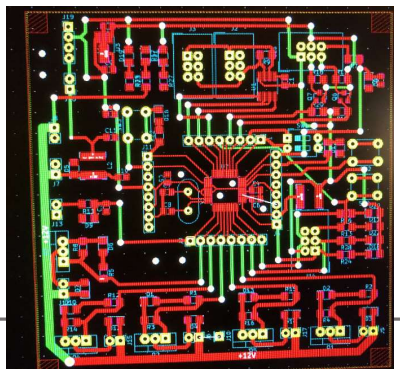
Über den 2-Fachen Dip-Schalter kann die Adresse der Steuereinheit eingestellt werden. Dabei kann zwischen 4 Adressen gewählt werden. Die eingestellte Adresse ist der Binärwert der Schalterstellung und somit von 1 – 4 einstellbar. Die Adresse 0 ist unserem Rasberry zugeordnet.

**Sensoreingänge:**

Die Eingänge „Sensor-Feuchtigkeit-Boden" und „Sensor Tür" sind beide mit einer Schutzbeschaltungen beschalten. Die Vorwiderstände bilden zusammen mit dem Blockkondensator ein RC-Glied, welches Spannungsspitzen glättet. Die Zenerdiode verhindert eine zu große, dauerhaft anliegende Spannung.

**Layout:**

Beim Layout wurde im Allgemeinen darauf geachtet, dass die Waagrechte und Senkrechte Verlegung von Leiterbahnen getrennt auf Vorder- und Rückseite verläuft. Leiterschleifen wurden Ebenfalls vermieden. Nicht benutzte Flächen der Platine wurden mit Masseflächen aufgefüllt um eine hohe EMV (Elektromagnetische Verträglichkeit)-Festigkeit zu erreichen



## Software Dokumentation

**Programmiersprache:** C
**Entwicklungsumgebung:** AtmelStudio 7.0
**Programmer:** ALL AVR (ISP)
**Schnittstellentestsoftware:** H-Term
**Taktrate:** 8Mhz
**Compiler**: GCC
**Fusebits**: 0x7F, 0xDF, 0xFF (L / H / E)

**Programmaufbau:**

**Steuerung der Ausgänge:**
Die Steuerung der Ausgänge erfolgt über das Aufrufen der Funktion „Set…()“ und den übergeben der Zustandsvariablen (On oder Off). Programmintern wird der übermittelte Wert über Bedingungen zugeordnet. Das Bit wird nun im PORT-Register gesetzt oder Rückgesetzt.

**Abfragen der Eingänge:**
Das Einlesen der Eingänge erfolgt über eine Funktion mit einen uint8_t Rückgabewert. Intern wird der Inhalt des gesamten Ports in einer lokalen Variable gespeichert. Über eine spezifische Maske werden alle nicht relevanten Bits in diesem Byte gelöscht. Stimmt die Maske überein, wird nach einem kurzen Debouncedelay der Eingang noch einmal ausgewertet, um ein Prellen auszuschließen. Der Funktion kann nun ein Rückgabewert übergeben werden.

**Moduladresse:**
Um die Software-Adresse unserer Steuereinheiten festzulegen können über die zwei Dip-Schalter vier mögliche Adressen konfiguriert werden. Dabei werden alle vier Möglichkeiten nach einander abgefragt. Jede Möglichkeit bekommt eine Zahl, welche im Falle des Zutreffens in eine Lokale Variable geschrieben wird. Der Wert der Lokalen Variable wird am Ende des Unterprogrammes übergeben. Die Adresse 0 besitzt der Master (Rasberry).
Diese Funktion wird einmal in „MyInitProgramm“ aufgerufen. Adressen können somit nur über ein Reset aktualisiert werden.

**Abfragen des ADC-Werts:**
Die Messung eines ADC-Werts, erfolgt nicht über einen „Interrupt“. Es wird lediglich im jeden Durchlauf des Hauptprogrammes eine Messung gestartet. Zuerst wird dabei im „ADMUX-Register“ der richtige „Channel“ ausgewählt. Wird eine „Single-Wandlung“ gestartet, wird in einer Schleife auf das Ergebnis gewartet und der Funktion übergeben. Zum Schluss wird der Channel des „ADMUX-Register“ auf 0 gesetzt.
Kommende Versionen dieser Funktionen sollen eine Mittelwertbildung mit eingebaut haben.

**RS485 Bus**
Den Code für das Bus-Interface bekamen wir von unserem Betreuer. Aus einen seiner Projekte haben wir uns die Programmteile kopiert und passten sie an unseren Code an. Dies hat uns sehr viel Zeit gekostet, da keiner von uns zuvor sich an ein solches Projekt gewagt hat. Um Zeit zu sparen, wurden Funktionen, wie das Auswerten der Prüfsumme nicht mit eingebunden. Im Hauptprogramm wird in jedem Durchlauf überprüft ob ein Datenpaket empfangen wird. Wird ein Datenpaket empfangen, löst dies einen „Interrupt“ aus.

**Aufbau des Receiver-Datenpakets:**

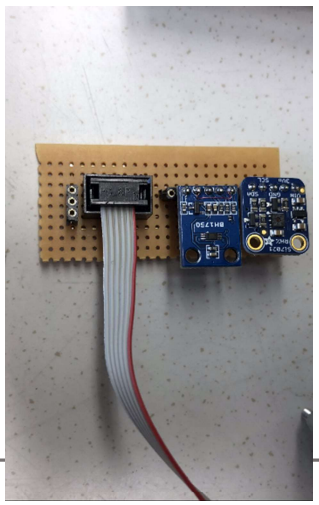
1. Startzeichen (ASCII: S // HEX: 53)
2. Prüfsumme (Prüfsumme ist immer 0)
3. Zieladresse (Adresse des Rasberry: 0)
4. Quelladresse (Globaler Wert aus der „GetAdress-Funktion“)
5. Funktionscode (Nicht belegt)
6. Datenlänge (Länge der Folgenden Bytes. Typ: 4Bytes)

**Aufbau des Transmitter-Datenpakets:**
Besteht aus 6 Datenbytes, wobei 4 Bytes für Daten gedacht sind. Durch aufrufen der Funktion RoBus4BSend können 4Byt gesendet werden.

Bus-Funktionen:
In einer ausgelagerten Datei, sind für jede Funktion der Buskommunikation Konstanten definiert. So können schnell neue Funktionen entstehen und geändert werden.
Dabei sind folgende Funktionen zu Stande gekommen:

- Grundfunktionen wie: Ping, ACK, Hello World und das Auslösen eines Reset
- Abfrage von Helligkeit, Temperatur, Luft- und Bodenfeuchtigkeit
- Abfragen wie Hauptprogramm-Modus, Zustand aller Ein- und Ausgänge
- Befehle wie das Übergeben der Grenzwerte für die Sensoren, Steuerung aller Ausgänge und Wahl des Hauptprogrammmodus.

**Sensoren:**
Um die relevanten physikalischen Werte für unser Gewächshaus messen zu können, kommen vier verschieden Sensoren zum Einsatz. Die Boden-Feuchtigkeit wird dabei mit einen analogen Sensor gemessen. Dieser gibt einen Spannungswert aus, welcher vom ADC Ausgewertet wird. Helligkeit, Feuchtigkeit und Temperatur werden von I²C-Sensoren gemessen. Die verschieden I²C-Routinen haben wir aus einen Buch „Hardware und C-Programmierung in der Praxis AVR“ übernommen. Zum Aufbau des Abfrageprogrammes haben wir uns an die Datenblätter der Sensoren gehalten.



**Hauptprogramm:**
Alle Unterfunktionen (Unterprogramme) dienen der Funktion unserer vier Hauptprogramme. Diese sind für die Steuerung des Lichtes, der Heizung, der Pumpe und der Lüftung zuständig. Diese Funktionen werden die über den Bus übergebenen, Grenzwerte übermittelt. Dadurch können die Istwerte mit den Sollwerten abgeglichen, und somit die Peripherie gesteuert werden. Um unnötig Schalten der Ausgänge um den Grenzwert vorzubeugen, werden immer maximale und minimale Grenzwerte übermittelt. Somit Bildet sich eine Hysterese.

**Heizung „ON“**

| | | |
|---|---|---|
| Temp | < | Grenzwert min „Temp“ |
| Humi | < | Grenzwert min „Humi“ |
| GND-Humi | > | Grenzwert min „GND-Humi“ |

**Heizung „OFF“**

| | | |
|---|---|---|
| Temp | > | Grenzwert max „Temp“ |
| Humi | > | Grenzwert max „Humi“ |
| GND-Humi | < | Grenzwert max „GND-Humi“ |

**Licht „On“**

| | | |
|---|---|---|
| Lumi | < | Grenzwert min „Lumi“ |

Licht „Off“

| | | |
|---|---|---|
| Lumi | > | Grenzwert max „Lumi“ |

**Lüftung „On“**

| | | |
|---|---|---|
| Temp | > | Grenzwert max „Temp“ |
| Humi | > | Grenzwert max „Humi“ |
| GND-Humi | > | Grenzwert max „GND-Humi“ |

**Lüftung „Off“**

| | | |
|---|---|---|
| Temp | < | Grenzwert min „Temp“ |
| Humi | < | Grenzwert min „Humi“ |
| GND-Humi | < | Grenzwert min „GND-Humi“ |

Bei der Steuerung der Pumpe ist uns erst im Nachhinein ein folgenschweres Problem aufgefallen. Ändert sich nach der Bewässerungszeit nicht schlagartig der Wert der Bodenfeuchtigkeit ins Gute, so wird in dem nächsten durchlauf des Hauptprogrammes, wider für dieselbe Zeit bewässern. Da dies durchaus möglich ist, wird es höchstwahrscheinlich zu Überschwemmungen kommen. Abhilfe dagegen wäre ein interner Timer, welcher nach der Bewässerung den Wert für ca. 10min auf „OK“ setzt. Da wir aber von vornherein nicht mit einen Timer in unserem Programm gerechnet haben, wird es für uns schwierig sein eine solche Funktion nachträglich einzubauen. Deshalb haben wir uns für eine Monostabile Kippstufe entschiede, welche erst nach 10 Min die Pumpe wider einschalten lässt. Diese wird am Ausgang der Steuerplatine Signale welche sich in diesem 10Min Zeitraum ergeben, ignorieren.

**Pumpe „On“**

| | | |
|---|---|---|
| GND-Humi | < | Grenzwert min „GND-Humi“ |

**Pumpe „Off“**

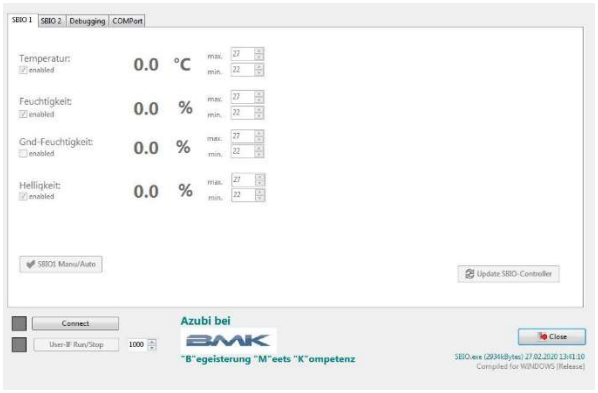
| | | |
|---|---|---|
| GND-Humi | > | Grenzwert max „GND-Humi“ |

# Software Dokumentation Rasberry Pi

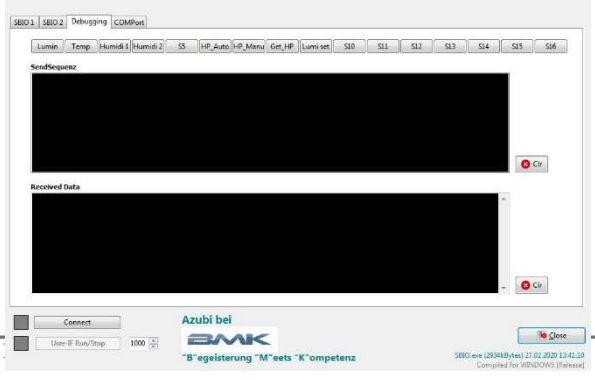
**Programmiersprache**: Lazarus
**Schnittstellen-Library:** LazSeriell
**Compiler**: Pascal-Compiler (FPC)
**Rasberry**: Rasberry 3 A

Als Hauptsteuereinheit viel die Entscheidung auf einen Rasberry Pi. Nachdem die Buskommunikation via H-Term sehr gut geklappt hat, dachten wir uns, dass ein solches Programm, angepasst an unsere Bedürfnisse, perfekt wäre. Zudem fehlte uns auch ein wenig die Zeit, eine eigene Hauptsteuereinheit zubauen, da es schon sehr lange gebraucht hat, bis eine fertige Steuereinheit entwickelt wurde. Da von uns noch nie jemand mit Lazarus oder einer solchen Hochsprache in Berührung gekommen ist, suchten wir uns bei einem Entwickler Hilfe. Zusammen mit ihm war das Grundgerüst in wenigen Stunden fertig. Dabei lernten wir nicht nur sehr viel, sondern waren auch begeistert, auf was für einen Stand die Technik heutzutage ist.

## Aufbau der Benutzeroberfläche.








**COMPort:**
Im Reiter „ComPort“ werden alle möglichen Einstellungen zur Seriellen Daten-Kommunikation getroffen.

**Debugging:**
Der Reiter „Debugging“ ist für das Manuelle Senden und Empfangen von Daten zuständig. Über die Buttons oben, können Shortcuts für Datenpakete angelegt werden. Diese werden dann über das obere Fenster gesendet und im unteren Fenster empfangen.

**SBIO1:**
Im Reiter „SBIO 1“ werden alle Messwerte wie Temperatur, Feuchtigkeit, GND-Feuchtigkeit und Helligkeit live angezeigt. Dazu wird jede Sekunde ein Datenpacket mit vier „Get“ Befehlen an die Steuereinheit geschickt. Die Steuereinheit gibt dabei die Rohen Werte der Sensoren weiter. Zur Auswertung der Daten ist die Main-Einheit zusändig. Da bei der Helligkeit immer ein sehr hoher Luminance-Wert zurück kommt, ist es unserer Meinung nach einfacher, diesen in einem Prozent-Wert auszugeben.
Über die einstellbaren Bereiche neben den Messwerten ist ein Senden und Einstellen der Grenzwerte Möglich. Sobald an diesen Bereichen eine Änderung vorgenommen wird, kann über den Button „Update SBIO-Controll“ das neue Grenzwerte-Paket gesendet werden.
Über den Button „Disconnect“ kann die Verbindung zur Steuereinheit unterbrochen werden.
Der Button „Program Run/Stopp“ ist für das Starten oder Stoppen der Daten-Pakete verantwortlich. Über das Einstellbare Feld daneben kann die Update-Rate der Messwerte eingestellt werden.

**SBIO 2:**
Der Reiter „SBIO2“ ist im Moment noch nicht belegt, wird aber, sobald es in unserem Gewächshaus eine zweite Ebene gibt, mit eingebaut.

## Mechanischer Aufbau

Das Grundgerüst des Gewächshauses besteht aus Item-Profile. Aus Kostengründen nutzen wir gebrauchte ITEM-Profile aus unserer Werkstatt. Unsere gewünschten Maße konnten dabei nicht realisiert werden. Zum jetzigen Zeitpunkt konnten wir uns noch nicht um eine seitliche Abdeckung des Grundgerüsts kümmern. Unser Plan dabei ist, jeweils zwei Seiten mit einer wasserabweisenden Pressspanplatte und zwei Seiten mit Plexiglas zu versehen. Die Pflanzenkiste soll ebenfalls mit Pressspanplatten und einer wasserdichten Folie gefertigt und abgedichtet werden.

Aktueller Stand des Grundgerüsts:

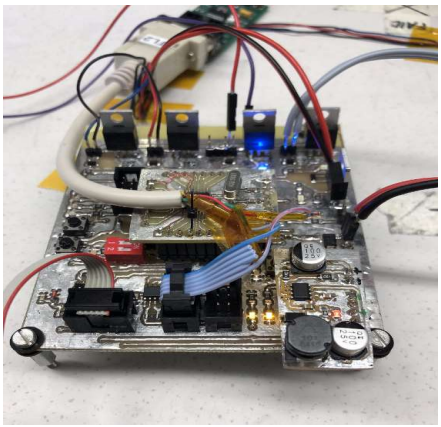
## Erfahrungen und Fehler

Das Projekt „SBIO“ hat uns in vielerlei Hinsicht weitergebracht. So haben wir viel im Bereich Mikrocontroller-Hardware-Entwicklung t, sowie das Programmieren in C und Lazarus gelernt. Ebenso erhielten wir nützliche Informationen im Bereich Layout-Entwurf und lernten mit KiCad umzugehen.

Leider lief nicht immer alles reibungslos und wir mussten mit immer mehr Abgängen im Team zurechtkommen, sodass wir am Ende nur noch zu viert an dem Projekt gearbeitet haben. Während des Programmierens stießen wir auf Fehler, welche wir nachträglich ausbessern mussten. Durch strukturierteres Arbeiten hätte vieles im Bereich Layout einfacher gestaltet werden können.

Fehler die gemacht wurden:

- MISO, SCK, MOSI – Leitungen vertauscht
- Pull-up Widerstand am Analogeingang vergessen. Pin hängt jetzt in der Luft und definiert sich seinen High-Pegel über Störungen aus der Luft.
- Zu große Dimensionierung der MOSFET’s
- Falsche Footprints ausgewählt, Bzw. im Vorfeld die Verfügbarkeit nicht überprüft



## Teamvorstellung

Paul Kaupp
**Alter**: 18 Jahre
**Hobbys**: Analoge Elektrotechnik, Wandern, Fotografieren und Schalplatten hören
**Aufgaben im Projekt**: Steuereinheit, Layout, Koordinator

Leon Kempter
**Alter**: 18 Jahre
**Hobbys**: Shisha Rauchen, Zeit mit Freundin verbringen
**Aufgaben im Projekt**: Beschaffung und Planung der Mechanik, Layout

David Dreyer
**Alter**: 18 Jahre
**Hobbys**: Tanzen, Klavierspielen, Japanisch, Sanitäter, Fitness, Motorrad fahren, Basteln/Werkeln und Geflügelzucht
**Aufgaben im Projekt**: Steuereinheit, Aufbau der Schaltung

Maximilian Hoffmann
**Alter**: 17 Jahre
**Hobbys**: Elektronik basteln, Klavier spielen und Modeleisenbahn bauen
**Aufgaben im Projekt**: Programmierung