Hygro#3

Carina Primas, Dominik Schmidt, Patrick Stillrich, Alexander Resnik

17. September 2020

Inhaltsverzeichnis

1 Einleitung

Mit diesem Produkt soll die Luftfeuchtigkeit innerhalb und außerhalb eines Gewächshauses kontinuierlich gemessen werden, um dadurch zu gewährleisten, dass die Pflanzen optimale Feuchtigkeitsbedingungen haben. Es soll beispielsweise eine Pumpe bei zu niedriger Luftfeuchtigkeit bedienen können und eine Belüftung bei zu hoher Luftfeuchtigkeit aktivieren. Der Funktionsbereich liegt zwischen -10°C und 45°C.

2 Technische Rahmenbedingungen

Das Produkt wird aus 3 Einheiten bestehen. Eine davon wird eine Basisstation sein, die anderen 2 werden Satelliten sein.
Jede Einheit wird über einen Sender und einen Empfänger verfügen. Mit diesen wird eine 2- Richtungskommunikation realisiert. An jedem Satelliten wird ein Feuchtigkeitssensor verbaut. Diese Feuchtigkeitssensoren werden mit einem Mikrocontroller verbunden. Ziel ist es zuerst eine Grundfunktion herzustellen, im Nachhinein soll es eine Möglichkeit geben weitere Sensoren hinzuzufügen, Beispielsweise einen Lichtsensor, einen Luftdrucksensor, Bodenfeuchte oder einen CO2- Sensor. Die Satelliten werden mit Hilfe eines Akkus versorgt, dabei wird die Spannung überwacht. Außerdem werden wir Schutzfunktionen für Überlast und Kurzschlüsse hinzugefügt. Die Basis wird über Relais verfügen. Hiermit kann dann eine Abluft / Zuluft geschaltet werden. Mittels 2 Tastern kann man die minimale und maximale Luftfeuchtigkeit anpassen. Dies passiert in einer Vorgegebenen Schrittweite. Des Weiteren soll es möglich sein die aktuellen Daten an einem Display auszulesen.

3 Hardware

3.0.1 Analyse und Vorgehensweise Hardware

Die Vorgehensweise in der Hardwareentwicklung bestand zum einen Teil aus dem Aufbau einer Schaltung mit einem Arduino Uno und den Sender- und Empfängermodulen mithilfe eines Bradboards. Nachdem diese Schaltung funktionierte, wurde die Schaltung analysiert und ein Schaltplan entwickelt. Hierfür wurde der im Internet verfügbare Schaltplan eines Arduino Uno analysiert und überarbeitet.

3.0.2 Stromlaufplan

Nachdem alle benötigten Bauteile identifiziert waren, wurde der Blockschaltplan gezeichnet.
Nach dem Erstellen eines Blockschaltplanes und der Software für den Prototypen, ging es nun an das Erstellen des Schaltplanes und die Entflechtung der Boarddatei für die Platine. Hierfür wurde die CAD-Software Eagle benutzt, da in den Bibliotheken dieser Software und auch im Internet für fast alle erdenklichen Bauteile bereits die sogenannten Footprints für die Board- als auch die Schaltplan-Symbole für die Schaltplandatei zu finden sind. Hierfür mussten diverse Anschlusspins des Microcontrollers neu belegt werden, da Sender und Empfänger auf unterschiedlichen Pins kommunizieren sollten, und die Software bereits so geschrieben war.

3.0.3 Spannungsversorgung

Da auf dem Arduino die Versorgungsspannung per USB eingespeist wird, wir jedoch ein Netzteil mit 12V angeschlossen haben, mussten wir die Spannung zwei mal auf kleinere Spannungswerte drosseln. Dies geschieht mithilfe zweier Microcontroller, die im Schaltplan mit U1 und U2 gekennzeichnet sind. Die 12V der Spannungsversorgung sind nötig, weil der Sender eine Versorgungsspannung von 12V hat und es einerseits einfacher ist, Spannungen herunterzutransformieren und zum anderen ist es in der Basis des Systems nicht unbedingt notwendig Energie zu sparen, da dieses sich per Netzteil versorgt und keine Rücksicht auf Laufzeiten von Batterien genommen werden musste.
Die LEDs sollen eine intakte Versorgung mit der jeweiligen Spannung darstellen. Für eine durchgehende Versorgung ohne Schwingungen dienen die zwei Kondensatoren am Ausgang des ICs.

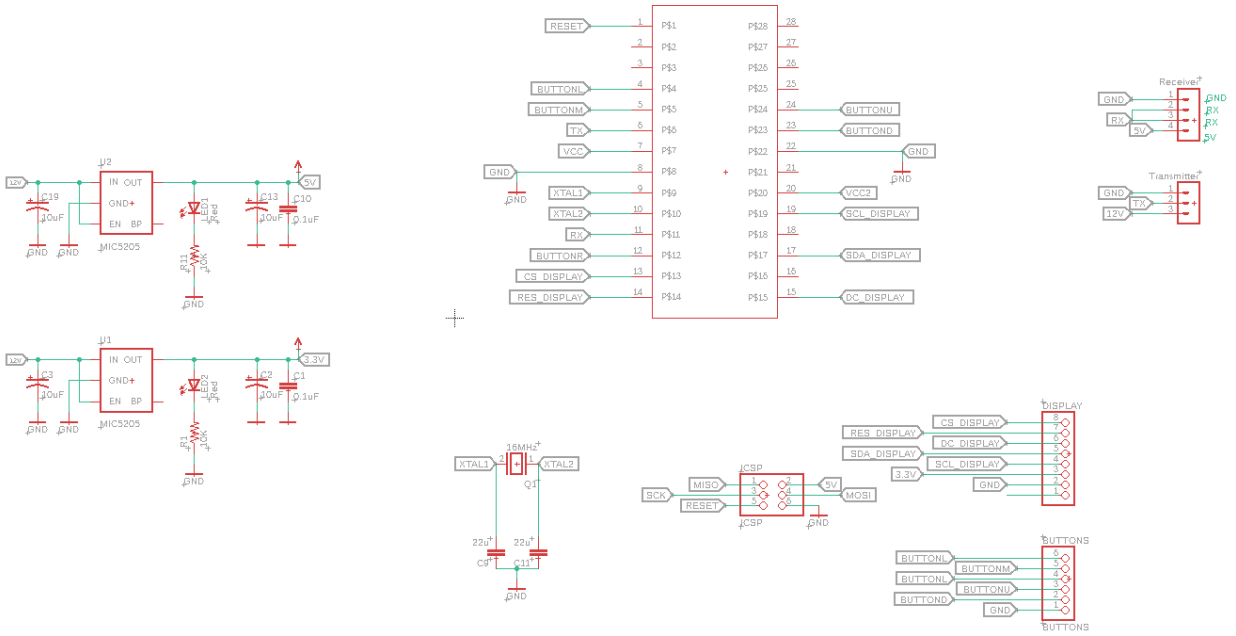
3.0.4 Steuerung

Die Eingänge der Knöpfe zur Steuerung wurden einzeln auf die Eingänge des Mikrocontrollers gelegt. Die Verbindung der Platine zu den jeweiligen Knöpfen erfolgt durch eine Stiftleiste.

3.0.5 Verbindung des Displays mit der Platine

Das Display verfügt auf der Rückseite über mehrere Anschlüsse, die mit einfachen Pinleisten mit der Platine verbunden wurden. Hierfür wurden folgende Signale angeschlossen:

GND und VCC als Spannungsversorgung. SCL, SDA, RES und CS dienen als Pins für die Datenübertragung per I²C-Bus. Der CS-Pin dient der Unterscheidung der beiden I²C-Busse. (Chip-Select) Hier bekommt das Display die Anweisung, ob und wann es die Daten auf dem I²C-Bus verarbeiten soll.



4 Auswahl der Bauteile

Da auf der Grundlage eines Arduino gearbeitet wurde, und von diesem Prinzip nach Möglichkeit nur wenig abgewichen werden sollte, wurde sich darum bemüht, möglichst die gleichen Bauteile wie auf dem bereits vorhandenen Arduino Uno zu verwenden.

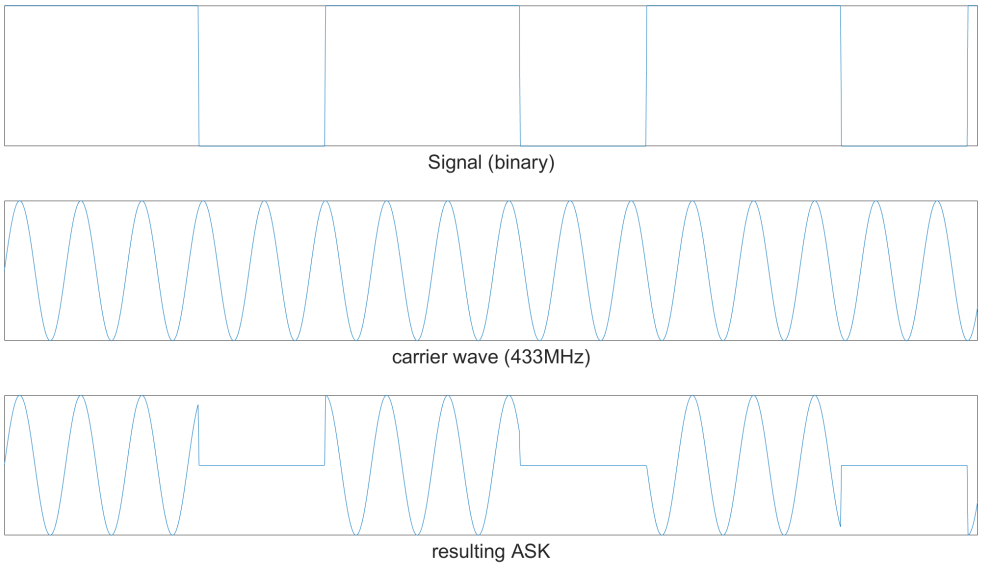
4.0.1 Der Microcontroller Atmega328

Der verbaute Atmega 328 verfügt über 16 Pins und hat einige Vorteile gegenüber vergleichbaren ICs. Er ist einfach aufgebaut, einfach zu programmieren, günstig und zählt zu den sogenannten „Low-Power-Microcontrollern“. …(mehr Infos)
Da der Atmega328 einen externen Quarz braucht, um genau zu funktionieren, musste ein passender Quarz gesucht werden. Je höher der Takt, desto größer ist auch sein Stromverbrauch. Da der Microcontroller jedoch mit höherer Frequenz auch schneller arbeiten kann, wurde ein Quarz mit 16 Hz bei diesem Projekt verwendet.

5 Software

5.1 Into

Bei der Funkverbindung wurde auf das Radiohead-Paket[1] zurückgegriffen. Dieses Paket unterstützt viele gängige Sender/Empfänger-Kombinationen. Des weiteren ist anzumerken das hier die “einfache“ASK[1] Modulationsart verwendet wird. Dies wird durch das sogenannte On-Off Keying realisiert.



Nachrichten des Radiohead-Pakets bestehen in unserer Anwendung aus unterschidlichen Teilen. Daraus ergibt sich eine stabile Funkverbindung.

36 Bits	12 Bits	8 Bits	16 Bits	n Bits	16 Bits
Training Preamble for timing	Start Symbol 0xb38	Nachrichtlänge	Frame Check Sequence	Nachrichten payload	FCS 0x0F

Alle Daten welche nach dem Start Symbol versendet werden codiert übertragen. Somit ist ein Byte 2x6 bit lang.

5.2 char bare protocol (cbp)

Im zuge dieser Arbeit wurde ein eigenes Protokoll entwickelt, welches auf eine Paket basierte Übertragung, mit einem Master und n Slaves, aufbaut. In diesem Fall wird das

[1] Amplitude Shift Keying/Amplitudenumtastung

Radiohead-ASK. Das Protokoll wurde cbp[2] genannt. Um in diesem Protokoll Daten zu Übertragen werden die Daten zu einem Char-array / einem String zusammengefügt. Verschiedene Daten werden durch einen Buchstaben gekennzeichnet. Anschließend werden die zu übertragenden Daten angehängt. Das format der Daten ist float oder int. Datensätze werden durch ein Komma getrennt. Ein Beispiel für einen solchen Datensatz könnte sein:

```
v3.8,t26.5,h33.2
```

Wie die Verschidenen Zeichen interpretiert werden wird Ebenfalls festgelegt:

Char	Einheit	Funktion
v	V	Batteriespannung
t	$°C$	Sensor Temperatur
h	%	Relative Luftfeuchtigkeit
a	$-$	Anzahl der verfügbaren Aktoren

5.3 Satelit

5.3.1 Versorgungsspannung

Um die Versorgunsspannung zu messen wird die interne $1.1V$ interne Spannungsreferenz des ATmega verwendet. Der verwendete ADC verwendet als maximal Spannung die Versorgungsspannung. Diese Entspricht also dem Maximum von 1024 (10 Bit). Durch Messung der Internen Referenzspannung können dann Rückschlüsse auf die Versorgungsspannung getroffen werden. z.B. ergibt die Messung der Referenz einen Wert von 350, so kann durch die Formel $\frac{1.1V}{ADCResult} * 1024$ berechnet werden. Siehe 7.1 Zeile 8.

5.3.2 low power

Die Anforderungen an die Laufzeit wurden durch ein “deep-sleep “realisiert. Hierfür wurde auf das git von Rocketscream zurückgegriffen[2]. Hier kann eine maximale sleep Dauer von $8s$ eingestellt werden. Für größere Zeiten muss ein Externer Interrupt festgelegt werden. Um eine weitere externe Schaltung zu vermeiden wird hier die Clock abgeschalten nicht abgeschaltet. Diese wird verwendet um einen Timer zu steuern, welcher nach $8s$ einen Interrupt erzeugt. Mit diesem Wacht der ATmega auf. Dies wird wiederholt um größere Zeiten zu erreichen. Siehe 7.2

[2]Carmens-beef-protocol

5.3.3 eeh210

5.3.4 cbp Realisierung

Die cbp Realisierung basiert darauf, dass sämtliche Daten sich durch, z.B. `dtostrf();`, zu einem String formatieren lassen. Diese Teilstrings werden anschließend durch Kommas getrennt aneinandergereiht. Hierfür wird hauptsächlich die funktion `strcat();` verwendet. Siehe 7.3

5.4 Basis

6 Ergebnis

7 Ausblick

8 Anhang

8.1 Akkuspannung

```
float fReadVcc() {
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
  delay(5); //delay for 5 milliseconds
  ADCSRA |= _BV(ADSC); // Start ADC conversion
  while (bit_is_set(ADCSRA, ADSC)); //wait until conversion is complete
  int result = ADCL; //get first half of result
  result |= ADCH << 8; //get rest of the result
  float batVolt = (iREF / result) * 1024; //calculate battery voltage
  return batVolt;
```

8.2 lowpower sleep

```
  digitalWrite(ENABLE_RXTX, LOW);
  for(;low_power_sleep<20;low_power_sleep++){
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
  }
  low_power_sleep = 0;
```

8.3 cbp Satelit

```
  char data[24] = "";
  char temp[5] = "v";
  strcpy(data, temp);
  dtostrf(fReadVcc(), 4, 2, temp);
  strcat(data, temp);

  strcat(data, ",h");
  dtostrf(rh, 3, 1, temp);
  strcat(data, temp);

  strcat(data, ",t");
  dtostrf(t, 3, 1, temp);
  strcat(data, temp);
```

8.4 basis

```
dummy
```

Literatur

[1] AirSpayce Pty Ltd, *RadioHead Packet Radio library for embedded microprocessors*. [Online]. Available: http://www.airspayce.com/mikem/arduino/RadioHead/

[2] Rocket Scream Electronics, *Low-Power*. [Online]. Available: https://github.com/rocketscream/Low-Power