IoT Hackathon -

Eine C-Code Blaupause für eigene Ideen

Nationaler IT Gipfel

Plattform Innovative Digitalisierung der Wirtschaft
Fokusgruppe Intelligente Vernetzung

1 IoT Hackathon IT-Gipfel 2016 3

1.1 Entwicklungsumgebung 6

1.1.1 Arduino-Plattform 6

1.1.2 IoT-Entwicklungskit (ESP-8266) 6

1.1.3 Interfacekomponenten (Library) einbinden 9

1.2 Blaupausen für eigene Projekte 10

1.2.1 Hallo Arduino - Hallo IoT-Kit 12

1.2.2 Die digitale Flaschenpost – Hallo Server 17

1.2.3 Unser Forschungsschiff – Hallo IoT 21

1.2.4 Forschungsschiff informiert sich – Hallo M2M 26

1.2.5 Forschungsschiff greift ein - Hallo Control 28

1.2.6 Wearables – Forschungsschiff kleidet sich 32

1.2.7 Die Flaschenpost wird flexibel – Hallo Interface 33

1.3 Für Hacker & Maker 35

1.3.1 Schaltplan 36

1.3.2 Low Energy Application 37

Vorwort

Wie gelingt der digitale Wandel unserer Gesellschaft? Welche Auswirkungen hat das Internet der Dinge auf die Wirtschaft? Wie hält die schulische Ausbildung mit den Anforderungen der modernen Arbeitswelt Schritt?

Diese und ähnliche Fragen thematisiert der jährlich stattfindende Nationale IT-Gipfel der Bundesregierung. Die Spitzenkräfte aus Politik, Wirtschaft, Wissenschaft und Zivilgesellschaft treffen sich in 2016/17, um sich zum Leitthema „Digitale Bildung" auszutauschen und Handlungsempfehlungen zu diskutieren.

Gemeinsam mit rund 200 Experten aus mehr als 150 Unternehmen und Institutionen bringt die Fokusgruppe „Intelligente Vernetzung" (FG2) des IT-Gipfels die digitale Vernetzung Deutschlands voran. Ein Ziel: Erste Programmierkenntnisse schon in der Schule vermitteln. Unter dem Motto „Das Internet der Dinge (IoT) anfassbar machen" plant die Expertengruppe die Austragung eines „Hackathon" für Schüler.

Schülergruppen aus Rheinland-Pfalz und dem Saarland treffen sich hierzu am Vortag des Gipfels am Umwelt-Campus in Birkenfeld und der HTW in Saarbrücken, um gemeinsam Ideen zum Thema zu entwickeln und mit Hilfe eines von der Expertengruppe bereitgestellten Baukastens auch gleich vor Ort umzusetzen. Die erfolgreichsten Gruppen stellen ihre Ergebnisse dann am nächsten Tag auf dem Nationalen IT-Gipfel vor.

Natürlich kann ein solches singuläres Ereignis keine systematische Bildungsstrategie ersetzen, es soll aber zeigen, dass die technischen Voraussetzungen keine unüberwindbare Hürde für Schulen darstellen und dass das Thema eine nachhaltige intrinsische Motivation der Schüler auslöst. Das Thema Internet der Dinge macht einfach Spaß und Lust auf mehr.

Klaus-Uwe Gollmer Guido Burger

1 IoT Hackathon - eine Blaupause für eigene Ideen

Autoren: Klaus Gollmer[1], Guido Burger[2]

Die digitale Transformation wird unsere Wirtschaft und Gesellschaft in den nächsten Jahrzehnten grundlegend verändern. Der selbstverständliche Umgang mit Sensoren und Kommunikationsmodulen, aber auch deren Programmierung bis hin zur Cloud-Anwendung ist eine Voraussetzung für neue Anwendungsideen und Geschäftsmodelle.

Das dazu notwendige **algorithmische Denken** sollte deshalb zukünftig schon in der Schule geübt werden. Hierzu stellen wir beispielhafte Ideen und ihre technische Umsetzung vor. Fokussiert auf ein schnelles Erfolgserlebnis bieten die kleinen Programme einen Kristallisationspunkt für die Beschäftigung mit dem Thema und der kreativen Umsetzung eigener Ideen.

Im Wettbewerb mit mehreren Teams (**Hackathon**) macht es noch viel mehr Spaß: es entstehen konkurrierende Ideen, aber vielleicht auch ganz neue Lösungen, da man mit anderen Teams zusammenarbeiten kann. Durch die vorbereiteten Beispiele gelingt der Einstieg kinderleicht. Ein für speziell für den Hackathon entwickeltes IoT-Board auf der Basis des bewährten ESP-8266, der Arduino- Entwicklungsumgebung und den Grove Interfacekomponenten erleichtert die praktische Umsetzung auch ohne elektrotechnische Kenntnisse.

Um den unterschiedlichen Kenntnisstand der Leser zu berücksichtigen haben wir uns für zwei Versionen der Blaupausen entschieden: Grafische Programmierung mit **Ardublock** bzw. die direkte Programmierung in der **Hochsprache C**.

---

[1] www.umwelt-campus.de

[2] www.fab-lab.eu

4 Das Internet der Dinge anfassbar machen

Durch einfache Modifikation der Beispiele entstehen daraus komplexere Lösungen für die kleinen Probleme des Alltags:

- **Umwelt:** Sorge dafür, dass die Bewohner von A-Bach und B-Tal in Zukunft rechtzeitig vor einem folgenschweren Unwetter gewarnt werden. Entwerfe einen geeigneten Umweltsensor (z. B. zur Messung der Bodenfeuchtigkeit am Berghang, des Pegels des nächsten Flusses usw.) und verknüpfe die Sensordaten mit dem Wetterbericht.
- **Verkehr:** Reduziere das Verkehrsaufkommen und die Umweltbelastung durch Schadstoffe. Entwickle eine IoT-Anwendung, damit eine Mülltonne vor der Müllabfuhr nur dann angefahren wird, wenn sie voll ist.
- **Industrie 4.0:** Hilf mit, die Arbeitsplätze der Fabrik AG zu sichern. Erfinde intelligente Sensoren (z.B. Stromverbrauch, Bewegung), um die Effizienz der Produktionsmaschinen und somit die Wettbewerbsfähigkeit der gesamten Firma zu verbessern.
- **Kleidung:** Nutze Sensoren und verleihe Kleidungstücken, zum Beispiel über innovative Lichtelemente, einen neuen Ausdruck. Dabei können Daten über die Bewegung eines Menschen erfasst werden (Quantified Self). Dadurch entstehen neue kreative Anwendungen für Freizeit und Sport sowie für ältere Menschen.
- **Virtuelle Welten:** Verknüpfe die virtuelle und die reale Welt unter Nutzung einer Spiele-Plattform im Internet. Spielerisch gelingt der Einstieg in die Sensorik und Programmierung.

Das System ist als **digitale Flaschenpost** konzipiert und trägt damit als Metapher die Nachricht des einfachen Internetzugangs für die Dinge dieser Welt in die Politik und an die Schulen und Hochschulen. In einem zweiten Schritt schicken wir die Flasche als **Forschungsschiff** auf die Reise, um unsere Umwelt zu erkunden und neue digitale Welten zu erobern. So verbindet sich das Internet mit der analogen Welt in der wir leben.

1.1 Entwicklungsumgebung

Die folgende Kurzeinführung soll den Zugang zu Werkzeugen, Technologie und Kreativitätsmethoden erleichtern. Zuerst wird die Arduino-Entwicklungsumgebung und die Hardware vorgestellt, um anschließend anhand einfacher Beispiele in die Welt des **Internets der Dinge** (IoT) einzudringen.

1.1.1 Arduino-Plattform

Mit der Open-Source Arduino-Entwicklungsumgebung steht eine in vielen Projekten bewährte Plattformtechnologie zur freien Verfügung. Softwareseitig bildet der GNU-C-Compiler den Kern einer einfach zu bedienenden Entwicklungsoberfläche. Auf dem Hackathon ist die Entwicklungsumgebung bereits installiert. Eine Installationsanleitung für den heimischen Rechner findet sich auf der Arduino-Hompage[3].

1.1.2 IoT-Entwicklungskit (ESP-8266)

Hardwareseitig sind unzählige kompatible Entwicklungsboards verschiedener Hersteller für wenig Geld verfügbar. Seit kurzem existiert mit dem ESP-8266 eine Plattform[4], die sich aufgrund des niedrigen Preises und der WLAN-Konnektivität ideal für Ausbildungszwecke eignet. Kommerziell erhältliche Entwicklungsboards, wie NodeMCU[5] oder Wemos D1 mini[6], enthalten alle für die Programmentwicklung und Ausführung notwendigen Komponenten inklusive USB und WLAN. Je nach Anwendungsfall werden die benötigten Ein- /Ausgabekomponenten, wie LED, Taster, Sensoren und Aktoren manuell hinzugefügt – ein im schulischen Alltag sehr fehleranfälliges Unterfangen.

---

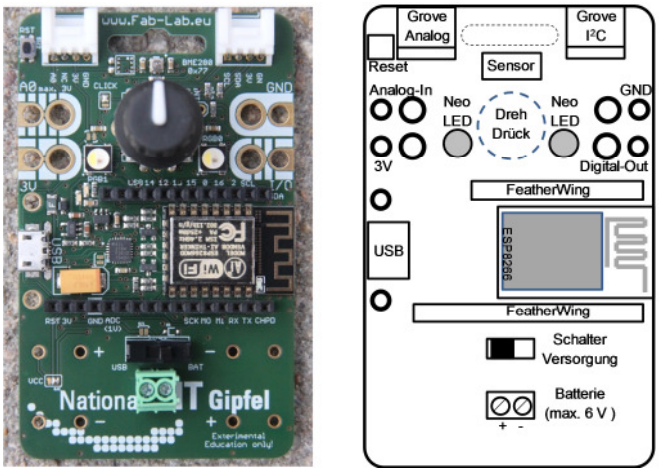
[3] arduino.cc/en/Main/Software

[4] mikrocontroller.net/articles/ESP8266

[5] nodemcu.com

[6] Wemos.cc

Hier reduziert das von uns speziell für Schüler entwickelte IoT-Kit die Verdrahtungsproblematik durch eine Reihe bereits auf dem Board integrierter Komponenten.



IoT- Abb. 1: Das Experimentier-Kit

Das **IoT-Experimentier-Kit** legt damit als quelloffenes Projekt (Schaltplan im letzten Kapitel) die Blaupause für die Ausbildung an Schulen und Hochschulen. Produziert und entwickelt in Deutschland folgt es folgenden Prinzipien:

- **Kein Löten:** Erweiterungen werden einfach angesteckt, und sind durch eine Vielzahl von Sensoren und Aktoren flexibel an die eigene Problemstellung anpassbar.
- **Sensorik:** ausgestattet mit einem Bosch-Sensortec Umweltsensor (Barometer, Luftfeuchtigkeit, Temperatur) wird das Internet der Dinge in wenigen Minuten erlebbar.
- **Interaktiv:** durch „Dreh/Drück“ Eingabe und unterschiedliche Anzeigen (farbige NeoPixel, einfarbige LED).
- **Flexible Energieversorgung:** über USB-Schnittstelle oder unter Verwendung von Standard Batterien bzw. Akkus. Das Laden via Solarstrom ist als „hack“ vorgesehen.
- **Kompatibel:** zur Schulausstattung (Bananenstecker, WLAN und USB) und vorhandenen Communities mit vielen Anwendungsbeispielen (Arduino, Adafruit, SeeedStudio Grove). Die Entwicklungsumgebung läuft auf Windows, Linux und MacOS.

Im Rahmen des Hackathon ist das Board ohne weitere Konfiguration direkt einsetzbar. Da es sich um ein sogenanntes „Third Party Board“ handelt, erfolgt die Anbindung an die Entwicklungsumgebung über den „Boardverwalter“. Um das Board auch am häuslichen PC nutzen zu können muss dort zuerst unter „Datei“ im Untermenü „Voreinstellungen“ die folgende Boardverwalter-URL eingetragen werden:

„http://arduino.esp8266.com/stable/package_esp8266com_index.json“.

Anschließend im Menü „Werkzeuge“ im Untermenü „Board“ den „Boardverwalter“ anwählen und die esp8266 Toolchain installieren. Nach erfolgreicher Installation steht schließlich das Board zur Anwahl zur Verfügung. Dazu unter Werkzeuge, das kompatible Board „NodeMCU V1.0 (ESP12E Modul)“ anwählen. Nach Anschluss des Kits an den USB-Port des PCs sollte der entsprechende Treiber vom Betriebssystem automatisch initialisiert werden. Bei Problemen kann der Treiber manuell nachinstalliert werden[7]

Anschließend unter „Werkzeuge“ -> „Port“ den vom Betriebssystem vergebenen COM-Port eintragen.

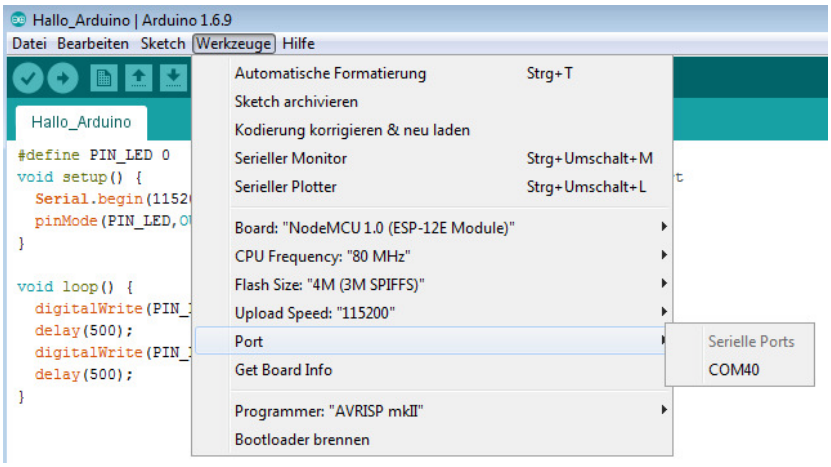


Abb. 2: Einbindung in die Arduino-Entwicklungsumgebung

[7] https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

1.1.3 Interfacekomponenten (Library) einbinden

Die bereits auf dem Board integrierten Interfacekomponenten benötigen zur einfachen Ansteuerung die Einbindung spezieller Bibliotheken. Damit kann die betreffende Komponente dann im Programmcode sehr einfach über Arduino-Befehle angesprochen werden. Ein sonst übliches intensives Studium der Datenblätter entfällt. Dieses flexible Bibliothekskonzept ist eine der Gründe für die enorme Verbreitung der Arduinoplattform. Insbesondere aufwendige Sensorprotokolle sind als Bibliotheken vorhanden und machen deren Nutzung komfortabel.

Auf dem Hackathon sind alle benötigten Bibliotheken bereits installiert. Diese können, ähnlich wie beim „Boardmanager", mit Hilfe eines „Librarymanager" eingebunden werden (Menü „Sketch"->"Bibliothek einbinden"). Um die Bibliotheken auch am häuslichen PC nutzen zu können, enthält die Tabelle alle im Rahmen unserer Blaupausen eingebundenen Komponenten.

Tab. 1: Einzubindende Bibliotheken

Interface-komponente	Wofür	Library
Bosch BME280	Temperatur, Druck, Feuchte	Sparkfun BME280 V1.0.0
NeoPixel	RGB-LED Anzeige	Adafruit NeoPixel V1.0.5
Dreh/Drück	Drehknopf (Rotary-Encoder)	Encoder by Paul Stoffregen V1.4.1
LED-Matrix	Anzeigematrix	Extern einzubinden[8] (Zip-Datei)
	Textausgabe	Adafruit GFX V1.1.5

[8] learn.adafruit.com/adafruit-15x7-7x15-charlieplex-led-matrix-charliewing-featherwing/

1.2 Blaupausen für eigene Projekte

Nach Einrichtung der Entwicklungsumgebung können wir nun mit der Umsetzung der Beispielprojekte beginnen. Wir starten mit einem einfachen „**Hallo Arduino**“ um uns mit der Bedienung der Entwicklungsplattform vertraut zu machen. Anschließend werden verschiedene Anwendungsszenarien erläutert:

- Im Projekt „**digitale Flaschenpost**“ senden wir eine Nachricht an unseren Flaschenpost–Server – hierdurch lernen wir die Anbindung an das Internet, nutzen eine WLAN Verbindung und eine einfache HTML Seite zur Eingabe der Nachricht, die Ausgabe der Nachricht nutzt eine Matrixanzeige.

  Ziel: Wir lernen eine Kommunikationsstrecke aufzubauen, welche wir für unser späteres Forschungsschiff benötigen!

  Mehr: Sicherheit ist für IoT wichtig, am Hackathon schauen wir bei den anderen Teams vorbei, welche IP Adresse nutzen die? Können wir Nachrichten an deren Flasche senden?

- Im Projekt „**Forschungsschiff**“ senden wir nun Daten von Sensoren in das Internet und visualisieren diese. Dabei nutzen wir wieder eine WLAN-Verbindung und ein einfaches REST (Representational State Transfer) basiertes Protokoll.
  Ziel: Wir können eine erste Datenauswertung machen: Wie entwickelt sich die Luftfeuchtigkeit? Von was ist diese abhängig (z.B. Temperatur, Sonnenstand)?

  Mehr: Welche Sensoren können das Forschungsschiff ergänzen? (Tabelle am Ende des Kapitels), wo könnte das Forschungsschiff ausgesetzt werden? (Tipp: es muss nicht im Wasser unterwegs sein: Auto, Tasche, Fahrrad, …).

- Im Projekt „**Forschungsschiff informiert sich**“ holen wir Wetterinformationen für unseren Standort aus dem Internet.

  Ziel: Technische Systeme kommunizieren untereinander.

  Mehr: Welche weiteren nützlichen Informationen werden angeboten (Zugverspätungen, Staumeldungen, Aktienkurse). Welche Alternativen zum WLAN gibt es (UMTS, LoRa)?

- Im Projekt „**Forschungsschiff greift ein**“ schauen wir uns eine komplexere regelungstechnische Anwendung an. Wir nutzen Regeln (Wenn Das Dann Tue Dies) und binden ohne Programmierung unser Smartphone in die Anwendung ein.

  Ziel: Nun geht es um den Mehrwert der Idee, wie kann ich diese im Alltag nutzen, wie kann ich diese an meine Bedürfnisse anpassen (was soll passieren wenn die Luftfeuchtigkeit im Keller zu hoch ist?)

  Mehr: Wie kann ich weitere Dienste des Internets einbinden? (Twitter: wenn das Eichhörnchen eine Nuss aus der Futterstation entnommen hat).
- Im letzten Projekt „**Wearables**“ verwandeln wir die Flaschenpost in ein tragbares Kleidungsstück.

  Ziel: Mobile Verwendung.

Ihr habt sicher ganz viele tolle Ideen ... ran an die Arbeit!

Die folgende Tabelle gibt einen zusammenfassenden Überblick über unsere Blaupausen und zeigt die dazu verwendeten Sensoren und Aktoren. Erweiterungen sind dank Grove[9]-System problemlos möglich. Zusätzliche Sensoren finden sich in Tabellen 3 bis 5 am Ende des Kapitels.

[9] exp-tech.de/pdf/products/grove/einstieg-grove.pdf

Tab. 2: Welche Interfacekomponenten werden genutzt?

Blaupause	**Internet**	**LED**	**Dreh/ Drück**	**BME 280**	**LED-Matrix**	**Grove Relais** (Digital)	**Grove Sensor** (Analog)
11.2.1 Hallo IoT-Kit		X Neo	X				
11.2.2 Flaschenpost - Hallo Server	Server				X		
11.2.3 Forschungsschiff Hallo IoT	Client Thingspeak	Neo		X	(X)		
11.2.4. Forschungsschiff Hallo M2M	Client M2M-API yahoo	Neo			(X)		
11.2.5 Forschungsschiff Hallo Control	Client IFTTT	Neo				Pumpe	Feuchte
11.2.6. Wearables		Neo			(X)		Loudness

1.2.1 Hallo Arduino - Hallo IoT-Kit

Die Einführung einer neuen Programmierumgebung erfolgt üblicherweise anhand eines einfachen Beispiels: Das Kultprogramm „Hallo-Welt“, welches einen kleinen Begrüßungstext ausgibt. Mangels eigenem Bildschirm beschränkt sich die Begrüßung bei unserem IoT-Kit auf ein einfaches Blinken der integrierten Leuchtdiode (LED). Diese Diode ist bei unserem Board mit dem Beinchen (Pin) GPIO0 (General Purpose I/O) des ESP-8266 verbunden. Da eine Spannungsausgabe auf einem Pin natürlich die Umgebung (hier in Form der angeschlossenen LED) aktiv beeinflusst und ggf.

Schaden anrichten kann, muss dieser Pin erst explizit als Ausgang frei geschaltet werden (Zeile 3 des Programmes).

```
1:   #define PIN_LED 0 // LED ist an Pin 0 angeschlossen
2:   void setup(){     // Initialisierung, wird nur einmalig ausgeführt
3:     pinMode(PIN_LED,OUTPUT);    // GPIO0 ist Ausgang
4:   }
5:
6:   void loop() { // Kontinuierliche Wiederholung
7:     digitalWrite(PIN_LED,HIGH); // LED an
8:     delay(500);                 // 500 ms warten
9:     digitalWrite(PIN_LED,LOW);  // LED aus
10:    delay(500);                 // 500 ms warten
11:  }
```

Ein typisches Arduino-Programm (auch Sketch genannt) besteht aus zwei Funktionen. Im Unterprogramm setup() (Zeilen 2-4) werden alle Initialisierungsaufgaben einmalig ausgeführt. Alle im Unterprogramm loop() (Zeilen 6-11) stehenden Befehle werden dagegen zyklisch wiederholt. Der Programmblock eines Unterprogrammes wird immer mit geschweiften Klammern {} umschlossen. Kommentare werden mit \\ eingeleitet. Die in der ersten Spalte angegebenen Programmzeilen dienen nur der vereinfachten Referenzierung im Beschreibungstext und sind nicht Bestandteil des eigentlichen Programmcodes.

In unserem Beispiel erfolgt die abwechselnde Ausgabe eines hohen Spanungspegels (HIGH, 3.3 V, Zeile 7) und eines niedrigen Spannungspegels (LOW, 0V, Zeile 9) an dem mit dem Symbol PIN_LED (Zeile 1) benannten Pin.

Unser Mikrocontroller ist in der Lage, diese Befehle mehrere tausend Male in der Sekunde zu wiederholen. So schnell kann das Auge nicht folgen, weshalb unser Programm künstlich durch Wartezeiten (delay) verlangsamt wird (Zeilen 8 und 10).

Zur Ausführung starten wir die Arduino Entwicklungsumgebung am PC und geben das obige Programm ein. Oder besser wir laden es über „Datei“ -> „Öffnen vom Datenträger“ bzw. auf dem Hackathon auch über „Datei“->„Beispiele“->“IoT_Hackathon“.

Alle vorgestellten Listings finden sich auch online unter http://div-konferenz.de/events/hackathon.

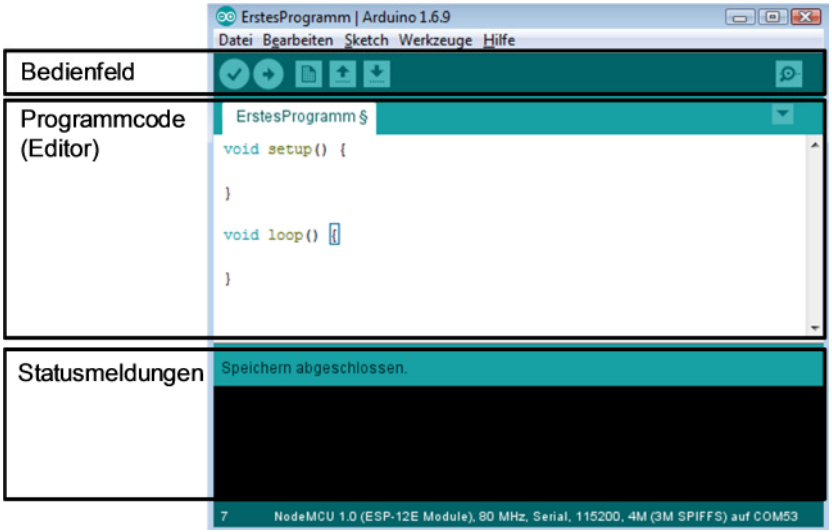


Abb. 3: Elemente der Arduino - IDE

Der Editor unterstützt den Programmierer durch verschiedene farbliche Hervorhebung von Befehlen und Kommentaren.

Im Bedienfeld besteht die Möglichkeit, das Programm zu speichern  , in Maschinecode zu übersetzen (überprüfen)  und schließlich auf die Zielhardware zu übertragen (hochladen, upload)  . Um zur Laufzeit des Programmes interaktive Nutzereingriffe zu ermöglichen, besitzt die Entwicklungsumgebung ein eingebautes Terminal-Programm (hier als Serial Monitor bezeichnet, Ausgaben im Programm mit Serial.print())  .

Etwaige Syntax-Fehler, Warnungen oder Probleme beim Hochladen werden im Statusbereich angezeigt.

Zur Ausführung übertragen wir unser Programm mittels Hochladen  an die Zielhardware. Nach kurzem Uploadprozess befindet sich das Programm im Hauptspeicher des IoT-Kits und wird sofort ausgeführt – die LED blinkt.

> Hurra, wir haben unser IoT-Kit erfolgreich programmiert!

Übrigens: Ein heruntergeladenes Programm ist auch nach Ausschalten und Wiederinbetriebnahme des Kits vorhanden. Erst ein erneutes Hochladen ändert die Programmausführung.

Jetzt wollen wir unser Experimentierboard erforschen. Dazu dient das folgende Beispiel „**Hallo Arduino IoT-Kit**“:

```
1:   #define HELL  63  // Textkonstante für Helligkeit der Neopixel
2:   #define BUTTON 2  // Drück-Funktion des Tasters
3:
4:   void setup() {    // Initialisierung
5:     kitInit();      // Ressourcen IoT-Kit initialisieren
6:   }
7:
8:   void loop() {     // Kontinuierliche Wiederholung
9:
10:    // Drehknopf abfragen
11:    int  wert = drehRead();    // Encoder lesen
12:    Serial.println(wert);      // Testausgabe aktuelle Position
13:
14:    // Anzeige der Drehstellung (Angabe Grenzen für gelb-Bereich)
15:    anzeigeAmpelSteigend(wert, 100, 200);    // am linken Pixel
16:
17:    // Druckknopf abfragen
18:    if (digitalRead(BUTTON)==LOW)            // Druckknopf abfragen
19:      anzeigePixelLinks(HELL,HELL,HELL);     // alle RGB leuchten
20:  }

21:  // Unterprogramm Ampel mit Neopixel-Signalisierung
22:  //  wert im Bereich von...bis -> Ampel zeigt gelb
23:  //  wert darüber -> Ampel rot, wert darunter -> Ampel zeigt grün
24:  void anzeigeAmpelSteigend(float wert, float von, float bis) {
25:    if (wert <= von)
26:       anzeigePixelLinks(0,HELL,0);          // grün leuchten
27:    else if (wert <= bis)
28:               anzeigePixelLinks(HELL,HELL,0);// gelb leuchten
29:         else anzeigePixelLinks(HELL,0,0);   // rot leuchten
30:  }
31:
32:  // Unterprogramm Initialisierung
33:  kitInit(){
34:   Serial.begin(115200);
35:  }
```

Das Kit besitzt einen Dreh/Dückknopf und zwei RGB-NeoPixel (farbige Leuchtdioden). Die **NeoPixel** können wir über einen Unterprogrammaufruf anzeigePixelLinks() in der Farbe und Intensität verändern. Jede Farbe wird durch die Anteile der Grundfarben rot / grün / blau beschrieben. Unsere Anzeigefunktion besitzt also drei Parameter für die Leuchtkraft der jeweiligen Grundfarbe (0…63). Die maximale Helligkeit ist auf einen Farbwert von 63 beschränkt, da das NeoPixel sonst unserer Taschenlampe Konkurrenz macht und eventuell die Augen blendet. **Also Vorsicht – nur unsere Anzeigefunktion zur Ansteuerung verwenden!**

Der **Dreh/Drückknopf** ist mit drei digitalen Eingangspins verbunden. Ein Pin (BUTTON, GPIO2) liegt immer am High-Pegel, es sei denn, wir drücken auf den Knopf. Dann schaltet dieser Pin gegen Masse (Low). Abfragen lässt sich der aktuelle Signalzustand einfach über einen digitalRead()-Befehl (Zeile 18).

Komplizierter ist die Sache mit der Drehfunktion. Der Knopf liefert bei Drehung zwei phasenverschobene Rechtecksignale, die im Interrupt ausgewertet werden (**Rotary-Encoder**). Das Interruptkonzept würde aber den Rahmen unserer kleinen Einführung sprengen – das heben wir uns besser fürs Studium auf. Nur so viel: Die Drehposition können wir durch drehRead() ermitteln und in der Variable wert auswerten: rechtsherum inkrementiert, linksherum dekrementiert den Wert.

Alles klar – was macht unser obiges Testprogramm?

Richtig, es signalisiert die Position des Drehknopfes durch die Ampelfarben. Und bei gedrücktem Knopf leuchtet das linke Pixel als Taschenlampe in weißer Farbe (alle Grundfarben maximale Helligkeit).

> Hurra, die Ein-/Ausgabemöglichkeiten sind klar!

Übrigens: Mit dem Rotary-Encoder lassen sich z.B. die Drehgeschwindigkeiten von Motoren ermitteln, oder die Position eines drehbaren Roboterarmes bestimmen.

Sicher ist aufgefallen, dass wir auf einmal zwei Reiter im Arduino-Editor haben. Die Datei IoT_Kit_Ressourcen.ino enthält zukünftig alle spezifischen Unterprogramme zur Ansteuerung der Interfacetechnik. Wir gliedern die Programme also in zwei Teile, der eigentlichen Anwendung und dem IoT-Kit-Interface.
Diese Modularisierung verschafft uns mehr Übersicht.
Wir wollen uns auf die Anwendung konzentrieren und die Hilfsfunktionen (wie z.B. die Ampel oder die Pixelansteuerung) einfach nur anwenden.

1.2.2 Die digitale Flaschenpost – Hallo Server

Natürlich lassen sich auch mehrere LEDs zu einer Matrix zusammenschalten. Bei unserem Kit geschieht dies durch Nutzung eines externen Erweiterungsboards: Das CharlieWing Matrix-Display[10]. Dieses wird auf die FeatherWing Steckleisten unseres Kits gesetzt und dient der erweiterten Visualisierung.

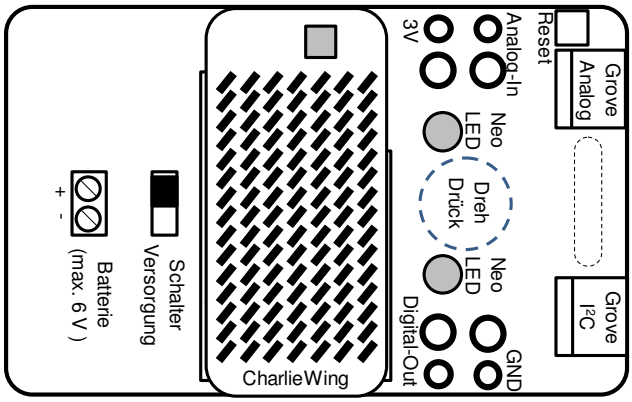


Abb. 4: IoT-Kit mit LED-Matrix

Durch Verwendung der entsprechenden Bibliotheksfunktionen gelingt so auch die Textausgabe des „Hallo IoT-Kit“ mit wenigen Programmzeilen:

```
36:  String ausgabetext = "Hallo IoT-Kit"; // text als globale Variable
37:
38:  void setup() {
39:    kitInit();                          // Kit initialisieren
40:  }
41:
42:  void loop() {
43:    matrixAnzeige(ausgabetext);         // Text anzeigen
44:  }
```

Natürlich ist die Ausgabe eines festen Textes nach einiger Zeit recht langweilig und wir wünschen uns mehr Flexibilität:

Da wir über ein internetfähiges Kit verfügen wäre es doch toll, wenn wir den auszugebenden Text per Web-Interface vom Handy aus ändern könnten. Dazu werden wir mit wenigen

[10] learn.adafruit.com/adafruit-15x7-7x15-charlieplex-led-matrix-charliewing-featherwing

Programmzeilen einen eigenen HTTP – Server implementieren und per Internet-Browser darauf zugreifen.

Als erstes müssen wir dazu eine Verbindung ins Internet aufbauen. Dies geschieht über das WLAN-Interface des ESP-8266. Ruck, zuck binden wir die entsprechenden Bibliotheken ein und definieren die WLAN Zugangsdaten.

```
1:   #include <ESP8266WiFi.h>           // Internet-
2:   #include <WiFiClient.h>            // Bibliotheken
3:   #include <ESP8266WebServer.h>
4:   ESP8266WebServer server(80);
5:
6:   // Zugangsdaten zum WLAN-Accesspoint
7:   #define AP_SSID "IoTHackathon"     // hier eigenes Netz
8:   #define AP_PASS "ITGipfel"         // und Passwort angeben
```

Bei der Initialisierung im setup() wird die Verbindung zum WLAN-Accesspunkt aufgebaut und unser IoT-Kit bekommt eine eigene IP-Adresse zugewiesen. Unter dieser Adresse sind wir ab sofort im WLAN-Netz erreichbar.

```
1:   // Anmeldung am WLAN
2:   WiFi.begin(AP_SSID, AP_PASS);           // Anmelden am WLAN
3:   anzeigePixelRechts(63,0,0);             // rechter Pixel: WLAN rot
4:   while (WiFi.status() != WL_CONNECTED) { // bis Verbindung steht
5:       delay(500); Serial.print(".");
6:   }                                       // dann
7:   anzeigePixelRechts(0,0,63);             // WLAN blau ok
8:
9:   ausgabetext = " Meine IP:" + WiFi.localIP().toString();
10:  Serial.println (ausgabetext);           // Zugangsdaten ausgeben
11:
12:  server.on("/", serverHomepage);         // Homepage definieren
13:  server.begin();                         // Server starten
```

Diese wichtige Adressinformation speichern wir im Ausgabetext für unsere Flaschenpostnachricht und zeigen sie natürlich sofort an. Können wir (oder andere) doch unter dieser vierstelligen Adresse (z.B. 192.168.2.11) unsere Homepage erreichen.
Eine Homepage besteht normalerweise aus einer Textdatei in einem speziellen Format (Hypertext Markup Language, HTML).

Da unser System keine Dateien verwalten kann, erstellen wir stattdessen einfach ein Textfeld mit dem gewünschten Inhalt:

```
1:   const char INDEX_HTML[] =
2:   "<html>"
3:     "<head>"
4:        "<title>ESP8266 Web Form Demo</title>"
5:      "</head>"
6:      "<body>"
7:        "<h1>Digitale Flaschenpost IoT-Hackathon</h1>"
8:        "<FORM action=\"/\" method=\"post\">"
9:          "<P>"
10:           "Deine Nachricht<br>"
11:           "<INPUT type=\"text\" name=\"message\"><br>"
12:           "<INPUT type=\"submit\" value=\"Send\">"
13:       "</P>"
14:       "</FORM>"
15:    "</body>"
16:  "</html>";
```

Und da wir nur eine einzige Webseite realisieren, liefern wir bei jeder Anfrage an den Server diesen Text als Antwort aus. Das dazu notwendige Unterprogramm ist schnell erstellt:

```
1:   void serverHomepage() {
2:    if (server.hasArg("message")) {        // Ggf. eine Nachricht
3:       ausgabetext = server.arg("message");// vom Client einlesen
4:    }                                      // aber immer
5:    server.send(200, "text/html", INDEX_HTML); // Homepage ausliefern
6:   }
```

Unsere HTML-Seite enthält mit dem Input-Tag ein im Browser des Clients editierbares Textfeld. Der Inhalt dieses Feldes wird über den Send-Button an unseren Web-Server übermittelt.
Über diesen Mechanismus aktualisieren wir unseren anzuzeigenden Text.

Stolz präsentieren wir die erste eigene Homepage:

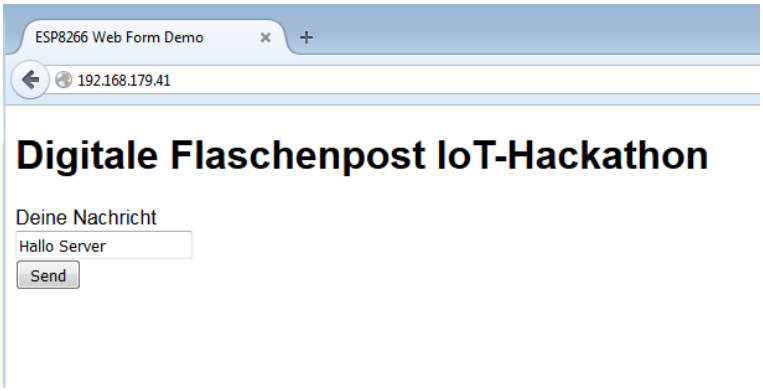


Abb. 5: Homepage unserer Flaschenpost

In einer Flasche verpackt, kann das Kit so eine digitale Nachricht an den Empfänger senden – ähnlich wie eine Flaschenpost in unserer analogen Welt.

> Hurra, wir haben den ersten eigenen Web-Server erstellt!

Übrigens: Es gibt keine Sicherheitsfunktionen. Wer unsere IP-Adresse kennt, der kann uns so eine Nachricht schicken!
(Tipp: Wir können die Flaschenpost z.B. mit einem geheimen Code vor dem eigentlichen Text schützen!)

1.2.3 Unser Forschungsschiff – Hallo IoT

Im folgenden Abschnitt wollen wir unsere Flaschenpost zusätzlich über verschiedene Sensoren direkt mit der Außenwelt verbinden (Datenlogger) und als Forschungsschiff auf Reisen schicken. Einfaches Beispiel dafür ist der eingebaute Bosch-Sensor BME280[11]. Dieser ist in der Lage, die Temperatur, die Luftfeuchte und den Luftdruck der Umwelt zu bestimmen. Über eine digitale Schnittstelle, den sogenannten I²C-Bus (Inter IC-Bus)[12], lassen sich die Messwerte vom Mikroprozessor auslesen und anzeigen:

```
1:   void loop() {         // Kontinuierliche Wiederholung der Messung
2:    float T = temperaturRead();         // Celsius
3:    float p = luftdruckRead()/100.;     // hPa
4:    float F = luftfeuchteRead();        // relative Feuchte %
5:
6:    // Temperaturüberwachung mit Neopixel-Signalisierung
7:    anzeigeAmpelSteigend(T, 20.0,30.0); // Grenzen von … bis für Gelb
8:
9:    // Ausgabe der Werte
10:   matrixAnzeige("T="+String(T)+"C");  // Temperaturanzeige
11:   matrixAnzeige("F="+String(F)+"%");  // Feuchteanzeige
12:   matrixAnzeige("p="+String(p)+"hPa");// Druckanzeige
13:
14:   // Hier erfolgt gleich die Datenspeicherung in der IoT-Cloud
15:  }
```

Zu einem Datenlogger gehört natürlich neben der eigentlichen Messdatenerfassung noch mehr, nämlich die Speicherung der gemessenen Werte. Üblicherweise geschieht das auf einem USB-Stick oder einer SD-Karte. Da wir ein internetfähiges Kit besitzen, können wir glücklicherweise auf diese externen Speicher verzichten und unsere Daten direkt dem Internet anvertrauen. In diesem Fall spricht der Informatiker von einer Cloud-Lösung, bei der ein im Internet verfügbarer Server unsere Daten entgegennimmt und für uns archiviert. Hierzu existieren eine Reihe von Diensten verschiedener Anbieter[13].

---

[11] https://www.bosch-sensortec.com/bst/products/all_products/bmp280

[12] http://www.elektronik-magazin.de/page/der-i2c-bus-was-ist-das-21

[13] phant.io, thingspeak.com

Im Rahmen des Hackathon verwenden wir die Open-Source Plattform Thingspeak. Zur Nutzung dieses Dienstes müssen wir uns vorher beim Server registrieren. Der offizielle Server ist weltweit unter thingspeak.com erreichbar. Alternativ dazu gibt es eine am Umwelt-Campus gehostete lokale Installation unter der URL www.thingspeak.umwelt-campus.de. Diese ist nur im Campus-Netz erreichbar, alle unsere Daten bleiben lokal.

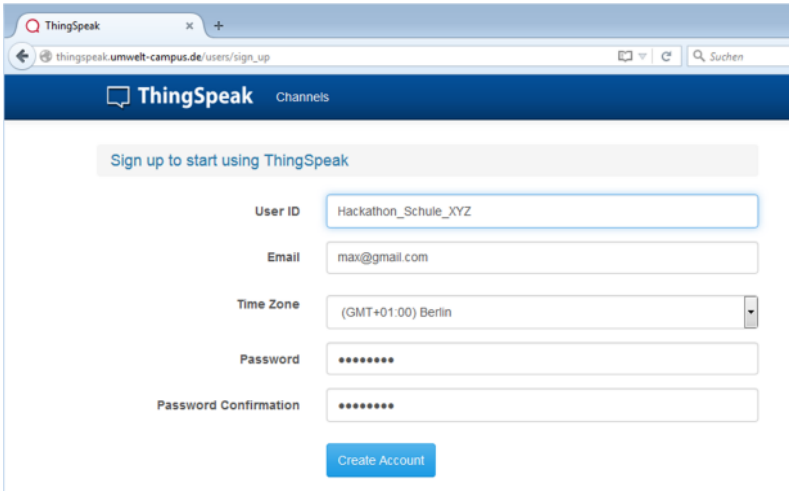


Abb. 6: Registrierung Thingspeak-Cloud

Anschließend können wir unter „Channels“ -> „New Channel“ den ersten Messkanal für den Hackathon konfigurieren. Dort erstellen wir drei Feldeinträge für Temperatur, Feuchte und Luftdruck. Wichtig ist die Reihenfolge, denn ein Feld wird bei der späteren Datenspeicherung durch den Feldindex adressiert. Wer möchte, kann die Messdaten später auch für andere Nutzer sichtbar machen (Make Public-Flag).

Nach Speichern der Konfiguration steht der Kanal ab sofort zum Zugriff bereit.

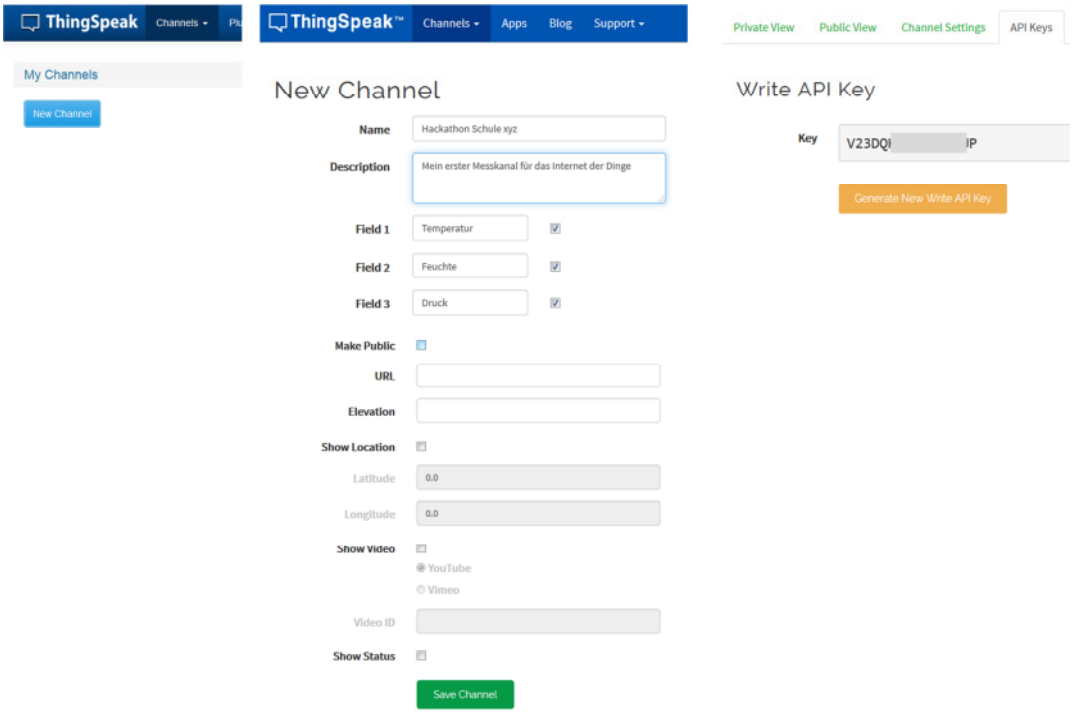


Abb. 7: Erstellung eines Messkanals

Bevor wir aber Daten dort ablegen können, benötigen wir eine Zugangsberechtigung, den sogenannten **API-Key**. Nur wer in Besitz dieses Schlüssels ist, kann Daten schreiben. Sicherheit spielt beim IoT eine große Rolle. Wir notieren uns also unseren Key und setzen ihn später an geeigneter Stelle im Programm ein (oder kopieren ihn mit cut & paste aus der Browseranzeige).

Nach erfolgreicher Konfiguration können die Daten mittels einfacher HTTP- GET Nachricht in der Datenbank des Servers abgelegt werden. Zum Test lässt sich diese GET-Nachricht von jedem Webbrowser aus generieren. Dazu im Browser die URL

```
http://api.thingspeak.com/update?api_key=V23DQyyyyUP&field1=34.5
```

aufrufen, wobei der API-Key an den eigenen Kanal angepasst werden muss. Der Server (host) übernimmt die Daten für die Temperatur (hier 34.5 für das erste Feld) und antwortet mit der laufenden Nummer des aktuellen Eintrags in seiner Datenbank.

Eine solche GET Nachricht läßt sich natürlich auch einfach durch unser IoT-Kit absenden:

```
1:   int httpGET(String host, String cmd, String &antwort) {
2:      WiFiClient client;             // Erstelle Client
3:      String text = "GET http://" + host + cmd + " HTTP/1.1\r\n";
4:             text = text + "Host:" + host + "\r\n";
5:             text = text + "Connection: close\r\n\r\n";
6:      int ok = client.connect(host.c_str(),80); // Connect, PORT 80
7:      if (ok) {
8:        client.print(text);          // Kommando absetzen
9:        Serial.print(text);          // Kommando absetzen
10:       for (int tout=1000;tout>0 && client.available()==0; tout--)
11:        delay(10);                  // warte max 10 s auf Antwort
12:       if (client.available() > 0)  // Antwort da
13:         while (client.available()) // Antwort des Servers auslesen
14:           antwort += client.readStringUntil('\r'); // bis cr
15:         else ok = 0;               // keine Antwort, timeout fehler
16:       client.stop();               // Kommunikation beenden
17:     }
18:     return ok;
19:  }
```

Nach erfolgreichem Verbindungsaufbau (Zeile 6) wird die Nachricht gesendet (Zeile 8) und im Timeout auf die vom Server zurückgelieferte Antwort gewartet (Zeile 10-12). Die aufrufende Funktion stellt einen Platzhalter (antwort) für die generierte Antwort bereit und bekommt diesen von der httpGET() Funktion gefüllt. Unsere loop()-Funktion erweitert sich also nur um wenige Zeilen:

```
1:   String antwort = " ";                // Platzhalter für Antwort
2:   String host = "api.thingspeak.com";  // host thingspeak.com
3:   String cmd = "/update?api_key=V23DQyyyyyUP";//API-Key (ändern)
4:   cmd=cmd+"&field1="+String(T)+"&field2="+String(F)+
     "&field3="+String(p)+"\r\n";
5:   httpGET(host,cmd,antwort);          // und abschicken
6:   delay(10000);                       // warte 10 s, dann nochmal
```

Zur individuellen Festlegung des Messkanals muss natürlich der API-Key in Zeile 4 angepasst werden.

Um den Code nicht unnötig zu verkomplizieren, verzichten wir auf die Auswertung der Antwort des Thingspeak-Servers und gehen davon aus, dass der Eintrag bei einem erfolgreichen Verbindungsaufbau auch ordnungsgemäß in der Datenbank abgelegt wurde.

Kontollieren können wir den Erfolg auf der Thingspeak-Seite im Internet (oder im Falle von thingspeak.umwelt-campus.de nur im Intranet):

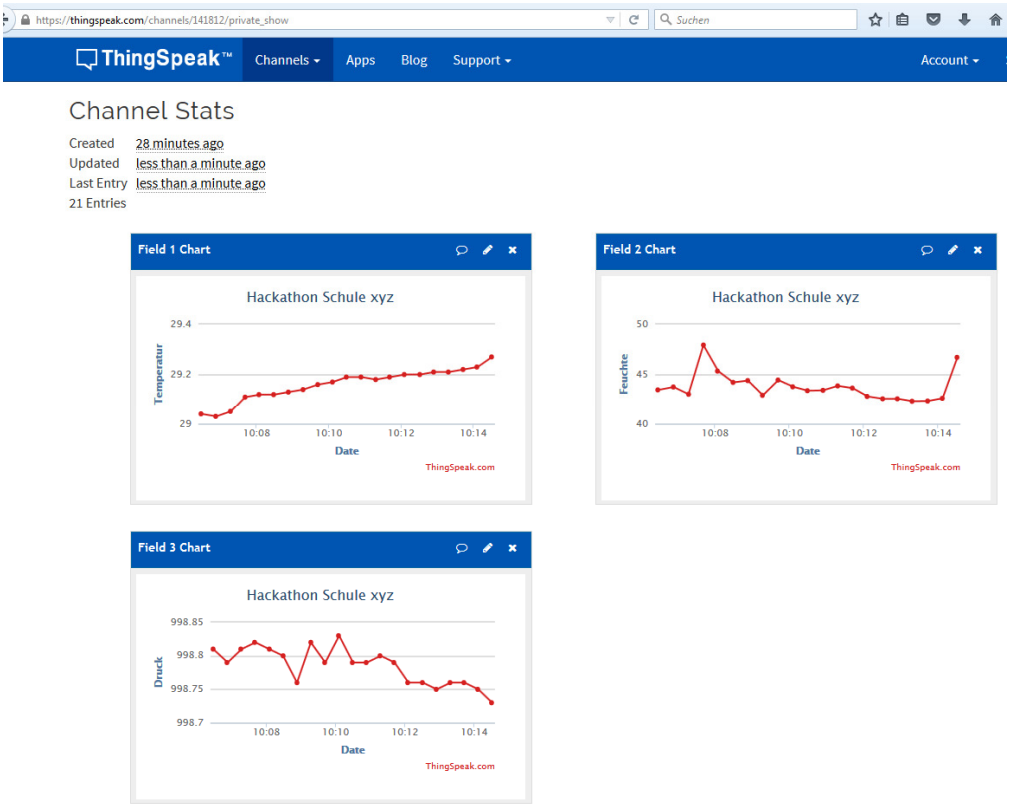


Abb. 8: Visualisierung eines Messkanals

Dort sind unsere Messwerte ab sofort weltweit verfügbar. Damit haben wir mit wenig Programmcode einen internetfähigen Datenlogger für Umweltinformationen erstellt.

> Hurra, wir haben unsere Flasche in ein Forschungsschiff mit IoT-Anbindung verwandelt!

Übrigens: Die Sensorik lässt sich ganz einfach über den Grove-Connector erweitern. Denkbar sind alle in einer Tabelle am Ende des Kapitels aufgeführten Sensoren.
Wir sollten auch den Energiebedarf im Auge haben (mehr dazu unter Maker & Hacker am Ende des Kapitels).

1.2.4 Forschungsschiff informiert sich – Hallo M2M

Natürlich kann ein IoT-Device auch selbst Informationen aus dem Internet abfragen. Dazu definieren die Diensterbringer spezifische Schnittstellen (API, Application Programming Interface), über die die Informationen bereitgestellt werden können. Das Prinzip ist ähnlich, wie beim Aufruf normaler Webseiten, nur die Antwort erfolgt in einem maschinenlesbaren Format (z.B. XML, Extensible Markup Language). Als Beispiel verwenden wir die Wettervorhersage von yahoo[14].
Mit Hilfe des bereits bekannten HTTP-GET können die aktuellen Wetterbedingungen für beliebige geografische Positionen abgerufen werden. Der Ort wird dabei über eine Identifikationsnummer (woeid) festgelegt[15]. Das folgende Programm ermittelt die aktuelle Windgeschwindigkeit für Saarbücken (woeid 690631, ggf. anpassen):

```
7:   void loop() { // Endlosschleife Wetterabfrage, hier Saarbrücken)
8:    String cmd = "/v1/public/yql?q=select%20wind%20from%20weather.
9:                 forecast%20where%20woeid=690631\r\n";
10:   String host = "query.yahooapis.com";
11:   String antwort;  // Platzhalter für Antwort)
12:
13:   // http-Get absetzen
14:   httpGET(host,cmd,antwort); // Anfrage senden, Antwort holen
15:
16:   // Antwort auswerten
17:   float s=parseSensorInfo(antwort,"speed"); // speed extrahieren
18:
19:   // Windüberwachung mit Neopixel-Signalisierung und Anzeige
20:   anzeigeAmpelSteigend(s, 12.0,61.0);
21:   matrixAnzeige("speed="+String(s*1.61)+"km/h");
22:
23:   delay(5000);                    // Warte bis zur nächsten Abfrage
24:  }
```

[14] https://developer.yahoo.com/weather/

[15] http://woeid.rosselliot.co.nz/

Die Funktion parseSensorInfo() wertet die maschinenlesbare Antwort aus und gibt die gesuchten Informationen für die Geschwindigkeit in Meilen pro Stunde (mph) zurück. Die Ausgabe erfolgt nach Umrechnung in km/h.

Hurra, wir können die Wettervorhersage nutzen!

Übrigens: Wenn wir die Windgeschwindigkeit mit eine Fallunterscheidung (if … then … else) überprüfen, können wir ab einer bestimmten Windgeschwindigkeit einen Alarm auslösen (z.B. durch Setzen eines digitalen Ausgangs ähnlich wie bei der blinkenden LED). Damit lässt sich eine einfache Überwachung z.B. einer Markise realisieren.

1.2.5 Forschungsschiff greift ein - Hallo Control

Die bedingte Ausführung von Programmteilen lässt sich prima dazu nutzen, um per Rückkopplung in ein System einzugreifen. Der Ingenieur spricht dann von einer Regelung. Als einfaches Beispiel bauen wir uns gleich eine automatische Bewässerung für die Blumentöpfe in der Fensterbank. Dazu benötigen wir einen Sensor, welcher die Leitfähigkeit der Erde ermittelt und als Maß für die Feuchtigkeit genutzt werden kann. Ist die Erde zu trocken, steuern wir eine Pumpe an, welche Wasser in den Topf befördert.

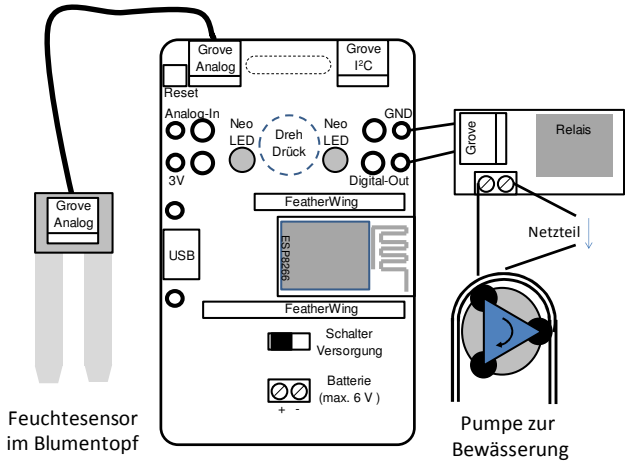


Abb. 9: Das Forschungsschiff als Bewässerungsregler

Mit unserer bisher gesammelten Erfahrung ist das programmtechnisch einfach umzusetzen:

```
1:   #define PIN_PUMPE 0    // Pumpe am GPIO0 angeschlossen
2:   int eventScharf = 1;   // nur eine Meldung an: If this then that
3:
4:   void setup(void) {     // Initialisierung
5:     kitInit();                 // IoT-Kit initialisieren
6:     pinMode(PIN_PUMPE,OUTPUT); // Pumpe als Ausgang ansteuern
7:   }
8:
9:   void loop() {          // Endlosschleife Regelung
10:    #define TROCKEN 100.0      // darunter wird gegossen
11:    #define ALARM   10.0       // kein Wasser im Tank ?
12:
13:    int wert = analogRead(A0);// Sensor einlesen
14:
15:    // Ampel und Matrixanzeige versorgen
16:    anzeigeAmpelFallend(wert,ALARM,TROCKEN);
17:    matrixAnzeige("Feuchte =" + String(wert)); // Text ausgeben
```

```
18:     // Regelkreis
19:     if (wert < TROCKEN) {          // Erde ist zu trocken
20:       digitalWrite(PIN_PUMPE,HIGH);// Pumpe anschalten
21:       delay(1000);                 // 5 Sekunden gießen
22:       digitalWrite(PIN_PUMPE,LOW); // Pumpe ausschalten
23:     }
24:
25:     // Alarmüberwachung: zusätzlich Meldung an IFTTT
26:     if ((eventScharf == 1) && (wert < ALARM)) {
27:       String host = "maker.ifttt.com";
28:       String cmd = "/trigger/Blume/with/key/mein Key"; // anpassen
29:       String antwort;              // Platzhalter für Antwort
30:       httpGET(host,cmd,antwort);   // Anfrage senden, Antwort holen
31:       Serial.println(antwort);
32:       eventScharf = 0;             // Nur ein Trigger reicht
33:     }
34:     delay(1000);                   // nach kurzer Zeit wiederholen
35:   }
```

In Zeile 6 schalten wir den Ausgang für die Pumpe scharf. In unserem Fall haben wir das Relais zur Pumpenansteuerung mit diesem I/O-Pin verbunden. Leider kann der Mikrocontroller nur kleine Lasten wie z.B. eine LED schalten. Hier steuern wir den Kontakt des Relais an. Das Relais wiederum schaltet dann die Verbindung zwischen Pumpe und Netzteil. **Achtung: wir verwenden natürlich nur Kleinspannungen – niemals die 230V aus der Steckdose schalten!**

Innerhalb der Regelschleife lesen wir das vom Feuchtesensor[16] gelieferte Spannungssignal. Dieses wird vom Analog-Digital-Wandler (ADC) des ESP in einen Wert zwischen 0 und 1023 gewandelt. Ist der Wert kleiner als die Grenze TROCKEN, so wird die Pumpe für kurze Zeit angeschaltet. (Zeilen 18-22). Die Erde wird feucht, die Leitfähigkeit steigt, der Sensor liefert höhere Werte.

[16] http://seeedstudio.com/wiki/Grove_-_Moisture_Sensor

Kritisch wird die Anwendung dann, wenn unser Vorratsgefäß leer ist. Dann trocknet die Erde weiter aus und wir erreichen eine untere Alarmgrenze. In diesem Fall wollen wir wieder unsere IoT-Fähigkeiten ins Spiel bringen und eine Meldung über Twitter, E-Mail oder SMS generieren.

Dazu bietet sich der Internet-Dienst „If this then that“ (IFTTT)[17] an.

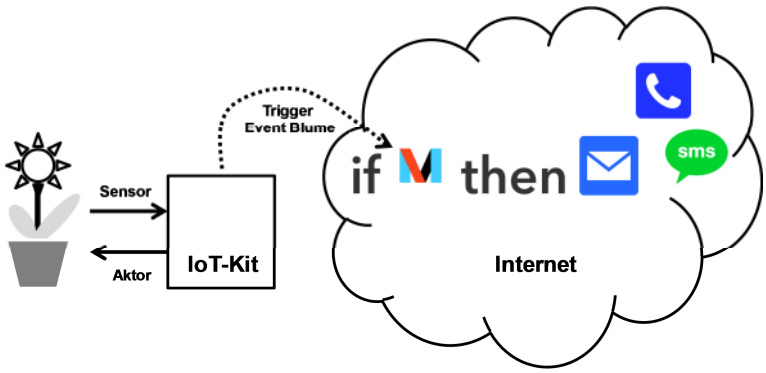


Abb. 10: Informationsfluss IFTTT

Auch hier müssen wir uns natürlich vorher registrieren:

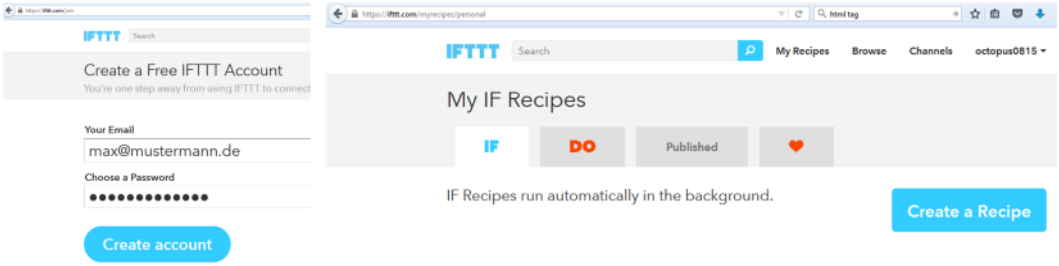


Abb. 11: Registrierung IFTTT

Mittels komfortabler Weboberfläche können wir anschließend mit einfachem Knopfdruck ein Rezept (Recipe) der Form IF …. then DO…. erstellen (d.h. konfigurieren, nicht programmieren). In unserem Fall ist die Triggerbedingung (IF) ein vom IoT-Kit erzeugter Event „Blume“ auf dem Makerkanal:

[17] https://ifttt.com

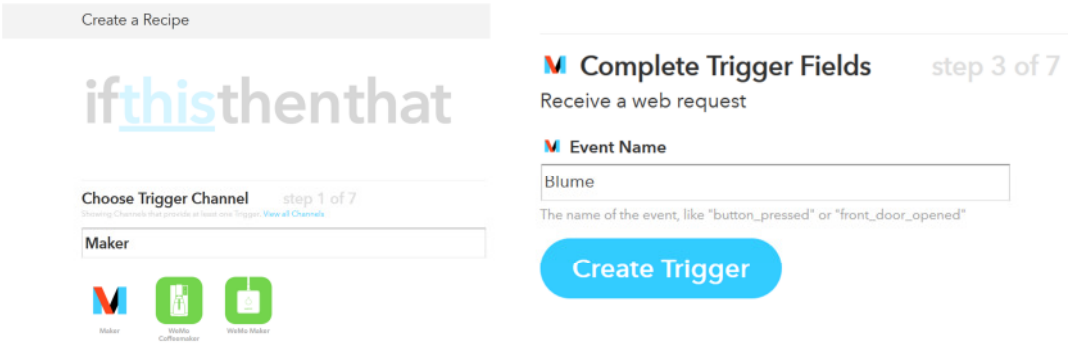


Abb. 12: Ereignis (Event) definieren

den Zugriffskey finden wir anschließend unter https://ifttt.com/maker und als Action lassen wir uns eine E-Mail schicken:

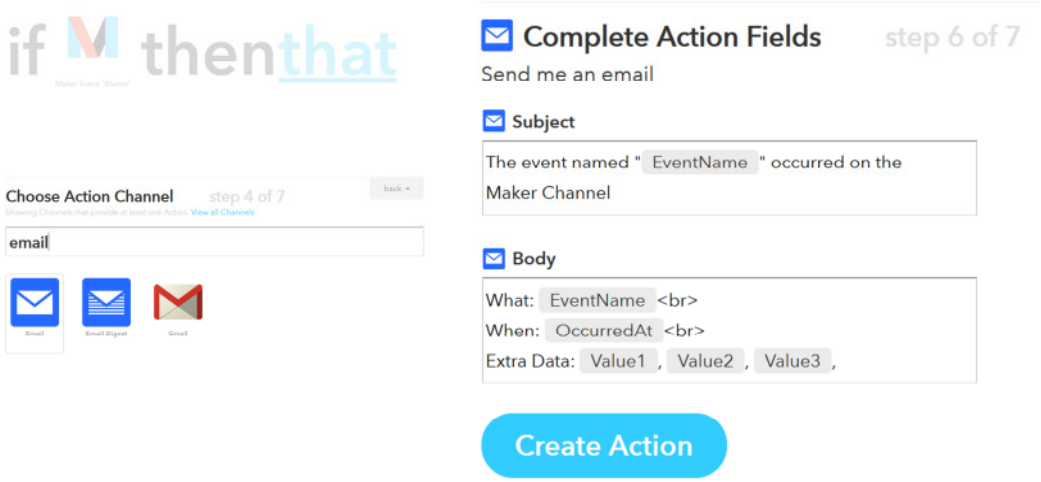


Abb. 13: Action Mailversand

Damit können wir uns jetzt im Urlaub beruhigt zurücklehnen. Die Blumen sind versorgt – und sollte mal was schiefgehen, erhalten wir eine mail und können die Nachbarn aktivieren.

> Hurra, wir haben eine Blumenbewässerung realisiert!

Übrigens: So ähnlich lassen sich viele Alltagsaufgaben überwachen. Zum Beispiel die Luftfeuchtigkeit im Keller durch Ansteuerung eines Lüfters, oder die Alarmanlage mit Überprüfung der Fensterkontakte und Ansteuerung einer Sirene.

1.2.6 Wearables – Forschungsschiff kleidet sich

Die einzeln ansteuerbaren bunten NeoPixel eröffnen viele neue Anwendungen. Als kleine Knöpfe lassen sie sich problemlos in Kleidungsstücke integrieren. Dazu gibt es leitfähiges Nähmaterial, mit denen die Anzeigeelemente geschickt befestigt und mit Energie versorgt werden können. Unser IoT-Kit ist aber an sich schon so kompakt, dass es problemlos am Körper befestigt werden kann. Mit einer mobilen Energieversorgung über Batterien können wir unser „Hallo Arduino"-Projekt sofort recyclen: Als Namenschild zur Kennzeichnung der Coaches am Hackathon. Fügen wir etwas Sensorik in Form eines Mikrofons hinzu, so haben wir eine mobile Lichtorgel. Je nach Umgebungslautstärke leuchten die Neopixel in einer anderen Farbe.

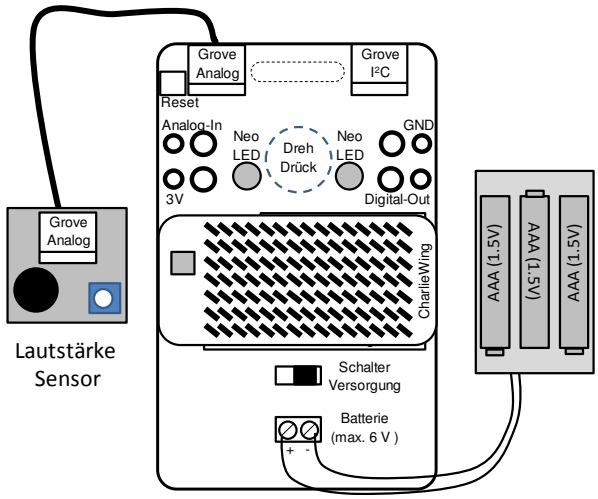


Abb. 14: Tragbares IoT-Kit mit Lärmsensor

Zur Nutzung der Batterie müssen wir den Wahlschalter zur Spannungsversorgung nach rechts stellen.

```
36:  void setup() {
37:    kitInit();   // initialisieren
38:  }
39:
40:  void loop() {  // lauschen und Ausgeben
41:     int wert = analogRead(A0);
42:     Serial.println(wert);
43:     anzeigeAmpelSteigend(wert, 100,200);
44:     matrixAnzeige("Coach");              // Text ausgeben
45:  }
```

Hurra, unser Kit ist wie ein Kleidungsstück tragbar!

Übrigens: Die Lichtorgel eignet sich auch prima als Lärmampel für die Schule ☺. Wir können die Funktion auch ohne Sensor testen. Einfach mit den Fingern die beiden Bananenbuchsen (ADC und 3 V) auf der linken Seite berühren. Über unseren Hautwiderstand gibt es eine leitende Verbindung zwischen 3 V und dem analogen Eingang. Dies Phänomen können wir zukünftig z.B. prima für einen Touch Sensor verwenden.

1.2.7 Die Flaschenpost wird flexibel – Hallo Interface

Die vorherigen Kapitel haben sicher zum Nachdenken über eigene Projektideen eingeladen. Zur Umsetzung fehlt jetzt noch die richtige Sensorik und Aktorik. Nachfolgende Tabellen zeigen eine Auswahl an zur Verwendung empfohlener Interfacetechnik:

Tab. 3: Erweiterungsmöglichkeiten Sensorik

Messgröße	Sensor	Informationen
Luftqualität (Erdbeeren bis Pups)	Air Quality Sensor 1.3	www.seeedstudio.com/wiki/Grove_-_Air_Quality_Sensor_v1.3
Beschleunigung Gravitation Lagebestimmung Vibrationen	9-axis Absolute Orientation Sensor	www.bosch-sensortec.com/bst/products/all_products/bno055
Leitfähigkeit (Erdfeuchte, Hautfeuchte, Wasserstand)	Feuchtig-keitssensor	seeedstudio.com/wiki/Grove_-_Moisture_Sensor
Magnetfeld	HMC5883L 3-Achsen Kompass	seeedstudio.com/wiki/Grove_-_3-Axis_Compass_V1.0
Lautstärke	Loudness Sensor	seeedstudio.com/wiki/Grove_-_Loudness_Sensor
Entfernung (5- 80 cm, Einparksensor)	Abstands-sensor (Sharp)	pololu.com/file/0J85/gp2y0a21yk0f.pdf
Elektr. Strom	Stromsensor (Wechsel-strom)	openenergymonitor.org/emon/buildingblocks/ct-sensors-interface
Elektr. Strom /Spannung	Stromsensor (Gleich-strom)	learn.adafruit.com/adafruit-ina219-current-sensor-breakout
UV-Licht (Sonnenbrand)	Grove - UV Sensor	seeedstudio.com/wiki/Grove_-_UV_Sensor
Bewegung	PIR Motion Sensor	seeedstudio.com/wiki/index.php?title=Twig_-_PIR_Motion_Sensor
Feinstaub	Dust Sensor	seeedstudio.com/wiki/Grove_-_Dust_sensor

Tab. 4: Erweiterungsmöglichkeiten Aktorik

Ausgabe	Aktor	Informationen
Elektr.Verbraucher Pumpe, Motor	Relais	seeedstudio.com/wiki/Grove_-_Relay
7x15 Matrix	Charlie-Wing	learn.adafruit.com/adafruit-15x7-7x15-charlieplex-led-matrix-charliewing-featherwing

Das IoT-Kit besitzt je eine analoge und digitale (I2C) Grove Schnittstelle. Sollen mehr Sensoren/Aktoren Verwendung finden, bieten sich folgender Komponenten an:

Tab. 5: Erweiterungen Schnittstellen

Komponenten	Einsatz	Informationen
Adafruit ADS1015	Analog I	learn.adafruit.com/adafruit-4-channel-adc-breakouts/overview
6 Output I/O Expander	Digital	learn.sparkfun.com/tutorials/sx1509-io-expander-breakout-hookup-guide
I2C-Hub	I2C	seeedstudio.com/depot/Grove-I2C-Hub-p-851.html
Sonstiges		
Mobilfunk	GSM	gemalto.com/deutschland/iot
Raspberry Pi	Server	raspberrypi.org/
Bosch Kit	Sensor	xdk.bosch-connectivity.com/

1.3 Für Hacker & Maker

Dieses Kapitel richtet sich an alle technisch vorgebildeten Interessierten, die tiefer in die Schaltung und die flexiblen Möglichkeiten unseres IoT-Kits einsteigen möchten.

1.3.1 Schaltplan

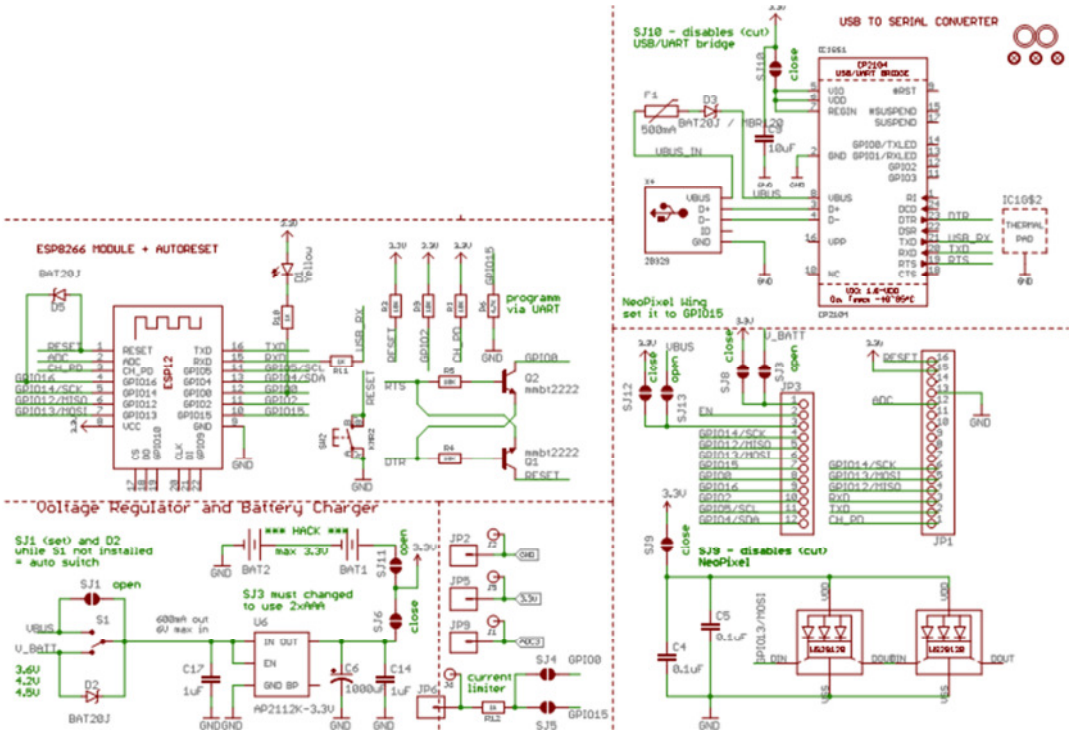


Abb. 15: Schaltplan

Unser Design soll zwei Zielgruppen adressieren: Zum einen die Programmieranfänger, die ein möglichst robustes Board benötigen und zum anderen die Spezialisten, die bei mobilen Anwendungen auch noch das letzte µA an Strom einsparen möchten. Hierzu existieren verschiedene Optionen:

1.3.1.1 Normalbetrieb (für Schüler dringend empfohlen)

Mit einem Wahlschalter (S1, unten links) kann zwischen der Versorgung über den **USB-Bus** oder einer Versorgung mittels Batterien gewählt werden. Dazu ist ein externes Batteriepack an die Schraubklemmen unterhalb des Wahlschalters anzuschließen. **Empfohlen: 3 Stück AAA Mikrozellen (Batterie oder NiMh Akkus)**. In beiden Fällen wird die anliegende Spannung über einen Spannungsregler (AP2112K) auf 3.3 Volt reguliert. Dieser stellt unabhängig vom Zustand der Batterie eine konstante Versorgungsspannung sicher. Der Schalter sorgt dafür, dass wir auch bei schwachen NiMh-Akkus noch genügend Spielraum für den Spannungsregler haben.

1.3.1.2 Mobiler Low Energy-Betrieb (für Spezialisten)

Der integrierte Spannungsregler (LDO) reduziert die Energieeffizienz der Schaltung (Ruhestrom 55 µA), zusätzlich kommt es zu einem Spannungsabfall und damit Leistungsverlust am LDO (DropOut Voltage 400 mV). Bei Verwendung von nur zwei Akkuzellen (2 x AAA) kann jedoch auf den Spannungsregler verzichtet werden. Hierzu existieren die Lötbrücken SJ6/SJ11, mit denen die Versorgungsspannung direkt auf zwei Zellen umgeschaltet werden kann.

Der ESP-8266 ist für den Betrieb zwischen 3 V und 3.6 V spezifiziert. Bei kleinerer Spannung (entladene Batterie) kann es zu unvorhergesehenen Betriebsverhalten kommen (deshalb hier als „hack" bezeichnet, für Anfänger nicht zu empfehlen). Dazu ist die Schraubklemme für die Batterien zu entfernen und stattdessen zwei Batteriehalter zu montieren.

Eine weitere Stromsparmöglichkeit bietet die Lötbrücke SJ10 zur Versorgung der USB-Bridge (CP2104). Diese hat auch im Ruhemodus (ohne USB-Verbindung) einen Strombedarf von 100 µA. Ein durchtrennen der Lötbrücke legt diesen Chip still, verhindert aber auch das Aufspielen neuer Programme per USB-Upload – also erst abschalten, wenn die Software ok ist!

Auch die beiden NeoPixel besitzen einen Ruhestrombedarf von ca. 800 µA. Wenn diese Pixel von der Anwendung nicht benötigt werden, kann die Lötbrücke SJ9 durchtrennt werden.

1.3.2 Low Energy Application

Nachdem wir so viele Energiesparoptionen kenngelernt haben, schauen wir ins Datenblatt[18] des ESP-8266 und stellen entsetzt fest, dass der mittlere Strombedarf im Normalbetrieb bei ca. 80 mA liegt. Was das für den mobilen Betrieb bedeutet, lässt sich einfach ausrechnen:

---

[18] http://www.esp8266.com/wiki/doku.php?id=esp8266_power_usage

Mikrozellen (AAA) mit ca. 800 mAh erlauben eine Betriebsdauer von maximal 800 mAh / 80 mA = 10 Stunden – in der Praxis eher weniger.

Betrachten wir das Programm unseres IoT-Forschungsschiffs, so stellen wir allerdings fest, dass wir die meiste Zeit im Wartemodus (delay) verbringen. Die eigentliche Datenerfassung und Thingspeak-Kommunikation ist dagegen schnell erledigt. Das ist typisch für Datalogger-Anwendungen.

Schauen wir also, welche Schlafmöglichkeiten der ESP bietet: Deep Sleep ist mit 10 µA angegeben. Selbst wenn sich der Bedarf in der Praxis eher bei 80 µA einpendelt, so bedeutet das eine Batterielebensdauer von (80mAh/ 80µA), d.h. 10000 Stunden (oder 416 Tagen).

Leider müssen wir zwischendurch zur Messung aufwachen, je nach Zykluszeit ergibt sich aber eine praktische Laufzeit von mehreren Tagen.

Um die Deep Sleep Fähigkeiten zu nutzen, können wir unseren Chip mit einem einzigen System-Befehl ESP.deepSleep(in µs) ins Träumen bringen.

Alle Mikrocontroller besitzen solche Energiesparmodi, die meisten wachen nach der Schlummerzeit auf und setzen die Programmausführung gleich an derselben Stelle fort. Nicht so unser ESP: Dieser führt nach Aufwachen einen Reset durch und beginnt erneut mit der Initialisierung. Deshalb müssen wir unsere ganzen Aktivitäten praktisch in die setup()-Routine verlagern.

```
1:   void setup() {  // Aufwachen
2:
3:   // BoschSensor: eine Messung initiieren, dann sleep (forced.mode)
4:
5:   //  Anmeldung möglichst schnell -> Zeit kostet Energie
6:   //  mit statischer IP-Adresse geht en viel schneller
7:   //  WiFi.config(IPAddress(192,168,179,40),IPAddress(192,168,179,1),
     //  IPAddress(255,255,255,0));            // statische IP
8:
9:   WiFi.begin(AP_SSID, AP_PASS);             // Anmelden am WLAN
10:    while (WiFi.status() != WL_CONNECTED) { // auf Verbindung
11:      delay(1);
12:  }
13:
14:  // Messwerte erfassen
```

```
15:  // Ausgabe an den Thingspeak Kanal – wie gehabt
16:
17:  // Jetzt Tiefschlaf bis zur nächsten Messung
18:   ESP.deepSleep(60000000, WAKE_RF_DEFAULT); // Sleep 1 Minute
19:  }
20:
21:  void loop(){};                              // vorher eingeschlafen
```

Je effizienter unser Programm wird, d.h. je weniger Zeit wir bis zum Erreichen des Sleep-Modus benötigen, desto mehr Energie können wir sparen. Ein wesentlicher Faktor ist die Anmeldung unseres IoT-Kits am WLAN-Accesspoint. Wie wir bei der Anmeldung am Fortschrittsbalken sehen können, dauert dies mehrere Sekunden – kostbare Zeit, die Batteriekapazität erfordert. Wer sich mit seinem Accesspoint auskennt, kann unserem Modul eine feste IP-Adresse zuweisen. Dann entfällt die Anmeldung über DHCP (Dynamic Host Configuration Protocol) und wir gewinnen ein paar wertvolle Sekunden. Hierzu den Kommentar in Zeile 5 entfernen und die statische IP eintragen.

Hurra, wir ermöglichen den längeren mobilen Einsatz!

Ein Problem des ESP-8266 sollte nicht verschwiegen: Im Deep-Sleep Modus werden auch die I/O-Leitungen nicht mit Energie versorgt. Die LED verlischt, die NeoPixel behalten ihre letzte Farbeinstellung – benötigen dafür aber zusätzliche Energie.

Tab. 6: Energiesparmaßnahmen und ihre Wirkung

ESP	USB	Neo	LDO	Strom	Bemerkung
activ	activ	sleep	active	100 mA	Versorgung über USB
activ	sleep	sleep	activ	74 mA	Batterie 3 x AAA
sleep	sleep	sleep	activ	1 mA	Batterie 3 x AAA
sleep	sleep	disable SJ9	activ	190 µA	Batterie 3 x AAA Neopixel abgeschaltet
sleep	disable SJ10	disable SJ9	activ	100 µA	Batterie 3 x AAA Neopixel und USB abgeschaltet
sleep	disable SJ10	disable SJ9	disable SJ11	**50 µA**	Batterie 2 x AAA („Hack“) Neopixel, USB, LDO abgeschaltet

Übrigens: Unsere Schaltung verfügt auch über die Möglichkeit eine Ladeschaltung für einen Lithium-Polymer Akkus nachzurüsten (nicht bestückt). Dann können wir das Board dauerhaft mit einer Solarzelle versorgen.

Nach Aufwachen aus dem Reset sind alle Inhalte des RAM-Speichers verloren. Als Programmierer stellt sich die Herausforderung, trotzdem einen sinnvollen Programmablauf zu realisieren. Das funktioniert nur deshalb, weil es einen kleinen nichtflüchtigen Speicherbereich innerhalb des Uhrenmoduls (RTC, Realtime Clock) gibt, den wir zur Zustandsspeicherung nutzen können. Eine Idee zur weiteren Verbrauchsreduzierung ist deshalb, die Ergebnisse der letzten Messung dort zu speichern. Nach Aufwachen überprüft das System, ob es eine signifikante Messwertänderung gegeben hat. Nur in diesem Fall wird das WLAN-Interface aktiviert und die neuen Messwerte in der Cloud gespeichert.