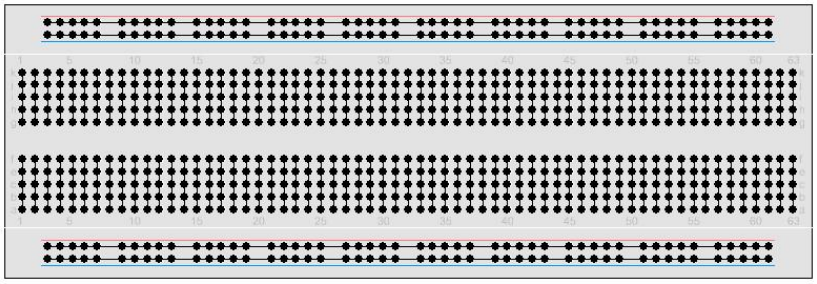
Kursprogramm Teil 1 (14 Stunden)

Erste Schritte

* Aufbau eines Breadboards, Anschluss des Heltec-Boards an den PC über USB
* Kennenlernen der Programmierumgebung IDE (Sketchbook, Board-Verwaltung, Bibliothek, serieller Port …)
* Aufbau eines Sketches (setup, loop)
* Ein erstes Programm: Blink mit boardinterner LED
* Eigenständige Modifikation der Blinkfrequenz (z.B. Morsezeichen, Leuchtfeuer)
* Blinkschaltung mit interner und externer LED

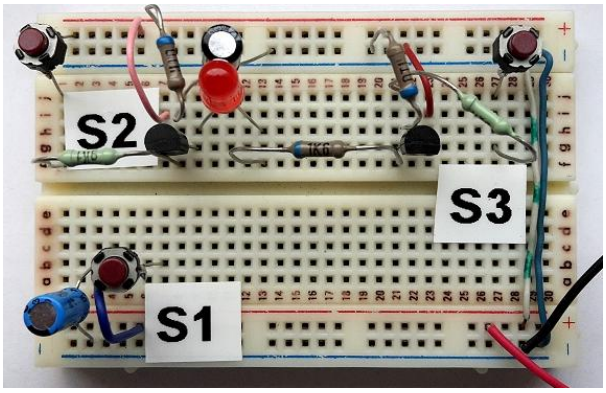
Arbeiten mit dem Steckboard (Breadboard)

Ein Breadboard (Steckplatine) ermöglicht es, schnell und einfach Schaltungen zu testen, ohne dass gelötet werden muss.
Unser Steckfeld mit insgesamt 830 Kontakten im 2,54-mm-Raster ist wie folgt aufgebaut:



Die oberen und unteren beiden Reihen sind waagerecht über die gesamte Breite des Boards miteinander verbunden. Sie sind für die Stromversorgung vorgesehen (rot = Plus, blau = Minus).
In den mittleren Reihen sind jeweils 5 Kontakte wie dargestellt miteinander verbunden.

Mit einem solchen Board kann man schnell eine Schaltung erstellen und ausprobieren:



Die Arduino IDE

Die **Arduino IDE (Integrated Development Environment)** ist die Entwicklungsumgebung, mit der du **Programme (Sketches)** für Arduino-Mikrocontroller schreiben, testen und auf das Board hochladen kannst.

Datei -> Einstellungen

- ⇨ Pfad für Sketchbook festlegen
- ⇨ Editorsprache Deutsch auswählen
- ⇨ Zusätzliche Boardverwalter-URLs: hier wird der Link des Board-Herstellers eingetragen

Datei -> Beispiele

- ⇨ fertige **Beispielprogramme** für häufige Anwendungen

Bearbeiten

- ⇨ Hier finden sich nützliche Funktionen, um den Programmcode zu bearbeiten

Werkzeuge

- ⇨ Auswahl des Boards
- ⇨ Auswahl des Ports
- ⇨ Upload Speed evtl. anpassen

Hilfe

- ⇨ Referenz zeigt eine Übersicht der Befehle

Code-Editor

- ⇨ Hier schreibst du deinen **Arduino-Code (Sketch)** in der Sprache **C/C++**.
- ⇨ Die IDE hebt Schlüsselwörter farbig hervor (Syntax-Highlighting) und nummeriert die Zeilen.

  Standardstruktur eines Sketches:

```
void setup() {
  // Wird einmal beim Start ausgeführt
}

void loop() {
  // Wird fortlaufend wiederholt
}
```

In der Kopfzeile

- ⇨ **Überprüfen**: Diese Schaltfläche **kompiliert** den Sketch – also übersetzt den Code in Maschinensprache.
  Dabei wird geprüft, ob **Syntaxfehler** oder andere Probleme im Code vorhanden sind.

- ⇨ **Hochladen**: Sendet den kompilierten Sketch über USB an den Mikrocontroller.
  Der Mikrocontroller führt dann das Programm automatisch aus.

- ⇨ **Fenster** mit dem ausgewähltem und verbundenen Board

- ⇨ Mit dem **seriellen Monitor** kannst du Textnachrichten vom Arduino empfangen oder senden.
  Ideal für Debugging oder Messwerte

  Beispiel:

```
void setup() {
  // Wird einmal beim Start ausgeführt
}

void loop() {
Serial.begin(9600);
Serial.println("Hallo Welt!");
delay(1500);
}
```

- ⇨ Der **Serielle Plotter** stellt Werte grafisch dar (z. B. Sensordaten über die Zeit).

Linke Spalte

- ⇨ **Sketchbook**: hier liegen unsere erstellten Programme

- ⇨ **Board-Verwaltung**: Zahlreiche Boards sind schon in der IDE hinterlegt. Wenn wir eine zusätzliche Boardverwalter-URL eingetragen haben, erscheint es hier und das zugehörige Software-Paket kann hier installiert werden.

- ⇨ **Bibliotheksverwalter**: hier können zusätzliche **Bibliotheken (Libraries)** installiert werden, z. B. für Sensoren, Displays, Motorsteuerungen usw.

Meldungsfenster / Konsole

- ⇨ Zeigt unten Informationen über den Kompiliervorgang, Warnungen oder Fehlermeldungen.
- ⇨ Hilfreich zum Debuggen, wenn etwas nicht funktioniert.

Der erste Scetch: Blink

Durch klicken auf Datei -> Beispiele -> Basics -> Blink wird ein neues IDE-Fenster geöffnet und der Sketch geladen.

- ⇨ über „Pfeil rechts“ in der Kopfleiste wird der Sketch auf den Controller geladen

Aufgabe: Ausgabe eines SOS-Signals über die eingebaute Leuchtdiode.

Vorschlag: Dit = 400 ms, Dah = 1200 ms

Info zu **SOS als Morsecode**

▪▪▬▬▬▪▪▪

Der Code **SOS**, drei kurz, drei lang, drei kurz (auch als *Dididididahdahdahdididit* ausgesprochen) wird ohne Pausen zwischen den Buchstaben gesendet.

Ein *Dah* ist dreimal so lang wie ein *Dit*.
Die Pause zwischen zwei gesendeten Symbolen ist ein *Dit* lang.

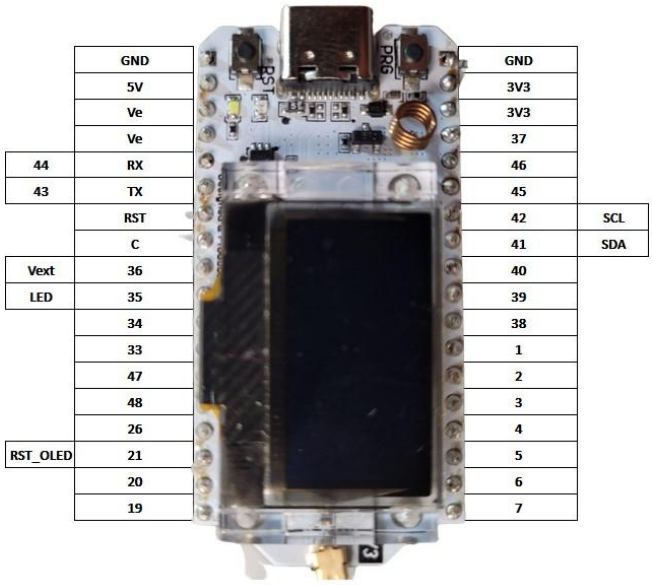
(Quelle: Wikipedia)

Einfache Programmierung:

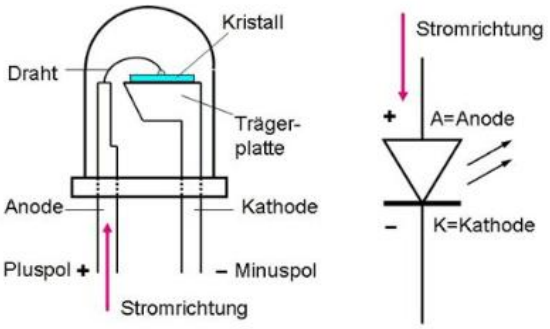
* eine LED ansteuern an Port 7
* Schaltung LED und Widerstand aufbauen
* eine Ampel mit 3 LEDs
* digitaler Eingang: Ein-/Aus-Schaltung mit Taster und LED oder Buzzer

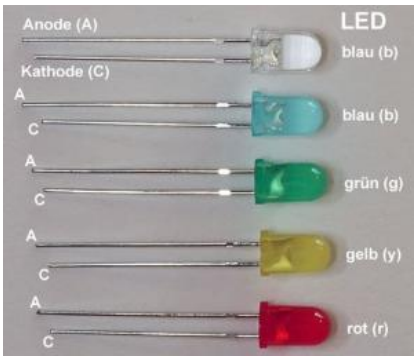
Arbeitsblatt Blinkschaltung mit interner und externer LED

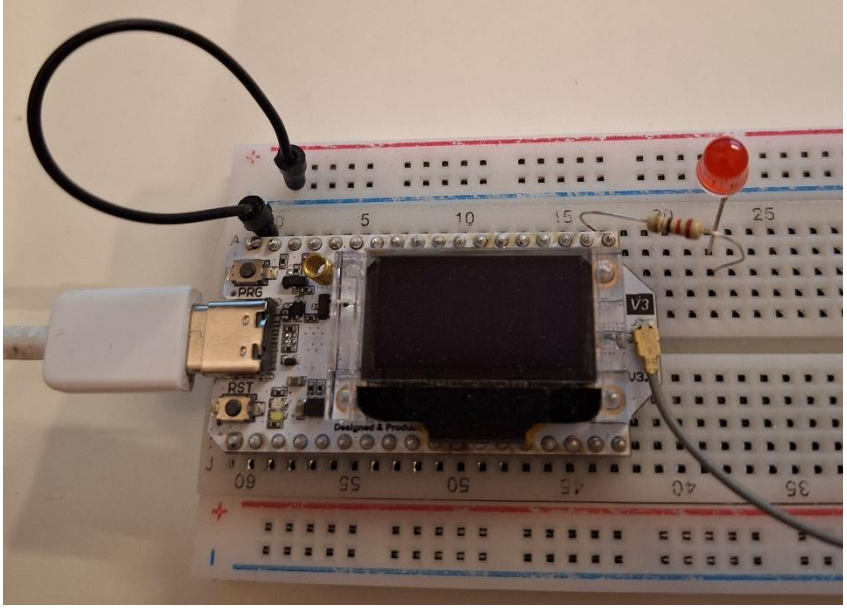
Aufgabe: Verbinde eine LED mit einem Vorwiderstand von 220 Ohm mit GPIO 7 und bringe sie abwechselnd mit der internen LED zum Blinken.



220 Ω LED







Lösungsvorschlag:

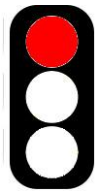
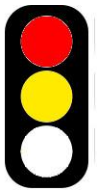
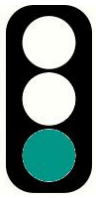
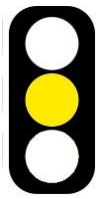
```
#define PinExt 7  // steuert die externe LED

/*
Die folgende setup() Funktion wird nur einmal durchlaufen, wenn das Board mit Strom versorgt oder
der Reset-Taster gedrückt wird
*/
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);  //definiert den digitalen Pin LED_BUILTIN als Output
  pinMode(PinExt, OUTPUT);      // definiert den digitalen PinExt als Output
}

/*
Die folgende loop() Funktion wird immer wieder wiederholt
*/

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // schaltet die interne LED ein
  digitalWrite(PinExt, LOW);         // schaltet die externe LED aus
  delay(1000);                       // wartet eine Sekunde

  digitalWrite(LED_BUILTIN, LOW);   // schaltet die interne LED aus
  digitalWrite(PinExt, HIGH);        // schaltet die externe LED an
  delay(1000);                       // wartet eine Sekunde
}
```

Grundlagen Programmierung in C:

* Variablen definieren, Abfragen, Schleifen, usw.
* eine Ampel mit 3 LEDs
* Bedarfsampel schalten
* digitaler Eingang: Ein-/Aus-Schaltung mit Taster und LED oder Buzzer
* eigene Ideen der TeilnehmerInnen (Lea Auto auf Arduinobais mit Ultraschallsensor)

Arbeitsblatt Ampelschaltung

Aufgabe: Eine Ampel soll nach einem festgelegten Takt geschaltet werden:
5 Sekunden rot, 1 Sekunde rot-gelb, 3 Sekunden grün und 1 Sekunde gelb

rot	rot-gelb	grün	gelb
			
5 Sekunden	1 Sekunde	3 Sekunden	1 Sekunde

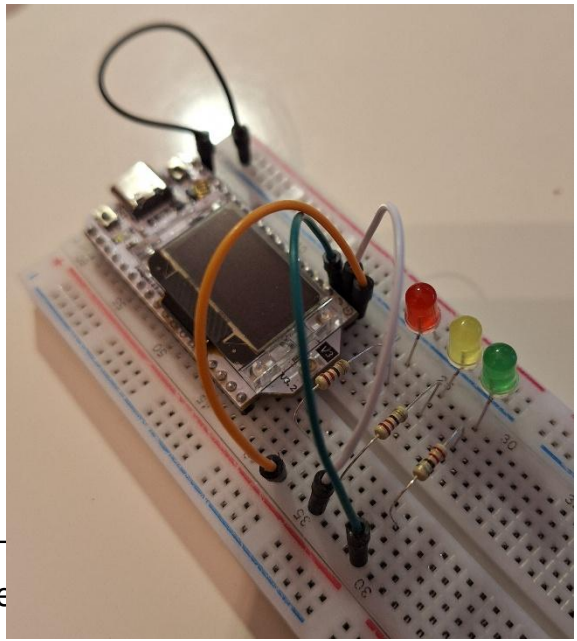
<u>Vorschlag Pinbelegung:</u>

LED rot mit Vorwiderstand 220 Ohm an GPIO 7
LED gelb mit Vorwiderstand 220 Ohm an GPIO 6
LED grün mit Vorwiderstand 220 Ohm an GPIO 5

Lege mit Variablen die Pins fest, an denen die LEDs angeschlossen sind.

<u>Lösungsvorschlag:</u>

```
const int ROT = 7;
const int GELB = 6;
const int GRUEN = 5;

void setup()
{
  pinMode(ROT, OUTPUT);
  pinMode(GELB, OUTPUT);
  pinMode(GRUEN, OUTPUT);
}
```



Quelle: CS, DB, NS Status: in Arbe

```
void loop()
{
  digitalWrite(ROT, HIGH);        // Schritt  1
  digitalWrite(GELB, LOW);        // Schritt  2
  delay(5000);                    // Schritt  3
  digitalWrite(GELB, HIGH);       // Schritt  4
  delay(1000);                    // Schritt  5
  digitalWrite(ROT, LOW);         // Schritt  6
  digitalWrite(GELB, LOW);        // Schritt  7
  digitalWrite(GRUEN, HIGH);      // Schritt  8
  delay(3000);                    // Schritt  9
  digitalWrite(GRUEN, LOW);       // Schritt 10
  digitalWrite(GELB, HIGH);       // Schritt 11
  delay(1000);                    // Schritt 12
}
```

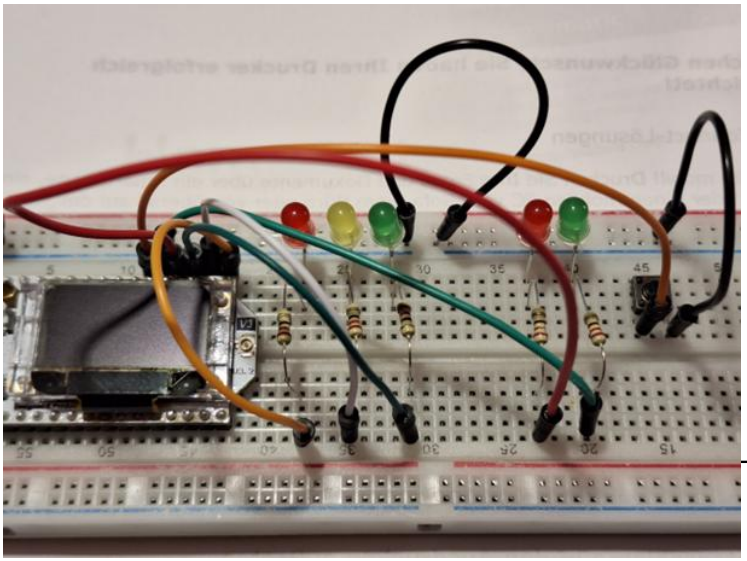
Arbeitsblatt Ampelschaltung mit Fußgängertaster

Aufgabe: Wenn jemand den Taster drückt, soll:

1. Die Autoampel von Grün → Gelb → Rot schalten.
2. Die Fußgängerampel auf Grün gehen.
3. Nach ein paar Sekunden wird sie wieder Rot, und die Autoampel schaltet zurück auf Grün.

Lösungsvorschlag:

Komponente	Pin	Beschreibung
Auto Rot	7	LED mit 220 Ω Widerstand
Auto Gelb	6	LED mit 220 Ω Widerstand
Auto Grün	5	LED mit 220 Ω Widerstand
Fußgänger Rot	4	LED mit 220 Ω Widerstand
Fußgänger Grün	3	LED mit 220 Ω Widerstand
Taster	2	gegen GND, interner Pullup aktiviert



//

Pins definieren

```
const int ROT_AUTO = 7;
const int GELB_AUTO = 6;
const int GRUEN_AUTO = 5;
const int ROT_FUSS = 4;
const int GRUEN_FUSS = 3;
const int TASTER = 2;

void setup()
{
 pinMode(ROT_AUTO, OUTPUT);
 pinMode(GELB_AUTO, OUTPUT);
 pinMode(GRUEN_AUTO, OUTPUT);
 pinMode(ROT_FUSS, OUTPUT);
 pinMode(GRUEN_FUSS, OUTPUT);
 pinMode(TASTER, INPUT_PULLUP);       // Taster schaltet gegen GND

 // Anfangszustand: Autos grün, Fußgänger rot
 digitalWrite(GRUEN_AUTO, HIGH);
 digitalWrite(ROT_FUSS, HIGH);
}

void loop()
{
 if (digitalRead(TASTER) == LOW)          // Warten bis Taster gedrückt wird (aktiv LOW)
 {
  fussgaengerPhase();                     // Funktion starten
 }
}

void fussgaengerPhase()                   // Funktion für die Fußgängerphase
{
 // Autos auf Rot schalten
 digitalWrite(GRUEN_AUTO, LOW);
 digitalWrite(GELB_AUTO, HIGH);
 delay(1000);
 digitalWrite(GELB_AUTO, LOW);
 digitalWrite(ROT_AUTO, HIGH);

 // Sicherheitszeit
 delay(1000);

 // Fußgänger Grün
 digitalWrite(ROT_FUSS, LOW);
 digitalWrite(GRUEN_FUSS, HIGH);
 delay(5000);

 // Fußgänger wieder Rot
 digitalWrite(GRUEN_FUSS, LOW);
 digitalWrite(ROT_FUSS, HIGH);

 // Sicherheitszeit
```

🧠 **Erklärung:**

- Der **Taster** nutzt den internen **Pullup-Widerstand**, deshalb liegt er im Ruhezustand auf **HIGH**, und beim Drücken auf **LOW**.
- Die Funktion fussgaengerPhase() führt eine **zeitlich gesteuerte Sequenz** aus:
  1. Autos auf Rot → Sicherheitspause
  2. Fußgänger grün
  3. Fußgänger rot → Sicherheitspause
  4. Autos wieder grün

Quelle: CS, DB, NS Status:

```
  delay(1000);

  // Autos wieder auf Grün
  digitalWrite(ROT_AUTO, LOW);
  digitalWrite(GELB_AUTO, HIGH);
  delay(1000);
  digitalWrite(GELB_AUTO, LOW);
  digitalWrite(GRUEN_AUTO, HIGH);
}
```

Programmierung Eltec-Board und OLED

* Bibliotheken einbinden
* Welche Sensoren gibt es? Wie funktioniert ein Sensor?
* Was ist ein Datenbus und wie funktioniert ein I2C Bus?
* Verteilen der Sensoren und Breadboard-Verkabelung
* Temperatur- und Luftfeuchtesensor: Messwerte auf dem OLED und I2c -Bus begonnen

Benötigte Libraries

Adafruit AHTX0 von Adafruit

2.0.5 installiert

Arduino library for the AHT10 and AHT20 sensors in the Adafruit shop Arduino library for the AHT10 and AHT20 sensors in the Adafruit shop
Mehr Information

2.0.5 ENTFERNEN

Heltec ESP32 Dev-Boards von Heltec Automation

2.1.5 installiert

Library for Heltec ESP32 (or ESP32+LoRa) based boards Includes: WiFi Kit 32, WiFi LoRa 32, Wireless Stick Lite, Wireless Shell, Vision Master,...
Mehr Information

2.1.5 ENTFERNEN

OneWire von Jim Studt, Tom Pollard, Robin James, Glenn Trewitt, Jason Dangel,...

2.3.8 installiert

Access 1-wire temperature sensors, memory and other chips.
Mehr Information

2.3.8 ENTFERNEN

Heltec_ESP32_LoRa_v3 von Rop Gonggrijp <rop@gonggri.jp>

0.9.2 installiert

Proper working library for "Heltec ESP32 LoRa v3" and "Heltec Wireless Stick v3" boards. No more frustration, no more puzzling it all together....
Mehr Information

0.9.2 ENTFERNEN

Adafruit SSD1306 von Adafruit

2.5.16 installiert

SSD1306 oled driver library for monochrome 128x64 and 128x32 displays SSD1306 oled driver library for monochrome 128x64 and 128x32...
Mehr Information

2.5.16 ENTFERNEN

Adafruit Unified Sensor von Adafruit <info@adafruit.com>

1.1.15 installiert

Required for all Adafruit Unified Sensor based libraries. A unified sensor abstraction layer used by many Adafruit sensor libraries.
Mehr Information

1.1.15 ENTFERNEN

Wie überträgt ein Sensor seine gemessenen Daten?

Es gibt heutzutage eine Vielzahl von Sensoren, die analoge Messwerte erfassen, z.B. Temperatur, Luftdruck, Windstärke, Helligkeit, Abstand, Drehzahl usw.

Die Umwandlung in digitale Werte erfolgt durch sogenannte **Analog-to-Digital Converter** (ADC = Analog-Digital-Wandler).
Diese Daten kann dann ein Mikrocontroller verarbeiten.

Warum ist ADC wichtig?

- Viele Sensoren liefern analoge Signale (z.B. Temperatur, Licht, Druck).
- Der Mikrocontroller arbeitet digital, also mit Zahlen.
- Der ADC übersetzt das analoge Signal in eine digitale Zahl, die dann weiterverarbeitet werden kann.
- In vielen Sensoren ist bereits ein ADC eingebaut, so dass sie einfach in digitalen Systemen verwendet werden können.

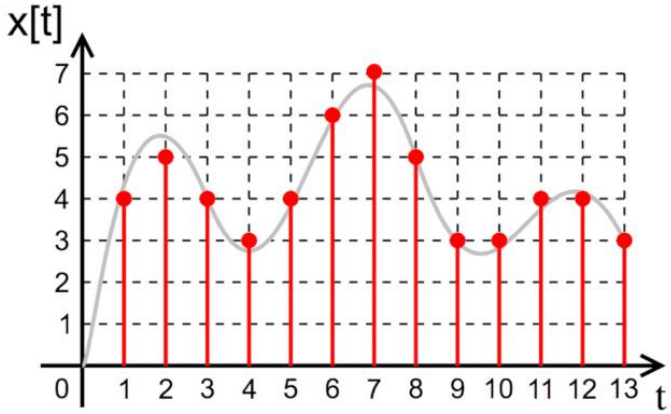
Beispiel: Ein Temperatursensor gibt eine Spannung von 0 bis 3,3 V aus, die der ADC in einen digitalen Wert von z.B. 0 bis 1023 (bei 10-Bit-Auflösung) umwandelt. So kann der Mikrocontroller die Temperatur berechnen.

Wie funktioniert ein ADC technisch?

Ein ADC wandelt ein kontinuierliches analoges Signal (z.B. Spannung) in einen digitalen Wert um.
Das passiert in mehreren Schritten:

- **Abtastung (Sampling):** Der ADC misst in regelmäßigen Abständen den aktuellen Spannungswert am Eingang.
- **Quantisierung:** Die gemessene analoge Spannung wird in einen digitalen Wert umgewandelt.

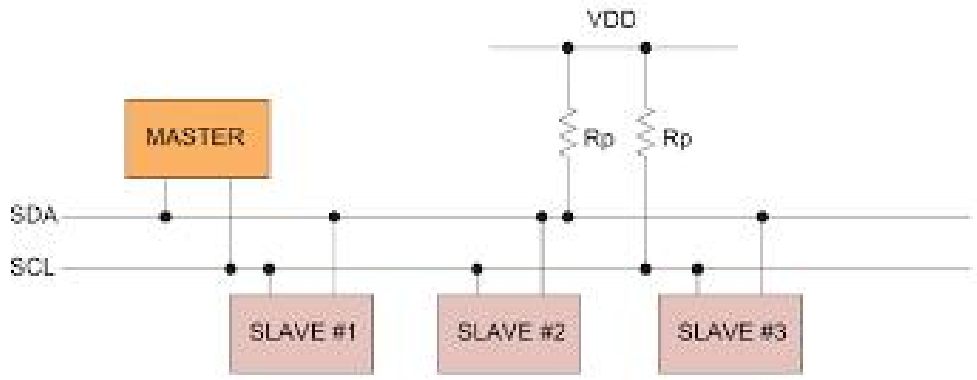
Die digitalen Werte erfolgt oft über



Übertragung der zum Mikrocontroller einen I2C Bus.

Der I²C Bus

Sender/Empfänger (Master & Slave)



Zeitachse (Takt SCL):

SCL: __/‾‾__/‾‾__/‾‾__ (Taktgeber, steuert wann Daten gelesen werden)

SDA: __1__0__1__1__0__0__1__ (Datenbits, synchron zum Takt)

Erklärung:

- Der Master sendet die Adresse des Slaves, dann Daten oder fordert Daten an.
- Die Daten auf SDA ändern sich nur, wenn SCL niedrig ist.
- Die Daten werden bei der steigenden Flanke von SCL (wenn SCL von 0 auf 1 geht) gelesen.
- Start und Stop sind spezielle Signale auf SDA, während SCL HIGH ist

Vorteil

- Es können mehrere Geräte an denselben Bus angeschlossen werden (z. B. Sensoren, Displays).
- Jedes Gerät hat eine eigene Adresse.
- Die Kommunikation wird vom Master-Gerät gesteuert (Takt und Start/Stop).
- Der Bus ist bidirektional: Daten können in beide Richtungen fließen, aber immer nur ein Datenpaket nach dem anderen.

Beispiel: Arduino als I²C-Master, der Daten an einen Slave sendet

Was du brauchst:

- Arduino (z. B. Arduino Uno)
- Ein I²C-Gerät oder ein zweiter Arduino als Slave
- Verbindung über SDA (Pin A4 beim Uno) und SCL (Pin A5 beim Uno)
- Gemeinsame Masse (GND)

Einfacher Arduino-Sketch:

```cpp
#include <Wire.h>  // Bibliothek für I2C

void setup() {
  Wire.begin();  // I2C als Master starten
  Serial.begin(9600);
}

void loop() {
  Wire.beginTransmission(8);  // Adresse des Slave-Geräts (z.B. 8)
  Wire.write("Hallo!");       // Daten senden
  Wire.endTransmission();     // Übertragung beenden

  Serial.println("Daten gesendet");
  delay(1000);                // 1 Sekunde warten
}
```

Erklärung:

- Wire.begin() startet den I²C-Bus als Master.
- beginTransmission(8) sagt, dass Daten an das Gerät mit Adresse 8 gesendet werden.
- Wire.write("Hallo!") sendet die Nachricht.
- endTransmission() beendet die Übertragung.
- Im Loop wird die Nachricht jede Sekunde gesendet.

So sieht ein einfaches Slave-Programm aus:

```cpp
#include <Wire.h>

void setup() {
  Wire.begin(8);                // I2C als Slave mit Adresse 8 starten
  Wire.onReceive(receiveEvent); // Funktion bei Empfang registrieren
  Serial.begin(9600);
}

void loop() {
  // Hier kann der Slave andere Dinge tun
}

void receiveEvent(int howMany) {
  while (Wire.available()) {
    char c = Wire.read();       // Empfangene Daten lesen
    Serial.print(c);            // Daten im Serial-Monitor ausgeben
  }
  Serial.println();
}
```

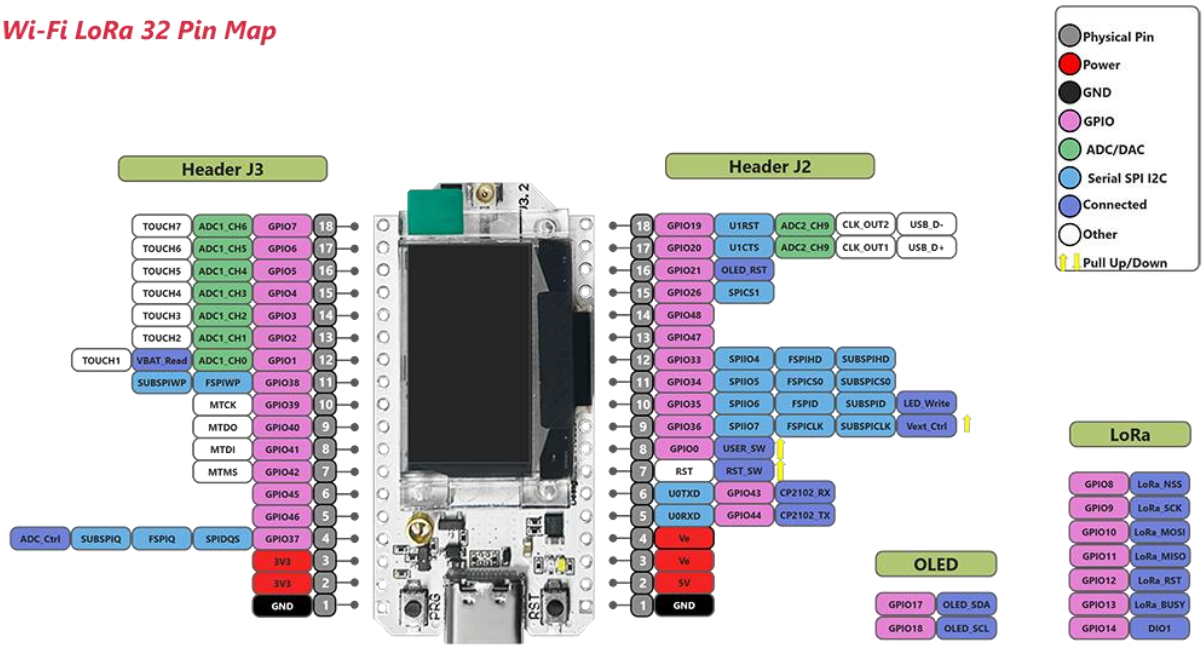
Erklärung:

- Wire.begin(8) macht den Arduino zum Slave mit Adresse 8.
- onReceive(receiveEvent) registriert eine Funktion, die aufgerufen wird, wenn Daten ankommen.
- In receiveEvent werden die empfangenen Bytes gelesen und ausgegeben.

So kannst du es testen:

- Verbinde zwei Arduinos mit SDA, SCL und GND.
- Lade das Master-Programm auf einen Arduino.
- Lade das Slave-Programm auf den anderen Arduino.
- Öffne den Serial-Monitor des Slaves: Du solltest „Hallo!“ jede Sekunde sehen.

Wi-Fi LoRa 32 Pin Map



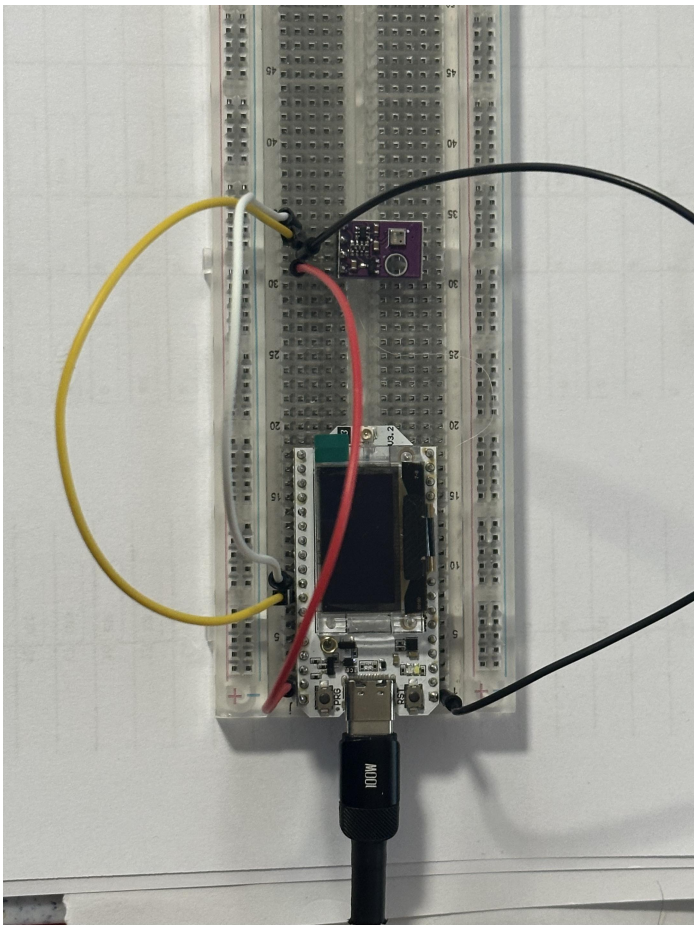
Wiederholungen

* Kurze Wiederholung I2C Bus und Sensoren
* Wiederholung: Ausgabe von Text und Variablen im seriellen Monitor (Ausgabe_SeriellerMonitor.ino)
* Wiederholung: Ausgabe und Positionierung von Text auf dem OLED Display (Ausgabe_OLED1.ino)
* Verkabelung AHT30 auf dem Breadboard
* Ausgabe der Sensordaten auf dem seriellen Monitor (AHT_seriell.ino)
* Ausgabe und Positionierung von Text auf dem OLED Display (Ausgabe_OLED.ino)
* Programmablauf erklären
* Neu zu entwickelnder Sketch: Ausgabe der Sensorwerte auf dem OLED Display

Sketch: AHT_seriell.ino

```
/*********************************************
* Demo AHT-Sensor mit Heltec WiFi LoRa 32 (v3) (AHT_seriell.ino)
*
* angepasst 12-2025 von ows-dieboe
*
*********************************************/
// Library für Sensor einbinden und Objekt anlegen
#include <Adafruit_AHTX0.h>
Adafruit_AHTX0 aht;  // für alle AHT-Typen

// Pins i2c Bus für Sensor
#define PIN_SDA2 40  // Pins sind  Boardbezogen
#define PIN_SCL2 41

void setup() {
 // Sensor initialisieren
 bool status = Wire1.begin(PIN_SDA2, PIN_SCL2);  // SDA + SCL übergeben für
"Wire1", da "Wire" von OLED genutzt wird!
 Serial.print("wire1 status: ");
 Serial.println(status);

 Serial.begin(115200);
 Serial.println("Adafruit AHT demo!");

 if (!aht.begin(&Wire1)) {
```

```
  Serial.println("Could not find AHT? Check wiring");
  while (1) delay(10);
 }
 Serial.println("AHT found");
}

void loop() {
 sensors_event_t humidity, temp;
 aht.getEvent(&humidity, &temp);  // populate temp and humidity objects with fresh
data
 Serial.print("Temperature: ");
 Serial.print(temp.temperature);
 Serial.println(" degrees C");
 Serial.print("Humidity: ");
 Serial.print(humidity.relative_humidity);
 Serial.println("% rH");

 delay(5000);
}
```



Ausgabe_OLED.ino

Ausgabe von Text und Variablen auf dem seriellen Monitor

```
/*
* Textausgabe auf seriellen Monitor und IF Bedingung
*/
// Variable ist in allen Funktionen verfügbar
int anzahl = 1;

// Initialiserung, wird nur EINMAL ausgeführt
void setup() {
  // Geschwindigkeit der seriellen Schnittstelle einstellen
  Serial.begin(115200);
}
// Dauerschleife, wird endlos ausgeführt
void loop() {

  // Ausgabe von Text
  Serial.print("Dieses ist Text");

  // Ausgabe einer Variable
  Serial.print(String(anzahl));
  anzahl++;

  // Ausgabe von Text und Variable
  Serial.println(" + Text" + String(anzahl));
  anzahl++;

  // Prüfung einer Bedingung
  if (anzahl == 7) {
    anzahl = 10;
  }
  delay(5000);
}

```

Fragen:

Warum wird ***anzahl*** VOR ***Setup()*** definiert?

Was ist der Unterschied zwischen ***Serial.print und Serial.println***?

Warum wird ***anzahl*** mit der ***Function String()*** ausgegeben?

Wie verhält sich die Ausgabe, wenn Du die Bedingung in der Zeile 27 auf ***anzahl=6*** prüfst und warum?