Sketching with Hardware

06: Programmieren

Den Arduino programmieren

- Microcontroller sind in Sachen Rechenleistung und Speicher eingeschränkt
- Programme werden in C oder C++ geschrieben
- Einfache Anwendungsfälle → Simple Programme
- Andere Patterns und Paradigmen als z.B. in Java

Grundlegende Programmstruktur

```
// declare variables here

// the setup routine runs once on startup
void setup() {

}

// the loop routine runs over and over again forever:
void loop() {

}
```

Arduinospezifische Funktionen

Funktion	Beschreibung
`delay(int millis)`	Hält das Programm für *`millis`* Millisekunden an
`pinMode(int pin, int direction)`	Definiert einen Pin als *`INPUT`* oder *`OUTPUT`*. Sollte während *`setup()`* aufgerufen werden.
`digitalRead(int pin)`	Liest den Wert eines Pins. Gibt *`HIGH`* (1) oder *`LOW`* (0) zurück.
`analogRead(int pin)`	Liest den Wert eines Analogpins (zwischen 0 und 255).
`digitalWrite(int pin, int value)`	Setzt einen Pin auf *`HIGH`* (5 V) oder *`LOW`* (0 V).
`analogWrite(int pin, int value)`	Setzt die Spannung an einem Pin auf einen Wert zwischen 0 (0 V) und 255 (5 V). Funktioniert nur mit Pins, die *PWM* unterstützen.
`Serial.println(“Hello”)`	Ausgabe über den *Serial Monitor*. Kann zum Debuggen verwendet werden. *`Serial.begin(9600)`* muss vorher aufgerufen worden sein.

Nicht blockierender Code

- Der Arduino kann nur einen Thread ausführen
- Parallelisierung von Tasks schwierig
- Die `delay()`-Funktion blockiert das gesamte Programm
- `delay()` kann vermieden werden, indem ein Timer benutzt wird

Einfacher Timer

```ino
int buttonPin = 5;
int ledPin = 13;
int ledState = LOW;

void loop() {
  // blink the LED
  if(ledState == LOW) {
    ledState = HIGH;
  }
  else {
    ledState = LOW;
  }

  // turn off LED if button is not pressed
  if(digitalRead(buttonPin) == LOW) {
    ledState = LOW;
  }

  digitalWrite(ledPin, ledState);
  delay(1000); // wait for a second
}
```

blockierend

```ino
// ...
long lastMillis = 0;

void loop() {
  // check time since last update
  if(millis() - lastMillis >= 1000) {
    lastMillis = millis();

    if(ledState == LOW) {
      ledState = HIGH;
    }
    else {
      ledState = LOW;
    }
  }

  // turn off LED if button is not pressed
  if(digitalRead(buttonPin) == LOW) {
    ledState = LOW;
  }

  digitalWrite(ledPin, ledState);
}
```

nicht blockierend

Codestruktur

- Arduino-Code ist oft simpel und sequentiell
- Wird schnell unübersichtlich
- Fehlersuche in Code und Hardware ist umständlich
- Gut strukturierter Code ist leichter zu lesen und einfacher zu debuggen

State Machine: Keksautomat

```
#define WAIT 1
#define ORDER 2
#define PAYMENT 3
#define DISPENSE 4

int state = WAIT;
int order;

// cookie dispenser functions
int getInput() { … }
int getOrder() { … }
void handlePayment(int product) { … }
void dispenseProduct(int product) { … }
```

```
void loop() {
  switch(state) {
    case WAIT:
      if(getInput() != 0) state = ORDER;
      else delay(1000);
      break;
    case ORDER:
      order = getOrder();
      state = PAYMENT;
      break;
    case PAYMENT:
      handlePayment(order);
      state = DISPENSE;
      break;
    case DISPENSE:
      dispenseProduct(order);
      state = WAIT;
      break;
}
```

Debug Levels

- Die Arduino IDE hat keinen eingebauten Debugger
- Serielle Ausgabe wird zum loggen des Programm-zustands verwendet
- Mithilfe von Debug Levels kann man schnell den Detailgrad der Ausgaben anpassen

```
#define NONE 0
#define ERROR 1
#define WARN 2
#define DEBUG 3
#define ALL 4
#define DEBUG_LEVEL WARN

void loop() {
  if(DEBUG_LEVEL >= WARN) {
    Serial.println(“Warning!”);
  }
}
```