Inhaltsverzeichnis

Übersicht

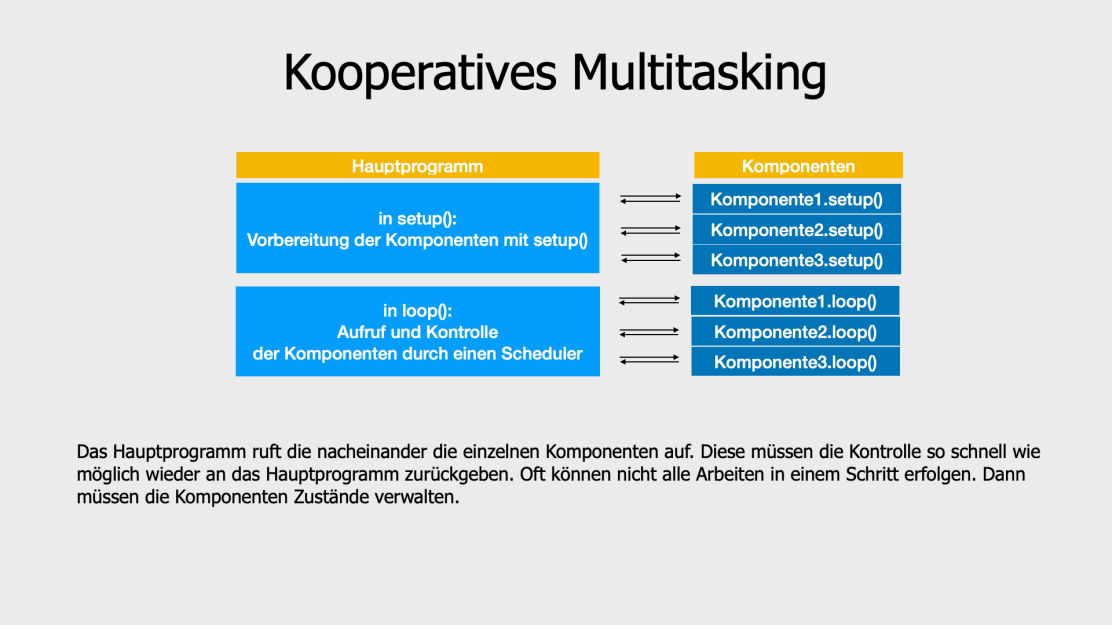
Multitasking mit dem Arduino ist für wenig erfahrene Entwickler immer eine grosse Herausforderung. Das von Arduino bereitgestellte Beispiel 'Blink without Delay' ist auch nicht besonders hilfreich. Deshalb möchte ich eine kleine Bibliothek vorstellen, die auch Einsteigern das Programmieren 'gleichzeitig' ablaufender Vorgänge ermöglicht.

Das geht nicht ohne Klassen und objektorientierte Programmierung. Ich habe in der Library 'arduino_multitasking' versucht, den Anwender möglichst von der dazugehörigen Syntax abzuschirmen. Es wird hauptsächlich mit den bekannten Begriffen wie setup() und loop() gearbeitet.

'arduino_multitasking' ist keine vollständige Bibliothek, die über den Library Manager installiert wird. Es handelt sich dabei nur um die Dateien arduino_multitasking.h und arduino_multitasking.cpp. Diese beiden Dateien werden in das Projektverzeichnis kopiert und stehen so zur Verfügung.

Implementiert ist ein kooperatives Multitasking, das vollständig ohne Interrupts auskommt.

Der Code ist experimentell. Er wurde speziell für eine Präsentation am Arduino und Elektronik Abend im FabLab Zürich erstellt. Er steht aber auf Github zur Verfügung. Es würde mich freuen, wenn ihr selbst damit experimentieren würdet.



Zwei Leds und eine Ampel

An einer Kurzpräsentation lässt sich nur ein einfaches Beispiel zeigen. So besteht die Hardware aus einem Arduino UNO und 5 Leds. Drei der Leds sind zu einer Ampel mit grün, gelb und rot zusammengefasst.

Wir haben folgende Funktionen:

- Eine rote Led blinkt mit einem Delay von 500 ms.
- Eine blaue Led blinkt mit einem Delay von 1200 ms.
- Eine Ampel zeigt rot für 8 Sekunden
- Danach gelb / rot für 2 Sekunden
- Danach grün für 6 Sekunden
- Danach gelb für 3 Sekunden
- Dann erfolgt ein Wechsel nach rot und der Zyklus startet neu.

Das Programmgerüst

Wir verwenden die Arduino IDE Version 2.x und erstellen ein neues Projekt. Wir können es auch aus der Vorlage MultitaskingFablab kopieren.

Wir haben im Projektverzeichnis 3 Dateien:
- MultitaskingFablab.ino
- arduino_multitasking.h
- arduino_multitasking.cpp

Wenn wir MultitaskingFablab.ino aus der Vorlage kopiert haben, steht uns bereits ein Programmgerüst zur Verfügung.

Der Anfang

```
#include "arduino_multitasking.h"

// Zustandskonstanten, optional, wird nur bei komplexeren Projekten benötigt
enum States {
  AUS
};
```

Import von arduino_multitasking
States benötigen wir noch nicht. Wir sehen später, was sie für einen Zweck haben.

Komponenten

Das Programm besitzt eine oder mehrere Komponenten. In unserem Fall sind das die Leds und die Ampel. Für die Komponente wird eine Klasse erstellt, mit der wir später ein oder mehrere Objekte erstellen können.

```
class DemoComponent : public Component {
  public:

    // Variablen (optional)

    // Funktionen (optional)
    void start() {}

    // setup() kann auch Parameter besitzen
    void setup() {}

    // loop() bestimmt das Verhalten
    void loop() {}
};
```

Die Komponenten müssen dann mit Hilfe der Klasse erstellt werden.

```
// Komponenten erstellen
DemoComponent demo;
```

Setup

```
void setup() {
  // Setup der Applikation
  Serial.begin(9600);

  // Setup der Komponenten
  demo.setup();

  // Komponenten dem Scheduler hinzufügen
  scheduler.add(demo);

  // Den Komponenten Aufgaben zuweisen
  demo.start();
}
```

In setup() werden zuerst die Teile initialisiert, die keine Komponenten sind.
Danach werden die setup() - Methoden der Komponenten aufgerufen.
Jetzt können alle Komponenten dem Scheduler hinzugefügt werden.
Zum Schluss teilen wir den Komponenten noch mit, was sie tun sollen.

Loop

In loop() wird immer zuerst scheduler.loop() aufgerufen.
Danach können weitere Funktionen aufgerufen werden. Diese dürfen aber nicht blockierend sein. Deshalb ist delay() hier verboten. Für Wartezeiten muss das nicht blockierende _delay() verwendet werden.

Die blinkende Led

Wir gehen von der Vorlage ***DemoComponent*** aus und erstellen eine Klasse ***BlinkLed***.

Der ***Kopf*** ist immer derselbe. Es wird lediglich der korrekte Name eingetragen.
```
class BlinkLed : public Component {
  public:
```

Wir wollen die Klasse für mehrere Leds verwenden. Deshalb müssen wir die Delay - Zeit und den Anschlusspin speichern.
```
    // Variablen
    int blinkDelay;  // Pause beim Blinken
    int _pin;   // Pin für die Led
```

Im ***Setup*** der Klasse übergeben wir den Pin, initialisieren den Output und setzen die Delay - Zeit auf 0.
```
    void setup(int pin) {
      _pin = pin;
      pinMode(_pin, OUTPUT);
      digitalWrite(_pin, LOW);  // Led aus
      blinkDelay = 0;  // nicht blinken
    }
```

Im ***Loop*** der Klasse wird geblinkt. Aber nur wenn blinkDelay nicht 0 ist. Sonst wird die Led ausgeschaltet.
```
    void loop() {
      if (blinkDelay) {
        wait(blinkDelay);
        digitalWrite(_pin, !digitalRead(_pin));
      } else {
        digitalWrite(_pin, LOW);
      }
    }
```

Die rote Led

Jetzt kann die Komponente für die rote Led erstellt werden.
```
BlinkLed roteLed;  // eine rote Led erzeugen
```

Im ***Setup*** des Programms bereiten wir alles vor.
```
void setup() {
  roteLed.setup(2);          // Rote Led mit Pin 2 initialisieren-
  scheduler.add(roteLed);    // Die Led muss dem Scheduler hinzugefügt werden.
  roteLed.blinkDelay = 500;  // Delayzeit von 500 ms.
}
```

Damit arbeitet die rote Led bereits.

Hinzufügen der zweiten Led

Eine blaue Led kann sehr einfach hinzugefügt werden.
```
BlinkLed roteLed;
BlinkLed blaueLed;

void setup() {
  roteLed.setup(2);   // rote Led an Pin 2
  blaueLed.setup(3);  // blaue Led an Pin 3

  scheduler.add(roteLed);   // Rote Led hinzufügen
  scheduler.add(blaueLed);  // Blaue Led hinzufügen

  roteLed.blinkDelay = 500;    // Blinken mit 500 ms
  blaueLed.blinkDelay = 1200;  // Blinken mit 1200 ms
}
```

Die Ampel

Die Ampel ist etwas komplizierter. Sie besteht aus 3 Leds, die einem Ablauf folgen. Das Beispiel zeigt, wie ein unabhängiger Handlungsstrang programmiert werden kann.

Zustände

Das ist der Moment, wo wir die States (Zustände) benötigen. Die Ampel kann ausgeschaltet, rot, rot-gelb, gelb oder grün sein. Dafür erstellen wir Konstanten.

```
enum States {
  AUS, GRUEN, GELB, ROT, ROT_GELB
};
```

Die Werte der Konstanten sind unerheblich. Sie müssen sich aber unterscheiden. Das wird mit ***enum*** sichergestellt.

Die Ampel - Klasse

Der Kopf

Für die Komponente beginnen wir wieder mit dem Kopf.

```
class Ampel : public Component {
  public:
    // Die Pins für die Leds müssen gespeichert werden.
    int _rot, _gelb, _gruen;
```

Setup

Die Pins werden gespeichert und die Ports vorbereitet. Der Zustand ist beim Start AUS.

```
    void setup(int rot, int gelb, int gruen) {
      _rot = rot;
      _gelb = gelb;
      _gruen = gruen;
      pinMode(_rot, OUTPUT);
      pinMode(_gelb, OUTPUT);
      pinMode(_gruen, OUTPUT);
      state = AUS;
    }
```

Loop

Der Loop der Komponente ist etwas komplexer. Wir müssen jeden Zustand einzeln behandeln und für den rechtzeitigen Wechsel in den nächsten Zustand sorgen. Beginnen wir mit AUS:

```
    void loop() {
      switch (state) {

        case AUS:
          digitalWrite(_gruen, LOW);
          digitalWrite(_gelb, LOW);
          digitalWrite(_rot, LOW);
          break;
```

Wenn der Zustand AUS ist, werden alle Leds ausgeschaltet. Dieser Zustand wird beibehalten, bis er von aussen geändert wird.

```
        case GRUEN:
          digitalWrite(_gruen, HIGH);
          digitalWrite(_gelb, LOW);
          digitalWrite(_rot, LOW);
          wait(6000);
          state = GELB;
          break;
```

Die grüne Led wird eingeschaltet, die anderen aus. Dieser Zustand wird für 6 Sekunden aufrecht erhalten. wait() ist ein nicht blockierender Delay innerhalb einer Komponente. Nach Ablauf der Zeit wird in den Zustand GELB gewechselt.

Die anderen Zustände funktionieren identisch.

```
        case GELB:
          digitalWrite(_gruen, LOW);
          digitalWrite(_gelb, HIGH);
          digitalWrite(_rot, LOW);
          wait(3000);
          state = ROT;
          break;

        case ROT:
          digitalWrite(_gruen, LOW);
          digitalWrite(_gelb, LOW);
          digitalWrite(_rot, HIGH);
          wait(8000);
          state = ROT_GELB;
          break;

        case ROT_GELB:
          digitalWrite(_gruen, LOW);
          digitalWrite(_gelb, HIGH);
          digitalWrite(_rot, HIGH);
          wait(2000);
          state = GRUEN;
          break;
```

state ist bereits in der Basisklasse ***Component*** enthalten, daher ist die Arbeit damit sehr einfach. Component wird durch die Bibliothek arduino_multitasking zur Verfügung gestellt.

Einbinden und starten der Ampel

```
BlinkLed roteLed;
BlinkLed blaueLed;
Ampel ampel;

void setup() {
  // setup der Komponenten
  roteLed.setup(2);
  blaueLed.setup(3);
  ampel.setup(11, 12, 13);

  // Komponenten zum Scheduler hinzufügen
  scheduler.add(roteLed);
  scheduler.add(blaueLed);
  scheduler.add(ampel);

  // Aufgaben verteilen
  roteLed.blinkDelay = 500;
  blaueLed.blinkDelay = 1200;
  ampel.state = ROT;  // Start mit ROT
}
```