Zeitsteuerungen ohne delay()

Die Realisierung von Zeitsteuerungen auf dem Arduion ohne die Verwendung der delay() Anweisung. Eine Erklärung am Beispiel einer kleinen elektrischen Eisenbahn.

Der Arduino ist eine feine kleine Plattform, mit der vielfältige Aufgaben auf unseren Mobas gesteuert werden können. Leider liegt - wie so oft - auch hier etwas Schweiß vor dem Spaß, denn das kleine Ding will ja auch programmiert werden.
Häufig gibt es da gerade mit Zeitsteuerungen bei vielen Einsteigern Probleme. Zunächst erscheint das mit dem delay() - Aufruf recht einfach, aber man merkt doch recht schnell, dass man damit in einer Sackgasse landet, aus der es keinen Ausweg gibt: mehrere Dinge gleichzeitig zu machen funktioniert so nicht.
Wie immer gibt es viele Wege nach Rom, aber für einen Einsteiger ist es erstmal wichtig wenigstens einen (möglichst einfachen) zu kennen.

Als Beispiel möchte ich dazu einen Wechselblinker (Andreaskreuz) und eine einfache Ampelsteuerung mit einem Arduino realisieren, so daß beides vollkommen unabhängig voneinander funktioniert. Ich werde dabei auch meine MobaTools verwenden, da die doch einiges vereinfachen.
Schritt für Schritt soll so dann das komplette Programm entstehen. Erst für den Wechselblinker, dann für die Ampel. Und zum Schluß wird sich zeigen, wie einfach man beides kombinieren kann, wenn man von Anfang an 'richtig' vorgegangen ist.

Der Schlüssel für das Ganze ist ein 'Zustandsautomat'. Mit dieser Technik lassen sich sehr viele Aufgaben auf dem Arduino erschlagen, weshalb es erstmal wichtig ist, dieses Prinzip zu verstehen.

Bevor wir mit dem Programmieren anfangen, müssen wir unser Problem dazu erstmal in kleine 'Häppchen' zerlegen. Diese 'Häppchen' sind die verschiedene Zustände, in denen sich das System befinden kann. Außerdem die Ereignisse, die - je nach Zustand - unterschiedliche Aktionen und Zustandswechsel auslösen.

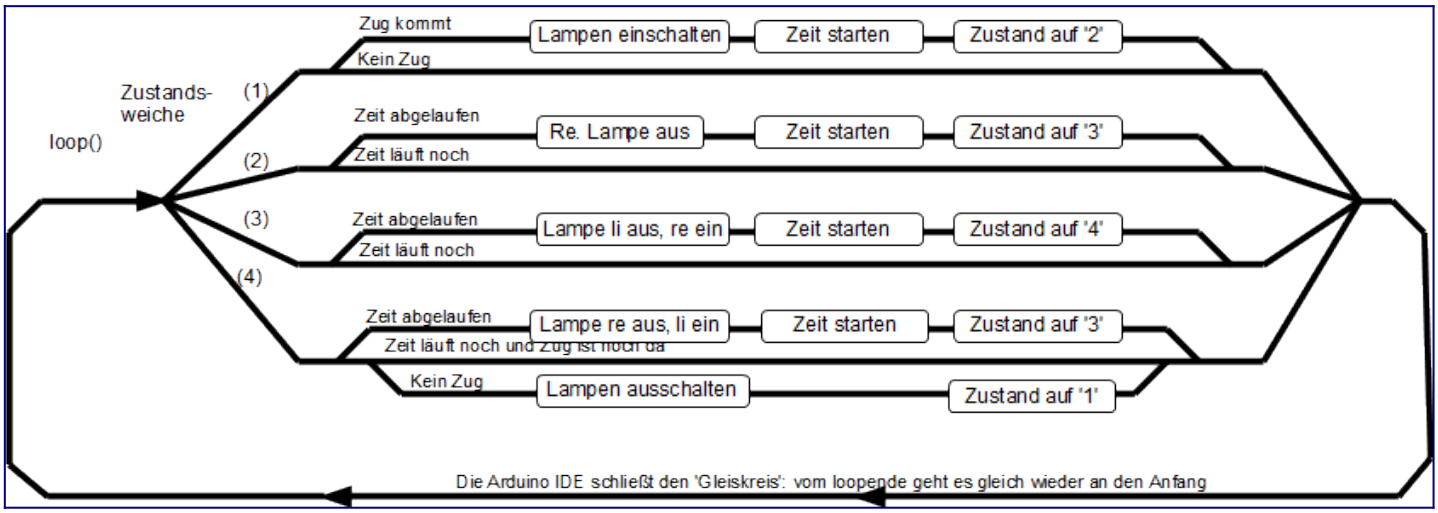
Fangen wir erstmal mit dem Wechselblinker am Bü an. Rechts und links von der Straße steht jeweils ein Andreaskreuz mit einem Blinklicht. Bei Annäherung eines Zuges gehen kurz beide Lampen an, dann blinken sie im Wechseltakt. Hat der Zug den Bü überquert, gehen die Lampen wieder aus.
Welche Zustände können wir nun für dieses System identifizieren:

1. Der Blinker ist im Ruhezustand. Alle Lampen sind aus. Das System wartet darauf, dass sich ein Zug nähert.
   >>Kommt der Zug, werden beide Lampen angeschaltet, die Stoppuhr gestartet, und wir wechseln in den 2. Zustand
2. Der Zug hat den Kontakt ausglöst, beide Lampen sind für einen kurzen Zeitraum an. Die Stoppuhr für diese Zeit läuft.
   >>Ist die Stoppuhr abgelaufen, schalten wir die rechte Lampe aus, starten die Stoppuhr erneut mit der Blinkzeit und wir wechseln in den 3. Zustand.
3. Jetzt ist nur noch die linke Lampe an, die rechte ist aus. Die Blinkzeit läuft.
   >>Ist die Blinkzeit abgelaufen, wird die linke Lampe aus, und die rechte Lampe eingeschaltet. Die Blinkzeit wird neu gestartet und wir wechseln in den 4. Zustand
4. Jetzt ist nur die rechte Lampe an, die linke ist aus. Die Blinkzeit läuft.
   >>Ist die Blinkzeit abgelaufen, wird die reche Lampe aus, und die linke Lampe eingeschaltet. Die Blinkzeit wird neu gestartet und wir wechseln wieder in den 3. Zustand
   >>hat der Zug den Bü verlassen, werden beide Lampen ausgeschaltet, und wir wechseln in den 1. Zustand.

Jetzt haben wir unseren Wechselblinker also in 4 Zustände zerlegt, in denen sich das System befinden kann. In jedem Zustand wartet dieser Automat nun auf ein Ereigns (mit '>>' markiert). Tritt das Ereignis ein, wird eine Aktion gestartet, und dann der Zustand gewechselt. Im letzten Zustand warten wir sogar auf 2 Ereignisse. Das Ereignis, dass zuerst eintritt, bestimmt die ausgeführte Aktion und den Zustandswechsel.
Damit haben wir unser Problem in kleine Schritte zerlegt, die wir nun auch gedanklich durchgehen können, ob das auch alles so passt.
Da ist noch keine einzige Zeile Code geschrieben.

Im nächsten Schritt will ich nun vorstellen, wie man diesen noch theoretischen Ablauf in konkreten Code umsetzen kann.

Die Liste im ersten Post müssen wir nun in einen Programmablauf umsetzten. Ich möchte das hier mal - die Programmierspezialisten mögen mir verzeihen - für die Einsteiger an einem Gleisplan demonstrieren. Der Arduino ist der Lokführer, der mit seiner Lok (=dem Prozessor) diesen Gleisplan abfährt.
Programmverzweigungen sind die Weichen und es gibt Stationen, an denen unser Lokführer etwas zu tun hat.



Am Anfang steht eine 4-Wege Weiche, die entsprechend dem aktuellen Zustand gestellt ist. Wie in einem Rangierbahnhof verzweigt es sich, und die folgenden Weichen in jedem Zweig sind je nachdem gestellt, ob ein erwartetes Ereignis eingetrofffen ist oder nicht. Am Ende kommen alle Zweige wieder zusammen, und es geht wieder von vorn los.
Ihr könnt jetzt mal in Gedanken (oder mit dem Finger) die 'Lok' kreisen lassen, und den Programmablauf durchspielen. In den weißen Kastchen ist was zu tun, ansonsten geht es ohne Unterbrechung immer im Kreis herum - mal hier, mal da lang

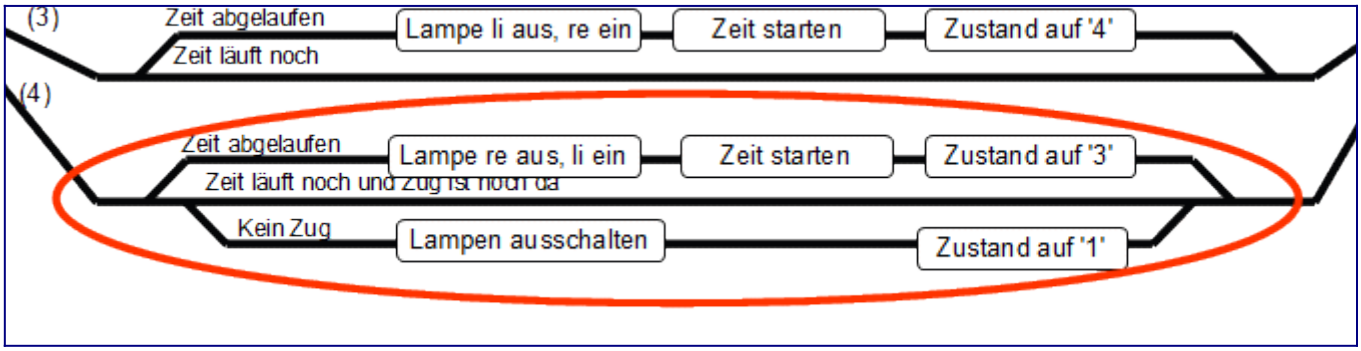
Natürlich gibt es für solche Abläufe auch informationstechnisch korrekte Symboldarstellungen - aber letztendlich zeigen die auch nichts anders .
Vielleicht macht es Euch so mehr Spaß, das ganze gedanklich durchzuspielen.

Aber zuerst vielleicht noch eine kleine Zeitbetrachtung zu unserem Ablaufplan. Unsere Arduino-Lok ist schnell - sehr schnell sogar. Betrachten wir mal die Zeiten, um dies es da geht, wenn wird die Station 'Zeit starten' abarbeiten. Für einen schönen Wechselblinker werden wir da Zeiten so um eine 1/2 Sekunde, also 500ms verwenden. Für unseren Arduino ist das eine halbe Ewigkeit. Wenn der dann beim nächsten Durchlauf an der Weiche mit 'zeit abgelaufen?' vorbeikommt, hat sich da sicher noch nichts getan, und er flitzt einfach durch das frei Gleis durch. Bis die Weiche sich umstellt auf 'zeit abgelaufen' ist unsere Arduino-Lok schon mehrere Tausend Male im Kreis gelaufen ohne irgendetwas tun zu müssen. Ein aktueller ICE ist nix dagegen .

Vielleicht dämmert euch auch schon ein wenig, wie wir es später schaffen, dass der Arduino mehrere Dinge 'gleichzeitig' macht. Richtig, wenn wir die weitere(n) Aufgaben genauso strukturieren wie unseren Wechselblinker, können wir das im 'Gleiskreis' einfach hintereinander hängen. Jeder dieser 'Rangierbahnhöfe' stellt seine Weichen dann intern selbst - so wie unser Wechselblinker auch - und die Arduino-Lok fährt dann nacheinander durch jeden hindurch. Und je nach Weichenstellung muss der Lokführer kurz was erledigen, oder es geht ohne weitere Stationen einfach hindurch.

Diese Ablaufpläne zeigen etwas sehr schön, was man später im Code nicht mehr so einfach erkennt: Die Parallelität und damit Exklusivität der jeweiligen Pfade. In jeder Runde kann immer nur genau eins von den parallelen Gleisen (Aufgaben) befahren werden. Wie in dem Moment die Weichen in einem anderen Pfad stehen ist da dann vollkommen irrelevant.
Später im Code steht alles untereinander, da ist das nicht mehr so gut zu erkennen. Deshalb ist es immer sinnvoll, sich die Abläufe vor dem Codieren mit so einem Plan klarzumachen. Da reicht auch eine einfache Bleistiftzeichnung. Und Gleispläne zeichnen könnt ihr doch schließlich alle

Den Plan oben habe ich nochmal ein klein wenig überarbeitet. Nicht das sich an den Abläufen grundsätzlich etwas geändert hätte. Aber die 'Weichen' sind etwas anders platziert. So sieht man besser, dass zu jeder Verzweigung auch das genaue Gegenstück beim Zusammenführen gehört. Man kann jetzt um so einen Verzweigungblock quasi einen Kringel drummachen mit genau einem Eingang und einem Ausgang, und das entspricht dann auch genau einem Programmblock im Code.



Nun wollen wir mal darangehen, unseren Ablaufplan auch in ablauffähigen Code umzusetzen. Die einfachen 'Weichen' sind sicher jedem klar: das sind einfache if-Anweisungen. Gibt es in beiden Zweigen etwas zu tun, brauchen wir auch einen else Zweig, ist ein Zweig ein 'leeres Gleis' reicht ein einfaches if.
Aber wo bekommen wir unsere 4-Wege-Weiche her?. Klar könnte man das in einzelne Verzweigungen auflösen, aber das ist nicht notwendig.
C++ bietet für sowas eine eigene Anweisung: die 'switch' Anweisung.
Die Grundstruktur sieht so aus:

```
switch ( switchVariable ) {
  case 1:
    // Codeblock, der ausgeführt wird, wenn die switchVariable den Wert 1 hat
    break; // Ende des Codeblocks, das Programm läuft hinter dem Switch-Block weiter
  case 2:
     // Codeblock, der ausgeführt wird, wenn die switchVariable den Wert 2 hat
    break; // Ende des Codeblocks, das Programm läuft hinter dem Switch-Block weiter
  default:
    // Codeblock, der ausgeführt wird wenn die switchVariable einen Wert hat, der in
```

```
    // keiner der obigen case-Zeilen vorkommt
    // der 'default-Block' ist optional. Fehlt das default:, so verhält er sich genauso, als wenn
    // dieser Codeblock leer ist. D.h. passt keine der obigen case-Zeilen, wird das gesamte switch
    // übersprungen, und es geht danach weiter
}
```

In Abhängigkeit vom Wert der switchVariable wird die passende case-Zeile angesprungen. Dann werden die dortigen Anweisungen ausgeführt, und mit dem 'break;' springt das Programm zum Ende des gesamten switch-Blocks und macht dahinter weiter.
Hinter dem 'case' Schlüsselwort sind nur Konstante erlaubt, und natürlich müssen sie eindeutig sein. D.h. 2 case-Zeilen mit dem gleichen Konstantenwert sind nicht erlaubt. Wenn man das 'break;' vergisst, springt das Programm nicht ans Ende des switch-Blockes, sondern macht einfach mit dem Code hinter der nächsten case-Zeile weiter. Für die Zahl der case-Zeilen gibt es keine fest Grenze.

Diese Anweisung ist nun perfekt für unseren Zustandsautomaten geeignet. Die switch-Variable enthält immer den Wert für den aktuellen Zustand unseres Automaten und so wird bei jedem Durchlauf der entsprechende Codeblock ausgeführt. Damit werden wir nun unseren 'Rangierbahnhof' realisieren.

Wie wir gesehen haben, ist die switch-Anweisung der richtige Rahmen, um unseren Ablaufplan in Code umzusetzten. Nun müssen also die einzelnen case-Blöcke noch mit Leben gefüllt werden. Zunächst möchte ich jeden case-Block - also die Automatenzustände - einzeln aus unserem Ablaufplan ableiten.
Dann bauen wir das zu der kompletten switch Anweisung zusammen. Zum Schluß müssen wir dann sehen, was wir noch 'drumherum' brauchen, damit es ein lauffähiger Sketch wird.

Als erstes ein paar Bemerkungen zu den verwendetetn Variablen und Funktionen:
bueBelegt
Diese Variable zeigt uns an, dass ein Zug auf den Bü zufährt. Sie wird TRUE sobald sich der Zug nähert, und FALSE wenn er den Bü komplett überquert hat. Im einfachsten Fall kann das ein entsprechender Belegtabschnitt sein, dann ist es einfach ein Einganspin, den wir abfragen. Evtl. kann dies aber auch eine komplexere Logik sein. In unserem Zustandsautomaten soll uns aber erstmal noch nicht interessieren, wie diese Variable gesetzt wird.
wBlinkerZustand
Das ist unsere Zustandsvariable, die bestimmt, in welchem Zustand sich unser Automat befindet. Also sozusagen die Stellung der 4-Wege Weiche am Anfang.

LampeLinks LampeRechts
Das sind SoftLed-Objekte der MobaTools. Damit können wir ganz einfach durch die Methoden .on() und .off() unsere Led's im Andreaskreuz weich auf- und abblenden.
StoppUhr
Das ist ein EggTimer Objekt der MobaTools. Damit können wir unsere Zeit setzen, und abfragen ob sie abgelaufen ist.

Wie wir die MobaTools Objekte einrichten und initiieren werden wir später sehen, wenn der switch-Block (also unser Zustandsautomat) zum kompletten Sketch erweitert wird.

Zustand 1 - Ruhezustand


```
  case 1: // Grundzustand, abfragen, ob sich ein Zug nähert ( z.B. über einen Belegtmelder )
    if ( bueBelegt ) {
      // Der Zug ist in den Belegtabschnitt vor der Schranke eingefahren
      // beide Lampen einschalten
      LampeLinks.on();
```

```
          LampeRechts.on();
          // Stoppuhr mit der Zeit die beide Lampen an sein sollen vorbelegen
          StoppUhr.setTime( STARTZEIT ); // StoppUhr 'aufziehen'
          // in den 2. Zustand wechseln
          wBlinkerZustand = 2; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen des case 2
ausgeführt
        }
        break; // Ende der Anweisungen für den 1. Zustand
```

Zustand 2 - beide Lampen sind an


```
      case 2: // beide Lampen ein, wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um
          LampeRechts.off();
          StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
          // in den 3. Zustand wechseln
          wBlinkerZustand = 3; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen des case 3
ausgeführt
        }
        break; // Ende der Anweisungen für den 2. Zustand
```

Zustand 3 - links an / rechts aus


```
      case 3: // linke Lampen ist an, rechte aus: wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um, Lampen umschalten
          LampeLinks.off();
          LampeRechts.on();
          StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
          // in den 4. Zustand wechseln
          wBlinkerZustand = 4;  // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen
                                // des case 4 ausgeführt
        }
        break; // Ende der Anweisungen für den 3. Zustand
```

Zustand 4 - links aus / rechts an


```
      case 4: // linke Lampen ist aus, rechte an: wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um, Lampen umschalten
          LampeLinks.on();
          LampeRechts.off();
          StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
          // wieder in den 3. Zustand wechseln
          wBlinkerZustand = 3; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen
                               // des case 3 ausgeführt
        }
        // Im 4. Zustand müssen wir auch noch abfragen, ob der Zug den Belegtabschnitt verlassen hat
        if ( not bueBelegt ) {
          // Der Zug ist komplett über den Bü gefahren, die Straße ist wieder frei
          // Beide Lampen ausschalten
          LampeLinks.off();
          LampeRechts.off();
          // wieder in den 1. Zustand wechseln. Eine Zeitüberwachung wird nicht gebraucht
          wBlinkerZustand = 1; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen
                               // des case 1 ausgeführt
        }
        break; // Ende der Anweisungen für den 4. Zustand
```

Einer der 4 Zustände ist übrigens nicht exakt so codiert wie es dem Ablaufplan entspricht. Erkennt ihr welcher? Und wo ist der Unterschied?

Nach den theoretischen Vorüberlegungen wollen wir uns jetzt daranmachen und das Ganze zu einem lauffähigen Sketch zusammenbauen. Dann bleibt das nicht so theoretisch, und es tut sich schon was, bevor wir das Ganze um eine unabhängige Ampelsteuerung erweitern (Multitasking).

Da rufen wir uns in Erinnerung, wie so ein Sketch aufgebaut ist. So ein Sketch besteht zumindest aus 3 Abschnitten:

1. Im ersten Block sagen wir dem Arduino, was wir alles benutzen wollen. Wir binden Bibliotheken ein, definieren die globalen Variablen und Libraryobjekte, die wir verwenden und ordnen gegebenenfalls konstanten Zahlen auch einen Namen zu, damit man sie im Bedarfsfall einfacher ändern kann (z.B. die Pin-Nummern).
2. der setup-Block. In der setup-Funktion werden alle die Dinge erledigt, die beim Starten des Sketches einmalig gemacht werden müssen. Dazu gehört die Festlegung ob die verwendeten Pins Ein- oder Ausgänge sind und das Initiieren der verwendeten Library-Objekte
3. der loop-Block. Das eigentliche 'Herzstück' des Sketches. Hier dreht unsere 'Arduino-Lok' dann endlos ihre Runden

Was gehört bei unserem Sketch nun alles in den ersten Block?
Da wir die MobaTools Lib verwenden wollen, muss die als erstes mal eingebunden werden:

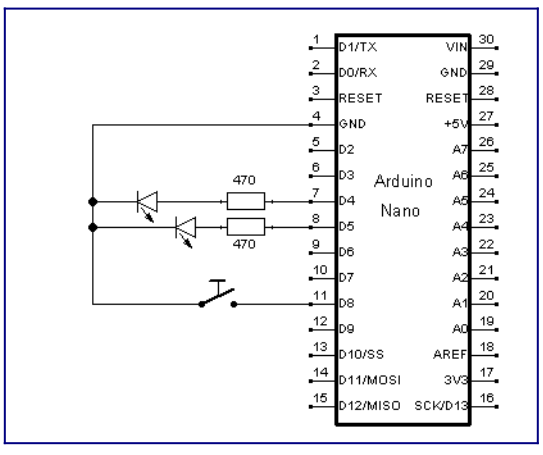
```
#include <MobaTools.h>
```

Wer sie noch nicht auf seinem Rechner hat, kann sie über den Bibliotheksmanager installieren (‘mobatools’ im Suchfeld eingeben)

Dann müssen wir festlegen, welche Pins wir verwenden wollen:

```
// Festlegen der Anschlüsse (Pinnummer) für Ein- und Ausgänge
const byte linksPin = 4;     // linke Lampe
const byte rechtsPin = 5;    // rechte Lampe
const byte belegtPin = 8;    // Eingang 'Zug kommt'
```

Die könnt ihr gegebenenfalls anpassen. Der 'belegtPin' Eingang schaltet in unserer ersten Version den Warnblinker einfach an bzw. aus. Die Schaltung dazu ist recht simpel (auf dem UNO funktioniert es mit den gleichen Pinnr. natürlich genauso):



Von den MobaTools verwenden wir den MoToTimer für die Zeitmessung, und MoToSoftLed für die 'weiche' Ansteuerung der Leds. Timer brauchen wir nur einen, Softleds aber 2 (für rechts und links)

Code:

```
// Benötigte MobaTools Objekte:
MoToTimer StoppUhr; // Zum Messen von Zeiten
MoToSoftLed LampeLinks;
MoToSoftLed LampeRechts;

// Zeiten für den Blinkrhythmus ( in ms )
const int STARTZEIT = 500;
const int BLINKZEIT = 600;
const int FADEZEIT = 200;
```

Für die Lampen des Wechselblinkers habe ich da auch gleich noch die Zeiten festgelegt: wie lange sind beim Start beide gleichzeitig an, und in welchem Rhythmus blinken sie danach. Mit den Zeiten kann man dann auch ein wenig experimentieren. FADEZEIT gibt an, wie lange das auf/Abblenden dauert.

Ja, und dann brauchen wir noch 2 Variable, die wir oben im Code für unseren Zustandsautomaten schon verwendet haben:

```
// Variable
boolean bueBelegt; // = TRUE, wenn sich ein Zug im Bereich des Bue befindet
byte    wBlinkerZustand;    // aktueller Zustand unseres Blinkerautomaten
```

Damit ist unser erster Block komplett.

Im Setup
müssen wir den Eingangspin als 'Eingang' definieren, und unsere 'SoftLed Objekte initiieren. Das Schalten der Pins auf 'Ausgang' macht die MobaTools Lib. Als letztes setzen wir dann noch den Startzustand unseres Blinkautomaten:

```
void setup() {

    // Pins einrichten
    pinMode( belegtPin, INPUT_PULLUP ); // Mit einem Schalter nach GND wird Blinker eingeschaltet
    LampeLinks.attach( linksPin );      // Zuordnung der Softleds zu den Ausgangspins
    LampeRechts.attach( rechtsPin );
    LampeLinks.riseTime( FADEZEIT );    // Überblendzeit einstellen
    LampeRechts.riseTime( FADEZEIT );

    wBlinkerZustand = 1;        // Startzustand des Blinkautomaten.
}
```

Nun noch zum loop()
Der besteht im Wesentlichen ja nur aus der switch-Anweisung mit den case-Blöcken von oben. Ganz am Anfang muss unser Lokführer aber noch kurz Station machen, unseren Eingangspin abfragen und den gelesenen Wert in der Variablen 'bueBelegt' ablegen:

Code: Alles auswählen

```
    bueBelegt = not digitalRead( belegtPin ); // ein LOW-Level am Eingangspin schaltet den Blinker ein
```

Der fertige Sketch
Damit haben wir jetzt den ganzen Sketch zusammem, den wir so in der IDE übersetzen und auf den Arduino laden können:

```
/* Tutorial 'Zeitabläufe ohne delay() '
```

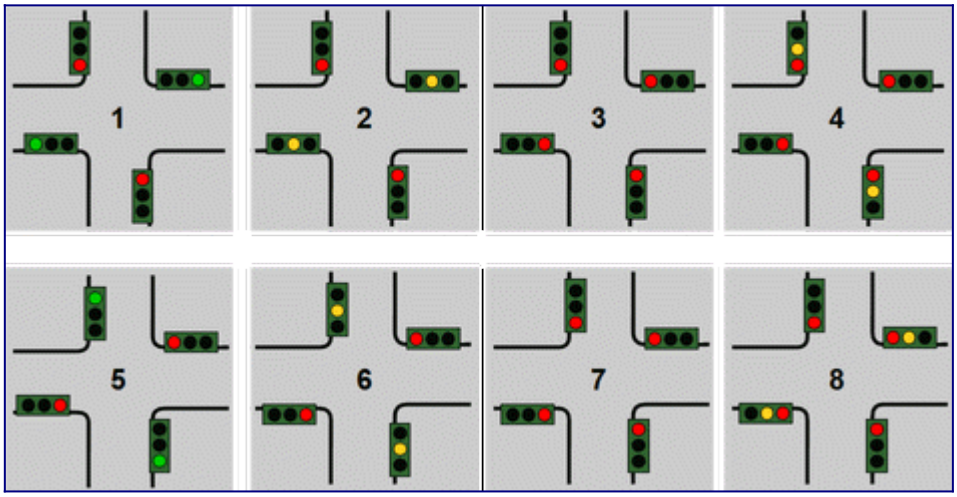
```
 *  Einfacher Wechselblinker ( unbeschrankter Bahnübergang )
 */

#include <MobaTools.h>

// Festlegen der Anschlüsse (Pinnummer) für Ein- und Ausgänge
const byte linksPin = 4;     // linke Lampe
const byte rechtsPin = 5;    // rechte Lampe
const byte belegtPin = 8;    // Eingang 'Zug kommt'

// Benötigte MobaTools Objekte:
MoToTimer StoppUhr; // Zum Messen von Zeiten
MoToSoftLed LampeLinks;
MoToSoftLed LampeRechts;

// Zeiten für den Blinkrhythmus ( in ms )
const int STARTZEIT = 500;
const int BLINKZEIT = 600;
const int FADEZEIT = 200;

// Variable
boolean bueBelegt; // = TRUE, wenn sich ein Zug im Bereich des Bue befindet
byte    wBlinkerZustand;    // aktueller Zustand unseres Blinkerautomaten


void setup() {

    // Pins einrichten
    pinMode( belegtPin, INPUT_PULLUP ); // Mit einem Schalter nach GND wird Blinker eingeschaltet
    LampeLinks.attach( linksPin );      // Zuordnung der Softleds zu den Ausgangspins
    LampeRechts.attach( rechtsPin );
    LampeLinks.riseTime( FADEZEIT );    // Überblendzeit einstellen
    LampeRechts.riseTime( FADEZEIT );

    wBlinkerZustand = 1;        // Startzustand des Blinkautomaten.
}

void loop() {

    // Belegtzusatnd abfragen. Das machen wir hier jetzterstmal ganz einfach: Letztendlich
    // schalten wir mit dem Eingang einfach den Wechselblinker ein und aus.
    // Im realen Praxisfall könnte hier gegebenenfalls auch ein komplexere Logik stehen, um zu
    // erkennen, ob ein Zug sich nähert oder den Bü wieder verlassen hat.
    bueBelegt = not digitalRead( belegtPin ); // ein LOW-Level am Eingangspin schaltet den Blinker ein

    switch ( wBlinkerZustand ) {
      // je nach Wert der Variable wBlinkerZustand, wird die entsprechende 'case' Zeile angesprungen
      case 1: // Grundzustand, abfragen, ob sich ein Zug nähert ( z.B. über einen Belegtmelder )
        if ( bueBelegt ) {
          // Der Zug ist in den Belegtabschnitt vor der Schranke eingefahren
          // beide Lampen einschalten
          LampeLinks.on();
          LampeRechts.on();
          // Stoppuhr mit der Zeit die beide Lampen an sein sollen vorbelegen
          StoppUhr.setTime( STARTZEIT ); // StoppUhr 'aufziehen'
          // in den 2. Zustand wechseln
          wBlinkerZustand = 2; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen
                               // des case 2 ausgeführt
        }
        break; // Ende der Anweisungen für den 1. Zustand
      //---------------------------------------------------------------------
      case 2: // beide Lampen ein, wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um
          LampeRechts.off();
          StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
          // in den 3. Zustand wechseln
          wBlinkerZustand = 3; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen
                               // des case 3 ausgeführt
        }
        break; // Ende der Anweisungen für den 2. Zustand
      //---------------------------------------------------------------------
      case 3: // linke Lampen ist an, rechte aus: wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um, Lampen umschalten
          LampeLinks.off();
          LampeRechts.on();
          StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
          // in den 4. Zustand wechseln
          wBlinkerZustand = 4; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen
                               // des case 4 ausgeführt
        }
        break; // Ende der Anweisungen für den 3. Zustand
```

```
      //--------------------------------------------------------------------
      case 4: // linke Lampen ist aus, rechte an: wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um, Lampen umschalten
          LampeLinks.on();
          LampeRechts.off();
          StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
          // wieder in den 3. Zustand wechseln
          wBlinkerZustand = 3; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen
                               // des case 3 ausgeführt
        }
        // Im 4. Zustand müssen wir auch noch abfragen, ob der Zug den Belegtabschnitt verlassen hat
        if ( not bueBelegt ) {
          // Der Zug ist komplett über den Bü gefahren, die Straße ist wieder frei
          // Beide Lampen ausschalten
          LampeLinks.off();
          LampeRechts.off();
          // wieder in den 1. Zustand wechseln. Eine Zeitüberwachung wird nicht gebraucht
          wBlinkerZustand = 1; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen
                               // des case 1 ausgeführt
        }
        break; // Ende der Anweisungen für den 4. Zustand
    } // Ende des switch-Blockes
}
```

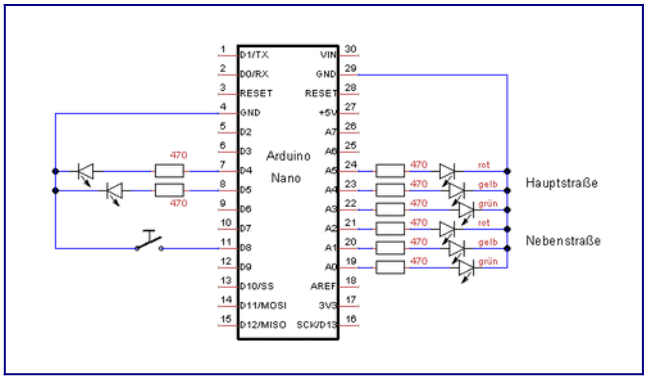
Ich habe versucht den Sketch möglichst ausführlich zu kommentieren, damit man - zusammen mit diesem Tutorial - auch versteht was da abläuft.

Die Ampel

Wir wollten ja auch noch ein Ampelschaltung einbauen. Also werden wir jetzt in gleicher Art und Weise einen Ampelsketch realisieren, und dann beide kombinieren.
Im einfachsten Fall haben wir nur eine Kreuzung, und da sehen unsere Ampelbilder dann so aus:



In der Reihenfolge müssen die Ampeln geschaltet werden (wobei die 'Übergangsphasen' mit dem gelben Licht natürlich kürzer zu sehen sind, als die reinen Rot/Grün Phasen 1+5).
Die 8 Signalbilder sind nun auch unsere Automatenzustände. Auf irgendwelche Eingänge brauchen wir erstmal nicht zu reagieren. Die Zustände werden einfach von 1 bis 8 und wieder bei 1 beginnend zeitgesteuert im Kreis herum geschaltet.
Damit wir mit dem Wechselblinker nicht in Konflikt kommen, müssen wir natürlich unsere Pins entsprechend verteilen. Die jeweils gegenüberliegenden Ampeln einer Straße können parallel geschaltet werden, so dass wir insgesamt 6 Ausgänge benötigen:

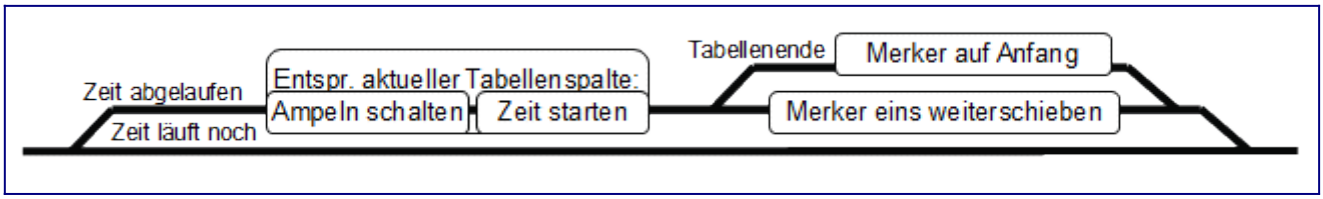


So das wäre nun die Basis für unseren Ampel-Sketch. Erweiterungen sind natürlich möglich. Ich hatte da schonmal an eine per Schalter aktivierbare Nachtschaltung gedacht (gelbes Blinklicht auf der Nebenstraße)

Im Prinzip könnten wir unsere Ampel jetzt genauso umsetzen, wie wir das beim Wechselblinker getan haben. Um Im Eisenbahnbild zu bleiben: Einen Bahnhof mit einer 8-fach Eingangsweiche. Dann in jedem Strang eine Weiche 'Zeit abgelaufen' und im 'abgelaufen' Gleis wieder die Aktionen.
Allerdings würden wir schnell feststellen, dass die Aktionen im Gleis 'Zeit abgelaufen' immer dieselben sind: Zeit neu aufziehen, Ampelausgänge schalten und auf nächsten Zustand stellen. Unterschiedlich sind eigentlich nur die Werte der jeweiligen Aktion: Zeitdauer, ob die Ampelausgänge ein- und ausgeschaltet werden sollen und die Nummer des nächsten Zustands. Wir können den Ablauf da auch eigentlich in einer Tabelle darstellen:

Merker:	⇩							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Zeitdauer Sek	30	3	3	3	20	3	3	3
Hauptstr. rot	-	-	X	X	X	X	X	X
Hauptstr. gelb	-	X	-	-	-	-	-	X
Hauptstr. grün	X	-	-	-	-	-	-	-
Nebenstr. rot	X	X	X	X	-	-	X	X
Nebenstr. gelb	-	-	-	X	-	X	-	-
Nebenstr. grün	-	-	-	-	X	-	-	-

Der Merker oben zeigt dann immer auf den aktuellen Zustand. Wir brauchen für unsere Lok also eigentlich nur ein Gleis mit der Verzweigung 'Zeit abgelaufen'. Die Mehrfachweiche am Anfang sparen wir uns. Dafür hängen wir an der Station obige Tabelle für unseren Lokführer auf. Jedesmal wenn er vorbeikommt, setzt er Timer und Ausgänge entsprechend der Werte in der aktuellen Spalte und schiebt den Merker eins weiter. Ist er am Ende der Tabelle angekommen, schiebt er den Merker wieder auf den Tabellenanfang. Der Ablaufplan ist also recht einfach. (Den Gleiskreis zu schliessen habe ich mir auf dem Bildchen jetzt mal geschenkt):



Im nächsten Schritt geht es dann darum, wie man so eine Tabelle programmtechnisch realisiert.

Wem beim Programmieren schonmal 'Arrays' über den Weg gelaufen sind, der weis auch, wie wir unsere Tabelle realisieren werden.
Arrays - auch Felder genannt - sind eine Reihe von Daten, auf die über einen Index (sozusagen die 'Spaltennummer') zugegriffen werden kann. Ein einfaches Array entspricht so also genau einer Tabellenzeile in unserer Tabelle. Die erste Zeile unserer Tabelle können wir z.B. so definieren:

```
const byte zeitDauer  [] = { 30  , 3  , 3  , 3  , 20 , 3  , 3  , 3  };
```

Die beiden eckigen Klammern sagen dem Compiler, das wir ein Array definieren. Üblicherweise steht dazwischen wieviele Elemente das Feld umfassen soll. Also z.B. [8]. Da wir das Array aber gleich mit Werten füllen, müssen wir die Länge des Feldes nicht angeben. Das erkennt der Compiler an der Zahl der Initiierungswerte zwischen den geschweiften Klammern {..}.
Beim Zugriff müssen wir aber beachten, dass - wie bei Computern üblich - die Zählweise bei 0 beginnt. Der erste Wert in der Zeile (die '30') wird also über den Index '0' angesprochen. Mit der Programmzeile

```
a = zeitDauer[0];
```

weisen wir also der Variablen a den Wert '30' zu. Mit

```
a = zeitDauer[4];
```

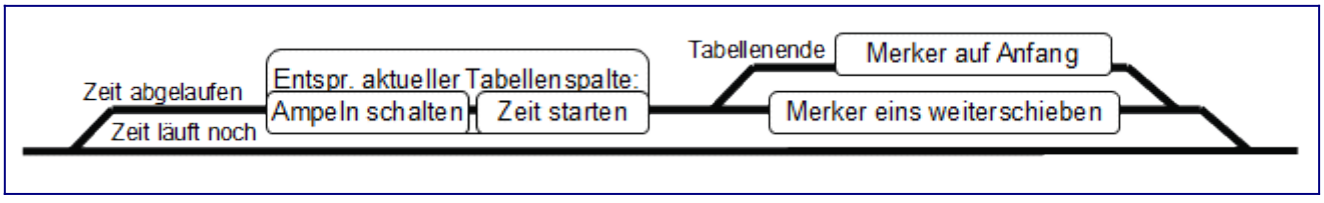
wäre es in unserem Fall der Wert '20'.

Definieren wir nun für jede Tabellenzeile so ein Array, dann sieht die ganze Tabelle so aus:

```
// Tabelle für die Ampelzustände
// Ampelphase:                1  | 2  | 3  | 4  | 5  | 6  | 7  | 8
// Tabellenindex:             0  | 1  | 2  | 3  | 4  | 5  | 6  | 7
const byte zeitDauer  [] = { 30  , 3  , 3  , 3  , 20 , 3  , 3  , 3  };
const bool hauptRot   [] = { OFF ,OFF , ON , ON , ON , ON , ON , ON };
const bool hauptGelb  [] = { OFF , ON ,OFF ,OFF ,OFF ,OFF ,OFF , ON };
const bool hauptGruen [] = {  ON ,OFF ,OFF ,OFF ,OFF ,OFF ,OFF ,OFF };
const bool nebenRot   [] = {  ON , ON , ON , ON ,OFF ,OFF , ON , ON };
const bool nebenGelb  [] = { OFF ,OFF ,OFF , ON ,OFF , ON ,OFF ,OFF };
const bool nebenGruen [] = { OFF ,OFF ,OFF ,OFF , ON ,OFF ,OFF ,OFF };
```

An die Fortgeschrittenen: natürlich ist das nicht die eleganteste, und vor allem nicht die platzsparendste Variante. Aber ich denke, für den Anfang am leichtesten nachzuvollziehen.

Die Umsetzung des gesamten Ablaufs der Ampelphasen entsprechend dem obigen Gleis- bzw Ablaufplans ist dann eigentlich recht kurz:




```
    if( !AmpelTimer.running() ) {
        // Zeit abgelaufen
        // Ampel schalten
        hRot.write( hauptRot[ampelTabIx] );
        hGelb.write( hauptGelb[ampelTabIx] );
        hGruen.write( hauptGruen[ampelTabIx] );
        nRot.write( nebenRot[ampelTabIx] );
        nGelb.write( nebenGelb[ampelTabIx] );
        nGruen.write( nebenGruen[ampelTabIx] );
        // Zeit setzen ( da der Timer in ms rechnet, müssen die Tabellenwerte mit 1000
```

```
        // multipliziert werden )
        AmpelTimer.setTime( zeitDauer[ampelTabIx] * 1000 );
        // Merker weiterrückem
        if( ampelTabIx >= tabMax ) {
            // Tabellenende erreicht
            ampelTabIx = 0;
        } else {
            ampelTabIx++;
        }
    }
```

So, nun müssen wir das nur noch mit den notwendigen Grunddefinitionen und Initiierungen kombinieren und bekommen dann den fertigen Ampelsketch:

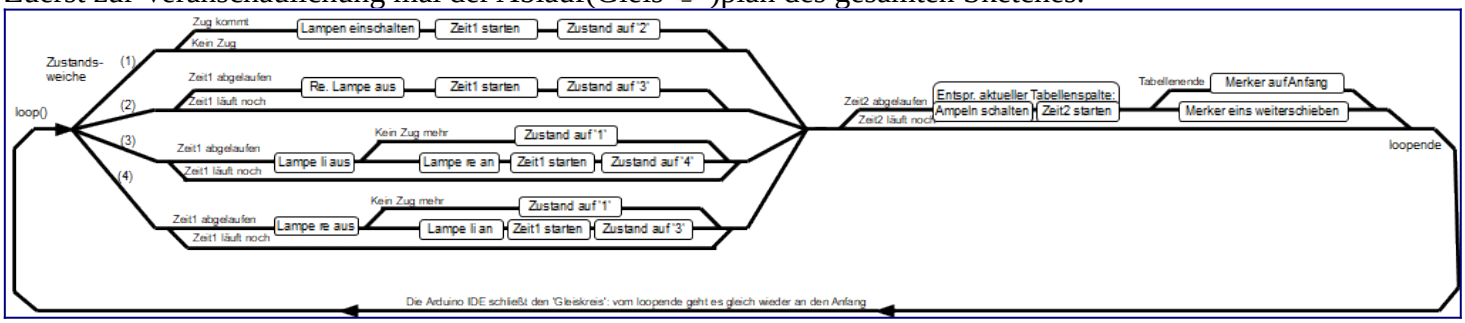
```
// Ampelsteuerung

// Ampelsteuerung an einer einfachen Kreuzung
// Die Ampelphasen werden über eine Tabelle definiert

#include <MobaTools.h>

// Definition der Ausgangspins:
const byte hRotPin   = A0;  // Ampel an der Hauptstraße
const byte hGelbPin  = A1;
const byte hGruenPin = A2;
const byte nRotPin   = A3;  // Ampel an der Nebenstraße
const byte nGelbPin  = A4;
const byte nGruenPin = A5;


// Tabelle für die Ampelzustände
// Ampelphase:               1  | 2  | 3  | 4  | 5  | 6  | 7  | 8
// Tabellenindex:            0  | 1  | 2  | 3  | 4  | 5  | 6  | 7
const byte zeitDauer  [] = { 30  , 3  , 3  , 3  , 20 , 3  , 3  , 3  };
const bool hauptRot   [] = { OFF ,OFF , ON , ON , ON , ON , ON , ON };
const bool hauptGelb  [] = { OFF , ON ,OFF ,OFF ,OFF ,OFF ,OFF , ON };
const bool hauptGruen [] = {  ON ,OFF ,OFF ,OFF ,OFF ,OFF ,OFF ,OFF };
const bool nebenRot   [] = {  ON , ON , ON , ON ,OFF ,OFF , ON , ON };
const bool nebenGelb  [] = { OFF ,OFF ,OFF , ON ,OFF , ON ,OFF ,OFF };
const bool nebenGruen [] = { OFF ,OFF ,OFF ,OFF , ON ,OFF ,OFF ,OFF };

const byte tabMax = sizeof(zeitDauer)-1; // Tabellenindex startet bei 0 ( deshalb die -1 für den
                                         // höchsten Index )
byte ampelTabIx;   // läuft von 0 .. 7

// Einrichten der notwendigen MoBaTools Objekte:
MoToTimer AmpelTimer;
MoToSoftLed hRot;
MoToSoftLed hGelb;
MoToSoftLed hGruen;
MoToSoftLed nRot;
MoToSoftLed nGelb;
MoToSoftLed nGruen;


void setup() {
    // Softled Objekte initiieren
    hRot.attach( hRotPin );
    hRot.riseTime( 200 );
    hGelb.attach( hGelbPin );
    hGelb.riseTime( 200 );
    hGruen.attach( hGruenPin );
    hGruen.riseTime( 200 );
    nRot.attach( nRotPin );
    nRot.riseTime( 200 );
    nGelb.attach( nGelbPin );
    nGelb.riseTime( 200 );
    nGruen.attach( nGruenPin );
    nGruen.riseTime( 200 );

    ampelTabIx = 0;     // Das System startet mit freier Fahrt auf der Hauptstraße

}

void loop() {
    // Tabellengesteuerte Ampel
    if( !AmpelTimer.running() ) {
        // Zeit abgelaufen
        // Ampel schalten
        hRot.write( hauptRot[ampelTabIx] );
        hGelb.write( hauptGelb[ampelTabIx] );
```

```
        hGruen.write( hauptGruen[ampelTabIx] );
        nRot.write( nebenRot[ampelTabIx] );
        nGelb.write( nebenGelb[ampelTabIx] );
        nGruen.write( nebenGruen[ampelTabIx] );
        // Zeit setzen ( da der Timer in ms rechnet, müssen die Tabellenwerte mit 1000
        // multipliziert werden )
        AmpelTimer.setTime( zeitDauer[ampelTabIx] * 1000 );
        // Merker weiterrückem
        if( ampelTabIx >= tabMax ) {
            // Tabellenende erreicht
            ampelTabIx = 0;
        } else {
            ampelTabIx++;
        }
    }
}
```

Der kombinierte Sketch

Nun wollen wir mal drangehen unsere beide Sketches (Wechselblinker und Ampel) zusammenzuführen.

Zuerst zur Veranschaulichung mal der Ablauf(Gleis)plan des gesamten Sketches.



Wir sehen da schon, dass die beiden Ablaufpläne einfach hintereinandergesetzt sind. Und genau so werden wir es später im Code machen.

Wobei der Ablaufplan ja eigentlich immer nur unseren 3. Block, den Loop, visualisiert. Wir hatten ja schonmal davon gesprochen, dass ein Sketch - grob gesehen - aus 3 Blöcken besteht:

1. Definitionen
2. Setup
3. Loop

Diesen Aufbau müssen wir auch bei unserem Gesamtsketch einhalten. Wir können also nicht einfach beide Sketche untereinander schreiben, denn dann wäre jeder Block doppelt vorhanden. Wir müssen die beiden Sketche blockweise zusammenkopieren. Das ergibt dann den folgenden Gesamtsketch:

```
// Wechselblinker + Ampelsteuerung

#include <MobaTools.h>

//vvvvvvvvvvvvvvvvvvvvvvvvv BEGIN WECHSELBLINKER vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
// Definitionen für den Wechselblinker
// Festlegen der Anschlüsse (Pinnummer) für Ein- und Ausgänge
const byte linksPin = 4;     // linke Lampe
const byte rechtsPin = 5;    // rechte Lampe
const byte belegtPin = 8;    // Eingang 'Zug kommt'

// Benötigte MobaTools Objekte:
EggTimer StoppUhr; // Zum Messen von Zeiten
SoftLed LampeLinks;
SoftLed LampeRechts;

// Zeiten für den Blinkrhythmus ( in ms )
const int STARTZEIT = 400;
const int BLINKZEIT = 500;
const int FADEZEIT = 450;
```

```
EggTimer EntPrellung; // Zur Tasterentporellung
const int PRELLZEIT = 100;

// Variable
boolean bueBelegt; // = TRUE, wenn sich ein Zug im Bereich des Bue befindet
byte    wBlinkerZustand;    // aktueller Zustand unseres Blinkerautomaten
//^^^^^^^^^^^^^^^^^^^^^^^^^ ENDE WECHSELBLINKER ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

//vvvvvvvvvvvvvvvvvvvvvvvv BEGIN AMPELSTEUERUNG vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
// Ampelsteuerung an einer einfachen Kreuzung
// Die Ampelphasen werden über eine Tabelle definiert

// Definitionen für die Ampelschaltung:
// Definition der Ausgangspins:
const byte hRotPin   = A0;  // Ampel an der Hauptstraße
const byte hGelbPin  = A1;
const byte hGruenPin = A2;
const byte nRotPin   = A3;  // Ampel an der Nebenstraße
const byte nGelbPin  = A4;
const byte nGruenPin = A5;


// Tabelle für die Ampelzustände
// Ampelphase:               1  | 2  | 3  | 4  | 5  | 6  | 7  | 8
// Tabellenindex:            0  | 1  | 2  | 3  | 4  | 5  | 6  | 7
const byte zeitDauer  [] = { 30  , 3  , 3  , 3  , 20 , 3  , 3  , 3  };
const bool hauptRot   [] = { OFF ,OFF , ON , ON , ON , ON , ON , ON };
const bool hauptGelb  [] = { OFF , ON ,OFF ,OFF ,OFF ,OFF , ON , ON };
const bool hauptGruen [] = {  ON ,OFF ,OFF ,OFF ,OFF ,OFF ,OFF ,OFF };
const bool nebenRot   [] = {  ON , ON , ON , ON ,OFF ,OFF , ON , ON };
const bool nebenGelb  [] = { OFF ,OFF ,OFF , ON ,OFF , ON ,OFF ,OFF };
const bool nebenGruen [] = { OFF ,OFF ,OFF ,OFF , ON ,OFF ,OFF ,OFF };

const byte tabMax = sizeof(zeitDauer)-1; // Tabellenindex startet bei 0 ( deshalb die -1 für den höchsten
Index )
byte ampelTabIx;   // läuft von 0 .. 7

// Einrichten der notwendigen MoBaTools Objekte:
EggTimer AmpelTimer;
SoftLed hRot;
SoftLed hGelb;
SoftLed hGruen;
SoftLed nRot;
SoftLed nGelb;
SoftLed nGruen;
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ENDE AMPELSTEUERUNG ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//---------------------------------------------------------------------------

void setup() {
    //vvvvvvvvvvvvvvvvvvvvvvvvv BEGIN WECHSELBLINKER vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
    // Initiierungen Wechselblinker
    // Pins einrichten
    pinMode( belegtPin, INPUT_PULLUP ); // Mit einem Schalter nach GND wird Blinker eingeschaltet
    LampeLinks.attach( linksPin );      // Zuordnung der Softleds zu den Ausgangspins
    LampeRechts.attach( rechtsPin );
    LampeLinks.riseTime( FADEZEIT );    // Überblendzeit einstellen
    LampeRechts.riseTime( FADEZEIT );

    wBlinkerZustand = 1;        // Startzustand des Blinkautomaten.
    //^^^^^^^^^^^^^^^^^^^^^^^^^ ENDE WECHSELBLINKER ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

    //vvvvvvvvvvvvvvvvvvvvvvvv BEGIN AMPELSTEUERUNG vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
    //Initiierungen für die Ampel
    // Softled Objekte initiieren
    hRot.attach( hRotPin );
    hRot.riseTime( 200 );
    hGelb.attach( hGelbPin );
    hGelb.riseTime( 200 );
    hGruen.attach( hGruenPin );
    hGruen.riseTime( 200 );
    nRot.attach( nRotPin );
    nRot.riseTime( 200 );
    nGelb.attach( nGelbPin );
    nGelb.riseTime( 200 );
    nGruen.attach( nGruenPin );
    nGruen.riseTime( 200 );

    ampelTabIx = 0;    // Das System startet mit freier Fahrt auf der Hauptstraße
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ENDE AMPELSTEUERUNG ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
//---------------------------------------------------------------------------
}

void loop() {
```

```
    //vvvvvvvvvvvvvvvvvvvvvvvvvv BEGIN WECHSELBLINKER vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
    // Wechselblinkerautomat
    // Belegtzusatnd abfragen. Das machen wir hier jetzterstmal ganz einfach: Letztendlich
    // schalten wir mit dem Eingang einfach den Wechselblinker ein und aus.
    // Im realen Praxisfall könnte hier gegebenenfalls auch ein komplexere Logik stehen, um zu
    // erkennen, ob ein Zug sich nähert oder den Bü wieder verlassen hat.
    //if ( not EntPrellung.running() ) {
    //    EntPrellung.setTime( PRELLZEIT );
        bueBelegt = not digitalRead( belegtPin ); // ein LOW-Level am Eingangspin schaltet den Blinker ein
    //}

    switch ( wBlinkerZustand ) {
      // je nach Wert der Variable wBlinkerZustand, wird die entsprechende 'case' Zeile angesprungen
      case 1: // Grundzustand, abfragen, ob sich ein Zug nähert ( z.B. über einen Belegtmelder )
        if ( bueBelegt ) {
          // Der Zug ist in den Belegtabschnitt vor der Schranke eingefahren
          // beide Lampen einschalten
          LampeLinks.on();
          LampeRechts.on();
          // Stoppuhr mit der Zeit die beide Lampen an sein sollen vorbelegen
          StoppUhr.setTime( STARTZEIT ); // StoppUhr 'aufziehen'
          // in den 2. Zustand wechseln
          wBlinkerZustand = 2; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen des case 2
ausgeführt
        }
        break; // Ende der Anweisungen für den 1. Zustand
      //--------------------------------------------------------------------------
      case 2: // beide Lampen ein, wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um
          LampeRechts.off();
          StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
          // in den 3. Zustand wechseln
          wBlinkerZustand = 3; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen des case 3
ausgeführt
        }
        break; // Ende der Anweisungen für den 2. Zustand
      //--------------------------------------------------------------------------
      case 3: // linke Lampen ist an, rechte aus: wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um, Lampen umschalten
          LampeLinks.off();
          // Im 3. Zustand müssen wir auch noch abfragen, ob der Zug den Belegtabschnitt verlassen hat
            if ( not bueBelegt ) {
                // Der Zug ist komplett über den Bü gefahren, die Straße ist wieder frei
                // Beide Lampen sind aus!
                // wieder in den 1. Zustand wechseln. Eine Zeitüberwachung wird nicht gebraucht
                wBlinkerZustand = 1; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen des case 1
ausgeführt
            }  else {
                LampeRechts.on();
                StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
                // wieder in den 4. Zustand wechseln
                wBlinkerZustand = 4; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen des case 3
ausgeführt
            }
        }
        break; // Ende der Anweisungen für den 3. Zustand
      //--------------------------------------------------------------------------
      case 4: // linke Lampen ist aus, rechte an: wir fragen ab, ob die Stoppuhr abgelaufen ist.
        if ( not StoppUhr.running() ) {
          // Die Stoppuhr läuft nicht mehr, die Zeit ist um, Lampen umschalten
          LampeRechts.off();
          // Im 4. Zustand müssen wir auch noch abfragen, ob der Zug den Belegtabschnitt verlassen hat
            if ( not bueBelegt ) {
                // Der Zug ist komplett über den Bü gefahren, die Straße ist wieder frei
                // Beide Lampen sind aus!
                // wieder in den 1. Zustand wechseln. Eine Zeitüberwachung wird nicht gebraucht
                wBlinkerZustand = 1; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen des case 1
ausgeführt
            }  else {
                LampeLinks.on();
                StoppUhr.setTime( BLINKZEIT ); // Stoppuhr 'aufziehen'
                // wieder in den 3. Zustand wechseln
                wBlinkerZustand = 3; // Ab dem nächsten loop-Durchlauf werden nur die Anweisungen des case 3
ausgeführt
            }
        }
        break; // Ende der Anweisungen für den 4. Zustand
    } // Ende des switch-Blockes
    //^^^^^^^^^^^^^^^^^^^^^^^^ ENDE WECHSELBLINKER ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
    //vvvvvvvvvvvvvvvvvvvvvvvv BEGIN AMPELSTEUERUNG vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
    // Tabellengesteuerte Ampel
    if( !AmpelTimer.running() ) {
        // Zeit abgelaufen
        // Ampel schalten
        hRot.write( hauptRot[ampelTabIx] );
        hGelb.write( hauptGelb[ampelTabIx] );
        hGruen.write( hauptGruen[ampelTabIx] );
        nRot.write( nebenRot[ampelTabIx] );
        nGelb.write( nebenGelb[ampelTabIx] );
        nGruen.write( nebenGruen[ampelTabIx] );
        // Zeit setzen ( da der Timer in ms rechnet, müssen die Tabellenwerte mit 1000
        // multipliziert werden )
        AmpelTimer.setTime( zeitDauer[ampelTabIx] * 1000 );
        // Merker weiterrückem
        if( ampelTabIx >= tabMax ) {
            // Tabellenende erreicht
            ampelTabIx = 0;
        } else {
            ampelTabIx++;
        }
    }
    //^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ENDE AMPELSTEUERUNG ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}
```

Voila - unser Arduino bedient Ampel und Wechselblinker 'parallel' und vollkommen unabhängig voneinander. Auch das lastet den Arduino noch bei weitem nicht aus - unsere 'Arduino-Lok' flitzt immer noch hauptsächlich durch die freien Gleise im Kreis, ohne etwas zu tun. 'HigSpeed'-Däumchendrehen 
P.S. Den Wechselblinker habe ich noch ein bisschen 'getuned' . Auch im Gesamt-Ablaufplan ist das eingeflossen. Erkennt ihr, was anders ist, und wie es sich auswirkt?

Das Ziel dieses Tutorials ist damit eigentlich erreicht: Wenn man beim Programmieren auf 'delay()' verzichtet, kann man erreichen, dass der Arduino mehrere Dinge unabhängig und scheinbar 'parallel' erledigt. Ich hoffe, es war verständlich, und ihr könnt mit diesem Verfahren nun auch andere Projekte in Angriff nehmen.