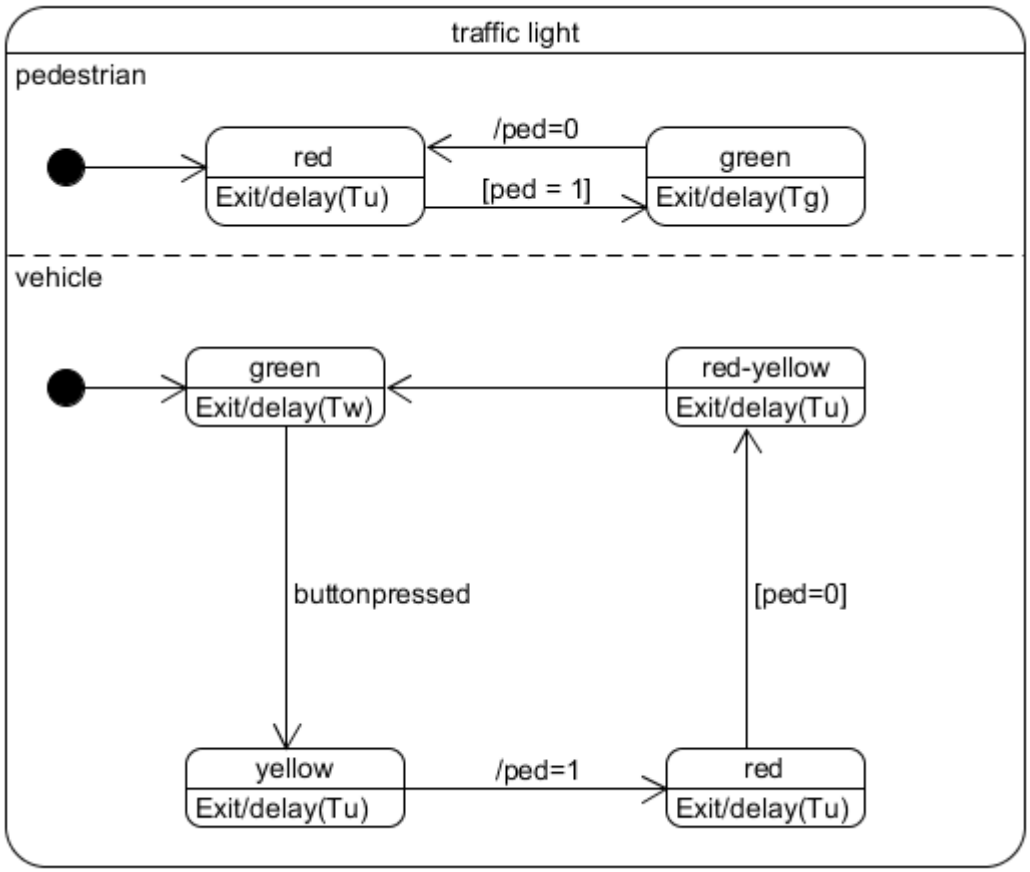
Embedded Systems - Praktikum 8

Malte Riechmann, André Kirsch

Aufgabe 1

Statechart



Implementierung

In der Implementierung geben zwei *uint8_t* Werte an, die angeben, welche der LEDs der Ampeln angeschaltet sind. Zum Umschalten der LEDs nutzen wir die Methode *updateLights*.

```
void updateLights() {
  uint8_t lightStates = pedTrafficLightStates << 3;
  lightStates |= vehTrafficLightStates;

  for (int i = 0; i < sizeof(pinLights); i++) {
    digitalWrite(pinLights[i], lightStates & 1);
    lightStates >>= 1;
  }
}
```

Zuerst verknüpfen wir die beiden Werte, die angeben, welche LEDs aktiv sind. Danach gehen wir einen Array mit den Pins der LEDs durch, schalten diese entweder an oder aus, indem wir sie mit einer 1 &-verknüpfen und verschieben die Bits der *lightStates* Variable um 1 nach rechts.

In der *loop* Funktion wechseln wir zwischen den States der Ampeln und updaten die LEDs. Dabei wird zuerst der Status der Fahrzeug Ampel umgeschaltet:

```
switch(vehTrafficLightStates) {
  case 0b001:
    if (btnTrafficLights.state()) {
      delay(tw_time);
      vehTrafficLightStates <<= 1;
    }
    break;
  case 0b010:
    delay(tu_time);
      vehTrafficLightStates <<= 1;
    ped = 1;
    break;
  case 0b100:
    if(ped == 0) {
      delay(tu_time);
      vehTrafficLightStates |= 0b010;
    }
    break;
  case 0b110:
    delay(tu_time);
    vehTrafficLightStates >>= 2;
    break;
}
```

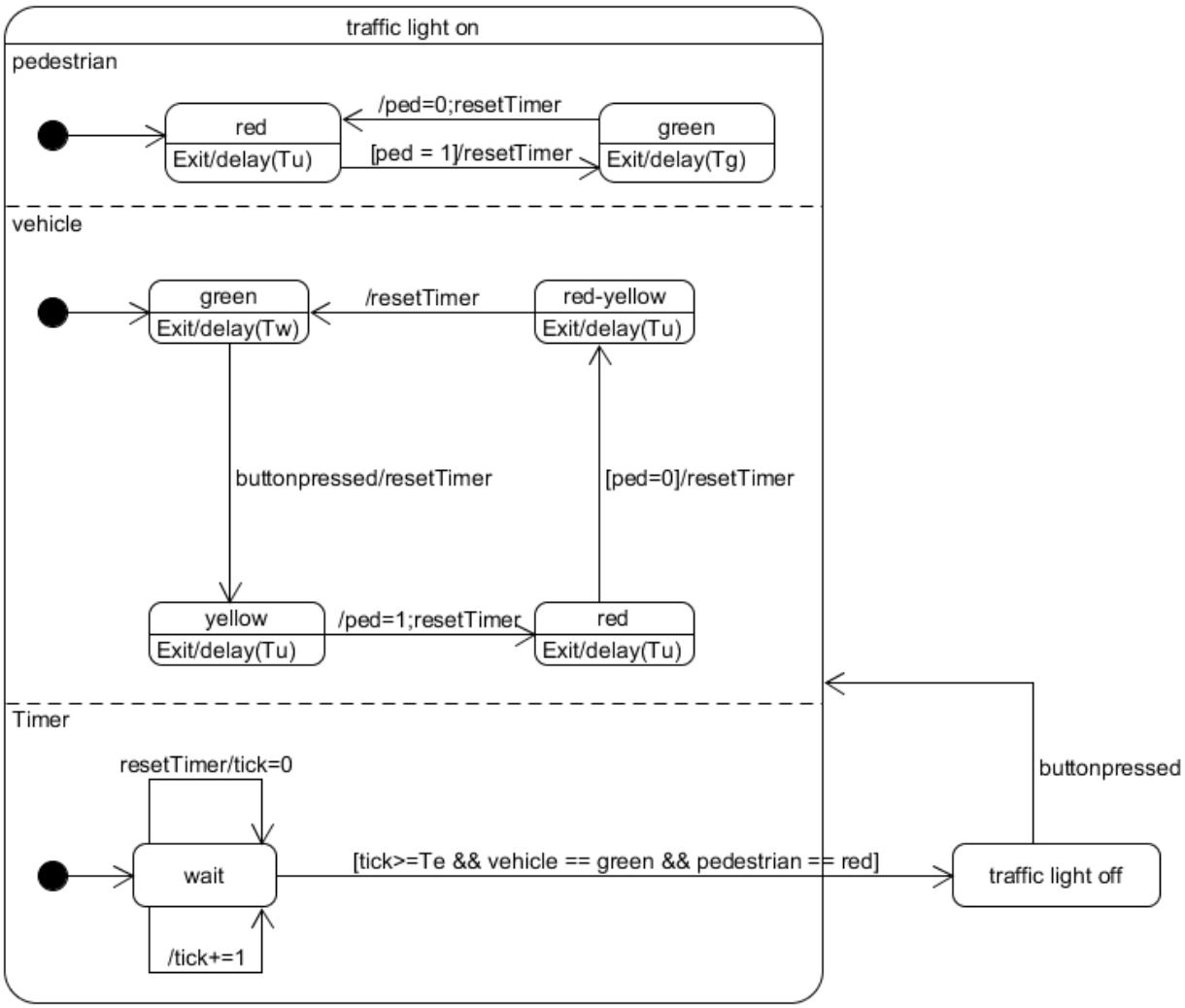
Dabei werden die Bits des Ampelstatus richtig gesetzt, nachdem eine gewisse Zeit abgewartet wurde. Im Fall 0b001 werden die Bits nur gesetzt, wenn der Button gedrückt wurde. Ansonsten zeigt die Ampel das grüne Licht an. Fall 0b010 setzt zusätzlich *ped* auf 1, welches im loop auch die Fußgängerampel umschalten lässt. Bei dem Fall 0b100 soll die Ampel auf rot-gelb geschaltet werden. Dies ist aber nur möglich, wenn *ped* wieder auf 0 gesetzt wurde, also die Fußgängerampel einmal grün und dann wieder rot geleuchtet hat. Der Fall 0b110 setzt die Ampel nur wieder auf den Ausgangspunkt zurück.

```
switch(pedTrafficLightStates) {
  case 0b01:
    if(ped) {
      delay(tu_time);
      pedTrafficLightStates <<= 1;
    }
    break;
  case 0b10:
    delay(tg_time);
    pedTrafficLightStates >>= 1;
    ped = 0;
    break;
}
```

Auch bei der Fußgängerampel werden die Bits nach einer gewissen Zeit richtig gesetzt. Dabei zeigt die Ampel standardmäßig rot und wird nur dann auf grün gesetzt, wenn *ped* gesetzt wurde. Im zweiten Fall wird *ped* wieder zurück auf null gesetzt.

Aufgabe 2

Statechart



Implementierung

Für die zweite Aufgabe haben wir eine Klasse Timer im Singleton-Pattern mit den Methoden *setCallback*, *setTimer* und *resetTimer* implementiert.

Im Konstruktor aktivieren und konfigurieren wir den Timer sowie den Master Interrupt Handler. Genauso berechnen wir die Zeit für eine Periode des Interupts.

```
Timer::Timer() {
  SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
  TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT);
  IntMasterEnable();

  period = SysCtlClockGet() / 2;
}
```

In der Methode *setCallback* setzen wir die Callback-Funktion. Dazu müssen wir den Timer und Timer Interrupt Handler deaktivieren sowie den Timer Interrupt leeren, um danach die Callback-Funktion setzen zu können. Danach aktivieren wir alles wieder.

```
void Timer::setCallback(void (*onTimeUp)(void)) {
  TimerDisable(TIMER0_BASE, TIMER_BOTH);
  TimerIntDisable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
  TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
  TimerIntRegister(TIMER0_BASE, TIMER_BOTH, onTimeUp);
  IntEnable(INT_TIMER0A);
  TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
  TimerEnable(TIMER0_BASE, TIMER_BOTH);
}
```

In der *setTimer* Methode setzen wir die Zeit neu und resetten den Timer, indem wir die *resetTimer* Methode aufrufen. Dabei wird der Timer Interrupt geleert und der Timer deaktiviert, um mit *TimerLoadSet* die neue Zeit zu setzen. Danach wird der Timer wieder aktiviert.

Der Inhalt der *loop*-Methode wurde stark gekürzt. Die switch-Statements wurden in einzelne Funktionen ausgelagert, die dem Timer als Callback-Funktion mitgegeben werden. Dafür wurde die *delay* Funktion mit der passenden Timer Methode ersetzt.

```
void onNextVehLightState() {
  switch(vehTrafficLightStates) {
    case 0b001:
      vehTrafficLightStates <<= 1;
      timer.setTimer(tu_time);
      break;
    case 0b010:
      vehTrafficLightStates <<= 1;
      ped = 1;
      timer.setCallback(onNextPedLightState);
      timer.resetTimer();
      break;
```

```
    case 0b100:
      if(ped == 0) {
        vehTrafficLightStates |= 0b010;
        timer.resetTimer();
      }
      break;
    case 0b110:
      vehTrafficLightStates >>= 2;
      timer.setCallback(onSleep);
      timer.setTimer(te_time);
      break;
  }
}

void onNextPedLightState() {
  switch(pedTrafficLightStates) {
    case 0b01:
      if(ped) {
        pedTrafficLightStates <<= 1;
        timer.setTimer(tg_time);
      }
      break;
    case 0b10:
      pedTrafficLightStates >>= 1;
      ped = 0;
      timer.setCallback(onNextVehLightState);
      timer.setTimer(tu_time);
      break;
  }
}
```

In der *loop* Funktion werden nun immer die LEDs geupdatet und der Button Status abgefragt. Sollte der Button aktiviert sein und sich die Verkehrampel im Status 1 befindet, werden die Callback Funktion und die Zeitspanne neu gesetzt und der Timer neu gestartet.

Für das Betreten des Energie-Spar-Modus haben wir eine Funktion *onSleep* geschrieben. Diese wird als Callback-Funktion aufgerufen, damit der Mikrocontroller in den Energie-Spar-Modus geht. Dabei werden alle LEDs ausgeschaltet und die Funktion *HibernateRequest* zum ausschalten aufgerufen.

```
void onSleep() {
  for (int i = 0; i < sizeof(pinLights); i++) {
    digitalWrite(pinLights[i], LOW);
  }

  HibernateRequest();
}
```

Damit aber in den Energie-Spar-Modus schalten und daraus wieder aufwachen möglich ist, werden in der *setup* Funktion noch einige Einstellungen getroffen.

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_HIBERNATE);
HibernateEnableExpClk(SysCtlClockGet());
HibernateGPIORetentionEnable();
HibernateWakeSet(pinTrafficLights);
```

Über *SysCtlPeripheralEnable* wird der Hibernate Modus aktiviert. Danach wird der Hibernate Modus eingestellt. Besonders wichtig ist, dass über *HibernateWakeSet* der Wake-Up Pin gesetzt wird.