Embedded Systems

Ampel-Timerinterrupt

Praktikum 8

Fachhochschule Bielefeld
Campus Minden
Studiengang Informatik

Beteiligte Personen:

Name	Matrikelnummer
Jan-Hendrik Sünderkamp	1153536
Peter Dick	1050185

---

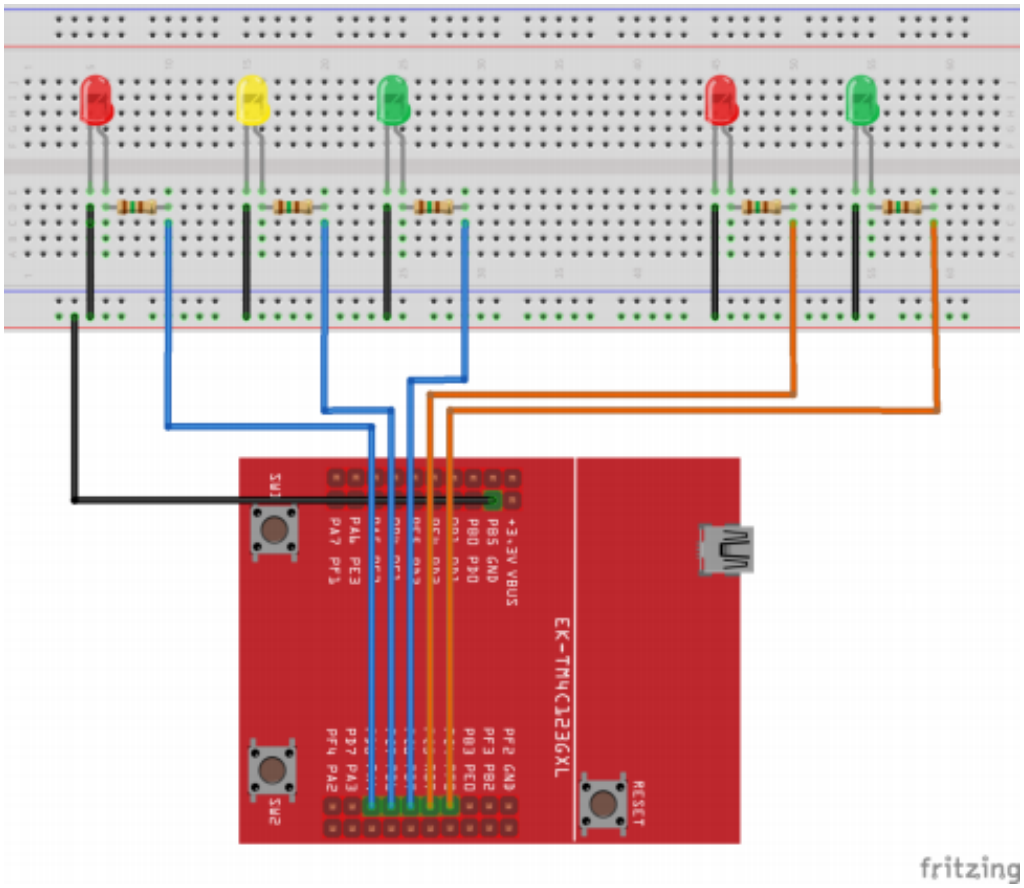
24. Juni 2018

Inhaltsverzeichnis

1 Vorbereitung

Informieren Sie sich über Timer-Interrupts in dem Dokument TivaWare^{TM} Peripheral Driver Library und lesen Sie im Workbook[1] Lab 4 und Lab 6 nach.

Schlagen Sie die entsprechenden Stellen für den Hibernation Mode im Workbook nach. Verwenden Sie die Schaltung gemäß dem Schaltbild 3. Verwenden Sie 150 Ω Widerstände. Für den Knopfdruck verwenden Sie den auf dem Board verbauten Knopf mit der Beschriftung SW2 (im Code ”PUSH2”). Die Schaltung entspricht einem Aufbau einer Fußgängerampel und einer Fahrzeugampel, wie man sie aus dem Straßenverkehr kennt.



Schaltbild 3

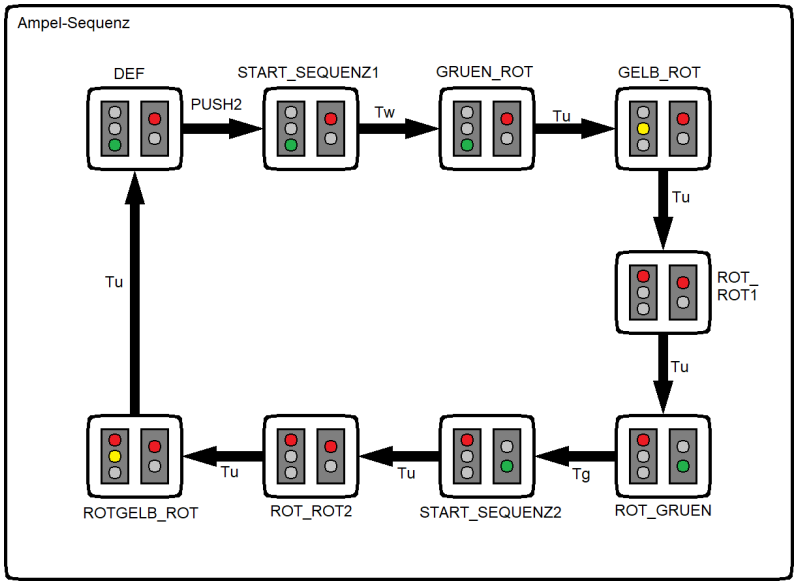
1 http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/GSW-TM4C123G-LaunchPad/TM4C123G_LaunchPad_Workshop_Workbook.pdf

2 Aufgabe 1

Die Funktionalität der Schaltung ist mittels des LaunchPads wie folgt umzusetzen:

- Im Default-Zustand ist die Fahrzeugampel grün, die Fußgängerampel rot.
- Mit einem Knopfdruck startet nach dem Ablauf einer ersten Zeitspanne (T_w) eine Umschaltsequenz der Ampeln. Diese Sequenz beginnt mit der Umschaltung der Fahrzeugampel von grün über gelb auf rot. Danach erfolgt eine Umschaltung der Fußgängerampel von rot auf grün. Jede Umschaltung dauert dabei eine zweite Zeitspanne (T_u). Nach einer dritten Zeitspanne (T_g) schalten die beiden Ampeln zurück. Diesmal erfolgt zuerst das Umschalten der Fußgängerampel von grün auf rot, dann der Fahrzeugampel von rot über gelb-rot auf grün. Jede Umschaltung der Ampeln dauert dabei wieder die zweite Zeitspanne (T_u).

Modellieren Sie das gewünschte Verhalten des Systems mittels eines Statecharts.



- PUSH2: Button SW2 wurde gedrückt.
- Tw: Zeitspanne Tw ist abgelaufen.
- Tu: Zeitspanne Tu ist abgelaufen.
- Tg: Zeitspanne Tg ist abgelaufen.

Implementieren Sie ausgehend von Ihrer Modellierung deren Funktionalität. Ihre Modellierung der Zustände muss in der Implementierung klar wieder zu finden sein. Die Verwendung von Timer-Interrupts ist für diese Aufgabe nicht erforderlich.

Klasse Led:

```
template <const uint8_t PORT>
class Led {
  public:
  Led(const uint8_t f_ledState = LOW) : m_ledState(f_ledState) {
    pinMode(PORT, OUTPUT);
    digitalWrite(PORT, m_ledState);
  }
  void an() {
    m_ledState = HIGH;
    digitalWrite(PORT, m_ledState);
  }
  void aus() {
    m_ledState = LOW;
    digitalWrite(PORT, m_ledState);
  }
  private:
  uint8_t m_ledState;
};
```

Klasse Button:

```
template <const uint8_t PORT>
class Button {
  public:
    Button(const uint8_t f_btnState = LOW) : m_btnState(
   f_btnState) {
      pinMode(PORT, INPUT);
    }

    uint8_t state() {
      m_btnState = digitalRead(PORT);
      return m_btnState;
    }

  private:
    uint8_t m_btnState;
};
```

Ampeln:

```
struct Fahrzeugampel {
  Led<LedPinRot2> rot;
  Led<LedPinGelb2> gelb;
  Led<LedPinGruen2> gruen;
};
```

```
struct Fussgaengerampel {
  Led<LedPinRot1> rot;
  Led<LedPinGruen1> gruen;
};
```

Zustände:

```
enum Status {
  DEF,
  START_SEQUENZ1,
  GRUEN_ROT,
  GELB_ROT,
  ROT_ROT1,
  ROT_GRUEN,
  START_SEQUENZ2,
  ROT_ROT2,
  ROTGELB_ROT
};
```

Implementierung Statechart

```
Status zustand = DEF;
```

```
void loop() {
  switch(zustand) {
    case DEF:
      Serial.println("DEF");
      fahrzeugampel.gelb.aus();
      fahrzeugampel.rot.aus();
      fahrzeugampel.gruen.an();
      fussgaengerampel.rot.an();
      if(button.state() == LOW) {
        zustand = START_SEQUENZ1;
      }
      break;
    case START_SEQUENZ1:
      Serial.println("START_SEQUENZ1");
      delay(Tw);
      zustand = GRUEN_ROT;
      break;
    case GRUEN_ROT:
      Serial.println("GRUEN_ROT");
      delay(Tu);
      zustand = GELB_ROT;
      break;
    case GELB_ROT:
      Serial.println("GELB_ROT");
```

```
        fahrzeugampel.gruen.aus();
        fahrzeugampel.gelb.an();
        delay(Tu);
        zustand = ROT_ROT1;
        break;
    case ROT_ROT1:
        Serial.println("ROT_ROT1");
        fahrzeugampel.gelb.aus();
        fahrzeugampel.rot.an();
        delay(Tu);
        zustand = ROT_GRUEN;
        break;
    case ROT_GRUEN:
        Serial.println("ROT_GRUEN");
        fussgaengerampel.rot.aus();
        fussgaengerampel.gruen.an();
        delay(Tg);
        zustand = START_SEQUENZ2;
        break;
    case START_SEQUENZ2:
        Serial.println("START_SEQUENZ2");
        delay(Tu);
        zustand = ROT_ROT2;
        break;
    case ROT_ROT2:
        Serial.println("ROT_ROT2");
        fussgaengerampel.gruen.aus();
        fussgaengerampel.rot.an();
        delay(Tu);
        zustand = ROTGELB_ROT;
        break;
    case ROTGELB_ROT:
        Serial.println("ROTGELB_ROT");
        fahrzeugampel.gelb.an();
        delay(Tu);
        zustand = DEF;
        break;
  }
}
```

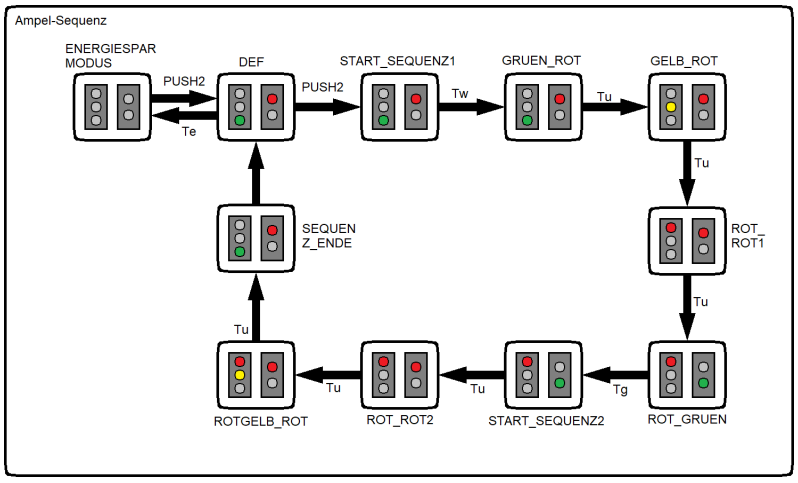
3 Aufgabe 2

Fügen Sie dem Zustandsautomaten und dem Programm aus Aufgabe 1 die Eigenschaft hinzu, dass das Board (und die Ampeln) nach einer Zeit (T_e) ohne Knopfdruck in einen Energiesparmodus geht, aus welchem es bei Knopfdruck aufwacht. Der Energiesparmodus darf nur erreicht werden, wenn die Fahrzeugampel grün (und die Fußgängerampel rot) ist. Dabei sollen auch alle LEDs ausgeschaltet werden.

Verwenden Sie für diese Aufgabe einen Timer-Interrupt. Schreiben Sie dazu eine Klasse *Timer*, die intern den Timer-Interrupt benutzt. Die Klasse *Timer* muss u.a. die Methoden *setTimer*(Zeitspanne) und *resetTimer*() bereitstellen. Der Ablauf einer mit *setTimer*(Zeitspanne) eingestellten Zeitspanne muss bei der Ereignisverarbeitung benutzt werden. Implementieren Sie für die Klasse *Timer* das Singleton-Pattern, so dass nur eine Instanz der Klasse *Timer* existiert. Die Funktion ISR für den Timer-Interrupt kann außerhalb der Klasse *Timer* sein. Die Funktion ISR kann über das Singleton-Pattern auf die Timer-Klasse zugreifen.
Statt die Interrupt Vector Tabelle zu ändern, wie es im Workbook gemacht wird, verwenden Sie die Funktion "TimerIntRegister" von Tiva (siehe Driver Lib Doku).

Das Programm darf nicht die Funktion *delay*() der Energia-Bibliothek verwenden. Sämtliche Zeitabläufe sind über die Klasse Timer durchzuführen.
Für die Zeitspannen soll gelten: $T_e > T_w \geqq T_g > T_u$. Die Zeitspannen sollen auf halbe Sekunden genau einstellbar sein.

Tipp: Sollte sich das Board beim Entwickeln nicht mehr flashen lassen, so ist es wahrscheinlich noch im Energiesparmodus. Starten Sie das Board neu, drücken Sie den SW2-Button (das Board verlässt den Energiesparmodus) und flashen Sie das Board.



- PUSH2: Button SW2 wurde gedrückt.
- Tw: Zeitspanne Tw ist abgelaufen.

- Tu: Zeitspanne Tu ist abgelaufen.
- Tg: Zeitspanne Tg ist abgelaufen.
- Te: Zeitspanne Te ist abgelaufen.

Klasse Timer:

```
class Timer {
  public:
  static Timer& getTimer() {
    static Timer timer;
    return timer;
  }

  void setTimer(int zeitspanne) {
    this->zeitspanne = zeitspanne;
    TimerEnable(TIMER0_BASE, TIMER_A);
  }

  unsigned long getZeitspanne() {
    return zeitspanne;
  }

  void resetTimer() {
    count = 0;
  }

  unsigned long getCount() {
    return count;
  }

  void subCount() {
    count--;
  }

  void addCount() {
    count++;
  }

  private:
  Timer() {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    IntMasterEnable();
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    unsigned long p = 40000000;            //  40MHz
    TimerLoadSet(TIMER0_BASE, TIMER_A, p);
```

```
    IntEnable(INT_TIMER0A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    TimerIntRegister(TIMER0_BASE, TIMER_A, ISR);
  }
  unsigned long count = 0;
  unsigned long zeitspanne;
  Timer(const Timer&);
  Timer & operator = (const Timer&);
};
```

Zustände:

```
enum Status {
  DEF,
  START_SEQUENZ1,
  GRUEN_ROT,
  GELB_ROT,
  ROT_ROT1,
  ROT_GRUEN,
  START_SEQUENZ2,
  ROT_ROT2,
  ROTGELB_ROT,
  ENERGIESPARMODUS,
  SEQUENZ_ENDE
};
```

Implementierung Funktion ISR:

```
void ISR(void) {
  bool buttonaktiv = button.state() == LOW;
  Timer& t = Timer::getTimer();
  TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
  if(t.getCount() == t.getZeitspanne() || zustand == DEF ||
   zustand == ENERGIESPARMODUS) {
    switch(zustand) {
    case DEF:
      Serial.println("DEF");
      fahrzeugampel.gruen.an();
      fussgaengerampel.rot.an();
      if(t.getCount() == Te - 1) {
        zustand = ENERGIESPARMODUS;
        t.resetTimer();
        t.setTimer(1);
      }
      if(buttonaktiv) {
        zustand = START_SEQUENZ1;
        t.resetTimer();
        t.setTimer(1);
```

```
    }
    break;
  case START_SEQUENZ1:
    Serial.println("START_SEQUENZ1");
    zustand = GRUEN_ROT;
    t.resetTimer();
    t.setTimer(Tw);
    break;
  case GRUEN_ROT:
    Serial.println("GRUEN_ROT");
    zustand = GELB_ROT;
    t.resetTimer();
    t.setTimer(Tu);
    break;
  case GELB_ROT:
    Serial.println("GELB_ROT");
    fahrzeugampel.gruen.aus();
    fahrzeugampel.gelb.an();
    zustand = ROT_ROT1;
    t.resetTimer();
    t.setTimer(Tu);
    break;
  case ROT_ROT1:
    Serial.println("ROT_ROT1");
    fahrzeugampel.gelb.aus();
    fahrzeugampel.rot.an();
    zustand = ROT_GRUEN;
    t.resetTimer();
    t.setTimer(Tu);
    break;
  case ROT_GRUEN:
    Serial.println("ROT_GRUEN");
    fussgaengerampel.rot.aus();
    fussgaengerampel.gruen.an();
    zustand = START_SEQUENZ2;
    t.resetTimer();
    t.setTimer(Tg);
    break;
  case START_SEQUENZ2:
    Serial.println("START_SEQUENZ2");
    zustand = ROT_ROT2;
    t.resetTimer();
    t.setTimer(Tu);
    break;
  case ROT_ROT2:
    Serial.println("ROT_ROT2");
```

```
        fussgaengerampel.gruen.aus();
        fussgaengerampel.rot.an();
        zustand = ROTGELB_ROT;
        t.resetTimer();
        t.setTimer(Tu);
        break;
      case ROTGELB_ROT:
        Serial.println("ROTGELB_ROT");
        fahrzeugampel.gelb.an();
        zustand = SEQUENZ_ENDE;
        t.resetTimer();
        t.setTimer(Tu);
        break;
      case SEQUENZ_ENDE:
        Serial.println("SEQUENZ_ENDE");
        fahrzeugampel.gelb.aus();
        fahrzeugampel.rot.aus();
        fahrzeugampel.gruen.an();
        zustand = DEF;
        t.resetTimer();
        t.subCount();
        break;
      case ENERGIESPARMODUS:
        Serial.println("ENERGIESPARMODUS");
        if(buttonaktiv) {
          zustand = DEF;
          t.resetTimer();
          break;
        }
        fahrzeugampel.gruen.aus();
        fahrzeugampel.gelb.aus();
        fahrzeugampel.rot.aus();
        fussgaengerampel.gruen.aus();
        fussgaengerampel.rot.aus();
        SysCtlPeripheralEnable(SYSCTL_PERIPH_HIBERNATE);
        HibernateEnableExpClk(SysCtlClockGet());
        HibernateGPIORetentionEnable();
        HibernateWakeSet(HIBERNATE_WAKE_PIN);
        HibernateRequest();
        break;
    }
  }
  if(!buttonaktiv) {
    t.addCount();
  }
}
```