

Praktikum 8

Jan Augstein, Michael Nickel

Aufgabe 1

```
#define TW 500
#define TU 250
#define TG 500

const uint8_t Pedestrian_Green = PC_4;
const uint8_t Pedestrian_Red = PC_5;
const uint8_t Vehicle_Green = PC_6;
const uint8_t Vehicle_Yellow = PC_7;
const uint8_t Vehicle_Red = PD_6;

uint8_t state = 0;

template <const uint8_t PORT_NB>
class Tled {
public:
    Tled(const uint8_t f_ledState = LOW)
        : m_ledState(f_ledState), m_disabled(false) {
        pinMode(PORT_NB, OUTPUT); // led is always output
        digitalWrite(PORT_NB, m_ledState); // set led to default state
    }
    //! If this led is disable, nothing happens, otherwise
    //! toggles state of led (from HIGH to LOW or from LOW to HIGH).
    void toggle_on() {
        if (m_disabled) {
            return; // somehow no longer active
        }

        m_ledState = HIGH;

        digitalWrite(PORT_NB, m_ledState); // set led to current state
    }
    void toggle_off() {
        if (m_disabled) // somehow no longer active
            return;
        m_ledState = LOW;

        digitalWrite(PORT_NB, m_ledState); // set led to current state
    }
    //! Turn led finally off (emergency stop), state is set LOW, functionality off.
    void off() {
        m_disabled = true;
        m_ledState = LOW;
        digitalWrite(PORT_NB, m_ledState); // set led to current state
    }
private:
    uint8_t m_ledState; // current state of led
    bool m_disabled; // disable flag (on if led is finally turned off)
};

template <const uint8_t PIN_NB>
class TButton {
public:
    TButton()
        : buttonState(LOW), lastButtonState(LOW), lastDebounceTime(0), debounceDelay(50) {
        pinMode(PIN_NB, INPUT);
    },
```

```

    }

    uint8_t state() {

        int returnValue = LOW;

        int currentState = digitalRead(PIN_NB);

        if (currentState != lastButtonState) {
            lastDebounceTime = millis();
        }

        if ((millis() - lastDebounceTime) > debounceDelay) {
            if (currentState != buttonState) {
                buttonState = currentState;
                if (buttonState == LOW) {
                    returnValue = HIGH;
                }
            }
        }

        lastButtonState = currentState;

        return returnValue;
    }

private:
    int buttonState;
    int lastButtonState;
    unsigned long lastDebounceTime;
    unsigned long debounceDelay;
};

TLed <Pedestrian_Green> p_Green;
TLed <Pedestrian_Red> p_Red;
TLed <Vehicle_Green> v_Green;
TLed <Vehicle_Yellow> v_Yellow;
TLed <Vehicle_Red> v_Red;
TButton <PUSH2> button;

void changeState() {
    if (state == 0) {
        v_Yellow.toggle_off();
        v_Red.toggle_off();
        v_Green.toggle_on();
        p_Red.toggle_on();
        Serial.println("z0: Fussgaengerampel: rot, Fahrzeugampel: gruen");
    } else if (state == 1) {
        Serial.println("z1: Fussgaengerampel: rot, Fahrzeugampel: gruen");
    } else if (state == 2) {
        Serial.println("z2: Fussgaengerampel: rot, Fahrzeugampel: gruen");
    } else if (state == 3) {
        v_Yellow.toggle_on();
        v_Green.toggle_off();
        Serial.println("z3: Fussgaengerampel: rot, Fahrzeugampel: gelb");
    } else if (state == 4) {
        v_Yellow.toggle_off();
        v_Red.toggle_on();
        Serial.println("z4: Fussgaengerampel: rot, Fahrzeugampel: rot");
    } else if (state == 5) {
        p_Red.toggle_off();
        p_Green.toggle_on();
        Serial.println("z5: Fussgaengerampel: gruen, Fahrzeugampel: rot");
    } else if (state == 6) {
        Serial.println("z6: Fussgaengerampel: gruen, Fahrzeugampel: rot");
    } else if (state == 7) {

```

```

        p_Red.toggle_on();
        p_Green.toggle_off();
        Serial.println("z7: Fussgaengerampel: rot, Fahrzeugampel: rot");
    } else if (state == 8) {
        v_Yellow.toggle_on();
        Serial.println("z8: Fussgaengerampel: rot, Fahrzeugampel: gelb-rot");
    }
}

void setup() {
    Serial.begin(9600);
    v_Green.toggle_on();
    p_Red.toggle_on();
    changeState();
}

void loop() {

    if (button.state()) {
        state++;
        changeState();
        delay(TW);

        state++;
        changeState();
        delay (TU);

        state++;
        changeState();
        delay(TU);

        state++;
        changeState();
        delay(TU);

        state++;
        changeState();
        delay(TG);

        state++;
        changeState();
        delay (TU);

        state++;
        changeState();
        delay(TU);

        state++;
        changeState();
        delay(TU);

        state = 0;
        changeState();

    }
}

```

Aufgabe 2

```
#include <stdint.h>
```

```

#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/hibernate.h"

#define TW 500
#define TU 250
#define TG 500
#define TE 2500

const uint8_t Pedestrian_Green = PC_4;
const uint8_t Pedestrian_Red = PC_5;
const uint8_t Vehicle_Green = PC_6;
const uint8_t Vehicle_Yellow = PC_7;
const uint8_t Vehicle_Red = PD_6;

uint8_t state = 0;

template <const uint8_t PORT_NB>
class TLed {
public:
    ///! Constructor takes state (HIGH, LOW) only if given.
    ///! Defaults: value for state = LOW, and is not disabled.
    TLed(const uint8_t f_ledState = LOW)
        : m_ledState(f_ledState), m_disabled(false) {
        pinMode(PORT_NB, OUTPUT); // led is always output
        digitalWrite(PORT_NB, m_ledState); // set led to default state
    }
    ///! If this led is disable, nothing happens, otherwise
    ///! toggles state of led (from HIGH to LOW or from LOW to HIGH).
    void toggle_on() {
        if (m_disabled) {
            return; // somehow no longer active
        }

        m_ledState = HIGH;

        digitalWrite(PORT_NB, m_ledState); // set led to current state
    }
    void toggle_off() {
        if (m_disabled) // somehow no longer active
            return;
        m_ledState = LOW;

        digitalWrite(PORT_NB, m_ledState); // set led to current state
    }
    ///! Turn led finally off (emergency stop), state is set LOW, functionality off.
    void off() {
        m_disabled = true;
        m_ledState = LOW;
        digitalWrite(PORT_NB, m_ledState); // set led to current state
    }
private:
    uint8_t m_ledState; // current state of led
    bool m_disabled; // disable flag (on if led is finally turned off)
};

template <const uint8_t PIN_NB>
class TButton {
public:

```

```

TButton()
: buttonState(LOW), lastButtonState(LOW), lastDebounceTime(0), debounceDelay(50) {
    pinMode(PIN_NB, INPUT);
}

uint8_t state() {
    // prepare the default return value
    int returnValue = LOW;

    int currentState = digitalRead(PIN_NB);

    if (currentState != lastButtonState) {

        lastDebounceTime = millis();
    }

    if ((millis() - lastDebounceTime) > debounceDelay) {

        if (currentState != buttonState) {
            buttonState = currentState;

            if (buttonState == LOW) {
                returnValue = HIGH;
            }
        }
    }

    lastButtonState = currentState;

    return returnValue;
}

private:
    int buttonState;
    int lastButtonState;
    unsigned long lastDebounceTime;
    unsigned long debounceDelay;
};

class Timer
{
public:
    static Timer& getInstance() {
        static Timer timer;
        return timer;
    }
    void setISRFunction(void (*ISRFunction)(void)) {
        TimerIntRegister(TIMER0_BASE, TIMER_A, ISRFunction);
    }
    void setTimer(unsigned long timespan_ms) {
        float hz = 1 / (timespan_ms / 1000.0f);
        uint32_t ui32Period = (SysCtlClockGet() / hz);
        TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period);

        //Timer und Interrupt
        TimerEnable(TIMER0_BASE, TIMER_A);
        IntEnable(INT_TIMER0A);
        TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    }
    void resetTimer() {
        TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    }
private:
    Timer() {
        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
        TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT);
    }
};

```

```

    timer.configure(TIMER_BASE, TIMER_CLK_DIVIDER);
}

};

Tled <Pedestrian_Green> p_Green;
Tled <Pedestrian_Red> p_Red;
Tled <Vehicle_Green> v_Green;
Tled <Vehicle_Yellow> v_Yellow;
Tled <Vehicle_Red> v_Red;
TButton <PUSH2> button;

void changeState() {
    if (state == 0) {
        v_Yellow.toggle_off();
        v_Red.toggle_off();
        v_Green.toggle_on();
        p_Red.toggle_on();
        Serial.println("z0: Fussgaengerampel: rot, Fahrzeugampel: gruen");
    } else if (state == 1) {
        Serial.println("z1: Fussgaengerampel: rot, Fahrzeugampel: gruen");
    } else if (state == 2) {
        Serial.println("z2: Fussgaengerampel: rot, Fahrzeugampel: gruen");
    } else if (state == 3) {
        v_Yellow.toggle_on();
        v_Green.toggle_off();
        Serial.println("z3: Fussgaengerampel: rot, Fahrzeugampel: gelb");
    } else if (state == 4) {
        v_Yellow.toggle_off();
        v_Red.toggle_on();
        Serial.println("z4: Fussgaengerampel: rot, Fahrzeugampel: rot");
    } else if (state == 5) {
        p_Red.toggle_off();
        p_Green.toggle_on();
        Serial.println("z5: Fussgaengerampel: gruen, Fahrzeugampel: rot");
    } else if (state == 6) {
        Serial.println("z6: Fussgaengerampel: gruen, Fahrzeugampel: rot");
    } else if (state == 7) {
        p_Red.toggle_on();
        p_Green.toggle_off();
        Serial.println("z7: Fussgaengerampel: rot, Fahrzeugampel: rot");
    } else if (state == 8) {
        v_Yellow.toggle_on();
        Serial.println("z8: Fussgaengerampel: rot, Fahrzeugampel: gelb-rot");
    }
}

void next()
{
    Timer::getInstance().resetTimer();
    state++;
    if (state == 9) {
        state = 0;
    }
    switch (state) {

        case 0:
            changeState();
            setSleep();
            break;

        case 1:
            changeState();
            Timer::getInstance().setTimer(TW);

```

```

        break;

    case 2:
        changeState();
        Timer::getInstance().setTimer(TU);
        break;

    case 3:
        changeState();
        Timer::getInstance().setTimer(TU);
        break;

    case 4:
        changeState();
        Timer::getInstance().setTimer(TU);
        break;

    case 5:
        changeState();
        Timer::getInstance().setTimer(TG);
        break;

    case 6:
        changeState();
        Timer::getInstance().setTimer(TU);
        break;

    case 7:
        changeState();
        Timer::getInstance().setTimer(TU);
        break;

    case 8:
        changeState();
        Timer::getInstance().setTimer(TU);
        break;
}

}

}

void setSleep()
{
    Timer::getInstance().setISRFunction(goSleep);
    Timer::getInstance().setTimer(TE);
}

void goSleep()
{
    p_Green.toggle_off();
    p_Red.toggle_off();
    v_Red.toggle_off();
    v_Yellow.toggle_off();
    v_Green.toggle_off();
    HibernateRequest();
    while (1)
    {
    }
}

void setup() {
    Serial.begin(9600);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_HIBERNATE);
    HibernateEnableFromClock(SYSCTL_CLOCK_HIBERNATE);
}

```

```
    HibernateEnableEXCLK(SYSCTL_CLOCKGET());
    HibernateGPIORetentionEnable();
    HibernateWakeSet(HIBERNATE_WAKE_PIN);
    setSleep();

    v_Green.toggle_on();
    p_Red.toggle_on();
    changeState();

}

void loop() {

    if (button.state() && state == 0) {
        Timer::getInstance().resetTimer();
        Timer::getInstance().setISRFunction(next);
        Timer::getInstance().setTimer(TW);
    }

}
```