Embedded Systems
Heizung-FreeRTOS
Praktikum 10

Fachhochschule Bielefeld
Campus Minden
Studiengang Informatik

Beteiligte Personen:

Name	Matrikelnummer
Jan-Hendrik Sünderkamp	1153536
Peter Dick	1050185

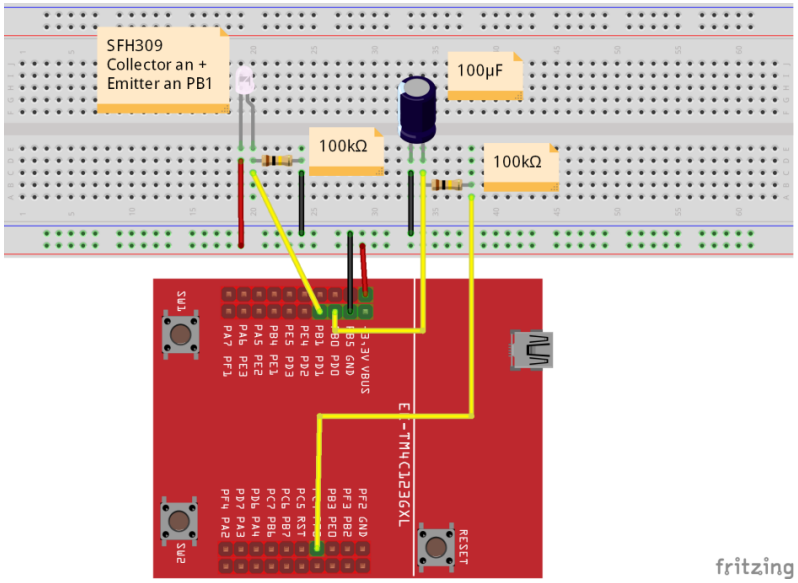
---

8. Juli 2018

Inhaltsverzeichnis

1 Vorbereitung

In der Vorlesung wird als Beispiel eines RTOS das FreeRTOS behandelt. Lesen Sie in der Dokumentation zu FreeRTOS (`http://www.freertos.org`) im Quick Start Guide und der API, wie in FreeRTOS Tasks angelegt und benutzt werden. Datenblatt des Fototransistors: `http://www.mouser.com/ds/2/311/SFH%20309,%20SFH%20309%20FA,%20Lead%20(Pb)%20Free%20Product%20-%20RoHS-368614.pdf`



Das Schaltbild zeigt eine RC-Kombination (Widerstand und Elektrolytkondensator) und einen Fototransistor zur Simulation eines Systems mit Temperaturregelung. Die Kapazität des Kondensators stellt dabei die Temperatur und der Widerstand die Heizung sowie Kühlung dar.

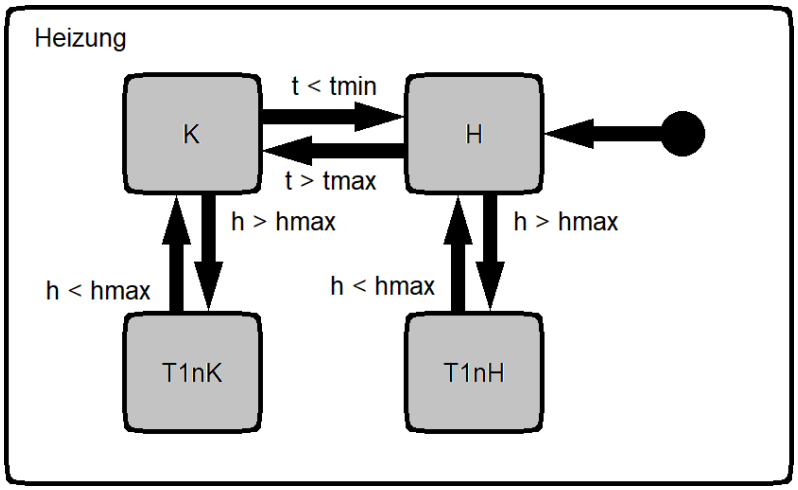
Es sollen nun zwei Prozesse/Tasks laufen. Ist der erste Task aktiv, wird er jede Sekunde aufgerufen. Der zweite Task hat eine höhere Priorität als der erste und wird alle zwei Sekunden aufgerufen.

In dem ersten Task wird die aktuelle Temperatur (=Spannung des Kondensators) ermittelt. Es wird unterschieden in Heizen und Kühlen. Beim Heizen (= Aufladen des Kondensators) wird solange Energie zugeführt, bis der ausgelesene Wert z.B. 600 Einheiten überschreitet. Dann findet das Kühlen (=Entladen) solange statt, bis der Wert z.B. 300 Einheiten unterschreitet. Nach einem Unterschreiten wird wieder das Heizen durchgeführt usw.

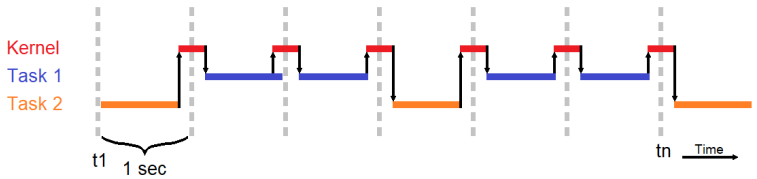
Der zweite Task überwacht die Helligkeit und regelt, dass die Heizung nur bei einer Helligkeit kleiner als z.B. 200 Einheiten aktiv ist.

2 Aufgabe 1

Modellieren Sie das beschriebene System mit einen Statechart. Skizzieren Sie die Aktivität der Prozesse exemplarisch als Zeitverlaufsdiagramm.

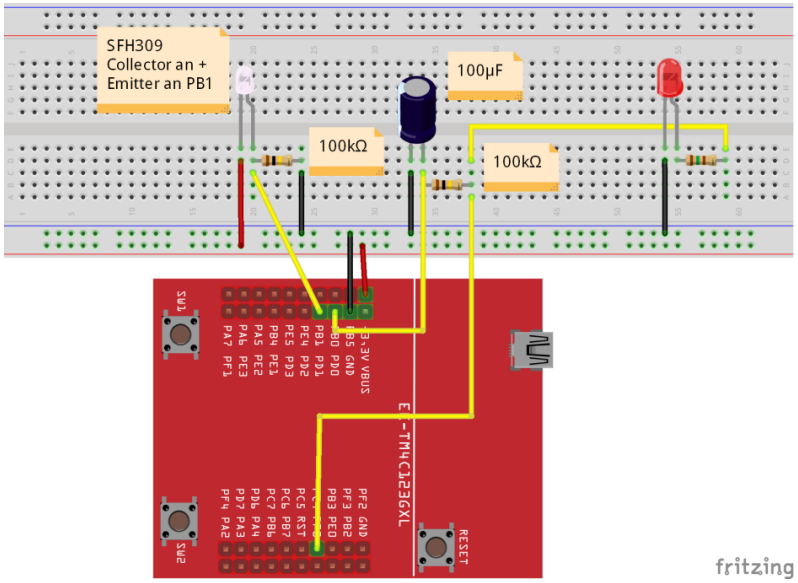


- K = kühlen
- H = heizen
- T1nK = Task 1 nickt aktiv, letzter Zustand kühlen
- T1nH = Task 1 nickt aktiv, letzter Zustand heizen
- h = Helligkeit
- hmax = max Helligkeit
- t = Temperatur
- tmax = min Temperatur
- tmin = min Temperatur



3 Aufgabe 2

Bauen Sie die dargestellte Schaltung auf und fügen Sie dieser eine LED (mit Vorwiderstand, 150 Ω) hinzu, so dass die LED leuchtet, wenn gerade geheizt wird. Die LED ist in die Schaltung zu integrieren, ohne dass ein weiterer Port des LaunchPads verwendet wird. D.h. das Programm für das LaunchPad in der nächsten Aufgabe soll die LED nicht explizit betrachten.



4 Aufgabe 3

Programmieren Sie das beschriebene System für das LaunchPad unter strikter Verwendung der FreeRTOS Bibliothek. Damit geht insbesondere ein Verbot der Energie-Funktion *delay()* einher.

Tipp: Sie dürfen die GPIO-Funktionen der Energie-Bibliothek, wie z.B. *pinMode()* oder *analogRead()*, verwenden.

Tipp: Als Startpunkt liegt ein Eclipse-Projekt im Ilias bereit, das die Onboad-LED des Boards unter Verwendung von FreeRTOS blinken lässt (s. main.cpp).

```
#include <Energia.h>

#if defined(PART_TM4C129XNCZAD)
#include "inc/tm4c129xnczad.h"
#elif defined(PART_TM4C1294NCPDT)
#include "inc/tm4c1294ncpdt.h"
#elif defined(PART_TM4C1233H6PM) || defined(PART_LM4F120H5QR)
```

```
#include "inc/tm4c123gh6pm.h"
#else
#error "**** No PART defined or unsupported PART ****"
#endif

#include "inc/hw_gpio.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/eeprom.h"

#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <freertos/semphr.h>

#ifdef __cplusplus
extern "C" {

void __init(void) {
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_EEPROM0);
  if (ROM_EEPROMInit() == EEPROM_INIT_ERROR) {
    if (ROM_EEPROMInit() != EEPROM_INIT_ERROR)
      EEPROMMassErase();
  }

  timerInit();

  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);
#ifdef TARGET_IS_SNOWFLAKE_RA0
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOR);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOS);
  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOT);
#endif
```

```
  //Unlock and commit NMI pins PD7 and PF0
  HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0x4C4F434B;
  HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x1;
  HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0x4C4F434B;
  HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= 0x80;

  // Deep Sleep mode init
  // DSLP clock = PIOSC / 16 = 1MHz
  // Note: Couldn't find the define constants for
   SysCtlDeepSleepPowerSet in the driverlib.
  //
#ifdef TARGET_IS_BLIZZARD_RB1
  ROM_SysCtlDeepSleepClockSet(SYSCTL_DSLP_DIV_16 |
   SYSCTL_DSLP_OSC_INT);
  SysCtlDeepSleepPowerSet(0x21); // FLASHPM = LOW_POWER_MODE,
   SRAMPM = STANDBY_MODE
  SysCtlLDODeepSleepSet(SYSCTL_LDO_1_00V); // Going lower tends
   to be very flaky and cause continual resets
  // particularly when measuring MCU current.
#endif
  //
#ifdef TARGET_IS_SNOWFLAKE_RA0
  ROM_SysCtlDeepSleepClockConfigSet(16, SYSCTL_DSLP_OSC_INT);
  SysCtlDeepSleepPowerSet(0x121); // TSPD, FLASHPM =
   LOW_POWER_MODE, SRAMPM = STANDBY_MODE
#endif
  //
} /* void _init(void) */

} /* extern "C" */
#endif

const uint8_t FotoPinIn = PD_1;
const uint8_t KondensatorPinIn = PD_0;
const uint8_t KondensatorPinOut = PC_4;
const int tmax = 1000;
const int tmin = 300;
const int hmax = 3300;
TaskHandle_t xTask1Handle = NULL;
boolean task1akive;

enum Status {
  K,
  H,
  T1nK,
```

```
  T1nH
};

Status zustand = H;

void loop() {
  int val = 0;
  switch(zustand) {
    case K:
    Serial.println("Zustand: K");
    if(task1akive) {
      val = analogRead(KondensatorPinIn);
      Serial.println(val, DEC);
      if(val < tmin) {
        zustand = H;
      } else {
        digitalWrite(KondensatorPinOut, LOW);
      }
    } else {
      val = analogRead(FotoPinIn);
      //Serial.println(val, DEC);
      if(val > hmax) {
        vTaskSuspend(xTask1Handle);
        zustand = T1nK;
      }
    }
    break;
    case H:
    Serial.println("Zustand: H");
    if(task1akive) {
      val = analogRead(KondensatorPinIn);
      Serial.println(val, DEC);
      if(val > tmax) {
        zustand = K;
      } else {
        digitalWrite(KondensatorPinOut, HIGH);
      }
    } else {
      val = analogRead(FotoPinIn);
      //Serial.println(val, DEC);
      if(val > hmax) {
        vTaskSuspend(xTask1Handle);
        zustand = T1nH;
      }
    }
    break;
```

```
    case T1nK:
    Serial.println("Zustand: T1nK");
    if(!task1akive) {
      val = analogRead(FotoPinIn);
      //Serial.println(val, DEC);
      if(val < hmax) {
        vTaskResume(xTask1Handle);
        zustand = K;
      }
    }
    break;
    case T1nH:
    Serial.println("Zustand: T1nH");
    if(!task1akive) {
      val = analogRead(FotoPinIn);
      //Serial.println(val, DEC);
      if(val < hmax) {
        vTaskResume(xTask1Handle);
        zustand = H;
      }
    }
    break;
  }
}

void setup() {
  // initialize the digital pin as an output.
  pinMode(FotoPinIn, INPUT);
  pinMode(KondensatorPinIn, INPUT);
  pinMode(KondensatorPinOut, OUTPUT);
  Serial.begin(9600);
}

void task1(void * pvParameters) {
  TickType_t xLastWakeTime;
  xLastWakeTime = xTaskGetTickCount();
  for (;;) {
    Serial.println("Task 1");
    task1akive = true;
    loop();
    vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(1000));
  }
  vTaskDelete( NULL );
}

void task2(void * pvParameters) {
```

```
  TickType_t xLastWakeTime;
  xLastWakeTime = xTaskGetTickCount();
  for (;;) {
    Serial.println("Task 2");
    task1akive = false;
    loop();
    vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(2000));
  }
  vTaskDelete( NULL );

}

int main(void) {
  setup();
  xTaskCreate(task1, "TASK 1", configMINIMAL_STACK_SIZE + 100,
   NULL, tskIDLE_PRIORITY + 1UL, &xTask1Handle);
  xTaskCreate(task2, "TASK 2", configMINIMAL_STACK_SIZE + 100,
   NULL, tskIDLE_PRIORITY + 2UL, NULL);
  vTaskStartScheduler();
}
```