HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Fakultät Ingenieurwissenschaften

Modul 5420 - Embedded Systems I

Beleg & Dokumentation

Author Soltan Laila, Lukas Göckeritz
Betreuer Professor Pretschner

17. März 2025

Inhaltsverzeichnis

1 Einleitung

Das vorliegende Projekt konzentriert sich im Rahmen des Moduls Embedded Systems auf die Entwicklung eines Real-Time-Arduino-Systems unter Verwendung eines Arduiono Uno. Im Zentrum des Projekts steht die Erfassung von Sensorinformationen, insbesondere von Temperaturdaten und Distanzedaten, die mittels einem digitalen Displays ausgegeben werden sollen.

2 Zielsetzung und Anwendungsbereich

Das Hauptziel dieses Projekts ist die Erfassung von Sensordaten.Das System soll in der Lage sein, kontinuierlich die Raumtemperatur zu messen und bei Überschreitung eines Schwellenwerts eine Benachrichtigung auszulösen. Der Anwendungsbereich des Systems erfolgt bei der Temperaturüberwachung, welche in Abhängigkeit zur Distanz des zu messenden Objektes sicher gestellt wird.

3 Themenüberblick und Motivation

Eingebettete Systeme sind ein zentraler Bestandteil moderner Automatisierungstechnik. Sie ermöglichen es, Sensordaten effizient zu erfassen und in Echtzeit zu verarbeiten. In unserem Projekt haben wir ein System entwickelt, das Temperatur- und Distanzmessungen durchführt und die Werte auf einem LCD-Display anzeigt. Zusätzlich werden LEDs genutzt, um bestimmte Schwellenwerte visuell zu signalisieren.

Unser Fokus lag darauf, ein zuverlässiges und reaktionsschnelles System zu entwickeln, das ohne eine Serververbindung oder Cloud-Anbindung autark arbeitet. Dazu haben wir FreeRTOS genutzt, um eine Multitasking-Umgebung zu schaffen, in der mehrere Prozesse gleichzeitig ablaufen können – etwa die kontinuierliche Sensordatenerfassung, die Echtzeit-Anzeige auf dem Display und die LED-Steuerung.

Durch dieses Projekt konnten wir unsere Kenntnisse im Bereich Embedded Systems, Echtzeitbetriebssysteme und Multitasking vertiefen. Besonders spannend ist die Herausforderung, mit begrenzten Hardware-Ressourcen eine effiziente und stabile Lösung zu realisieren.

4 Hardware und Software

Hardware:

- Arduino Uno: Als zentrales Steuergerät dient ein Arduiono Uno, welcher die Messwerte des Abstands- und des Temperatursensors ausliest und diese über LEDs und ein Display ausgibt.
- Ultrasonic Distance Sensor 5V HC-SR04: Ein kostengünstiger Ultraschall-Distanzsensor, welcher Abstände von 2cm bis 400cm erfassen kann.
- Temperatursensor TMP36GT9Z: Ein analoger Temperatursensor, welcher Temperaturen im Bereich von -40°C bis 125°C messen kann.
- LCD 1602A: Ein kleines Display zur Anzeige der zu messenden Größen.
- 3 LEDs: Einface LEDs zur optischen Ausgabe von Betriebszustand, Distanzunterschreitung und Temperaturüberschreitung.

Software:

- Visual Studio Code: Diese Entwicklungsumgebung wurde genutzt, da sie eine flexible und benutzerfreundliche Plattform für die Programmierung des Arduino bietet. Zudem ermöglicht sie durch verschiedene Erweiterungen eine komfortable Entwicklung mit eingebetteten Systemen.
- ARTe: Diese Software ist speziell für die Entwicklung von Echtzeitsystemen konzipiert. Sie erleichtert das Arbeiten mit Echtzeitbetriebssystemen (RTOS) und ermöglicht eine präzisere Steuerung von Multitasking-Prozessen.
- FreeRTOS: Dieses Echtzeitbetriebssystem ist essenziell für das Projekt, da es eine effiziente Taskverwaltung ermöglicht. Es stellt sicher, dass die Temperatur- und Distanzmessung in Echtzeit verarbeitet werden, ohne dass sich Aufgaben gegenseitig blockieren.
- Python: Python wurde verwendet, um unterstützende Skripte zu schreiben, beispielsweise zur Datenauswertung oder zur Kommunikation zwischen dem PC und dem Arduino. Durch seine einfache Syntax eignet es sich gut für schnelle Tests und Prototyping.

5 Verkabelung und Installation

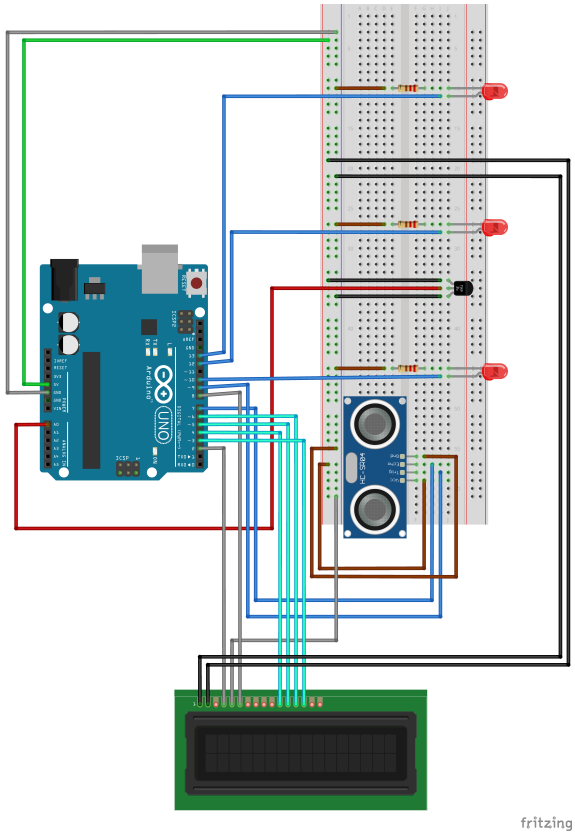

Abbildung 1: Verkabelung Arduino mit Bauteilen

Installation von Visual Studio Code: Visual Studio Code wurde als Entwicklungsumgebung genutzt. Die Installation erfolgte über die offizielle Website (https://code.visualstudio.com) und wurde anschließend für die Arbeit mit eingebetteten Systemen konfiguriert.

Installation von ARTe (Arduino Real-Time Environment): ARTe wurde über das Terminal in Visual Studio Code installiert. Dabei haben wir die notwendigen Pakete direkt über den Paketmanager von Arduino bzw. durch das Klonen des entsprechenden GitHub-Repositories eingebunden:

```
git clone https://github.com/arte-framework/arte.git
```

```
cd arte
```

Anschließend konnten wir ARTe in unsere Entwicklungsumgebung integrieren.

Installation von Python v3.13.2: Wir haben die neueste Version von Python über die Kommandozeile in Windows installiert:

```
winget install Python.Python
```

Nach der Installation wurde überprüft, dass die aktuelle Version korrekt eingerichtet ist:

```
python --version
```

Installation von FreeRTOS: FreeRTOS wurde über das Terminal durch Klonen des offiziellen GitHub-Repositories installiert:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Kernel.git
```

```
cd FreeRTOS-Kernel
```

Anschließend konnten wir die benötigten Bibliotheken in unsere Entwicklungsumgebung einbinden.

6 Programmierung

In diesem Abschnitt wird der Code für die Steuerung des Systems erläutert. Die Implementierung basiert auf FreeRTOS und nutzt verschiedene Tasks zur parallelen Verarbeitung von Sensordaten und Anzeigeinformationen. Der Code ist aus Gründen der Übersicht in einzelne Teile zerlegt. Diese sind in der "mainDatei in Visual Studio zusammengefügt und hier chronologisch dargestellt.

6.1 Bibliotheken und Pin-Definitionen

```
#include <Arduino.h>
#include <Arduino_FreeRTOS.h>
#include <LiquidCrystal.h>

// Pin-Definitionen
#define TRIGGER_PIN 9
#define ECHO_PIN 7
#define LED_HC 10
#define LED_PULSED 13
#define LED_TEMPERATURE 12
#define TEMP_SENSOR A0

// LCD-Pins
#define LCD_RS 2
#define LCD_E 8
#define LCD_D4 4
#define LCD_D5 5
#define LCD_D6 6
```

```
#define LCD_D7 3

// LCD-Objekt erstellen
LiquidCrystal lcd(LCD_RS, LCD_E, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
```

Listing 1: Bibliotheken und Pin-Definitionen

6.2 Setup und Initialisierung

```
void setup() {
    Serial.begin(115200);
    lcd.begin(16, 2);
    lcd.setCursor(0, 0);
    lcd.print("LCD Test");

    Serial.println("Setup gestartet...");

    pinMode(TRIGGER_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
    pinMode(LED_HC, OUTPUT);
    pinMode(LED_PULSED, OUTPUT);
    pinMode(LED_TEMPERATURE, OUTPUT);

    Serial.println("Pins initialisiert!");

    digitalWrite(TRIGGER_PIN, LOW);
    digitalWrite(LED_HC, LOW);
    digitalWrite(LED_PULSED, LOW);
    digitalWrite(LED_TEMPERATURE, LOW);

    analogReference(INTERNAL);

    Serial.println("Tasks erstellen...");
    xTaskCreate(TaskUltrasonicSensor, "Ultrasonic", 100, NULL, 1, NULL);
    xTaskCreate(TaskBlinkLED, "BlinkLED", 100, NULL, 1, NULL);
    xTaskCreate(TaskReadTemperature, "Temperature", 100, NULL, 1, NULL);

    Serial.println("Setup abgeschlossen!");
}
```

Listing 2: Setup-Funktion

6.3 Tasks

6.3.1 Task 1: Entfernungsmessung mit Ultraschallsensor

```
void TaskUltrasonicSensor(void *pvParameters) {
    while (1) {
        digitalWrite(TRIGGER_PIN, HIGH);
        delayMicroseconds(10);
        digitalWrite(TRIGGER_PIN, LOW);
        delayMicroseconds(5);

        long duration = pulseIn(ECHO_PIN, HIGH, 100000);
        long distance = duration / 29 / 2;

        if (distance > 0 && distance < 30) {
            digitalWrite(LED_HC, HIGH);
        } else {
            digitalWrite(LED_HC, LOW);
        }

        Serial.print("Entfernung: ");
        Serial.print(distance);
        Serial.println(" cm");

```

```
        lcd.setCursor(0, 0);
        lcd.print("Entfernung: ");
        lcd.print(distance);
        lcd.print("cm  ");

        vTaskDelay(500 / portTICK_PERIOD_MS);
    }
}
```

Listing 3: Task zur Entfernungsmessung

6.3.2 Task 2: LED-Blinken alle 7 Sekunden

```
void TaskBlinkLED(void *pvParameters) {
    while (1) {
        for (int i = 0; i < 5; i++) {
            digitalWrite(LED_PULSED, HIGH);
            vTaskDelay(500 / portTICK_PERIOD_MS);
            digitalWrite(LED_PULSED, LOW);
            vTaskDelay(500 / portTICK_PERIOD_MS);
        }
        vTaskDelay(7000 / portTICK_PERIOD_MS);
    }
}
```

Listing 4: Task zum Blinken der LED

6.3.3 Task 3: Temperaturmessung alle 3 Sekunden

```
void TaskReadTemperature(void *pvParameters) {
    while (1) {
        float sum_temp = 0;
        for (int i = 0; i < 3; i++) {
            sum_temp += (analogRead(TEMP_SENSOR) * 110.0) / 1024;
            vTaskDelay(500 / portTICK_PERIOD_MS);
        }

        float average_temp = sum_temp / 3;

        if (average_temp > 23) {
            digitalWrite(LED_TEMPERATURE, HIGH);
        } else {
            digitalWrite(LED_TEMPERATURE, LOW);
        }

        Serial.print("Temperatur: ");
        Serial.print(average_temp);
        Serial.println(" C ");

        lcd.setCursor(0, 1);
        lcd.print("Temp: ");
        lcd.print(average_temp);
        lcd.print("C   ");

        vTaskDelay(3000 / portTICK_PERIOD_MS);
    }
}
```

Listing 5: Task zur Temperaturmessung

7 Test, Validierung und Probleme

Nach der erfolgreichen Implementierung des Systems wurden verschiedene Tests durchgeführt, um die Funktionalität der Sensoren, LEDs und des Displays zu validieren. Dazu haben wir ein Video

zur Demonstration der Funktion ins Git Repository hochgeladen.

7.1 Funktionsprüfung der LEDs

Die drei LEDs im System dienen als visuelle Indikatoren für verschiedene Zustände:

- **Betriebs-LED:** Diese LED zeigt an, dass das System aktiv ist und die Messungen durchgeführt werden.
- **Temperatur-LED:** Sobald die gemessene Temperatur den Schwellenwert von 23 °C überschreitet, wird diese LED aktiviert.
- **Distanz-LED:** Die LED für die Abstandsmessung leuchtet, solange ein Objekt innerhalb von 30 cm erkannt wird. Überschreitet die gemessene Distanz diesen Wert, erlischt die LED.

Alle LEDs funktionierten erwartungsgemäß und ermöglichten eine klare visuelle Rückmeldung über die Systemzustände.

7.2 Anzeige der Messwerte auf dem LCD

Das LCD-Display wurde getestet, indem die aktuellen Messwerte des Temperatur- und Ultraschallsensors ausgegeben wurden. In der ersten Zeile wurde die gemessene Entfernung angezeigt, während die zweite Zeile die Temperaturwerte ausgab. Bei den Tests wurde überprüft, ob die Werte korrekt aktualisiert und alte Werte überschrieben wurden, um eine lesbare Darstellung zu gewährleisten.

7.3 Probleme und Lösungsansätze

Während der Inbetriebnahme des Systems traten Spannungsversorgungsprobleme auf. Der Arduino allein konnte nicht genügend Strom liefern, um sowohl die Sensoren als auch das LCD und die LEDs zuverlässig zu betreiben. Dies führte zu fehlerhaften Messwerten und unregelmäßigem Verhalten der Komponenten.

Lösung: Um die Spannungsversorgung zu stabilisieren, wurde zusätzlich zum 5 V-Anschluss des Arduinos ein externes 12 V-Netzteil verwendet. Dadurch konnte eine konstante Versorgung sichergestellt und die Funktionalität aller Komponenten verbessert werden.

8 Ergebnisse und Diskussion

Nach der erfolgreichen Implementierung und Durchführung zahlreicher Tests konnten folgende Ergebnisse erzielt werden:

Das System zur Messung von Temperatur und Entfernung arbeitet erwartungsgemäß. Die drei verbauten LEDs zeigen unterschiedliche Zustände an:

- Eine LED leuchtet dauerhaft, um anzuzeigen, dass das System aktiv ist.
- Eine weitere LED schaltet sich ein, sobald die gemessene Temperatur den Schwellenwert von 23°C überschreitet.
- Die letzte LED geht aus, wenn die gemessene Entfernung einen Wert von 30 cm überschreitet.

Zusätzlich werden die erfassten Messwerte auf dem LCD-Display ausgegeben, sodass Nutzer die aktuellen Temperatur- und Distanzwerte jederzeit ablesen können.

Während der Inbetriebnahme des Systems traten jedoch Probleme auf. Insbesondere war die Spannungsversorgung über den Arduino nicht ausreichend, um alle Komponenten zuverlässig zu betreiben. Dies äußerte sich in einer instabilen Funktion des Displays und gelegentlichen Fehlfunktionen der Sensoren. Um dieses Problem zu beheben, wurde neben dem 5V-Anschluss des Arduinos zusätzlich ein externes 12V-Netzteil verwendet. Dies stellte die stabile Funktion des gesamten Systems sicher.

Hinsichtlich möglicher Verbesserungen könnte zukünftig eine energieeffizientere Lösung untersucht werden, um den Bedarf an externer Stromversorgung zu reduzieren. Ebenso wäre eine Optimierung der Messintervalle denkbar, um die Reaktionszeiten des Systems an spezifische Anwendungsfälle anzupassen.

9 Fazit

Das Projekt verdeutlicht die Machbarkeit und Effektivität eines einfachen, aber funktionalen Systems zur Echtzeitüberwachung von Temperatur und Entfernung.

- Durch die Kombination eines Temperatursensors, eines Ultraschallsensors und eines LCD-Displays konnten relevante Umweltparameter erfolgreich erfasst und ausgegeben werden.
- Die Implementierung eines Multi-Tasking-Ansatzes mittels FreeRTOS hat sich als vorteilhaft erwiesen, um eine gleichzeitige Verarbeitung mehrerer Sensordaten zu ermöglichen.
- Die Nutzung eines externen Netzteils als Lösung für die Spannungsproblematik hat gezeigt, dass der Strombedarf bereits bei kleineren eingebetteten Systemen eine kritische Rolle spielt.

Während der Umsetzung des Projekts wurden verschiedene Herausforderungen bewältigt, darunter Probleme mit der Spannungsversorgung und die Anpassung der FreeRTOS-Tasks zur stabilen Verarbeitung der Sensordaten. Diese Schwierigkeiten führten zu einem besseren Verständnis der eingesetzten Technologien und deren praktische Anwendung.

Für zukünftige Erweiterungen könnte das System um eine drahtlose Datenübertragung erweitert werden, beispielsweise durch die Anbindung an eine IoT-Plattform zur Speicherung und Analyse der Sensordaten. Ebenso wäre eine energieeffizientere Umsetzung denkbar, um die Abhängigkeit von externer Stromversorgung zu minimieren.

Zusammenfassend bot dieses Projekt wertvolle Einblicke in die Entwicklung eingebetteter Systeme und die Herausforderungen der Sensorintegration. Die erzielten Ergebnisse zeigen, dass mit geringem Aufwand und durchdachter Planung robuste Überwachungssysteme realisiert werden können.