Arduino: Secure Fridge **SoSe'18**
Hardwarenahe Systemprogrammierung Leonard Koll, 1133123
Prof. Dr. Peter Sturm Paul Kugener, 1071658

1 Projektbeschreibung

Das Ziel der Veranstaltung war die Einarbeitung in hardwarenahe Systeme sowie deren Planung und Programmierung. Als begleitendes Praxisprojekt wurde ein Sicherheitssystem für die Überwachung von Kühlschränken im Bürobereich entwickelt. Bei öffentlichen Kühlschränken, welche von vielen Personen genutzt werden, ist oft nicht klar einsehbar, wer den Kühlschrank nutzt und welche Produkte entnommen werden.

Das entwickelte Sicherheitssystem protokolliert mit Hilfe einer Kamera die Nutzung des Kühlschranks. Bei jeder Öffnungs des Kühlschrankes wird der Nutzer durch ein RFID-Token identifiziert. Die Token-ID und das aufgenommene Foto werden auf einer Micro-SD gespeichert.

Das folgende Kapitel beschreibt die Einzelkomponenten des Projektes. Danach wird der Systemaufbau dargestellt und der Funktionsablauf des Systems näher erläutert. Anschließend wird eine eigene Kommunikationslibrary vorgestellt, welche spezifisch auf dieses Projekt zugeschnitten wurde.

2 Komponenten

Microcontroller-Board Der *Arduino Uno*, basierend auf dem Atmega328 Chip, stellt das Herzstück des Projekts dar. Im Arduino wird der Kontrollfluss des ganzen Systems bestimmt. Alle weiteren Sensoren (Komponenten) werden an den Arduino angeschlossen und deren Daten werden im Arduino verarbeitet.

Kamera Als Kamera wurde die *Adafruit TTL JPEG Camera (VC0706 chipset)* benutzt. Die Kamera zeichnet sich vor allem durch ihre Bedienbarkeit und die umfangreiche Bibliothek aus. So kann beispielsweise durch eine simple Setter-Methode die Auflösung der Fotos festgelegt werden (640x480, 320x240 oder 160x120). Auch liegt der aktuelle Schnappschuss des Kamera-Moduls als komprimierte JPEG-Datei im Buffer vor und kann so in speichergerechten Teilen vom Arduino eingelesen werden.

Nutzeridentifizierung Zur Nutzererkennung wird ein *MFRC522 RFID Reader* verwendet. Das Modul ermöglicht das Lesen der Informationen eines RFID-Tokens durch elektromagnetische Wellen. Unterschiedliche RFID-Token repräsentieren in unserem Projekt die verschiedenen Mitarbeiter. Durch die Token-Überprüfung kann festgestellt werden, ob dem Nutzer die Nutzung des Kühlschrank erlaubt war oder nicht.

Speichermedium Die erfassten Daten können durch ein *Micro-SD-Card Modul* auf einer Micro-SD-Karte abgespeichert werden. Die Daten setzen sich zusammen aus dem Bildmaterial, welches von der Kamera gelesen wird und den Informationen der Nutzer, welche durch das RFID Modul erfasst werden.

Button Das Öffnen der Kühlschranktür wird durch das Drücken eines Buttons simuliert. Das Signal des Buttons dient als Trigger für einen Protokollierungsauftrag.

3 Systemaufbau

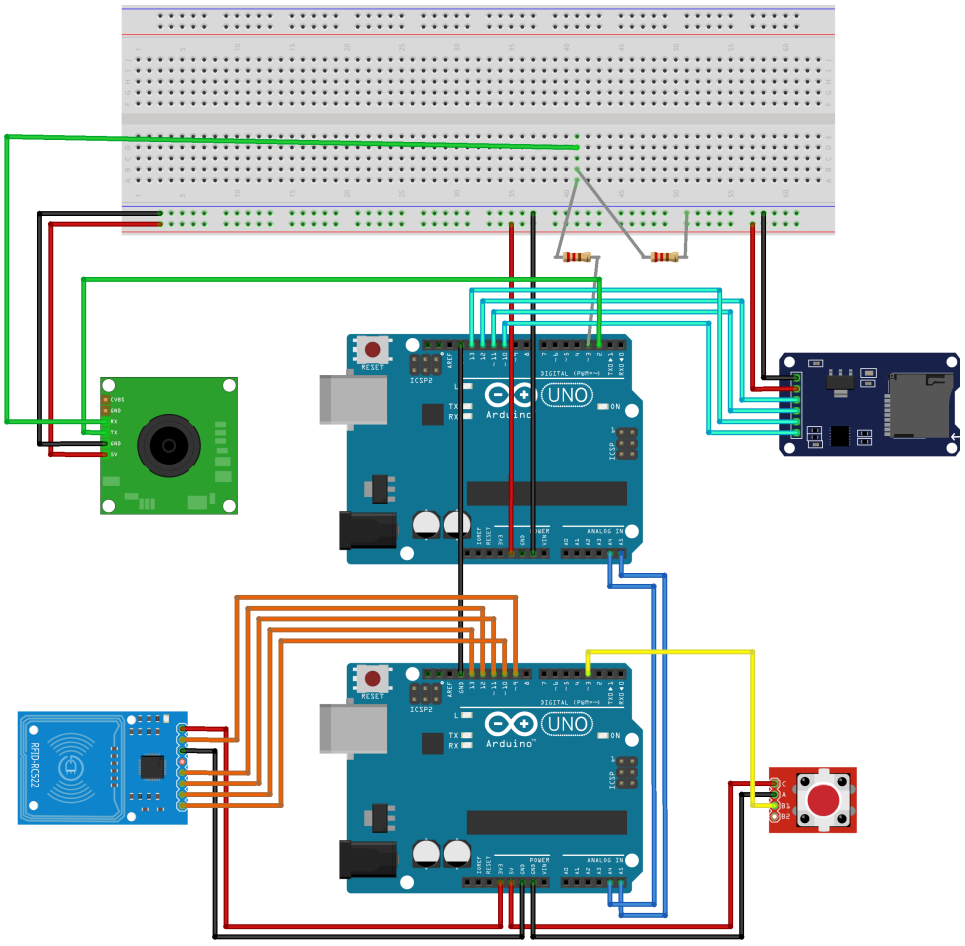

Abbildung 1: Verbindung der Komponenten in Steckplattenansicht

Der Aufbau des Überwachungssystems lässt sich in zwei Teilsysteme eingliedern. Beide Teilsysteme bestehen, wie in Abbildung 1 zu erkennen ist, aus

jeweils einem *Arduino Uno* und zwei weiteren Komponenten.

Als erstes Teilsystem kann das in der Abbildung untere System betrachtet werden. Die verbundenen Komponenten sind der Button und das RFID-Modul. Das zweite Teilsystem beinhaltet die Kamera und das SD-Karten-Modul. In der Abbildung ist zu erkennen, dass zwischen dem `RX`-Pin der Kamera und dem Arduino zwei *Resistoren* geschaltet sind.

Die Kommunikation zwischen den beiden Teilsystemen läuft über zwei Verbindungen von analogen Arduino-Pins. Die Untergliederung des Systems in zwei Komponenten ermöglicht eine räumlich getrennte Anbringung von Identifikationseinheit und Kamera. Die Schnittstellenanbindungen der Komponenten ist in der nachfolgenden Tabelle ablesbar.

Arduino 1		Arduino 2	
RFID_SDA	arduino_d10	CAMERA_TX	arduino_d2
RFID_SCK	arduino_d13	CAMERA_RX	resistor → GND
RFID_MOSI	arduino_d11		resistor → arduino_d3
RFID_MISO	arduino_d12	CAMERA_GND	GND
RFID_IRQ	not connected	CAMERA_VCC	5V
RFID_RST	arduino_d9		
RFID_GND	GND	SD_CS	arduino_d10
RFID_VCC	3.3V	SD_SCK	arduino_d13
		SD_MOSI	arduino_d11
BTN_S	arduino_d3	SD_MISO	arduino_d12
BTN_-	GND	SD_GND	GND
BTN_+	5V	SD_VCC	5V

4 Funktionsablauf

Der Funktionsablauf des Sicherheitssystems wurde im Sequenzdiagramm in Abbildung 2 dargestellt.

Ein Protokollierungsauftrag startet mit dem Öffnen der Kühlschranktür, was durch das Drücken des Buttons simuliert wird. Das System ermittelt daraufhin unter Zurhilfenahme des RFID-Moduls die Identität des Nutzers. Wird ein Token erkannt, wird kontrolliert, ob diesem Nutzer der Zugriff zum Kühlschrank gestattet ist. Anhand dieser Informationen wird nun ein `Request-String` gebildet und an den zweiten Arduino übertragen. Dieser String beinhaltet mehrere Informationen:

1. Wurde ein Nutzer-Token benutzt?
2. Ist der Nutzer berechtigt, den Kühlschrank zu benutzen?
3. Nutzer-ID

Das Eintreffen des `Request-String` löst beim zweiten Arduino ein *Event* aus, welches die Aufnahme eines Fotos sowie dessen Speicherung auf der Miro-SD Karte triggert. Zusätzlich wird in einem Text-Log hinterlegt, welcher `Request-String` mit welcher Foto-Datei assoziiert ist. Dies beendet einen Protokollierungsauftrag. Das Analysieren der Kühlschranknutzung kann folglich durch die Betrachtung der Logs und den Vergleich mit dem Bildmaterial durchgeführt werden.

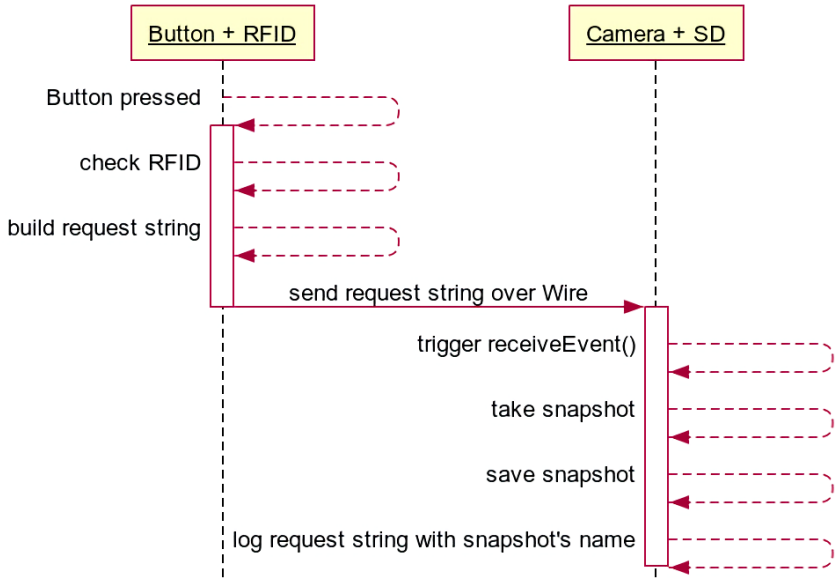

Abbildung 2: Funktionsablauf anhand eines Sequenzdiagramms

5 Eigene Library zur Kommunikation

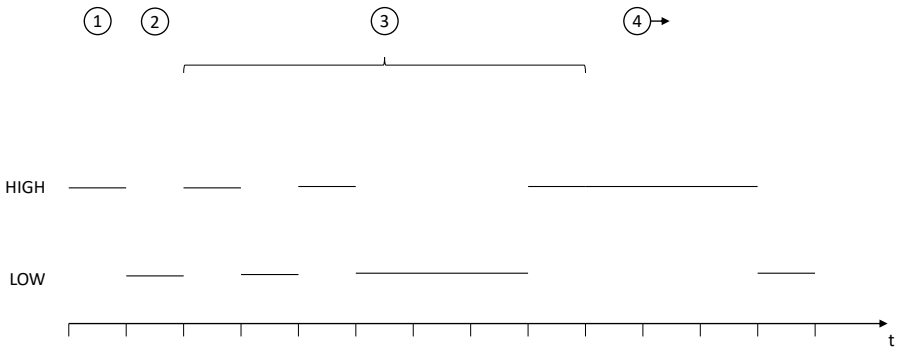

Abbildung 3: Ablaufschema Signalübertragung

Abbildung 1 zeigt die ursprüngliche Konfiguration des Projektes, in welcher die Kommunikation zwischen den Arduinos über die Wire-Bibliothek (I2C) und die Ports A4 (data line / Datenübertragung) und A5 (clock line / Synchronisation) abgewickelt wurde. Motivation für eine eigene Implementierung war der Wunsch, in einem Teilbereich des Projektes auf den Komfort und die Abstraktion einer heruntergeladenen Bibliothek zu verzichten. Geplant war außerdem der Einsatz eines 433 Mhz Sender/Receiver Paares anstelle einer Kabelverbindung, wesshalb das Protokoll ein Heartbeatsignal enthält.

Die Bibliothek sieht die Verwendung von nur einem Kabel über einen jeweils frei wählbaren Digitalpin vor. Abbildung 3 visualisiert die Impulse während einer Datenübertragung. Da es sich um ein binäres Signal handelt, wird nur zwischen HIGH und LOW unterschieden. Auf der x-Achse ist die Zeit aufgetragen. In der vorliegenden Konfiguration der Library entspricht eine Zeiteinheit 2ms. Werden keine Daten übertragen, so ist die Leitungs spannungsfrei (LOW). Sobald Kommunikationsbedarf besteht, wird ein HIGH-Implus versendet (1), welcher zugleich zur zeitlichen Synchronisation von Sender und Empfänger dient. Der zweite Implus fungiert als Heartbeat-Indikator. Ist er HIGH, so handelt es sich um ein Heartbeat-Signal und es sind keine weiteren Signale zu erwarten. Ist er LOW, so wechselt der Empfänger in den Datenempfangsmodus. Die Abschnitte (3,4) zeigen die Anfänge einer Datenübertragung; Das Byte in Abschnitt (3) beschreibt exemplarisch die ASCII-Codierung eines "Q".

```
1  class Sender
2  {
3  public:
4    Sender(int pin);
5    void heartbeat ();
6    void send(String* msg, int msgLength);
7  };
8
9  class Receiver
10 {
11 public:
12   Receiver(int pin, int heartbeatAlarm, void (*fireAlarm)(),void
       (*stopAlarm)() );
13   void listen(String* dest, int msgLength);
14 };
```

Listing 1: Verwendung der Connector Library

Listing 1 zeigt die Signaturen der entwickelten Bilbiothek. Die Initialisierung des Sender- und des Empfängerobjektes im Arduino-Sketch erfordert die Angabe des jeweils verwendetetn Pins. Darüber hinaus benötigt das Empfängerobjekt eine Millisekunden-Angabe, die spezifiziert, nach wie vielen Sekunden ausbleibenden Heartbeats ein Alarm ausgelöst werden soll.

Der auszuführende Code wird mit dem Funktionspointer *fireAlarm* referenziert; ist die Verbindung wiederhergestellt, ruft die Bibliothek *stopAlarm* auf. Der Alarm wird nur bei erstmaligem Ausbleiben des Heartbeats ausgelöst. Der Sender verfügt entsprechend über die parameterlose Funktion *heartbeat*, welche theoretisch beliebig oft aufgerufen werden kann.

Die *send*-Funktion erwartet einen Pointer auf den zu sendenden String sowie dessen Länge. Im Zuge des Sendevorgangs wird keine Kopie angelegt. Die *listen*-Funktion erwartet einen Pointer auf den String, in welchem die empfangenen Daten abgelegt werden sollen. Die Längenangabe muss mit der des korresponiderenden *send*-Aufrufes übereinstimmen. *listen* überwacht den Heartbeat bis zum Eintreffen einer Nachricht und terminiert, sobald der Empfangsvorgang abgeschlossen ist.

Das entwickelte Verfahren unterstützt lediglich Simplexübertragungen mit fixer Länge, ohne Prüfsumme und ohne zwischengeschaltete zeitliche Synchronisation bei langen Nachrichten. Die Einsatzmöglichkeiten sind daher sehr projektspezifisch.

6 Repository

`https://github.com/LeonardKoll/arduinofridge`
Der Master-Branch bietet Unterstützung für das I2C basierte Setup aus Abbildung 1. Der Branch i2cReplace beinhaltet die im Vortrag verwendete Codeversion mit eigener Kommunikationsimplementierung. Die entwickelte Library liegt unter lib/Connector.