Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

Dokumentacija implementacije

Smart Parking Sistem

Projekat iz predmeta Ugradbeni sistemi

Članovi tima: Džana Balta
Mea Hrbenić
Anida Nezović

Implementacija projekta se sastoji od dva .cpp fajla i jednog .kt fajla. Dva C++ kôda su pisana u okruženju Keil Studio, a treći kôd pisan je u okruženju Android Studio.

Ulaz na parking

Prvi programski kôd koji će biti objašnjen odnosi se na problem unosa šifre, provjere ispravnosti unesenog i realizacije slanja obavijesti na LCD displej. Kôd je odvojen u sljedeće cjeline:

1. Predprocesorske direktive (mbed biblioteka, te biblioteke neophodne za rad sa LCD displejom)
2. Definisane početne vrijednosti globalnih varijabli, koje koristimo u funkcijama
3. Prototipovi funkcija koje su vezane za rad aplikacije
4. Tijela gore navedenih funkcija

Najkompleksnija funkcija, te ujedno i funkcija koja nam je najviše problema stvarala jeste **`unosNaTastaturu()`**. Redom smo postavljali stanja 0111, 1011, 1101, 1110 na digitalne izlaze i za svako stanje očitavali digitalni ulaz. Time smo uspjeli iskoristiti kôd za unos, koji smo pisali na laboratorijskoj vježbi 3. Ispunjenje jednog od *if uslova*, znači da je taster pritisnut i da taj unos treba spasiti u neku varijablu, što je u našem slučaju niz charova **`uneseno`**. Varijabla **`rez`** čuva vrijednosti pojedinačnih digitalnih ulaza.

Zbog mehaničke nesavršenosti prekidača dešava se da kontakti zatitraju pri uključivanju[1], te smo zbog toga morali dodati uslove **`rez != '1', rez != '2', rez != '3'`** itd. Svakim pritiskom na taster, varijabla **`rez`** dobije odgovarajuću vrijednost, koja bi se stalno dodavala u niz charova **`uneseno`**, da nismo stavili pomenute uslove.

Ovim smo onemogućili da se u budućnosti kao ispravna šifra postavi kombinacija koja ima dva ista broja ili znaka jedan pored drugog. Znamo da se ovaj problem mogao riješiti upotrebom prekida i debouncinga, ali zbog prirode samog projekta, bili smo primorani da ostavimo gore opisano rješenje.

Funkcija **`provjeriSifru()`** treba provjeriti da li je unesena šifra jednaka 4242 ili bilo kojoj kombinaciji koju korisnik bude želio postaviti.

Funkcije **`defaultScreen()`**, **`promijeniObavijest()`** i **`greska()`** su vezane za rad sa LCD displejom. Upozorenje *Ulaz je zabranjen!* ispisat će se na crvenoj pozadini i to nam omogućava funkcija **`defaultScreen()`**. Poruka *Ulaz je slobodan!* na zelenoj pozadini, bit će ispisana uz pomoć funkcije **`promijeniObavijest()`**. Funkcija **`greska()`** omogućava da se korisnik obavijestiti o pogrešno unesenoj šifri, porukom *Pogresna sifra! Pokusajte ponovo.*

U **`main()`** funkciji prvo pozivamo funkciju **`defaultScreen()`** pretpostvljajući da šifra još uvijek nije unesena. U *while petlji* omogućili smo da se funkcija **`unosNaTastaturu()`** stalno poziva i da se prilikom svakog unosa vrši provjera. Kada korisnik unese ispravnu šifru, pojavit će se poruka o slobodnom ulazu, koja će biti aktivna pet sekundi. Nakon tih pet sekundi, ponovo se postavlja obavijest o zabrani, a prethodno unesena šifra se

[1] S. Konjicija, E. Sokić (2019) *Ugradbeni sistemi: Hardverski aspekti, Elektrotehnički fakultet Univerziteta u Sarajevu*

poništava i postavljaju se sve nule. Ukoliko šifra nije poništena, znamo da je riječ o pogrešno unesenoj.

Parkiranje vozila

U drugom C++ kôdu[2] koristimo analogni ulaz infracrvenog senzora, koji je spojen na razvojni sistem FRDM-KL25Z. Kako aplikacija na ovom razvojnom sistemu komunicira sa aplikacijom na android uređaju, moramo koristiti MQTT protokol, koji omogućava da ova dva klijenta razmjenjuju poruku posredstvom MQTT brokera. Da bismo omogućili bežičnu komunikaciju, potreban nam je mikrokontroler ESP8266, jer posjeduje WiFi modul. Iz tog razloga smo uključili biblioteke **"MQTTNetwork.h"**, **"MQTTmbed.h"**, **"MQTTClient.h"** i **"ESP8266Interface.h"**.

Varijabla **signal** se postavlja na 0 ili 1 u zavisnosti od očitavanja infracrvenog senzora. Isprobavanjem različitih očitavanja senzora, odlučili smo da kada senzor očita vrijednost veću od 1.3 V, što je otprilike 20 cm udaljenosti objekta od senzora, varijabla **signal** bude postavljena na 1. Taj signal moramo publishati na temu **SENZOR**, koju smo definisali na početku kôda kao **#define SENZOR "projekat/senzor".**

Treći kôd je pisan u programskom jeziku Kotlin. Kako postoje različite Android biblioteke koje se koriste pri implementaciji MQTT protokola, mi smo odlučili koristiti *Eclipse Paho* biblioteku.

Prvo smo kreirali klijenta koji će se pretplatiti na broker:

```
var client: MqttAndroidClient? = null
client = MqttAndroidClient(getApplicationContext(),
"tcp://broker.hivemq.com", MqttClient.generateClientId());
```

Varijable **brojSlobodnihMjesta, brojZauzetihMjesta, maxBroj** i **zauzeti** koristimo kako bismo vodili evidenciju o mjestima na parkingu. Treba napomenuti da trenutno naš parking ima samo jedno parking mjesto, jer smo koristili samo jedan senzor udaljenosti, pa se kôd mogao napisati i bez ovih varijabli.

Da bi aplikacija na android uređaju mogla primati podatke, ona mora biti spojena na temu **SENZOR** i to provjeravamo na sljedeći način:

```
if (topic!!.contains("projekat/senzor"))
```

Funkcija **provjeSignal(s:String)** kao parametar prima ***String*(message!!.getPayload())** što ustvari predstavlja vrijednost varijable **signal** iz drugog kôda. U skladu sa odgovarajućom vrijednošću, broj mjesta, koji se prikazuje na ekranu uređaja, mijenja se i time obavještava korisnika o zauzetosti mjesta na parkingu.

[2] Koristili smo kôd sa laboratorijske vježbe 6, te ga prilagodili našim zahtjevima.