TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDU40LT

Priit Trink 134300IAPB

DIISELMOOTORIGA SÕIDUAUTO EELSOOJENDUSE AUTOMATISEERIMINE

Bakalaureusetöö

Juhendaja: Enn Õunapuu
PhD
Dotsent

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Priit Trink

17.05.2016

Annotatsioon

Antud töö eesmärk on luua seade, mis automatiseerib ning laseb nutitelefoniga juhtida diiselmootoriga auto mootori ja salongi eelsoojenduse sisse lülitamist. Selleks, et seade luua, tuleb välja töötada riistvaraline lahendus, tarkvaraline lahendus ning nutitelefoni rakendus.

Põhiprobleemideks on auto aku eluea vähenemine külma mootoriga käivitades ning suurenenud heitgaaside hulk külma mootoriga sõitmisel. Lisaks eelsoojenduse manuaalselt sisse lülitamine võib olla kasutajale tülikas eriti, kui seadet kasutatakse regulaarselt (näiteks iga päev).

Töö tulemusteks on lülitusseadme riistvaraline ja tarkvaraline lahendus ning Androidi nutitelefoni rakendus. Lülitusseadmele saab sisse programmeerida ajagraafiku, mille järgi eelsoojendust sisse lülitatakse. Ajagraafik on sekundi täpsusega. Mootori- ja salongisoojendust saab sisse ja välja lülitada seadmel olevast nupust. Wi-Fi-võrgu siseselt seadme kontrollimiseks on Androidi rakendus, mis võimaldab soojendust ümber lülitada ning 24 tunni piires lülitust erakorraliselt ette tellida.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 5 peatükki, 13 joonist, 1 tabelit.

Abstract

Automation of a Diesel Car’s Preheating System

The purpose of this thesis is to create a device that automates and allows a smart phone to have control over a diesel car’s preheating system. To carry this out, a hardware and software solution is needed for the switching device and a creation of an app for a modern smartphone.

The main problems for this project are that starting up a diesel car with a cold engine may reduce the car’s battery life and a cold diesel engine produces more exhaust fumes than a warm one. Plugging in the preheating system manually can be obstructive for the user especially when the device is used every day. It may require a reminder or waking up earlier than needed. Keeping the heater turned on all the time is costly and unpractical.

The results of the project are the software and hardware solution for the switching device and a working app for an Android device. The switching device allows the user to program a timetable into it. The timetable points out when the preheating system should be turned on with an accuracy of one second. The user can also turn on the heater by pressing a button on the device. The device can be controlled over Wi-Fi with a smartphone. The phone’s application can switch the heater on or off and also allows to order a switch within 24 hours.

The thesis is in Estonian and contains 34 pages of text, 5 chapters, 13 figures, 1 table.

Lühendite ja mõistete sõnastik

LED	*Light-Emitting Diode*, valgusdiood
SSD	*Solid State Drive*, pooljuhtketas
SD	*Secure Digital*, välkmäluga mälukaardi formaat
USB	*Universal Serial Bus*, universaal-jadasiin
IP	*Internet Protocol*, internetiprotokoll
JSON	*JavaScript Object Notation*, lihtsustaud andmevahetusvorming, mis põhineb JavaScripti programmeerimiskeele alamhulgal
bps	*Bits per second*, bitte sekundis, andmeedastuskiiruse ühik
DHCP	*Dynamic Host Configuration Protocol*, dünaamiline hostikonfiguratsiooni protokoll
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
NTP	*Network Time Protocol*, võrguaja protokoll
MVC	*Model-view-controller*, mudel-vaade-kontroller
XML	*Extensible Markup Language*, laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

1.1 Taust ja probleem

Külma diiselmootoriga sõitmine võib kahjustada auto komponente ja ka keskkonda. Külm diiselmootor tekitab rohkem tahma kui soe mootor ning käivitamine on raskem, mis viib aku kiirema tühjenemiseni. Kuna Eestis esineb talvisel perioodil külmakraade, siis probleem on aktuaalne – eriti lühikestel sõitudel linnas, kus mootor ei jõua oma optimaalse töötemperatuurini.

Samuti on kasulik, kui enne autoga sõitma asumist on selle salong soe. See aitab auto kasutaja aega kokku hoida, kuna tuuleklaasilt jää kraapimine on oluliselt lihtsam (kui mitte ebavajalik, sest jää klaasilt on sulanud). Niimoodi välditakse ka kriimude tekkimist esiklaasile. Ühtlasi on kasutajal mõnusam istuda sooja autosse.

Selleks, et ennetada külma mootori käivitamist, on loodud toode Defa WarmUp [1], mis soojendab nii auto mootorit, kui ka salongi. Selleks, et seadet kasutada, tuleb ühendada voolukaabel auto küljes olevasse pistikusse. Seejärel hakkab vastavalt voolu olemasolule eelsoojendus tööle.

Eelsoojendust ei ole mõistlik kasutada igal ajahetkel, kui see on võimalik, kuna eelsoojendus tarbib voolu võimsusega 1400W ning Defa käsiraamat ütleb, et üle kolme tunni ei tasu seadet sees hoida [2]. Sellest tuleneb ka probleem: juhtme ühendamine autoga iga kord, kui tahetakse eelsoojendust kasutada, on ebamugav. See nõuab tihti meeldetuletust või varem üles ärkamist.

Probleemi lahenduseks on seade, mis lülitab eelsoojendust sisse ja välja vastavalt ajagraafikule ning võimaldab teha erakorralisi lülitusi nutitelefonist. Sellise seadme ehitamist käesolev projekt uuribki.

Töö on kasulik targa kodu inseneridele, inimestele, kes kasutavad Defa eelsoojendust ning ka Defa eelsoojendussüsteemide tootjatele, kellel oleks potentsiaali antud lülitusseadet toota.

Töö praktiline ja kirjalik osa valmivad kodus. Prototüübi testimine toimus 2015 sügisel ja 2016 talvel, olles igapäevases kasutuses töö autori poolt.

1.2 Ülesande püstitus

Antud töö **eesmärgid**:

1. töötada välja riistvaraline lahendus projekti jaoks;
2. töötada välja mikrokontrolleri tarkvara;
3. töötada välja nutitelefoni rakendus.

Nõuded lülitusseadmele:

1. võimaldada teha eelsoojenduse lülitusi ajagraafiku alusel. Ajagraafik võib korduda kasutaja valitud arv nädala tagant;
2. võimaldada teha lülitusi nutiseadmest;
3. võimaldada nutiseadmest ette tellida eelsoojendust vabalt valitud kellaajal 24 tunni piires;
4. võimaldada teha lülitusi seadme pealt.

1.3 Metoodika

Eesmärkide saavutamiseks uuritakse erinevaid riistvaralisi tehnoloogiaid, mis võimaldaksid kõikide nõuete täitmist. Valitakse välja sobiv võrgutehnoloogia, mille abil lülitusseade ja nutitelefon hakkavad omavahel suhtlema.

Töötades välja seadme riistvaralise lahenduse, projekteeritakse ning valmistatakse vastav tarkvara nii lülitusseadmele, kui ka nutitelefonile.

Lahendust testitakse valmimise käigus igapäevastes kasutustingimustes.

Töö tegemisel tuginetakse tootjate poolt Internetis leitavatele juhistele ning Tallinna Tehnikaülikooli informaatika erialalt saadud teadmistele.

1.4 Ülevaade tööst

Töö on jaotatud peatükkideks vastavalt projekti eesmärkidele – esimeses peatükis tuleb juttu riistvaralise lahenduse uurimustest ning tulemustest, teises räägitakse lülitusseadme tarkvaralisest lahendusest ning kolmandas nutitelefoni rakendusest.

2 Riistvaraline lahendus

Antud peatüki eesmärk on uurida ning töötada välja riistvaraline lahendus antud projektile arvestades järgmiseid nõudeid:

1. suudab olla funktsioneeriv seade Wi-Fi-võrgus;
2. suudab teha lülitusi 220V vooluringis;
3. kuvab kasutajale infot eelsoojenduse ja seadme oleku kohta;
4. võimaldab teha lülitust seadme pealt.

2.1 Mikrokontroller

Mikrokontrolleri nõuded on:

- peab olema ühendatav elektriskeemi;
- peab olema ümberprogrammeeritav;
- peab olema kasutajasõbralik moodus, et lisada juurde mooduleid.

Mikrokontrolleri valimisel kaaluti kahte prototüüpimiseks mõeldud arendusplaati: Arduinot ning Raspberry Pid.

Raspberry Pi on pisike, ligikaudu pangakaardi-suurune trükkplaadist koosnev arvuti [3]. Projekti jaoks kõige sobivamate omadustega mudel oleks „Model A+“, mis maksab Oomipoes 37€ [4]. Arvutil puudub SSD või kõvaketas. Selle tõttu oleks vaja operatsioonisüsteemi jooksutamiseks osta SD-kaart, mis maksab lisaks ~10€. Raspberry Pi operatsioonisüsteem on Raspian, mis baseerub Debianil [5]. Tänu sellele saab seadmega kasutada väga palju erinevaid programmeerimistehnoloogiaid. Samuti on võimalus Raspberry Piga kasutada kolmanda osapoole operatsioonisüsteeme.

Arduino on avatud lähtekoodiga ühe plaadi mikrokontroller [6]. Selle küljes on sisend- ja väljundviigud, mille abil on lihtne anda mikrokontrollerile sisendeid ja saada vastavalt programmile väljundeid. Programmeerimine Arduino peale käib Wiring keeles, mis on sarnane keelega C++ mõne muudatuse ja lihtsustusega [6]. Programm kompileeritakse

arendusmasinas Arduino programmiga, mis töötab Windowsil, Macil kui ka Linuxil. Peale kompileerimist, laeb Arduino tarkvara programmi koodi mikrokontrollerisse läbi USB kaabli. Kõige sobivam Arduino mudel antud projektiks on Leonardo. Erinevalt Arduino klassikalisest mudelist Uno, on Leonardol mikro-USB pesa ning Oomipoes maksab Leonardo 4€ vähem ehk 22€ (Uno hind on 26€). Mikro-USB on antud projekti puhul parem, kuna Androidi telefonidel on sama pesa ning selle tõttu on arenduse käigus vaja vähem erinevaid kaableid.

Mikrokontrolleritest otsustati kasutada Arduino Leonardo arendusplaati. Raspberry Pi on antud projekti jaoks liiga paljude võimalustega ning selle tõttu suur osa selle arendusplaadi potentsiaalist oleks kadunud. Arduino kontroller on odavam ning antud projekti raames piisavate omadustega.

2.2 Seadme aeg ja traadita side

Töö eesmärgiks on võimalus saada lülitada seadet sisse/välja nutitelefonist. Selle tõttu tuleb välja valida kasutatav tehnoloogia, mis seda võimaldaks. Vältimaks juhtmete rägastikku, otsustatakse kasutada mõnda traadita side tehnoloogiat.

Esimene kaalutav tehnoloogia on Bluetooth. Eelmainitu on hea, kuna selle kasutamine ei nõua peale nutitelefoni ja projekteeritava seadme ühtegi kolmandat elektroonilist seadet (näiteks ruuter). Lisaks on Bluetoothiga seadmete ühendamine oluliselt lihtsam, kui Wi-Figa, kuna see ei nõua sisse programmeeritult ühtegi nime ega parooli.

Küll aga Bluetoothi kasutamine jätab alles ühe probleemi. Nimelt selleks, et Arduino mikrokontroller teaks ajagraafiku järgi lülitusi teha, on sellel vaja teada hetkelist aega. Bluetoothi ei saaks aja saamiseks kasutada, kuna see nõuaks kohe peale seadme algseadistamist ühendust nutitelefoniga. See võib olla küllaltki tülikas.

Ajaprobleemi lahendamiseks saab panna mikrokontrollerile juurde kellamooduli, mis töötab oma patarei toitel. Seda ei ole tarvis teha, kui sidetehnoloogiana kasutada Wi-Fit. Tänu Wi-File oleks seadmel Internetiühendus ning Internetist on võimalik hetkeline aeg piisavalt täpselt kätte saada. Seega sidetehnoloogia valikuks osutub Wi-Fi.

Kuna Arduino arendusplaadil puudub originaalis Wi-Fi moodul, siis see tuleb seadmele eraldi osta ning külge panna. Arduino arendusplaadid on disainitud sellisena, et neile

oleks võimalik lihtsalt juurde lisada erinevaid mooduleid, ilma et sisend- või väljundviikusid läheks kaduma. Selliseid mooduleid, mida saab Arduino peale kinnitada kutsutakse kilpideks [7].

Wi-Fi kilbina kasutatakse CC3000 WiFi Shield Expansion Boardi, mis Ebays maksab 18.22€ [8]. Kaardil on mikro-SD pesa, mida saab kasutada päringutele vastamiseks. Lisamooduli tööle panemiseks seadme tarkvaras tuleb kasutada Adafruit CC3000 teeki [9]. Teegi kasutamisest tuleb täpsemalt juttu mikrokontrolleri tarkvara osas.

2.3 Seadme muud riistvaralised komponendid

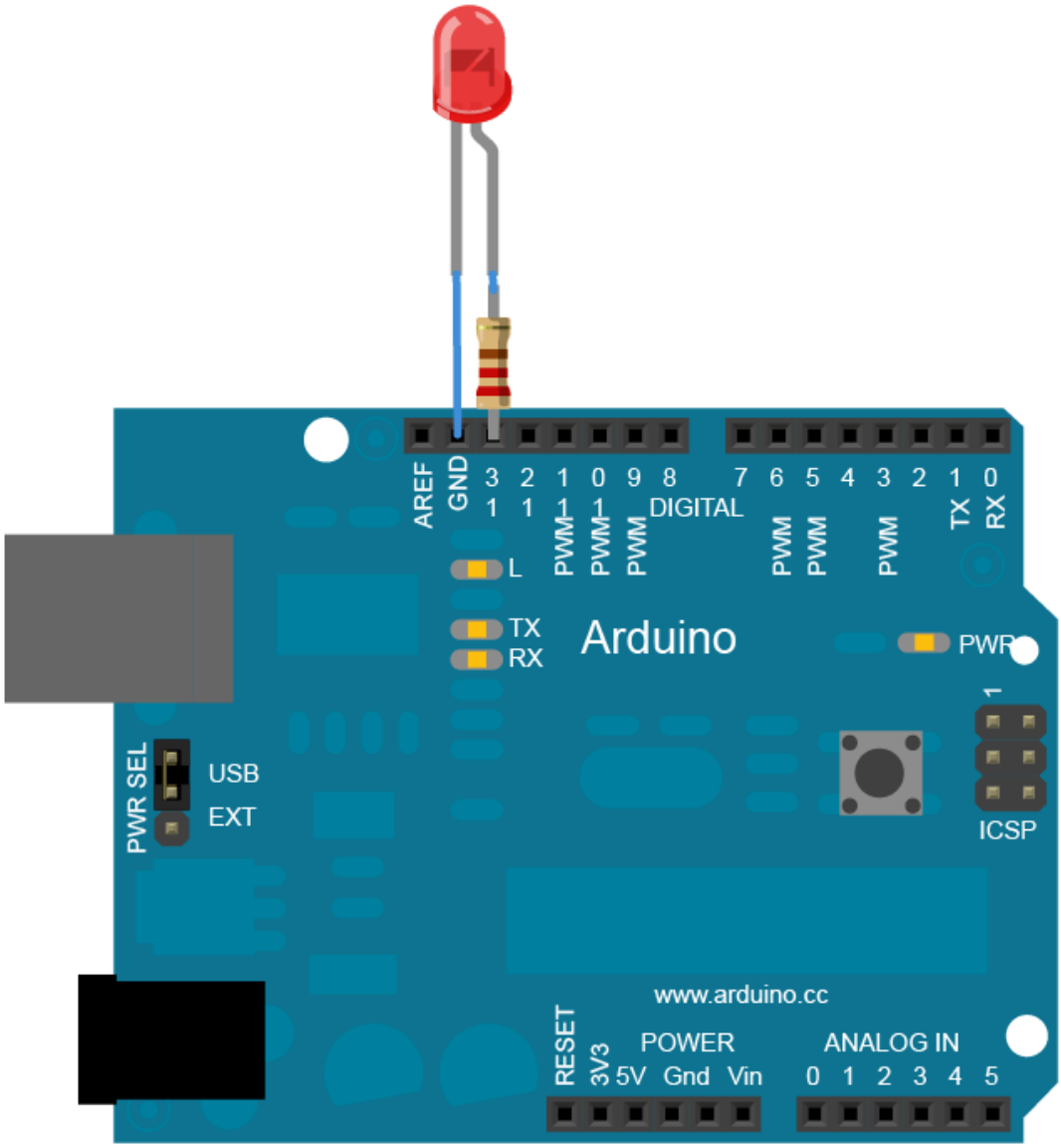
Selleks, et Arduinoga oleks võimalik teha lülitusi 220 voldises vooluvõrgus, on tarvis releemoodulit [10]. Moodul ühendub kontrolleriga läbi toite-, maandus- ja kontrolljuhtme. Kui kontrolljuhtmes on kõrgem pinge (5V), laseb releemoodul 220V pingega voolu läbi. Kui kontrolljuhtmes on madalam pinge (0V), siis releemoodul voolu läbi ei lase. Arduinol jooksev programm juhib seda, kui suurt pinget kontrolljuhtmesse lastakse.

Elektriskeemi prototüüpimiseks osutub valituks 840-augune maketeerimislaud Oomipoest [11]. Valitakse suur maketeerimislaud, et hiljem saaks skeemi lihtsalt ümber disainida. Hea disain on oluline, kuna prototüübi reaalsetes tingimustes kasutamine peab olema efektiivne ja mugav.

Maketeerimislaua peale pannakse punane ja roheline LED. LED-ide eesmärgid on:

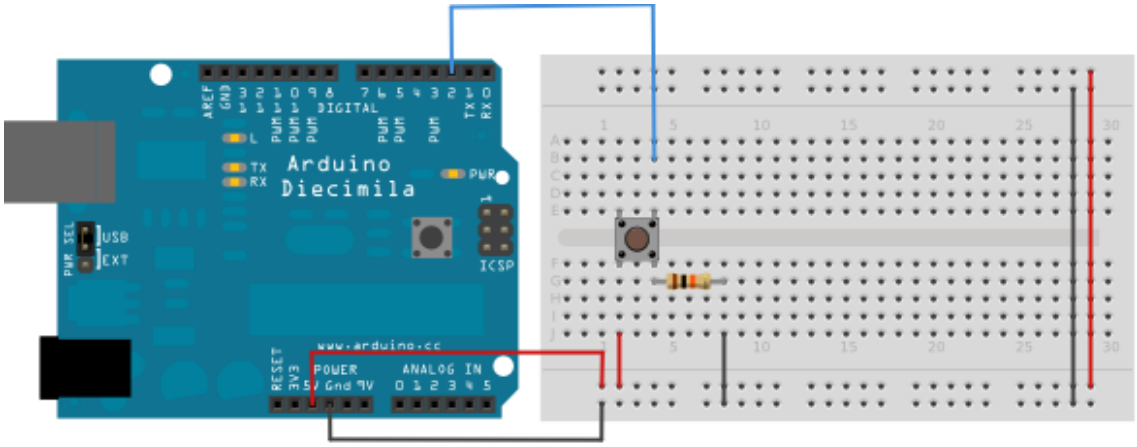
- kuvada infot eelsoojenduse töötamise kohta. Punane LED põleb siis, kui ei eelsoojendus tööta ning roheline siis, kui töötab;
- kuvada vea- ja progressiteateid erinevat tüüpi vilkumistega;
- kuvada IP-aadressi 8 viimast bitti. IP-aadress koosneb 32 bitist, kuid kõiki ei ole tarvis näidata, sest koduses Wi-Fi-võrgus on kõikidel seadmetel esimesed 24 bitti samad. IP-aadressi kuvamine teeb arendusprotsessi lihtsamaks, kuna vigasid on lihtsam tuvastada. IP-aadressi kuvatakse kaheksa vilkumisega. Vilkumise teeb kas roheline või punane LED (kindlasti mitte korraga). Roheline tähistab bitti 1 ja punane bitti 0.

LED-i ühendamisel Arduinoga on tarvis 220 oomist takistit, mis tuleb paigutada enne LED lampi (Joonis 1. Arduino LED-iga ühendamise skeem.



Joonis 1. Arduino LED-iga ühendamise skeem. Allikas: https://www.arduino.cc/en/tutorial/blink.

Lülituste tegemiseks seadme küljest pannakse maketeerimislaua külge nupp. Lühike vajutus nupule peab tegema eelsoojenduse ümberlülituse. Pikk vajutus, mis kestab rohkem kui 3 sekundit, kuvab kasutajale LED-ide abil IP-aadressi. Nupu lisamisel on tarvis 10 kilo-oomist takistit (Joonis 2. Nupu ühendamine Arduinoga. Allikas: https://www.arduino.cc/en/Tutorial/Button).



Joonis 2. Nupu ühendamine Arduinoga. Allikas: https://www.arduino.cc/en/Tutorial/Button.

3 Mikrokontrolleri tarkvara

Antud peatüki eesmärk on uurida Arduino plaadi programmeerimise võimalusi ning valmistada tarkvara, mis järgiks järgmisi nõudmisi:

1. Wi-Fi-kaart peab funktsioneerima saades ühendust Internetiga ning reageerides HTTP päringutele;
2. kuvatakse infot seadme seisundi kohta LED-ide abil;
3. antakse nupule funktsionaalsus;
4. omatakse kontrolli relee lülituste üle.

Iga Arduino koodi projekt nõuab ino-laiendiga faili, mille tunneb ära Arduino tarkvaraarenduseks mõeldud programm. Selles failis peab olema kaks meetodit: `setup()` ja `loop()` (Joonis 3). Arduino programm kontrollib, kas fail on õigesti kirjutatud, kompileerib koodi ning soovi korral paigaldab selle läbi USB kaabli arendusplaadile. Ino-faili kood on mõne üksiku lihtsustusega süntaktiliselt sama C++ keeles kirjutatud koodiga [6]. Kui ino-fail on olemas, saab projekti lisada veel faile, mille faililaienditeks on .h (päisefail), .c (programmeerimiskeele C fail), ning .cpp (programmeerimiskeele C++ fail). Käesolevas projektis kasutatakse C++ keelt, kuna erinevalt C-st saab sellega kasutada klasse. Klassid on vajalikud, et järgida objektorienteeritud programmeerimise põhimõtteid.

```
// Globaalsete muutujate ja sõltuvuste loomine.

void setup() {
    // Funktsioon, mis kutsutakse esile programmi
    // käivitades.
}

void loop() {
    // Funktsioon, mis kutsutakse korduvalt esile
    // seni kuni plaat välja lülitatakse.
}
```

Joonis 3. Arduino programmi kohustuslikud meetodid.

Mikrokontrolleri tarkvara arendamiseks kasutatakse tekstiredaktorit Sublime Text 3. Programmikoodi kompileerimiseks kasutatakse Arduino arendustarkvara.

Programmi testitakse projektis kasutatava Arduino arendusplaadiga. Programmi silumisel kasutatakse Arduino kompilaatoriga kaasa tulevat staatilist klassi `Serial`, mis saadab Arduino plaadi programmist tuleva info läbi USB kaabli arvutisse (Joonis 4).

```
Serial.begin(9600); // Avab pordi andmeedastuskiirusega
                    // 9600 bps. Meetod tuleb programmi
                    // käivitades välja kutsuda.

Serial.print("Tere, maailm"); // Saadab läbi USB kaabli
                              // info "Tere, maailm", mida
                              // suudab lugeda Arduino
                              // arendustarkavara.
```

Joonis 4. Arduino programmi silumine.

3.1 Kasutajaliides

3.1.1 Seadme nupp

Maketeerimislaual oleval nupul on kaks eesmärki: teha kohene ümberlülitus ning nupu 3-sekundise peal hoidmise korral kuvada kasutajale seadme IP-aadress. Seega peab tarkvara tuvastama nii lühikese vajutuse kui ka pika vajutuse.

Selleks, et nupu vajutust tuvastada, tuleb defineerida, milliselt viigult nupp vajutuse korral signaali annab. Programmikoodis saab seda teha meetodiga `pinMode(pinNumber, INPUT)`. `pinNumber` defineerib, milliselt viigult nupu signaal tuleb Arduino plaadile ning `INPUT` defineerib, et tegu on sisendviiguga. Nupuvajutusel tulevat signaali saab tuvastada loogikatehtega `digitalRead(pinNumber) == HIGH`, kus `pinNumber` on viigunumber ning `HIGH` tähistab kõrgemat pinget, mida nupuvajutus esile kutsub.

Kui kutsuda esile eelsoojenduse ümberlülitus ainult kontrollides eelmainitud loogikatehet, siis seade teeks ümberlülituse iga kord, kui programm oma tsüklit kordab. See tähendab, et kui nupu vajutamisel hoitakse nuppu mõnda aega all, siis ümberlülitamist jõutakse esile kutsuda mitu korda. Et probleemi vältida, tehakse reegel, et eelsoojendus lülitatakse ümber alles siis, kui nupp peale vajutust lahti lastakse.

Nupu kolme sekundi peal hoidmise tuvastamiseks on vaja programmil lugeda aega. Siis kui nuppu on all hoitud nõutud aeg, tühistatakse nupu lahtilaskmisel üleskutsutav ümberlülitus ning selle asemel kutsutakse välja meetod, mis kuvab kasutajale IP-aadressi.

Nupu funktsionaalsuse tagamiseks on loodud sellele eraldi klass `Button`. Antud klass sõltub objektidest `Timetable` (kutsub esile relee lülituse) ning `Alerter` (kutsub esile kasutajale nähtavad teated LED-ide abil).

3.1.2 Märguande LED-id

Maketeerimislaual asuvate LED-ide eesmärgiks on kuvada järgmiseid teateid:

1. veateated;
2. progressiteated seadme käimapanemisel;
3. IP-aadressi 8 viimase biti kuvamine;
4. eelsoojenduse oleku kuvamine.

Teadete 1-3 eest vastutab objekt `Alerter`. Teate 4 eest vastutab objekt `Relay`, mille põhieesmärgiks on teha lülitus releemoodulis.

Igal LED-il on Arduino arendusplaadi peal oma viik. Viigu defineerimiseks on vaja programmi käivitamisel kutsuda välja meetod `pinMode(pinNumber, OUTPUT)`. `OUTPUT` tähendab, et tegu on seadme väljundiga. LED-i põlema panemiseks on vaja Arduino arendusplaadiga välja saata vastavast viigust kõrgem pinge ning LED-i kustutamiseks madalam pinge. Programmi koodis saab kõrgemat pinget esile kutsuda meetodiga `digitalWrite(pinNumber, HIGH)` ning madalat pinget analoogselt `digitalWrite(pinNumber, LOW)`.

Antud projekti tarkvaras on iga LED defineeritud klassiga `Led`, mis on teistest klassidest sõltumatu. `Led`i konstruktori argumendiks on viigu number. Klassi liides on disainitud nii, et kood oleks võimalikult lihtne ja loetav. Näiteks punast LED-i saab peale defineerimist välja kutsuda lausega `redLed.on()`.

Eelpool sai mainitud, et teadete ja IP-aadressi kuvamisel kasutatakse klassi `Alerter`. `Alerter` sõltub kolmest `Led` objektist: roheline LED, punane LED ja Arduino plaadi peal olevast väiksest sisse-ehitatud kollasest LED-ist. `Alerter` koosneb meetoditest, mis kutsuvad välja erinevaid kombinatsioone LED-ide vilkumistest (Joonis 5).

```
void Alerter::displayIPInBinary() {
	this->storeOldStatuses();
	this->green->off();
	this->red->off();
	delay(1500);

	unsigned char number = this->lastIpNumber;

	for (int i = 0; i < 8; i++) {
		if (number & 0x80) { // when bit is 1
			this->green->on();
			delay(750);
			this->green->off();
			delay(600);
		} else { // when bit is 0
			this->red->on();
			delay(750);
			this->red->off();
			delay(600);
		}

		number = number << 1;
	}

	delay(1500);
	this->resetOldStatuses();
}
```

Joonis 5. Näide klassi Alerter meetodist.

3.2 Võrgutoimingud

Seadmel on kaks erinevat võrgutoimingut: päringu tegemine Internetist ning päringutele reageerimine. Samuti on tarvis seade ühendada Wi-Fi-võrku. Adafruit CC3000 teek tuleb koos näidetega, kuidas neid võrgutoiminguid teha.

Wi-Fi-võrku ühendamisel on vaja defineerida kolm konstanti: võrgu nimi, võrgu parool ning võrgu andmeturbeprotokoll. Kasutatava teegiga võrku ühendamisel on vaja esmalt välja kutsuda meetod, mis ühendab seadme Wi-Fi pöörduspunkti. Seejärel kutsutakse välja meetod, mis DHCP abil hangib seadmele IP-aadressi (Joonis 6).

```
void Wifi::connect() {
      if (!this->cc3000->connectToAP(WLAN_SSID, WLAN_PASS, WLAN_SECURITY)) {
        this->alerter->displayError();
        while(1);
      }
      this->alerter->progress(2);

      /* Wait for DHCP to complete */
      while (!this->cc3000->checkDHCP()) {
        delay(100);
      }
      this->alerter->progress(3);

      while (!displayConnectionDetails()) {
        delay(1000);
      }
}
```

Joonis 6. Wi-Fi-võrku ühendamine.

Wi-Fi-võrguga ühendamine tehakse klassis `Wifi`. `Wifi` on sõltuv klassidest `Adafruit_CC3000` ning `Alerter`, mis annab kasutajale infot ühendamisprotsessist või vigadest.

Eeldades, et seade on saanud ühenduse võrguga, kus Internet on kättesaadav, on võimalik teha seadmega veebist päringuid. Seadmel on vaja teha Internetist päring selleks, et teada saada ajatempel. Ajatempel tuleneb NTP ajaserverist ning võrdub sekunditega alates 1. jaanuarist 1970 [12].

Eeldades, et seade on ühendatud Wi-Fi-võrku, on võimalik esitada seadmele HTTP päringuid. Erinevad GET päringud, mida antud projektis seadmele esitatakse, on kolm:

1. `/` – eelsoojenduse oleku kuvamiseks. Tagastab kas tõese või väära.
2. `/switch` – kutsub välja eelsoojenduse ümberlülituse. Tagastab eelsoojenduse uue oleku: tõene või väär.
3. `/<ajatempel>` – lisab erakorralise lülituse ajatempli programmi mällu ning aktiveerib selle. Pärast erakorralise eelsoojenduse elluviimist selle olek deaktiveeritakse, et järgmisel ajatsüklil erakorraline tellimus ei jõustuks. Tagastab hetkelise eelsoojenduse oleku.

3.3 Seadme aeg

Seadme ajaga seotud nõuded:

1. eelsoojendus võib töötada ainult nii kaua, kui kasutaja on selle ette määranud. See tähendab, et kui kasutaja on lülitanud eelsoojenduse sisse, siis programm lülitab selle automaatselt välja näiteks ühe tunni pärast. Sellel moel välditakse liigseid elektrikulusid, kui eelsoojendus unustatakse sisse;
2. ajagraafikut peab olema võimalik teha iga päeva kohta eraldi ning graafik peab korduma vastavalt kasutaja valitud arvu nädala kaupa. See tähendab, et kui kasutaja ajagraafik erineb üle nädala, siis tal on võimalus teha kummagi nädala kohta eraldi ajakava ning see jääb korduma iga kahe nädala tagant.

Ajatempleid hoitakse Arduino programmis andmetüübiga *unsigned long*. *Unsigned* sellepärast, et negatiivseid arve ajatemplites ei eksisteeri. *Long* selle tõttu, et Arduino programmides *long* on suurusega 4 baiti erinevalt muude programmeerimiskeeltega, kus *long*i suurus on 8 baiti. Kuna *int* on Arduino programmides suurusega 2 baiti, siis *int*i erinevate väärtuste arv oleks liiga väike, et hoida nädalajagu sekundeid (erinevaid väärtusi saab olla ainult $2^{16} = 65\,536$).

Selleks, et Arduinol jooksev programm suudaks hoida järge ajast, on Arduinole tehtud eraldi teek, mis ei nõua välist riistvara. Ajateek suudab ajast järge pidada sekundi täpsusega. Programmi käivitades on vaikimisi aeg 0, kuid seda saab muuta, kasutades meetodit `setTime(timeStamp)`.

Eelsoojenduse töötamisaega saab piirata jättes mällu ümberlülituse tegemise aja. Eeldame, et süsteemi mälus on loodud konstant, kui kaua sekundites võib eelsoojendus töötada. Iga kord, kui programm jõuab uude tsüklisse, kontrollib see aega meetodiga `now()`. Kui `now()` väärtus on suurem, kui ümberlülituse tegemise aja ja lubatud töötamise aja summa, siis lülitatakse eelsoojendus välja.

Selleks, et programm suudaks kinni pidada ajagraafikust nädalate kaupa luuakse reegel: kui aja väärtus ületab nädalate arvu ja sekundite arvu nädalas korrutisega, siis aeg võrdsustatakse nulliga ning hakkab uus ajaline tsükkel. See tähendab, et kui seade pannakse käima, peab süsteem tuvastama oma aja vastavas tsüklis. Eelmises peatükis oli juttu sellest, et seade saab pärida aega Internetist. Internetist päritav aeg on võrdne

sekunditega alates 1. jaanuarist 1970 ning selle tõttu on vaja ajatempel ümber arvutada selliseks, et see ühtiks kasutaja ajalise tsükliga (Joonis 7). Ajaline tsükkel peab algama kell 00.00 esmaspäeval.

```
aeg = (ajatempel - kalibratsioon) % (nädalate_arv * sekundid_nädalas);
```

Joonis 7. Ajatempli konverteerimine.

3.3.1 Abiprogramm ajagraafiku loomiseks

Ajagraafikut hoitakse programmis ajatemplite massiivina. Kuna ajatemplid on sekundites alates mingisuguse nädala esmaspäevast, on neid ükshaaval küllaltki tülikas välja arvutada.

Probleemi lahenduseks tehakse abiprogramm, mis võtab sisendiks inimesele loetava JSON faili (Joonis 8) ning väljastab vastavalt infole C keele süntaksiga massiivi ning muud mikrokontrolleri tarkvarale vajalikud parameetrid (Joonis 9). Programmi väljund tuleb hiljem sisestada Arduino plaadi programmikoodi.

```
[
	[
		{
			"day": "Mo",
			"time": "9:45"
		},
		{
			"day": "Tu",
			"time": "9:45"
		},
		{
			"day": "We",
			"time": "9:45"
		},
		{
			"day": "Fr",
			"time": "9:45"
		}
	],
	[
		{
			"day": "Mo",
			"time": "12:30"
		},
		{
			"day": "We",
			"time": "9:45"
		},
		{
			"day": "Th",
			"time": "14:15"
		}
	]
]
```


Joonis 8. Abiprogrammi sisend.

```
const int weekCount = 2;
const int numOfIntervals = 7;

times[0] = 35100L;
times[1] = 121500L;
times[2] = 207900L;
times[3] = 380700L;
times[4] = 649800L;
times[5] = 812700L;
times[6] = 915300L;
```

Joonis 9. Abiprogrammi väljund.

Rakendus kirjutatakse Pythonis, kuna programmi nõuded on lihtsad ning programmeerimine antud keeles samuti. Pythoni programm ei nõua kompileerimist ning

JSON koodi töötlemine on väga programmeerija sõbralik. Programm on kirjutatud lähtudes objektorienteeritud programmeerimise tavadest.

3.4 Eelsoojenduse lülitusloogika

Eelsoojendust võib sisse lülitada kolm tegurit:

1. ajagraafik;
2. erakorraliselt tellitud aeg;
3. hetkeline ümberlülitus.

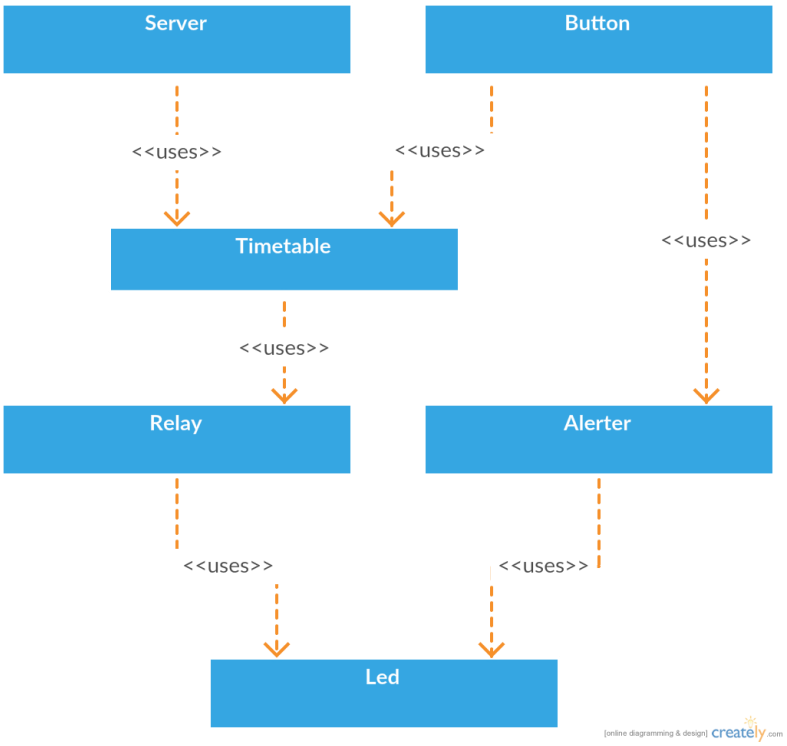
Kõigi eelnevate tegurite olekud on kas tõesed või väärad, seega erinevaid loogikajuhtumeid on $2^3 = 8$ (Tabel 1). Tänu sellele tekib hea võimalus ühe teguriga tühistada teine. Näiteks kui ajagraafik on põhjustanud eelsoojenduse sisse lülitamise, saaks ümberlülituse või ette tellitud ajaga selle tühistada kontrollides kõikide tegurite olekuid.

Tabel 1. Eelsoojenduse lülitusloogika. 0 – ei tööta/vale, 1 – töötab/tõene.

Ajagraafik	**Erakorraline aeg**	**Ümberlülitus**	**Eelsoojendus**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

3.5 Tulemused

Mikrokontrolleri tarkvara ehitati üles kasutades objektorienteeritud programmeerimise tavasid. Mainitud programmeerimistehnika on projekti puhul oluline, kuna nõudmiste realiseerimiseks on tarvis kirjutada palju koodi ning objektorienteeritud programmeerimise tõttu tekkiv liides teeb koodi oluliselt paremini loetavaks ning hallatavaks. Kood on jaotatud klassidesse ning nende vahel on vajalikud sõltuvused (Joonis 10).



Joonis 10. Lihtsustatud klassidiagramm sõltuvustega.

Tarkvara kasutab 97% Arduino Leonardo arendusplaadi mälust. Aruduino kompilaatori info alusel kasutab programm 28 032 baiti olemasolevast 28 672 baidist. See tähendab, et antud projekti jaoks on mälu piisavalt, kuid edaspidine funktsionaalsuse lisamine on väga piiratud. Probleem on lahendatav, ostes suurema mäluga Arduino arendusplaadi või ostes mõne muu mikrokontrolleri, mis nõuaks antud tarkvara konverteerimist.

Lülitusseade täidab esitatud nõuded edukalt. Vea- ja progressiteateid kuvatakse, eelsoojenduse olekut kuvatakse, vajutus ja pealhoie makteerimislaual oleval nupul töötavad, Internetist ajatempli saamine ja nõutud ajatsüklisse konverteerimine toimib (seade peab järge ajagraafikust) ning seade reageerib HTTP päringutele nii nagu nõutud. Seadet on testitud igapäevastes tingimustes ning on piisavalt usaldusväärne, et seda igapäevaelus kasutada.

4 Nutitelefoni rakendus

Antud peatüki eesmärk on uurida Androidi nutitelefonidele rakenduste tegemist ning konstrueerida programm vastavalt kolmele funktsionaalsele nõudele:

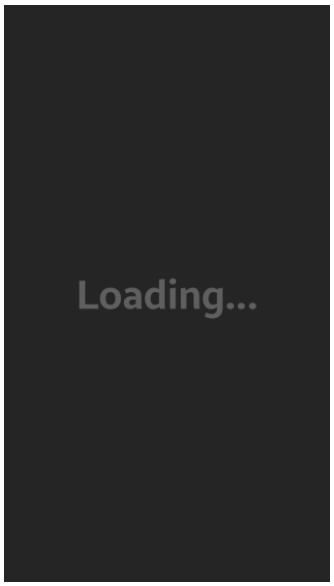
1. peab saama mikrokontrollerilt kätte eelsoojenduse oleku;
2. peab suutma esile kutsuda eelsoojenduse ümberlülituse;
3. peab saama 24 tunni piires ette tellida erakorralist eelsoojendust.

Rakenduse arendamiseks kasutatakse Android Studio programmi [14]. Tegu on tasuta integreeritud programmeerimiskeskkonnaga, mis baseerub JetBrainsi Intellij IDEA tarkvaral ning on spetsiaalselt mõeldud Androidi rakenduste tegemiseks. Android Studio on sobivaim vahend nutitelefoni rakenduste tegemiseks Androidile, kuna see sisaldab palju šabloone ning uut projekti alustades loob programm MVC mustri alusel projekti jaoks vajalikud kaustad ning failid. Android Studios kasutatakse automatiseeriva tööriistana vaikimisi Gradle'it. Programmeerimiskeeled, mida kastutatakse on XML ja Java. XMLis tehakse Androidi projektis kujunduslikud lahendused. Javas antakse rakendusele funktsionaalsus.

4.1 Kasutajaliides

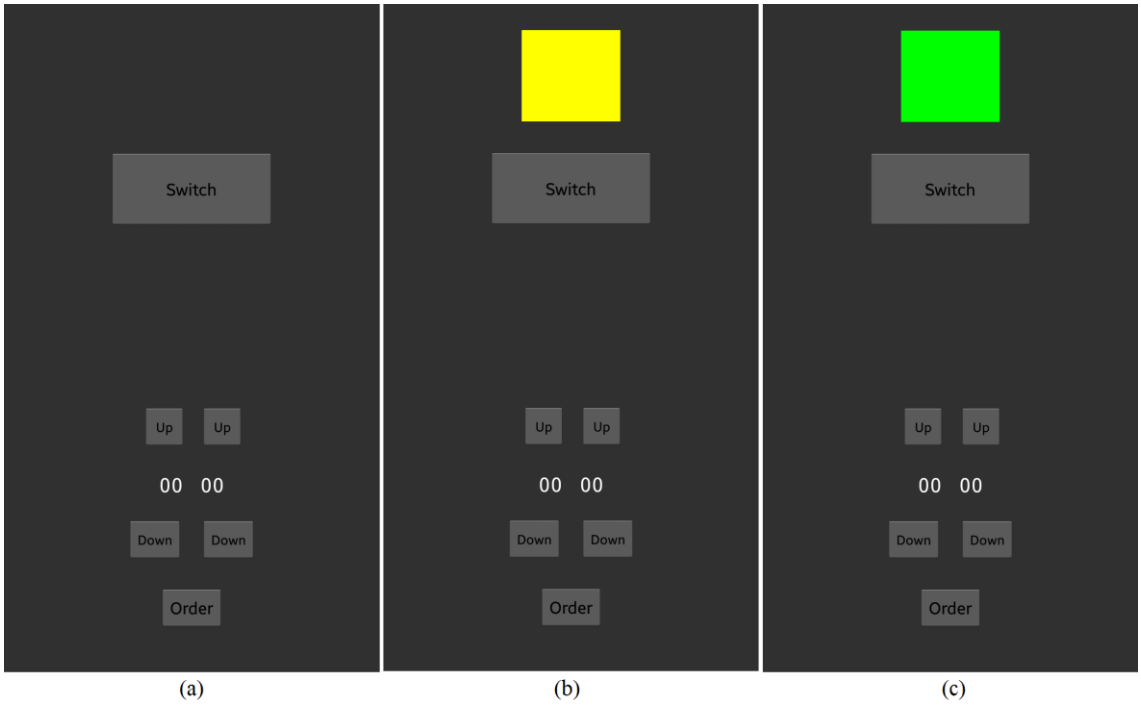
Nutitelefoni rakenduses on kaks vaadet: laadimisvaade ja käsuvaade.

Laadimisvaade kuvatakse kasutajale kohe programmi käivitades (Joonis 11). Selle vaate ajal üritab rakendus saada ühendust mikrokontrolleriga, saates sellele HTTP päringu eelsoojenduse oleku tuvastamiseks. Kui rakendus on saanud päringule vastuse, suunab rakendus automaatselt kasutaja edasi käsuvaatele. Laadimisvaade on oluline, kuna mikrokontrolleriga ühenduse võtmine võib võtta aega. Vastasel juhul, kui oleks ainult käsu vaade, siis sellel ajal, kui rakendus üritab mikrokontrolleriga ühendust võtta, tuleks käsunupud ära keelata. See ei ole hea lahendus, kuna nuppude nägemisel võib kasutajal tekkida tahe neid vajutama hakata.



Joonis 11. Androidi rakenduse laadimisvaade.

Nagu enne sai mainitud, on teiseks vaateks käsuvaade (Joonis 12). Sellel vaatel saab kasutaja näha eelsoojenduse olekut, teha nupust kohene ümberlülitus ning ette tellida erakorralist soojendust soovitud kellaajaks 24 tunni piires.



Joonis 12. Androidi rakenduse käsuvaade: (a) eelsoojendus ei tööta, (b) päring on saadetud, aga vastust pole veel saabunud, (c) eelsoojendus töötab.

Ümberlülitamise nupu kohal on kast, millel võib olla kolm värvi: läbipaistev, kollane ja roheline. Läbipaistev tähendab, et eelsoojendus ei tööta. Roheline on märk sellest, et

eelsoojendus töötab. Kollane tähendab, et päring on saadetud välja, aga vastust pole veel tulnud. Kollane kast on oluline, kuna mikrokontrollerilt pärimine võib võtta aega. Lisaks püsiv kollane kast võib olla märk sellest, et lülitusseadmel on esinenud viga ja mikrokontroller tuleb lähtestada. Kui kast on läbipaistev või roheline, võib kindel olla, et päringule reageeriti edukalt.

Allpool eelsoojenduse oleku kasti asub lülitusnupp (Joonis 12). Antud nupul on samasuguse funktsionaalsus nagu on maketeerimislaual asetseval nupul, mis kutsub esile ümberlülituse. Pärast virtuaalse nupu vajutust uuendatakse eelsoojenduse oleku kasti värvi.

Kõige all asetseb kellaaja valimine ning tellimusnupp, millega on võimalik tellida eelsoojendust ette 24 tunni raames. Vajutades nuppu *Order*, kuvatakse kasutajale märguanne musta hõljuva pilve sees, mida Androidi arenduskeskkonnas kutsutakse *toast*iks. *Toast*id on mõeldud Androidi programmides kasutajale teabe kuvamiseks nii, et programmi kasutamist ei peaks jätma pooleli. [15].

4.2 Võrgupäringud

Selleks, et rakendus saaks suhelda mikrokontrolleriga ning anda sellele käske, on tarvis programmiga teha HTTP päringuid. Projektis kastutavad GET päringud on kirjeldatud peatükis 3.2 lk 21.

HTTP päringute tegemiseks kasutan klassi `java.net.HttpURLConnection` [16]. Antud objekti saab kätte kutsudes välja funktsiooni `openConnection()` objektil, mis on `java.net.URL` klassist. Päringu vastuse saab kätte kasutades klassi `java.io.InputStream` (Joonis 13).

```
URL url = new URL("http://www.android.com/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
try {
    InputStream in = new BufferedInputStream(urlConnection.getInputStream());
    readStream(in);
} finally {
    urlConnection.disconnect();
}
```

Joonis 13. HTTP päringu tegemine.

HTTP päringud on asünkroonsed, mis tähendab, et neid jooksutatakse eraldi lõimel. Kuna antud projektis võib juhtuda, et esimesele päringule ei pruugi alati vastust tulla, siis päring tuleb esitada uuesti. Tekib probleem, kus üks kõrvaline lõim kutsub välja järgmise lõime, mille tõttu kaob kontroll päringute üle. Selle tagajärjel võib rakenduse kasutajaliides töötada valesti.

Eelmainitud probleemi lahendamiseks on loodud klass `android.os.Handler`, mis võimaldab vältida segadust lõimude tööde vahel pannes käima kõik lõimed kasutajaliidese lõimest ehk põhilõimest [17]. `android.os.Handler` hoiab endas sünkroniseeritud nimekirja erinevate prioriteetidega asünkroonsetest käskudest, mida tuleb täita. Käske saab `Handler`ile anda, saates selle sõnumi mistahes lõimest. Eelmainitud objekt kontrollib pidevalt oma sünkroniseeritud listi ning käskude olemasolul täidab neid vastavalt järjekorrale. Igal vaatele on mõistlik teha oma `Handler` objekt.

4.3 Tulemused

Androidi rakenduse välja töötamine oli edukas. Nii laadimis- kui ka käsuvaatel HTTP päringut töötavad. Kummalgi vaatel on oma `Handler` objekt, mis omab kontrolli lõimude töö ning vaadete uuendamise üle.

Rakendust testiti kolmel Androidi seadmel: Samsung Galaxy S4, Samsung Galaxy S5 mini ja Samsung Galaxy A5. Kõikidel eelmainitud seadmetel programm töötas ilma iseärasusteta. Kõik telefonid tulid toime kasutajale info edastamisega ning eelsoojenduse ümberlülitamisega. Samuti töötas erakorralise eelsoojenduse ettetellimine.

5 Kokkuvõte

Antud lõputöö **eesmärgid** olid:

1. töötada välja riistvaraline lahendus projekti jaoks;
2. töötada välja mikrokontrolleri tarkvara;
3. töötada välja nutitelefoni rakendus.

Eeltoodud eesmärgid said täidetud ning vastavad ka sissejuhatuses mainitud nõuetele.

Töö **tulemused** on:

1. riistvaralise lahenduse välja töötamine;
2. mikrokontrolleri tarkvara projekteerimine ja loomine;
3. nutitelefoni rakenduse välja töötamine.

Projekti tulemusel valminud seade sobib kasutajale juhul, kui tal on olemas vastav tarkvara ning programmeerimisoskused. Selleks, et seadet saaks kasutada inimene, kellel ei ole teadmisi programmeerimises, tuleb seade edasi arendada selliseks, et ajagraafik ja muud seaded oleksid muudetavad nutiseadmest.

Parimaks lahenduseks peab töö autor mikrokontrolleri tarkvara. Arduino programmi luues kasutas autor ülikoolist õpitud teadmisi kõige rohkem ning objektorienteeritud programmi haldamine ning parandamine klassideks jaotamise tõttu osutus arvatust lihtsamaks.

Kuna lülitusseade on ühendatud Internetiga, on selle edasiarendamise võimalused piiramatud. Idee, kuhu antud seadme edasi arendamisega saab pürgida, on Google'i kalendriga sünkroniseerimine. Seade saaks arvutada vastavalt kalendrisündmuse ajale ja asukohale, võttes arvesse, kui kaua asukohta sõitmine aega võtab, eelsoojenduse aja. Sellisel viisil puuduks kasutajal vajadus luua seadmele eraldi ajagraafik ning graafik püsiks automaatselt uuendatuna, mis nõuaks kasutaja poolt minimaalset sekkumist seadme töösse.

Samuti sobib seade üheks osaks koduautomaatika lahendustes. Targa kodu lahendused pakuvad võimalusi nagu elektriseadmete kaugjuhtimine, elektrikulude jälgimine ja kütte kontrollimine. Auto eelsoojenduse lülitamine sobib eelmainitud võimaluste sekka hästi, kuna see on osa elektriseadmete kaugjuhtimisest.

Käes olev projekt on autori jaoks olnud väga kasulik. Autor on õppinud, kuidas kasutada mikrokontrollerit, et muuta füüsilisel tasemel erinevate seadmete olekuid ning hoida seadmeid omavahel ühenduses läbi õhu. Kõik kogemused, mille autor on antud projektiga omandanud, on avanud tema jaoks avarama maailma infotehnoloogiliste lahenduste osas, et muuta inimeste igapäevaelu paremaks.

Kasutatud kirjandus

[1] Defa WarmUp, Defa. [WWW]
http://www.defa.com/et/automotive/warmup/ (12.04.2016)

[2] Defa käsiraamat, Defa AS [WWW]
http://www.defa.com/file/741816b6aa9/490278-E46%20EE%20TH%202009.pdf
(12.04.2016)

[3] Raspberry Pi, Wikipedia [WWW]
https://et.wikipedia.org/wiki/Raspberry_Pi (22.04.2016)

[4] Raspberry Pi A+ moodul 256MB, Oomipood [WWW]
http://www.oomipood.ee/product/2447906/raspberry-pi-a-moodul-256mb&s=raspberry%20pi (22.04.2016)

[5] Raspbian, Raspbian [WWW]
https://www.raspbian.org/ (22.04.2016)

[6] Arduino, Wikipedia [WWW]
https://et.wikipedia.org/wiki/Arduino (22.04.2016)

[7] Shields, Arduino [WWW]
https://www.arduino.cc/en/Main/arduinoShields (22.04.2016)

[8] CC3000 WiFi Shield Expansion Board with SD Slot for Arduino UNO R3 MEGA 2560 R3, Ebay [WWW]
http://www.ebay.com/itm/CC3000-WiFi-Shield-Expansion-Board-with-SD-Slot-for-Arduino-UNO-R3-MEGA-2560-R3-/131676481614?hash=item1ea887a84e:g:zYYAAOSwkZhWStQU (22.04.2016)

[9] Adafruit CC3000 Library, GitHub [WWW]
https://github.com/adafruit/Adafruit_CC3000_Library (22.04.2016)

[10] Relee moodul 1 kanaliga, IseTegija [WWW]
http://isetegija.ee/toode/relee-moodul-1-kanaliga/ (22.04.2016)

[11] Maketeerimislaud 64*171mm 840 punkti, Oomipood [WWW]
http://www.oomipood.ee/product/bx-4112n/maketeerimislaud-64-171mm-840-punkti&s=maketeerimislaud (23.04.2016)

[12] Adafruit_CC3000_Library / examples / InternetTime / InternetTime.ino, GitHub [WWW]
https://github.com/adafruit/Adafruit_CC3000_Library/blob/master/examples/InternetTime/InternetTime.ino (07.05.2016)

[13] Arduino Time Library, Arduino [WWW]
http://playground.arduino.cc/Code/Time (07.05.2016)

[14] Android Studio, Android [WWW]
http://developer.android.com/sdk/index.html (08.05.2016)

[15] Toasts, Android Developers [WWW]
http://developer.android.com/guide/topics/ui/notifiers/toasts.html (08.05.2016)

[16] HttpURLConnection, Android Developers [WWW]
http://developer.android.com/reference/java/net/HttpURLConnection.html (09.05.2016)

[17] Handler, Android [WWW]
http://developer.android.com/reference/android/os/Handler.html (09.05.2016)