TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Krista Kippar

JUHTMEVABA HINDAMISSEADME PROTOTÜÜP

bakalaureusetöö

Juhendaja: Peeter Ellervee
PhD

Tallinn 2020

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: Krista Kippar

05.08.2020

Annotatsioon

Huvi tagasiside vastu on suurenenud ning selle põhjusel tehakse järjest enam erinevaid värkvõrgu lahendusi andmete kogumiseks. Käesolevas töös on loodud värkvõrgu seadme prototüüp, mis on juhtmevaba ja võimaldab anda tagasiside nuppude näol. Prototüüp põhineb Adafruit HUZZAH32 *Feather* arendusplaadil, millel on Espressifi integreeritud Wi-Fi mooduliga mikrokontrolleril ESP32. Arendusplaadile sisendite andmiseks ja RGB LED-i juhtimiseks on loodud lihtsakoelise elektroonikaskeemiga riistvara makettplaadile. Riistvarale vastav loodud tarkvara arendamiseks kasutatakse ESP-IDF raamistiku. Kasutajalt saadud andmed laetakse *Amazon Web Service IoT Core* serverisse. Prototüübi ja serveri vaheliseks suhtluseks on protokolliks valitud laialtkasutatav MQTT. Lõpptulemusena valmis prototüüp, millel baas funktsionaalsused töötavad. Võttes arvesse töö käigus selgunud tüüpvead ja lisa võimalused on selle informatsiooni põhjal prototüübi edasi arendamiseks seadmeks piisav.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 25 leheküljel, 7 peatükki, 18 joonist, 3 tabelit.

Abstract

Wireless Grading Device Prototype

Most of the daily problems, for example collecting feedback from customers, can be solved with Internet of Things devices. Wireless grading device prototype is an example, how to create an IoT device from the ground up. The process started with choosing a proper hardware and a software development environment. As a result Adafruit HUZZAH32 Feather development board, which is based on microcontroller ESP32 with integrated Wi-Fi module, was chosen as a base for this prototype hardware and ESP-IDF as a development environment. Software source code is written in C++ and compiled using ESP-IDF provided script.

This device includes three buttons and RGB LED light to give feedback, which information is sent to Amazon Web Service IoT core server by using MQTT protocol. X.509 certificates are applied for secure communication between the device and the server. In addition, secure boot and encrypted flash can be used to secure the data on the flash and prevent program code being overwritten.

As a result, basic software and hardware requirements were satisfied. However, power consumption optimization was not included at the beginning of the process. Nevertheless, it can be taken into account, if the prototype is developed into an actual device in the future. Likewise, this prototype can be used for further development, such as designing proprietary PCB – printed circuit board and add device provisioning for manufacturing.

The thesis is in Estonian and contains 25 pages of text, 7 chapters, 18 figures, 3 tables.

Lühendite ja mõistete sõnastik

USB	*Universal Serial Bus*, universaalne järjestikliin
CLI	*Command line interface*, käsurealiides
IoT	*Internet of Things*, asjade internet , nutistu, värkvõrk
IDE	*Integrated Development Environment*, integreeritud arenduskeskkond
ESP-IDF	*Espressif IoT Development Framework*, Espressifi nutistu arendusraamistik
Wi-Fi	*Wireless Fidelity*, raadiokohtvõrk, traadita internet
OS	*Operating system*, operatsioonisüsteem
FreeRTOS	*Free Real-Time Operating System*, reaalajaline operatsioonisüsteem mikrokontrolleritele ja mikroprotsessoritele
AP	*Access point*, pääsupunkti
SSID	*Service Set Identifier*, teenusekogumi identifikaator ehk teisisõnu võrgu nimetus
AWS	*Amazon Web Service*, Amazoni poolt pakutav pilveandmetöötlus platvorm
MCU	*Microcontroller unit*, mikrokontroller
PWM	*Pulse-width modulation*, pulsilaiusmodulatiooni
MQTT	*Message Queuing Telemetry Transport*, värkvõrgu ja masinate vahelise suhtluse transpordiprotokoll
Alglaadimine	*bootloader*
Draiver	*Driver*, Välisseadme juhtimisprogramm
Säilmälu	*Non-volatile storage*, muutmälu, milles andmes säilivad ka pärast toite väljalülitamist
eFUSE	*Electrical Fuse*, elektriline sulavkaitse

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Järjest enam igapäeva ellu on tulemas erinevad värkvõrgu lahendused. Samuti on üha tähtsamaks muutunud inimeste arvamus ja tagasiside kasutavate teenuste suhtes. Ühendades mõlemad nii IoT tehnoloogia kui ka probleemi on leitavad erinevad lahendused. Tihti peale on sellised ülesanded lahendatud suurte puutetundlikute ekraanide ja otse vooluvõrku ühendatud statsionaarsete seadmetega, mida võib tihti kohata kauplustes. Lähenedes probleemile teise nurga alt, siis sellised seadmed võiksid olla rohkem minimalistlikumad ja kompaktsemad, et neid oleks võimalik kergesti paigaldada ja nende asukohta vajadusel muuta.

Käesoleva töö eesmärgiks on luua juhtmevaba hindamisseadme prototüüp, millega on võimalik registreerida 3 erineva nupu vajutust ja edastada info üle juhtmevaba võrgu serverisse. Selleks koostatakse seadmele nõuded, valitakse välja esialgne riistvara ning koostatakse algne elektroonikaskeem süsteemi tarkvara arendamiseks ja testimiseks. Edasi tehakse valmis füüsiline riistvara prototüüp, millele hakatakse looma tarkvara. Seadme riistvarast tulenevalt valitakse välja sobivad arendustööriistad tarkvara loomiseks ja võimalik kommunikatsiooni viis serveriga, kuhu hakatakse saatma andmeid. Samuti uuritakse võimalikke lahendusi, kuidas kindlustada turvaline suhtlus serveriga ja kaitsta seadmel olevat programmikoodi ning jooksvaid andmeid.

Eelnevalt kirjeldatud tööprotsess peaks andma ülevaate üldise sardsüsteemi loomise protsessist, sellest tulenevatest probleemidest ja lahendustest. Peale esimest töötavat lahendust analüüsitakse, kui edukas oli süsteemi planeerimine, kui töökindel on valminud prototüüp ning mida saaks paremaks teha ja/või tuleks kindlasti muuta. Töö tulemusena peaks valmima prototüüp, mille edasiarendusena oleks võimalik seda kasutada juhtmevaba hindamisseadmena.

2 Süsteemi planeerimine

Juhtmevaba hindamisseade peab kasutaja jaoks koosnema vähemalt 3 nupust, millega anda tagasisidet ning RGB LED tulukesest, et anda kasutajale infot seadmel toimiva kohta. Selleks, et kasutajalt tulnud sisendist teada saada ilma seadet füüsiliselt puutumata, tuleks andmeid edastada üle juhtmevaba ühenduse. Võttes tarvidusele Wi-Fi ühenduse pole vajalik lisaseadme ehk vastuvõtja olemasolu, nagu paljud teised traadita ühenduste lahendused, näiteks *Bluetooth* ehk sinihammas ja *ZigBee* vajavad, sest et see on niivõrd laialdaselt kasutatav. Samuti traadita internet võimaldab üleslaetavatele andmetele asukohast sõltumata ligi pääseda.

Süsteemi loomisel alustatakse vastava seadme riistvara nõuete ja põhikomponentide valimisest eelnevate projekti ainetes koostatud süsteemide põhjal. Põhjuseks on see, et valitavast mikrokontrollerist oleneb sellele prototüübile arendatav tarkvara. Seejärel pannakse paika tarkvara nõuded ning selle loomiseks valitakse välja sobilik arenduskeskkond.

2.1 Riistvara komponentide valik ja nende nõuded

Süsteemi loomiseks ja testimiseks on vaja valida põhikomponendid ja nende põhjal koostada nõuded. Põhikomponentideks on mikrokontroller süsteemi juhtimiseks, 3 nuppu kasutajalt sisendite saamiseks, 1 nupp seadme lähtestamiseks ja RGB LED tuluke kasutajale tagasisideme andmiseks.

Mikrokontrolleril peab olema Wi-Fi moodul sisse ehitatud või võimalusel omama viike, millel on võimalik luua suhtlus välismooduliga. Samuti peaks olema vähemalt 8 kasutatavat viiku, millest 3 viiku on PWM ehk pulsilaiusmodulatiooni väljundi võimekusega ja 4 viiku digitaal- ja/või analoogsisendi võimekusega

Prototüübi plaat peab võimaldama välimise toiteallika, näiteks akupanga, ühendust. Samuti peab sisaldama 12 väljaviiku: 4 nupu ja ühe RGB LED-i ühenduseks ning pesad kontrolleri paigaldamiseks.

Õppeaine Arvutid ja Süsteemi projekti käigus loodud esialgse idee prototüüp oli loodud Arduino Nano kontrollerile kasutades ESP8266 mikrokiipi, mis on eraldi seisev Wi-Fi moodul[15]. Peale seda projekti uuriti rohkem erinevaid Wi-Fi mooduleid ning leiti eelnevate kontrollerite asendusena ESP32, millel on integreeritud Wi-Fi moodul ning mille on 2 eraldi MCU-l(Xtensa 32-bit LX6) seisvat tuuma[2]. Sellise kontrolleriga leiti firma Adafruiti poolt toodetud Feather HUZZAH32 ESP32-WROOM arendusplaat, millel on toodud välja vajalikud väljaviigud ja mikro USB ühendus kontrolleri programmeerimiseks ja suhtlemiseks üle UART-i.

Tabel 1. ESP8266 ja ESP32 Wi-Fi moodulite spetsifikatsioonide võrdlus[1],[2].

Ühik	**ESP8622**	**ESP32**
Wi-Fi kanali töörežiim	HT20	HT20/HT40
Kanali laiused	20MHz	20/40MHz
Kõrgeim andmevoo kiirus	72.2 Mbps	150 Mbps
Virtuaalset Wi-Fi liidest	2	4

Integreeritud mooduliga kontrolleri kasutusele võtmine vähendab inimlikku vigade teket, näiteks viikude valesti ühendamine omavahel või mürarikas ühendusliin, mille põhjustajaks on halb joode makettplaadil. Samuti kahe eraldi protsessori tuuma olemasolu võimaldab rööprähklemist, mille abil saab luua sõltumatuid protsesside töötlust tarkvaraliselt. ESP32-le annab ka eelise 40MHz kanali laiuse kasutamine, mis võimaldab andmeid saata suurema kiirusega.

2.2 Tarkvara nõuded

Tarkvara peamiseks eesmärgiks on nupu vajutuse registreerimine ehk 3 sisendi lugemine. Samuti peab tagama nupu vajutuse täpsuse ning sisendi aktiveerimisel tuleb see registreerida koos aja markeeringuga. Informatsioon vajutuse kohta tuleb saata serverisse. Kinnitamaks, et nupp on vajutatud ja registreeritud, tuleb kasutajale sellest märku anda LED tulega. Andmaks kasutajale, ilma et seade oleks ühenduses arvutiga, teada Wi-Fi võrku ühendumise olekust, tuleks samuti kasutada LED tuld. Lisaks peab olema draiver RGB LED-i kontrollimiseks. RGB LED-i värvuse indikatsiooni värvid ja nende kirjeldus on välja toodud Tabel 2.

Tabel 2. Indikatsiooni LED tule kirjeldus ja tähendused.

Värvus	**Toiming**
Roheline	Nupp 1 vajutusel
Sinine	Nupp 2 vajutusel
Punane	Nupp 3 vajutusel
Vilkuv hele roheline	Ühendumine Wi-Fi võrku esimesel sisse lülitamisel
Konstantne kollane	Wi-Fi võrgu parameetrid: SSID ja salasõna on seadmele sisestamata

Seadme paindlikumaks kasutamiseks tuleks Wi-Fi võrgu seadistamiseks luua käsurealiides ehk CLI, millega oleks võimalik sisestada vajalikud parameetrid. Vastasel juhul peaks Wi-Fi võrgu sätted kirjutama otse programmikoodi, mis juhul seadet oleks võimalik ühendada ainult ühte teatud võrku. Samuti turvalisuse huvides tuleks liidesele lisada salasõna sisestamine vältimaks kolmanda osapoole ligipääsu seadmele. Lisaks peaks kasutaja saama valida kasutatavate käskude kuvamisel nende kirjelduste keelt nii eesti kui ka inglise keele vahel. Vajalikud käsud koos parameetrite ja seletustega on väljatoodud Tabel 3.

Tabel 3. Käsurealiidese CLI käsud koos parameetrite ja seletustega.

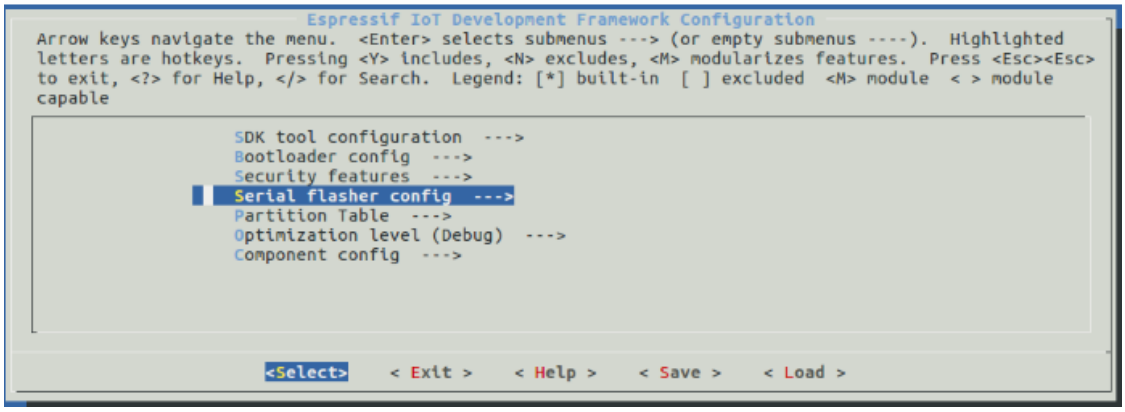
Käsk	**Parameetrid**	**Seletus**
help	puudub	Kuva kõik kättesaadavad käsud koos parameetrite seletustega
wifi-add	SSID SALASÕNA	SSID Wi-Fi võrgu nimi PASSWORD Wi-Fi võrgu salasõna
	--delete	Kustuta seadistatud Wi-Fi võrgu parameetrid
wifi-scan	puudub	Kuva terminali seadmele leitavad Wi-Fi võrgud
wifi-info	puudub	Kuva terminali ühendatud võrgu SSID

Käsk	Parameetrid	Seletus
pw	-l [SALASÕNA] -n [UUS SALASÕNA]	CLI liidese salasõna muutmine. -l [SALASÕNA] – viimati sisestatud salasõna -n [UUS SALASÕNA] – uus salasõna
	--logout	CLI liidesest välja logimine
lng	[keel]	Muudab käsustiku keelt 0 – inglise keel 1 – eesti keel (NB! Muutuse sisseviimiseks on vajalik seadme lähtestamine käsurealt või füüsiliselt)
restart	puudub	Seadme tarkvaraline lähtestamine

2.3 Programmeerimiskeele ja arenduskeskkonna valik

Praegusele süsteemile eelnev Arvutisüsteemide aine projekt, mis põhines samuti ESP32-l kasutati programmeerimiskeelt C ja programmikoodi kompileerimiseks ning *flashimiseks*/kõrvetamiseks kasutati Arduino IDE-t[16]. Et muuta süsteemi lihtsamalt hallatavaks ja arusaadavamaks, mindi üle objektorienteeritud keelele C++, mis võimaldab kirjutada komponentide teeke abstraktsemalt ja üksteisest vähem sõltuvamaks.

Programmikoodi kompileerimiseks ja kõrvetamiseks võeti Arduino IDE asemel kasutusele tootja poolt loodud raamistik nimega ESP-IDF, mida saab jooksutada erinevate operatsioonisüsteemide terminalides. Kasutatav arendusraamistik võimaldab käsurealt programmikoodi kompileerida, kõrvetamist kontrollerile, välkmälu ehk *flash*i kustutamist/tühjendamist ning lisaks ka süsteemi konfigureerimist kasutades *menuconfig*i (Joonis 1). Seadme konfigureerimine programmikoodis kaudsel teel võimaldab programmikoodi modifitseerida vastavalt riistvarale. Saab sätestada erinevad reegleid ning määrata kontrolleri viike ilma, et peaks otseselt muutma programmikoodi.



Joonis 1. Kuvatõmmis ESP-IDF konfigureerimismenüüst *menuconfig[5]*

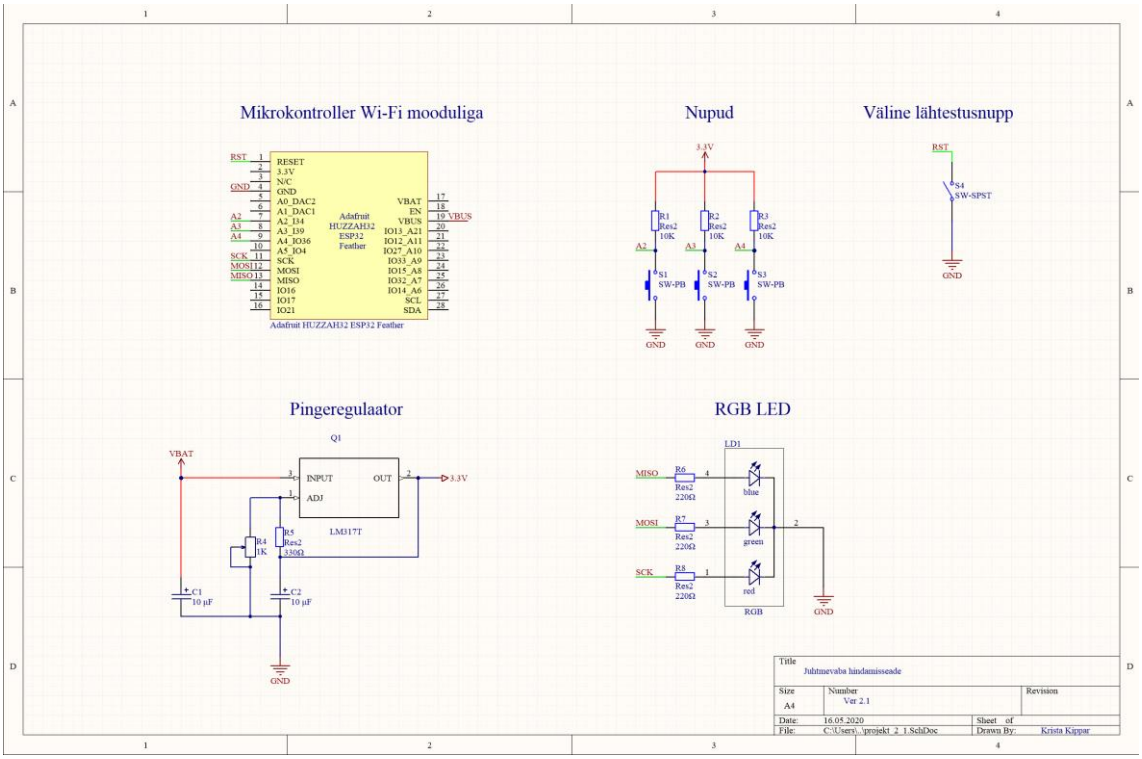
Projekti kuuluvate põhiprogrammi ja sellele teekide failide loomiseks ning haldamiseks kasutati platvormi CLion. See IDE genereerib CMake failide ja programmikoodis defineeritu teekide põhjal kiirpääsud kasutatavatele funktsioonidele, muutujatele ja teistele programmikoodi osadele, mis muudab koodi kirjutamise efektiivsemaks. Samuti annab märku võimalikest vigadest programmikoodis ilma seda kompileerimata[14].

3 Riistvara koostamine ja realiseerimine

Süsteemile luuakse lihtsakoeline elektroonikaskeem, et saaks testida seda seadmena. Füüsiliseks testimiseks luuakse algne trükkplaadi prototüüp makettplaadile, millele on joodetud vajalikud pesad kontrolleri jaoks ning nuppude ja RGB LED-i ühendamiseks väljaviigud.

3.1 Elektroonikaskeemi koostamine

Vastavalt peatükis 2.1 väljatoodud nõuetele ja põhikomponentidele loodi Joonis 2 elektroonikaskeem, mida oleks võimalik lihtsalt realiseerida makettplaadile ilma, et oleks vaja eraldi koostada trükkplaat.



Joonis 2. Seadmele koostatud elektroonikaskeem

Kuna kontrolleri loogika töötab 3.3 voldil[2] siis sellepärast konverteeritakse pingeregulaatori LM317T abil viigul nimega VBAT, mis on akupanga mikro USB ühenduselt tulev voolupinge 5 volti, vastavaks väljundiks, vältimaks süsteemi ülepingestamist ning läbi põlemist. Lisaks on väline lähtestusnupp, mille jaoks poleks vaja seadeldise lähtestamiseks seda korpusest välja võtta. Sellele polnud vaja lisa

takistit, kuna kontrolleril on see juba olemas[8]. Samuti sisendviikudel olevate nuppude sisend ei „ujuks“ on lisatud *pull-up* ehk ülestõmbetakistid, mille väärtuseks on võetud laialt levinud 10 kΩ, et tagada võimalikult väike voolutarve, mis on umbes

$$I = \frac{U}{R} = \frac{3.3V}{10k\Omega} = 330\mu A$$

1 nupu kohta, kui nupp on vajutatud.

Kasutajale süsteemi hetke olekust tagasiside andmiseks on lisatud ühise katoodiga RGB LED tuluke, mida saab kontrolleri väljundi abil juhtida. LED-i katoodi jalgadele on lisatud 220Ω takistid. Arvutamise järgi peaks olema takisti 68 Ω, aga kuna katsetamisel LED oli autori arvates liiga kirgas siis valiti katse-eksitus meetodil lõpuks 220 Ω-d takistid.

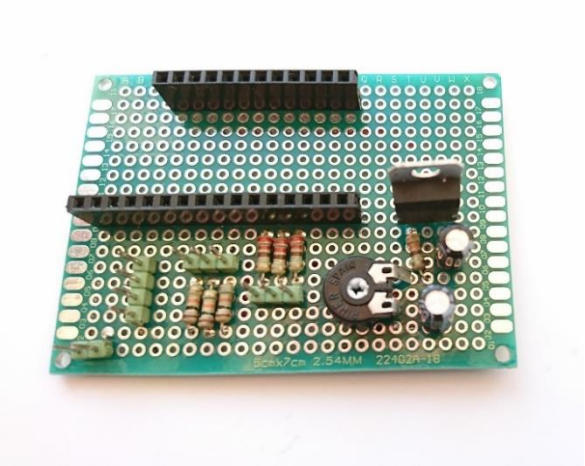
3.2 Trükkplaadi prototüübi valmistamine makettplaadile

Peatükis 3.1 koostatud elektroonikaskeemil joodeti makettplaadile esimene versioon, kus toodi eelnevalt mainitud komponentide ühendamiseks plaadiga vastavad väljaviigud nuppude, RGB LED-i ühendamiseks ning Adafruifi Feather ESP32 plaadi jaoks pesad. Ühendamiseks välised komponendid makettplaadile loodud riistvara prototüübiga, joodeti neile külge pistikuotstega juhtmed(Joonis 3).

Joonis 3.Pilt väliskomponendid koos joodetud juhtmetega.

Välja toodavate viikude asetus sõltub nuppude ja LED-i kontrollerite viikudega, selle tõttu, et enamus kasutavatest viikudest asub plaadi vasakul pool(Joonis 4). Adafruidi Feather plaadi jaoks asetatud pistikud paikneva nii, et ülaossa jäävasse mikro USB pesa

asetseks püsti, et vältida akupanga kaablilt tulevat pistiku paigalt nihkumist gravitatsioonijõu mõjul. Makettplaadi joode on näha Lisas 1.



Joonis 4. Pilt makettplaadile joodetud prototüübist.

4 Tarkvara

Töö käigus luuakse juhtmevaba hinnangu seadmele vastav tarkvara, mis peaks rahuldama peatükis 2.2 väljatoodud nõudeid. Parema kujutluspildi loomiseks, milline see peaks välja nägema, luuakse sellele vastavalt nõuetele arhitektuur, mille põhjal kirjutatakse valmis programmikood.

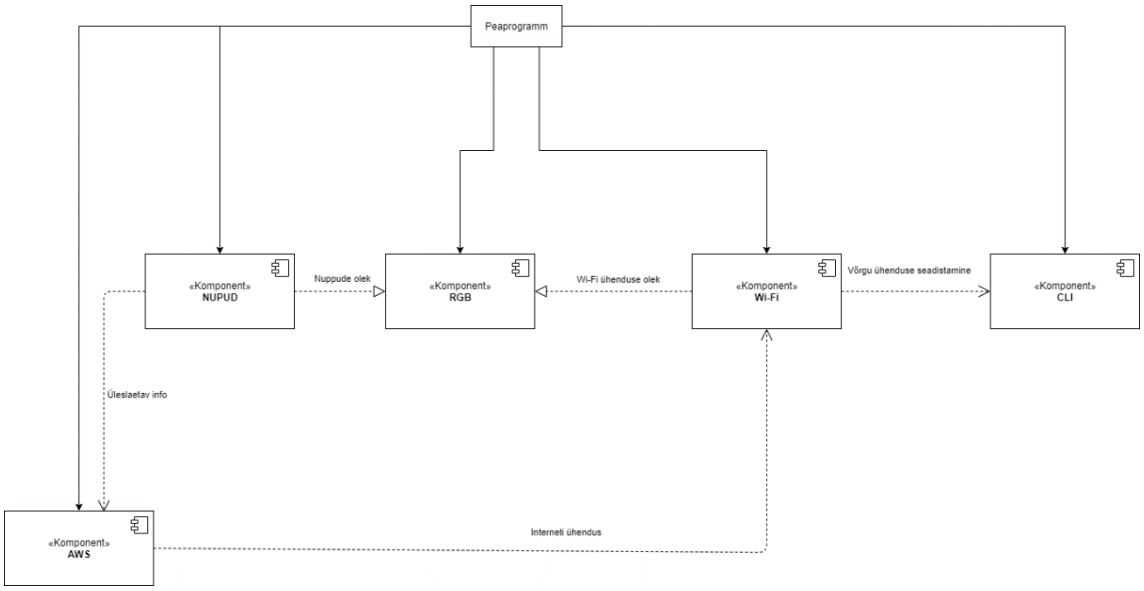
4.1 Arhitektuur

Tarkvara koosneb 5 peakomponendist ja põhiprogrammist. Igal komponendil on teatud riistvaralise või tarkvaralise komponendi oleku lugemine või juhtimine. Põhiprogrammis on kirjeldatud üldine juhtimisloogika ja initsialiseeritakse komponendid vastavalt sõltuvusest teiste komponentidega. Komponentideks on CLI, RGB, Wi-Fi, NUPUD, AWS.

Komponendis :

- CLI luuakse baas funktsioonid, mis võimaldab registreerida erinevaid käske teiste moodulite jaoks.
- RGB asub LED-i juhtimiseks vajalik riistvaraprogramm, mida teised moodulid saavad kasutada
- Nupud võimaldatakse registreerida nupu vajutus ja loob sellele vastava *interrupt*i ehk katkestuse, mil hetkel salvestatakse vastava nupu olek ja vajutuse aeg, mis antakse edasi AWS komponendile ja kasutajale inditseeritakse RGB LED värvi näol.
- AWS asub vajalikud atribuudid serveriga suhtluseks ja vajalikud sertifikaadid ning privaatvõti.
- Wi-Fi luuakse ühendus CLI-s seadistatud võrguga.

Lootava tarkvara üldarhitektuur ja komponentide omavahelised seosed on kirjeldatud Joonis 5.

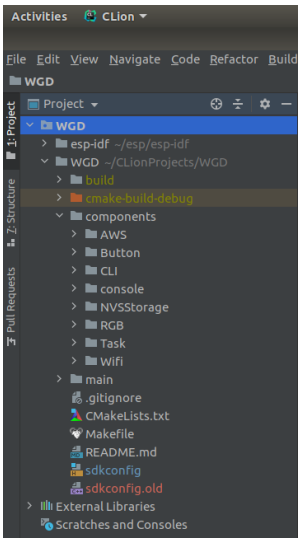


Joonis 5. Tarkvara üldine arhitektuur

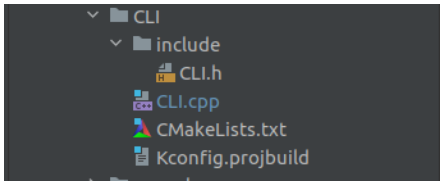
4.2 Programmikood

Programmikood põhineb suuresti ESP-IDF projekti koodibaasil ja Neil Kolbani näitekoodidest ning mõlemad on saadaval GitHubi repositooriumist[7],[6]. Käesoleva töö käigus koostatud programmikood asub GitLab repositooriumis[21]. Projekt sisaldab põhiprogrammi ja enda loodud teekidega komponente, mis asuvad eraldi kaustas(Joonis 6). Komponentide kaustad peavad asuma kaustas *components*, et neid kompileeritaks koos ESP-IDF komponentidega, mis annab ka vajadusel ümber kirjutada raamistiku komponendid nagu näiteks käesolevas projektis komponent *console.*

Komponendi kaust (näide Joonis 7) koosneb *include* kaustast, kus asub teegi päis fail, teegi lähtefail, CMake loendite tekstifail *CMakelists* (näide Joonis 8) ning mõningate komponentidel (CLI, NUPPUD, RGB) Kconfig.projbuild fail (näide Joonis 9) *menuconfigi* jaoks.



Joonis 6. Kuvatõmmis projekti ülesehituses programmis CLion



Joonis 7. Komponendi CLI kausta ülesehitus

```
set(COMPONENT_SRCS "CLI.cpp")
set(COMPONENT_ADD_INCLUDEDIRS include)
set(COMPONENT_REQUIRES Task console NVSStorage)
register_component()
```

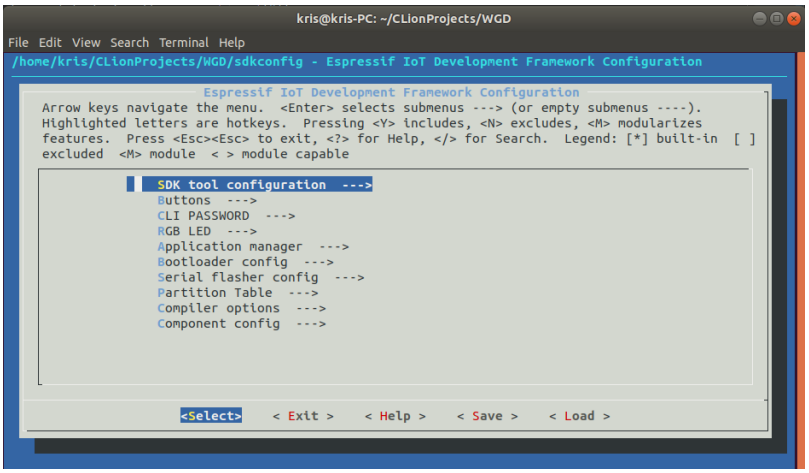
Joonis 8. Komponent CLI CMakelists.txt fail

```
menu "CLI PASSWORD"

    config PASSWORD
        string "Default password"
        default "1234abcd"
    help
        Default CLI password.

    config PASSWORD_LEN
        int "Maximum password length"
        default 8

endmenu
```

Joonis 9. Komponent CLI Kconfig.projbuild fail

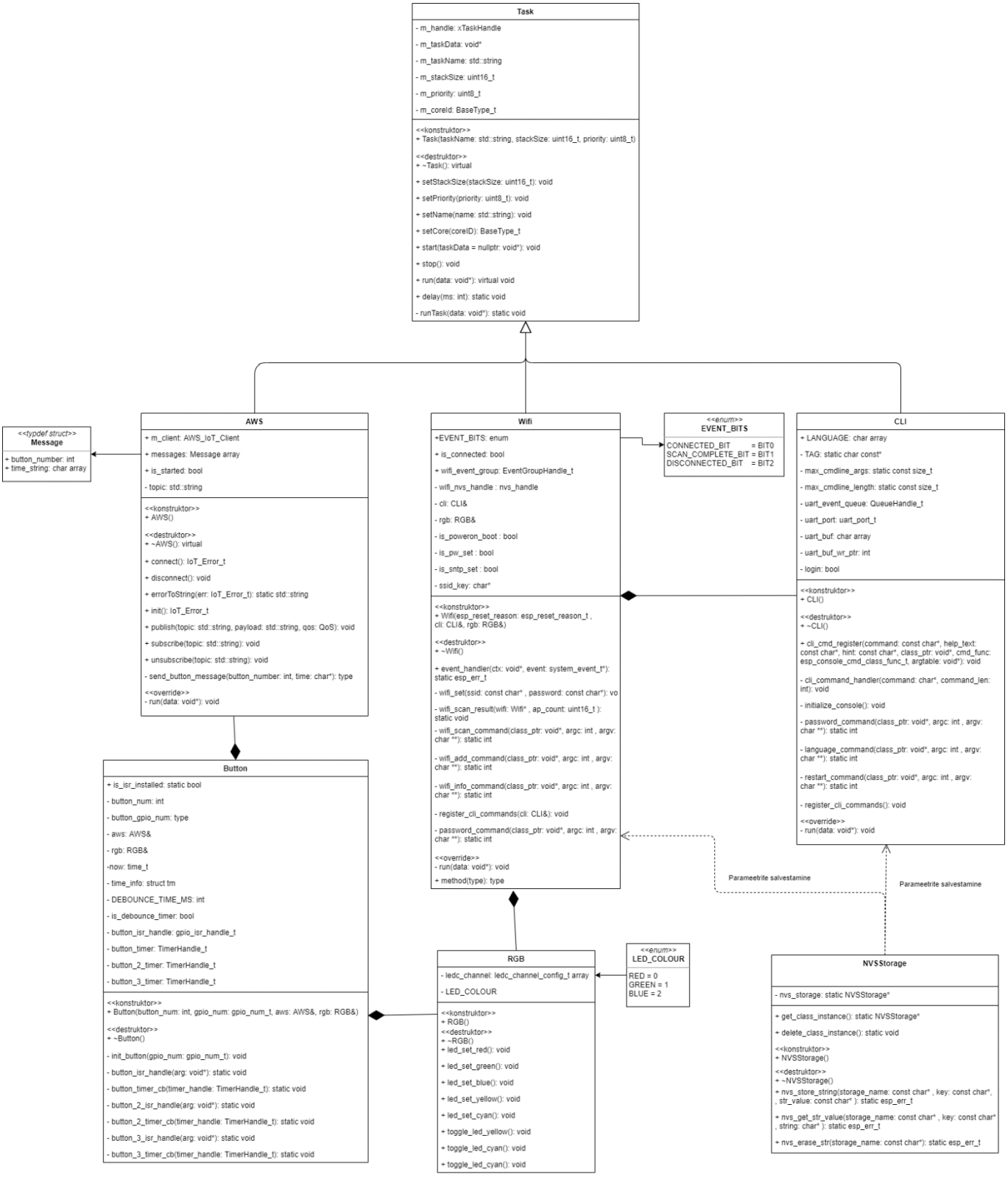
CMakelists faile kasutatakse komponentide haldamiseks ja registreerimiseks. CMake on vabavaraline tööriist, mis aitab hallata lähtekoodi ehitamist ning muudab arendajale projekti komponentide haldamiseks lihtsamaks[17]. Seda tööriista kasutab Espressifi loodud skript idf.py, kus CMake abil viidatakse preprotsessorile kasutatavad teegid ning mille lähtekoodist kompileeritakse masinkood, kui skripti parameetriks antakse sõna „*build*“.

Lihtsustamaks programmikoodis riistvaralisi muudatusi on täiendatud peatükis 2.3 mainitud konfiguratsiooni menüüd nimega *menuconfig*. Selle abil saab muuta vastavate nuppude ja RGB LED tulukese sisendviike(Vt Lisa 2 ja 4). Samuti saab seadistada algupärase salasõna, mille abil on võimalik logida sisse käsurea liidesele kasutades terminali, kui seade on ühendatud arvutiga üle mikro USB kaabli(Vt Lisa 3).

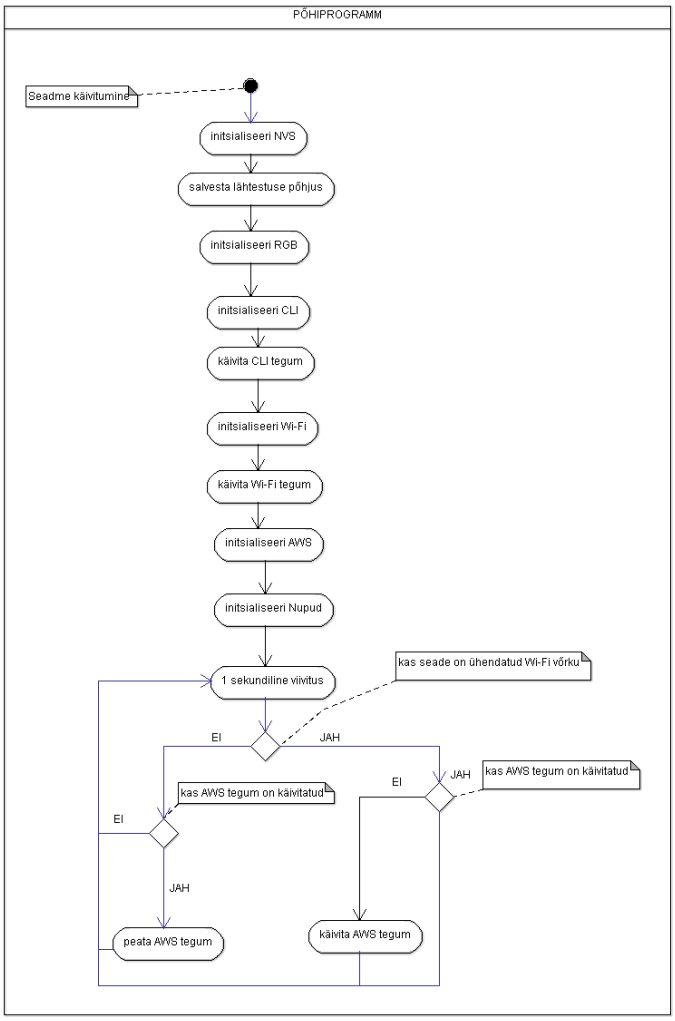


Joonis 10. Kuvatõmmis projektile kohandatud *menuconfigist*

Programmikoodis komponentide teegid on jagatud eraldi klassidesse, et põhiprogrammis saaks luua vastavad objektid. Klassi konstruktoris, funktsioon millega luuakse objekt, initsialiseeritakse vajalikud komponendid vastavate parameetritega ja Task alamklassidele AWS, Wifi ja CLI luuakse eraldi tegum, mis käivitatakse põhiprogrammis. Põhiprogramm on kirjeldatud UML-is Joonis 12 ja vastava funktsiooni programmikood on välja toodud Lisa 5. Erinevate klasside omavahelised sõltuvused on kirjeldatud klassidiagrammil, mis on kujutatud Joonis 11.



Joonis 11. Komponentide klassidiagramm



Joonis 12. UML põhiprogrammist programmikoodi põhjal

4.3 Muutused tarkvara arhitektuuris

Programmikoodi luues tulid välja mõningad puudused peatükis 4.1 loodud arhitektuuris. Tarkvara töökindlamaks muutmiseks on otsustatud komponentidest, AWS, Wi-Fi ja CLI, luua eraldi tegumid, mille töös hoidmise eest hoolitseb operatsioonisüsteem *FreeRTOS*. Selleks, et igas komponendis endas ei peaks looma tegumit, on lisatud juurde komponent Tegum. Lisaks on vajalik salvestada Wi-Fi ja käsurealiidese parameetrit, nii et seadme lähtestamisel need säiliksid. Selleks võeti kasutusele NVS mälu ning sealt lugemiseks ja kirjutamiseks loodi komponent NVS.

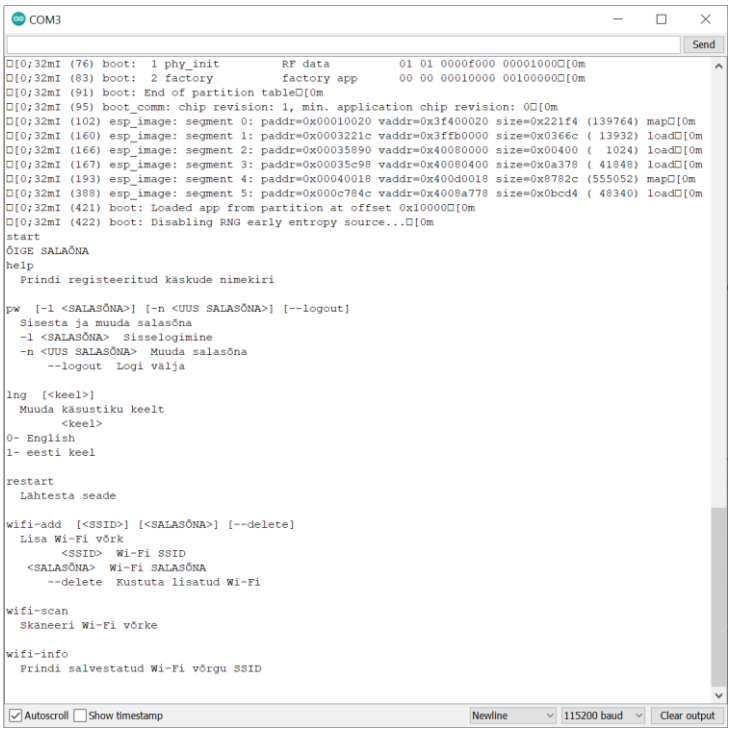
Tegumi komponent on objektorienteeritud programmeerimise koha pealt ülemklass, mis loob protsessoril jooksval operatsioonisüsteemil *FreeRTOS* tegumi iga alamklassi kohta. Programmikoodi tegumiteks jagamine lihtsustab põhiprogrammi koostamist ja jätab iga tegumi töötamise operatsioonisüsteemi kätte. *FreeRTOS* on mikrokontrolleritele ja väikestele mikroprotsessoritele loodud reaalaja operatsioonisüsteem, mis ei nõua palju mälumahtu samas pakkudes võimalikult palju arendusvõimalusi ja funktsionaalsusi [20].

NVS (*Non-Volatile Storage*) ehk säilmälu tuli kasutusele võtta, et salvestada seadmesse käsurealiidese salasõna ja Wi-Fi võrgusätted ka pärast seda, kui seade on välja lülitatud. See loodi eraldi komponendina, kuna nii komponendid CLI kui ka Wi-Fi kasutavad samu operatsiooni säilmälust lugemist ja kirjutamist. Nii on vähetatud programmikoodi duplitseerimist ja võimalike vigade vältimist.

4.4 Komponendi valideerimine riistvaral

Peale selle, kui komponendi lähtekood on valmis kirjutatud, siis testitakse, kuidas käitub tarkvara koostatud riistvaral. Selle tegevuse eesmärgiks on testida programmikoodi loogikat ja töökindlust ning prototüübi riistvara. CLI-l ehk käsurea liidesel ja RGB LED-i juhtimisel on peatükis 2.2 väljatoodud nõuded täidetud. Kuid komponentide valideerimisel tulid välja 1 põhiline riistvarakoostamise viga.

Käsurea liidest saab kasutada erinevate seriaal terminalidega(Näiteks Arduino IDE Serial Monitor), millega on võimalik saata seadmele erinevaid andmeid. Vastuvõetavate ja saadetavate andmete edastamiskiiruseks on 115200 boodi. Saadetavate käskude lõpus peab olema lisatud rea lõppu tähis. Tihti peale seda saab seadistada terminal programmis. Eestikeelse käskude kirjeldus on välja toodud Joonis 13 ja ingliskeelne Lisa 6.



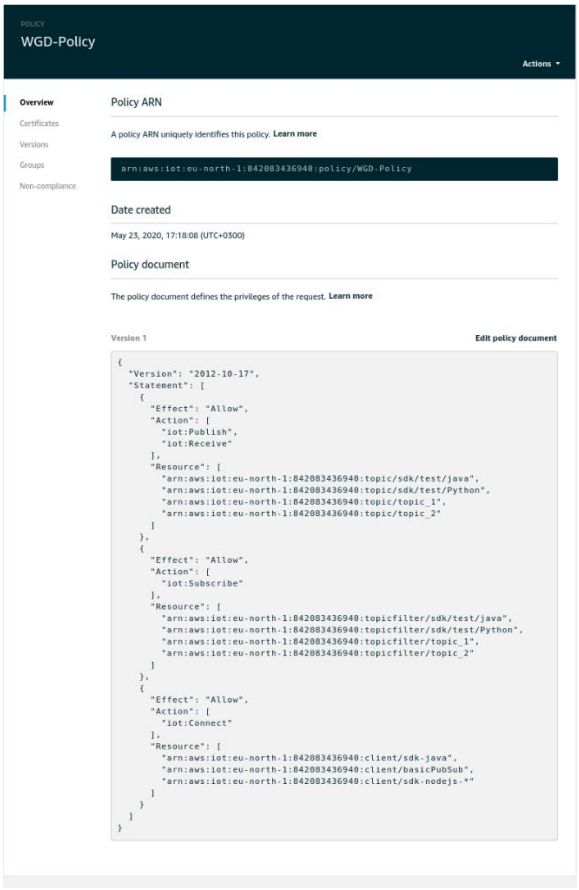
Joonis 13. Eestikeelne käsurea liides

Nuppude valideerimisel selgus, et riistvara koostamisel viigud numbritega 36 ja 39 ei ole kasutatavad, kui kasutatakse ADC ehk analoog-digitaal konverterit, sest see põhjustab sisendpinge ujumist, mille tõttu, kui Wi-Fi, mis kasutab ADC-d oli sisse lülitatud, tekkisid järjest katkestused, et nuppu on vajutatud, kuigi nuppu ei oldud vajutatud. Riistvaralisel koostamisel selle vea põhjustatud dokumentatsioonis mitte välja toodud erand, mida on mainitud sisend/väljund komponendi teegi päisfailis[7]. Kuna tegemist oli prototüüpplaadil ainult ridade ümber jootmisega, siis see parandus viidi sisse jooksvalt. Viigud 36 ja 39 asendati viikude 25 ja 26. Parandus on väljatoodud Lisa 7 pildil.

4.5 Juhtmevaba suhtlus serveriga

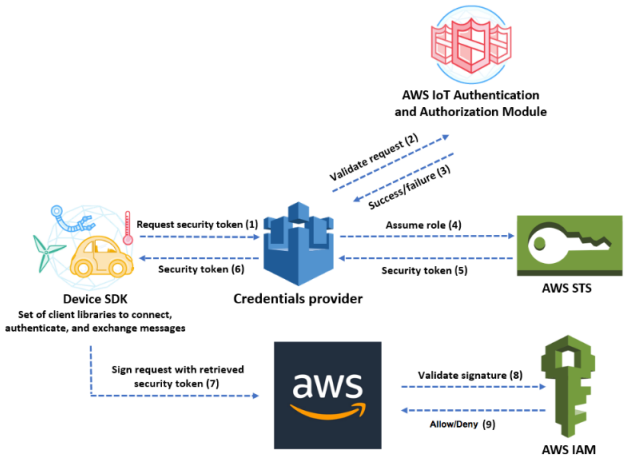
Suhtluseks serveriga on valitud MQTT, kuna see on Espressifi ESP-IDF programmikoodis komponendina saadaval[4]. MQTT on transpordiprotokoll, mis on just arendatud välja välkvõrgu seadmete suhtluseks. Suhtlus põhineb kliendi ja serveri ehk *broker*i omavahelisel teate avaldamise ehk *publish*i ning tellimise ehk *subsrcibe*'i põhjal kasutades *topic*uid ehk teemasid, mille põhjal loetakse ja kirjutatakse serverisse sõnumeid koos andmetega[3].

Serveriks on valitud Amazoni poolt pakutav pilveteenus *AWS IoT Core*. Seal on väga lihtne hallata erinevaid värkvõrgu seadmeid ja nendelt tulevaid andmeid. Seadme registreerimiseks tuleb luua *policy* ehk poliis(Vt Joonis 14. Näide seadme poliisist), mis kirjeldab erinevate seadmete reegleid, mille põhjal võib seade teha erinevaid toiminguid serveriga. Samuti igale poliisile genereeritakse ja registreeritakse erinevad sertifikaadid ja võtmed, mida saab anda seadmele turvaliseks suhtluseks üle võrgu.



Joonis 14. Näide seadme poliisist

Suhtluseks serveriga algab pääsmiku ehk *security tokeni* küsimisest. Selleks teeb seade HTTPS päringu kriteeriumite pakkujale ehk *credential providerile*, mis edastab päringu edasi *AWS IoT Core'i* autentimise ja volituse moodulile. Moodulis valideeritakse seadme sertifikaat ja verifitseeritakse seadme pääsmiku saamise õigsust. Kui seadmel puuduvad õigused, saadetakse seadmele selle kohane info tagasi. Vastasel juhul, kui valideerimine õnnestus, kriteeriumite pakkuja saadab seadmele pääsmiku, mille abil saab seade saata erinevaid päringuid serverile. Päringute tegemise õigsust otsustatakse poliisi põhjal võtme valideerimisel.[11]



Joonis 15. Seadme autentimine *AWS IoT Core*'iga[12]

5 IoT seadme turvalisus

Kasutaja andmete kaitsemiseks peab olema juhtmevaba ühenduse andmeliiklus kuidagi kaitstud. Samuti tuleb kaitsta serverit, et sinna ei ühendaks ennast kolmanda osapoole seade. Kui seadme turvalisuse ühenduse informatsioon asub seadmel endal, siis tuleb ka seadmel olevaid andmeid kaitsta, et neile puuduks riistvaraline ligipääs programmikoodi lugemiseks või seadme programmikoodi üle kirjutamiseks.

5.1 Seadme tarkvara turvalisus

Vältimaks seadmel olevat informatsiooni ja programmikoodi võtmist, mahalaadimist või kustutamist on MCU ESP32 tootja Espressif välja pakkunud 2 lahendust, mida on võimalik kontrolleril kasutada. Nendeks on turvaline alglaadimine ja välkmälu krüpteerimine.

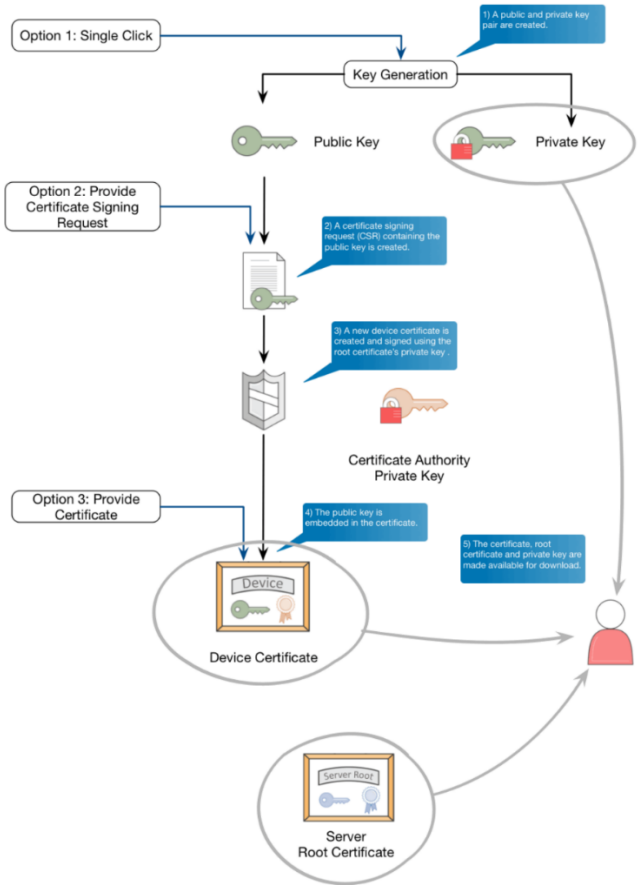
Turvaline alglaadimine ehk *secure bootloader*, kindlustab selle, et ainult enda loodud programmikoodi on võimalik jooksutada kontrolleril. Selleks kasutatakse kahte võtit: turvalise alglaadimise ja *secure boot signing* ehk turvalise buudi allkirjastamise võti. Turvalise alglaadimise võti genereeritakse protsessori alglaadimise sisemise riistvaralise juhusliku numbri generaatoriga ning salvestatakse eFUSE mälublokki. Vältimaks tarkvaralist ligipääsu võtmele on see mälublokk kaitstud lugemise ja kirjutamise eest. Turvalise buudi allkirjastamise võti sisaldab kahte avaliku ja privaatse võtme paari, mis genereeritakse programmikoodi ehitamisel. Avalik võti on kompileeritud tarkvaralisse alglaadimisse ja verifitseeritakse alglaadimise teises etapis.[18]

Välkmälu krüpteerimiseks krüpteerimisvõti genereeritakse ja salvestatakse samamoodi nagu turvalise alglaadimise puhul. Krüpteerimiseks kõrvetatakse lihttekstina andmed ESP32-le ja alglaadimine krüpteerib selle sisu esimesel MCU alglaadimisel. Kõiki andmeid ei krüpteerita vaid ainult partitsiooni tabelit ja seal tabelis kirjeldatud krüpteerimislipuga ja *„app“* tüüpi partitsioone. Kui mikrokontrolleril on esimesel alglaadimisel krüpteerimine lubatud, siis on võimalik sellele maksimaalselt 3 korda programmikoodi kõrvetada aga tarkvaraarenduse käigus on võimalik valida ka

eelgenereeritud välkmälu krüpteerimine, mille abil saab sellisest piirangust mööda pääseda.[19]

5.2 Juhtmevaba ühenduse turvalisuse tagamine

AWS IoT Core'iga, kui selles töös kasutatava MQTT serverina, saab ühenduda avalikuvõtme X.509 sertifikaatidega, mis kasutavad TLS ehk transpordi turbeprotokolli mõlemapoolset autentimisprotokolle[11]. Seadmel on 2 sertifikaati, millega käib seadme autentimine ning privaatse võtmega käib volituse krüpteerimine, mis genereeritakse AWS kasutaja konsoolis vastavalt Joonis 16 kirjeldatud mudelile. Need sertifikaadid ja privaatvõti kirjutatud programmikoodi sisse ning kui hakkab toimuma suhtlus, siis sellega tegeleb juba sellele vastavalt kirjutatud komponent AWS, mida on mainitud peatükis 4.1.



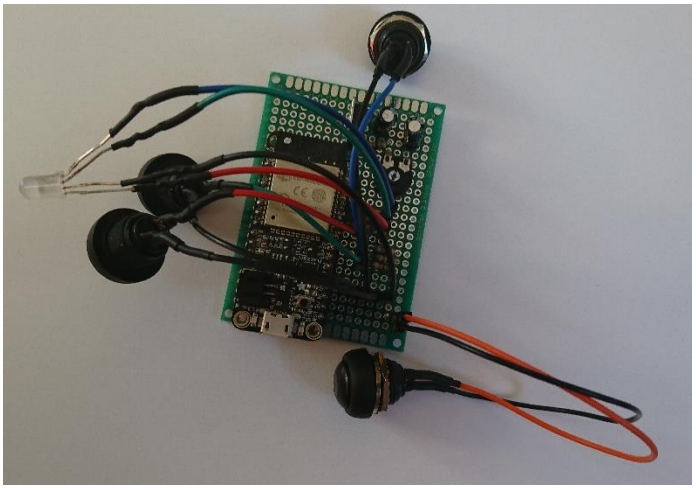
Joonis 16. Seadmele AWS-i sertifikaadi ja võtmete loomise mudel[13]

6 Hindamisseadme prototüübi testimine ja analüüs

Testimise käigus uuritakse, kas prototüüp on võimeline tagama nõuetele vastavad funktsionaalsused. Analüüsitakse loodud prototüübi riistvaralisi ja tarkvaralisi puudusi. Testimise ja analüüsi põhjal pakutakse välja erinevaid võimalusi, kuidas saaks prototüüpi edasi arendada, et selleks saaks lõpuks juhtmevaba hindamisseade.

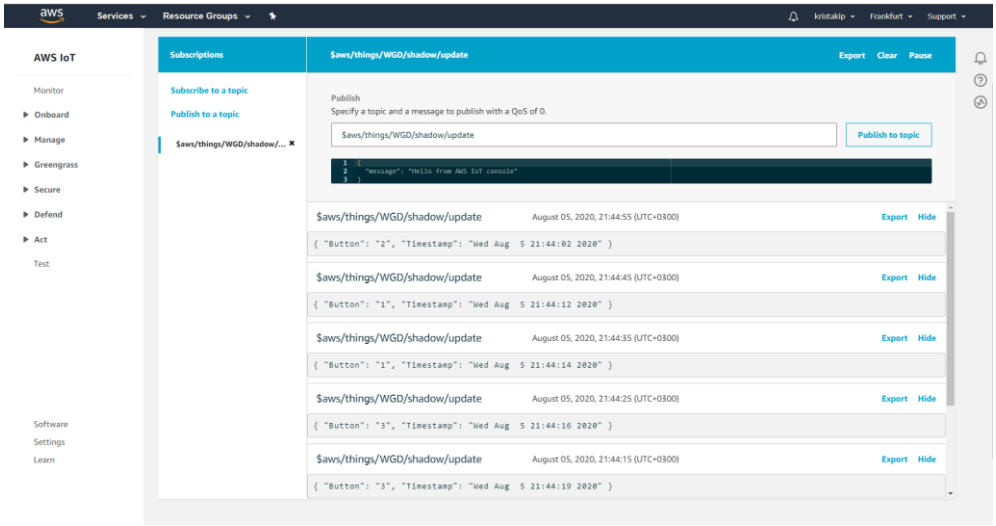
6.1 Funktsionaalsuse testimine

Prototüübi testimiseks ühendati makettplaadile vastavad vajalikud komponendid(Vt Joonis 17) ja see ühendati akupangaga. Enne akupangaga ühendamist, seadistati arvuti abiga ühendatava Wi-Fi võrgu parameetrid.



Joonis 17. Prototüüp

Pealmised funktsionaalsused nupuvajutuse registreerimine ja andmete jõudmine serverisse on saavutatud. Tulemus on näha Joonis 18. Andmeid saab koguda läbi AWS kasutaja konsooli Test aknast ja neid jälgida reaalajas ning vajadusel eksportida nii JSON-i kui ka CVS failina. Infona edastatakse vastava nupu number ning ajamärge, millal vajutus toimus.



Joonis 18. Kuvatõmmis kohale jõudnud andmetes AWS kasutaja konsoolis

6.2 Prototüübi puudused

Prototüübi testimise käigus jõuti järeldusele, et süsteemi planeerimisel ei oldud arvestatud voolutarbe optimeerimisega. Praeguses olukorras teoreetiliselt seade minimaalselt tarvitaks 68 mA ja maksimaalselt 240mA [2]. Võttes akupanga, mille mahtuvuseks on 2500mAh, siis prototüübi tööeluiga oleks maksimaalselt

$\frac{2500mAh}{68mA} \approx 36h$ ja minimaalselt $\frac{2500mth}{240nA} \approx 10h$.

Esimesena testimise käigus panti tähele, et seadet ei ole võimalik sisse ja välja lülitada, et seda saab saavutada ainult vooluallika järele ja lahti ühendamisel. Teiseks oleks võinud prototüübile lisada programmi unerežiim, kuid praeguse riistvara puhul ei saa seda rakendada, kuna tarkvaraliselt on mikrokontrollerit ainult „üles äratada“ siis, kui üks määratud sisendist läheb kõrgesse olekusse[5]. Selle jaoks on vajalik teha suurem riistvara muudatus, kus nuppude väljaviigud tuleb ülestõmbamise asemel hoopis alla tõmmata. Samuti tuleks muuta nuppude vajutamise loogikat programmikoodis.

6.3 Riist- ja tarkvaraline edasi arendus

Seadme optimeerimiseks ja kompaktsuse saavutamiseks tuleks koostada eraldi trükkplaat, millele saaks lisada suurema mälumahuga ESP32 mudel, seadme sisse ja välja lülitamine ja programmeerimise ning silumise UART-ist eraldi seisev vooluahel.

Sealhulgas ka vajadusel eraldi trükkplaat, kus asuvad seadmel olevad nupud ja RBG LED, ning mida saaks kaabliga ühendada põhiplaadi külge, kus asub ESP32.

Käesoleva projektis kasutati Adafruidi poolt toodetud eraldi arendusplaati, mille ESP32 WROOM-32-l on ainult 4GB välkmälu. Suurem välkmälu maht võimaldaks seadmele teha OTA-t (*Over the Air Updates*) ehk juhtmevaba programmikoodi uuendamist, mis tulevikus võimaldaks tarkvara uuendada ilma seadet puutumata.

Eraldi seisva vooluahel võimaldaks lisada erinevaid akusid ilma, et neil peaks olema väljundpinge vastavalt mikro-USB standardile. Läbi selle saaks seadme tööaega mõjutada vastavalt kasutaja soovidele ja kasutamisele. Samuti saaks lisada seadme sisse- ja väljalülitamise lüliti.

Kui trükkplaadi disain on otsustatud ja lõpetatud, ning võimalikud kasutatavad akud välja valitud, tuleks koostada seadmele eraldi korpus, mida saaks kasutaja vastavalt enda äranägemise järgi kujundada.

Tarkvaraliselt, kui seade jõuab tootmisfaasi, tuleks ära otsustada, kuidas teha seadme sätestamine ehk *device provisioning*, mille abil antakse igale seadmele unikaalsed sertifikaadid serveriga suhtluseks. Robustsem meetod oleks iga seadme programmeerimisel kaasa anda eraldi genereeritud privaatvõti ja sertifikaat, kuid kui jätkata kasutamist AWS teenust AWSIoT on võimalik seadmel küsida esimesel ühendumisel serveriga endale sertifikaadid.

7 Kokkuvõte

Käesoleva töö tulemusena sai loodud juhtmevaba hindamisseadme prototüüp. Prototüübi riistvara disainiti Adafruit HUZZAH32 *Feather* arendusplaadi, millel on Espressifi integreeritud Wi-Fi mooduliga mikrokontrolleril ESP32, põhjal ning tarkvara arendamisel kasutati ESP-IDF raamistiku. Vastavalt nõuetele ja arhitektuurile tarkvara arendamise käigus, peale baas funktsionaalsuste: nupu vajutuse registreerimine, kasutajale selle kohase tagasiside andmise RGB LED abil ja info jõudmine serverisse, loodi ka käsurea liides, mille abil saab seadistada Wi-Fi võrku ühendamiseks vastavad parameetrid.

Andmete kogumiseks kasutati *Amazon Web Service IoT Core* serverit, mille turvalise suhtlemiseks seadmega kasutati MQTT transpordiprotokolli, kasutades pakutava AWS teenuse pakutud turvalisuse põhimõtteid. Peale turvalise andmeside on võimalik seadel asuvaid andmeid ja programmikoodi kaitsta kasutades turvalist alglaadimist ja välkmälu krüpteerimist.

Prototüübi testimise käigul avastati, et nõuete koostamisel ei võetud arvesse voolutarbe optimeerimist, mis on ka selle prototüübi suurim puudus. Kuid seda saab rakendada tulevasel edasiarendusel, mille käigus on selle töö põhjal võimalik koostada vastav trükkplaat, millel oleks eraldi seisev vooluahel ja suurema välkmäluga ESP32 moodul.

Kasutatud kirjandus

[1] ESP8266 Datasheet [WWW/PDF]https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf(30.04.2020)

[2] ESP32-WROOM-32 Datasheet [WWW/PDF]https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf(30.04.2020)

[3] MQTT Version 3.1.1 OASIS Standard [WWW/PDF]http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf(05.05.2020)

[4] ESP-IDF Programming Guide: ESP-MQTT, Espressif [WWW] https://docs.espressif.com/projects/esp-idf/en/v3.3.2/api-reference/protocols/mqtt.html(05.05.2020)

[5] ESP-IDF Programming Guide: Get Started, Espressif [WWW] https://docs.espressif.com/projects/esp-idf/en/v3.3.2/get-started/index.html(15.05.2020)

[6] Neil Kolban, ESP32 Snippets [lähtekood]https://github.com/nkolban/esp32-snippets(29.01.2020)

[7] Espressif, Espressif IoT Development Framework [lähtekood]https://github.com/espressif/esp-idf/tree/v3.3.2(29.01.2020)

[8] Downloads, Adafruit HUZZAH32 - ESP32 Feather [WWW]https://learn.adafruit.com/adafruit-huzzah32-esp32-feather/downloads(30.04.2020)

[9] Pinouts, Adafruit HUZZAH32 - ESP32 Feather[WWW]https://learn.adafruit.com/adafruit-huzzah32-esp32-feather/pinouts(30.04.2020)

[10] AWS IoT Developer Guide: How AWS IoT works [WWW]https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html(04.05.2020)

[11] AWS IoT Developer Guide: Authorizing direct calls to AWS services [WWW]https://docs.aws.amazon.com/iot/latest/developerguide/authorizing-direct-aws.html(04.05.2020)

[12] AWS IoT Developer Guide: Security [WWW]https://docs.aws.amazon.com/iot/latest/developerguide/security.html(04.05.2020)

[13] Understanding the AWS IoT Security Model , The Internet of Things on AWS – official Blog, Nick Corbett[WWW]https://aws.amazon.com/blogs/iot/understanding-the-aws-iot-security-model/(04.05.2020)

[14] Quick CMake Tutorial, JetBrains [WWW/HTML] https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html(20.07.2020)

[15] Loengu hindamise seade, Krista Kippar [WWW/PDF]http://www.tud.ttu.ee/web/krkipp/arvutidjasusteemidprojekt_Kippar_2.pdf(02.03.2020)

[16] Juhtmevaba hindamise seade, Krista Kippar [WWW/PDF]http://www.tud.ttu.ee/web/krkipp/Arvutisusteemid_projekt_Kippar_163896.pdf(10.05.2020)

[17] CMake Overiview [WWW]https://cmake.org/overview/ (25.07.2020)

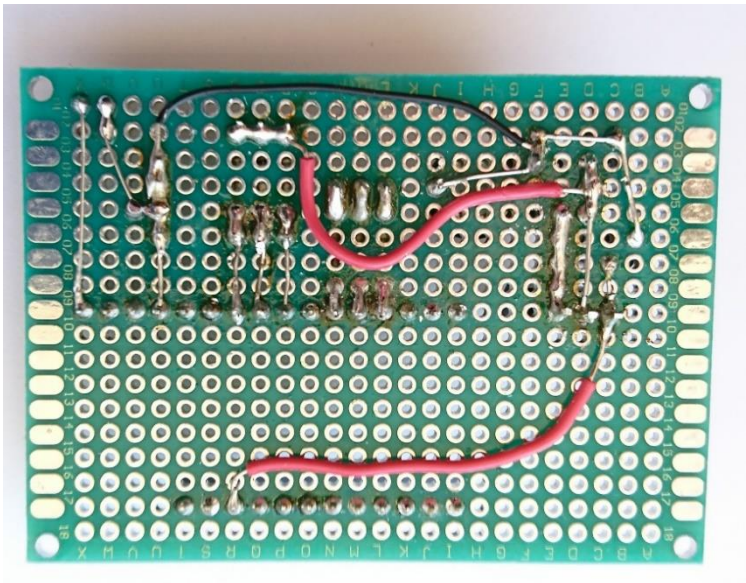
[18] Secure boot, ESP-IDF Programming Guide[WWW]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/secure-boot-v1.html(17.05.2020)

[19] Flash Encryption, ESP-IDF Programming Guide [WWW]https://docs.espressif.com/projects/esp-idf/en/v3.3/security/flash-encryption.html#flash-encryption(17.05.2020)

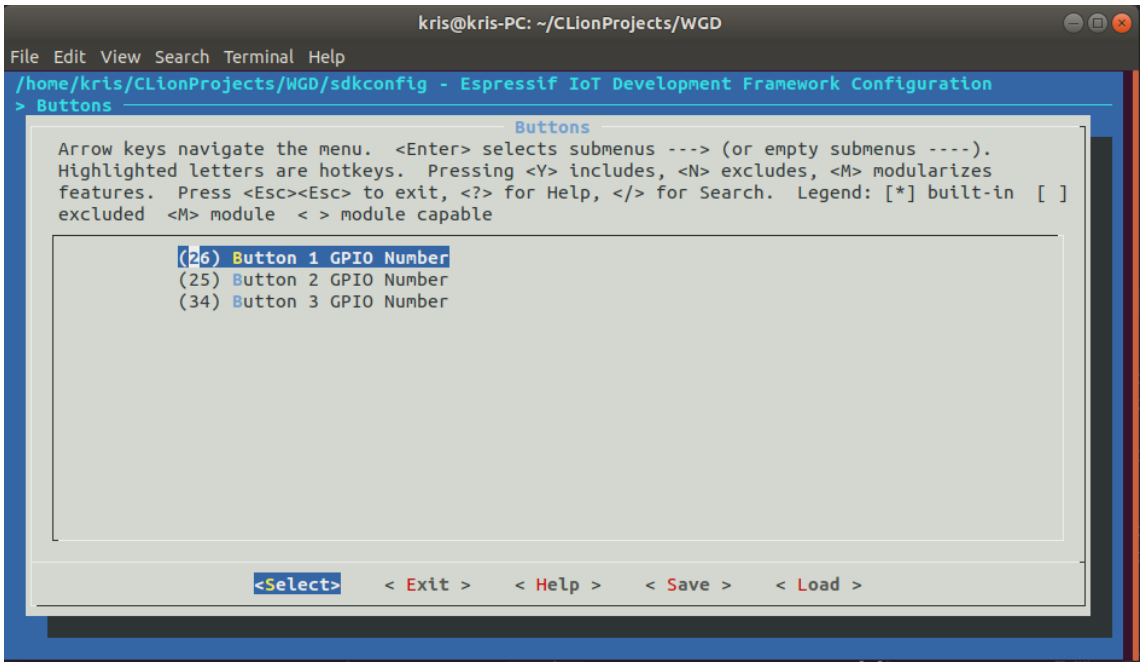
[20] The FreeRTOS [WWW]https://www.freertos.org/RTOS.html(25.07.2020)

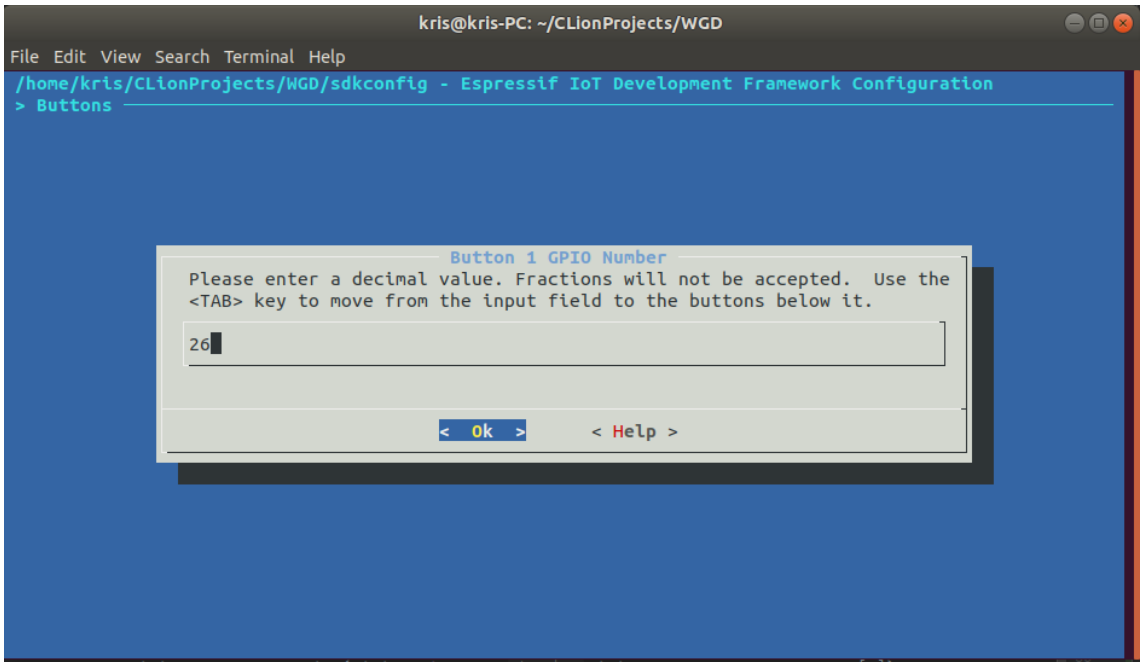
[21] Krista Kippar (2020), WGD[lähtekood] https://gitlab.pld.ttu.ee/krkipp/WGD.git(03.08.2020)

Lisa 1 – Pilt prototüübi plaadi jootest

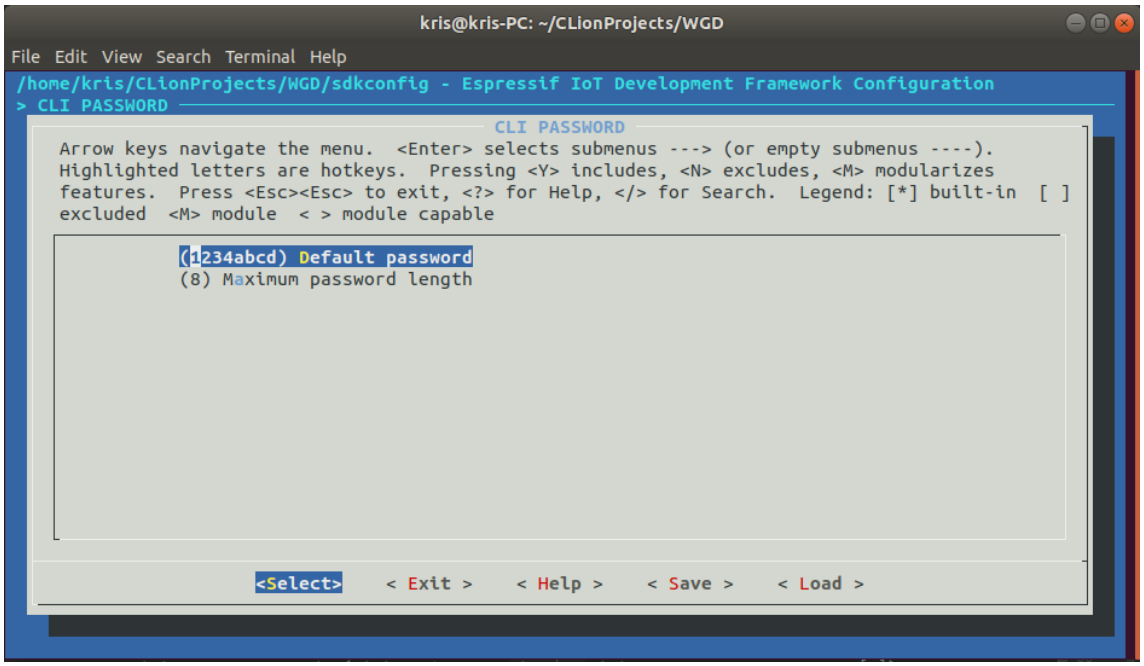


Lisa 2 Näide *menuconfigis* nuppude määramine

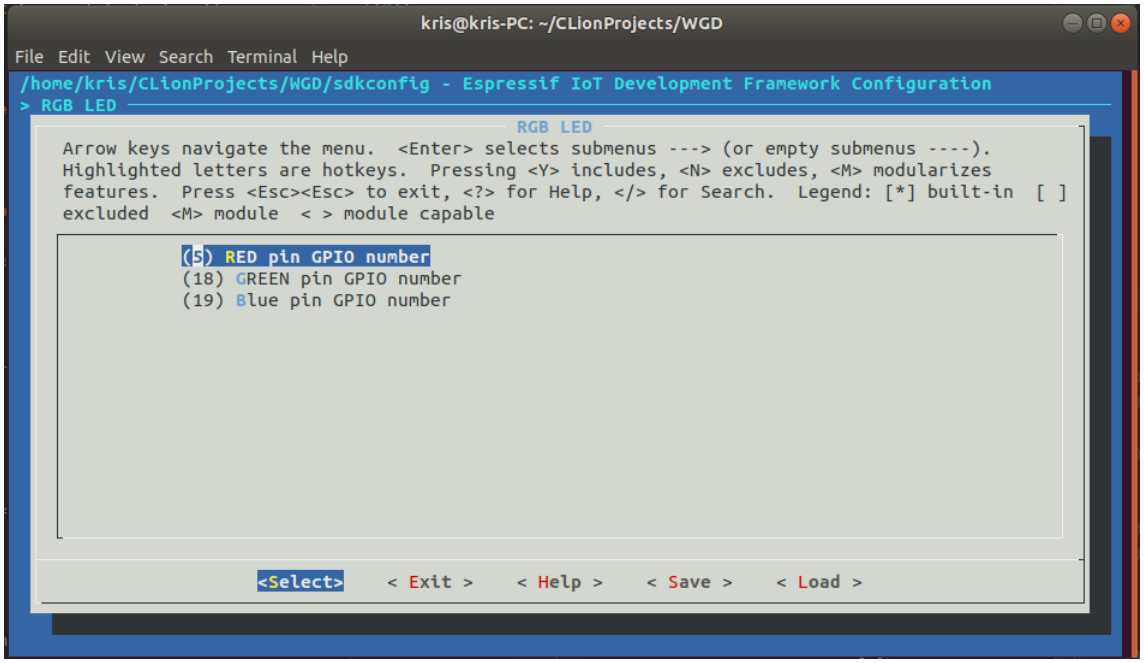




Lisa 3 Näide *menuconfigis* algse salasõna seadistamine



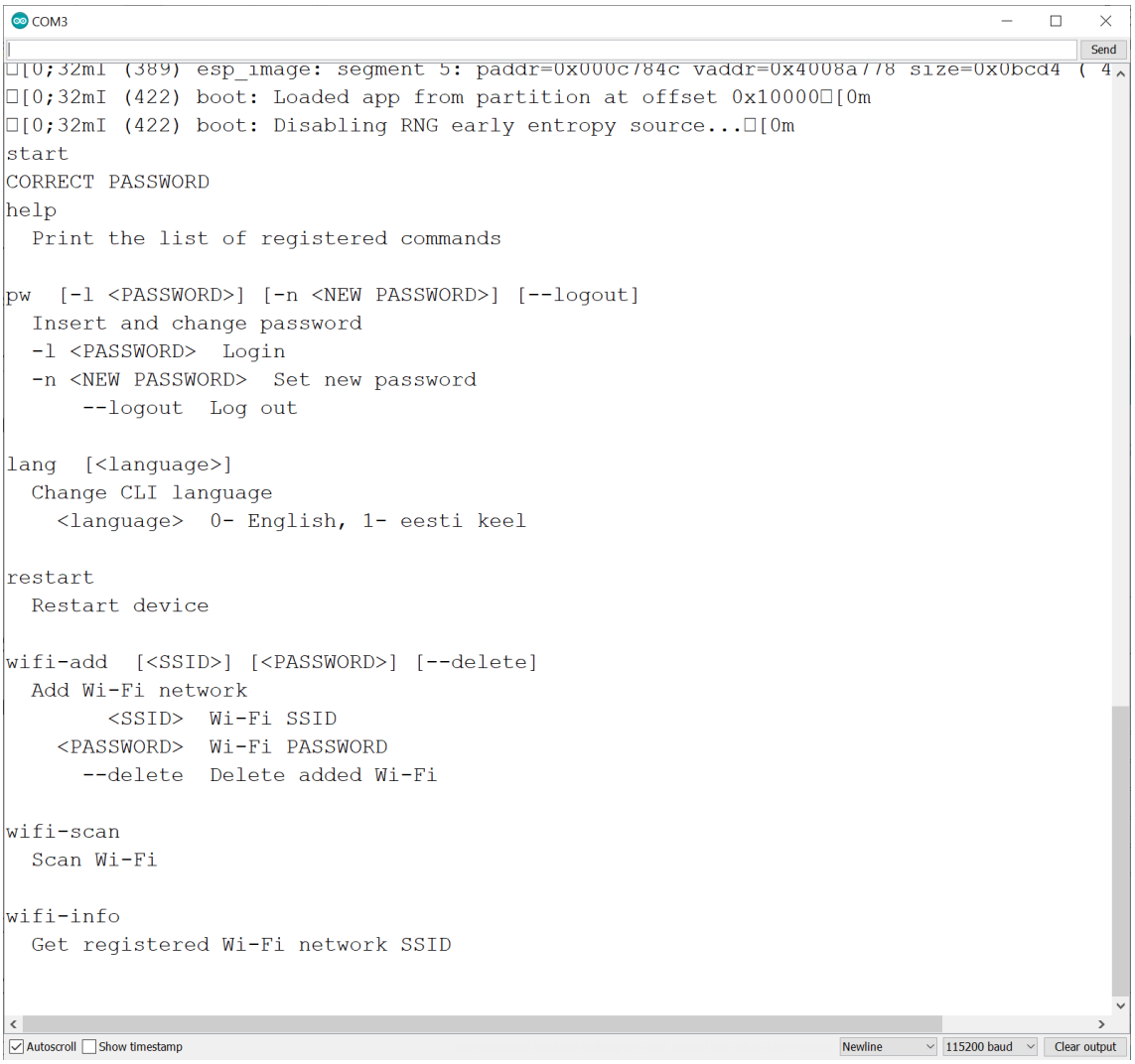
Lisa 4 Näide *menuconfigis* RGB LED viikude määramine



Lisa 5 Tarkvara põhiprogramm

```
void app_main()
{
    //initialize NVS storage
    auto nvs = NVSStorage::get_class_instance();
    esp_reset_reason_t reset_reason = esp_reset_reason();
    //printf("start\n");
    //initialize RGB LEB
    auto rgb = new RGB();
    //initialize console
    auto cli = new CLI();
    //start command line interface task
    cli->start();
    //initialize
    auto wifi = new Wifi(reset_reason, *cli, *rgb);
    // start Wifi task
    wifi->start();
    //initialize AWS task
    auto aws = new AWS();
    //initialize buttons and their interrupts
    auto button1 = new Button(1,(gpio_num_t)CONFIG_BUTTON1_GPIO_NUM, *aws,
*rgb);
    auto button2 = new Button(2, (gpio_num_t)CONFIG_BUTTON2_GPIO_NUM, *aws,
*rgb);
    auto button3 = new Button(3, (gpio_num_t)CONFIG_BUTTON3_GPIO_NUM, *aws,
*rgb);

    int i = 0;
    //main loop
    for (;; ) {
//        for developing
//        if(i%10 == 0){
//            printf("%d seconds...\n", i);
//        }
//        i += 1;
        vTaskDelay(1000 / portTICK_PERIOD_MS);
        if (wifi->is_connected){
            if (!aws->is_started){
//                printf("AWS START\n");
                aws->start();
                aws->is_started = true;
            }
        } else{
            if (aws->is_started){
                aws->stop();
                aws->is_started = false;
            }
        }
    }}
```

Lisa 6 Ingliskeelne käsurealiides



Lisa 7 Nuppude viikude riistvaraline joote parandus