Tampereen ammattikorkeakoulu

Testausautomaatio sulautetun järjestelmän kehityksessä

Jani Iivonen

OPINNÄYTETYÖ
Syyskuu 2023

Tietotekniikka
Sulautetut järjestelmät ja elektroniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Sulautetut järjestelmät ja elektroniikka

IIVONEN, JANI:
Testausautomaatio sulautetun järjestelmän kehityksessä

Opinnäytetyö 54 sivua, joista liitteitä 20 sivua
Syyskuu 2023

Opinnäytetyön tarkoituksena oli tutustua testausautomaatiojärjestelmän luomiseen ja käyttämiseen sulautetun järjestelmän kehityksessä. Työ toteutettiin Huld Oy:n toimeksiannosta. Lisäksi Huld Oy:n sisäiseen käyttöön toteutettiin ohje koodin kääntämisestä ja lataamisesta STM32-mikrokontrollerille Docker-kontista.

Opinnäytetyössä tutustuttiin testausautomaatiojärjestelmässä käytettyihin työkaluihin ja ohjelmistoihin teoriatasolla. Tämän jälkeen luotiin testattavaksi laitteeksi yksinkertainen ilmanvaihdon ohjausjärjestelmä. Testattavan laitteen ulostulojen tilaa tarkkailtiin Arduino-mikrokontrollerilla.

Testattavalle laitteelle luotiin testausautomaatiojärjestelmä Jenkins-automaatioserverillä, jolla ohjattiin Docker-kontteja. Testausautomaatiojärjestelmään luotiin kaksi Docker-konttia. Toisessa Docker-kontissa käännettiin STM32-mikrokontrollerille ladattava koodi binäärimuotoon ja toisessa Docker-kontissa ladattiin koodit Robot Frameworkilla testerinä toimivalle Arduino-mikrokontrollerille ja testattavana laitteena toimivalle STM32-mikrokontrollerille. Koodien lataamisen jälkeen ajettiin Robot Frameworkilla testit, joilla voitiin varmistua testattavana laitteena toimivan STM32-mikrokontrollerin tarkoituksenmukaisesta toiminnasta.

Lopputuloksena saatiin toimiva testausautomaatiojärjestelmä, jolla voitiin testata ilmanvaihdon ohjausjärjestelmän toimintaa eri lämpötiloilla. Robot Frameworkilla ohjattiin ilmanvaihdon ohjausjärjestelmää UART-sarjaväylän yli ja ilmanvaihdon ohjausjärjestelmän tiloja ja tuuletinnopeutta tarkkailtiin Arduino-mikrokontrollerin avulla, joka lähetti ilmanvaihdon ohjausjärjestelmän ulostulojen tilan UART-sarjaväylän yli Robot Frameworkille.

Asiasanat: testausautomaatio, sulautetut järjestelmät, Robot Framework, Jenkins, Docker

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Embedded Systems and Electronics

IIVONEN, JANI:
Test Automation in Embedded Systems Development

Bachelor's thesis 54 pages, appendices 20 pages
September 2023

The purpose of this thesis was to learn about the usage of test automation in embedded system development. The theoretical section explored the tools and the programmes needed for testing embedded systems. The practical part consisted of the creation of a simple test automation system for the STM32 microcontroller. An air conditioning controller was created for a device under the test that ran on the STM32 microcontroller. An Arduino microcontroller was used in the test automation system to read the STM32 microcontroller pin states. The Jenkins was used as a test automation server that controlled the two Docker containers. One Docker container compiled the code to binary that would be flashed to the STM32. Another container ran the Robot Framework tests. The Robot Framework tests flash Arduino and STM32 microcontrollers and tests that code loaded to STM32 works as expected.

The result of this thesis is a working test automation system that downloads the latest version of the code from GitHub, compiles code that will be flashed to STM32 microcontroller and tests flashed code to ensure it is working correctly. The test automation system generates a report from the test results. Additionally, instructions were created about compiling and flashing code to the STM32 microcontroller from the Docker container.

Key words: test automation, embedded systems, Robot Framework, Jenkins, Docker

SISÄLLYS

1 JOHDANTO

Testausautomaatio on nykypäivänä iso osa ohjelmistokehitystä. Sen avulla voidaan varmistaa koodin toiminta automaattisesti jokaisen muutoksen jälkeen. Testausautomaatiolla voidaan säästää ohjelmistokehittäjän aikaa pitkällä aikavälillä, sillä testit koodille tehdään automaattisesti aina, kun versionhallintajärjestelmään ladataan uusi versio. Lisäksi testausautomaatiolla tehtävät testit tapahtuvat aina ennalta määritellyn testiskenaarion mukaan. Testit tehdään aina samalla tavalla, jolloin virheiden mahdollisuus pienenee toisin kuin manuaalisessa testauksessa, jossa inhimilliset virheet ovat mahdollisia.

Testausautomaatiolla voidaan testata ohjelmistojen lisäksi myös sulautettuja järjestelmiä. Opinnäytetyön toimeksiantaja Huld Oy antoi tehtäväksi tutustua testausautomaation käyttöön ja testien luomiseen sulautetun järjestelmän kehityksessä. Automaatioserverinä käytettiin tässä opinnäytetyössä Jenkinsiä, jolla ohjattiin Docker-kontteja. Testit toteutettiin Robot Frameworkilla. Testausautomaatioon tutustumisen lisäksi tehtiin Huld Oy:n sisäiseen käyttöön ohje siitä, kuinka kääntää ja ladata koodi STM32-mikrokontrollerille Docker-konttien avulla.

Luvuissa 2 ja 3 perehdytään testausautomaatiojärjestelmän luomiseen tarvittaviin ohjelmistoihin ja työkaluihin teoriatasolla. Tämän jälkeen toteutetaan yksinkertainen testausautomaatiojärjestelmä, jossa koodit ladataan mikrokontrollereille ja suoritetaan testattavalle laitteelle testit.

2 TESTAUSAUTOMAATIO

Testausautomaatiota käytetään yleensä ohjelmistokehityksessä automatisoimaan manuaalisesti tehtävät testit. Tällä säästetään testaajan aikaa pitkällä aikavälillä, koska testejä ei tarvitse tehdä manuaalisesti aina, kun testattavaan ohjelmaan tai laitteeseen tulee muutoksia. Testausautomaatio minimoi oikein tehtynä testauksen aikana tapahtuvat virheet, jolloin testit ovat joka kerta identtisiä ja testeistä saatavat raportit ovat näin ollen vertailukelpoisia. (Sambamurthy 2023, 1.)

Testausautomaatiolla voidaan testata esimerkiksi ohjelmistoja ja sulautettuja järjestelmiä. Sen tavoitteena on varmistaa testattavan ohjelman tai laitteen tarkoituksenmukainen toiminta ja löytää mahdolliset virheet (Sambamurthy 2023, 1).

Testaaminen on suositeltavaa aloittaa mahdollisimman aikaisessa vaiheessa, jolloin mahdolliset viat voidaan löytää mahdollisimman nopeasti ja ne ehditään korjata. Löydetyt virheet on helpompi korjata mahdollisimman aikaisessa vaiheessa, esimerkiksi jos virhe jäisi valmiiseen ohjelmistoon, voisi sen korjaus hajottaa muita ominaisuuksia. Testausta on lisäksi suositeltavaa suorittaa mahdollisimman usein, jotta testien välillä ohjelmistoon tai muuhun testattavaan asiaan ehtii tulla mahdollisimman vähän muutoksia. Tällöin myös virheet huomataan nopeasti ja ne voidaan korjata. (Sambamurthy 2023, 1.)

Testaamista ei yleensä voida toteuttaa täysin automaattisesti, vaan osa testeistä tulee tehdä manuaalisesti. Esimerkiksi testattavan ohjelmiston tai asian toimintoja voidaan testata ilman ennalta tehtyä testaussuunnitelmaa, jolloin tavoitteena on tutkia testattavan laitteen tai asian toimintoja ja etsiä virheitä. (Sambamurthy 2023, 1.)

3 TYÖKALUT JA OHJELMISTOT

3.1 Git

Git on ilmainen ja avoimeen lähdekoodiin perustuva versionhallintajärjestelmä (Free and Open Source n.d.). Versionhallintajärjestelmiä käytetään tiedostomuutosten seuraamiseen. Esimerkiksi ohjelmistotuotannossa käytetään usein versionhallintajärjestelmiä, koska versionhallintajärjestelmän avulla moni ohjelmistokehittäjä voi kehittää ohjelmistoa samanaikaisesti. Versionhallintajärjestelmä mahdollistaa myös tiedostojen palauttamisen aikaisempaan versioon. Ohjelmistotuotannossa tämä mahdollistaa palaamisen edelliseen toimivaan versioon, jos ohjelmiston kehityksessä tapahtuu virhe ja ohjelmisto ei toimi odotetusti tai jos jokin tiedosto poistuu vahingossa. (Chacon & Straub 2023, 10–13.)

Git tehtiin alun perin Linuxin kernelin kehitystä varten. Linuxin kernelin suuren koon takia Gitin tuli pystyä hallitsemaan suuria repositorioita nopeasti. (Small and Fast n.d.) Gitin nopeus perustuu siihen, että suurin osa toiminnoista tapahtuu paikallisesti, joten internetyhteys ei hidasta Gitin toimintaa (Chacon & Straub 2023, 15–16).

Gitin avulla projektiin on mahdollista luoda haaroja. Esimerkiksi ohjelmistokehitysprojektissa voidaan jollekin uudelle ominaisuudelle luoda oma haara, jossa ominaisuutta kehitetään ja ominaisuuden ollessa valmis voidaan ominaisuus liittää päähaaraan. (Chacon & Straub 2023, 70–75.)

Yleensä Gitin repositorioiden jakamista ja säilyttämistä varten luodaan etärepositorio. Etärepositoriolla tarkoitetaan yleensä verkossa olevaa serveriä, johon paikalliseen repositorioon tehdyt muutokset voidaan ladata. Tällöin tehdyt muutokset ovat muiden käyttäjien ladattavissa ja etärepositorio toimii varmuuskopiona paikalliselle repositoriolle. Etärepositoriolle käyttäjä voi luoda oman serverin tai käyttää kolmannen osapuolen palvelua. Yksi parhaiten tunnetuista kolmannen osapuolen etärepositorio palveluista on GitHub. GitHub tarjoaa ilmaisen tallen-

nustilan repositorioille ja verkkopohjaisen käyttöliittymän. GitHubin avulla käyttäjien on helppo tallentaa repositorioita, jakaa niitä ja kehittää ohjelmistoja yhdessä muiden käyttäjien kanssa. (Spinellis 2012.)

3.2 VirtualBox

VirtualBox on Oraclen kehittämä virtualisointiohjelma, joka mahdollistaa käyttöjärjestelmien virtualisoinnin. VirtualBoxia voidaan käyttää usean virtualisoidun käyttöjärjestelmän ajamiseen samanaikaisesti yhden käyttöjärjestelmän sisällä. VirtualBoxin avulla jokaiselle ajettavalle käyttöjärjestelmälle luodaan oma virtuaalinen tietokone, mikä mahdollistaa käyttöjärjestelmän simuloimisen oikeassa tietokoneessa. (Oracle n.d.,1.)

VirtualBox mahdollistaa vanhojen käyttöjärjestelmien ajamisen nykyaikaisessa tietokoneessa, jolloin virtualisoidun käyttöjärjestelmän avulla voidaan käyttää nykyaikaisten käyttöjärjestelmien kanssa yhteensopimattomia ohjelmistoja. VirtualBox on myös hyvä työkalu testaamiseen silloin, kun testaamiseen tarvitaan monta käyttöjärjestelmää. VirtualBoxilla voidaan esimerkiksi luoda usean virtuaalisen tietokoneen verkko ja testata kehitettävää ohjelmaa ilman tarvetta usealle fyysiselle tietokoneelle. VirtualBox mahdollistaa myös käyttöjärjestelmän helpon varmuuskopioinnin. Tällöin vanhaan varmuuskopioon on helppo palata, jos käyttöjärjestelmä ei enää toimi. (Oracle n.d.,2.)

3.3 Ubuntu

Ubuntu on ilmainen ja avoimeen lähdekoodiin perustuva käyttöjärjestelmä, joka on haarautettu Debianista. Debian on vapaaehtoisten ylläpitämä Linux jakeluversio. Linuxin kernelin päälle on rakennettu useita eri jakeluversioita, kuten Ubuntu ja Debian. Jakeluversiot on suunniteltu erilaisiin käyttötarkoituksiin. Osassa jakeluversioita on graafinen käyttöliittymä ja osaa käytetään komentoriviltä. Jotkin jakeluversiot on suunniteltu käyttämään mahdollisimman vähän muistia ja toimimaan hyvin pienitehoisilla tietokoneilla tai sulautetussa järjestelmässä. (Nixon 2010,1.)

Ubuntun etuna voidaan pitää sen helppokäyttöisyyttä ja helppoa asennusta. Ubuntu soveltuu graafisen käyttöliittymänsä takia aloittelevalle Linuxin käyttäjälle. Ubuntun asennuksen yhteydessä käyttäjän on mahdollista asentaa tarvittavat ohjelmat automaattisesti. Näihin ohjelmiin kuuluu esimerkiksi selain ja toimisto-ohjelmat. Käyttäjän ei tarvitse syöttää komentoja komentoriville käyttääkseen yleisimpiä Ubuntun ominaisuuksia. Ubuntu kuitenkin mahdollistaa kokeneemmille käyttäjille laajan muokattavuuden. Koska Ubuntu on avoimen lähdekoodin käyttöjärjestelmä, voi kuka tahansa ladata Ubuntun lähdekoodin verkosta ja muokata sen haluamakseen ja jakaa sitä. Muokatun käyttöjärjestelmän jakamisen ehtona on, että muokattu versio tulee jakaa samoilla ehdoilla kuin Ubuntu. Eli muokatun version lähdekoodin tulee olla vapaasti saatavilla. (Nixon 2010,1.)

3.4 Jenkins

Jenkins on avoimeen lähdekoodiin perustuva automaatioserveri. Jenkins voidaan asentaa Docker-konttina tai sovelluksena (Jenkins User Documentation n.d.). Jenkins on mahdollista asentaa sovelluksena yleisimpiin käyttöjärjestelmiin, kuten Linuxiin, Windowsiin ja MacOS:ään (Installing Jenkins n.d.).

Jenkinsin avulla on mahdollista automatisoida ohjelmistojen kääntäminen, asentaminen ja testaaminen (Jenkins User Documentation n.d.). Jenkinsin avulla ohjelmistojen testaaminen voidaan automatisoida täysin. Jenkins voidaan konfiguroida siten, että se testaa ohjelmiston aina, kun versionhallintajärjestelmään tulee uusi versio ohjelmistosta. Tämä nopeuttaa ohjelmistokehitystä huomattavasti, koska ohjelmistokehittäjän ei tarvitse kääntää koodia ja testata sitä, sillä Jenkinsillä tämä voidaan automatisoida. (Pipeline n.d.)

Jenkinsiin on mahdollista luoda nodeja. Nodet toimivat Jenkins-serverin ohjaamina ja niillä hoidetaan ohjelmiston kääntäminen ja testaus. Nodet voivat olla fyysisiä tietokoneita tai pilvipalvelupohjaisia tietokoneita. Jenkins-nodeja voidaan luoda käyttöjärjestelmiin, jotka tukevat Java-ohjelmia, Docker- ja Kubernetes-kontteihin ja yleisimpiin pilvipalveluihin. Yksi Jenkins-serveri voi ohjata useaa nodea, jolloin ohjelmisto voidaan testata usealla eri konfiguraatiolla. Nodeissa voidaan käyttää eri käyttöjärjestelmiä ja oheislaitteita, jolloin Jenkins-serveri voi

valita sopivimman noden kyseisen ohjelmiston testaamiseen. Kun Jenkins-serveriin on yhdistetty useampi node, voidaan yhdellä Jenkins-serverillä testata useaa ohjelmistoa samanaikaisesti. Nodejen luonti mahdollistaa Jenkinsin hyvän skaalautuvuuden. Jos esimerkiksi nodeja tarvitaan lisää, voidaan niitä helposti luoda lisää pilvipalveluihin. (Managing Nodes n.d.)

3.5 Docker

Docker on ohjelmisto, jolla voidaan luoda käyttöjärjestelmästä eristettyjä kontteja, joiden sisällä voidaan ajaa ohjelmia. Docker-kontti on eristetty muusta järjestelmästä. Docker-kontissa oleva ohjelma ei pääse käsiksi tietokoneen muihin tiedostoihin eikä toisiin kontteihin, ellei näin ole erikseen määritelty. Docker-kontteja on mahdollista ajaa paikallisesti, virtuaalikoneessa tai pilvipalvelussa. Docker voidaan asentaa mille tahansa käyttöjärjestelmälle. (Overview n.d.)

Docker-kontin luontiin käytetään imagea. Image on pohja Docker-kontille, jonka päälle määritellään halutut toiminnot. Dockerista löytyy valmiina imaget yleisimpiin tarpeisiin, kuten Ubuntun image. Docker-kontin luontiin tarvitaan Dockerfile, jossa määritellään käytettävä image ja parametrit. Esimerkiksi Dockerfilessä voidaan määritellä, että käytetään Ubuntun imagea ja asennetaan halutut ohjelmat. Jos Docker ei löydä imagea paikallisesti, se lataa sen verkosta. Dockeria voidaan käyttää ohjelmistotestaukseen, sillä testattava ohjelmisto voidaan aina ajaa uudessa Docker-kontissa, jolloin edelliset testit eivät vaikuta testien tulokseen. Docker-kontteja on myös helppo poistaa ja jakaa. Esimerkiksi, jos Docker-kontissa ajettavasta ohjelmistosta löytyy virhe, voidaan virhe korjata ja jakaa uusi päivitetty image. (Docker objects n.d.)

3.6 Robot Framework

Robot Framework on Python-ohjelmoinkieleen pohjautuva avoimen lähdekoodin automaatiotestauksen sovelluskehys. Robot Frameworkilla voidaan luoda hyväksyntätestejä, yksikkötestejä ja ohjelmistorobotiikkaa (RPA). Robot Frameworkia käytetään avainsanoilla, joiden avulla määritellään haluttu toiminto ja sen para-

metrit. Robot Frameworkiin on mahdollista asentaa kirjastoja, jolloin Robot Frameworkin toimintoja on mahdollista laajentaa erilaisilla valmiilla kirjastoilla. Käyttäjät voivat myös tehdä itse omia kirjastoja. (Introduction n.d.)

Robot Frameworkin toiminta voidaan jakaa kahteen osaan, ohjelmistorobotiikkaan ja testaamiseen. Ohjelmistorobotiikkaa käytetään yleensä suorittamaan erilaisia toistuvia tehtäviä, jotka on mahdollista automatisoida. Tällaisia automatisoitavia tehtäviä ovat esimerkiksi tietokantaan tallentaminen tai sieltä lukeminen, tietojen hakeminen lomakkeilta ja laskelmien tekeminen. Prosessiautomaatiolla pyritään poistamaan ihmisiltä rutiininomaisia toistuvia tehtäviä. Tällä tehostetaan ihmisten työskentelyä, sillä tietokone suorittaa tehtävät paljon ihmistä nopeammin ja tarkemmin. (Taulli 2020,1.)

Robot Frameworkilla tehtävän testauksen tavoitteena on löytää testattavasta järjestelmästä virheitä ja raportoida löydetyt virheet. Testattava järjestelmä voi olla esimerkiksi tietokoneohjelma tai mikrokontrolleri-pohjainen järjestelmä. Automatisoidulla testauksella voidaan nopeuttaa testaamista ja vähentää ihmisten työtaakkaa. Automatisoidut testit etenevät aina samalla tavalla, joten testin aikana ei pääse syntymään virheitä, jotka olisivat ihmisen toteuttamina mahdollisia. (Bisht 2013,1.)

3.7 STM32

STM32 on 32-bittinen mikrokontrollerituoteperhe, joka pohjautuu ARM Cortex-m -prosessoriin. STM32-tuoteperhe tarjoaa hyvän suorituskyvyn, tuen reaaliaikasovelluksille, digitaalisen signaalin käsittelyn, matalan virrankulutuksen ja hyvän liitettävyyden. (STM32 32-bit Arm Cortex MCUs n.d.)

STM32-tuoteperheestä löytyy useita erilaisia mikrokontrollereita suurimpaan osaan käyttötarkoituksista. Tuoteperheestä löytyy tehokkaita mikrokontrollereita, jotka on suunniteltu raskaaseen käyttöön. Lisäksi löytyy yleiskäyttöön soveltuvia mikrokontrollereita, vähävirtaisia mikrokontrollereita ja mikrokontrollereita, joista löytyy langattomia yhteyksiä. (STM32 32-bit Arm Cortex MCUs n.d.)

STM32-tuoteperheen mikrokontrollerit ovat saatavilla myös Nucleo-tuoteperheen kehitysalustoissa. Nucleo-tuoteperheen kehitysalustat sisältävät tarvittavan elektroniikan STM32-mikrokontrollerin käyttöön ja ohjelmointiin. Tällöin kehitysalustaan voidaan vain liittää halutut komponentit ja ladata koodi STM32-mikrokontrollerille. Nucleo-tuoteperheen kehitysalustoissa on samat liittimet kuin Arduinossa, jolloin Arduinolle tehdyt lisäosat sopivat myös Nucleo-tuoteperheen kehitysalustoihin. (STM32 Nucleo Boards n.d.)

3.8 Arduino

Arduino on avoimeen lähdekoodiin pohjautuva kehitysalusta, joka on suunniteltu halvaksi, monipuoliseksi ja aloittelijaystävälliseksi. Arduino-kehitysalustat tarjoavat monipuoliset liitännät, joiden avulla Arduinoon voidaan liittää esimerkiksi erilaisia nappeja, valoja, antureita ja moottoreita. Arduinon ohjelmointi tapahtuu Arduino-ohjelmointikielellä ja IDE:nä voidaan käyttää Arduino IDE:ä. Vaikka Arduino on suunniteltu aloittelijaystävälliseksi, voidaan sitä käyttää laajasti monimutkaisissakin projekteissa, kuten esimerkiksi tieteellisissä mittalaitteissa. (What is Arduino? n.d.)

Arduino-kehitysalustojen etuna voidaan pitää laajaa käyttäjäkuntaa, jolloin tietoa ja Arduinolla toteutettuja esimerkkiprojekteja löytyy paljon. Tämä auttaa aloittelevaa Arduinon käyttäjää, sillä ongelman kohdatessaan käyttäjä löytää helposti ratkaisun yleisimpiin ongelmiin. (What is Arduino? n.d.)

4 TESTATTAVA LAITE

4.1 Vaatimusmäärittely

Testattavaksi laitteeksi päätettiin luoda ilmanvaihdon ohjausjärjestelmä, jolle määriteltiin vaatimukset, joiden pohjalta tehtävät testit voitiin suunnitella. Ilmanvaihdon ohjausjärjestelmä koostuu kahdesta erillisestä laitteesta. Toinen laite on ohjauspaneeli, jolla laitteen käyttäjä voi määritellä parametrit. Ohjauspaneeli myös mittaa ympäristön lämpötilan. Toinen laite on ilmanvaihdon ohjausjärjestelmä, joka ohjaa ilmanvaihdon tuulettimen nopeutta sekä lämmitystä, jäähdytystä ja tuuletusta. Ohjauspaneeli ja ilmanvaihdon ohjausjärjestelmä kommunikoivat toistensa kanssa UART-sarjaväylän yli.

Ilmanvaihdon ohjausjärjestelmässä tuli olla PWM-ulostulo tuulettimen nopeussignaalille. Tuulettimen nopeus määritettiin PWM-signaalin pulssisuhteella välillä 0–100 %. Tuulettimen nopeus seuraa lineaarisesti PWM-signaalin pulssisuhdetta. PWM-signaalin pulssisuhteen ollessa 0 %, ei tuuletin pyöri. PWM-signaalin pulssisuhteen ollessa 100 %, tuuletin pyörii täysillä. Ilmanvaihdon ohjausjärjestelmässä tuli myös olla ulostulot lämmitykselle, jäähdytykselle ja tuuletukselle. Vain yksi ulostulo kerrallaan voi olla päällä, jolloin esimerkiksi jäähdytys ja lämmitys ei voi olla samaan aikaan päällä. Lämmityksen, jäähdytyksen tai tuuletuksen ulostulon jännitteen ollessa 3.3 volttia, on kyseinen toiminto päällä. Ulostulon jännitteen ollessa 0 volttia, on kyseinen toiminto pois päältä.

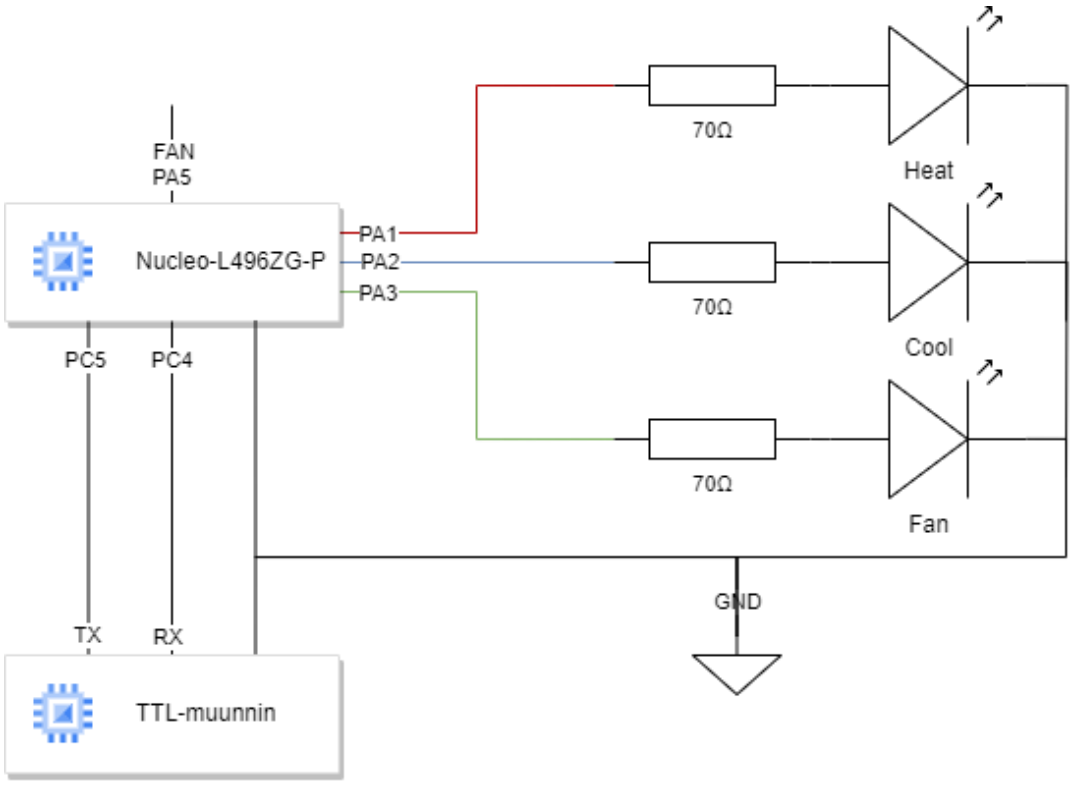
Ohjauspaneeli lähettää UART-sarjaväylän kautta ilmanvaihdon ohjausjärjestelmälle viestin, jossa on parametrit asetetulle tilalle (automaatti tai manuaali), päälle kytketylle toiminnolle (lämmitys, jäähdytys tai tuuletus), tuuletinnopeudelle, ohjauspaneelin mittaamalle lämpötilalle ja tavoitelämpötilalle. Ilmanvaihdon ohjausjärjestelmä lähettää ohjauspaneelille viiden sekunnin välein UART-sarjaväylän kautta viestin, jossa on ilmanvaihdon ohjausjärjestelmän senhetkiset parametrit. Viestissä on parametrit ilmanvaihdon ohjausjärjestelmän asetetulle tilalle (automaatti tai manuaali), päälle kytketylle toiminnolle (lämmitys, jäähdytys tai tuuletus), tuuletinnopeudelle, viimeisimmälle ohjauspaneelilta vastaanotetulle mitatulle lämpötilalle ja tavoitelämpötilalle.

Ilmanvaihdon ohjausjärjestelmässä on kaksi päätilaa automaatti- ja manuaalitila. Automaattitilassa ilmanvaihdon ohjausjärjestelmä laskee ohjauspaneelilta saadun mitatun lämpötilan ja tavoitelämpötilan perusteella tuuletinnopeuden ja kytkee tarpeen mukaan lämmityksen, jäähdytyksen tai tuuletuksen päälle. Manuaalitilassa käyttäjä voi ohjauspaneelin kautta määrittää ilmanvaihdon ohjausjärjestelmälle tuuletinnopeuden ja kytkeä lämmityksen, jäähdytyksen tai tuuletuksen päälle. Ilmanvaihdon ohjausjärjestelmä toimii annetuilla parametreilla, kunnes se saa ohjauspaneelilta uudet parametrit.

Ilmanvaihdon ohjausjärjestelmän tuli myös sietää virheellisiä UART-viestejä. Jos ilmanvaihdon ohjausjärjestelmä sai virheellisen viestin, tuli sen lähettää ohjauspaneelille ”Error”-viesti, jolloin ohjauspaneeli voi lähettää edellisen viestin uudelleen.

4.2 Toteutus

Laite tehtiin vain testausautomaatiojärjestelmän toiminnan esittelyyn, joten laitetta päätettiin yksinkertaistaa. Lämmityksen, jäähdytyksen ja tuuletuksen ohjaus päätettiin korvata ledeillä, joista näkee, onko lämmitys, jäähdytys tai tuuletus päällä. Ohjauspaneelina käytettiin TTL-muunninta, jonka avulla järjestelmää pystyttiin ohjaamaan tietokoneen kautta. Kuviossa 1 testattavan laitteen kytkentä. Ohjauspaneeli lähettää UART-sarjaväylän kautta ilmanvaihdon ohjausjärjestelmälle viestin, jossa on asetettu tila (automaatti tai manuaali), tieto siitä onko lämmitys, jäähdytys tai tuuletus kytketty manuaalisesti päälle, tuulettimen nopeus välillä 0–100 %, mitattu lämpötila ja asetettu tavoitelämpötila.



KUVIO 1. Ilmanvaihdon ohjausjärjestelmän kytkentä.

Automaattitilassa ilmanvaihdon ohjausjärjestelmä säätää tuuletinnopeuden ja kytkee lämmityksen, jäähdytyksen tai tuuletuksen päälle perustuen tavoitelämpötilaan ja mitattuun lämpötilaan. Tuulettimen nopeus lasketaan kahdella kaavalla riippuen siitä, onko mitattu lämpötila suurempi vai pienempi kuin tavoitelämpötila. Jos mitattu lämpötila on pienempi kuin tavoitelämpötila – 1 °C, lasketaan tuulettimen nopeus kaavalla 1

$$\text{tuuletinnopeus} = (\text{tavoitelämpötila} - \text{mitattu lämpötila}) \cdot 10. \quad (1)$$

Jos kaavasta saadaan tuulettimen nopeudeksi yli 100 prosenttia, pakotetaan tuuletinnopeus 100 prosenttiin. Tällöin myös kytketään lämmitys päälle. Jos taas mitattu lämpötila on suurempi kuin tavoitelämpötila + 1 °C, lasketaan tuulettimen nopeus kaavalla 2

$$\text{tuuletinnopeus} = (\text{mitattu lämpötila} - \text{tavoitelämpötila}) \cdot 10. \quad (2)$$

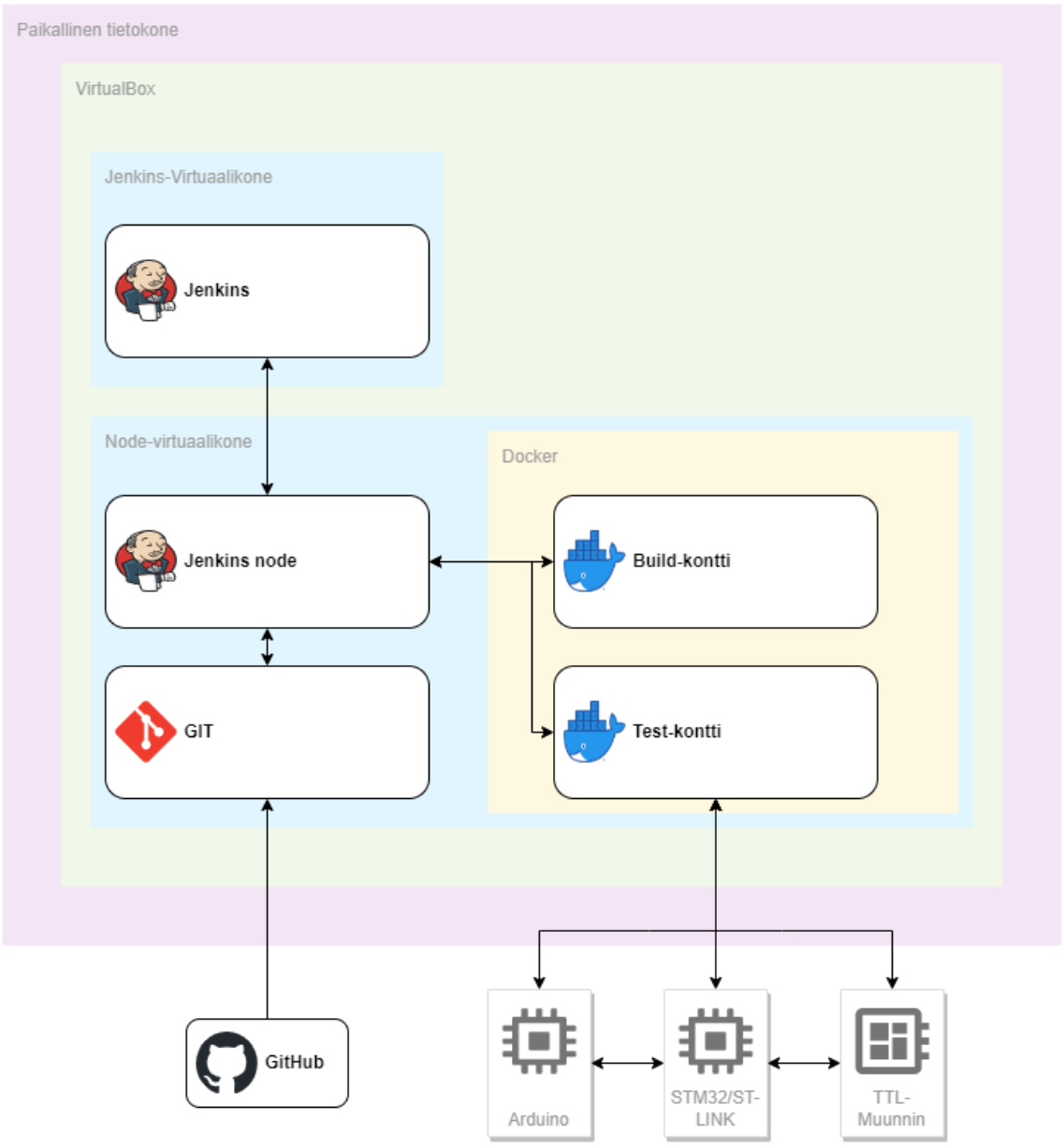
Jos tuuletinnopeudeksi saadaan yli 100 prosenttia, pakotetaan tuuletinnopeus 100 prosenttiin. Tällöin myös kytketään jäähdytys päälle. Jos mitattu lämpötila on tavoitelämpötila ± 1 °C, asetetaan tuulettimen nopeudeksi 20 prosenttia, koska ilmanvaihtoa ei haluttu kokonaan kytkeä pois päältä ja tuuletus pienellä teholla riittää. Tällöin myös kytketään tuuletus päälle ja lämmitys ja jäähdytys pois päältä. Tuulettimen nopeus on 100 prosenttia, jos mitattu lämpötila eroaa tavoitelämpötilasta yli 10 celsiusastetta. Mitatun lämpötilan lähestyessä tavoitelämpötilaa, tuuletinnopeus pienenee.

Manuaalitilassa ohjauspaneeli kertoo ilmanvaihdon ohjausjärjestelmälle tuuletinnopeuden ja kytketäänkö lämmitys, jäähdytys vai tuuletus päälle. Järjestelmä toimii asetetuilla arvoilla, kunnes se saa ohjauspaneelilta uuden käskyn.

5 TESTAUSAUTOMAATIO SULAUTETUN JÄRJESTELMÄN KEHITYKSESSÄ

5.1 Tavoite

Työn tavoitteena oli luoda järjestelmä, jolla voidaan testata STM32-mikrokontrollerilla olevan koodin toimintaa ja varmistua sen oikeasta toiminnasta. Kuviossa 2 on kuvattuna testausautomaatiojärjestelmän toiminta.



KUVIO 2. Testausautomaatiojärjestelmän toiminta.

Paikalliselle tietokoneelle on asennettu VirtualBox-virtualisointiohjelma. VirtualBoxiin on luotu kaksi virtuaalista tietokonetta Jenkins ja Node, joihin molempiin on asennettu käyttöjärjestelmäksi Ubuntu. Työ olisi ollut mahdollista toteuttaa myös siten, että virtuaalikoneet olisi korvattu fyysisillä tietokoneilla. Tässä työssä päädyttiin kuitenkin käyttämään virtuaalikoneita, koska luotu testausautomaatiojärjestelmä ei tule oikeaan käyttöön, vaan sen tarkoitus oli tutustuttaa työn tekijä testausautomaatiojärjestelmän luomiseen.

Jenkins-virtuaalikoneeseen on asennettu Jenkins-serveri, joka ohjaa testausautomaatiojärjestelmän toimintaa ja tarjoaa verkkokäyttöliittymän testausautomaatiojärjestelmän hallintaan.

Node-virtuaalikone suorittaa testit Jenkins-serverin ohjaamana. Node-virtuaalikoneeseen on asennettu Jenkinsin etähallintaohjelma, jolla Jenkins-serveri pystyy hallitsemaan Node-virtuaalikonetta SSH-yhteyden yli ja suorittamaan testit Node-virtuaalikoneella. Git, jolla testattava repositorio voidaan ladata GitHubista ja Docker, jolla luodaan kontit build ja test. Build-kontissa käännetään STM32:lle ladattava koodi binäärimuotoon. Test-kontissa käännetään Arduinolle ladattava koodi ja ladataan koodit Arduinolle ja STM32:lle. Koodien lataamisen jälkeen testataan STM32:lle ladatun koodin toimivuus.

Node-virtuaalikoneeseen on liitetty USB-laitteina Arduino, STM32/ST-LINK ja TTL-muunnin. Arduinoa käytetään testausautomaatiojärjestelmässä lukemaan ja asettamaan STM32:n pinnien tila. Tällöin voidaan varmistua, että STM32:n pinnit ovat niissä tiloissa, missä niiden kuuluisi olla. Arduinolla voidaan myös asettaa STM32:n pinnien tila halutuksi, jos esimerkiksi testauksessa halutaan simuloida napin painallusta, voidaan STM32:n pinni nostaa Arduinolla ylös. Testausautomaatiojärjestelmä ohjaa Arduinoa sarjaväylän yli.

STM32/ST-LINK:iä käytetään koodin lataamiseen STM32:lle. Tässä työssä käytettävässä NUCLEO-L496ZG-P-kehitysalustassa on sisäänrakennettu ST-LINK. TTL-muunninta käytetään kommunikointiin STM32:n kanssa UART-sarjaväylän yli. Tällöin testin aikana STM32:lle voidaan lähettää komentoja ja simuloida UART-sarjaväylän yli kommunikoivaa laitetta.

5.2 Virtuaalikoneiden käyttöönotto

Testausautomaatiojärjestelmää varten luotiin VirtualBoxiin kaksi virtuaalikonetta. Toinen virtuaalikone toimi Jenkins-serverinä ja toinen Jenkinsin nodena. Molempiin virtuaalikoneisiin asennettiin käyttöjärjestelmäksi Ubuntu. Molemmille virtuaalikoneille annettiin käyttöön 4 GB ram-muistia ja 2 prosessoriydintä. Virtuaalikoneille luotiin molemmille oma 50 GB:n virtuaalinen kiintolevy.

Käyttöjärjestelmien asennuksen jälkeen asennettiin molempiin virtuaalikoneisiin viimeisimmät päivitykset. Kun päivitykset olivat asentuneet, molemmat virtuaalikoneet sammutettiin, jotta virtuaalikoneet voitiin yhdistää samaan virtuaaliseen lähiverkkoon ja linkittää Node-virtuaalikoneeseen USB-laitteet. VirtualBoxin asetuksissa luotiin uusi NAT-verkko. Molemmat virtuaalikoneet liitettiin luotuun NAT-verkkoon. Lisäksi Node-virtuaalikoneeseen linkitettiin testausautomaatiojärjestelmän tarvitsemat USB-laitteet, eli Arduino, ST-LINK ja TTL-muunnin. USB-laitteiden linkittäminen mahdollisti niiden käytön virtuaalikoneelta. Tällöin pystyttiin ohjelmoimaan mikrokontrollerit ja testaamaan ohjelmiston toiminta Node-virtuaalikoneella. TTL-muunnin lisättiin Node-virtuaalikoneeseen myös sarjaporttina. Node-virtuaalikoneessa tarkistettiin, että lisätyt laitteet näkyvät. Lähiverkon toimivuus testattiin pingaamalla toista virtuaalikonetta toisella. Node-virtuaalikoneeseen asennettiin myös Git, tämän avulla projektit pystyttiin lataamaan esimerkiksi GitHubista.

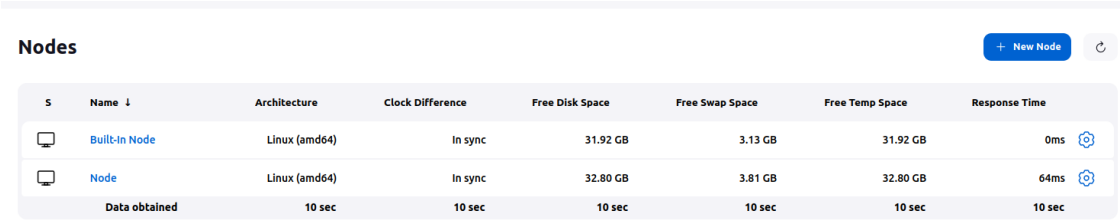
Node-virtuaalikoneeseen tuli myös kytkeä VirtualBoxin asetuksista ”Enable Nested VT-x/AMD-V” -asetus päälle. Tämä asetus mahdollistaa virtualisoinnin virtuaalikoneen sisällä. Esimerkiksi tässä työssä Node-virtuaalikoneen sisällä käytettiin Dockeria, joka tarvitsee virtualisointituen. Oletuksena asetusta ei saanut päälle VirtualBoxin käyttöliittymästä, vaan asetus tuli aktivoida Windowsin komentokehotteesta komennolla ” VBoxManage modifyvm <Virtuaalikoneen nimi> --nested-hw-virt on”. Komennon syöttämisen jälkeen VirtualBoxissa Node-virtuaalikoneen asetuksissa kohta ”Enable Nested VT-x/AMD-V” näkyi aktivoituna. (Nested Virtualization n.d.)

5.3 Jenkinsin käyttöönotto

Jenkins-serverin asentamista varten Jenkins-virtuaalikoneelle tuli asentaa Curl. Curlia käytetään Jenkinsin asennuksessa tarvittavien tiedostojen lataamiseen. Asennuksen jälkeen Jenkins-serverin asennus tapahtui Jenkinsin kotisivuilta löytyvän ohjeen mukaan. Kun Jenkins-serveri oli asennettu ja käynnistetty, mentiin selaimella Jenkins-serverin luomalle sivulle, jossa asennettiin asennusohjelman suosittelemat lisäosat ja luotiin käyttäjä. Jenkinsiin tuli vielä asentaa lisäosa Robot Frameworkille, jotta Robot Frameworkin tuottamat raportit voitiin tallentaa. Tämän jälkeen Jenkins-serverin asennus oli valmis. Jenkinsin asetuksista asetettiin Jenkinsin sisäänrakennetun noden suoritettavien prosessien määrä nollaan, tällöin Jenkins ohjaa kaikki työt suoritettavaksi ulkoiselle nodelle.

Seuraavaksi Jenkins-serveriin yhdistettiin node. Node-virtuaalikoneeseen tuli asentaa Java, jotta Jenkins pystyi käyttämään Node-virtuaalikonetta nodena. Jenkins-serverin verkkosivulla luotiin uusi node, jonka jälkeen määritettiin uudelle nodelle nimi, samanaikaisesti suoritettavien prosessien määrä, tiedostopolku, kuinka usein Jenkins käyttää nodea, yhteyden tiedot ja miten node on saatavissa.

Jenkinsin haluttiin yhdistävän nodeen SSH-yhteyden kautta. Tätä varten tuli luoda Jenkins-virtuaalikoneelle SSH-avainpari. Luodun SSH-avainparin yksityinen avain lisätään Jenkinsiin luotavan noden asetuksiin. Julkinen osa SSH-avainparista lisättiin Node-virtuaalikoneen ”authorized_keys” -kansioon. Turvallisuuden takia Jenkinsin haluttiin varmistavan, että se yhdistää oikeaan tietokoneeseen. Tämä toteutettiin valitsemalla SSH-yhteyden asetuksista ”Host Key Verification Strategy” -vaihtoehto ”Known hosts file Verification Strategy”. Tällöin Jenkins varmistaa SSH-yhteyden luonnin yhteydessä, että Node-virtuaalikoneen SSH-avainparin julkinen osa löytyy Jenkins-virtuaalikoneen ”known_hosts” -kansiosta. Tällöin Node-virtuaalikoneelle tuli luoda SSH-avainpari, jonka julkinen osa lisättiin Jenkins-virtuaalikoneen ”known_hosts” -kansioon. Uuden noden asetusten tallentamisen jälkeen node käynnistettiin ja varmistettiin Jenkinsin käyttöliittymästä, että Jenkins on saanut muodostettua yhteyden nodeen (kuva 1).



KUVA 1. Jenkins on muodostanut yhteyden nodeen onnistuneesti.

5.4 Dockerin käyttöönotto

Node-virtuaalikoneeseen tuli asentaa Docker engine. Jenkinsillä ohjattiin Docker-kontteja, joissa käännettiin mikrokontrollereille ladattavat koodit ja testattiin STM32:lle ladattu koodi. Tällöin voitiin luoda uudet kontit jokaiselle koodin käännökselle ja testaukselle, jolloin voitiin aina aloittaa samasta tilanteesta.

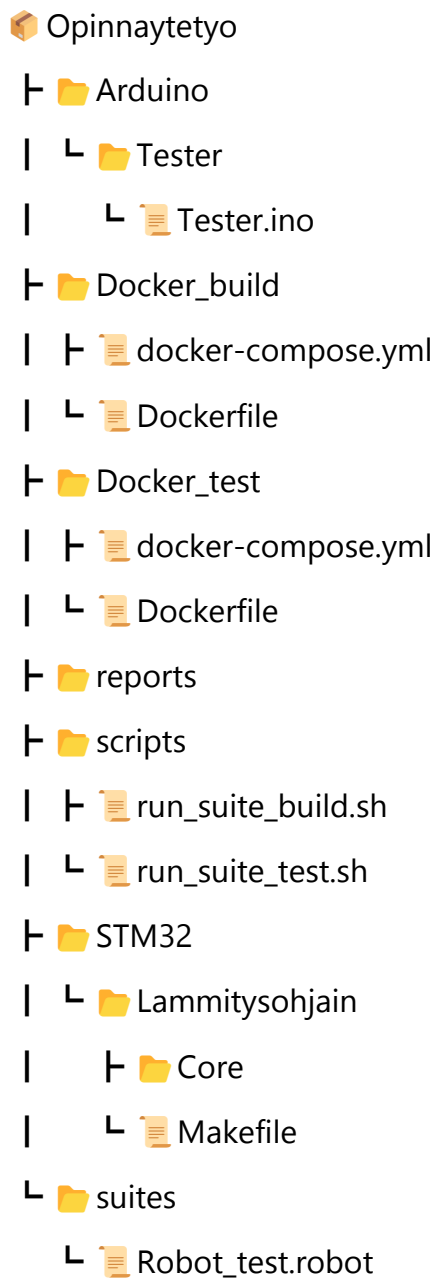
Docker enginen asennus aloitettiin varmistamalla, että Node-virtuaalikoneesta löytyy tuki KVM-virtualisoinnille. KVM-lyhenne tulee sanoista kernel virtual machine, joka mahdollistaa virtualisoinnin ja virtuaalikoneiden luonnin Linuxissa (Kernel Virtual Machine n.d.). Asennus tehtiin Dockerin kotisivuilta löytyvien ohjeiden mukaan. Docker enginestä asennettiin viimeisin versio ja asennuksen valmistuttua Docker enginen toimivuus testattiin ajamalla Dockerin ”hello-world” -image. Node-virtuaalikoneen käyttäjä tuli lisätä docker käyttäjäryhmään, jolloin Docker-kontteja pystyttiin ajamaan ilman sudo-komentoa.

5.5 Projektin luominen ja Github

Projektia varten tuli luoda kansiorakenne, jossa oli ohjelmakoodit, Docker-konttien luontiin tarvittavat määritykset ja Robot Framework -koodi. Työn versionhallintajärjestelmänä päädyttiin käyttämään GitHubia, sen helppokäyttöisyyden ja ilmaisuuden takia. GitHubiin luotiin uusi etärepositorio projektille. GitHubista kloonattiin etärepositorio tietokoneelle, jossa repositorioon lisättiin tarvittavat muutokset.

Kuviossa 3 on esitetty repositorion kansiorakenne. Arduino-kansio sisältää testeriin ladattavan koodin. Docker_build -kansio sisältää koodin kääntämistä varten luotavan Docker-kontin määritykset. Docker_test -kansio sisältää testaamista

varten luotavan Docker-kontin määritykset. Reports-kansioon tallennetaan Robot Frameworkin tuottamat raportit. Scripts-kansiossa on koodit, joilla käynnistetään STM32:lle ladattavan koodin käännös ja Robot Framework -testit. STM32-kansio sisältää testattavan laitteen koodin. Suites-kansiossa on Robot Frameworkin testikoodi.



KUVIO 3. Repositorion kansiorakenne.

Luotu kansiorakenne ladattiin GitHubiin, josta Jenkins pystyi sen myöhemmin lataamaan.

5.6 Docker-kontin luominen

5.6.1 Docker_build -kontti

Docker_build -kontissa testattavalle mikrokontrollerille ladattava ohjelma käännettiin binäärimuotoon. Docker_build -kansiossa olevaan Dockerfileen lisättiin tarvittavat määritykset, jotta testattavan laitteen koodi voitiin kääntää binäärimuotoon. Docker-kontin imageksi valittiin Ubuntu, koska Ubuntuun on mahdollista asentaa build-essential- ja gcc-arm-none-eabi -ohjelmat. Näiden ohjelmien avulla koodi on mahdollista kääntää binäärimuotoon, jotta koodi voidaan ladata mikrokontrollerille. Ohjelmien asennuksen jälkeen Dockerfile suorittaa run_suite_build.sh -tiedoston, jossa tehdään koodin käännös make-komennolla. Make-komento luo STM32_Project -kansion alle build-kansion, johon se tallentaa luodun binääritiedoston.

Docker-compose -tiedostoon määriteltiin käytettävä docker-compose -tiedoston tiedostomuoto, käytettävä image, joka tässä tapauksessa oli Dockerfilen pohjalta luotu kontti, kontille käyttöön annettavan muistin määrä ja kansiot, joihin Docker-kontille annetaan pääsy. Tässä tapauksessa kontille annettiin pääsyoikeus scripts-kansioon, jossa sijaitsee koodin kääntämiseen tarvittavat komennot ja STM32-kansioon, jossa käännettävä koodi sijaitsee.

5.6.2 Docker_test -kontti

Docker_test -kontissa suoritetaan koodien lataaminen mikrokontrollereille ja testataan testattavan mikrokontrollerin toiminta. Docker_test -kansiossa olevaan Dockerfileen määritettiin käytettäväksi imageksi Ubuntu. Dockerfileen määritettiin myös asennettavat ohjelmat. Docker-konttiin asennettiin curl, python3, python3-pip, stlink-tools ja arduino-cli. Pipin avulla asennettiin Pythoniin Robot Framework, pyserial ja robotframework-seriallibrary.

Docker-compose -tiedostoon määritettiin docker-compose -tiedoston tiedostomuoto, käytettävä image, joka tässä tapauksessa oli testaukseen käytettävän Dockerfilen pohjalta luotu kontti, kontille käyttöön annettavan muistin määrä, ympäristömuuttujat, konttiin liitettävät laitteet, tässä tapauksessa Arduino, ST-LINK ja TTL-muunnin sekä kansiot, joihin kontille annettiin käyttöoikeus.

Lisättäviin laitteisiin tuli lisätä ST-LINK:in tiedostopolku, jolloin Dockerilla oli riittävät oikeudet STM32:n ohjelmointiin. Helpoin tapa olisi ollut lisätä docker-compose -tiedostoon ”privileged:true”. Tällöin Dockerilla olisi ollut laajennetut käyttöoikeudet Node-virtuaalikoneeseen ja myös riittävät oikeudet päästä ST-LINK:in tiedostopolkuun (Matthias & Kane 2015,10). Docker_test -kontille ei kuitenkaan haluttu antaa enempää käyttöoikeuksia kuin oli tarpeellista, joten päädyttiin lisäämään ST-LINK:in tiedostopolku laitteena docker-compose -tiedostoon. Käytettävä tiedostopolku oli muotoa ”/dev/bus/usb/väylä/laite”, jossa väylä oli väylän numero, johon ST-LINK oli kytketty ja laite oli ST-LINK:in numero. Jos samassa väylässä olisi ollut muita laitteita, joita ei haluttu lisätä Docker-konttiin olisi docker compose -tiedostoon voitu määritellä tiedostopolku, joka sisältää laitteen numeron. Tässä tapauksessa päädyttiin lisäämään kaikki väylän laitteet, koska ST-LINKIN numero voi vaihtua uudelleenkäynnistyksen tai ST-LINK:in irrotuksen ja uudelleenkytkemisen yhteydessä, eikä väylässä ollut laitteita, joita ei haluttu lisätä Docker-konttiin. Tarkempi ohje ST-LINK:in liittämisestä Dockeriin liitteessä 1.

Projektin kansioista lisättiin suites, jossa on Robot Framework -koodi, scripts, jossa on Robot Framework -koodin käynnistämiseen tarvittava koodi, reports, johon Robot Frameworkin luomat raportit tallennetaan, Arduino, jossa on Arduinolle ladattava koodi ja STM32, jossa on build-kontissa käännetty binäärikoodi, joka ladataan STM32:lle.

5.7 Robot Frameworkin käyttöönotto

Robot Framework ja sen tarvitsemat kirjastot asennettiin Docker_test -kontin luonnin aikana ja Robot_test -testitiedosto luotiin projektin luomisen aikana, joten Robot Frameworkin käyttöönotto oli valmis.

Suites-kansiossa sijaitsevaan Robot_test -tiedostoon lisättiin Robot Frameworkilla tehtävät testit. Testien aikana käännetään Arduino-koodi ja ladataan koodit Arduinolle ja STM32:lle sekä testataan STM32:lle ladatun koodin toimivuus. Arduino-koodi käännettiin Robot Frameworkilla, koska haluttiin kokeilla erilaisia tapoja kääntää koodia. Arduino-koodin olisi voinut kääntää myös samassa kontissa, jossa STM32:lle ladattava koodi käännettiin. Koodien lataaminen mikrokontrollereille päätettiin tehdä Robot Frameworkilla, koska tällöin pystyttiin varmistumaan latauksen onnistumisesta ja keskeyttämään testi, jos koodien lataaminen mikrokontrollereille ei jostain syystä onnistu.

5.8 Jenkins-projektin luonti

Jenkinsiin luotiin uusi projekti. Projektin tyypiksi valittiin ”Freestyle project”, koska tässä työssä tehtävä projekti oli yksinkertainen, eikä siinä ollut montaa eri vaihetta. Jos projektissa olisi monta vaihetta, jossa projekti suoritetaan, olisi kannattanut valita projektityypiksi ”Pipeline”.

Jenkins-projektin asetuksiin lisättiin linkki aikaisemmin luotuun GitHub-etärepositorioon, tällöin Jenkins pystyi lataamaan muutokset GitHubista. Jenkinsillä GitHubiin kirjautumista varten GitHub-käyttäjälle tuli luoda ”personal access token”. Tämä luotiin GitHubin asetuksista. Luotu ”personal access token” kopioitiin Jenkinsiin luotavan GitHub käyttäjän salasanaksi.

Jenkinsiin luotavan projektin asetuksista valittiin, että Jenkins säilyttää vain 10 viimeisintä buildia. Tällä voidaan säästää tietokoneen kiintolevytilaa.

Jenkins-projektin vaiheiksi luotiin kaksi ”Execute shell” -vaihetta. Ensimmäisessä vaiheessa annettiin Jenkinsille suoritusoikeus tiedostoon run_suite_build, luotiin Docker-kontti koodin kääntämiseen tehdyn Dockerfilen pohjalta, käynnistettiin luotu kontti ja liitettiin tarvittavat kansiot konttiin docker compose up -komennolla. (kuva 2)

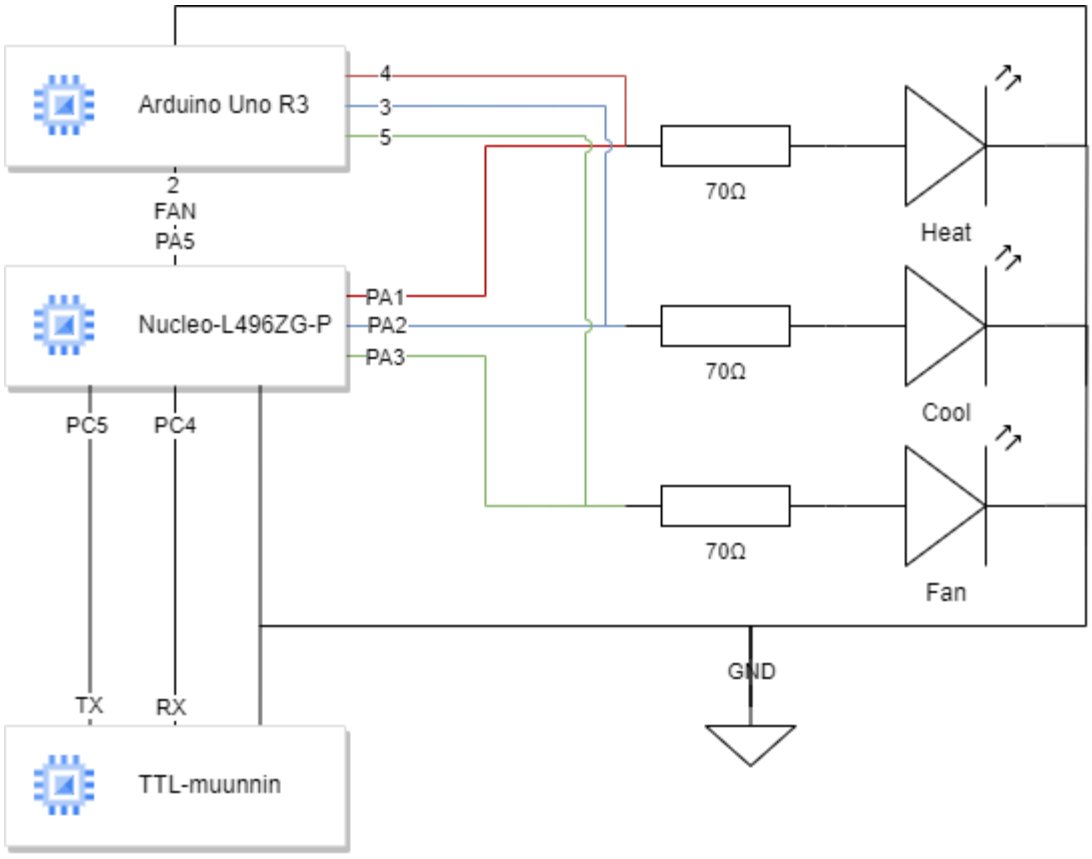
Jenkins-projektin toisessa vaiheessa annettiin Jenkinsille suoritusoikeus run_suite_test -tiedostoon, luotiin Docker-kontti testaamiseen tehdyn Dockerfilen pohjalta, käynnistettiin luotu kontti ja liitettiin tarvittavat kansiot ja USB-laitteet konttiin docker compose up -komennolla. (kuva 2)

KUVA 2. Jenkins-projektiin luodut vaiheet.

Jenkinsiin luotavan projektin ”Post-build Actions” -kohtaan määritettiin Robot Framework -testien julkaiseminen. Tällöin Robot Framework -testien tulokset julkaistaan buildin valmistuttua.

5.9 Testeri

Testerinä käytettiin Arduino-kehitysalustaa, sen helppokäyttöisyyden ja helpon ohjelmoinnin takia. Arduinon tehtävä testausautomaatiojärjestelmässä oli varmistaa, että STM32:n ulostulot olivat kytkeytyneet oikein. Arduinolla mitattiin STM32:n lämmitys-, jäähdytys- ja puhallinpinnien tilaa, sekä tuulettimen ulostulon PWM-signaalin pulssisuhdetta. Arduino laski PWM-pulssisuhteen STM32:n lähettämän PWM-signaalin päälläolo- ja poissaoloajan perusteella. Robot Frameworkista lähettiin Arduinolle sarjaväylän kautta status-komento, johon Arduino vastasi viestillä, jossa oli PWM-signaalin pulssisuhde sekä tuuletuksen, jäähdytyksen ja lämmityksen tila. Kuviossa 4 testauksessa käytetty kytkentä.



KUVIO 4. Testauskytkentä

5.10 Robot Framework -testit

Testien alussa käännetään Arduino-koodi, tarkistetaan, onko Arduino ja STM32 kytkettynä ja ladataan käännetty koodi Arduinolle ja build-kontissa käännetty koodi STM32:lle.

Koodien lataamisen jälkeen testataan STM32:lle ladatun koodin toimivuus. Ensimmäisessä testissä testataan koodin toimintaa automaattitilassa, kun sille annetaan satunnaisia mitattuja lämpötiloja ja tavoitelämpötiloja. Mitattu lämpötila ja tavoitelämpötila arvotaan jokaisella suorituskerralla. Lämpötilat voivat olla välillä 0–40 celsiusastetta. Testi toistetaan 10 kertaa, jotta voidaan varmistua laitteen toimivuudesta useilla eri mitatuilla lämpötiloilla ja tavoitelämpötiloilla. Robot Frameworkkiin luotiin STM32_auto-avainsana, jolla automaattitilaa voitiin testata. STM32_auto-avainsanalle syötettiin argumentteina mitattu lämpötila ja tavoitelämpötila. STM32_auto-avainsana lähettää STM32:lle viestin, jossa asetetaan mitattu lämpötila ja tavoitelämpötila. Tämän jälkeen odotetaan STM32:n vastausta, jonka avulla pystytään varmistumaan, että lämpötilat ovat oikein. Tämän jälkeen Arduinolle lähetetään status-viesti. Arduino vastaa tiedolla STM32:n pinnien tilasta ja tiedolla tuulettimen PWM-signaalin pulssisuhteesta prosentteina. Tämän jälkeen lasketaan tuuletinnopeus prosentteina STM32_auto-avainsanalle syötetystä mitatusta lämpötilasta ja tavoitelämpötilasta. Tuuletinnopeuden pitäisi olla sama kuin STM32:n laskema tuuletinnopeus. Lopuksi STM32_auto-avainsana tarkistaa, että STM32 on automaattitilassa, laskettu tuuletinnopeus vastaa STM32:n laskemaa tuuletinnopeutta ja Arduinon mittaama tuuletinnopeuden PWM-signaalin pulssisuhde on laskettu tuuletinnopeus ± 5 %. Arduinon PWM-signaalin pulssisuhteen mittauksessa sallitaan viiden prosentin virhe, sillä tuuletinnopeudessa riittää, että tuuletinnopeus on lähellä laskettua nopeutta.

Toisessa testissä testataan laitteen toimintaa, kun mitattu lämpötila on pienempi kuin tavoitelämpötila ja mitattu lämpötila lähenee tavoitelämpötilaa. Tavoitelämpötilaksi asetettiin 20 celsiusastetta ja mitatuille lämpötiloille luotiin lista, joka käydään testin aikana läpi. Testissä käytettiin samaa STM32_auto-avainsanaa kuin ensimmäisessä testissä.

Kolmannessa testissä testattiin STM32:n toimintaa, kun mitattu lämpötila on suurempi kuin tavoitelämpötila. Tavoitelämpötilaksi asetettiin 20 celsiusastetta ja mitattu lämpötila toteutettiin for-loopilla, jossa mitattu lämpötila on ensin 50 celsiusastetta ja lämpötila laskee yhden celsiusasteen jokaisella for-loopin kierroksella, kunnes lämpötila on 15 celsiusastetta. Tällöin jäähdytyksen tulee olla päällä, kunnes mitattu lämpötila saavuttaa 21 celsiusastetta. Mitatun lämpötilan ollessa 19–21 celsiusastetta jäähdytyksen ja lämmityksen tulee olla pois päältä ja tuulettimen nopeuden tulee olla 20 prosenttia. Mitatun lämpötilan laskiessa alle 19 celsiusasteen lämmityksen tulee kytkeytyä päälle ja tuuletinnopeuden kasvaa, mitatun lämpötilan pienentyessä. Viestien lähetykseen ja toiminnan varmistamiseen käytettiin samaa STM32_auto-avainsanaa kuin kahdessa aikaisemmassa testissä.

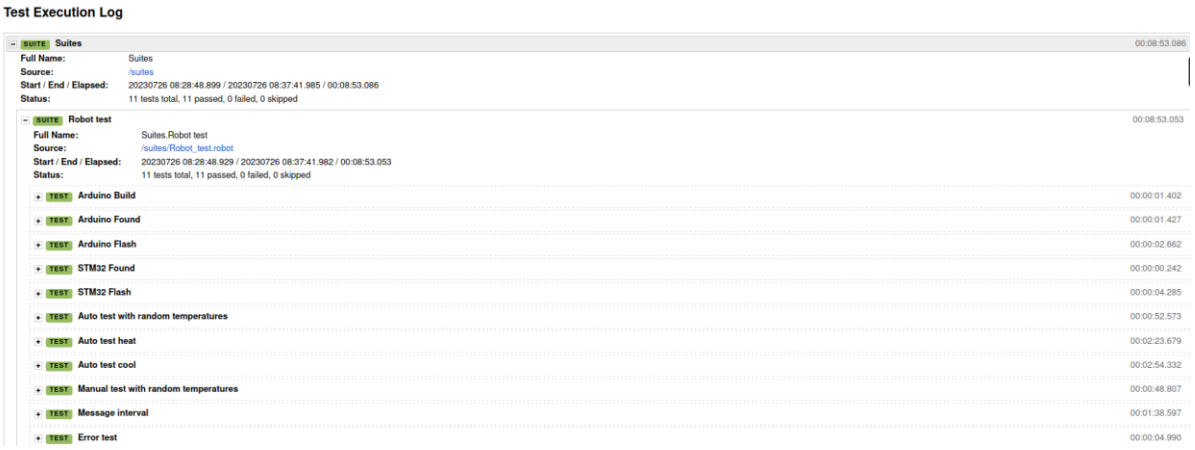
Neljännessä testissä testataan STM32:n manuaalitilaa satunnaisilla mitatuilla lämpötiloilla, tavoitelämpötiloilla, tuuletinnopeudella ja tiloilla. Kaikki arvot arvotaan jokaisella suorituskerralla. Lämpötilat voivat olla välillä 0–40 celsiusastetta. Tuuletinnopeus voi olla välillä 0–100 %. Tila voi olla fan, heat tai cool. Testi toistetaan 10 kertaa, jotta voidaan varmistua laitteen toiminnasta eri parametreillä. Testiä varten luotiin avainsana STM32_manual, johon syötetään argumenteiksi lämmityksen ja jäähdytyksen tila, tuuletinnopeus, mitattu lämpötila ja tavoitelämpötila. STM32_manual -avainsana lähettää STM32:lle viestin, jossa on argumenteissa annetut tiedot. Tämän jälkeen odotetaan STM32:n vastausta, jolla voidaan varmistaa, että asetetut parametrit ovat oikein. Arduinolle lähetetään statusviesti, jolloin Arduino vastaa viestillä, jossa on tieto STM32:n pinnien tilasta ja tieto tuulettimen PWM-pulssisuhteesta prosentteina. Arduinon lähettämästä viestistä tarkistetaan, että tuuletin, lämmitys ja jäähdytys pinnit ovat kytkeytyneet oikein ja että tuuletinnopeuden mitattu PWM-signaalin pulssisuhde vastaa asetettua pulssisuhdetta. Mitattu tuuletinnopeus voi olla asetettu tuuletinnopeus ± 5 %, eli mitatussa tuuletinnopeudessa sallitaan viiden prosentin virhe, sillä tuulettimen tarkalla nopeudella ei ole suurta merkitystä. Tärkeämpää on, että tuuletinnopeus on lähellä asetettua tuuletinnopeutta.

Viidennessä testissä mitataan STM32:n ajastinkeskeytyksen toimintaa. Testi odottaa, että STM32 lähettää viestin, jossa on sillä hetkellä asetetut arvot. Viestin vastaanottamisen jälkeen testi tallentaa ajan, jolloin viesti vastaanotettiin. Testi odottaa seuraavaa viestiä ja tallentaa ajan, jolloin toinen viesti vastaanotettiin.

Tällöin voidaan laskea kahden viestin välinen aika. Ajastinkeskeytys on asetettu viiteen sekuntiin. Testi kuitenkin hyväksyy viestien väliseksi ajaksi 4.5–5.5 sekuntia. Vaihteluväli hyväksytään, sillä viestien välinen aika ei ole kriittinen järjestelmän toiminnan kannalta. Viestien välisen ajan mittaus toistetaan 10 kertaa, jotta voidaan varmistua, että useampi viesti tulee hyväksyttävän aikaikkunan sisällä.

Kuudennessa testissä STM32:lle lähetetään virheellinen komento ja testataan vastaako STM32 viestillä ”Error”. Tällöin tiedetään, että annetussa komennossa oli virhe ja komento voitaisiin lähettää uudestaan.

Suoritettujen testien tuloksista Robot Framework koosti raportin, josta testien onnistuminen voitiin todeta helposti. Testiraporttia pystyi tarkastelemaan Jenkinsin käyttöliittymästä. Jos testi epäonnistui testiraportista, voitiin tutkia mikä aiheutti testin epäonnistumisen. Kuvassa 3 Robot Frameworkin testiraportti Jenkinsin käyttöliittymässä.



Kuva 3. Robot Frameworkilla tehtyjen testien raportti

6 POHDINTA

Opinnäytetyön tuloksena saatiin testausautomaatiojärjestelmä, jolla voidaan testata STM32-mikrokontrolleripohjaisen ilmanvaihdon ohjausjärjestelmän toiminta. Ilmanvaihdon ohjausjärjestelmästä testattiin manuaali- ja automaattitilojen toiminta, lämmityksen ja jäähdytyksen toiminta eri lämpötiloilla ja eri tavoitelämpötiloilla sekä tuuletinnopeuden pieneneminen mitatun lämpötilan lähestyessä tavoitelämpötilaa. Ilmanvaihdon ohjausjärjestelmästä testattiin myös ajastinkeskeytyksen toiminta mittaamalla UART-sarjaväylän kautta lähetettyjen viestien välistä aikaa sekä toiminta, jos ilmanvaihdon ohjausjärjestelmälle lähetetään virheellinen viesti. Testausautomaatiojärjestelmästä saatiin toimimaan tarkoituksenmukaisesti ja luotettavasti. Työn ohessa tuotettu ohje STM32-mikrokontrollerin ohjelmoimisesta Dockerin ja Robot Frameworkin avulla saatiin valmiiksi. Ohjeessa kerrotaan vaihe vaiheelta, kuinka STM32-mikrokontrolleri voidaan liittää Jenkinsin ohjaamaan Docker-konttiin ja kuinka ladattava koodi voidaan kääntää ja ladata mikrokontrollerille. Ohjeen avulla henkilö, jolla on perustiedot Jenkinsin ja Dockerin käytöstä, osaa liittää STM32:n Jenkinsin ohjaamaan Docker-konttiin, sekä kääntää ja ladata koodin STM32:lle. Opinnäytetyö toteutui hieman suunniteltua aikataulua nopeammin.

Opinnäytetyön aihe oli haastava, koska minulla ei ollut aikaisempaa kokemusta testausautomaatiosta. En ollut aiemmin käyttänyt Jenkinsiä enkä Robot Frameworkia. Dockerin käytöstä minulla oli hieman aikaisempaa kokemusta. Jenkinsin ja Robot Frameworkin perustoiminnot kuitenkin oppi nopeasti ja pystyin toteuttamaan yksinkertaisia testejä. Opinnäytetyön loppupuolella myös monimutkaisemmat testit onnistuivat. Opinnäytetyön aiheen haastavuus oli odotettavissa, sillä opinnäytetyön tavoitteena oli opiskella testausautomaatiojärjestelmän toteuttamista, jotta voin hyödyntää opinnäytetyössä oppimiani taitoja myöhemmin työelämässä. Opinnäytetyön teoriaosuuden kirjoittaminen kuitenkin toi jonkinlaisen ymmärryksen järjestelmien toimintaan ja se auttoi paljon opinnäytetyön toteutuksessa.

Opinnäytetyön toteutusta olisi helpottanut, jos olisi ollut olemassa jokin valmis testattava laite. Tällaista testattavaa laitetta, jota olisi voinut esitellä julkisesti, ei kuitenkaan löytynyt. Päädyin keksimään ja toteuttamaan testattavan laitteen itse. Testien suunnittelu ja toteuttaminen olisi voinut olla mielenkiintoisempaa, jos olisin saanut suunnitella testit jonkun muun suunnittelemalle laitteelle. Testattava laite jäi myös hieman yksinkertaiseksi, koska laitteen toteuttamiseen ei ollut kovin paljoa aikaa. Mikäli testattavasta laitteesta olisi halunnut tehdä monimutkaisemman, opinnäytetyö ei olisi valmistunut aikataulussa. Tästä johtuen testitkin jäivät hieman yksinkertaisiksi.

Työssä olisi ollut mielenkiintoista päästä testaamaan jotakin monimutkaisempaa järjestelmää Jenkinsin pipeline -projektilla, jolloin olisi voinut luoda monivaiheisempia töitä.

Toteutus onnistui omasta mielestäni hyvin suunnitellussa aikataulussa. Opin paljon uutta testausautomaatiojärjestelmän toteutuksesta ja osaan nyt toteuttaa yksinkertaisen testausautomaatiojärjestelmän. Tästä on hyvä jatkaa testausautomaatiojärjestelmien opiskelua. Huomasin myös, että testaaminen on todella mielenkiintoista ja testausautomaatiojärjestelmien suunnittelu on todella mielekästä työtä.

LÄHTEET

Bisht, S. 2013. Robot framework test automation. Create test suites and automated acceptance tests from scratch. E-kirja. 1st edition. Birmingham: Packt Publishing. Viitattu 26.5.2023. Vaatii käyttöoikeuden.
https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9910651569605973

Chacon, S. & Straub, B. 2023. Pro Git. E-kirja. Apress. Viitattu 12.5.2023
https://git-scm.com/book/en/v2

Docker objects. n.d. Docker. Verkkosivu. Viitattu 21.5.2023.
https://docs.docker.com/get-started/overview/#docker-objects

Free and Open Source. n.d. git. Verkkosivu. Viitattu 12.5.2023.
https://git-scm.com/about/free-and-open-source

Installing Jenkins. n.d. Jenkins. Verkkosivu. Viitattu 18.5.2023.
https://www.jenkins.io/doc/book/installing/

Introduction. n.d. Robot Framework Foundation. Verkkosivu. Viitattu 25.5.2023.
https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#introduction

Jenkins User Documentation. n.d. Jenkins. Verkkosivu. Viitattu 18.5.2023.
https://www.jenkins.io/doc/

Kernel Virtual Machine. n.d. KVM. Verkkosivu. Viitattu 24.5.2023.
https://www.linux-kvm.org/page/Main_Page

Managing Nodes. n.d. Jenkins. Verkkosivu. Viitattu 18.5.2023.
https://www.jenkins.io/doc/book/managing/nodes/

Matthias, K. & Kane, S. P. 2015. Docker. Up & Running. E-kirja. O'Reilly Media. Viitattu 7.6.2023. Vaatii käyttöoikeuden.
https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_proquest_ebookcentral_EBC4442189

Nested Virtualization. n.d. Oracle. Verkkosivu. Viitattu 24.5.2023.
https://docs.oracle.com/en/virtualization/virtualbox/6.0/admin/nested-virt.html

Nixon, R. 2010. Ubuntu. up and running. E-kirja. First edition. Beijing: O'Reilly. Viitattu 29.5.2023. Vaatii käyttöoikeuden.
https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9910690004305973

Oracle. n.d. Oracle VM VirtualBox. User Manual. E-kirja. Viitattu 15.5.2023.
https://download.virtualbox.org/virtualbox/7.0.8/UserManual.pdf

Overview. n.d. Docker. Verkkosivu. Viitattu 21.5.2023.
https://docs.docker.com/get-started/

Pipeline. n.d. Jenkins. Verkkosivu. Viitattu 18.5.2023.
https://www.jenkins.io/doc/book/pipeline/

Sambamurthy, M. 2023. Test Automation Engineering Handbook. Learn and implement techniques for building robust test automation frameworks. E-kirja. 1st ed. Birmingham, England: Packt Publishing. Viitattu 2.6.2023. Vaatii käyttöoikeuden.
https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9911371312905973

Small and Fast. n.d. git. Verkkosivu. Viitattu 12.5.2023.
https://git-scm.com/about/small-and-fast

Spinellis, D. 2012 Git. E-kirja. IEEE software. Viitattu 29.5.2023. Vaatii käyttöoikeuden. https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_gale_infotracacademiconefile_A300360591

STM32 32-bit Arm Cortex MCUs. n.d. STMicroelectronics. Verkkosivu. Viitattu 19.9.2023.
https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

STM32 Nucleo Boards. n.d. STMicroelectronics. Verkkosivu. Viitattu 20.9.2023.
https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html

Taulli, T. 2020. The Robotic Process Automation Handbook. A Guide to Implementing RPA Systems. E-kirja. 1st ed. Berkeley, CA: Apress. Viitattu 26.5.2023. Vaatii käyttöoikeuden.
https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9911130235305973

What is Arduino?. n.d. Arduino. Verkkosivu. Viitattu 19.9.2023.
https://www.arduino.cc/en/Guide/Introduction

LIITTEET

Liite 1. Ohje ST-LINK:in liittämisestä Docker-konttiin

1 (20)

29-Aug-23

Compiling and Flashing STM32 Code Using Docker

huld

(jatkuu)

Contents

huld

1 Purpose of these instructions

The purpose of these instructions is to demonstrate how to compile code in Docker container and how to flash compiled code to STM32 using Docker container. In these instructions we are using Jenkins to run Docker containers and Robot Framework to flash code to STM32.

2 Requirements

You need to have few things ready for these instructions. There are a few things to consider before using these instructions.

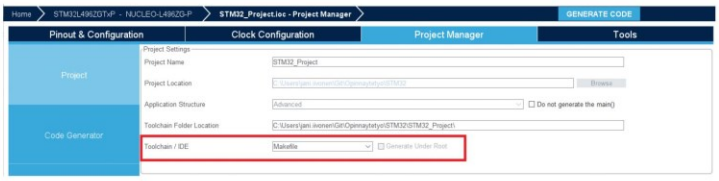
- Basic knowledge about Docker, Jenkins, GitHub and Robot Framework.
- Jenkins is installed and working.
  - How to install Jenkins. https://www.jenkins.io/doc/book/installing/linux/
- Docker-engine is installed and working on the Jenkins node. You should be able to run Docker containers without sudo access.
  - How to install Docker-engine. https://docs.docker.com/engine/install/ubuntu/
  - How to run Docker containers without sudo. https://docs.docker.com/engine/install/linux-postinstall/
  - Add jenkins user to docker user group with command sudo usermod -aG docker jenkins
- You have the Jenkins project. In this case we use a Jenkins Freestyle project.
- Ubuntu is used for these instructions some commands may vary with different operating system.

3 Create project

- Create the file structure for the project. Here is an example of what the file structure could look like. In this example we use GitHub to save this project and share it with Jenkins.

```
Opinnaytetyo
├ Docker_build
| ├ docker-compose.yml
| └ Dockerfile
├ Docker_test
| ├ docker-compose.yml
| └ Dockerfile
├ reports
├ scripts
| ├ run_suite_build.sh
| └ run_suite_test.sh
├ STM32
| └ STM32_Project
| | ├ .mxproject
| | ├ Makefile
| | ├ startup_stm32l496xx.s
| | ├ STM32L496ZGTxP_FLASH.ld
| | └ STM32_Project.ioc
├ suites
| └ Robot_test.robot
└ README.md
```

- Docker_build
  - Contains files needed to build and run Docker container to compile code.
- Docker_test
  - Contains files needed to build and run Docker container to flash code to STM32.
- Reports
  - Robot Framework saves test results here.
- Scripts
  - Contains startup scripts for the container.
- STM32
  - Contains STM32 project that is compiled and flashed to STM32.
  - You must save your STM32 project in Makefile format.
    - For example, if you are using STM32CubeMX in Project Manage tab select Toolchain/IDE to be Makefile



- Suites
  - Contains Robot Framework code.

4 Compiling code in Docker

Once the file structure is created, the next step is to create a Docker container where the STM32 project will be compiled.

4.1 Dockerfile

Add following commands to the Dockerfile located in the Docker_build folder.

```
#Download latest ubuntu base image
FROM ubuntu:latest
#Install needed applications to compile STM32 project
RUN apt-get update && \
        apt-get clean && \
        apt-get install -y \
        build-essential \
        gcc-arm-none-eabi

#Default script to run
CMD ["/scripts/run_suite_build.sh"]
```

4.2 Run_suite_build.sh

Add following commands to the run_suite_build.sh file. Make command compiles the STM32 project from makefile.

```
#!/usr/bin/env bash
set -e
#Compile STM32 project. /STM32/STM32_Project/ is the path to Makefile.
CMD="make -C /STM32/STM32_Project/"

echo ${CMD}

``${CMD}``
```

4.3 Docker-compose.yml

Add the following commands to the docker-compose.yml file located in the Docker_build folder.

```
version: '3.3'
services:
    build:
        network_mode: none
        image: build #Image name. This is specified when building docker
container from Dockerfile
        shm_size: "256M" #Set container partition size

        volumes: [
           "$PWD/scripts:/scripts", #Folder where scripts are located.
           "$PWD/STM32:/STM32" #Folder where STM32 project is located.
        ]
```

Now the building container is ready, and you can push file structure and codes to your GitHub repository.

4.4 Jenkins commands

In your Jenkins project, configure Jenkins to pull code from the GitHub repository where you pushed file the structure and codes.

Create new execute shell build step in your Jenkins project. Add following commands to build step.

```
chmod +x ./scripts/run_suite_build.sh
docker build -f ./Docker_build/Dockerfile -t build .
docker run --rm build bash
docker compose -f ./Docker_build/docker-compose.yml up
```

```
chmod +x ./scripts/run_suite_build.sh
```

Gives Docker execution rights to run_suite_build script.

```
docker build -f ./Docker_build/Dockerfile -t build .
```

Docker build command builds container from Dockerfile. -f flag specifies file path to Dockerfile. -t flag specifies name of the built image.

```
docker run --rm build bash
```

Docker run command runs selected image. --rm flag removes container when it exits. Build is the name of the image that will be run.

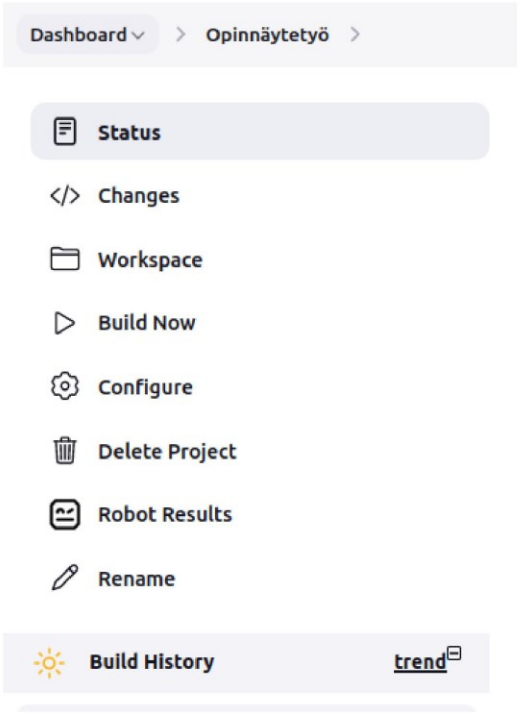
Save your Jenkins project modifications and go to Jenkins dashboard.

huld

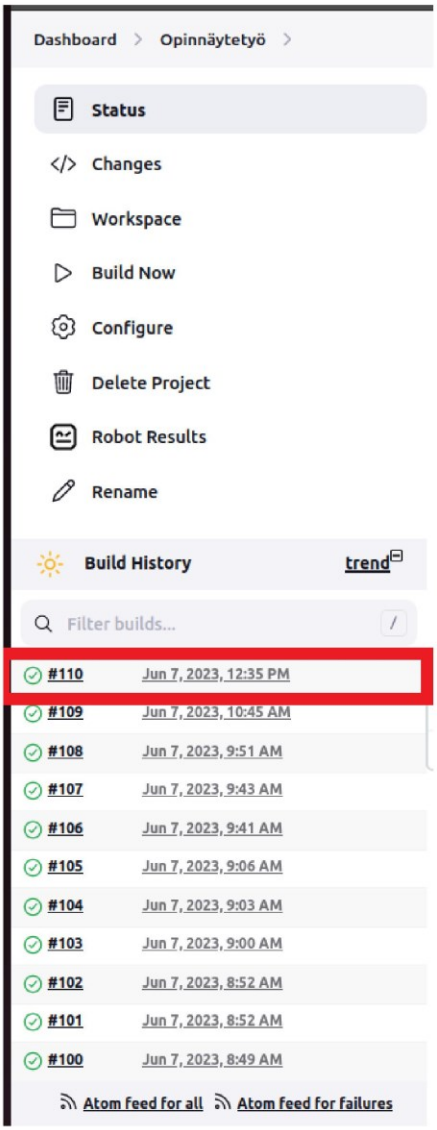
4.5 Test compiling code

Click on your project name and you should see the project window.

Click “build now”.

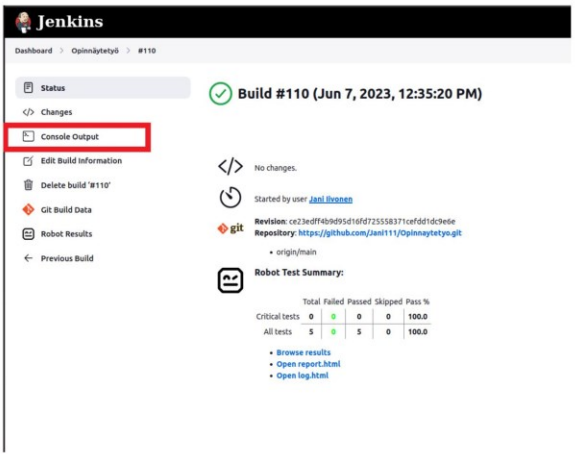


Click on the number of the latest finished build.



You should see a page like this.

Click on console output.



Scroll to the end of the output.

The end of the output should be like this.

Now the compiled binary is located in the STM32 project folder within the build folder.

5 Flashing compiled code to STM32

5.1 Dockerfile

Add the following commands to your Dockerfile located in the Docker_test folder.

```
#Download latest ubuntu base image
FROM ubuntu:latest

#Install needed programs to flash and test STM32
RUN apt-get update && \
        apt-get clean && \
        apt-get install -y \
            python3 \
            python3-pip \
            stlink-tools \
            usbutils


#Install and upgrade pip
RUN python3 -m pip install --upgrade pip setuptools wheel

#Install Robot framework
RUN python3 -m pip install robotframework

#Default script to run
CMD ["/scripts/run_suite_test.sh"]
```

5.2 Run_suite_test.sh

Add the following commands to the run_suite_test.sh. This shell script launches Robot Framework code that will flash compiled STM32 project to STM32.

```
#!/usr/bin/env bash
set -e

# Run an individual test suite if the TEST_SUITE environmental variable
is set.
if [ -z "$TEST_SUITE" ]; then
    TEST_SUITE=""
fi
#Launch the Robot Framework code that will flash the compiled STM32 code
to the STM32
CMD="robot --console verbose --outputdir /reports /suites/$TEST_SUITE"

echo ${CMD}

``${CMD}``
```

5.3 Docker-compose

Now you can create a docker-compose file for flashing STM32. There are two ways to add ST-LINK to Docker. Option one is to run Docker container in privileged mode. This gives basically root access to your operating system, so it is not very safe when we only want to add ST-LINK to the container. Option two is to add ST-LINK's path to docker compose file. If you want to easily add ST-LINK to the container and don't mind giving the Docker container root privileges, move to section 5.3.1. If you don't want to give Docker container root privileges, move to section 5.3.2.

5.3.1 Option 1

Open terminal on your computer where you have your ST-LINK connected.

Run the following command.

```
ls -l /dev/serial/by-id/
```

Output should be like this.

Find the row that says ST-LINK. In this case, the ST-LINK device is /ttyACM0

Add the following commands to your docker-compose file located in the Docker_test folder. Change the device to match your ST-LINK device. For example, if your ST-LINK device is /ttyACM1 edit /dev/ttyACM0:/dev/ttyACM0 to /dev/ttyACM1:/dev/ttyACM1

```
version: '3.3'
services:
    test:
        network_mode: none
        image: test  #Image name. This is specified when building
docker container from Dockerfile
        shm_size: "256M" #Set container partition size

        privileged: true #Set container to run in privileged mode
        devices: #Add necessary devices.
          - /dev/ttyACM0:/dev/ttyACM0


        volumes: [ #Add needed folders
           "$PWD/suites:/suites",
           "$PWD/scripts:/scripts",
           "$PWD/reports:/reports",
           "$PWD/Arduino:/Arduino",
           "$PWD/STM32:/STM32"
        ]
```

5.3.2 Option 2

Open a terminal on your computer where you have your ST-LINK connected.

Run the following command.

```
ls -l /dev/serial/by-id/
```

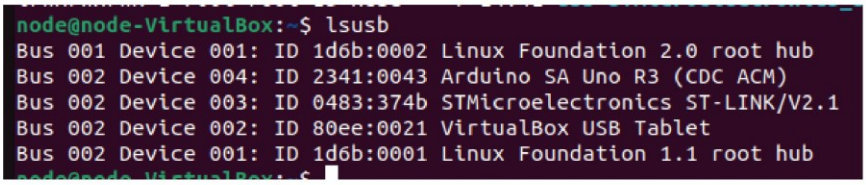
Output should be like this.

Find the row that says ST-LINK. In this case, the ST-LINK device is /ttyACM0

Then run the following command.

```
lsusb
```

Output should be like this.



Find the row that says ST-LINK.

Add the following commands to your docker-compose file located in the Docker_test folder.

Change the device to match your ST-LINK device. For example, if your ST-LINK is /ttyACM1 edit /dev/ttyACM0:/dev/ttyACM0 to /dev/ttyACM1:/dev/ttyACM1.

Also edit the /dev/bus/usb/002:/dev/bus/usb/002 line to match your ST-LINK bus and device. The path for ST-LINK is given in the form of

/dev/bus/usb/ST-LINK_bus/ST-LINK_device:/dev/bus/usb/ST-LINK_bus/ST-LINK_device

where ST-LINK_bus is the bus number that the lsusb command gives for the ST-LINK and ST-LINK_device is the device number that the lsusb command gives for the ST-LINK.

For example, if the lsusb command gives to you ST-LINK Bus 001 Device 001 the

“/dev/bus/usb/ST-LINK_bus/ ST-LINK_device: /dev/bus/usb/ST-LINK_bus/ ST-LINK_device” line should be /dev/bus/usb/001/001:/dev/bus/usb/001/001

You should only specify the device number if the bus has devices that you don't want to add to your Docker container. The device number for ST-LINK can change when you reattach ST-LINK or reboot your computer.

If you don't have devices in the same bus as the ST-LINK that you don't want to add to the Docker container, you could only specify the bus. For example, if the lsusb command says that the ST-LINK is in bus 001, you would give the following command

- /dev/bus/usb/001:/dev/bus/usb/001

In this example, ST-LINK is in bus 002, and we don't have any devices in bus 002 that we don't want to add to the Docker container, so we only specify the bus.

```
version: '3.3'
services:
    test:
        network_mode: none
        image: test  #Image name. This is specified when building
docker container from Dockerfile
        shm_size: "256M" #Set container partition size

        devices: #Add necessary devices.
          - /dev/ttyACM0:/dev/ttyACM0
          - /dev/bus/usb/002:/dev/bus/usb/002

        volumes: [ #Add needed folders
           "$PWD/suites:/suites",
           "$PWD/scripts:/scripts",
           "$PWD/reports:/reports",
           "$PWD/Arduino:/Arduino",
           "$PWD/STM32:/STM32"
        ]
```

5.4 Robot Framework

Add the following commands to the Robot_test.robot file located in the suites folder.

```
*** Settings ***
Library  Process

*** Test Cases ***
STM32 Found
    ${stmproberesult} =    Run Process  st-info  --probe
    Log    ${stmproberesult.stdout}
    Log    ${stmproberesult.stderr}
    Should Not Contain    ${stmproberesult.stdout}    Found 0 stlink
programmers
    Should Be Empty   ${stmproberesult.stderr}

#Flash STM32 with compiled binary. Pass test if output contains Flash
written and verified! jolly good!
STM32 Flash
    ${stmflashresult} =    Run Process  st-flash  --connect-under-
reset  write  /STM32/STM32_Project/build/STM32_Project.bin  0x8000000
    Log    ${stmflashresult.stdout}
    Log    ${stmflashresult.stderr}
    Should Contain    ${stmflashresult.stderr}   Flash written and
verified! jolly good!
```

STM32 Found test checks that ST-LINK can be found with the st-info -probe command. Test passes if ST-LINK is present and error variable is empty.

```
STM32 Found
    ${stmproberesult} =    Run Process  st-info  --probe
    Log    ${stmproberesult.stdout}
    Log    ${stmproberesult.stderr}
    Should Not Contain    ${stmproberesult.stdout}    Found 0 stlink
programmers
    Should Be Empty   ${stmproberesult.stderr}
```

STM32 Flash- test flashes compiled code to STM32 with st-flash command. --connect-under-reset flag is used because without it STM32 sometimes halts after flash. /STM32/STM32_Project/build/STM32_Project.bin specifies path to the compiled code binary.

Test passes if flash command output contains “Flash written and verified! jolly good!”.

```
STM32 Flash
    ${stmflashresult} =    Run Process  st-flash  --connect-under-
reset  write  /STM32/STM32_Project/build/STM32_Project.bin  0x8000000
    Log    ${stmflashresult.stdout}
    Log    ${stmflashresult.stderr}
    Should Contain    ${stmflashresult.stderr}   Flash written and
verified! jolly good!
```

huld

Now the flashing container is ready. Now you can save changes and push file structure to your GitHub repository.

5.5 Jenkins commands

Go to your Jenkins website and select your project.

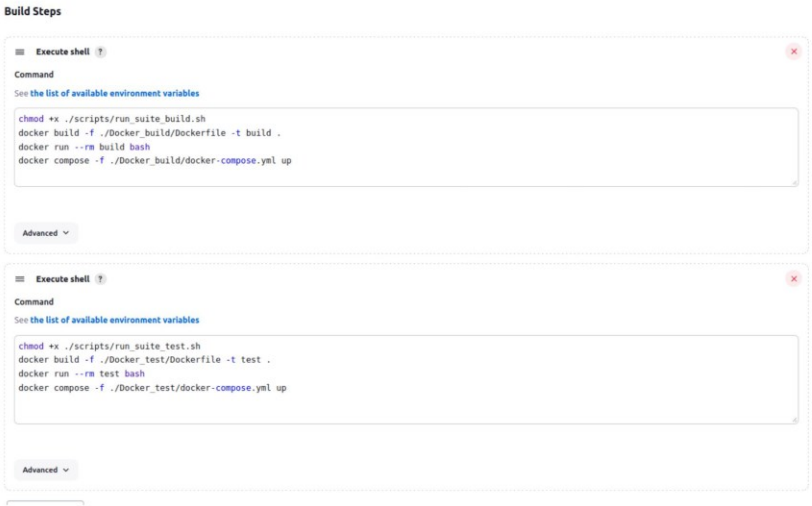
Go to configure your project settings.

Add new Execute shell build step.

Add the following command to the new build step.

```
chmod +x ./scripts/run_suite_test.sh
docker build -f ./Docker_test/Dockerfile -t test .
docker run --rm test bash
docker compose -f ./Docker_test/docker-compose.yml up
```

Your build steps should look like this.

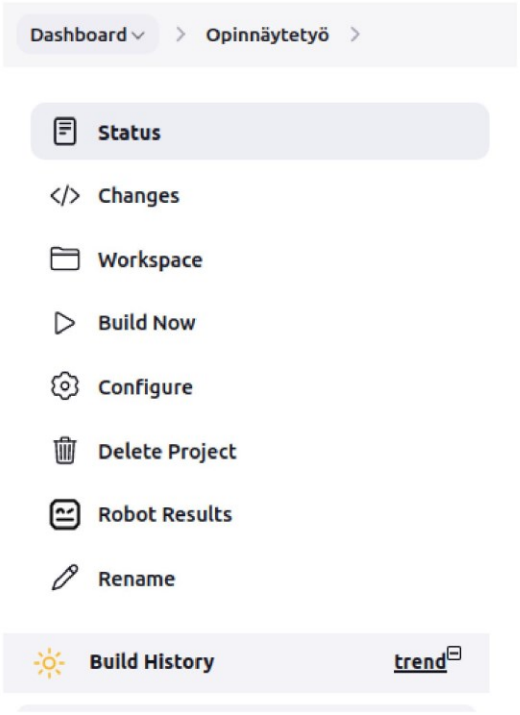


Save your Jenkins project settings and go back to your Jenkins project dashboard.

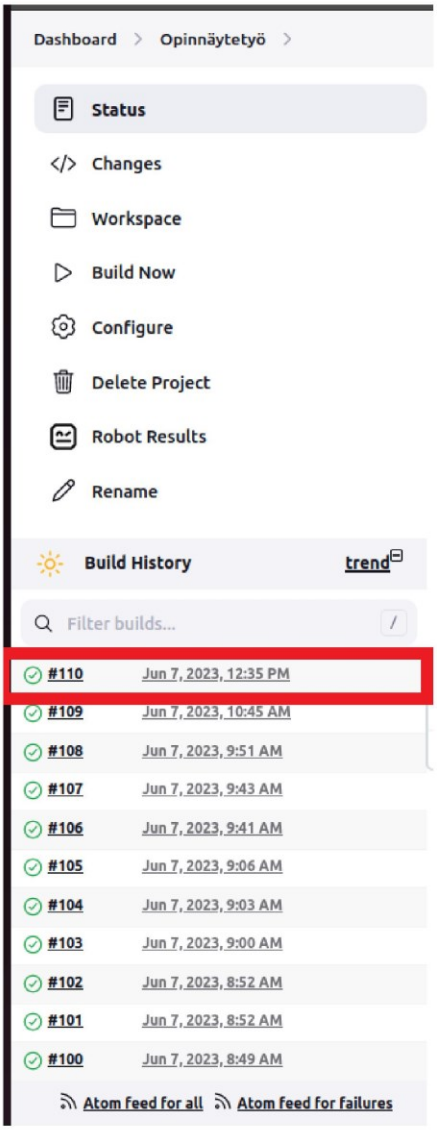
5.6 Test flashing to STM32

Click your project name, and then you should see the project window.

Click “Build Now”

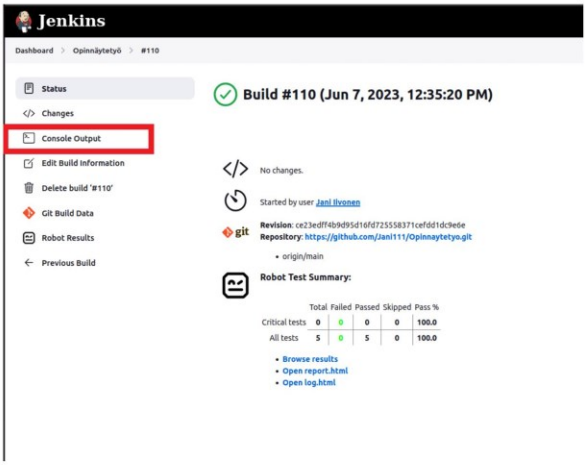


Click the number of the latest finished build.



Then you should see a page like this.

Click “Console Output”.



Scroll to the end of the output.

Your output should look like this. If both tests were marked as “PASS”, your STM32 is now successfully flashed.

```
12:56:45 Attaching to docker_test-test-1
12:56:46 docker_test-test-1  | robot --console verbose --outputdir /reports /suites/
12:56:46 docker_test-test-1  | ==============================================================================
12:56:46 docker_test-test-1  | Suites
12:56:46 docker_test-test-1  | ==============================================================================
12:56:46 docker_test-test-1  | Suites.Robot test
12:56:46 docker_test-test-1  | ==============================================================================
12:56:46 docker_test-test-1  | STM32 Found                                                           | PASS |
12:56:46 docker_test-test-1  | ------------------------------------------------------------------------------
12:56:50 docker_test-test-1  | STM32 Flash                                                           | PASS |
12:56:50 docker_test-test-1  | ------------------------------------------------------------------------------
12:56:50 docker_test-test-1  | Suites.Robot test                                                     | PASS |
12:56:50 docker_test-test-1  | 2 tests, 2 passed, 0 failed
12:56:50 docker_test-test-1  | ==============================================================================
12:56:50 docker_test-test-1  | Suites                                                                | PASS |
12:56:50 docker_test-test-1  | 2 tests, 2 passed, 0 failed
12:56:50 docker_test-test-1  | ==============================================================================
12:56:50 docker_test-test-1  | Output:  /reports/output.xml
12:56:50 docker_test-test-1  | Log:     /reports/log.html
12:56:50 docker_test-test-1  | Report:  /reports/report.html
12:56:50 docker_test-test-1 exited with code 0
12:56:50 Robot results publisher started...
12:56:50 -Parsing output xml:
12:56:50 Done!
12:56:50 -Copying log files to build dir:
12:56:50 Done!
12:56:50 -Assigning results to build:
12:56:50 Done!
12:56:50 -Checking thresholds:
12:56:50 Done!
12:56:50 Done publishing Robot results.
12:56:50 Finished: SUCCESS
```