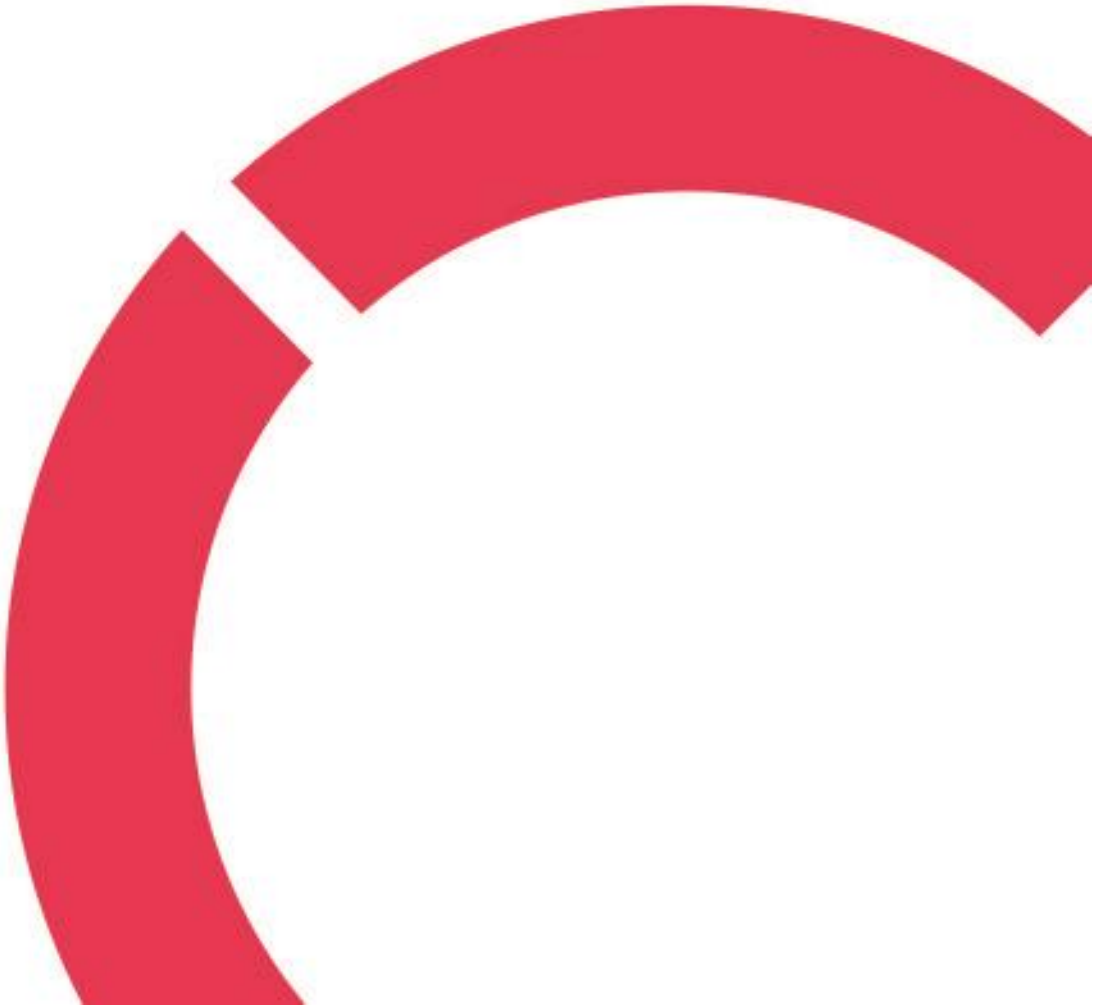
Lassi Kananen

STM32-KEHITYSALUSTA JA -MIKROKONTROLLERIT

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Sähkö- ja automaatiotekniikan koulutus
Huhtikuu 2024



TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Huhtikuu 2024	**Tekijä/tekijät** Lassi Kananen
Koulutus Sähkö- ja automaatiotekniikka		☒ AMK ☐ YAMK
Työn nimi STM32 KEHITYSALUSTA JA MIKROKONTROLLERIT		
Työn ohjaaja Hannu Ala-Pöntiö		**Sivumäärä** 39 + 11
Työelämäohjaaja		

Opinnäytetyön aiheena oli tutkia sulautettujen järjestelmien parissa laajasti käytettyjä STM32-mikrokontrollereita ja -kehitysalustaa ja selvittää, kuinka haastavaa yksinkertaisen koekytkentälaitteen toteutus tässä ympäristössä on.

Opinnäytetyössä käytettiin Nucleo-F446RE ja Nucleo-F072RB kehityskortteja, sekä STM32CubeIDE kehitysympäristöä. Ohjelmointi toteutettiin C-kielellä ja HAL-funktioita hyödyntäen. Erilaisia perusohjauksia ja tiedonsiirtoväyliä testattiin ensin erikseen ja lopuksi nämä yhdistettiin toimivaksi koekytkentälaitekokonaisuudeksi.

Kirjallisessa osuudessa pyrittiin kuvaamaan työn toiminnallinen osuus mahdollisimman kattavasti, ongelmakohdat mukaan lukien. Kirjallisen raportin tavoitteena oli tuoda lukijalleen lisää ymmärrystä STM32-mikrokontrollereista, niiden ohjelmoinnista ja ohjelmointityökaluista.

Opinnäytetyölle asetettuihin tavoitteisiin päästiin kirjallisen ja toiminnallisen osion osalta. Kehitysalustaa ei voi haasteellisuutensa vuoksi suositella perusteiden opiskeluun, mutta pohjatietoa omaavalle henkilölle kehitysalusta sopii.

Asiasanat
Hardware Abstraction Layer, STMicroelectronics, STM32, Sulautetut järjestelmät.

ABSTRACT

Centria University of Applied Sciences	**Date** April 2024	**Author** Lassi Kananen
Degree programme Electrical and automation engineering		
Name of thesis STM32 DEVELOPMENT PLATFORM AND MICROCONTROLLERS		
Centria supervisor Hannu Ala-Pöntiö		**Pages** 39 + 11
Instructor representing commissioning institution or company		

The subject of the thesis was to examine widely used STM32 microcontrollers and development platforms in the field of embedded systems and to find out how challenging it is to create a proof-of-concept-prototype in this environment.

C language and HAL functions were used for programming. Multiple basic controls, such as PWM, I/O and ADC were first tested separately, and finally, they were combined into a functional proof-of-concept device.

The written report aimed to provide a comprehensive overview of the functional aspects of the project, including any challenges encountered along the way. Its objective was to enhance the reader's understanding about STM32 microcontrollers, programming techniques, and associated tools.

The thesis successfully accomplished its set objectives in both the written and functional sections. Due to its complexity, the development platform cannot be recommended for beginners, but it is suitable for individuals with some prior knowledge with embedded systems.

Key words
Embedded systems, Hardware Abstraction Layer, STMicroelectronics, STM32.

KÄSITTEIDEN MÄÄRITTELY

ADC

Analog-to-digital-converter, analogia-digitaalimuunnin on piiri, joka muuntaa analogisen signaalin digitaaliseksi.

ARDUINO

Erityisesti opiskelu- ja harrastekäytössä suosittu kehityskortti ja -alusta.

EEPROM-muisti

Electrically Erasable Programmable Read-Only Memory. Uudelleenkirjoitettava muistityyppi, joka säilyttää tietonsa myös virran katketessa.

FLASH-muisti

Uudelleenkirjoitettava muistityyppi, joka säilyttää tietonsa virran katketessa.

I/O-portti

Input/Output, eli tulo- tai lähtöportti. Portit joko vastaanottavat tai lähettävät haluttua signaalia.

JTAG

Joint Test Action Group. Standardoitu vianhakurajapinta.

Kehitysalusta

Laajempi käsite, joka kattaa kehityskortin lisäksi myös kehityskortin ohjelmointia varten tarvittavat ohjelmistot.

Kehityskortti

Kehityskortilla tarkoitetaan useimmiten fyysistä piirilevyä, joka sisältää mikrokontrollerin tai -prosessorin, sekä esimerkiksi I/O-porteille varattuja liittimiä.

LISÄKORTTI

Kehityskorttiin suoraan liitettävissä oleva valmis piirilevy, joka sisältää usein erilaisia liittimiä sekä sensoripiirejä.

MCU

Micro Controller Unit, mikrokontrolleri. Integroitu piiri, joka sisältää prosessorin, muistia ja I/O-liitäntämahdollisuudet. Käytetään usein sulautettujen järjestelmien ohjaukseen.

MPU

Micro Processor Unit, mikroprosessori. Kuin mikrokontrolleri, mutta sisältää enemmän laskentatehoa ja kykenee suorittamaan monimutkaisempia käyttöjärjestelmiä, esimerkiksi Linuxia.

MSB

Most Significant Byte, eniten merkitsevä bitti, eli bitti, jonka kakkosen potenssi on suurin.

RFID

Radio Frequency Identification, radiotaajuuksinen etätunnistusteknologia, jota hyödynnetään esimerkiksi kulkulätkissä.

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

KUVIOT

KUVAT

TAULUKOT

1 JOHDANTO

Opinnäytetyön tarkoituksena on tutkia STM32-kehityskortteja yleisesti sekä analysoida, kuinka haastavaa opiskelijatasoisen projektin, kuten esimerkiksi yksinkertaisen mittalaitteen toteuttaminen, STM32-kehitysympäristössä on. Tavoitteena on kattaa perusteita, kuten I/O- ja PWM-ohjauksia sekä sulautetuissa järjestelmissä ja laitevalmistuksessa yleisesti käytettyjä SPI- ja I2C-tiedonsiirtoväyliä. Lopputuloksena pyritään luomaan STM32-kehitysalustalle kattava ja käytännönläheinen aloitusopas, joka tarjoaa lukijalleen lisävaihtoehdon projektilähtöisten kurssien kehitysalustavalikoimaan ja tukee aiheeseen liittyvää itsenäistä opiskelua.

Sulautetut järjestelmät ovat keskeisessä asemassa nykypäivän teknologiakentällä ja ne ovat laajassa käytössä erilaisissa sovelluksissa teollisuudessa, kulutuselektroniikassa sekä esimerkiksi lääketieteessä. Kehitysalustojen ja mikrokontrollerien, kuten STM32:n, ymmärtäminen ja hallinta on olennaista monille insinöörikoulutuksen aloille ja siksi niiden tutkiminen opinnäytetyön aiheena on ajankohtaista ja merkityksellistä. STM32-kehitysalusta valikoitui opinnäytetyön aiheeksi, sillä STMicroelectronicsin valmistamat mikrokontrollerit ja -prosessorit ovat laitesuunnittelun saralla hyvin suosittuja. Aiheen valintaan vaikutti myös kirjoittajan halu siirtyä opetuskäytössäkin suositusta Arduino-kehitysalustasta seuraavaan ja syventää omaa ymmärrystään erilaisten mikrokontrollerien ja kehitysalustojen sähköisestä ja ohjelmallisesta toiminnasta.

Opinnäytetyö pyrkii tarjoamaan käytännönläheisen näkökulman STM32-kehitysalustoihin ja niiden käyttöön tarjoamalla konkreettisia esimerkkejä sekä vinkkejä mikrokontrollerin eri toimintojen ohjelmoimiseksi. Lisäksi työssä esitellään myös käytettyjen teknologioiden olennainen teoreettinen osuus, joka luo perustan käytännön toteutuksille. Lopuksi esitellään harjoitelluista toiminnoista muodostettu koekytkentälaite, jolla demonstroidaan, miten toimintojen yhdistäminen toimivaksi järjestelmäksi tapahtuu.

2 STM32 KEHITYSALUSTA

STM32 on ranskalais-italialaisen STMicroelectronicsin kehittämä, Arm Cortex®-M arkkitehtuuriin perustuva 32-bittinen mikrokontrolleriperhe, joka on laajassa käytössä nykypäivän laitevalmistuksessa. Sarjaan kuuluu lukuisia erilaisia mikrokontrollereita, jotka poikkeavat toisistaan muun muassa väylä-ominaisuuksien, kellotaajuuksien, virrankulutuksen, muistien ja hinnankin suhteen.

Mikrokontrollereiden lisäksi STMicroelectronics tarjoaa ison valikoiman erilaisia mikrokontrolle-reihinsa perustuvia kehityskortteja. STMicroelectronics jaottelee kehityskorttinsa kolmeen eri malliin, Discovery-, Nucleo-, sekä Eval-kortteihin. Myös muut valmistajat ovat tuottaneet STM32 mikrokont-rollereihin perustuvia kehityskortteja, joista tunnetuimpia esimerkkejä ovat harrastuskäytössä suositut ”Blue Pill” ja ”Black Pill”-nimiset kehityskortit (Posch 2021).

Kehityskorttien mikrokontrollereiden ohjelmointia, virheenetsintää ja -korjausta sekä esimerkiksi sar-javäylien monitorointia ja ohjelmien siirtoa varten tarvitaan erilaisia PC:lle ladattavia ohjelmia. Ohjel-mistojen osalta tässä opinnäytetyössä keskitytään käyttämään STMicroelectronicsin ohjelmistovalikoi-masta löytyvää STM32CubeIDE:ä, joka pitää sisällään välttämättömimmät toiminnot mikrokontrolle-reiden ohjelmoimiseksi.

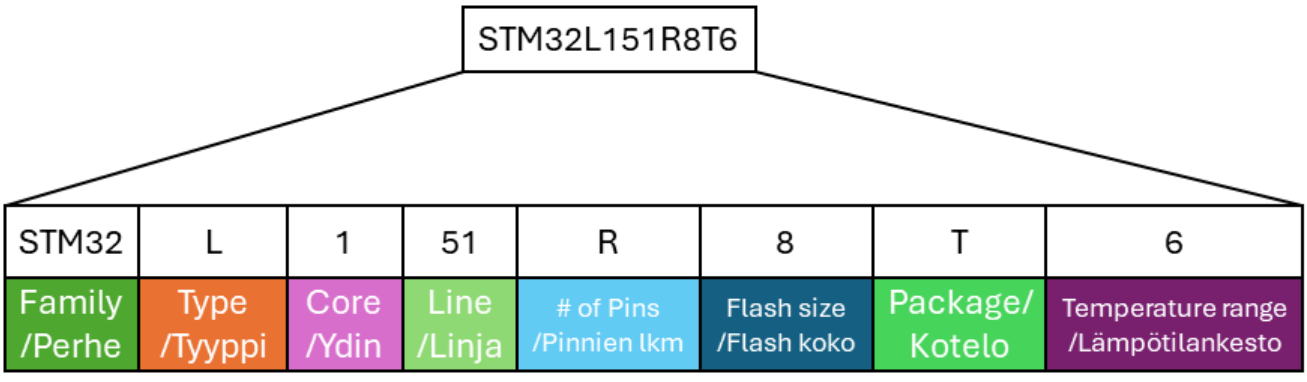
2.1 STM32 Kehityskortit

STMicroelectronics jaottelee valmistamansa kehityskortit kolmeen malliin. Merkittävimpänä erona ke-hityskorttimallien välillä voidaan pitää niiden käyttökohteita. Discovery-mallit on tarkoitettu sovellus-kohtaisten ominaisuuksien testaamiseen (STM32 Discovery Kits). Discovery-kehityskortit sisältävät-kin usein erilaisia oheislaitteita, kuten esimerkiksi näyttöjä ja mikrofoneja. Nucleo-mallin kehityskortit on luotu prototyyppien valmistusta, sekä uusien ideoiden testaamista varten (STM32 Nucleo Boards). Eval-kehityskortit on suunniteltu toimimaan täysimittaisena malliratkaisuna laitesuunnittelijoille. Ke-hityskortit sisältävätkin erilaisia piirejä, kuten antureita, lähetinvastaanottimia, muistirajapintoja sekä näyttöjä. (STM32 Eval Boards). Toinen merkittävä ero kehityskorttien välillä on hinta, Nucleo- sekä Discovery-kehityskortit maksavat kymmeniä euroja, kun taas vaativampaan käyttöön tarkoitetut Eval-mallin kehityskortit maksavat satoja euroja.

Opinnäytetyössä käytettävät Nucleo-malliset kehityskortit jaotellaan vielä kolmeen alakategoriaan: Nucleo-32, -64 ja -144 kortteihin. Nimessä oleva numerosarja viittaa kehityskortin pinnien lukumäärään. Eroja Nucleo-kehityskorteissa on kuitenkin myös FLASH-muistin määrässä, suorituskyvyssä ja siinä, minkä tyyppisen mikrokontrollerin kehityskortit sisältävät. 32-pinniset kehityskortit sisältävät maksimissaan satoja kilotavua muistia ja mikrokontrollerit tyypiltään on joko valtavirran (eng. mainstream) tai matalan tehonkulutuksen (eng. ultra-low-power) mikrokontrollereita. Suurin osa 144-pinnisistä kehityskorteista sisältää yli megatavun verran muistia ja mikrokontrollerien tyyppinä on pääasiassa joko matala tehonkulutus tai korkea suorituskyky (eng. high-performance). Nucleo-64 kehityskortit asettuvat aiemmin mainittujen välimaastoon, niin muistin määrässä, kuin mikrokontrollerien tyyppienkin suhteen mitattuna. (SMT32 Nucleo Boards.)

2.2 STM32-Mikrokontrollereiden nimeämiskäytännöt

Digikeyn artikkelissa kuvaillaan STM32-mikrokontrollereiden nimeämiskäytännöt seuraavalla tavalla: Jokaisen mikrokontrollerin nimi alkaa tekstillä STM32, joka määrittelee mihin tuoteperheeseen mikrokontrolleri kuuluu. Tuoteperheen jälkeen seuraava kirjain tarkoittaa mikrokontrollerin tyyppiä. 2. numero osoittaa mitä ARM-arkkitehtuurin ydintä mikrokontrolleri hyödyntää. Seuraavat kaksi numeroa määrittelevät mikrokontrollerin linjan, joka kertoo laitekohtaisista ominaisuuksista, kuten nopeudesta tai mitä oheislaitteita mikrokontrolleri pitää sisällään. Seuraava kirjain ilmaisee pinnien lukumäärän. Tämän jälkeinen kirjain tai numero ilmaisee mikrokontrollerin FLASH-muistin koon, jonka jälkeen seuraa käytetty kotelotyyppi. Viimeisellä numerolla osoitetaan mikrokontrollerin lämpötilankesto. (Digikey, 2020.)



KUVIO 1. STM32-mikrokontrollereiden nimeämiskäytännöt (mukaillen Digikey, 2020)

Nimeämiskäytännöistä tulee huomioida, että ne muuttuvat ajan kuluessa tuoteperheiden mukana. Tämä saattaa tuoda mukanaan epäjohdonmukaisuuksia. Esimerkiksi F2-tyypin kontrollerit ovat todellisuudessa ”High Performance”-kategoriaa. Lisäksi osa ydintä ilmaisevista numeroista viittaa samaan ytimeen kuten 1,2 ja 3,4. (Digikey, 2020.)

TAULUKKO 1. Mikrokontrollerien tyypit, ytimet ja pinnien lukumäärät (mukaillen Digikey, 2020)

TYPE		CORE		# OF PINS	ss
F	Foundation	0	ARM Cortex M0	F	20
G	Mainstream	1	ARM Cortex M3	G	28
L	Low-Power	2	ARM Cortex M3	K	32
H	High-Performance	3	ARM Cortex M4	T	36
W	Wireless	4	ARM Cortex M4	S	44
		7	ARM Cortex M7	C	48
				R	64 or 66
				V	100
				Z	144
				I	176

TAULUKKO 2. FLASH-muistin määrä, kotelotyyppi ja lämpötilankesto (mukaillen Digikey, 2020)

FLASH SIZE		PACKAGE		TEMPERATURE	
4	16 KByte	P	TSSOP	6	-40°C ~ 85°C
6	32 KByte	H	BGA	7	-40°C ~ 105°C
8	64 KByte	U	VFQFPN		
B	128 KByte	T	LQFP		
C	256 KByte	Y	WLCSP		
D	384 KByte				
E	512 KByte				
F	768 KByte				
G	1024 KByte				
H	1536 KByte				
I	2048 KByte				

2.3 STM32 Ohjelmistot

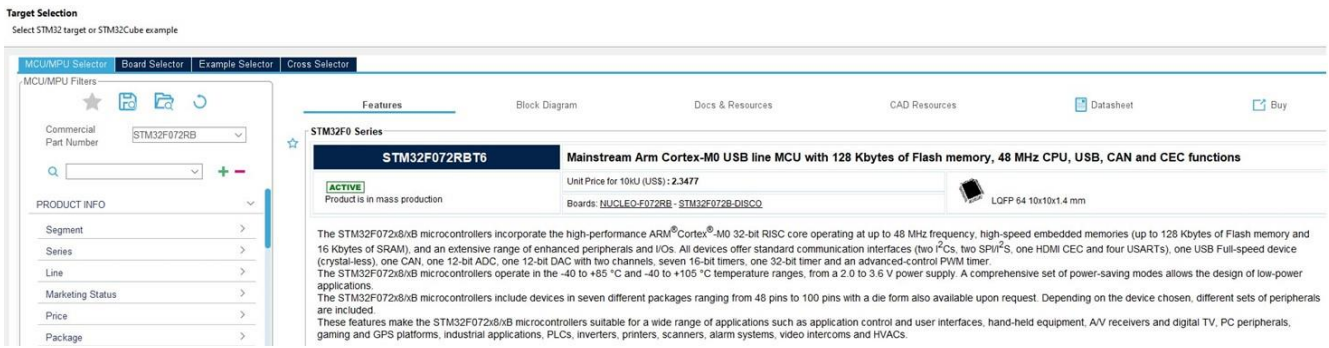
STM32-mikrokontrolleriperheen ohjelmointia, konfigurointia ohjelman diagnosointia varten tarvitaan useita erilaisia ohjelmia, joilla jokaisella on oma käyttötarkoituksensa kehitysprosessissa. STMicroelectronicsin kehittämiä ohjelmia ovat konfigurointityökalu STM32CubeMX, ohjelmointiympäristö STM32CubeIDE, mikrokontrollerin muistin lukemiseen, kirjoittamiseen ja varmentamiseen JTAG- ja SWD-vianhakurajapintojen kautta tarkoitettu STM32CubeProgrammer sekä ohjelman käytönaikaista monitorointia ja diagnosointia varten kehitetty STM32CubeMonitor. (STM32 Software Development Tools.) Ohjelmista on nykypäivänä koottu isompia ohjelmistokokonaisuuksia, eikä jokaista tarvitse ladata erikseen. Seuraavat alaluvut keskittyvät esittelemään opinnäytetyössä käytettyjen ohjelmistojen käyttötarkoituksia sekä toimintaperiaatteita.

2.3.1 STM32CubeIDE

STM32CubeIDE on ohjelmointiympäristö, joka pitää sisällään koodin luomiseen, kääntämiseen ja virheenkorjaukseen, sekä oheislaitteiden konfiguroimiseen tarvittavat työkalut. Ohjelma tukee C- ja C++-ohjelmointikieliä ja sen ohjelmointiominaisuudet perustuvat Eclipse®/CDT™ viitekehykseen ja GCC-kääntäjään ja vianhakutoiminnot GDB-vianhakijaan. Ohjelma sisältää myös työkaluja muistin käytön, sekä koodin analysoimiseen. STM32CubeIDE:hen on sisällytetty aiemmin omana erillisenä ohjelmana toiminut (projektinluonti- ja) konfigurointiohjelma STM32CubeMX. (Integrated Development Environment for STM32.)

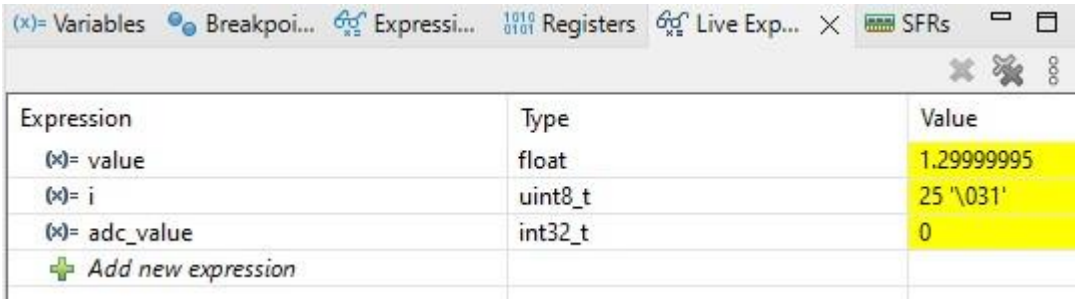
2.3.2 STM32CubeIDE:n olennaisia ominaisuuksia

Target Selector-näkymässä valitaan kehityskortti, jolle ohjelmaa ollaan luomassa. Mikäli kehityskortti ei ole STMicroelectronicsin valmistama, valitaan toiselta välilehdeltä pelkkä STM32-mikrokontrolleri. Target selector-näkymässä voi hakea nopeasti erilaisia mikrokontrollereita ja kehitysalustoja haluttujen ominaisuuksien, esimerkiksi muistin määrän perusteella. Lisäksi näkymä niputtaa yhteen kaikki tuotekohtaiset manuaalit sekä datalehdet ja esimerkiksi kehityskorttien kytkentäkuvat.



KUVA 1. Target selector-näkymä MCU/MPU-välilehdeltä

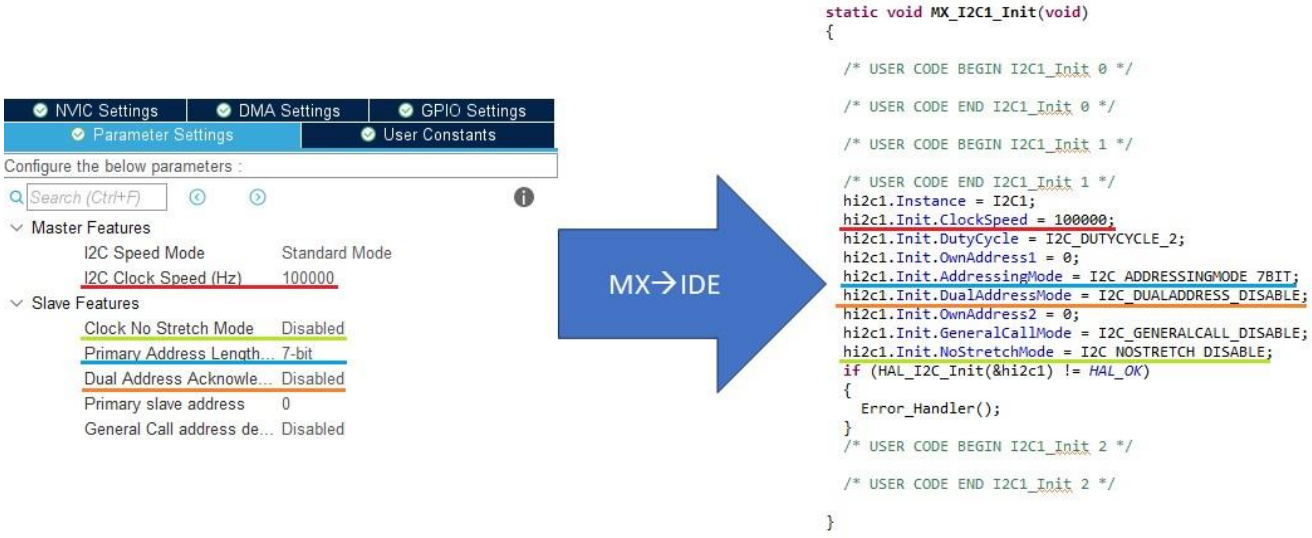
STM32CubeIDE sisältää myös vianetsintätyökalun. Työkalun avulla on mahdollista seurata muuttujien tiloja ja arvoja, rekisterien tiloja, sekä asettaa breakpoint-pisteitä, jotka pysäyttävät ohjelman toiminnan halutun rivin kohdalle. Opinnäytetyötä tehdessä vianetsintätyökalua käytettiin pääasiassa muuttujien tilan reaaliaikaiseen seuraamiseen vaa'an ohjelmoinnissa. Kuvassa 2 seurataan value-nimistä muuttujaa, jonka reaaliaikaisesti päivittyvä arvo on vaakayksikön tulostamaa raakadataa.



KUVA 2. Muuttujien tilan seurantaa vianhakutyökalulla

2.3.3 STM32CubeMX

STM32CubeMX on konfigurointityökalu, joka mahdollistaa mikrokontrollereiden ja mikroprosessoreiden toimintojen, kuten IO-pinnien tai analogia-digitaalimuuntimen konfiguroinnin graafisen käyttöliittymän avulla. Työkalulla voidaan myös laskea mikrokontrollerin tai -prosessorin tehonkulutus, tai säätää niiden kellotaajuuksia. (STM32Cube initialization code generator.) STM32CubeMX:n käyttö nopeuttaa ohjelmointiprosessia, sillä se generoi säädettyjen asetuksien perusteella automaattisesti koodia, joka muuten olisi kirjoitettava manuaalisesti (KUVA 3).

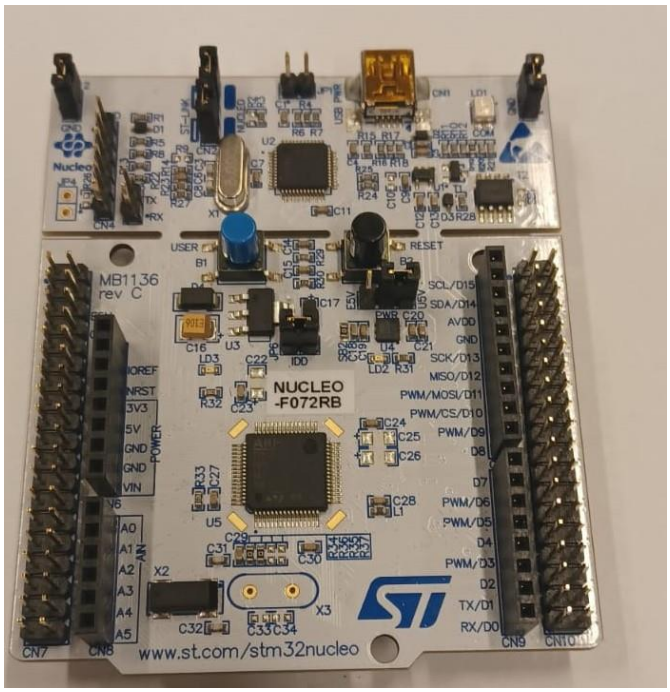


KUVA 3. STM32CubeMX generoi I2C-väylälle asetetuista parametreistä koodia

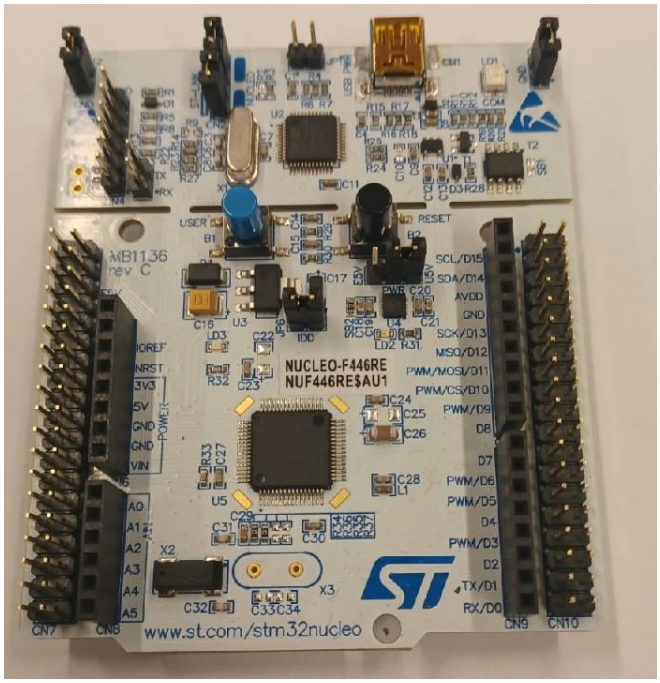
3 OPINNÄYTETYÖN KEHITYSKORTIT

Opinnäytetyössä käytettyjen kehityskorttien valintaprosessi oli laajan tuotevalikoiman takia haastava. Hakua kuitenkin saatiin rajattua haluttujen ominaisuuksien, kuten hinnan, väylämahdollisuuksien sekä käytettävien I/O-pinnien lukumäärän avulla. Lopulliset rajaukset hakuun asetti kuitenkin käytetyn verkkokaupan, Elfa Distrelecin valikoima, joka kavensi hakuvaihtoehtoja merkittävästi ja josta oli helppo kopioida tuotenimiä Target Selector-näkymään tarkempaa vertailua varten. Kehityskorteiksi kaavailtiin aluksi Nucleo-144 kokoluokan tuotteita, mutta lopulta päädyttiin Nucleo-64 kokoisiin Nucleo-F446RE ja Nucleo-F072RB kehityskortteihin.

Kehityskortteja oli järkevintä hankkia kaksi, sillä tällä menettelemällä näin on mahdollista perehtyä kahden STM32-mikrokontrollerin välisen kommunikaation toteuttamiseen. Kehityskorttien tilausvaiheessa ei myöskään ollut vielä täysin selvää, mitä erilaisia oheislaitteita ja/tai sensoreita niillä tultaisiin ohjaamaan, joten Nucleo-64 kokoluokan kehityskortteihin integroitu, Arduino Uno-kehityskortin kanssa yhteensopiva liitinrivistö olisi voinut osoittautua hyödylliseksi. Arduino Unon kanssa yhteensopiva liitinrivistö mahdollistaa Arduino Uno-kehityskorteille valmistettujen lisäkorttien suoran kytkemisen Nucleo-64-kehityskorttiin. (KUVA 4.) Arduino Uno, sekä sille valmistetut lisäkortit ovat Centria ammattikorkeakoululla laajasti opetuskäytössä, joten tätä hyödyntämismahdollisuutta ei kannattanut sulkea pois.



KUVA 4. Nucleo-F072RB-kehityskortti. Sisempi liitinrivistö on Arduino-yhteensopiva



KUVA 5. Nucleo-F446RE-kehityskortti

4 STM32-MIKROKONTROLLERIEN OHJELMOINTI

Opiskellessa ennestään tuntemattoman kehitysalustan toimintaa, on järkevintä aloittaa toiminnallisuuksien erillisestä testaamisesta ennen kuin yrittää tuottaa valmista prototyyppikokonaisuutta. Tällä menettelyllä saadaan hankittua pohjatietoa alustasta ja kartoitettua oman osaamisen ja aikaresurssien riittävyyttä. Tässä luvussa kuvataan, miten projektin luonti, sekä erilaisten perusohjausten toteuttaminen STM32-kehitysalustalla tapahtuu käyttäen STM32CubeMX-konfigurointityökalua, C-kieltä sekä HAL-komentoja. Testaukset suoritettiin Nucleo-F072RB kehityskortilla, mutta ne ovat täysin sovellettavissa mille tahansa STM32-kehityskortille mikrokontrollerista riippumatta. Osa testituloksista on saatu oskilloskoopilla mittaamalla, sillä se on nopea ja luotettava tapa kuvata visuaalisesti sähköistä toimintaa, osassa ns. toiminnallinen testitulos, kuten LED-valon syttyminen tai tekstin näkyminen LCD-näytöllä katsottiin riittäväksi.

4.1 Hardware Abstraction Layer (HAL)

HAL, eli Hardware Abstraction Layer sekä LL, eli Low Layer ovat STM32-mikrokontrollereille tarkoitettuja ohjelmointirajapintoja. Ohjelmointirajapinnat sisältävät valmiita toimintoja ja funktioita laitteen ohjelmointia varten. HAL rajapinnoista korkeatasoisempi eli mikrokontrollerin toimintaa peittävämpi, sekä ominaisuuksiin keskittyneempi ja se takaa ohjelmalle paremman siirrettävyyden eri mikrokontrollerien välillä. (STMicroelectronics 2020, 1.) HAL on jaettavissa kahteen osaan, yleiseen sekä laajennusosaan. Yleiset HAL funktiot ovat käytettävissä kaikissa STM32-mikrokontrellereissa ja ne liittyvät usein esimerkiksi IO-ohjauksiin tai erilaisten toimintojen alustuksiin. HAL laajennusosan funktiot ovat mikrokontrollerikohtaisempia ja tarkoitettuja yleensä vain tietyn MCU:n yksittäiselle toiminnolle. (STmicroelectronics 2020, 19–20.)

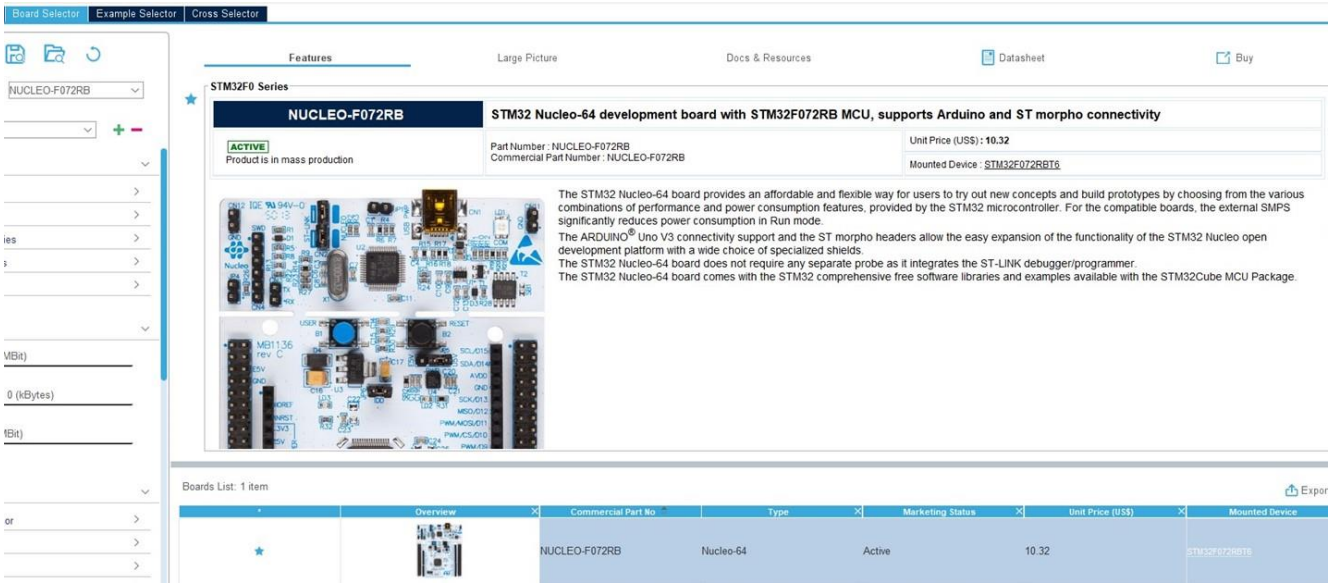
LL on matalamman tason rajapinta, joka soveltuu suorituskyvyllisen nopeutensa ja keveytensä vuoksi raskaampien ohjelmien optimointiin. LL on mikrokontrollerikohtainen ja se vaatii ohjelmoijalta syvällisempää ymmärrystä mikrokontrollerin rekisteritason toiminnasta. (STMicroelectronics 2020, 1.) HAL on ohjelmointirajapinnoista aloittelijaystävällisempi ja tästä syystä opinnäytetyössä keskityttiinkin vain sen käyttöön. STMicroelectronics on dokumentoinut jokaisen STM32-mikrokontrollerityypin

HAL- ja LL-rajapinnat ja manuaalit löytyvät hakusanoilla ”description of stm32XX and lowlayer drivers” (jossa XX = mikrokontrollerin tyyppi esim. F0). Manuaaleissa on lähes 1400 sivua, mutta ne ovat luetteloitu asiallisesti ja kattavat kaikki mikrokontrollerin käytössä olevat HAL- ja LL-funktiot.

4.2 Projektin luonti ja yksinkertainen I/O-ohjaus

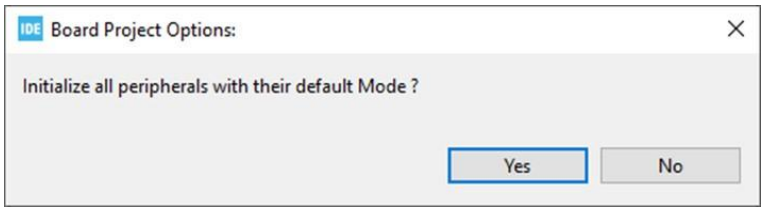
Mikrokontrollerin ohjelmointia opeteltaessa tehdään yleensä ensimmäisenä yksinkertainen LED-valoa vilkuttava ohjelma, useissa kehityskorteissa valmistajasta riippumatta onkin valmiiksi juotettuna tähän tarkoitukseen sopiva LED-valo. Kuten kuvassa 4 on havaittavissa, NUCLEO-F072RB kehityskortissa tämä LED-valo on kytketty porttiin A, pinniin numero 5 ja se on oletuksena merkitty STM32CubeMX:ssä nimellä ”LD2 [Green LED]” (KUVA 8).

Luotaessa uutta projektia STM32CubeIDE:hen, on työnkulku seuraavanlainen: Ensimmäisenä etsitään ja valitaan Target selector-näkymästä käytössä oleva kehityskortti (KUVA 6). Target selector-näkymään pääsee takaisin vain aloittamalla uuden projektin, joten tässä vaiheessa on suotavaa tallentaa kehityskortista tarpeelliseksi oletetut dokumentit, kuten esimerkiksi laitekohtaiset datalehdet. Kehityskortin, tai minkä tahansa tuotteen voi merkitä suosikiksi klikkaamalla sinistä tähtisymbolia, tämä nopeuttaa tuotteen etsintää seuraavalla kerralla.



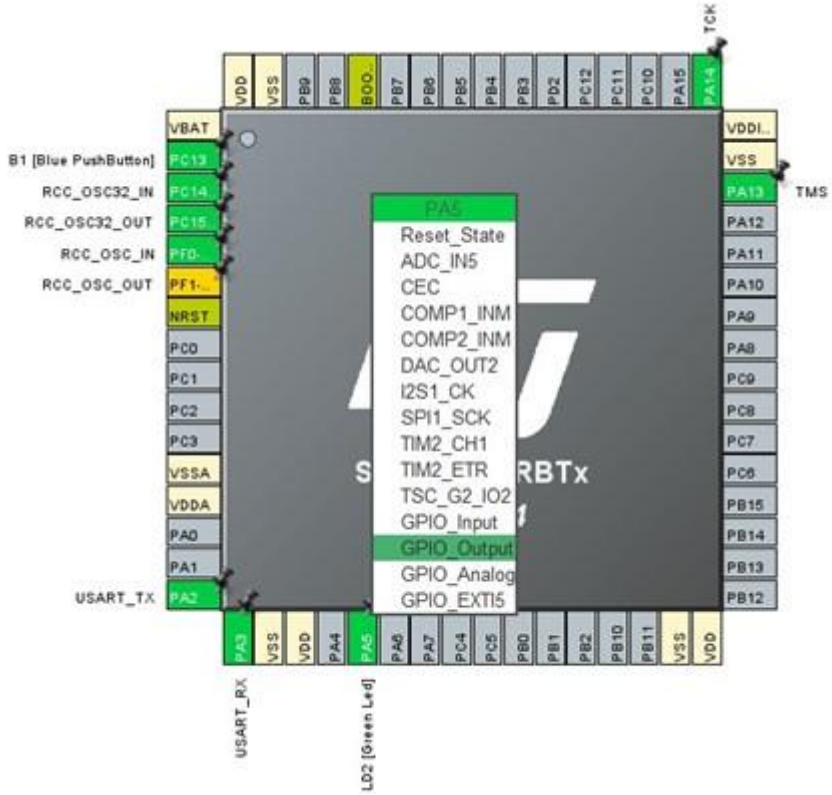
KUVA 6. Target selector näkymä käytössä olevasta kehityskortista

Kehityskortin valinnan jälkeen ilmestyvässä ikkunassa ohjelma haluaa tietää, alustetaanko kaikki oheislaitteet niiden oletusmoodiin, valitsemalla ”kyllä”, STM32CubeMX alustaa kehityskortissa olevat oheislaitteet, kuten LED-valot sekä kytkimet käyttäjälle valmiiksi (KUVA 7).



KUVA 7. Valintaikkuna oheislaitteiden alustuksesta

Siirryttäessä STM32CubeMX konfigurointityökaluun käyttäjä voi konfiguroida mikrokontrollerin pinnit haluamallaan tavalla valitsemalla ohjelman antamista vaihtoehdoista. Kuvassa 8 kehityskortin LED-valolle tarkoitettua pinniä ollaan konfiguroimassa lähdöksi.



KUVA 8. Pinnin PA5 konfigurointivaihtoehdot

Kun projekti tallennetaan STM32CubeMX ikkunassa, konfigurointiasetukset tallentuvat ioc-tiedosto-muotoon ja ovat löydettävissä projektin kansiorakenteesta. Tämän jälkeen ohjelma kysyy käyttäjän lupaa koodin luomiseen. Valittaessa ”kyllä”, ohjelma luo valittuihin konfigurointiasetuksiin perustuvan valmiin koodipohjan nimellä ”main.c”. Ohjelma jaottelee generoimansa valmiin koodin osioihin, jotka ovat merkitty kommenttirivien avulla. On tärkeää huomioida, että käyttäjä muistaa lisätä itse tuottamansa koodirivit ”User code begin” ja ”User code end” kommenttirivien väliin (KUVA 9). Mikäli lisätty koodi ei ole näiden kommenttirivien välissä ja myöhemmin STM32CubeMX:ssä tehdään uusi konfiguraatio, sekä koodin generointi, tuotettu koodi katoaa.

Simple_IO_test.ioc main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart2;

```

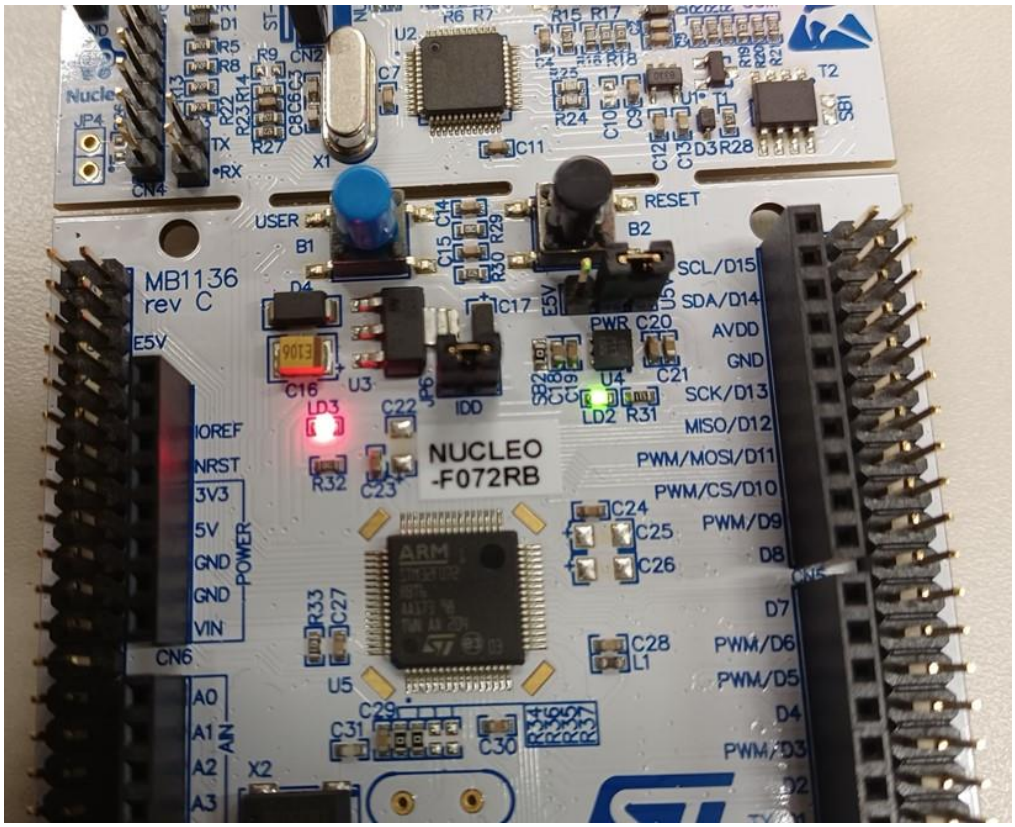
KUVA 9. Aloitusnäkymä koodin generoimisen jälkeen

Kuvassa 10 on nähtävissä LED-valon vilkkumiseen tarvittava yleiset HAL-funktiot. Komento ”HAL_GPIO_TogglePin” saa valon vilkkumaan ja komento ”HAL_Delay” määrittää vilkkumisnopeuden, tässä tapauksessa LED-valo vilkkuu 500 millisekunnin välein.

```
97   while (1)
98   {
99     /* USER CODE END WHILE */
100
101    /* USER CODE BEGIN 3 */
102    //Painamalla CTRL+SPACE saa auki valikkonäkymän josta näkee IDE:n tarjoamat vaihtoehdot HAL-komennoille, porteille ja pinneille
103      HAL_GPIO_TogglePin(GPIOA, GPIO_Pin)
104      HAL_Delay(500);
105  }
106  /* USER CODE END 3 */
107 }
108
109
110
111
112
113
114
115
116
117
118
119
120
```

```
# GPIO_PIN_15
# GPIO_PIN_2
# GPIO_PIN_3
# GPIO_PIN_4
# GPIO_PIN_5
# GPIO_PIN_6
# GPIO_PIN_7
# GPIO_PIN_8
# GPIO_PIN_9
# GPIO_PIN_All
# GPIO_PIN_MASK

Press 'Ctrl+Space' to show Template Proposals
```

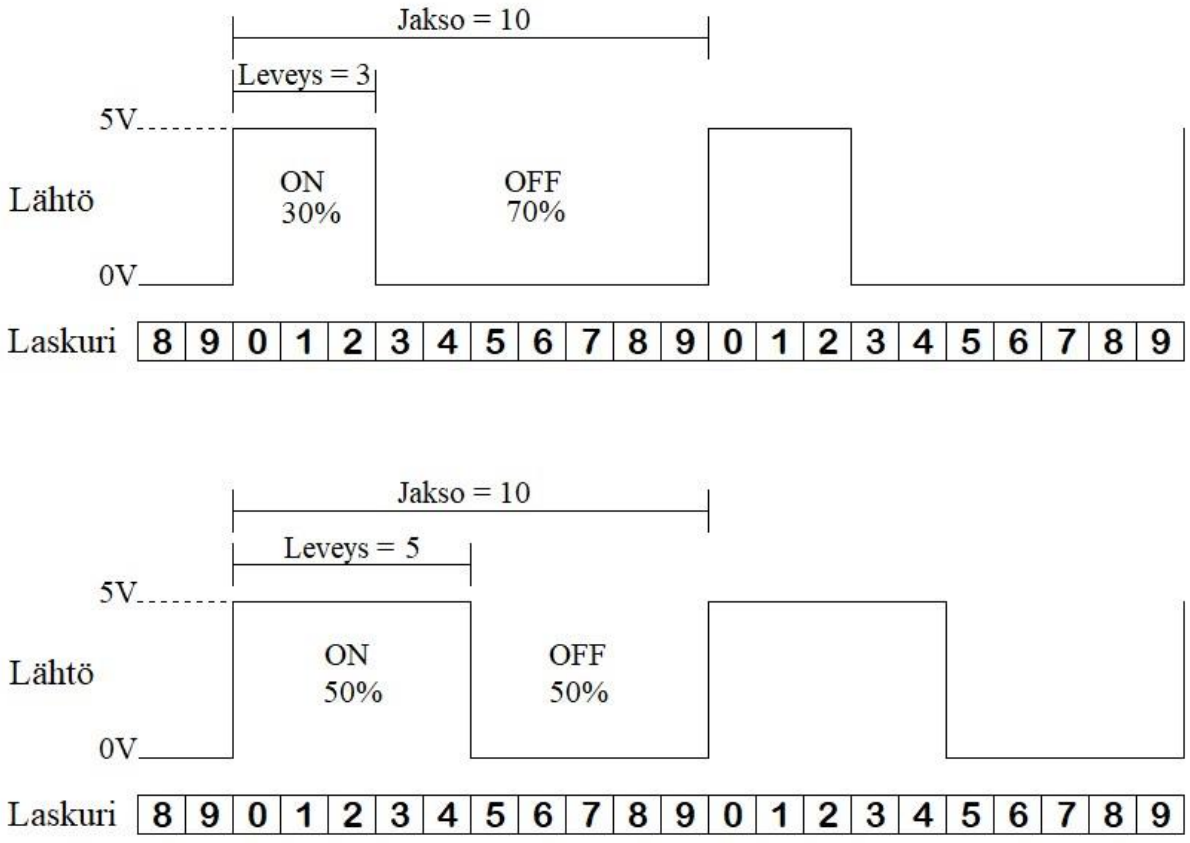
KUVA 10. LED-valon vilkuttamiseen tarvittavat HAL-funktiot



KUVA 11. Vihreä LED-valo LD2 vilkkumassa

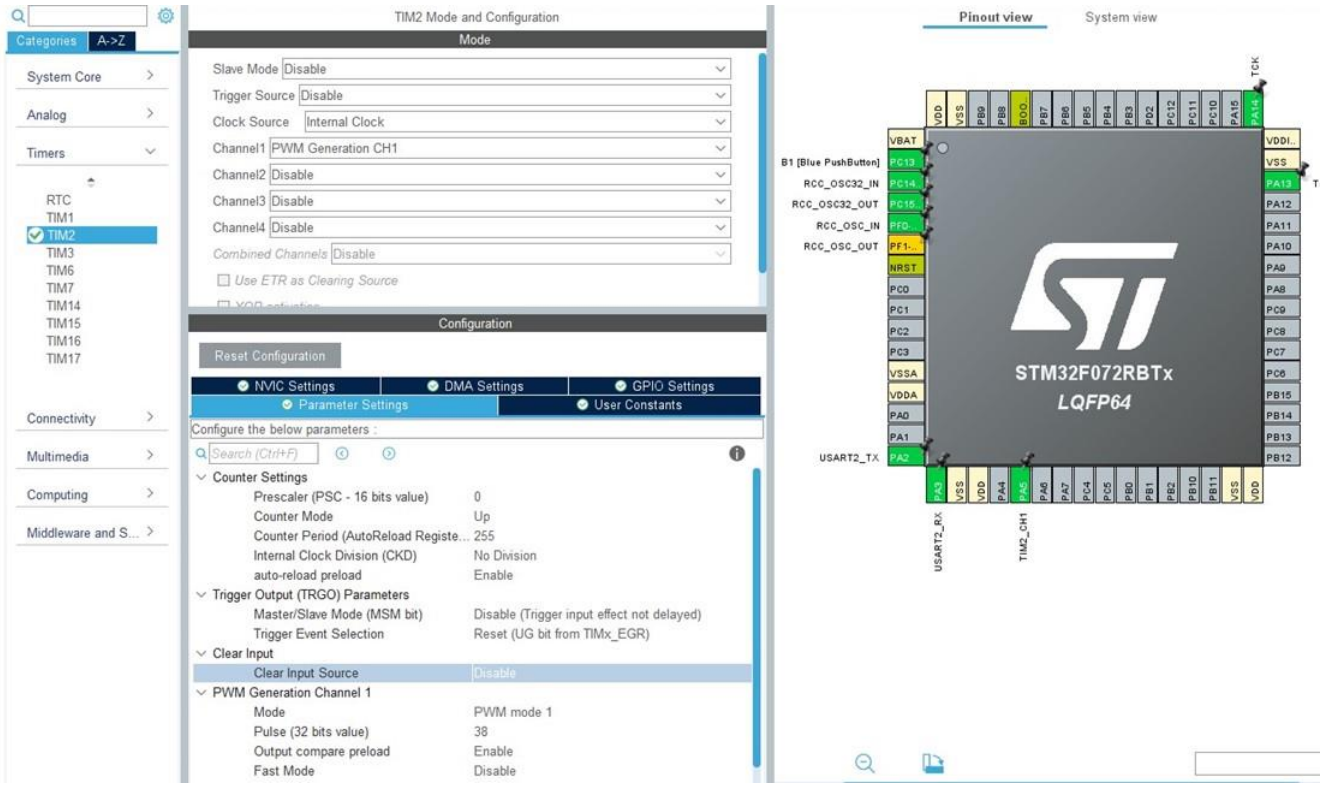
4.3 PWM-ohjaus

PWM (Pulse Width Modulation), eli pulssinleveysmodulaatio on tapa rajoittaa laitteelle menevän tehon määrää kytkemällä signaalia vuorotellen ON- ja OFF-tilan välillä nopealla taajuudella. Signaalin keskimääräinen jännite määräytyy sen mukaan, kauanko signaali on ON-tilassa suhteessa OFF-tilaan. Tätä ON- ja OFF-tilan suhdelukua kutsutaan pulssisuhteeksi ja se on taajuuden kanssa toinen keskeinen parametri PWM-signaalin ohjaamiseksi. (Dietrich 2020, luku ” What is Pulse Width Modulation?”.) PWM-teknologiaa hyödynnetään runsaasti erilaisissa moottori- ja servomoottoriohjauksissa sekä esimerkiksi valonsäätimissä (Colley 2020). Pulssinleveysmodulaation laajan käyttöasteen vuoksi ohjauksen erillinen testaaminen katsottiin viisaaksi, sillä ohjausta voi hyödyntää useiden erilaisten laite- tai prototyyppiratkaisujen luomisessa.



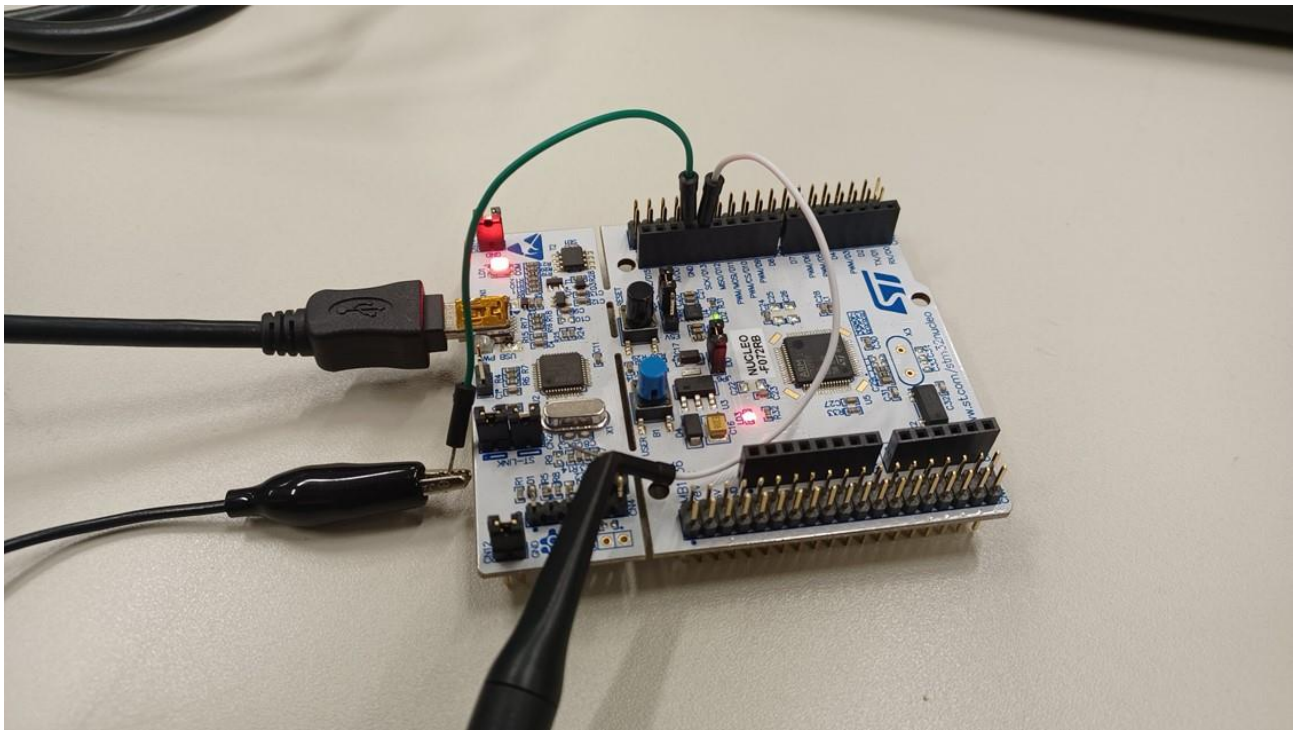
KUVA 12. Esimerkkejä erilaisista pulssisuhteista (mukaillen Colley 2020)

Toteutin PWM-pulssin tuoton seuraavalla tavalla. Etsin ja valitsin STM32CubeMX konfigurointityökalusta mikrokontrollerin pinnin, joka piti sisällään ajastintoiminnon. Ajastimen valikkorakenteesta valitsin ajastimelle kanavan, joka tukee PWM-signaalin tuottoa ja tämän jälkeen säädin parametriasetuksista ”Counter period”-riviltä laskurin arvoon 255 ja ”Pulse”-riviltä pulssin arvoon 38. (KUVA 13.)



KUVA 13. Konfigurointiparametrit PWM-pulssin tuottamiseksi

Näillä asetuksilla ajastimen laskuri laskee lukuun 255, näistä luvuista pulssi on ON-tilassa 38 luvun ajan. Laskennallinen pulssinsuhde saadaan jakamalla pulssien lukumäärää laskurin arvolla eli:
Pulssisuhde=38/255 = 14.9%
Koodiin lisäsin HAL-funktion ”HAL_PWM_START(&htim2, TIM_CHANNEL_1);” joka käynnistää PWM-generoinnin ajastimella 2, kanavassa 1. Lopuksi asetin oskilloskoopin mittauspäät kiinni kehityskorttiin (KUVA 14). Mittaustuloksen perusteella totesin PWM-pulssin tuoton onnistuneeksi (KUVA 15).



KUVA 14. Oskilloskoopin mittauspäät asetettuna kehityskorttiin

KUVA 15. Oskilloskoopin mittauskuva PWM-pulssista

4.4 SPI ohjauksen testaaminen (isäntälaite)

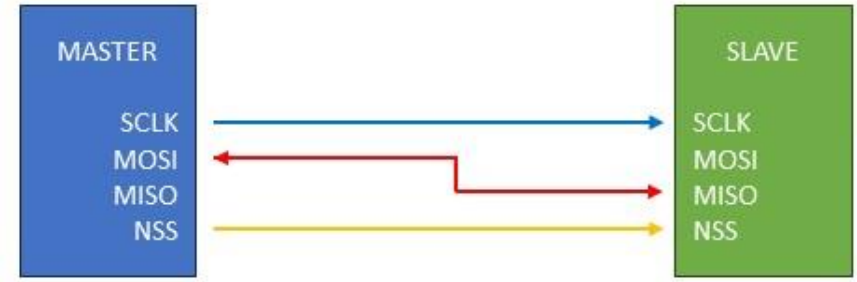
Serial Peripheral Interface, eli SPI on Motorolan 1980-luvulla kehittämä väylästandardi (McCreary 2020, luku ” The history of SPI”). Väylän toiminta perustuu isäntä-orja (eng. master-slave) periaatteeseen sekä neljään johtimeen. Johtimet ovat SCLK (Serial Clock), MOSI (Master-Out-Slave-In), MISO (Master-In-Slave-Out) sekä NSS (Slave Select). Isäntälaite tuottaa SCLK-, eli kellosignaalin, jota käytetään tiedonsiirron synkronoimiseen, varsinaisen dataliikenteen tapahtuessa MISO- ja MOSI-johtimissa. NSS-signaalia käytetään erityisesti usean orjalaitteen järjestelmässä, jossa isäntälaitteen on valittava, minkä laitteen kanssa kommunikoidaan. (SPI Backround, luku ”Theory of operation”.)

Teknologiaa hyödynnetään muun muassa SD-muistikorttien, sekä RFID-lukijoiden kaltaisten oheislaitteiden ohjaamisessa mikrokontrollerin avulla (Campbell, luku "Introduction to SPI communication"). SPI-väylä on tarkoitettu lyhyen kantaman tiedonsiirtoon (SPI Protocol, luku "Disadvantages of SPI"). Käytännössä lyhyt kantama tarkoittaa sitä, että teknologia soveltuu luotettavimmin tiedonsiirtoon piirilevyn sisäisessä kommunikaatiossa, jossa mikrokontrollerin ja ohjattavan laitteen, tai toisen mikrokontrollerin välinen etäisyys pysyy senttikokoluokassa.

Tiedonsiirto SPI-väylällä on mahdollista konfiguroida tapahtumaan kolmella eri tavalla. Tavat ovat Full duplex, half duplex ja simplex. Full duplex konfiguraatiota käyttämällä dataa voidaan siirtää kaksisuuntaisesti, sekä yhtäaikaisesti isäntä- ja orjalaitteiden välillä käyttäen MOSI- ja MISO-signaaleja (KUVIO 2).

KUVIO 2. Full duplex-konfiguraatio (mukaillen Fastbitlab 2019)

Half duplex konfiguraatiossa isäntälaitteen MOSI-signaali, sekä orjalaitteen MISO-signaali ovat kytkettyinä ristiin ja tiedonsiirto tapahtuu vuorotellen (KUVIO 3).



KUVIO 3. Half duplex-konfiguraatio (mukaillen Fastbitlab 2019)

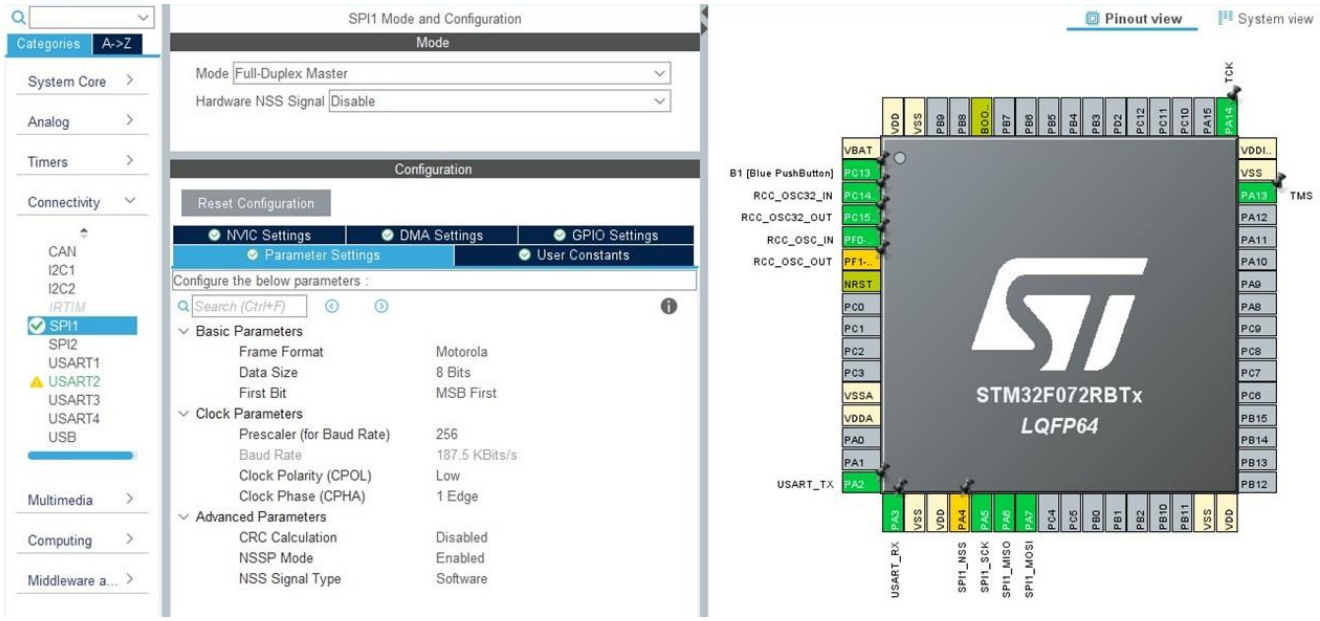
Simplex menetelmässä vain toinen datalinjoista, MISO tai MOSI on käytössä ja tiedonsiirto tapahtuu niin, että isäntä- ja orjalaitteet on alustettu toimimaan ”vain lähetys” tai ”vain vastaanotto” tiloissa. (KUVIO 4).

KUVIO 4. Simplex konfiguraatio (mukaillen Fastbitlab 2019)

Käytettävä konfiguraatio valitaan käyttökohteiden asettamien vaatimusten perusteella (Fastbitlab 2019). Esimerkiksi, mikäli käyttökohde, kuten oheislaite, vaatii suurta, nopeaa ja kaksisuuntaista tiedonsiirtoa, on suotavinta valita full duplex-konfiguraatio. Mikäli oheislaite tarvitsee esimerkiksi vain käskyn jonkin toiminnon suorittamisen aloittamiseksi, on järkevintä valita simplex-konfiguraatio.

SPI on hyvä esimerkki niin sanotusti ”löyhästi” standardoidusta tiedonsiirtoprotokollasta ja väylärat-kaisun hyödyntämisessä on suurta vaihtelua niin käyttökohteiden kuin toteutuksenkin suhteen. Tämä tuo kehittäjälle paljon vapauksia väylän käytön suhteen, mutta toisaalta myös tarkoittaa sitä, että SPI väylää hyödyntävien oheislaitteiden ohjaaminen on selvitettävä erikseen niiden datalehdistä.

Osaksi opinnäytetyötä SPI-väylä valikoitui, sillä väylää hyödynnetään paljon elektroniikkasuunnittelussa ja sulautetuissa järjestelmissä. Aluksi pidin tiedonsiirron vähimmäisvaatimuksena simplex-konfiguraatiota, jossa isäntälaite lähettää ”vain vastaanotto”-tilassa olevalle orjalaitteelle käskyjä erilaisten toimintojen käynnistämiseksi. Tähän vähimmäisvaatimukseen pääseminen ei ollut erityisen haastavaa ja sainkin onnistuneesti isäntälaitteen konfiguroitua (KUVA 16).

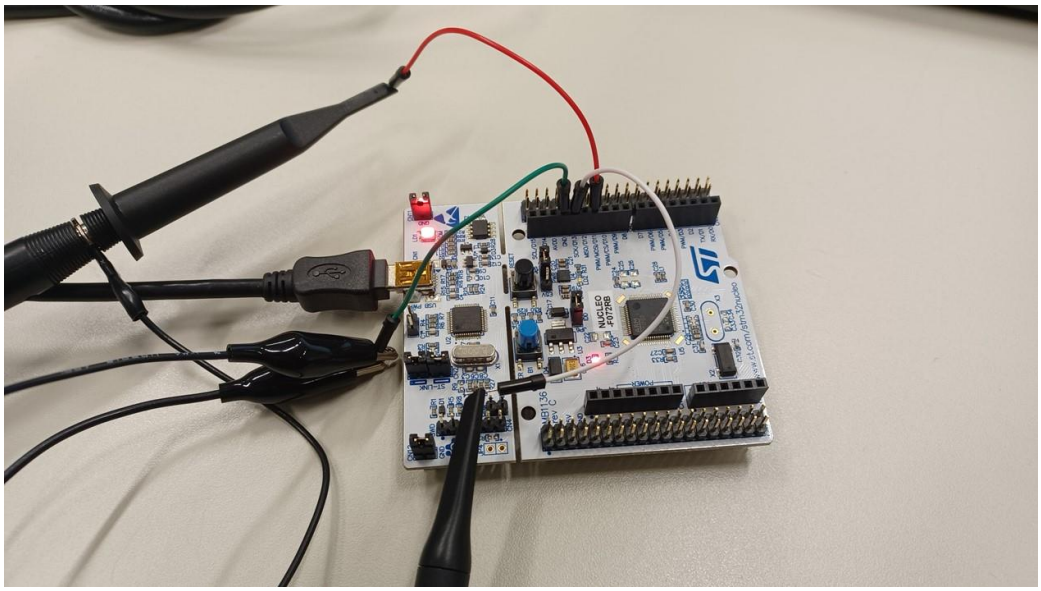


KUVA 16. Konfigurointiasetukset SPI-väylän isäntälaitetta varten

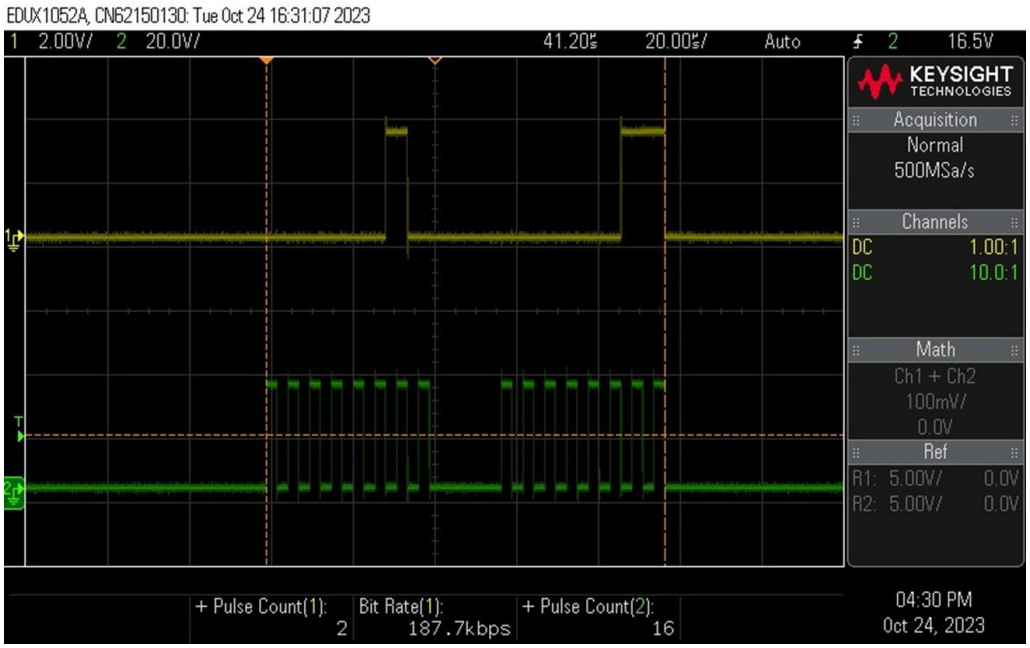
Kuvassa 17 on nähtävissä kirjoittamani testikoodi, joka lähettää HAL-funktioiden avulla SPI-väylälle binäärimuotoisena numeroita 2, sekä 3. Tuloksen varmentamiseksi kiinnitin oskilloskoopin mittauspäät kehityskortin MOSI- ja SCLK-pinneihin (KUVA 18). Mittaustuloksesta pystyin tekemään tulkinnan, että ohjelmointi ja konfigurointi onnistuivat (KUVA 19).

```
int main(void)
{

  HAL_Init();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_SPI1_Init();
  uint8_t data_to_send = 0x2;  // Lähetettävä data
  uint8_t data_to_send1 = 0x3;  // Lähetettävä data
  while (1)
  {
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);  // Valitse SPI-laite
      HAL_SPI_Transmit(&hspi1, &data_to_send, 1, HAL_MAX_DELAY);  // HAL komento datan (nro 2) lähettämiseen
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);  // Poista SPI-laitteen valinta

      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);  // Valitse SPI-laite
      HAL_SPI_Transmit(&hspi1, &data_to_send1, 1, HAL_MAX_DELAY);  // // HAL komento datan (nro 3) lähettämiseen
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);  // Poista SPI-laitteen valinta
      HAL_Delay(10);
  }
}
```

KUVA 17. Ohjelmakoodi numeroiden lähettämiseksi väylälle



KUVA 18. Oskilloskoopin mittauspäät kiinnitettynä SCLK- ja MOSI- pinneihin

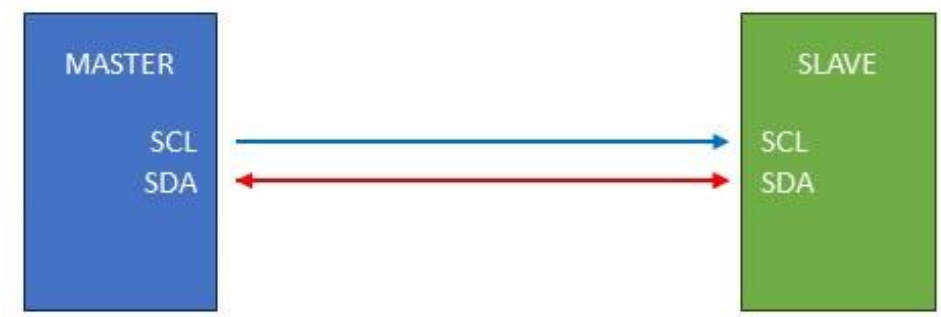


KUVA 19. Oskilloskoopin mittaustulos. Alempana SCLK-signaali

4.5 I2C-väyläohjaus ja LCD-näyttö

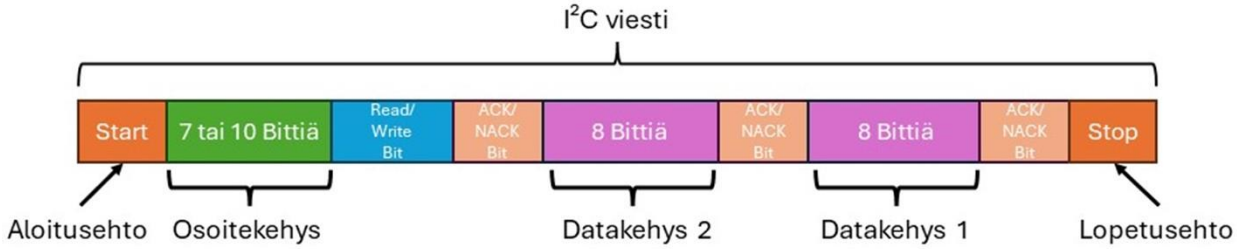
I2C-väyläprotokolla kehitettiin vuonna 1982 Philipsin toimesta. Alkuperäinen versio protokollasta, standard mode, salli käytettävän vain 100kbit/s nopeutta sekä 7-bittisiä osoitteita rajoittaen väylälle liitettävien laitteiden määrän 112:ta. Kun protokolla määriteltiin julkisesti vuonna 1992, oli siihen lisätty 400kbit/s nopeutta tukeva fast-mode ja mahdollisuus käyttää 10-bittisiä osoitteita. (Sparkfun, luku ”A Brief History of I2C”.) Nykyisen määrittelyn mukaan väylä tukee myös 1mbit/s (Fast-mode Plus) ja 3,4mbit/s (High-speed mode) nopeuksia, sekä yksisuuntaisena myös 5mbit/s nopeutta (Ulta-fast mode). Nykyinen väylän enimmäislaitemäärä määräytyy väyläkohtaisen kapasitanssin mukaan. (NXP Semiconductors 2021, 3–4.)

SPI:n tavoin I2C-protokolla on synkroninen, eli isäntälaitteen tuottama kellosignaali tahdistaa laitteiden välisen tiedonsiirron. SPI-väylästä poiketen, I2C-väylällä käytetään vain kahta signaalijohdinta, SDA (Serial Data) ja SCL (Serial Clock), eikä I2C väylällä ei ole erillistä NSS-signaalia, vaan orjalaitteiden valinta tapahtuu osoitteiden avulla. Lisäksi I2C-väylä tukee multi-master ominaisuutta, jossa isäntälaitteita on useita. (Campbell, luku "Introduction to I2C communication".) Väyläprotokollaa hyödynnetään usein sulautetuissa järjestelmissä erilaisten oheislaitteiden, kuten I/O-rajapintojen, EEPROM-muistien ohjauksessa mikrokontrollerin avulla (Hopkins 2020).



KUVIO 5. I2C-tiedonsiirtoprotokollan toimintaperiaate (mukaillen Campbell)

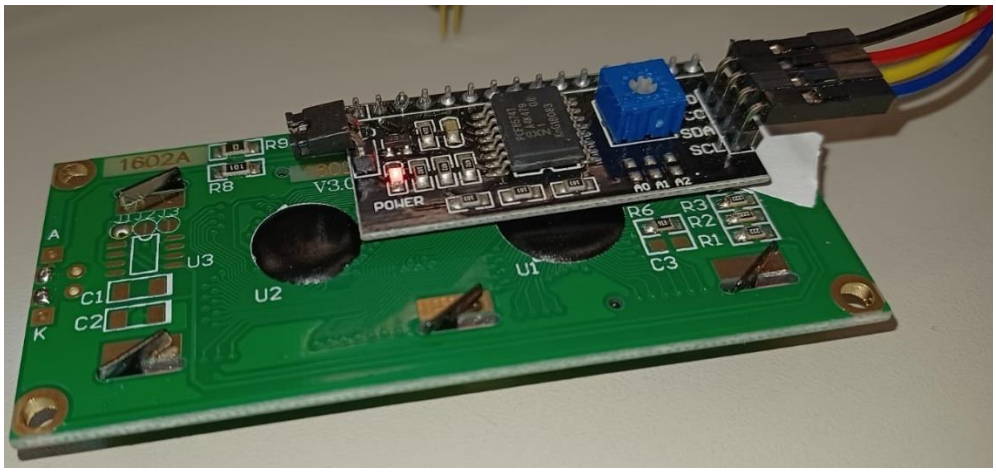
I2C on tarkasti määritelty tiedonsiirtoprotokolla ja siirrettävän datan on täytettävä määritellyt vaatimukset, jotta väylää käyttävät laitteet kykenevät lukemaan sitä (Sparkfun, luku ”Protocol”). Laitteiden välinen kommunikaatio I2C-väylällä tapahtuu viestien avulla. Jokainen viesti koostuu ehdoista, biteistä ja kehyksistä (KUVA 20).



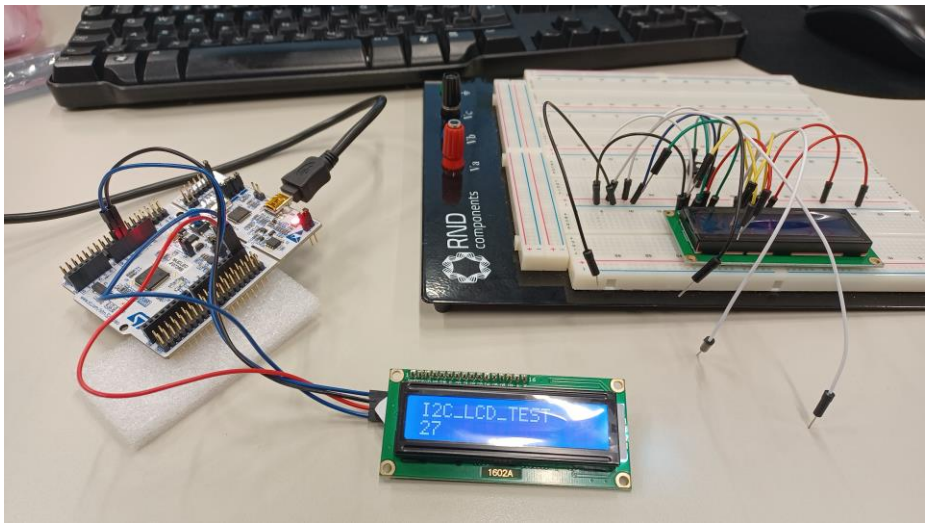
KUVA 20. I2C-viestin rakenne (mukaillen Campbell)

Aloitusehdossa SDA-signaalin jännitetaso vaihtuu HIGH-tilasta LOW-tilaan, ennen kuin SCL-signaali tekee vastaavan HIGH→LOW-tilamuutoksen. Lopetusehdossa SDA:n jännitetaso vaihtuu LOW→HIGH, sen jälkeen, kun SCL:n tila on vaihtunut LOW→HIGH. Osoitekehys sisältää 7 tai 10 bitin sarjan, joka määrittää jokaiselle orjalaitteelle yksilöllisen osoitteen. Read/Write, eli luku- ja kirjoitusbitti täsmentää, lähettääkö isäntälaite dataa orjalaitteelle (LOW-tila), vai pyydetäänkö orjalaitteelta dataa (HIGH-tila). ACK/NACK (acknowledge/not acknowledge), eli kuittausbitti lähetetään jokaisen lähetetyn kehyksen jälkeen ja sen tehtävä on ilmaista lähettäjälle, onko datakehyksen vastaanotto onnistunut. (Campbell, luku ”How I2C works”.) Datakehykset lähetetään merkitsevin bitti (MSB) edellä ja ne koostuvat aina 8 bitistä (Campbell, luku ”The dataframe”).

Tiesin tarvitsevani opinnäytetyöhön näyttöratkaisun ja päätin hyödyntää aiemmin hankitun kokemuksen vuoksi 1602 LCD-näyttöä, mutta päädyin lopulta käyttämään 1602A-versiota, sillä se sisältää kuvassa 21 näkyvän PCF8574 I/O-laajennuspiirin. Laajennuspiiri mahdollistaa LCD-näytön ohjaamisen I2C-väylän kautta, jolloin näytön ohjaamiseen tarvittavien johtimien määrä väheni 16 kappaleesta 4 kappaleeseen (KUVA 22). Myös esimerkkiratkaisuja ja -koodeja oli löydettävissä 1602A-version ohjaamiseen enemmän, lisäksi 12C-väyläprotokollan yleisyys sulautetuissa järjestelmissä kannusti tähän lisää. Hyödynsin LCD-näytön ohjaamisessa löytämiäni esimerkkiratkaisuja, jotka testattuani totesin toimivaksi. STM32CubeMX-ohjelman I2C-väylän konfigurointivalikon asetuksia ei tarvinnut LCD-näytön käyttöä varten muuttaa, vaan pelkät oletusasetukset riittivät.



KUVA 21. PCF8574 I/O-laajennuspiiri

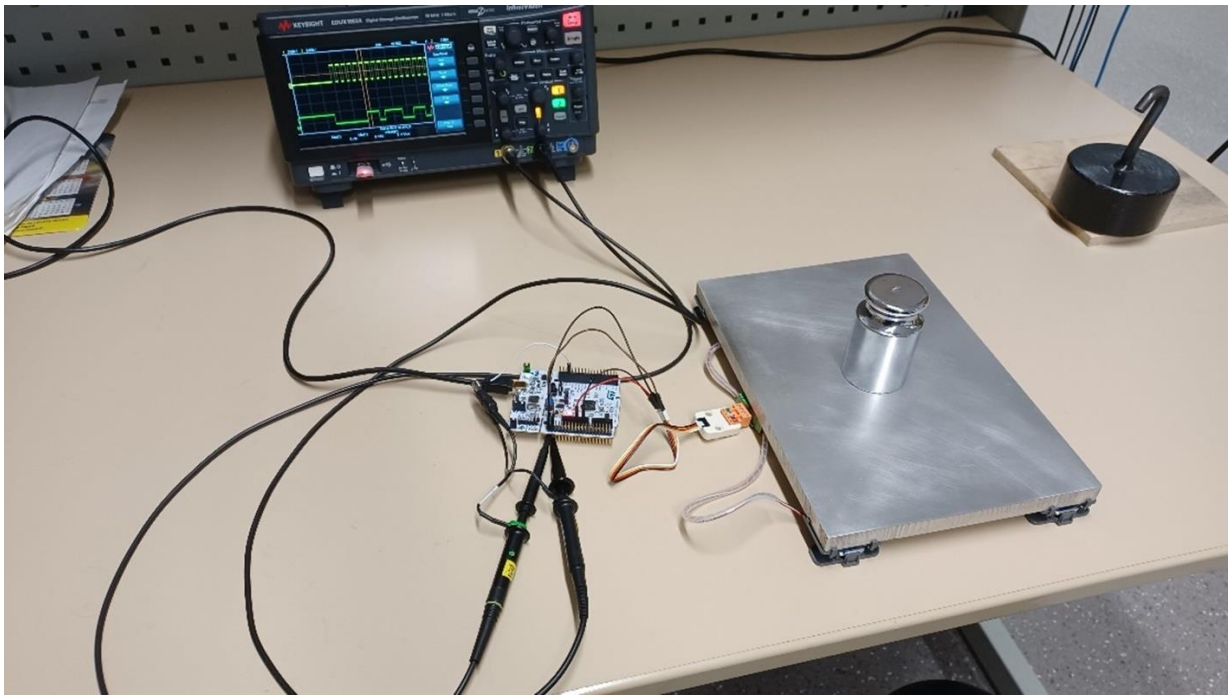


KUVA 22. LCD-näyttö toiminnassa. Taustalla 1602-versio johtoineen

4.6 M5STACK-Vaakayksikkö

Tässä vaiheessa opinnäytetyöprosessia idea osaamista havainnollistavasta laitteesta oli jo selvillä, lisäksi tiedossa oli, että laite tarvitsee vaakayksikön. Koulun laboratoriotiloista löytyi valmiita M5STACK-vaakayksiköitä, jotka sisälsivät neljä venymäliuska-anturiin perustuvaa kuormasolua. Venymäliuska-antureiden analoginen mittaustieto viedään vaakayksikön sisällä olevalle HX711 ADC-piirille, joka muuntaa mittaustiedon digitaaliseen muotoon. Vaakayksikön lähettämän digitaalisen signaalin tulkitseminen mikrokontrollerilla oli kohtalaisen haastava tehtävä, mutta perehtymällä, HX711-piirin datalehteen, sekä erilaisiin esimerkkiratkaisuihin saavutettiin kuitenkin riittävän tarkka lopputulos.

Kun HX711-piirille syötetään 25–27 positiivista kellopulssia PD_SCK porttiin, piiri siirtää datan ulos DOUT portista yksi bitti kerrallaan (MSB ensin), kunnes kaikki 24 bittiä on siirretty. PD_SCK porttiin syötettyjen kellopulssien lukumäärä määrittää, mitä piirin kanavaa käytetään ja millä vahvistuskertoimella. Piirin tuottamat 24 bittiä, ovat 2-komplementtimuodossa. (Avia Semiconductor, 3.) Eli HX711-piirin ohjaaminen mikrokontrollerilla tapahtuu yksinkertaistetusti niin, että mikrokontrolleri tuottaa lähtöportistaan HX711 tarvitsemat 25–27 kellopulssia ja tulkitsee tuloporttiin piirin takaisin lähettämän datan.



KUVA 23. Ensimmäinen oskilloskooppimittaus alumiinipohjalevyn kanssa

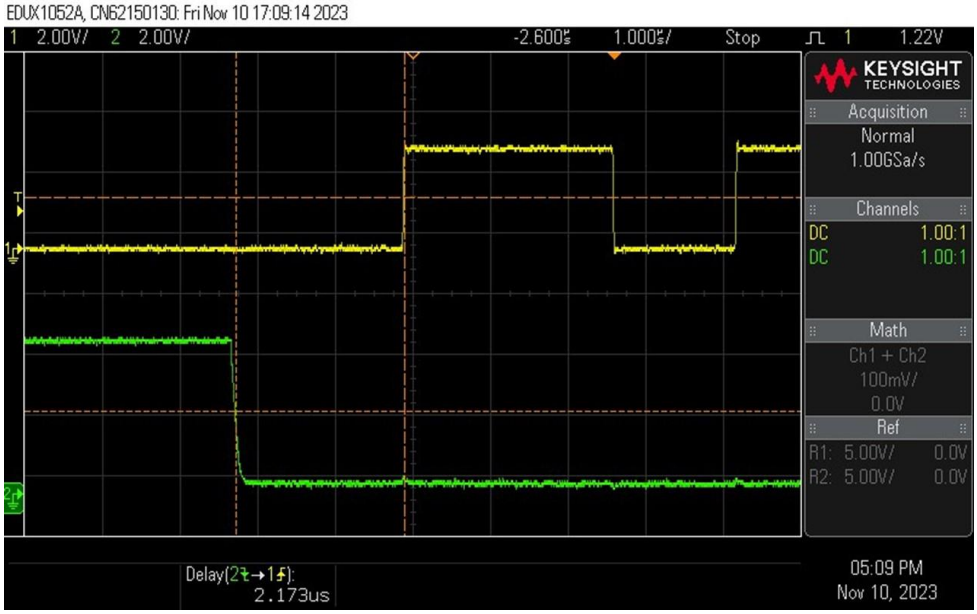
```
//Vaa'an funktio
float readHX711() {

    do {
        while (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_12)) {
            HAL_Delay(10);
        }
        //Kellopulssien lukumäärän säätö (käytössä 27)
        for (i = 0; i < 27; i++) {
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_SET);



            adc_value <<= 1;

            if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_12)) {
                adc_value++;
            }

            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_RESET);


        }

        for (int i = 0; i < 128; i++) {
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_RESET);
        }


        while (adc_value >= 255) {
            adc_value >>= 8;
        }

    } while (0);

    return adc_value / 10.0;
}
```

KUVA 24. Vaakayksikön tulkintaan tarvittava funktio

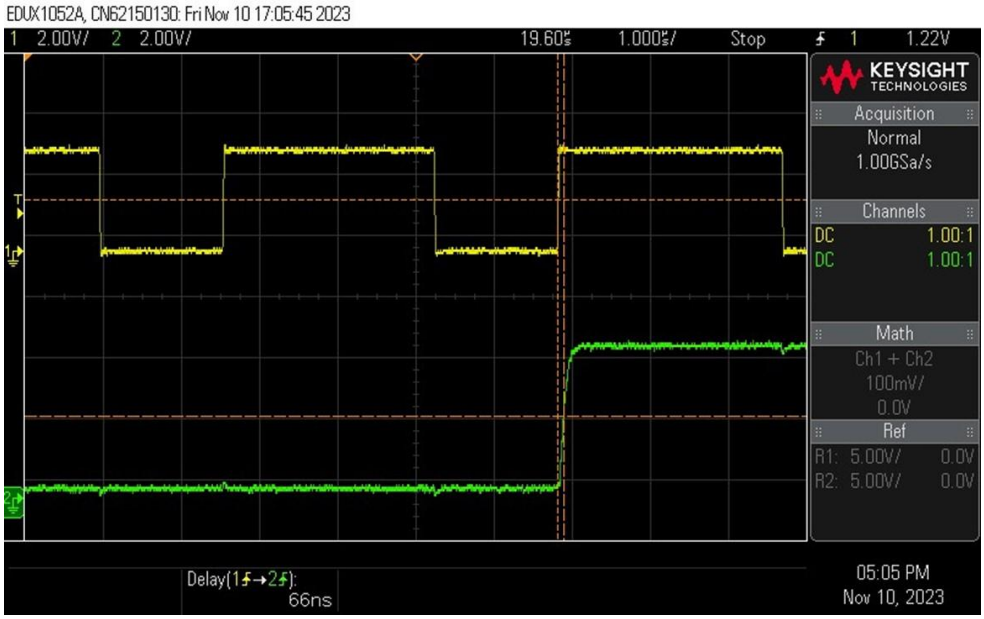
Etsin internetistä esimerkkiratkaisuja, jotka toteuttaisivat nämä toiminnot ja muokkasin löytämäni ohjelmakoodin käyttötarkoituksiini sopivaksi. Lisäksi suoritin piirille oskilloskooppimittaukset, joissa mittasin kello- ja datalinjojen toimintaa. Vertaamalla mittaustuloksia piirin datalehdestä saatuihin arvoihin, pystyin toteamaan, että piirin tuottama data on oikeanlaista. Mittaustulokset on esitetty kuvissa 25, 26 ja 27. Vaa'an lukemien monitoroinnissa oli myös paljon apua STM32CubeIDE:n vianhakutyökalusta, jonka avulla voitiin seurata eri muuttujien tilaa reaaliajassa.

TAULUKKO 3. HX711 piirin ajoitukset (mukaillen Avia Semiconductor, 5)

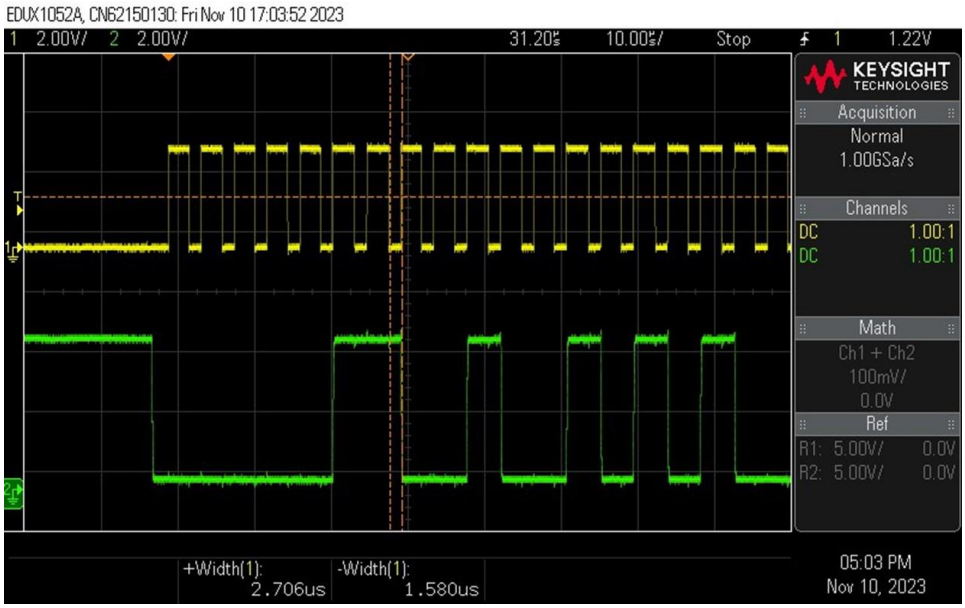
Symbol	Note	MIN	TYP	MAX	Unit
T1	DOUT falling edge to PD_SCK rising edge	0.1			µs
T2	PD_SCK rising edge to DOUT data ready			0.1	µs
T3	PD_SCK high time	0.2	1	50	µs
T4	PD_SCK low time	0.2	1		µs



KUVA 25. T1 mittaustulos

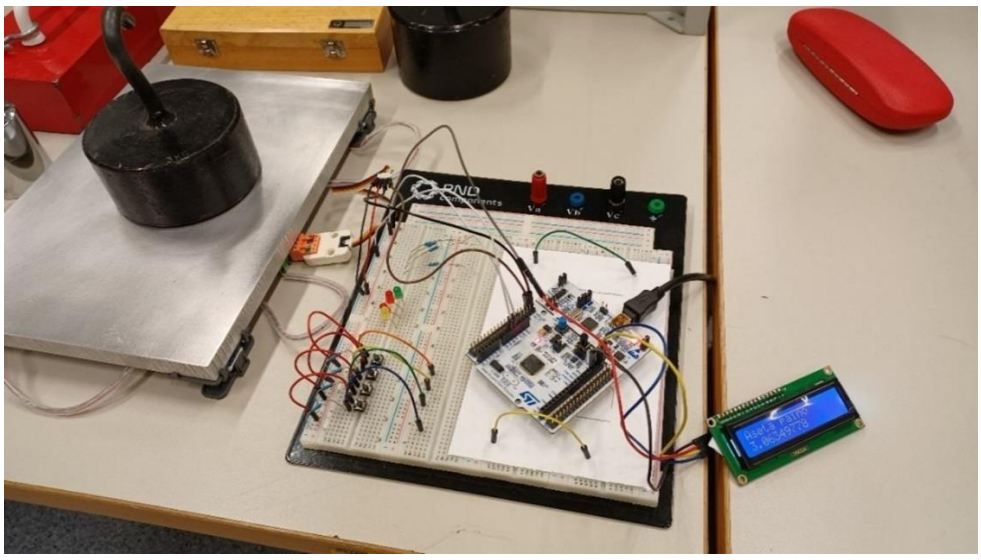


KUVA 26. T2 mittaustulos



KUVA 27. Mittaustulokset T3, T4

Vaakayksikölle kannatti tässä vaiheessa ohjelmoida myös toiminto taaraukselle, sekä määritellä kalibrointikerroin raakadatan muuttamiseksi kilogrammoiksi, sillä nämä ominaisuudet olivat jatkoa ajatellen olennaisia. Kalibrointikertoimen määrittely, sekä taaraus, oli nopeinta toteuttaa yhdistämällä aiemmin toimivaksi todettu LCD-näytön ohjauksessa käytetty koodi vaakayksikön koodin kanssa, tällöin vaa'an lukemia pystyttiin seuraamaan suoraan numeroarvoina ilman PC-laitteistoa. Kalibrointikertoimen määrittelyn suoritin käyttämällä tunnettuja painoja. Lisäsin painon vaa'alle ja merkitsin ylös vaakayksikön tuottaman raakadatan. Jakamalla raakadata tunnetulla painolla, saadaan kerroin. Lopullisen kertoimen muodostin laskemalla saaduista kertoimista mediaanin.



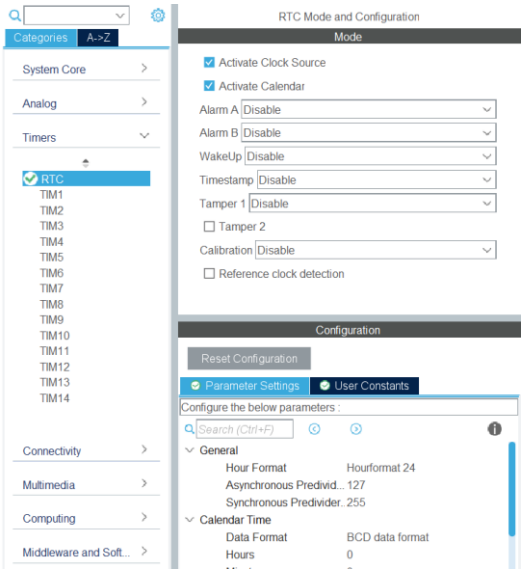
KUVA 28. LCD-näyttö lisätty kytkentään. 3kg punnus

4.7 Muut testatut toiminnot

Opinnäytetyöprosessin aikana testattiin myös sellaisia oheislaitteita ja toimintoja, jotka eivät päätyneet prototyyppilaitteeseen. Tässä luvussa kuvataan mitä nämä oheislaitteet ja toiminnot olivat, sillä näitä toimintoja voi pitää yleishyödyllisinä ja sitä kautta käyttökelpoisina joissakin muissa toteutuksessa, tai esimerkiksi opinnäytetyössäprosessissa syntyneen laitteen jatkojalostuksessa.

4.7.1 RTC

RTC (Real Time Clock), eli reaaliaikainen kello, on itsenäisesti toimiva laskuri/ajastin, jossa on myös kalenteriominaisuus. Useimmat STM32-kehityskorteista on varustettu tällä ominaisuudella. Reaaliaikaisen kellon sisällyttämistä opinnäytetyöhön harkitsin, sillä sen avulla olisi ollut mahdollista toteuttaa erilaisia ajastustoimintoja, lisäksi sen aktivointi oli hyvin yksinkertaista, joten kokeileminen kannatti. Ajastustoimintoja ei kuitenkaan sisällytetty projektiin aikaresurssien säästämisen vuoksi.



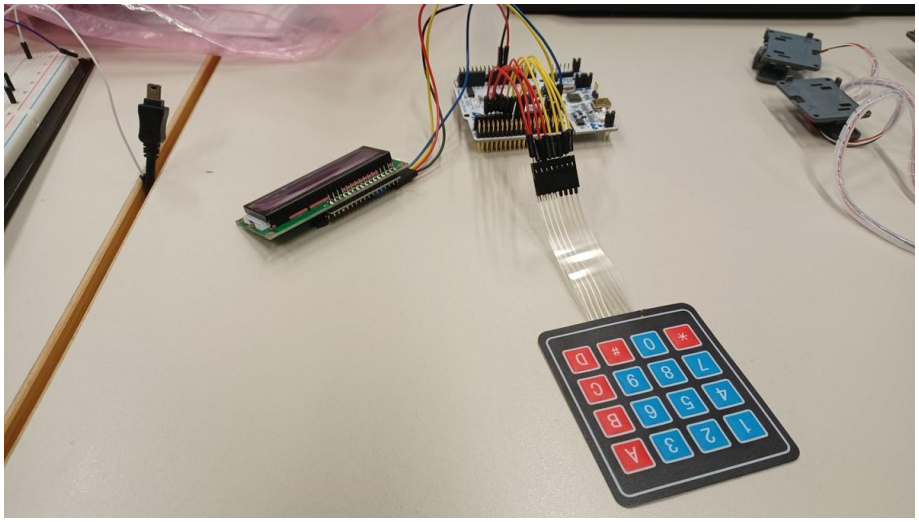
KUVA 29. Reaaliaikaisen kellon aktivointi konfigurointityökalussa

```
  while (1)
  {

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
          get_time();
          lcd_clear();
          lcd_set_cursor(0, 0);
          lcd_write_string(time);
          lcd_set_cursor(1, 0);
          lcd_write_string(date);
          HAL_Delay(1000);
  }
  /* USER CODE END 3 */
}
void get_time()
{
 RTC_DateTypeDef gDate;
 RTC_TimeTypeDef gTime;
//Hae kellonaika
 HAL_RTC_GetTime(&hrtc, &gTime, RTC_FORMAT_BIN);
//Hae päivämäärä
 HAL_RTC_GetDate(&hrtc, &gDate, RTC_FORMAT_BIN);
/* Aikaformaatti: hh:mm:ss */
 sprintf((char*)time,"%02d:%02d:%02d",gTime.Hours, gTime.Minutes, gTime.Seconds);
// Päivämääräformaatti: dd-mm-yy
 sprintf((char*)date,"%02d-%02d-%2d",gDate.Date, gDate.Month, 2000 + gDate.Year);
}
```

KUVA 30. Kellon ja päivämäärän hakuun tehty funktio sekä tulostus LCD näytölle

4.7.2 Näppäimistö

Opinnäytetyön testausvaiheessa käytiin läpi myös vaihtoehtoja käyttöliittymän ohjaamiseksi. Monipuolisimpana valintana pidettiin 4x4 matriisinäppäimistöä (KUVA 31). Ohjauksen toteutus matriisinäppäimistöllä osoittautui kuitenkin luultua haastavammaksi ja luotettavaa tulosta ei saatu aikaan. Näppäimistön tulkinta toimi parhaimmillaankin niin, että LCD-näytölle tulostui satunnaisia numeroita näppäimistön esimerkiksi tärähtäessä tai liikahtaessa, lisäksi usein näppäintä painettaessa sen arvo saattoi näkyä näytöllä useampaan otteeseen. Vikojen arveltiin johtuvan mahdollisista kytkinvärähtelyistä, sillä näppäimistössä ei ollut niitä suodattavia varten suodatuskondensaattoreita. Ajan säästämiseksi katsoin parhaaksi luopua näppäimistön käytöstä, sillä valmiissa laitteistossa käyttöliittymän ohjauksen voi toteuttaa muillakin tavoilla.



KUVA 31. 4x4 Matriisinäppäimistö kytkettynä kehityskorttiin

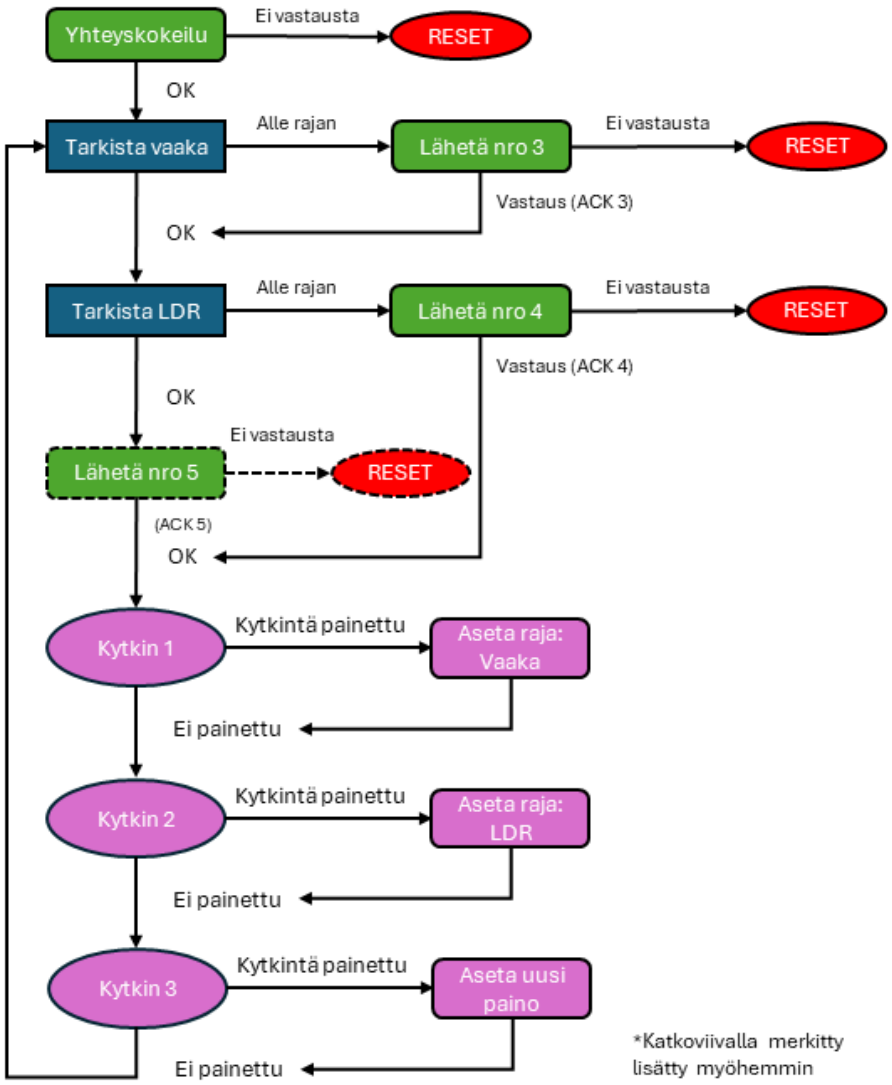
5 KASTELUJÄRJESTELMÄ

Jo opinnäytetyön määrittelyvaiheessa oli selvää, että opituista asioista on pystyttävä muodostamaan toimiva koekytkentälaite, joka vastaisi tasoltaan ammattikorkeakouluopintojen projektityö-painotteisten kurssien laitteita. Inspiraatiota osaamistani havainnollistavaan laitteeseen etsin aluksi internetistä, mutta lopulta päädyin käyttämään omaa ideaani automaattisesta huonekasvien kastelujärjestelmästä. Kastelujärjestelmän oli määrä toimia niin, että laite punnitsee huonekasvin ruukkua, sekä mittaa valon määrää ja mikäli ruukun paino tai valon määrä putoaa alle käyttäjän asettaman rajan, käynnistää laite vesipumpun tai lampun. Laite-ideani kattoi käytännössä kaikki tutkimusongelmassa määrittelemäni, sekä luvussa 4 testaamani ominaisuudet, joten laitteen ohjelman koodaaminen oli suurimmaksi osaksi jo aiemmin testattujen ohjelmien yhdistelyä. Tässä luvussa esittelen vain ne ohjelmakoodin osat, joita en erikseen testannut, tai joihin olen tehnyt testausvaiheen jälkeen merkittäviä päivityksiä. Esittelen tässä luvussa myös SPI-väylän kanssa kohtaamani ongelmat.

Aloitin kastelujärjestelmän suunnittelun hahmottelemalla STM32F446RE-kehityskortin, eli järjestelmän isäntälaitteen ohjelman pääsilmukan toimintaa paperille. Hahmottelun tuloksena syntyi vuokaavioesitys, joka toimi ohjelman perusrunkona (KUVIO 6). Ohjelman koostaminen, sekä laitteen kytkennän tekeminen vuokaavioesityksen ja erillistestattujen toiminnallisuuksien ansiosta tapahtui suoraviivaisesti ja nopeasti. Käytännössä tässä vaiheessa ainoa täysin uusi toiminto, jonka toimintaan minun oli perehdyttävä, oli valovastuksen lukeminen kehityskortin ADC:n avulla. Valovastuksen tulkinta oli kuitenkin yksinkertaista ja tapahtui vain kolmea HAL-funktiota hyödyntäen, STM32CubeMX-työkalun parametriasetusten pysyessä oletusarvoissaan (KUVA 32).

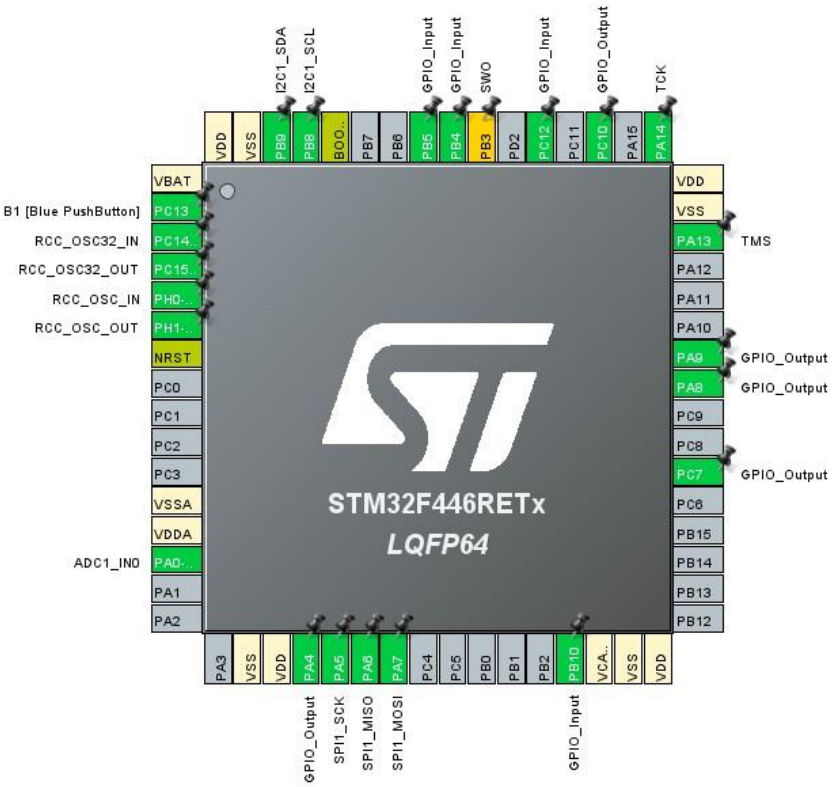
```
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, 1);
LDR_raakadata = HAL_ADC_GetValue(&hadc1);
LDR_mapped = ((LDR_raakadata - LDR_MIN) * 100) / (LDR_MAX - LDR_MIN);
```

KUVA 32. Valovastuksen lukemiseen tarvittavat HAL-komennot



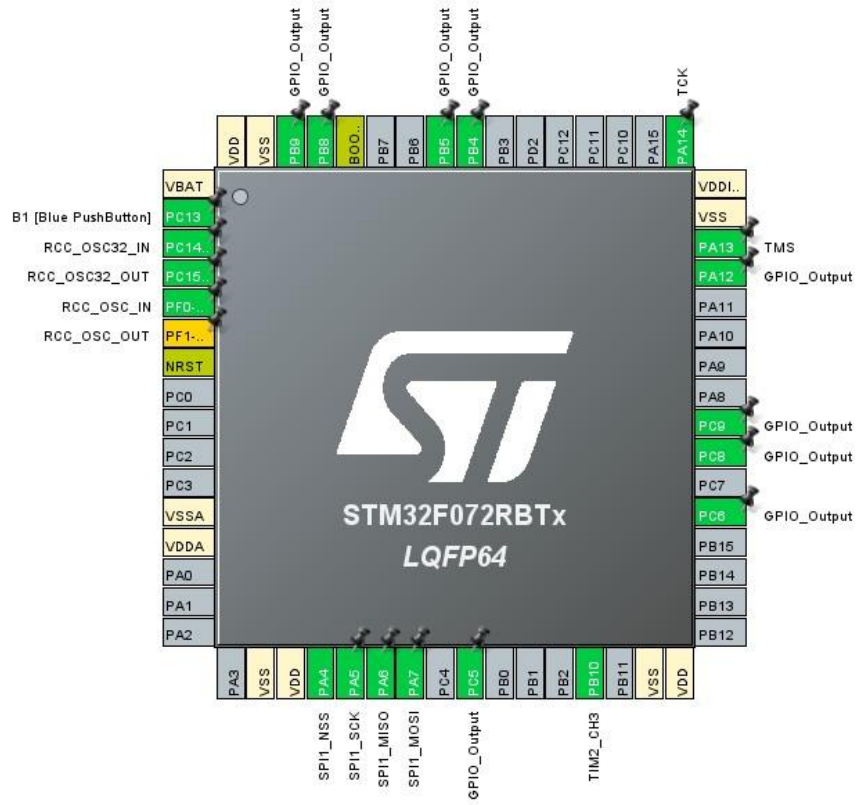
KUVIO 6. Vuokaavioesitys isäntälaitteen toiminnasta

Isäntälaitteen pääsilmukassa suoritettavan koodin päätin toteuttaa tilakoneen avulla. Tilakone toimii kasvattamalla jokaisen toiminnon, esimerkiksi vaa'an lukemisen jälkeen erillisen tilamuuttujan arvoa. Kasvanut tilamuuttujan arvo toimii ehtona seuraavan toiminnon suorittamiseksi. Isäntälaite tarkistaa yhden ohjelmakierron aikana vaakayksikön, sekä LDR-vastuksen tilat ja lähettää käyttäjän asettaman raja-arvon alittuessa orjalaitteelle komennon valon tai pumpun käynnistämistä varten. Ensimmäisellä ohjelmakierrolla suoritetaan myös yhteyskokeilu. Lähetettävä komento on binäärimuotoinen numero-arvo ja lähetyksen jälkeen odotetaan ACK-vastausta, mikäli vastausta ei tule, isäntälaite käynnistää orjalaitteen uudelleen. Jokaisella ohjelmakierrolla tarkistetaan myös, onko raja-arvokytkimiä painettu, mikäli on, siirrytään asettamaan raja-arvo valovastukselle tai vaakayksikölle. Isäntälaitteen ohjelman pääkoodi on nähtävissä liitteessä 3.



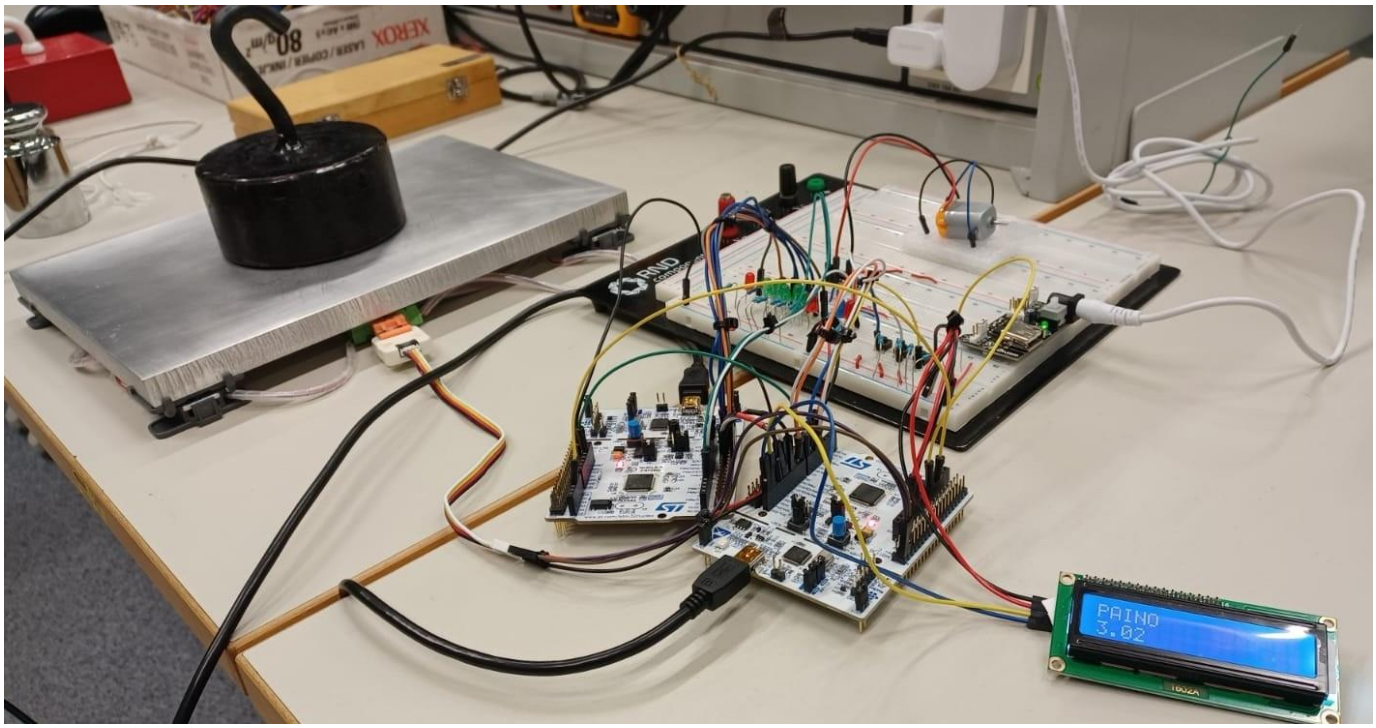
KUVA 33. Isäntälaitteen pinnikonfiguroinnit

Järjestelmän orjalaitteena toimivan STM32F072RB-kehityskortin ohjelma on huomattavasti lyhyempi ja sopivaksi ratkaisuksi sen toteuttamiseksi katsoin ”switch-case”-rakenteen, joka on kokonaisuudessaan nähtävissä liitteessä 2. Pääsilmukassa sijaitsevassa switch-case-rakenteessa tulkitaan isäntälaitteen SPI-väylälle lähettämiä bittejä (numerot 2-5) ja suoritetaan bitin arvoa vastaava toiminto, sekä vastataan isäntälaitteelle ACK-komennolla.



KUVA 34. Orjalaitteen pinnien konfigurointi

Muutin lopuksi laitteen toimintaa alkuperäisestä vuokaavioesityksestä niin, että jokaisella ohjelmakierrolla isäntä-kehityskortti lähettää tiedon myös siitä, jos valovastuksen arvo ei ole alle sallitun raja-arvon. Tämä muutos on oleellinen siksi, että mikäli valovastuksen raja-arvo alitetaan ja orjalaitteelle lähetetään käsky valon sytyttämisestä, on valo saatava myös sammumaan. Muut koodeihin tekemäni muutokset olivat pääasiassa esteettisiä, eli sisällytin toistuvia ohjelmarakenteita funktioiden sisälle, sillä tämä lyhensi ohjelman pääsilmukan pituutta merkittävästi ja paransi luettavuutta. Laitteen vaaka näyttää lukemat noin 0.5 kg tarkkuudella, tarkkuutta yritin lisätä työn loppuvaiheilla ohjelmallisesti muokkaamalla toimintaa niin, että näytettävä lukema koostetaan usean mittauksen mediaanista. Sain mediaaniin perustuvan mittauksen funktion ohjelmoitua, mutta mittaustulokset eivät pitäneet paikkaansa. Valitettavasti aika ei riittänyt toiminnan parantamiseen ja laitteen riittävään testaamiseen, joten päädyin tyytymään saavutettuun tarkkuuteen, joka kuitenkin on käyttötarkoitus huomioon ottaen riittävä.



KUVA 35. Valmis koekytkentälaite. 3kg punnus

Eniten haasteita kastelujärjestelmän rakennusprosessissa tuotti kehityskortit yhdistävä SPI-väylä. Ensimmäisillä testauskerroilla isäntä- ja orja-kehityskorttien välinen kommunikaatio ei toiminut luotettavasti. Selkeyttääkseni itselleni ongelman laatua lisäsin kytkentään molemmille kehityskorteille LED-valoja, jotka ohjelmoin syttymään ja sammumaan laitteiden vastaanottaessa ja lähettäessä dataa. Ohjelmoin myös isäntälaitteelle debug, eli vianhaku-printtauksia, jotka tulostivat orjalaitteelta vastaanotetun datan LCD-näytölle, sekä näyttivät, mikäli orjalaite olisi resetoitu.

Vianhakua tehostavat toimenpiteet auttoivat ja sain paikallistettua yhden ongelman, olin epähuomiossa ohjelmoinut myös orjalaitteen ohjelman pääsilmukkaan tilakonerakenteen ja unohtanut lisätä loppuun tilamuuttujan nollauksen, joka olisi palauttanut tilakoneen takaisin alkutilaansa. Tämä aiheutti sen, että orjalaite ei kyennyt vastaanottamaan kuin yhden käskyn kerrallaan. Useamman käskyn vastaanotto onnistui niissä tilanteissa, kun isäntälaite oli resetoinut orjalaitteen, sillä resetointi palautti tilakoneen alkutilaansa. Ongelma ratkesi poistamalla turhaksi osoittautunut väärin koodattu tilakone ohjelmasta.

Seuraava ongelma kommunikoinnissa oli se, että kun laitteisto oli toiminut muutaman ohjelmakierron verran virheettömästi, alkoivat orjalaitteen vastaanottoa indikoivat LED-valot vilkkua väärällä hetkellä ja laite suoritti toimintoja omatoimisesti, vaikka isäntälaitteelta ei olisikaan lähetetty väylälle käskyjä.

Tämän ongelman ratkaisuun kului paljon aikaa ja etsin vikaa useasta paikasta. Säädin esimerkiksi käskyjen lähetysten ajoituksia, SPI-väylän nopeutta ja vaihdoin väylällä käyttämäni johtimet lyhyimpiin mahdollisiin. Lopulta myös mittasin väylää oskilloskoopilla ja huomasin, että virheelliset käskyt vaikuttivat olevan häiriöpiikkejä, jotka orja-kehityskortti onnistui tulkitsemaan isäntä-kehityskortin lähettämiksi käskyiksi. Ongelma ratkesi aktivoimalla orja-kehityskortin SPI-konfiguraatiovalikosta Hardware NSS- eli Slave Select-signaalin. Tällöin laite kykenee ottamaan käskyjä vastaan vain, mikäli NSS-signaali on isäntälaitteen toimesta asetettu LOW-tilaan. Isäntälaitteen pääkoodiin lisäsin koodirivit, jotka aktivoivat NSS-signaalin aina, ennen kuin käsky lähetetään. Nämä muutokset korjasivat yhteysongelmat.

6 YHTEENVETO JA POHDINTA

Opinnäytetyön aiheena oli tutkia STM32-kehityskortteja ja -ohjelmistoja ja selvittää kuinka vaikeaa yksinkertaisen perusohjauksista koostuvan laitekokonaisuuden toteuttaminen tässä ympäristössä on. Lisäksi oppimisprosessin kirjallisen kuvauksen tulisi toimia lukijalleen käytännönläheisenä aloitusoppaana STM32-kehitysalustalle. Opinnäytetyön tavoitteisiin päästiin, vaikka SPI-väylän kanssa kohdatut yhteysongelmat tuottivatkin haasteita. Vaakayksikön lisäämistä lopulliseen laitekokonaisuuteen ei suunniteltu etukäteen ja aiemman kokemuksen puutteen vuoksi selvitystyö vaati vaivannäköä. Lopputuloksen voi kuitenkin vaa'an tarkkuuden osalta, lähtökohdat huomioon ottaen, katsoa olevan riittävän hyvä.

Opinnäytetyön aloitusvaiheessa STM32-kehitysalustan toiminnan opiskelu vei aikaa. Verrattuna esimerkiksi Arduinon kehitysalustan toimintaan, STM32 ei juuri peitä ohjelmoitaessa mikrokontrollerin toimintaa helppokäyttöisyyden taakse. Mikrokontrollerin näkökulmasta tämä keventää suoritettavaa koodia, mutta ohjelmoijan näkökulmasta tämä tuo paljon lisätyötä, sillä yksinkertaisenkin toiminnon ohjelmointi voi vaatia useita erilaisia konfigurointeja sekä koodirivejä. Juuri tästä syystä STM32-kehitysalustan ei voi katsoa soveltuvan mikrokontrollereiden ensimmäisten perusteiden opiskeluun. Mikäli pohjatietona on esimerkiksi kokemusta aloittelijaystävällisemmästä Arduinosta, sopii kehitysalusta varmasti seuraavaksi askeleeksi sulautetuista järjestelmistä kiinnostuneelle.

Kirjallisesta työstä suoriuduttiin hyvin, huolimatta siitä positiivisesta ongelmasta, että kirjoittajan työllisyystilanne parani merkittävästi kirjoitusprosessin alkuvaiheilla. Tämä vähensi luonnollisesti kirjoittamiseen käytettävää aikaa ja siirsi työskentelyn pääasiassa iltoihin ja viikonloppuihin. Erityisen toimivaksi ratkaisuksi voi katsoa oppimispäiväkirjan pitämisen opinnäytetyön määrittelyvaiheesta saakka, sillä päiväkirjan avulla niin toiminnallisen, kuin kirjallisenkin osion tavoitteet pysyivät alusta loppuun saakka selkeinä.

Kastelujärjestelmän jatkojalostukseksi elektroniikkasuunnittelijana työskentelevä kirjoittaja on haaveillut suunnittelevansa piirilevyn, johon kehityskortit ja osa oheislaitteista voidaan liittää. Piirilevyratkaisu poistaa laitteesta paljon epävarmoja johtoliitoksia sekä mahdollistaa varmemman ja vesivahinkovapaamman kenttätestauksen esimerkiksi olohuoneessa. Ohjelmallisen jatkojalostuksen voisi aloittaa siitä, mihin opinnäytetyössä jäätiin, eli mediaaniin perustuvan mittaustiedon lisäämiseen.

LÄHTEET

AVIA Semiconductor. HX711. Saatavissa: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf Viitattu 15.2.2024.

Campbell, S. Basics of the I2C communication protocol. Circuit Basics. Saatavissa: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/ Viitattu 7.2.2024.

Campbell, S. Basics of the SPI communication protocol. Circuit Basics. Saatavissa: https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/ Viitattu 3.2.2024.

Colley, S. 2020. Pulse-width Modulation (PWM) Timers in Microcontrollers. All about circuits. Saatavissa: https://www.allaboutcircuits.com/technical-articles/introduction-to-microcontroller-timers-pwm-timers/ Viitattu 1.2.2024.

Dietrich, S. 2020. Understanding the Basics of Pulse Width Modulation (PWM). Control automation. Saatavissa: https://control.com/technical-articles/understanding-the-basicsof-pulse-width-modulation-pwm/ Viitattu 1.2.2024.

Digikey. 2020. Understanding STM32 Naming Conventions. Saatavissa: https://www.digikey.fi/en/maker/tutorials/2020/understanding-stm32-naming-conventions Viitattu 23.10.2023.

Fastbitlab. 2019. STM32 SPI Lecture 6: SPI bus configuration discussion: Full duplex, Half-duplex, and Simplex. Fastbitlab. Saatavissa: https://fastbitlab.com/spi-bus-configuration-discussion-full-duplex-half-duplex-simplex/ Viitattu 3.2.2024.

Hopkins, J. 2020. Understanding the Differences Between UART and I2C. Total Phase. Saatavissa: https://www.totalphase.com/blog/2020/12/differences-between-uart-i2c/ Viitattu 9.2.2024.

Integrated Development Environment for STM32. Saatavissa: https://www.st.com/en/development-tools/stm32cubeide.html Viitattu 6.11.2023.

McCreary, D. 2020. SPI – What It Is, How It Works, and What It Means For You. Netburner. Saatavissa: https://www.netburner.com/learn/spi-what-it-is-how-it-works/ Viitattu 2.2.2024.

NXP Semiconductors. 2021. UM10204. Saatavissa: https://www.nxp.com/docs/en/user-guide/UM10204.pdf Viitattu 5.2.2024.

Posch, M. 2021. Blue Pill vs Black Pill: Transitioning from STM32F103 to STM32F411. Hackaday. Saatavissa: https://hackaday.com/2021/01/20/blue-pill-vs-black-pill-transitioning-from-stm32f103-to-stm32f411/ Viitattu 30.10.2024.

Sparkfun. 2013. I2C. Saatavissa: https://learn.sparkfun.com/tutorials/i2c/all Viitattu 7.2.2024.

SPI Background. Total Phase. Saatavissa: https://www.totalphase.com/support/articles/200349236-spi-background/ Viitattu 2.2.2024.

SPI protocol. Javapoint. Saatavissa: https://www.javatpoint.com/spi-protocol Viitattu 3.2.2024.

STM32 Discovery Kits. Saatavissa: https://www.st.com/en/evaluation-tools/stm32-discovery-kits.html Viitattu 30.10.2024.

STM32 Eval Boards. Saatavissa: https://www.st.com/en/evaluation-tools/stm32-eval-boards.html Viitattu 30.10.2024.

STM32 Nucleo Boards. Saatavissa: https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html Viitattu 30.10.2024.

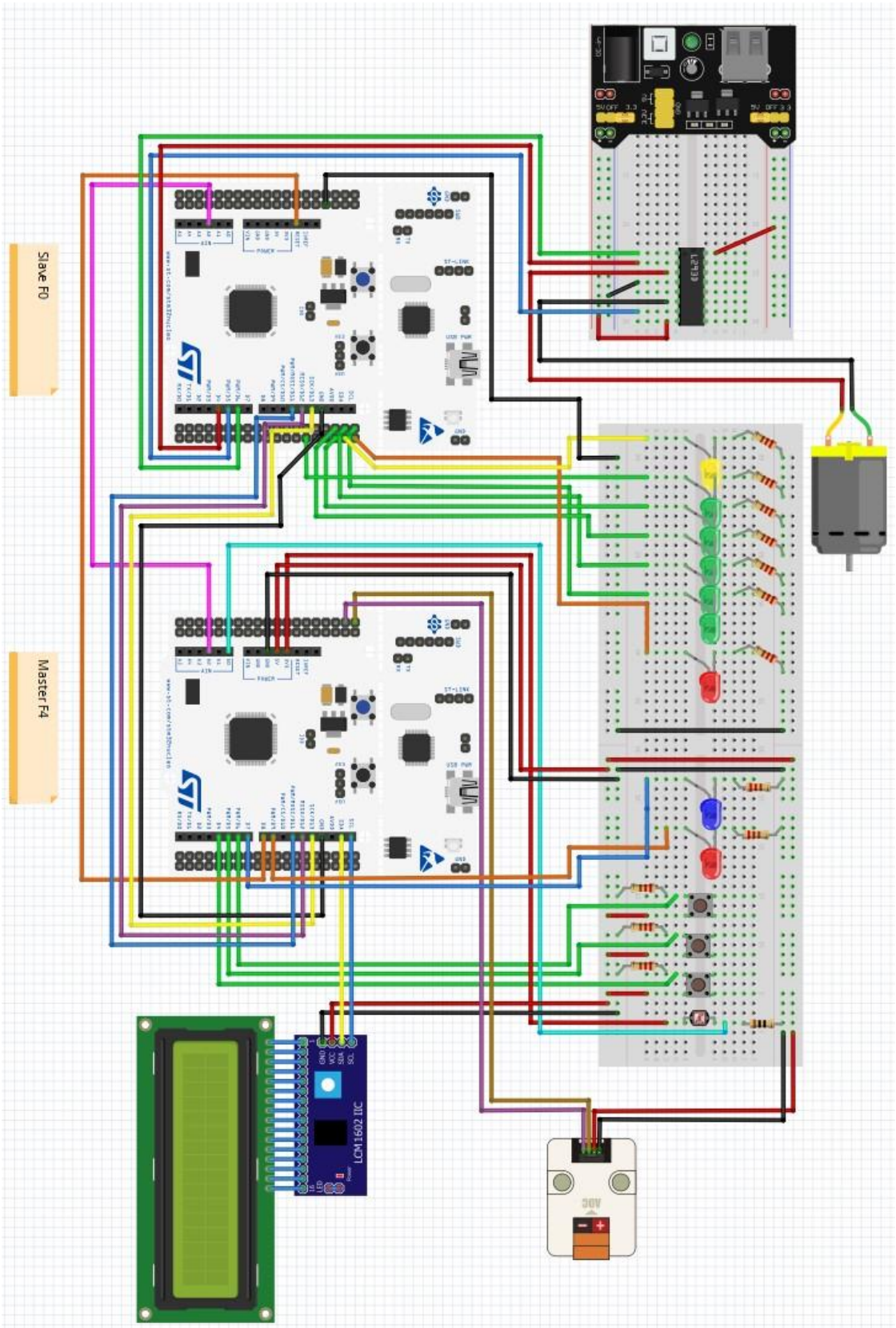
STM32 Software Development Tools. Saatavissa: https://www.st.com/en/development-tools/stm32-software-development-tools.html Viitattu 6.11.2023.

STM32Cube initialization code generator. Saatavissa: https://www.st.com/en/development-tools/stm32cubemx.html Viitattu 6.11.2023.

STMicroelectronics. 2020. UM1785. Saatavissa: https://www.st.com/resource/en/user_manual/um1785-description-of-stm32f0-hal-and-lowlayer-drivers-stmicroelectronics.pdf Viitattu 5.1.2024

LIITE 1

Kastelujärjestelmän kytkentäkuva



LIITE 2/1

Orjalaitteen pääsilmukan koodi

```
while (1)
{
/* USER CODE BEGIN 3 */

// Aloitetaan datan luku SPI-väylältä, jos NSS-signaali on vedetty LOW-tilaan. Vastaanotetun datan perusteella suoritetaan vastaanotto-LED funktio
 if (HAL_GPIO_ReadPin(GPIOA, NSS_PIN) == GPIO_PIN_RESET){

HAL_SPI_Receive(&hspi1, &received_data, 1, 2);

if (received_data == 2) {
VastaanottoLED_BIT2();
//1s viive, joka syntyy LED-valon vilkutuksesta
}
if (received_data == 3) {
VastaanottoLED_BIT3();
//1s
}
if (received_data == 4) {
VastaanottoLED_BIT4();
//1s
}
if (received_data == 5) {
VastaanottoLED_BIT5();
//1s
}

//Switch-case rakenne, jossa vastaanotetun datan arvon (2-5) perusteella suoritetaan jokin tietty toiminto

switch (received_data)
{
case 2: //Yhteyskokeilu
LähetysLED();
1s

HAL_SPI_Transmit(&hspi1, &ack_bit_YK, 1, timeOut);
received_data = 0;
// State = 0;
break;

case 3://Pumppu
HAL_GPIO_WritePin(DIR_PORT, DIR_PIN, RESET);
HAL_GPIO_WritePin(DIR1_PORT, DIR1_PIN, SET);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);
```

```
HAL_Delay(1000);
HAL_GPIO_WritePin(DIR_PORT, DIR_PIN, RESET);

HAL_GPIO_WritePin(DIR1_PORT, DIR1_PIN, RESET);
HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_3);

LähetysLED();
//1s

HAL_SPI_Transmit(&hspi1, &ack_bit_PUMP, 1, timeOut);
received_data = 0;
break;

case 4: //LDR-arvo alitettu, käynnistä LED
HAL_GPIO_WritePin(LED2_PORT, LED2_PIN, SET);
HAL_Delay(1000);

LähetysLED();
//1s

HAL_SPI_Transmit(&hspi1, &ack_bit_LED, 1, timeOut);
received_data = 0;
// State = 0;
break;

case 5:
//LDR-arvo ylitetty, sammuta LED
HAL_GPIO_WritePin(LED2_PORT, LED2_PIN, RESET);
HAL_Delay(1000);

LähetysLED();
//1s

HAL_SPI_Transmit(&hspi1, &ack_bit_LED2, 1, timeOut);
received_data = 0;
break;

default:
break;
}

// Vastaanotetun datan resetointi seuraavaa ohjelmakiertoa varten
received_data = 0;
}
  }
  /* USER CODE END 3 */
}
```

LIITE 3/1

Isäntälaitteen pääsilmukan koodi

```
while (1)
  {
//////////////YK//////////////
//Testataan yhteys slave-laitteeseen
//Jos YK=OK, siirrytään eteenpäin, jos YK=NOK, resetoidaan slave-laite
//Tilamuuttuja pitää huolen että YK suoritetaan vain kerran, käynnistyksen yhteydessä
//Reset on asetettava HIGH-tilaan (NRST)
HAL_GPIO_WritePin(GPIOA, RST_PIN, GPIO_PIN_SET);

if (State == 0) {
//YK
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_yk); //"YK"
HAL_Delay(3000);

LähetysLED();
YK_SENT = 1; //Asetetaan lippumuuttuja arvoon 1
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); //SLAVE SELECT
HAL_Delay(timeout_NSS);
HAL_SPI_Transmit(&hspi1, &data_to_send, 1, timeout_transmit); // HAL komento datan (nro 2)
lähettämiseen

HAL_Delay(3000); //Viive jonka aikana slaven on vastattava
HAL_SPI_Receive(&hspi1, &received_data, 1, timeout_receive); //Vastaanotto slavelta
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); //SLAVE SELECT

//Jos YK=OK, väläytä LED-valoa, aseta tilamuuttujan arvoon +1, jos ei, resetoi slave-laite
if (YK_SENT == 1){ //Lippumuuttuja, jos mitään ei ole lähetetty, ei tutkita onko mitään vastaanotettu
if (received_data == 2) {
VastaanottoLED();
debugprint_YK();
received_data = 0;
YK_SENT = 0;
}

else            {
ResetSlave();
debugprint_else_YK();
                }
                }
                }
State = 2;
//////////////YK//////////////
```

LIITE 3/2

```
//////////////Tarkista vaaka//////////////
//Tarkistetaan vaa'an lukema, verrataan sitä raja-arvoon, tulostetaan arvo LCD-näytölle
//Jos lukema alle raja-arvon, lähetetään slave-laitteelle BIT X, odotetaan ACK X ja siirrytään eteen-
päin
//Jos ACK X ei vastaanoteta, resetoidaan slave-laite
if (State == 2) {

value = readHX711();

//Raakadatan käsittely
VaakaLukema = (value - TareValue)  / 0.3639372; /// 0.0979273;
adc_value = 0;

print_vaaka_while(); //"PAINO"

//Vaa'an raja-arvo saadaan kertomalla ruukun lähtöpainoa käyttäjän asettamalla prosenttiluvulla (ole-
tuksena 25%)
//Mikäli mitattu paino on pienempi kuin raja-arvo, lähetetään hälytys slavelle ja slave kytkee pumpun
päälle
Vaaka_limit = Alkupaino * Vaaka_prosentti;

if (VaakaLukema < Vaaka_limit)          {
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_vaakahaly); //"ALERT: VAAKA"
HAL_Delay(2000);

LähetysLED();
PUMP_SENT = 1;
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
HAL_Delay(timeout_NSS);
HAL_SPI_Transmit(&hspi1, &pump_bit, 1, timeout_transmit); // HAL komento datan (nro 3) lähettä-
miseen
}

HAL_Delay(3000); //Viive jonka aikana slaven on vastattava
HAL_SPI_Receive(&hspi1, &ACK1, 1, timeout_receive); //Vastaanotto slavelta
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); //SLAVE SELECT

if (PUMP_SENT == 1){
if (ACK1 == 3) {
VastaanottoLED();
debugprint_vaaka();
ACK1 = 0;
PUMP_SENT = 0;
}
```

```
else            {

ResetSlave();
debugprint_else_vaaka();
                }
                }
                }
State = 3;
//////////////Tarkista vaaka//////////////
//////////////Tarkista LDR-vastus//////////////
//Tarkistetaan LDR-vastuksen lukema, verrataan sitä limit-muuttujan arvoon, tulostetaan arvo LCD-näytölle
//Jos lukema alle raja-arvon, lähetetään slave-laitteelle BIT Y, odotetaan ACK Y ja siirrytään eteen-päin
//Jos ACK Y ei vastaanoteta, resetoidaan slave-laite
if (State == 3) {

HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, 1);
LDR_raakadata = HAL_ADC_GetValue(&hadc1);
LDR_mapped = ((LDR_raakadata - LDR_MIN) * 100) / (LDR_MAX - LDR_MIN);

//Valon määrän printtaus
sprintf(uint_to_str2, "%d", LDR_mapped);
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_valonmaara);
lcd_set_cursor(1, 0);
lcd_write_string(uint_to_str2);
HAL_Delay(3000);
//Valon määrän printtaus

if (LDR_mapped < LDR_limit) {

lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_ldrhaly); //ALERT: LDR
HAL_Delay(3000); //Alkuperäinen 5000ms

LähetysLED();
LED_SENT = 1;
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
HAL_Delay(timeout_NSS);
HAL_SPI_Transmit(&hspi1, &led_bit, 1, timeout_transmit); // HAL komento datan (nro 4) lähettämiseen
}
```

```
HAL_Delay(3000); //Viive jonka aikana slaven on vastattava
HAL_SPI_Receive(&hspi1, &ACK2, 1, timeout_receive); //Vastaanotto slavelta
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); //SLAVE SELECT

if (LED_SENT == 1) {
if (ACK2 == 4) {
VastaanottoLED();
debugprint_ldr1();
ACK2 = 0;
LED_SENT = 0;
}

else            {
ResetSlave();
debugprint_else_ldr1();
                }
                }

if (LDR_mapped >= LDR_limit) {

lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_ldr_OK); //"LDR OK"
HAL_Delay(3000); //Alkuperäinen 5000ms

LähetysLED();
LED2_SENT = 1;
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
HAL_Delay(timeout_NSS);
HAL_SPI_Transmit(&hspi1, &led_bit2, 1, timeout_transmit); // HAL komento datan (nro 5)
lähettämiseen
}

HAL_Delay(3000); //Viive jonka aikana slaven on vastattava
HAL_SPI_Receive(&hspi1, &ACK3, 1, timeout_receive); //Vastaanotto slavelta
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); //SLAVE SELECT

if (LED2_SENT == 1){
if (ACK3 == 5) {
VastaanottoLED();
debugprint_ldr2();
ACK3 = 0;
LED2_SENT = 0;
                                                    }
else            {
ResetSlave();
debugprint_else_ldr2();
```

```
                }
                }
                }
State = 4;
//////////////Tarkista LDR-vastus//////////////

//////////////Raja-arvovalikko: Vaaka//////////////
//Tarkistetaan onko kytkintä 1 painettu
//Jos kytkintä 1 painettu, siirrytään näyttämään raja-arvo vaa'alle
//Jos kytkintä 1 ei ole painettu, siirrytään eteenpäin
if (HAL_GPIO_ReadPin(GPIOB, LIMIT1_PIN) == GPIO_PIN_SET && State == 4){
//Raja-arvo: Vaaka
sprintf(float_to_str2, "%.0f", Vaaka_prosentti*100);
lcd_clear();
_set_cursor(0, 0);
lcd_write_string(text_rajavaaka);
lcd_set_cursor(1, 0);
lcd_write_string(float_to_str2);
HAL_Delay(3000);
//Raja-arvo: Vaaka

Jos nappia painetaan toisen kerran, siirrytään asettamaan raja-arvo
if (HAL_GPIO_ReadPin(GPIOB, LIMIT1_PIN) == GPIO_PIN_SET){
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_nappi);
lcd_set_cursor(1, 0);
lcd_write_string(text_aseta);
HAL_Delay(2000);

//Jos nappia painetaan kolmannen kerran, raja-arvolaskuri lähtee etenemään arvosta 0.25
if (HAL_GPIO_ReadPin(GPIOB, LIMIT1_PIN) == GPIO_PIN_SET){
Vaaka_prosentti = 0.25;
print_vaaka_raja_arvo();

}
//0.5
if (HAL_GPIO_ReadPin(GPIOB, LIMIT1_PIN) == GPIO_PIN_SET){
Vaaka_prosentti +=0.25;
print_vaaka_raja_arvo();

}
//0.75
if (HAL_GPIO_ReadPin(GPIOB, LIMIT1_PIN) == GPIO_PIN_SET){
Vaaka_prosentti +=0.25;
print_vaaka_raja_arvo();
```

```
}

//Näytetään mikä arvo asetettu
sprintf(float_to_str2, "%.0f", Vaaka_prosentti*100);
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_asetettu);
lcd_set_cursor(1, 0);
lcd_write_string(float_to_str2);
HAL_Delay(3000);
}
}

else            {
State = 5;
}

//////////////Raja-arvovalikko: LDR//////////////
//Tarkistetaan onko kytkintä 2 painettu
//Jos kytkintä 2 painettu, siirrytään asettamaan raja-arvo LDR-vastukselle
//Jos kytkintä 2 ei ole painettu, siirrytään eteenpäin
if (HAL_GPIO_ReadPin(GPIOB, LIMIT2_PIN) == GPIO_PIN_SET && State == 5){
//Raja-arvo: LDR
sprintf(int_to_str, "%d", LDR_limit);
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_rajaldr
lcd_set_cursor(1, 0);
lcd_write_string(int_to_str);
HAL_Delay(2000);
//Raja-arvo: LDR


//Jos nappia painetaan toisen kerran, siirrytään asettamaan raja-arvo
if (HAL_GPIO_ReadPin(GPIOB, LIMIT2_PIN) == GPIO_PIN_SET){
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_nappi);
lcd_set_cursor(1, 0);
lcd_write_string(text_aseta);
HAL_Delay(2000);

//Jos nappia painetaan kolmannen kerran, raja-arvolaskuri lähtee etenemään arvosta 0.25
if (HAL_GPIO_ReadPin(GPIOB, LIMIT2_PIN) == GPIO_PIN_SET){
LDR_limit = 25;
print_ldr_raja_arvo();
}
```

```
//0.5
if (HAL_GPIO_ReadPin(GPIOB, LIMIT2_PIN) == GPIO_PIN_SET){
LDR_limit +=25;
print_ldr_raja_arvo();
}

//0.75
if (HAL_GPIO_ReadPin(GPIOB, LIMIT2_PIN) == GPIO_PIN_SET){
LDR_limit +=25;
print_ldr_raja_arvo();
}

//Näytetään mikä arvo asetettu
sprintf(float_to_str2, "%d", LDR_limit);
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_asetettu);
lcd_set_cursor(1, 0);
lcd_write_string(int_to_str);
HAL_Delay(3000);
}
}

else            {
State = 6;
}

//////////////Taarausvalikko//////////////
//Tarkistetaan onko kytkintä 3 painettu
//Jos kytkintä 3 painettu, siirrytään asettamaan uusi paino
//Jos kytkintä 3 ei ole painettu, siirrytään eteenpäin
if (HAL_GPIO_ReadPin(GPIOB, TARE_PIN) == GPIO_PIN_SET && State == 6){
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_nappi);
lcd_set_cursor(1, 0);
lcd_write_string(text_uusi_paino);
HAL_Delay(2000);

for (int lkm = 10; lkm >= 1; lkm--) {
//Aseta paino
sprintf(int_to_str2, "%d", lkm);
print_vaaka_aloituspaino();
//Aseta paino
}

value = readHX711();
```

```
Alkupaino = (value - TareValue) / 0.3639372;
adc_value = 0;

sprintf(float_to_str2, "%.2f", Alkupaino);
lcd_clear();
lcd_set_cursor(0, 0);
lcd_write_string(text_asetettu);
lcd_set_cursor(1, 0);
lcd_write_string(float_to_str2);
HAL_Delay(3000);
}

else		{
State = 2; //Palautus silmukan alkuun
}
  } //while
  /* USER CODE END 3 */
}
```