wControl – modulaarinen kiinteistön ohjausjärjestelmä

Juha Sinkkonen

Opinnäytetyö
Joulukuu 2015
Automaatioteknologian
ylempi koulutus

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Automaatioteknologia
ylempi koulutus

Juha Sinkkonen
wControl – modulaarinen kiinteistön ohjausjärjestelmä

Opinnäytetyö 56 sivua, joista liitteitä 5 sivua
Joulukuu 2015

Työ on toteutettu tarpeesta ja kiinnostuksesta kehittää uudenlainen kiinteistön ohjausjärjestelmä, joka mahdollisesti tullaan julkaisemaan avoimen lähdekoodin periaatteenmukaisesti. Työssä käsitellään lyhyesti myös joitain väylätekniikoita, sekä niiden soveltuvuutta järjestelmän verkko topologiaan. Lisäksi käsitellään digitaalitekniikan lähtöjen ja tulojen sovittamista suuremmalla käyttöjännitteellä toimivien toimilaitteiden osalta yhteensopivaksi. Työssä esitetään myös idea kehitettävästä automaatiojärjestelmästä, sekä siihen liittyvistä rajoitteista. Myös projektin ohjelmallista sekä fyysisistä rakennetta tullaan esittelemään työssä.

Työssä ei tulla julkaisemaan projektissa käytettyä ohjelmistokoodia, pois lukien joitain otsikko tiedostoja, tai järjestelmän fyysisissä komponenteissa käytettyjä piirikaaviota. Tämä johtuu koodin yhdenmukaisen formaatin puuttumisesta sekä sen keskeneräisyydestä. Työn luonteesta johtuen asiat tullaan esittelemään yleisellä tasolla sekä erilaisten kaavioiden avulla joissa kuvaillaan järjestelmän toimintaa sekä kokoonpanoa.

Asiasanat: automaatio, kiinteistöautomaatio, väylätekniikka

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in Automation Technology

Juha Sinkkonen
wControl – modular building control system
Title of thesis2

Bachelor's thesis 56 pages, appendices 5 pages
December 2015

This work has been carried out on the need and the interest of develop a new kind of building control system that will eventually be published with the open-source principle. The work also briefly discusses some of the bus technologies and those suitability for the system network topology. In addition there will be explain, of how to match digital technology inputs and outputs to use higher voltage levels that they should. The thesis also presents the idea of the developed automation system. Also the project's programmatic and physical structures will be introduced to the work.

The work does not come to publish the software code used in the project, except for some header files, or the circuit diagram used in the physical components of the system. This is due to the lack of a uniform format of the code and its incompleteness. Due to the nature of work matters will be presented on a general level as well as through a variety of diagrams describing the operation of the system and configuration.

Key words: automation, building automation, bus technology

SISÄLLYS

LYHENTEET JA TERMIT

UART	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface
I^2C	Inter-Integrated Circuit
TWI	Two Wire Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IDE	Integrated Development Environment
OSI	Open Systems Interconnection
HTTP	Hypertext Transfer Protocol
FTP	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
IP	Internet Protocol
RS-232	Recommended Standard 232
MSB	Most Significant Bit
ACK	Acknowledgement

1 JOHDANTO

Työn tavoitteena oli kehittää perinteisen releohjauksen rinnalle varteenotettava vaihtoehto kiinteistön eri toimintojen ohjauksessa. Työn aihe valikoitui omasta tarpeesta sekä kiinnostuksesta automaatioon ja elektroniikkaan. Markkinoilla nykyisin olevat kevyeen kiinteistöautomaatioon suunnatut järjestelmät olivat mielestäni, joko liian rajoittuneita laajennuksien suhteen, hankalia käyttöönottaa tai hinnoiteltu tavallisen kuluttajan ulottumattomiin. Tarkoituksena oli kehittää yksinkertaisesti käyttöön otettava järjestelmä, mutta joka voisi tarvittaessa suorittaa monimutkaisempiakin ohjauksia tarpeen mukaan. Järjestelmän modulaarisuus antaa tällöin mahdollisuuden hajauttaa järjestelmää useaan eri pisteeseen Ethernet verkon toimiessa järjestelmän runkoverkkona. Työssä tullaan sivuamaan erilaisia tekniikoita joita hyödyntämällä järjestelmää on tarkoitus lähteä rakentamaan. Koska kyseessä on vielä täysin proto- asteella toimiva kehitys, ei komponenttien toiminnoista ja käyttäytymisestä ole tarkkoja laskelmia, vaan asioita on yksinkertaistettu toimivan lopputuloksen aikaansaamiseksi. Työssä tullaan myös käsittelemään pintapuolisesti ohjelmoitavien piirien ominaisuuksia ja tutkia mahdollisuuksia hyödyntää niitä järjestelmän suunnittelussa ja toteutuksessa. Näiden lisäksi työssä tullaan esittelemään pinta puolisesti joidenkin väyläprotokollien toimintaa sekä näiden hyödyntämistä työssä.

Projekti on saanut alkunsa jo vuonna 2011, jolloin ensimmäisiä kokeiluja ja prototyyppejä rakennettiin. Ensimmäisissä kokeiluissa oli tarve saada tietokone liitettyä fyysiseen ympäristöön erilaisten tulojen ja lähtöjen avulla. Tällöin ei vielä ollut tietämystä sarjaliikenneväylä tekniikasta tai Ethernet verkon toiminnasta, mutta ohjelmoinnista kuitenkin oli jonkin asteinen perustietämys. Ensimmäiset kosketukset mikropiirien maailmaan ovat jo vuonna 2002, mutta varsinainen kiinnostus heräsi Texas Instrumentsin julkaisun myötä. Launchpad kehitysympäristö julkaistiin 2010, jolloin edullinen hinta sekä helpohko ohjelmoitavuus olivat avainasemassa järjestelmään tutustumisessa. Silloinen tietämättömyys elektroniikasta kuitenkin hidastivat järjestelmään tutustumista.

Vuonna 2012 Arduino ympäristö levisi tietoisuuteen ja erittäin helpon ohjelmointi ympäristön, sekä hyvän yhteisön tuen myötä kiinnostus aiheeseen heräsi jälleen uudelleen. Useiden erillisten sovellusten ja pienempien kokeilujen myötä ajatus kokonaisvaltaisesta kiinteistöautomaatio ympäristöstä saivat aivan uuden ulottuvuuden ja varsinaista kiinteistöautomaatio projektia lähdettiin kehittelemään.

2 JÄRJESTELMÄN KUVAUS

2.1 Yleistä

Järjestelmän rakennetta suunniteltaessa oli päällimmäisenä mielessä hajautettavuus, jolloin kaikkien toimilaitteiden ei fyysisesti tarvinnut sijaita samassa tilassa. Järjestelmän rakenteeseen vaikutti suuresti myös se, että oma tarve oli liittää useita eri rakennuksia samaan järjestelmään ja näin hallita kaikkien rakennusten ohjauksia keskitetysti. Pitkään markkinoilla ja useita kehitysvaiheita kohdannut Siemens LOGO! olisi ollut oiva ratkaisu kiinteistön hallintaan, mutta projektin alkutaipaleella, ei ko. järjestelmästä ollut saatavilla kuin enintään 24 tulon ja 20 lähdön järjestelmiä. Tämä ei tietenkään riittänyt, koska järjestelmään tulisi voida liittää erillisiä painikkeita, antureita sekä muita toimilaitteita huomattavasti enemmän.

Järjestelmä koostuu siis erilaisista toimilaitteista, jotka ovat lähiverkkoyhteyden avulla yhteydessä keskusyksikköön, joka prosessoi toimilaitteilta saadun datan ja sen perusteella lähettää toimilaitteille vasteen.

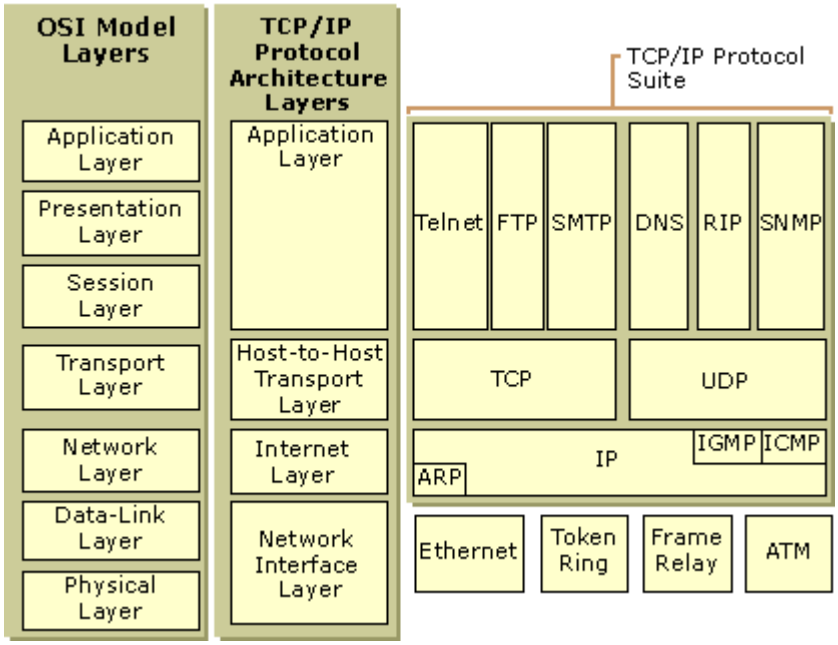
2.2 Verkkorakenne

Runkoverkoksi valikoitunut yleiskaapelointiverkko ei vielä sinänsä aseta mitään ehtoja tiedon siirrolle. Yleiskaapelointi, joka tunnetaan myös nimeltä ”ATK-verkko”, on kaapelointi rakenteensa ja täten myös häiriösuojauksen kannalta oiva ratkaisu runkoverkoksi.

OSI-malli (Open Systems Interconnection) on 80-luvun alussa ISO:n (International Organization for Stanrdardization) kehittämä malli. OSI-malli ei kuitenkaan sellaisenaan koskaan yleistynyt, sillä ennen sitä kehitetty TCP/IP protokolla pino oli käytettävyydeltään sekä käytön määrässä OSI-mallia edellä. OSI-malli toimii kuitenkin referenssiluonteensa vuoksi tärkeänä osana nykyisiä verkkoprotokollia. OSI-mallin kerrosajatus on sama kaikissa käytettävissä olevista protokollista, jolloin jokainen kerros voidaan toteuttaa omana kokonaisuutenaan. (Anttila, A. 2000. 30–35)

OSI-mallin mukaan yleiskaapelointiverkko eli fyysinen kerros on järjestelmän alimmainen taso ja luo pohjan järjestelmän rakenteelle. Sama pätee myös TCP/IP – mallin ta-

pauksessa (Kuva 1). Fyysisen kerroksen päällä sijaitseva Verkko- ja Internet-kerros (Internet Layer), jonka tarkoitus on vastata viestien reitityksestä. Seuraavaksi pinossa on kuljetuskerros (Transport Layer), joka vastuussa tiedon kuljetuksesta sovellukselle. Kuljetus kerroksessa käytetyt protokollat ovat TCP(Transmission Control Protocol) ja UDP(User Datagram Protocol). Ylimpänä pinossa on sovelluskerros, joka hoitaa sanoman prosessoinnin. Yleisimmät sovelluskerroksen protokollat ovat HTTP (Hypertext Transfer Prtocol), FTP(File Transfer Protocol) ja SMTP(Simple Mail Transfer Protocol). (Microsoft, TCP/IP Protocol Architecture.)



Kuva 1. TCP/IP protokollan arkkitehtuuri. (Microsoft, TCP/IP Protocol Architecture.)

2.3 Käytettävät protokollat

Kun oli päätetty käytettävä runkoverkon tyyppi, oli vuorossa ratkaista tiedonsiirrossa käytettävien protokollien valinta. Koska tiedonsiirrossa haluttiin käyttää jo olemassa olevia sovelluksia ja laitteita, ei lähdetty kehittämään uutta tiedonsiirtomenetelmää yleiskaapelointiverkossa. IP – protokolla (Internet Protokcol) on reititettävä protokolla joka vastaa IP – osoitteesta, reitityksestä sekä pirstoutuneiden pakettien uudelleen kokoamisesta. IP – protokollan valinta tuntui luontevalta, sillä tällöin voitaisiin hyödyntää jo olemassa olevia tietorakenteita sekä reitittimiä. Seuraavaksi valittavana oli käytettävä kuljetusprotokolla. Vaihtoehtoina TCP ja UDP.

2.3.1 TCP (Transmission Control Protocol)

TCP – protokolla on ns. yhteydellinen yhteys, joka tarkoittaa sitä, että ensiksi muodostetaan ja varmistetaan yhteys haluttuun kohteeseen. Seuraavaksi lähetetään haluttu data kohteeseen, jossa vastaan ottaja varmistaa saadun datan ja sen oikeellisuuden sekä lähettää lähettäjälle kuittauksen vastaanotetusta datasta. Lähetyksen päätyttyä onnistuneesti suljetaan yhteys molemmin puolisesti. TCP yhteys on siis luotettava ja tietoinen mahdollisesti virheellisesti vastaan otetusta viestistä, mutta vaatii verkkokapasiteettia, sillä pelkästään lähetettävän sanoman otsikko kenttä vaatii 192 – bittiä. Tämä ei tietenkään ole optimaalinen ratkaisu lyhyille sanomille. (Anttila, A. 2000. 133–135)

2.3.2 UDP (User Datagram Protocol)

UDP – protokolla käyttää ns. yhteydetöntä lähetysmallia. Protokollalla ei ole ns. kättely metodia, kuten TCP – protokollalla, eli se ei varmista vastaanottajaa ennen viestin lähetystä. Koska vastaanottajaa ei varmisteta, ei voida olla varmoja viestin perille menosta, eikä siinä mahdollisesti esiintyvistä virheistä. UDP on siis riisuttu lähes kaikista TCP – protokollan hyvistä ominaisuksista. (Anttila, A. 2000. 167–169)

Miksi sitten tällainen on olemassa ja miksi sitä käytetään? UDP – protokollan suurimpina etuina ovat nopeus. Koska se ei sisällä kättelyitä, virheenkorjausta tai kuittauksia voidaan keskittyä olennaiseen, eli tiedon kuljettamiseen lähettäjältä vastaanottajalle. UDP:n otsikkokenttä sisältää vain 64-bittiä, joka on huomattavasti pienempi verrattuna TCP – protokollaan. (Anttila, A. 2000. 167–169)

Offset	Octet	0	1	2	3
Octet	Bit	0-7	8-15	16-23	24-31
0	0	Lähdeportti		Kohdeportti	
4	32	Pituus		Tarkistussumma	

Taulukko 1. UDP otsikko

Koska UDP – protokollan kevyt rakenne tukee suunniteltavan järjestelmän käyttämiä sanomia, on se selkeä valinta järjestelmän runkoverkon kuljetuskerrokseksi. Toisin kuin TCP – protokolla käytettäessä, jos lähetettävät sanomat ovat lyhyitä ja niitä lähetetään usein, joudutaan joka kerta suorittamaan yhteyden avaus eli kättely, viestin lähettäminen, kuittaus sekä yhteyden sulkeminen. UDP – protokollan tapauksessa viestille annetaan osoite, lähetetään se matkaan ja toivotaan parasta. (Anttila, A. 2000. 167–169)

2.4 Väylät

Ensimmäisen kehitysversion perusajatuksena oli, että jokainen toimilaite järjestelmässä tullaan liittämään lähiverkkoyhteyden kautta järjestelmän keskusyksikköön eli CPU:hun. Aluksi ajatus tuntui toimivalta ja sitä kehiteltiin jonkin matkaa. Järjestelmämallista saatiin aikaan prototyyppi, jota testattiin ja todettiin toimivaksi puutteistaan huolimatta. Kun järjestelmää alettiin miettiä suuremassa mittakaavassa, huomattiin kuitenkin eräs heikkous mallissa. Mikäli järjestelmään tultaisiin liittämään useita kymmeniä tai jopa satoja toimilaitteita, tulisi se kuormittamaan käytettävää verkkoyhteyttä sekä vaatisi keskusyksiköltä hyvän kapasiteetin käsitellä useaa verkkoyhteyttä samanaikaisesti. Tällöin lähtökohtana oli, että lähiverkko olisi ns. yleinen, eli järjestelmä toimisi muun tietoliikenteen ohessa samassa verkossa. Eli miten lähdetään karsimaan verkkoon liitettyjen laitteiden määrää? Eräs ratkaisumalli voisi olla, että rakennetaan suurempia komponentteja joissa on enemmän toimintoja ja näin ollen saataisiin toimilaitekantaa karsittua. Tällaisella ajatusmallilla menetettäisiin kuitenkin järjestelmän joustava ja modulaarinen rakenne. Ratkaisuvaihtoehdoksi päätettiin ruveta tutkimaan mahdollisen aliväylän rakentamista. Koska käytettävässä tekniikassa oli jo valmiiksi olemassa ns. rautatason väyliä, päätettiin niitä tutkia tarkemmin. Atmeli:n valmistamissa esim. Atmega -sarjan piireissä on lähes poikkeuksetta vakiona käytettävissä mm. seuraavat väylätekniikat: UART, SPI, TWI (I^2C).

2.4.1 UART

UART (Universal asynchronous receive/transmitter) on sarjaliikenne piiri, joka voidaan konfiguroida tukemaan useita eri protokollan väyliä, esim. RS-232.

Esimerkiksi sarjaliikenneväylä RS-232 käyttää kahta linjaa tiedonsiirtoon ja on ns. asynkroninen (ei sisällä erillistä kellolinjaa). Väylään liitetyt laitteet ovat siis etukäteen määritelty toimimaan tietyllä nopeudella. Lisäksi, kun puhutaan mikrokontrollereista, tulee niiden kellotaajuus olla lähes sama, jotta väylässä kulkeva tieto ei vääristyisi. (SparkFun. I^2C Tutorial.)

Asynkroninen sarjaliikenne on lisäksi suunniteltu toimimaan ainoastaan kahden erillisen laitteen välillä ja useamman laitteen liittäminen samaan sarjaliikenneväylään on hankalaa

ja vaatii usein ulkopuolisen laitteiston toimiakseen halutulla tavalla. (SparkFun. I^2C Tutorial)

2.4.2 SPI

SPI (Serial Peripheral Interface Bus) – väylä on synkroninen sarjaliikenne väylä, joka on tarkoitettu lyhyiden matkojen kommunikointiin. Väylän on kehittänyt Motorola, Inc. ja siitä on tullut standardi mm. SecureDigital – muistikorttien tiedonsiirto järjestelmänä. SPI – väylä käyttää Master-Slave tyyppistä kommunikointi arkkitehtuuria, jossa voi olla vain yksi Master. SPI – väylä käyttää kommunikointiin neljää linjaa, joista yksi on laitekohtainen SS – linja. SS – linja (Slave Select) eli orjan valinta, jolla valitaan mihin orjalaitteeseen ollaan kulloinkin yhteydessä. SPI – väylä vaatii siis 3+n kappaletta johtimia toimiakseen(n = orjalaitteiden lukumäärä). SS – linjan lisäksi käytettävät linjat SCLK(Serial Clock) eli kellolinja, MOSI(Master output, Slave input) datalinja, MISO(Master input, Slave output) datalinja. (SPI - Serial Peripheral Interface.)

Väylän hyvinä puolina voidaan todeta sen nopeus sekä Full-Duplex – ominaisuus, joka mahdollistaa datan samanaikaisen lähettämisen ja vastaanottamisen. Väylän käyttäminen laajemmissa kokonaisuuksissa kuitenkin hankaloituu, sillä jokainen väylään liitetty laite vaatii oman lähtönsä (SS – linja) ohjainlaitteelta. Tähän tarkoitukseen olisi tietenkin mahdollisuus käyttää erillisiä siirtorekistereitä, jolla rajoitetta voitaisiin kiertää. Mikäli haluttaisiin käyttää SPI – väylää järjestelmän aliväylänä, toisi se haasteita fyysisen laitteen suunnittelussa ja toteutuksessa. Jos väylään liitettäviä laitteita olisi useita kymmeniä, tulisi fyysisellä komponentillakin olla useita kymmeniä ulkoisia liittimiä, joihin väylälaitteet liitettäisiin.

SPI – väylä toimii kuitenkin omassa roolissaan kehitettävän järjestelmän osalta, sillä useat eri anturit ja piirit tukevat SPI – yhteyden käyttämistä, jolloin paikallisia yhteyksiä voidaan käyttää väylää hyödyntäen.

2.4.3 TWI (I^2C)

TWI- (Two Wire Interface) ja I^2C (Inter-Integrated Circuit) – väylät tarkoittavat käytännössä samaa asiaa. I^2C – väylän on alun perin kehittänyt Philips chips vuonna 1982. Tuolloin väylän nopeudeksi oli määritelty 100kbit/s ja osoiteavaruudeksi 7-bittiä, sillä monet sovellukset eivät tarvitse nopeampaa tiedonsiirtoa. Vuonna 1992, ensimmäisessä julki-

sessa määrittelyssä, väylän nopeudeksi lisättiin 400kbit/s(Fast mode), sekä tuki 10-bittiselle osoiteavaruudelle. Lisäksi nykyisin on määritelty väylälle nopeudet 1 Mbit/s (Fast-mode plus), 3.4 Mbit/s (High-speed mode) sekä 5 Mbit/s (Ultra-fast mode). (SparkFun. I2C Tutorial.)

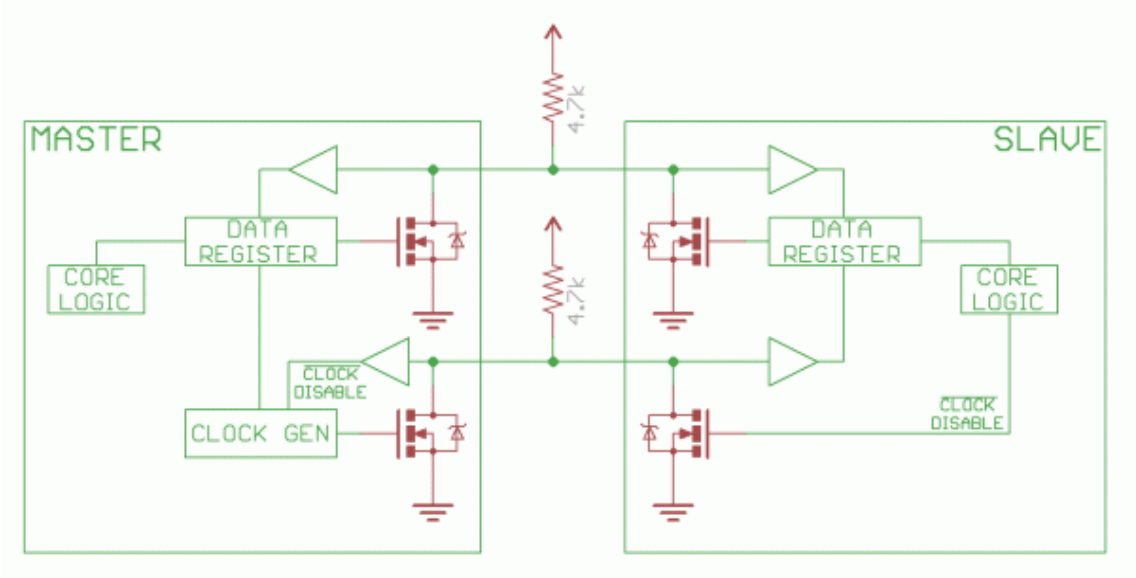
TWI tai TWSI (Two-Wire Serial Interface) on siis käytännössä sama väylä kuin I^2C, mutta toteutettu eri valmistajien johdolla esim. Atmel. Vaikka I^2C ei varsinaisesti ole rekisteröity tavaramerkki ja I^2C koskevat patentit ovat rauenneet, pitäytyvät osa piirien valmistajista edelleen TWI nimikkeessä. Joissakin tapauksissa TWI merkintä voi myös tarkoittaa puutteellista toteutusta I^2C – väylän ominaisuuksista. (I^2C Wikipedia.)

Alun perin I^2C – väylän on siis suunniteltu toimimaan 7-bittisellä osoiteavaruudella. Normaali 7-bitin osoiteavaruus antaa siis mahdolliset 128 eri osoitetta, joista osoite 0 on varattu ns. broadcast osoitteeksi. Tämän lisäksi on myös joukko muita varattuja osoitteita. Jokaisella väylään liitetyllä Slave – laitteella on siis oma yksityinen osoitteensa. Väylässä toimii Slave – laitteen lisäksi nk. Master, joka toimii väylän operoijana. Väylän laitteilla voi olla siis kaksi eri mahdollista roolia Master sekä Slave. Joissain tapauksissa voi olla tarpeen, että on useampi Master yksikkö, jolloin voidaan sanoa, että väylä ns. Multi-Master tyyppinen. (I^2C Wikipedia.)

Väylän rakenne koostuu kahdesta eri linjasta, SDA (Serial Data Line) ja SCL (Serial Clock Line). Kellolinjan signaalia (SCL) ohjaa aina väylään liitetty Master. On kuitenkin tapauksia, jolloin Slave – yksikkö joutuu pakottamaan kellolinjan pysymään alhaalla hidastaakseen Master - yksikön lähettämän datan määrää tai vaatiakseen aikaa valmistellessaan omaa lähetystään Master - yksikölle. Tällöin puhutaan termistä ”clock stretching” eli kellopulssin venyttämisestä. (SparkFun. I^2C Tutorial.)

Toisin kuin UART tai SPI, I^2C – väylä toimii ”open dran” periaatteella (vrt. transistori open collector). Tämä tarkoittaa sitä, että väyläohjain pystyy asettamaan väylän alatilaan (pull down), mutta ei asettamaan takaisin väyläjännitteen tasolle. Tällöin väylän linjoihin tulee asettaa ylös vetovastukset (pull-up resistor), joiden avulla linjojen jännitetaso saadaan nostettua väylän jännitettä vastaavaksi. Tämän menettelyn eräänä hyvänä etuna on se, että väylässä ei voi olla laitetta joka yrittää nostaa signaalitasoa ylös, kun jokin toinen

väylään liitetty laite on asettamassa sitä alatilaan. Mikäli tällainen menettely olisi mahdollinen, saattaisi linjoissa syntyä hallitsemattomia kiertovirtoja jotka voisivat vahingoittaa väyläohjaimia. (SparkFun. I^{2}C Tutorial.)

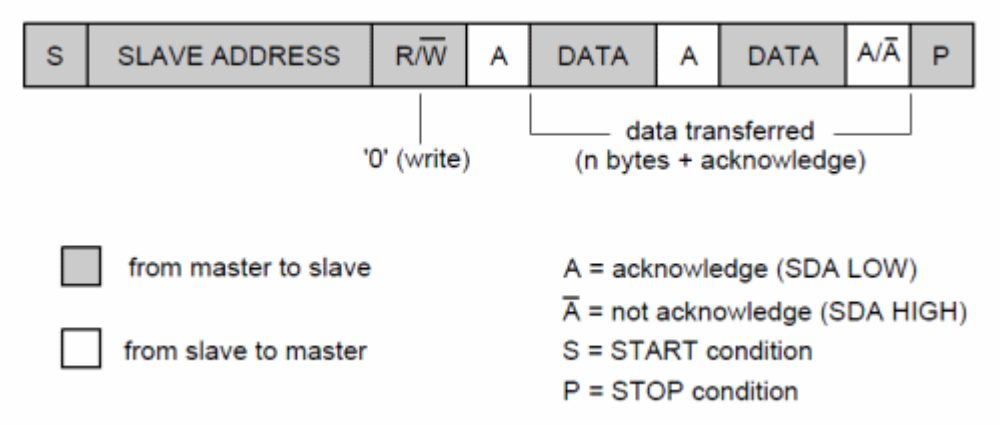


Kuva 2. I^{2}C väylän sähköinen periaate sekä ylös vetovastukset väylän linjoissa.

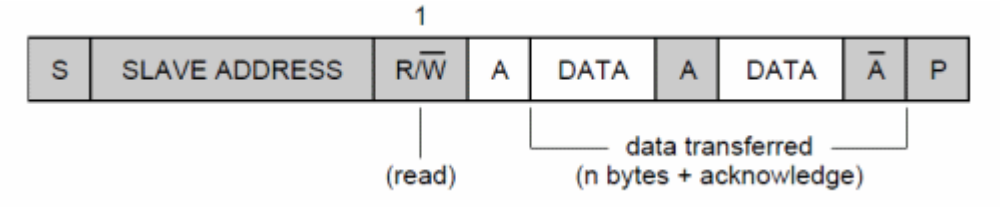
Koska väylän laitteet eivät pyri vetämään väylän linjoja ylätilaan, antaa se joustavuutta väylään liitettävien laitteiden suhteen. Jos väylään on liitetty eri käyttöjännitteellä toimivia laitteita esim. 5 ja 3.3V, voitaisiin väylä toteuttaa näiden kahden laitteen välillä ilman erillistä tasomuunninta (Level Shifter). Väylän ylös vetovastukset yksinkertaisesti kytkettäisiin heikomman käyttöjännitteen omaavan laitteen käyttöjännitteeseen, jolloin väylän ylätilan jännite ei nouse pienemmän käyttöjännitteen omaavalle laitteelle haitalliseksi. Tämä tietenkin tarkoittaa sitä, että laite, joka omaa suuremman käyttöjännitteen, tunnistaa väylän yläsignaalitason (High-level Input).

Väylän lähettämä viestikehys on erittäin yksinkertainen ja sisältää vain tarvittavan tiedon nopeaa tiedonsiirtoa varten. Väylän käyttämä tiedon siirto on MSB joka tarkoitta, että eniten merkitsevä bitti on ensimmäisenä. (I^{2}C Bus Specification.)

Kuva 2 sekä kuva 3 esittävät viestikehyksen, viestin lähetyksestä sekä vastaanotosta Slave ohjaimelta.

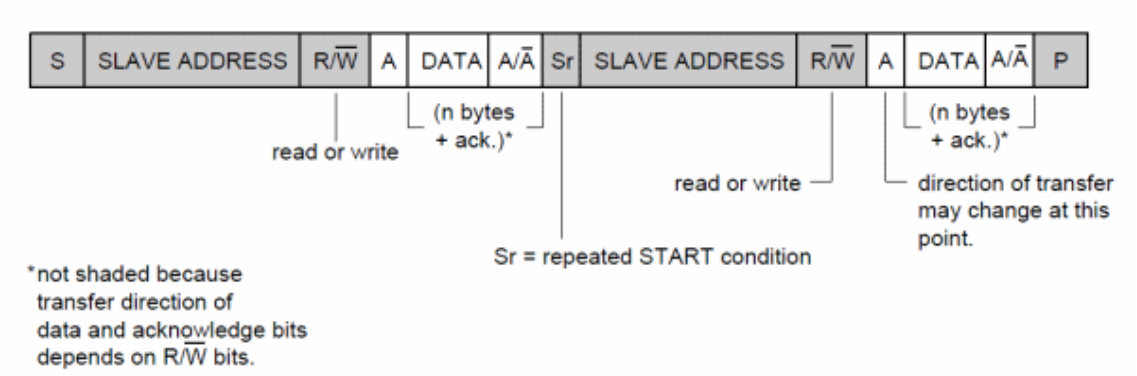


Kuva 3. I²C Dataframe Master write to Slave. (I²C Bus Specification.)



Kuva 4.I²C Dataframe Master read from Slave. (I²C Bus Specification.)

Mikäli on tarvetta kirjoittaa ja lukea Slave – laitteelta voidaan se toteuttaa ilman yhteyden katkaisua STOP – bitillä ja aloittamista uudelleen START – bitillä seuraavasti:

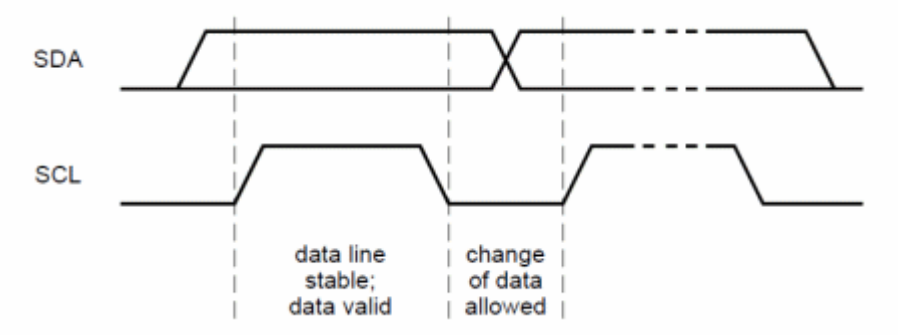


Kuva 5. I²C Dataframe Master Read and/or write from/to Slave. (I²C Bus Specification.)

Vaikka itse viesti joka I²C – väylän kautta lähetetään, vaikuttaa yksinkertaiselta. Se miten se lähetetään, on hieman monimutkaisempi, kuin UART tai SPI ratkaisuissa. Signaalit, joita I²C väylässä lähetetään, täytyy noudattaa määrättyä protokollaa, tullakseen tunnistetuksi I²C viestiksi. Esimerkissä käytettyjen Atmega – piirien osalta asiaa helpottaa se, että piireihin on sisäänrakennettu rautatason TWI (I²C) – protokolla, jolloin kaikkiin pienempiin yksityiskohtiin ei tarvitse kiinnittää sen suurempaa huomiota. (SparkFun. I²C Tutorial.)

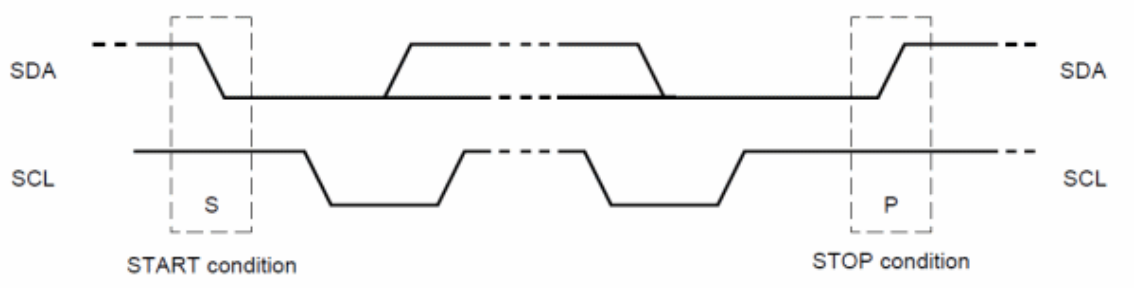
Kuten edellä esitetyistä viestikehyksistä voimme todeta, viesti koostuu kahdesta eri osasta: osoitteesta sekä datasta. Osoite kertoo siis mihin Slave – ohjaimeen tieto on menossa ja dataosa, tietoa jota ollaan lähettämässä. Tiedot lähetetään kahdeksan bitin jaksoissa, joita seuraa aina ACK-bitti.(I²C Bus Specification.)

Väylän määrittely ei varsinaisesti rajoita lähetettävän datan määrää, mutta rajoittavaksi tekijäksi saattaa muodostua esimerkiksi vastaanottajan kyky varastoida lähetetty data.



Kuva 6. I²C – väylä stabiilissa tilassa sekä bitin lähetys. .(I²C Bus Specification.)

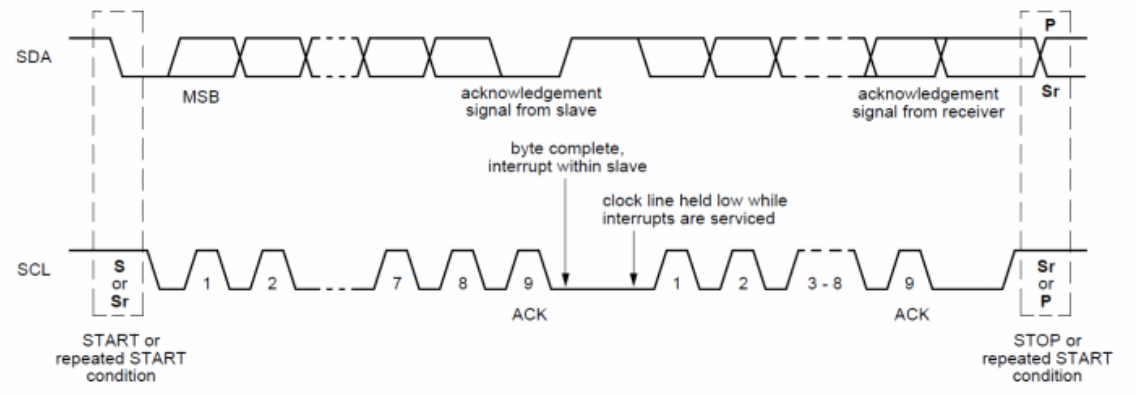
Kuten kuvasta 6 huomamme, väylä on ns. stabiilissa tilassa, kun molemmat linjat (SDA, SCL) ovat ylätilassa. Kun kellopulssi asetetaan alatilaan, sallii se tällöin tiedon lähettämisen tai vastaanottamisen. (I²C Bus Specification.)



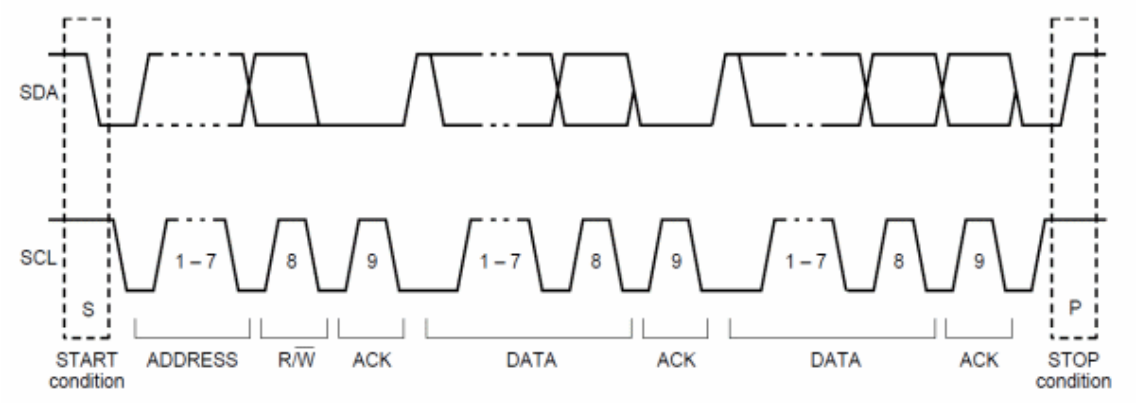
Kuva 7. I²C aloitus sekä lopetusbitti. (I²C Bus Specification.)

Kuten aiemmin totesimme, jokainen uusi lähetys alkaa aloitus bitillä. I²C – väylässä tämä on toteutettu siten, että datalinja vedetään ylätasosta alatasoon kellolinjan ollessa ylätasossa. Vastaavasti viesti lopetetaan lopetusbitillä. Lopetusbitti muodostuu vastaavasti kuin aloitus bitti, mutta tällöin datalinja asetetaan alatilasta ylätilaan, kellolinjan ollessa ylätilassa. Mikäli väylässä on useampi Master – ohjain, voivat ne käyttää väylää ainoastaan, kun se on vapaa. Jos yksi Master – ohjain on lähettänyt aloitusbitin väylään, eivät

muut Masterit pysty väylää käyttämään ennen kuin aloitusbitin lähettänyt on lopettanut lähetyksen ja lähettänyt lopetusbitin. (I²C Bus Specification.)



Kuva 8. I²C - Väylän tahdistus. (I²C Bus Specification.)



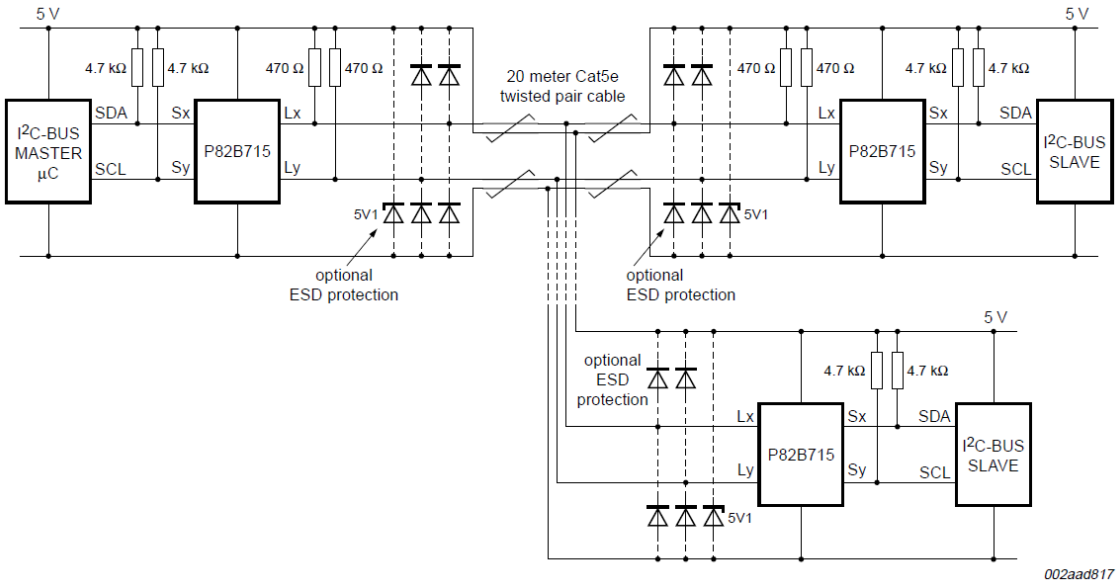
Kuva 9. I²C - Väylän data. (I²C Bus Specification.)

Kuvat 8 ja 9 esittää miten tiedonsiirto väylässä tapahtuu. Tiedon siirto aloitetaan siis aloitusbitillä, jota seuraa seitsemän osoitebittiä. Tämän jälkeen lähetetään bitti, joka määrää kirjoitetaanko vai luetaanko Slave – ohjaimelta tietoa. Tämän jälkeen Slave – ohjain lähettää ACK – bitin Master – ohjaimelle. ACK eli ”acknowledge” – bitti osoittaa, että Slave – ohjain on tietoinen Master – ohjaimesta. Tämän jälkeen alkaa varsinainen data jota ollaan kirjoittamassa tai jota ollaan lukemassa. Tämä jatkuu 8-bitin sarjoissa, joihin sisältyy aina Slave – ohjaimen ACK – bitti. Mikäli Slave – ohjaimeen ollaan kirjoittamassa, päättää Master – ohjain viestin viimeisen Slave – ohjaimen ACK – bitin jälkeen. Jos taas ollaan lukemassa Slave – ohjaimelta, lähettää Master – ohjain ensiksi rekisterin osoitteen mistä tai mitä luetaan. Tämän jälkeen Slave – ohjain lähettää 8-bitin sarjan takaisin Master – ohjaimeen. Kaikissa tiedonsiirron vaiheissa Master – ohjain kuitenkin aina määrää kellopulssin. (I²C Bus Specification.)

Yleisenkäytännön mukaan Slave – ohjain on jaettu rekistereihin joista voidaan lukea ja joihin voidaan kirjoittaa. Jos Master – ohjain haluaa kirjoittaa dataa tiettyyn rekisteriin Slave – ohjaimella, tulee sen ensiksi lähettää rekisterin osoite ensimmäisessä data lähetyksessä, ja sen jälkeen itse varsinainen data. Tätä mallia tullaan käyttämään myös kehitystyön osalta, jolloin eriohjainten kesken on käytettävissä 256 (8-bittiä) eri rekisteriosoitetta.

Suunnitellussa järjestelmässä väylän käyttöä rajoittavaksi tekijäksi muodostuu kaapeloinnin, ohjainten väylään aiheuttama kapasitanssi, joka ei saisi ylittää 400pF. Sellaisenaan väylä ei siis sovellu käytettäväksi, kuin muutamien metrien matkalle. Koska väylän tekniikka vaikuttaa muuten erittäin lupaavalta, lähdettiin etsimään ratkaisuja jolla väylän pituutta voitaisiin lähteä kasvattamaan, väylän laadun tai nopeuden siitä kuitenkaan kärsimättä.

Lyhyen etsinnän jälkeen NXP Semiconductorsin valikoimista löytyi useampikin lupaava vaihtoehto tarkoitukseen. P82B715 I^2C-bus extender on I^2C – väylään tarkoitettu väylän laajentaja. Se muuttaa väylän virraksi alkuperäisen 3mA sijasta 30mA sekä kasvattaa väylän sallimaa kapasitanssia aina 3000pF asti. Komponentti ei vaadi ympärilleen juurikaan ympäristökomponentteja ja soveltuukin käyttöön sellaisenaan. Kuten normaalissakin I^2C väylässä, myös tässä ylösvetovastukset ovat avain asemassa väylän toimivuuden kannalta. (NXP Semiconductors. P82B715 Product data sheet.)

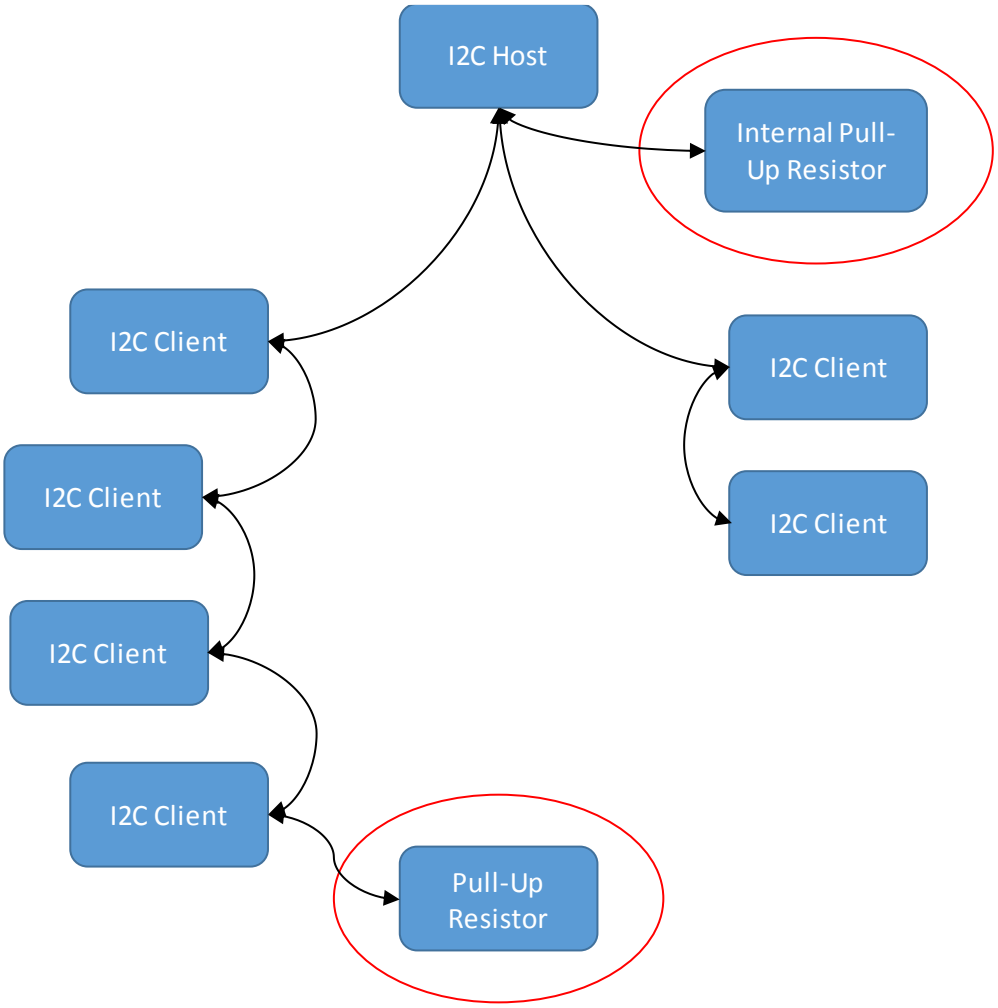


Kuva 10.P82B715 Quick design-in poin-to-point/multi-point circuit for 5V bus. (NXP Semiconductors. P82B715 Product data sheet.)

Komponentti yksinkertaisuudessaan kytketään sisäisen (”Internal”) ja ulkoisen (”External”) väylän väliin. Molemmin puolin komponenttia on välttämätöntä asettaa ylös vetovastukset väylän toimivuuden kannalta. Lisäksi joitain suojadiodeja suositellaan käytettäväksi kapasitanssista johtuvan jännitteen kohoamisen ehkäisemiseksi. (NXP Semiconductors. P82B715 Product data sheet.)

Tilanne vaikuttaa kovinkin lupaavalta ja helpolta toteuttaa, mutta väylän laajentajalta lähtevä nk. ulkoisen väylän ylösvetovastusten suuruus määräytyy vaaditun signaalin nousuajan sekä väylän kapasitanssin perusteella. Tämä tuottaa jokseenkin ongelmia vastusten mitoituksen suhteen, sillä järjestelmää on tarkoitus rakentaa moduuli kerrallaan, eikä väylän tarkkaa mittaa silloin voida tietää. Asiaa helpottaisi jos kaikki järjestelmän komponentit sijaitsisivat samassa tilassa / kotelossa, niin voitaisiin ajatella väylän olevan joitain metrejä. Koska komponentteja voidaan sijoittaa lähes vapaasti eri paikkoihin, tulee ylösvetovastusten arvoksi asettaa kompromissi väylän pituuden sekä komponenttien virran keston välillä. Mitä pienempi ylösvetovastus väylään liitetään, sitä suuremman virran joutuu laitteisto kestämään, kun linjoja asetetaan alatilaan. Lisäksi väylärakenteessa olisi suotavaa, että ylösvetovastukset olisivat väylän molemmissa päissä. Alkupään vastuksien paikka on tietenkin selvä ja ne sijaitsevat kiinteästi Master – ohjaimella. Loppupään vastuksen sijoitus aiheuttaa jälleen pientä ihmetystä, sillä verkkotopologia on käytännössä tähtimäinen, eli jokainen Client on kytketty väylään rinnan, eikä näin ollen sisällä varsinaista loppupäätä. Vaikka väylällä ei varsinaista loppupäätä olekaan, päätettiin väylän nk.

loppupään vastukset asettaa väylän ns. viimeisimpään Slave – ohjaimeen, joka todennäköisesti on myös kauimpana tähtipisteestä (Kuva 11).

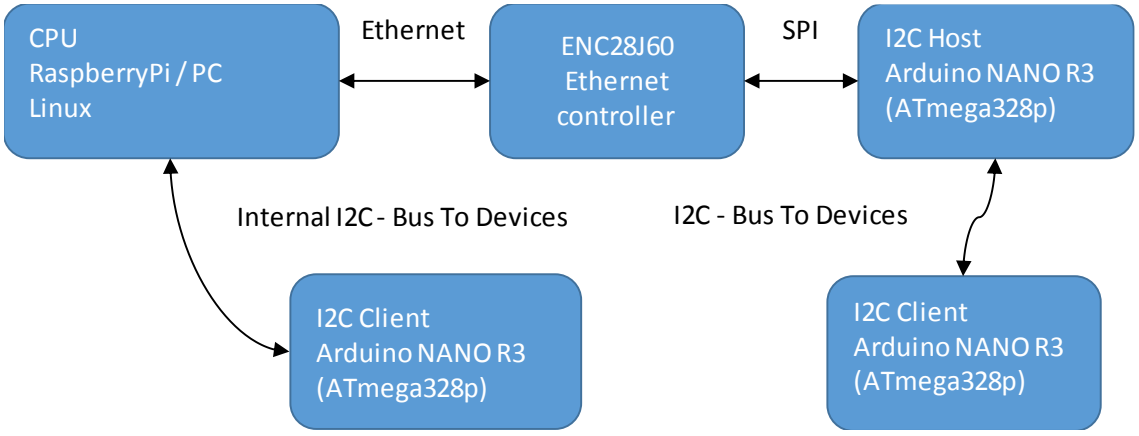


Kuva 11. Väylän topologia sekä päätevastuksen sijainti

3 LAITTEISTO

3.1 Järjestelmä

Suunniteltava järjestelmä koostuu siis keskusyksiköstä, joka hoitaa yhteydet toimilaitteisiin sekä prosessoi toimilaitteilta saadun tiedon. Saadun tiedon perusteella keskusyksikkö lähettää käsittelemiensä tiedon takaisin toimilaitteille. Toimilaitteet voivat suorittaa fyysisiä toimintoja kuten releen aktivointi, transistori lähdön päälle. Järjestelmän yhteydet muodostuvat pääasiassa erilaisten väylien välityksellä. Keskusyksikkö on yhteydessä Host - ohjaimeen Ethernet ohjaimen kautta, joka puolestaan on yhteydessä Host - ohjaimeen SPI – väylän välityksellä. Host -ohjain on puolestaan yhteydessä alamaisiinsa I^2C – väylän välityksellä. Mikäli käytettävä keskusyksikkö tukee suoraan I^2C – väylää, voidaan toimilaitteet kytkeä suoraan keskusyksikköön ilman Ethernet verkkoa (Kuva 12). Tämä on hyvä ominaisuus, jos tarkastellaan laitteistoa suppeassa ympäristössä.



Kuva 12. Järjestelmärakenne

Kuten kuvasta käy ilmi keskusyksikkönä tulee toimimaan joko x86 tai AMR pohjainen keskusyksikkö jonka käyttöjärjestelmänä toimii Linux. Linux jakeluja on olemassa lukemattomia erilaisia, mutta työhön valikoitui Debian nimeä kantava jakelu, jolla on laaja pakettitarjonta. Lisäksi jakelun valintaan vaikutti se, että se on saatavilla molemmille prosessori arkkitehtuureille. Tällöin käytettävään ohjelmistoon ei tarvitse muutoksia tehdä siirryttäessä arkkitehtuurista toiseen.

Perinteisesti PC koneissa (x86) sekä esimerkiksi RaspberryPi (ARM), on molemmissa saatavilla verkko-ominaisuus, eli tietoliikenne yhteydet tarjoava nk. verkkokortti on joko integroitu tai erillisenä lisäkortilla saatavilla. Tällöin tehtäväksi jää ainoastaan huolehtia vastaanottajan (I^2C Host) verkko-ominaisuuksista.

Vaikka RaspberryPi sisältääkin valmiin verkkoyhteyden, niin sisältää se myöskin sisäisen I^2C väylän. Sisäistä I^2C väylää hyödyntämällä voidaan toimilaitteita liittää suoraan keskusyksikköön, jolloin järjestelmä ei tarvitse lainkaan verkkoresursseja tai mikäli niitä ei ole saatavilla. Tämä on erinomainen ominaisuus varsinkin pienissä järjestelmissä, joissa tarvitaan enintään joitain kymmeniä toimilaitteita. Myös PC ympäristöön on saatavilla vastaavanlaisia laajennuksia, jolloin USB – liitynnän kautta voidaan kommunikoida I^2C – väylään.

3.2 Rakenne

3.2.1 Ethernet

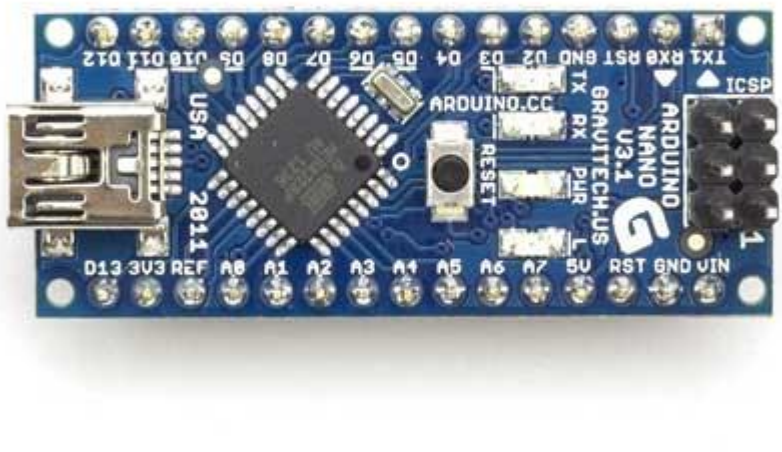
Järjestelmän ”selkärankana” toimii siis Ethernet verkko, jossa kuljetusprotokollana käytetään UDP – tekniikkaa. Vaikka UDP – tekniikka onkin epävarma ja varmuutta sanomien perille menosta ei voida saada, on sen nopeus silti ratkaisevassa asemassa, kun käsitellään lyhyitä sanomia. Mikäli järjestelmää tullaan käyttämään verkkoympäristössä, tarvitsee se toimiakseen reitittimen tai kytkimen. Järjestelmä ei vaadi erillistä DHCP – palvelinta, koska kaikki järjestelmässä käytetyt laitteet voidaan määrittää käyttämään kiinteää IP – osoitetta. Käytännössä kaikki standardit täyttävät reitittimet, kytkimet ja keskittimet toimivat järjestelmän kanssa, koska verkossa siirretty data on standardimuotoista UDP dataa.

3.2.2 I2C Host

Itse Host – ohjain pohjautuu Atmel Corporationin valmistamaan ATmega – sarjan 328p – piiriin. Piirin kellotaajuus voidaan asettaa enintään 20MHz, mutta tässä työssä se toimii 16MHz taajuudella. Piiriltä löytyvät tarvittavat väyläliitännät, kuten: UART, SPI ja I2C. Piirin valintaan vaikutti erityisesti sen hyvä suorituskyky, helppo saatavuus ja hinta. Lisäksi piiri on käytössä Arduino ympäristön kehitysalustoissa sekä moduuleissa, joten siihen lähestyminen oli helppoa. Arduino ympäristön myötä myös liityntä erilaisiin väyliin(I^2C, ja SPI), oli sujuvaa.

Työssä tullaan käyttämään valmista moduulia prototyypin toteuttamiseen, sillä moduulissa on valmiina tarvittavat ympäristökomponentit ja näin säästytään erillisen testialustan rakentamiselta (Kuva 13). Lisäksi moduuli sisältää CH340G ”USB to Serial” muuntimen,

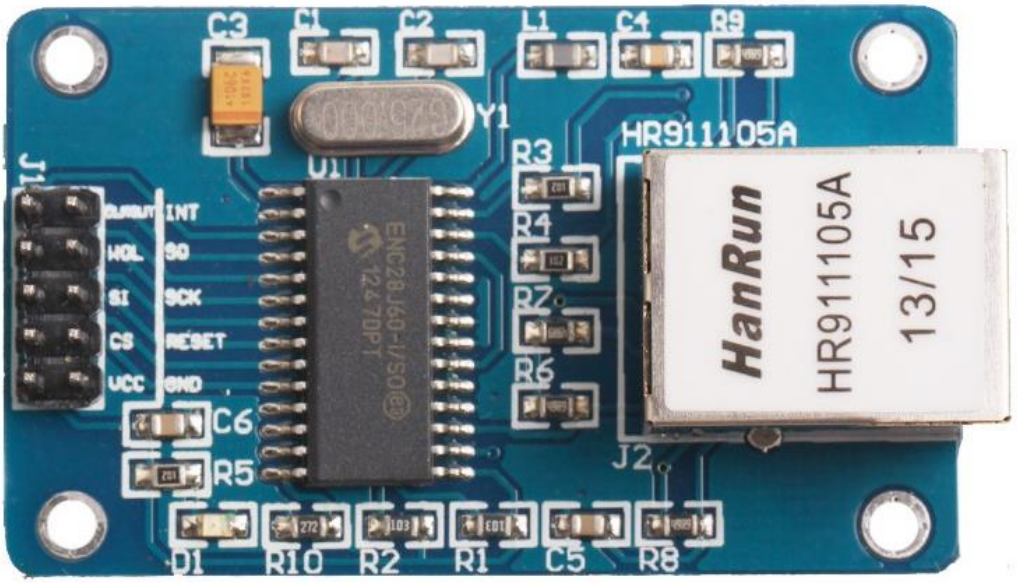
joka helpottaa piirin liittämistä tietokoneeseen. USB – liitännän avulla päästään käsiksi piirin sarjaliikenteeseen jolla voidaan tutkia helposti piirissä tapahtuvia toimintoja ja näin suorittaa virheenkorjausta (debugging). Lisäksi käytettävä piiri on varustettu Arduino – käynnistyslataimella, joka mahdollistaa ohjelman lataamisen piirin muistiin suoraan USB – liitynnän kautta. Moduuli tarjoaa myös valmiiksi piirilevyn reunoille reititetyt I/O nastat, jolloin itse prototyypin suunnittelu helpottuu huomattavasti.



Kuva 13. Arduino Nano Front. (Arduino.cc.)

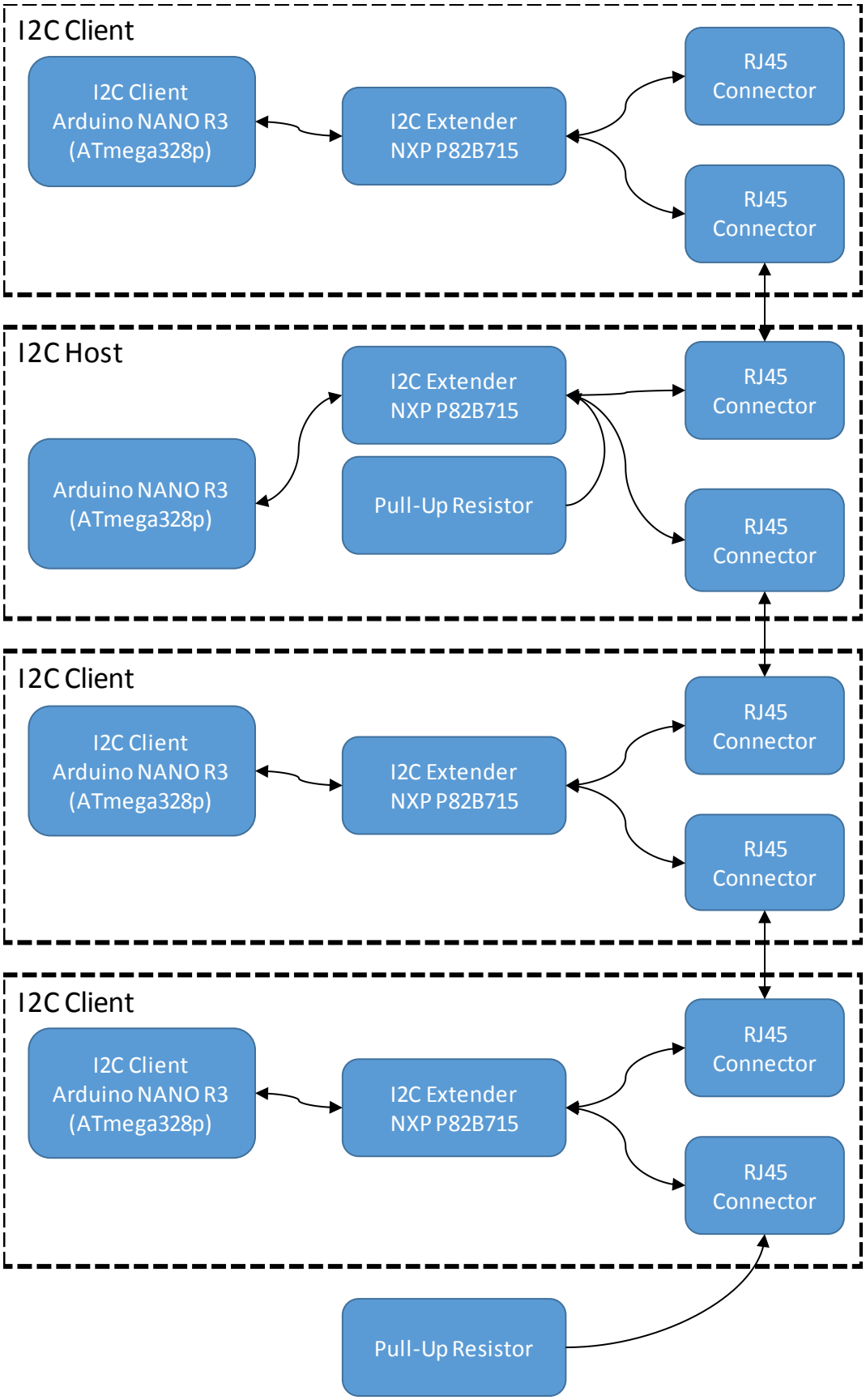
Jotta Host – ohjainta voitaisiin ohjata, tarvitsee se yhteyden keskusyksikköön. Yhteys keskusyksikköön on siis toteutettu Ethernet yhteyden avulla ja koska piiriltä ei ole suoraa tukea verkkoyhteydelle, tarvitsee se erillisen verkko-ohjaimen. Tähän ongelmaan apua tarjoaa Microchip Technology Inc. joka tarjoaa kohtuuhintaisen sekä luotettavan piirin verkkoliikenteen hoitamiseen. ENC28J60 – piiri tarjoaa vaatimattomalta kuulostavan 10Base-T verkkoyhteyden, mutta on enemmän kuin tarpeeksi tämän hetkisiin järjestelmän vaatimuksiin. Piiri käyttää SPI – väylää kommunikointiin Host – ohjaimen kanssa, jolloin ENC28J60 toimii eräänlaisena siltana Ethernet verkon ja SPI väylän välillä. Tarjolla on myös monia muita varteen otettavia vaihtoehtoja, kuten WIZnet Co. Ltd:n valmistama W5100 ja tästä uudempi versio W5200.

Molemmista edellä mainituista verkko-ohjaimista oli myös tarjolla valmis moduuli, johon tarvittavat ympäristökomponentit oli jo valmiiksi asennettuina. Microchip:in ENC28J60 piirin moduulikomponentti (Kuva 14) on hieman WIZnet:in W5200 kookkaampi, mutta hinnaltaan huomattavasti halvempi.



Kuva 14. ENC28J60 Ethernet moduuli

Kun verkkoyhteys sekä ohjain saatiin määriteltyä, oli aika keskittyä käytettävän aliväylän suunnitteluun. Koska ”Host to Client” yhteydessä käytettävä väylä I2C asettaa rajoituksia pituuden suhteen, lähdettiin asiaa ratkomaan kuten, kappaleessa 2.4.3 käsiteltiin ja päädyttiin seuraavanlaiseen ratkaisuun (Kuva 15).



Kuva 15. I^2C - väylän periaate

Kuten kuvasta voidaan todeta, avuksi otettiin väylän laajentajat jotka tarjoavat oikein mitoitettuina, jopa 50m matkan väyläliikenteelle. Väylälaitteiden kytkemiseksi toisiinsa

päädyttiin standardoituun RJ-45 liittimeen, jota löytyy mm. yleiskaapelointi verkoista. RJ-45 liittimiä löytyy suojattua(Shielded) sekä suojaamattomia(Unshielded). Työssä päädyttiin toteuttamaan parikaapeliyhteys suojatuin liittimin ja kaapelein, sillä erillisen suojafolion ansiosta, väylä on paremmin suojattu ympäristön häiriöiltä. Kaapeloinnin osalta valittiin perinteinen parikaapeli, joka parikiertonsa ja erillisen häiriösuojauksen ansiosta soveltuu mainiosti käytettäväksi valitussa väylässä sekä erilaisissa ympäristöissä.

Parikierrolla haetaan symmetriaa, joka puolestaan vähentää magneettikentän aiheuttamia häiriöitä, mutta toisaalta ei aiheuta häiriöitä ympäristöönsä. Parikaapelin yksittäisessä parissa tulisi siis kulkea itseisarvoltaan yhtä suuret, mutta vastakkaissuuntaiset virrat. Tällöin johtimien aiheuttamat magneettikentät kumoavat toisensa.

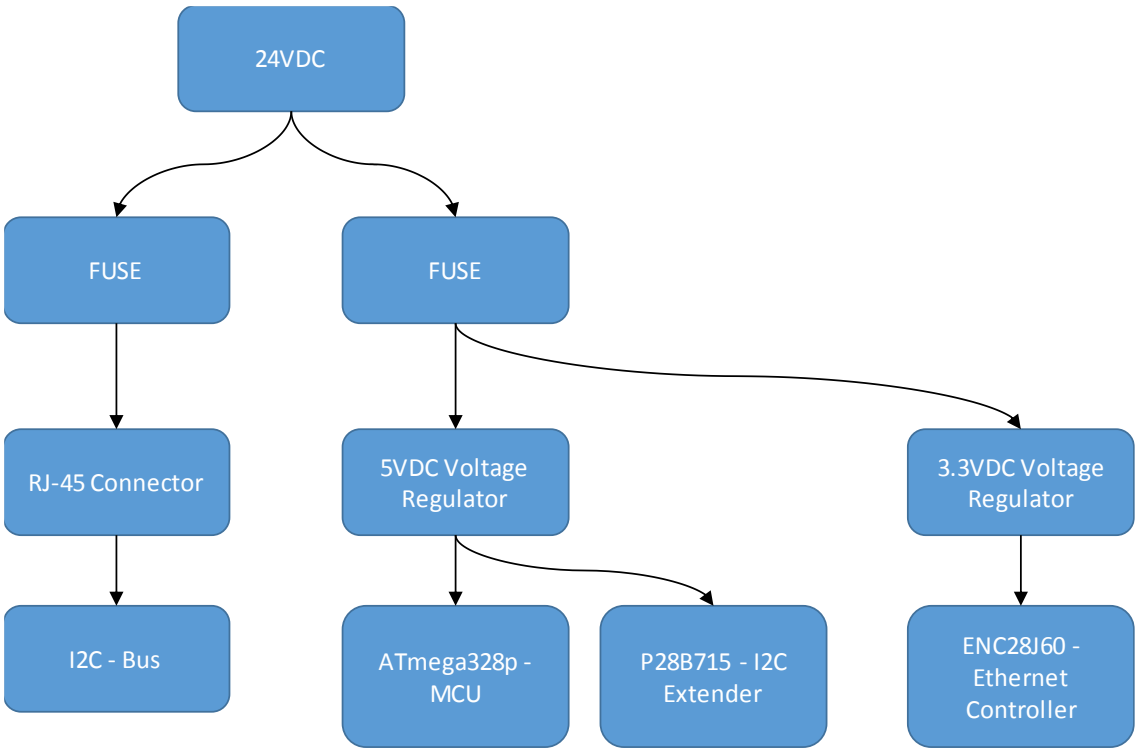
Kuten luvussa 2.4.3. todettiin I^2C – väylätekniikka käyttää ainoastaan kahta johdinta, ns. kellolinjaa sekä datalinjaa. nämä kaksi erillistä linjaa eivät toteuta em. sääntöä, jolla parikierron hyöty saataisiin käyttöön. Mikäli väylän linjat kytkettäisiin samaan pariin, häiritsisivät ne toisiaan (ylikuuluminen), jolloin menetettäisiin huomattava osa väylän pituudesta. Parempi tapa on siis yhdistää linjat maa potentiaalin kanssa samaan parikiertoon. Kuten edellä totesimme, ei ratkaisu itsessään tuota symmetriaa johtimiin, koska toinen parin johtimista on kytketty maapotentiaaliin ja toinen toimii väyläjohtimena. Tämä on kuitenkin järkevin tapa kytkeä tämän tyyppinen väylä kierrettyyn parikaapeliin. Toisaalta tällä järjestelyllä saadaan minimoitua väylän ylikuuluminen, koska parit eivät enää kulje samassa parikierrossa. Lisäksi saamme vähennettyä syntyvää induktanssia sekä saamme säilytettyä kaapelin kapasitanssin yhtenäisenä koko kaapelin matkan.

Väylän lisäksi kaapelissa tullaan kuljettamaan toimilaitteiden tarvitsema sähköenergia, sillä kaapelissa olevat vapaat parit soveltuvat tähän erinomaisesti. Johtimia tullaan käyttämään seuraavan taulukon mukaisesti (Taulukko 2).

PIN	Pari	Väri	Sisältö
1	3	Valkoinen/Vihreä	+24VDC
2	3	Vihreä	– GND
3	2	Valkoinen/Oranssi	– GND
4	1	Sininen	I2C -SDA (LX)
5	1	Valkoinen/Sininen	– GND
6	2	Oranssi	I2C - SCL (LY)
7	4	Valkoinen/Ruskea	
8	4	Ruskea	
Shield		Kirkas	-GND

Taulukko 2. RJ-45

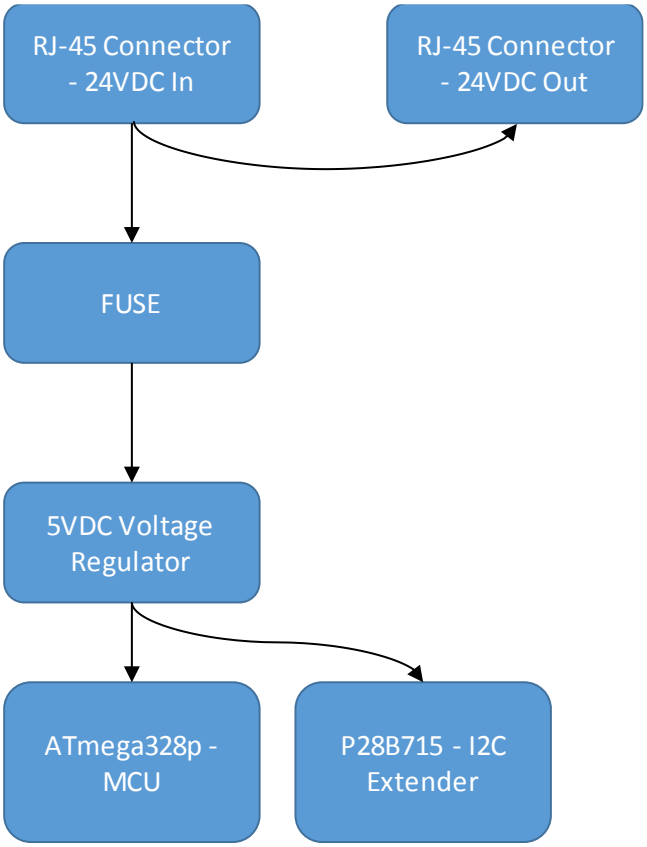
Koska käyttöjännitteeksi valittiin 24VDC ja kuitenkin piirit vaativat tätä pienempiä jännitteitä, tulee käyttöjännite säätää sopivaksi kullekin piirille. Host – ohjaimen MCU (Microcontroller Unit), joka hoitaa varsinaisen prosessoinnin, toimii tässä tapauksessa 5VDC jännitteellä. Samoin I^2C – väylää laajentavat P28B715 piirit. Toisin kuin edellä mainitut, Host – ohjaimen verkko-ohjain käyttää 3.3VDC jännitettä, joten se tulee saamaan oman jännitesyöttönsä erillisestä jännitteensäätimellä (Kuva 16)



Kuva 16. Sähköenergian jako Host - ohjaimen sisällä

3.2.3 I2C Client

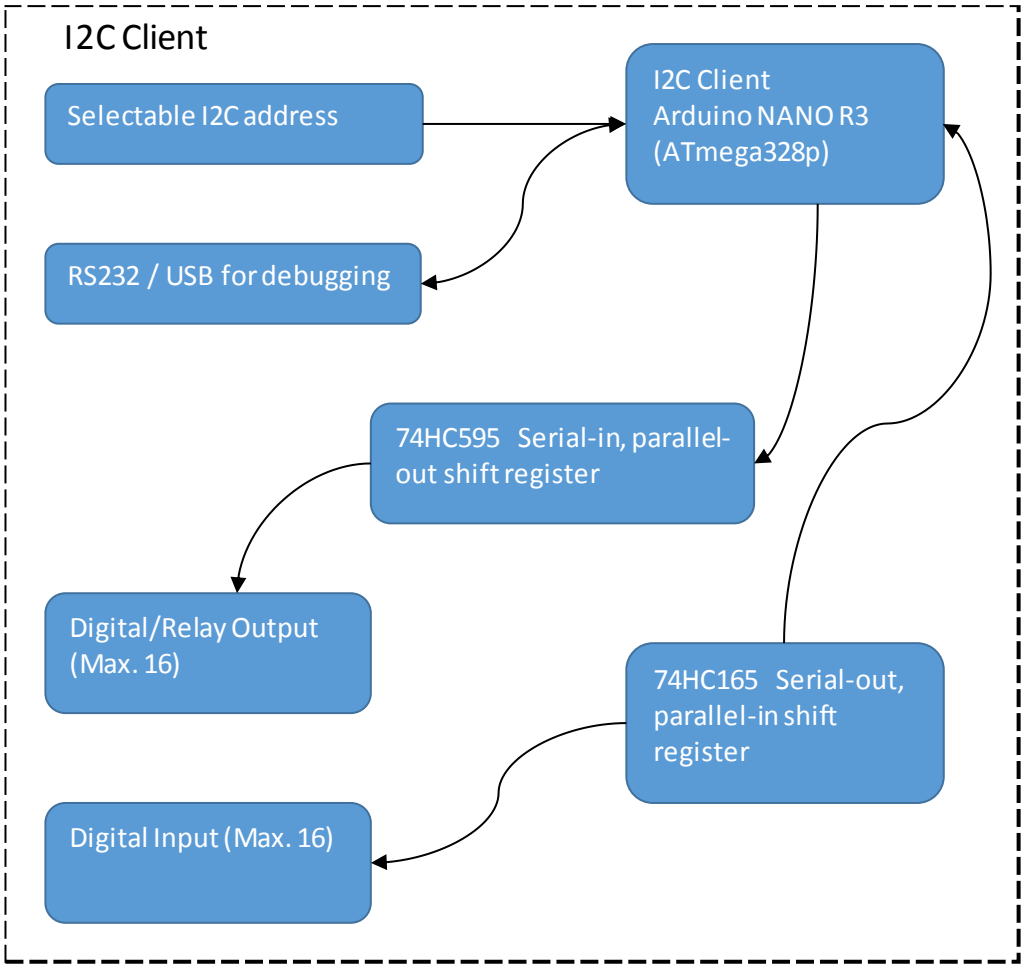
Client – ohjaimien kuten Host -ohjaimen käyttöjännite on siis pääasiassa 5VDC, mutta koska eräät toiminnot vaativat tätä korkeamman jännitteen, syötetään 24VDC myös aliverkon parikaapelissa. Kuten Host – ohjain, myös Client – ohjain vaatii erillisen jännite-regulaattorin sovittaakseen syötettävän jännitteen sopivaksi MCU:lle (Kuva 17).



Kuva 17. Sähköenergian syöttö Client - ohjain

Käyttöjännitteen lisäksi kaapeloinnissa kulkee maapotentiaali, joka on tärkeä osa väylän toimivuuden kannalta. Maapotentiaali toimii siis vertailutasona väyläsignaaliin nähden.

Toimilaitteita, jotka ovat liitettynä väylään, voidaan käyttää eri tarkoituksiin, kuten digitaaliset -tulot ja -lähdöt, relelähdöt tai analogiset signaalit, kuten lämpötila, valoisuus ja ilmanpaine. Kehityksen ollessa vielä alkuvaiheessa keskityttiin yksinkertaisimpiin toimijoihin eli digitaalisiin I/O laitteisiin(Kuva 18).



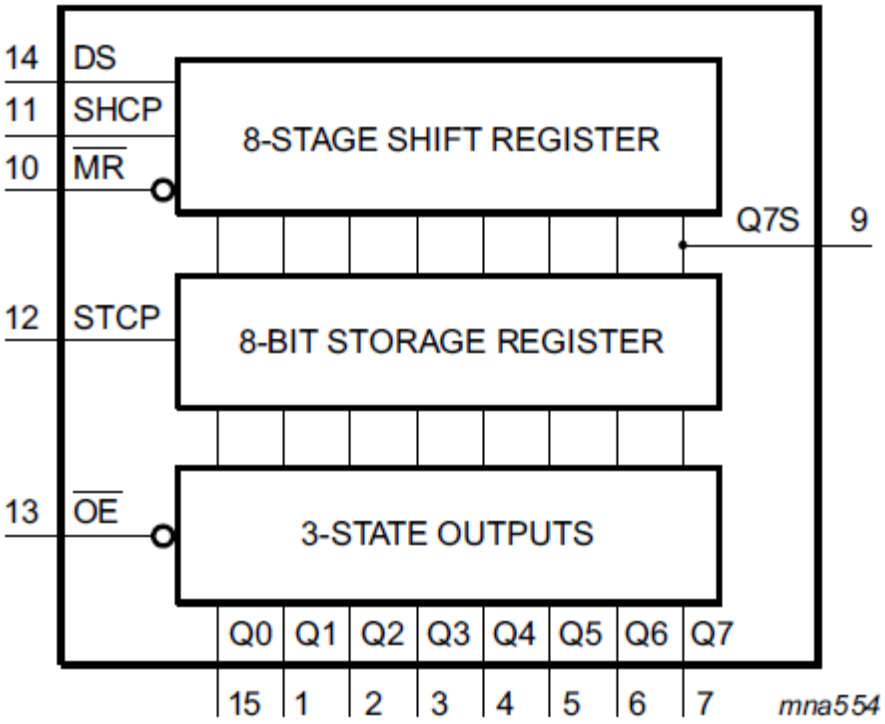
Kuva 18. Digitaalisen I/O CLient - ohjaimen periaate

Kuten kuvasta voimme todeta Client – ohjain sisältää myös ohjaimen, joka tässäkin tapauksessa on Atmega328p. Ohjain on varustettu Arduino käynnistyslataimella, jolloin sen uudelleen ohjelmointi sekä testaus on helpompaa, kuin suoraan piirille ohjelmoimalla.

Koska pelkkä ohjain sisältää vain 14 digitaalista I/O liityntää sekä 8 analogista tuloa, joita voidaan myös hyödyntää digitaalisina tuloina ja lähtöinä, joudutaan käyttämään erillistä siirtorekisteriä, jotta saadaan haluttu I/O määrä ulkoisia tuloja ja lähtöjä varten. Tästä voisi kuitenkin nopeasti tehdä johtopäätöksen, että jos tulot ja lähdöt määriteltäisiin erillisille Client – ohjaimille ei erillisiä siirtorekistereitä tarvita. Kaikkia MCU:n I/O pinnejä ei kuitenkaan päästä suoraan hyödyntämään, sillä osa pinneistä on varattu I2C – väylälle sekä ohjaimen konfigurointia varten näppäimistölle ja näytölle.

74HC595 on yksinkertainen 8-bittinen sarjaliikenne rekisteri, joka käytännössä ottaa sisäänsä syötetyt bitit ja asettaa ne lähtönastoihin haluttuna ajankohtana. Rekisteri vaatii toimiakseen kellolinjan, datalinjan sekä erillisen latauslinjan, jolla määritetään milloin

riittävä määrä bittejä on piiriin syötetty ja koska ne ladataan lähtörekisteriin (Kuva 8). (NXP Semiconductors. 74HC595 Product Datasheet.)



Kuva 19. 74HC595 toiminnallinen kaavio (NXP Semiconductors. 74HC595 Product Datasheet.)

Symbol	Pin	Description
Q1	1	parallel data output 1
Q2	2	parallel data output 2
Q3	3	parallel data output 3
Q4	4	parallel data output 4
Q5	5	parallel data output 5
Q6	6	parallel data output 6
Q7	7	parallel data output 7
GND	8	ground (0 V)
Q7S	9	serial data output
$\overline{MR}$	10	master reset (active LOW)
SHCP	11	shift register clock input
STCP	12	storage register clock input
$\overline{OE}$	13	output enable input (active LOW)
DS	14	serial data input
Q0	15	parallel data output 0
V_{CC}	16	supply voltage

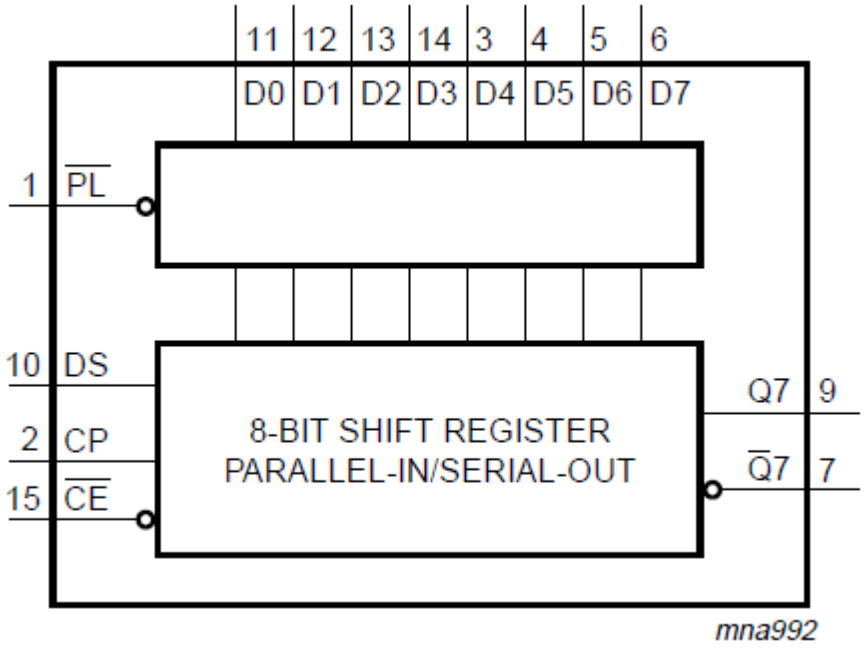
Taulukko 3. 74HC595 Pin description (NXP Semiconductors. 74HC595 Product Datasheet.)

Taulukosta 3 sekä kuvasta 19, voimme todeta, että piirillä on kahdeksan lähtöä, joita ohjataan erillisellä sarjaliikenne tulolla. Huomaamme myös, että piirissä lähtö Q7S on mää-

ritelty uloslähettäväksi data lähdöksi. Tämä tarkoittaa sitä, että piiriä on mahdollista ketjuttaa keskenään. Ketjuttamalla piirejä saadaan aikaan suurempia siirtorekisteri kokonaisuuksia, kuitenkaan käyttämättä enempää ohjainpiirin I/O pinnejä MCU:lta.

Kuvasta 18 huomaamme, että suunnitellussa järjestelmässä toimilaitteen lähtöjen määrä on määritelty enintään kuuteentoista. Tällöin käytetyn siirtorekisterin datalähtö ominaisuus tulee käyttöön. Tällöin ohjainlaitteen tuleekin syöttää siirtorekisterille 16 – bitin sarja, jolloin ensimmäiset kahdeksan bittiä menevät ketjun ensimmäiselle siirtorekisterille. Kun viimeistä kahdeksaa bittiä aletaan kirjoittamaan siirtorekisteriin, huomaa ensimmäinen siirtorekisteri tulleen saaneeksi jo nämä kahdeksan tarvitsemaansa bittiä, ja tällöin aina sen saadessa uuden bitin, se lähettää pinosta alimmaisen bitin eteenpäin, tässä tapauksessa seuraavalle siirtorekisterille, kunnes kaikki loputkin kahdeksan bittiä ovat saavuttaneet päämäärän. Vaikka rekisteriin on jo siirretty tarvittava määrä bittejä, eivät piirin lähdöt ole vielä asettuneet haluttuun tilaan. Kun viemme pulssin STCP tuloon, siirtää rekisteri bitit varastosta lähtöihin. (NXP Semiconductors. 74HC165Product Datasheet.)

74HC165 on vastaavanlainen 8-bittinen rekisteri, kuin 74HC595, mutta toiminta on päinvastainen. Piiri tallentaa kulloinkin voimassa olevat tulot erillisellä lataus pinnillä, jolloin tulot siirtyvät rekisteriin. Rekisteristä tulot saadaan sarjamuotoisena ohjaimeen, erillisen kellopulssin ohjaamana. Myöskin 74HC165 vaatii toimiakseen kellolinjan, datalinjan sekä erillisen latauslinjan (Kuva 9). (NXP Semiconductors. 74HC165 Product Datasheet.)



Kuva 20. 74HC165 Toimintakaavio (NXP Semiconductors. 74HC165 Product Datasheet.)

Symbol	Pin	Description
$\overline{PL}$	1	asynchronous parallel load input (active LOW)
CP	2	clock input (LOW-to-HIGH edge-triggered)
$\overline{Q7}$	7	complementary output from the last stage
GND	8	ground (0 V)
Q7	9	serial output from the last stage
DS	10	serial data input
D0 to D7	11, 12, 13, 14, 3, 4, 5, 6	parallel data inputs (also referred to as Dn)
$\overline{CE}$	15	clock enable input (active LOW)
V_{CC}	16	positive supply voltage

Kuva 21. 74HC165 Pin description (NXP Semiconductors. 74HC165 Product Datasheet.)

Tässäkin piirissä on vastaavanlainen ominaisuus, jolla piirejä voidaan ketjuttaa ja saada aikaan useampia tuloja järjestelmään. Tällä hetkellä suunniteltu tulojen enimmäismäärä yhdellä toimilaitteella on rajoitettu 16 tuloon, joten kahdella siirto rekisterillä homma hoituu.

Rekitereiden kuten monen muunkin IC piirin kanssa työskennellessä on syytä muistaa, että digitaalista lähtöä tai tuloa ei milloinkaan pitäisi jättää ns. ilmaan roikkumaan eli ”vapaaksi”. Tällä tarkoitetaan sitä, että ko. tuloon tai lähtöön ei ole kytketty mitään ja on tällöin potentiaalivapaassa tilassa. Näissä tapauksissa piiri voi toimia, ei toivotulla tavalla, koska tällöin ei ole varmuutta missä tilassa sen tulo tai lähtö pitäisi olla. Yleisesti lähtöjen

osalta tätä ongelmaa ei ole, sillä normaalisti niitä kuormitetaan jollain kuormalla, joka automaattisesti asettaa lähdön tiettyyn tilaan, kun sen ohjaaminen lopetetaan.

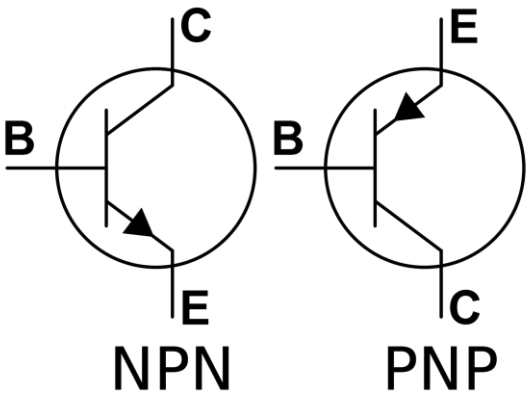
Ongelmaksi muodostuukin piirien tulot ja varsinkin mikäli niihin on liitetty erilaisia kytkimiä tai koskettimia. Esimerkiksi, siirtorekisterin tuloon tuodaan piirin käyttöjännitteen suuruinen jännite, joka nostaa piirin tulon yläasentoon (HIGH), jolloin jännite kiertää kytkimen tai koskettimen kärkien kautta. Tällöin ei vielä varsinaista ongelmaa ole, mutta kun kosketinpiiri avautuu, ei piiri enää tiedä missä tilassa sen tulo pitäisi olla. Miten asetamme tulon takaisin alatasoon (LOW)? Ratkaisuksi ongelmaan on alasvetovastus, joka pakottaa tulon takaisin alatilaan, kun jännite on hävinnyt tulosta.

Nämä apupiirit eivät toimi kovinkaan laajalla käyttöjännitealueella, ja tässä projektissa niitä käytetäänkin suositellulla 5V käyttöjännitettä.

Sellaisenaan piirit eivät ole vielä hirveän käyttökelpoisia, sillä 74HC595 tarjoaa 5V ulostulon ja enintään 35mA virran lähtöä kohden, kuitenkin niin, että kokonaisvirta ei saa ylittää 70mA virtaa. (NXP Semiconductors. 74HC165 Product Datasheet.)

Tulorekisteri (74HC165) sen sijaan voisi toimia jo sellaisenaan järjestelmässä, kunhan muistaa, että enintään sen käyttöjännitteen suuruiset jännitetulot ovat sallittuja. Tämän suhteen ei kuitenkaan lähdetty oikomaan, vaan päätettiin niin, että tulo kuin lähtörekisterit tulee toteuttaa 24VDC järjestelmään yhteensopivaksi, jolloin asennustilanteessa ei ole epäselvyyttä käytetystä jännitteestä.

Koska lähtörekisteri antaa vain enintään 35mA/70mA, tarvitsisi se myös virtavahvistusta. Virta ja jännitevahvistimeksi valittiin yleiskäyttöinen ja edullinen 2N3906 PNP-transistori (Kuva 22). Samoin tulopuolen rekisteri saatiin sovitettua 24V järjestelmään yhteensopivaksi transistori tekniikalla. Tähän tarkoitukseen otettiin käyttöön edellisen transistorin vastakohta eli 2N3904 NPN-transistori (Kuva 22).



Kuva 22. NPN ja PNP-transistorin piirrosmerkit ja merkinnät

2N3906 PNP-transistorilla on kohtalainen vahvistuskerroin 100–300, joka riittää hyvin tuottamaan riittävän virran digitaalisille ulostuloille. Koska valittu transistori on PNP-tyypin transistori, tarvitsee se pienemmässä potentiaalissa olevan jännitteen kannalle (Base), kuin emitterissä (Emitter) oleva jännite.

Kun transistorin kanta asetetaan alempaan potentiaaliin emitteriin nähden, alkaa emitteristä kollektorille virrata virta. Transistori vaatii kannalle emitterijännitettä pienemmän jännitteen tullakseen johtavaksi ja lähes emitterijännitettä vastaavan jännitteen muuttuakseen auki tilaan (ei johtavaksi). Koska siirtorekisterimme pystyy antamaan vain maapotentiaalin tai käyttöjännitteen, eli 5V, ei kytkentä sellaisenaan toimi, eli transistori ei koskaan olisi täysin auki tilassa. Joudumme siis lisäämään yhden ylimääräisen transistorin, jolla ohjaamme varsinaista lähtöä ohjaavaa transistoria. Ohjaavaksi transistoriksi valittiin jo edellä esitelty 2N3906 NPN-tyypin transistori. NPN-transistorin toiminta poikkeaa PNP-transistorin toiminnasta. NPN-transistoria käytettäessä kytkimenä on sen emitteri normaalisti kytketty maapotentiaaliin ja ohjattava suure kytkettynä kollektoriin. Tällöin kannalle syötettävä jännite on emitterijännitettä korkeammassa potentiaalissa, jotta saadaan transistori johtavaan tilaan, ja vastaavasti samassa potentiaalissa emitterijännitteen kanssa jotta transistori saadaan auki tilaan.

Jotta transistorien kantavirrat eivät nousisi tarpeettoman suuriksi, tulee niihin lisätä kantavastus. Kantavastus voidaan yksinkertaistetun mallin mukaan määrittää vahvistus kertoimen, halutun ohjausvirran sekä kantavirran mukaan. Koska järjestelmä koostuu kahdesta peräkkäisestä transistorista, ensimmäisen transistorin, jota siirtorekisteri ohjaa, ohjausvirta muodostuu viimeisen transistorin kantavirrasta. Mitoitusta lähdetään siis tarkastelemaan ohjattavan kuorman perusteella.

Mahdollisia ohjattavia kohteita voisivat olla esimerkiksi erilaiset kytkimien merkkilamput, liitynnät toiseen I/O:hon tai ulkoiset releohjaukset. Näiden rajausten perusteella asetettiin lähdön enimmäisvirta 50mA.

Yksinkertaistetun mallin mukaan voidaan siis todeta:

$$I_B = \frac{I_C(Max)}{h_{FE}(Min)}$$

Kuten aiemmin totesimme, 2N3906-PNP transistorin vahvistus kerroin on luokkaa 100–300. Datalehdestä voimme myös varmistaa transistorin maksimi kollektorivirran, joka on 200mA. Tämä riittää mainiosti suunniteltuun lähdön enimmäisvirtaan. Kantavirraksi saadaan siis:

$$\frac{0{,}05A}{100} = 0{,}005A = 5mA$$

Koska transistori on kytketty 24V jännitteeseen, tulee sitä käyttää laskiessa kantavastuksen suuruutta:

$$\frac{24V}{0{,}005A} = 48000\Omega = 4{,}8k\Omega$$

Saatu vastusarvo pyöristetään lähimpään helposti saatavilla olevaan arvoon, joka tapauksessamme on 4,7kΩ.

Samalla periaatteella laskemme myös ohjaavan transistorin kantavastuksen arvon. Tällöin transistorin mitoitus perustuu edellä lasketun transistorin kantavirtaan. Myös 2N3904 NPN-transistorin vahvistuskerroin on 100 – 300.

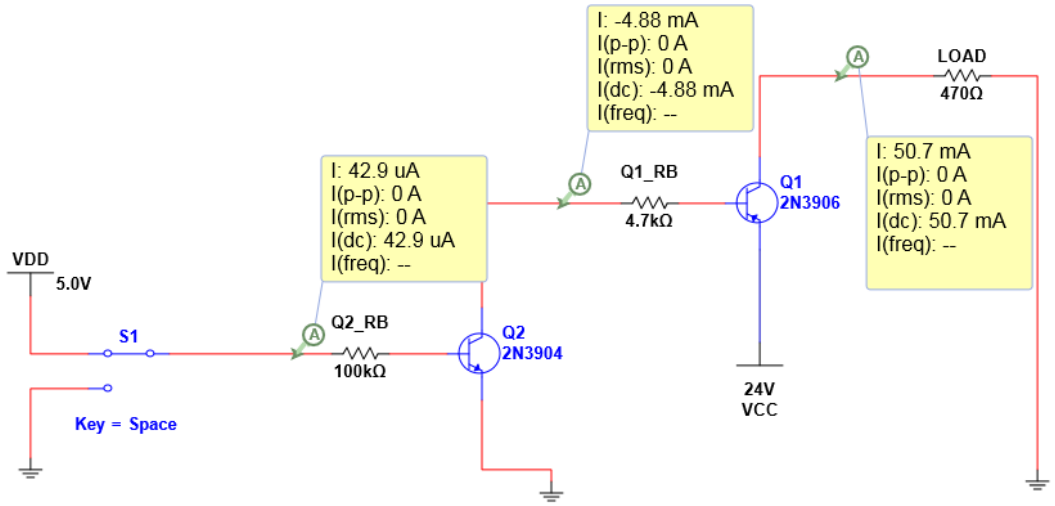
$$\frac{0{,}005A}{100} = 0{,}00005A = 50\mu A$$

Koska aputransistorin kannan ohjaus tulee siirtorekisteriltä, käytetään tällöin siirtorekisterin antamaa kannan jännitettä 5V.

$$\frac{5V}{0{,}00005A} = 100000\Omega = 100k\Omega$$

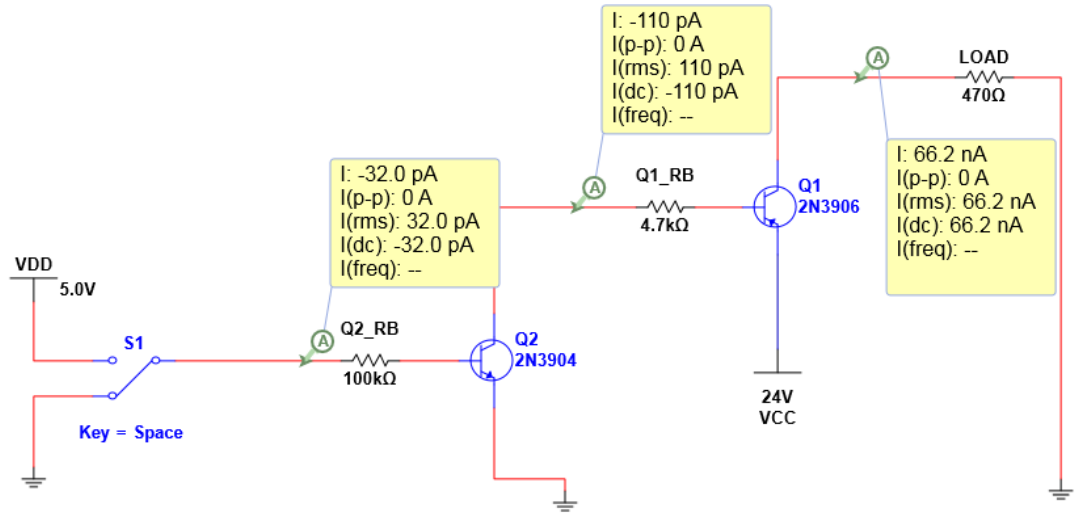
Kun alustavat laskelmat oli tehty, lähdettiin asiaa vielä varmistamaan simulointiohjelmalla.

Ilmaiseksi saatavilla oleva National Instruments:in Multsim Component Evaluator Mouser Electronics Edition on erinomainen työkalu elektroniikka piirien simuloimiseen.



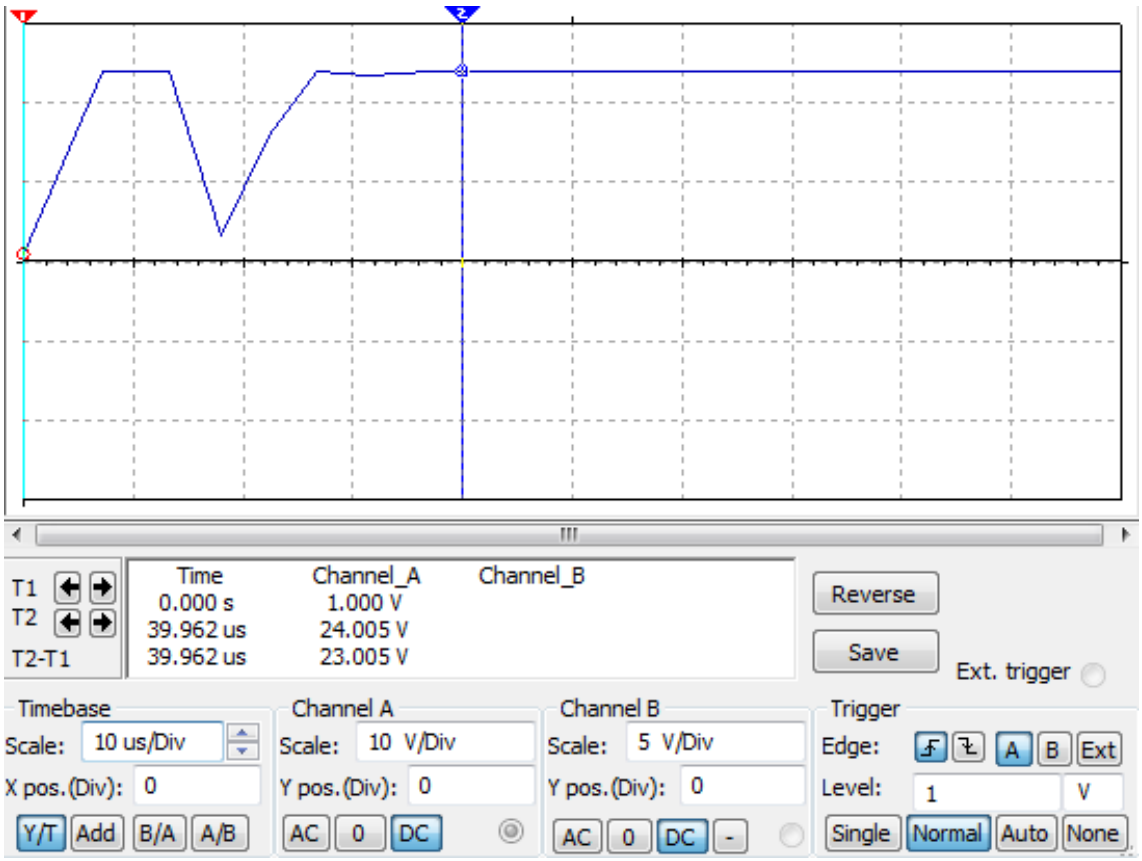
Kuva 23. Transistorikytkennän simulointi. Aktiivinen tila

Kuva 23 esittää käytettyä piiriä transistorien kantavastuksien varmistamiseksi. Kuvassa siirtorekisteri, jota simuloidaan kytkimellä S1, on tällä hetkellä aktiivisessa tilassa. Simuliointituloksista voimme havaita laskennan olevan hyvinkin lähellä simuloitua mallia. Malliin on lisäksi asetettu laskennallinen kuorma, jolloin lähtöön saadaan haluttu 50mA virta.



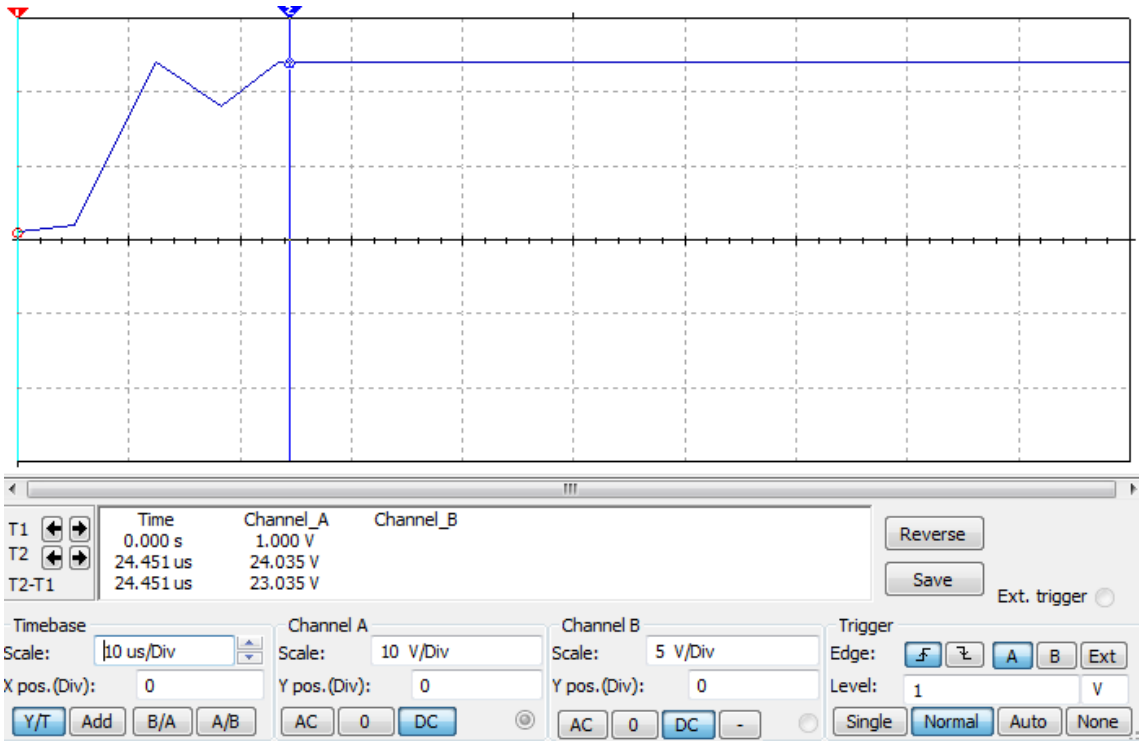
Kuva 24. Transistorikytkennän simulointi. Passiivinen tila

Kuvassa 24 on siirtorekisterin lähtö asettunut alatilaan, jolloin myös kuormalle syötettävä virta on lähes nollassa. Kytkennästä johtuen kuormassa kuitenkin esiintyy pientä vuotovirtaa, jonka lisäksi siellä vaikuttaa muutamien kymmenien mikrovolttien suuruinen jännite. Tämä johtuu Q1 transistorin osittain vain osittaisesta sulkeutumisesta. Lisäksi transistorin siirtyminen avoimesta tilasta suljettuun tilaan aiheuttaa huolestuttavaa huojuntaa (Kuva 25).



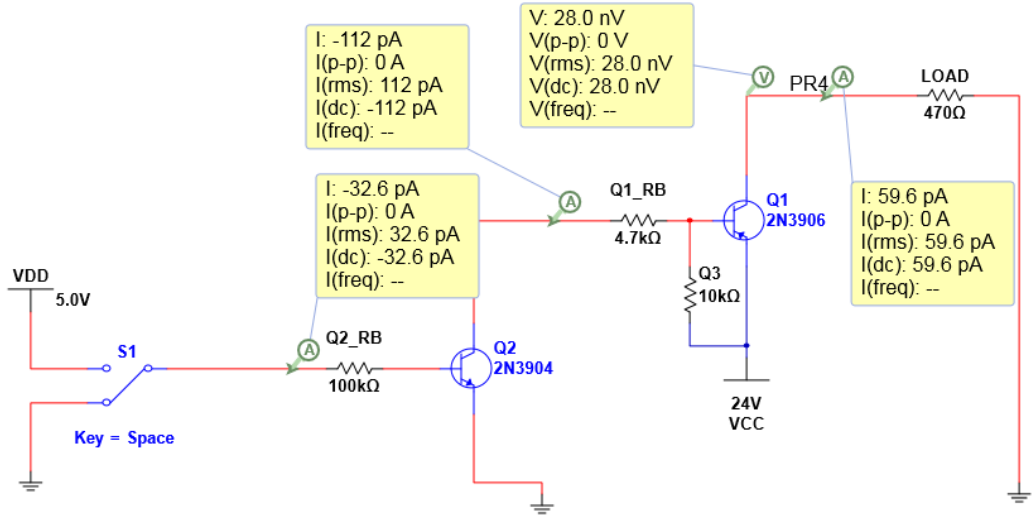
Kuva 25. Transistorin Q1 Uec jännite, ilman Q1 kannan ylös vetovastusta

Tilanne korjaantuu, kun lisätään transistoriin Q1 kannalle erillinen ylös vetovastus, joka pakottaa transistorin suljettuun tilaan, kun transistori Q2 on sulkeutunut (Kuva 26).



Kuva 26. Transistorin Q1 Uec jännite, Q1 ylös vetovastuksen kanssa

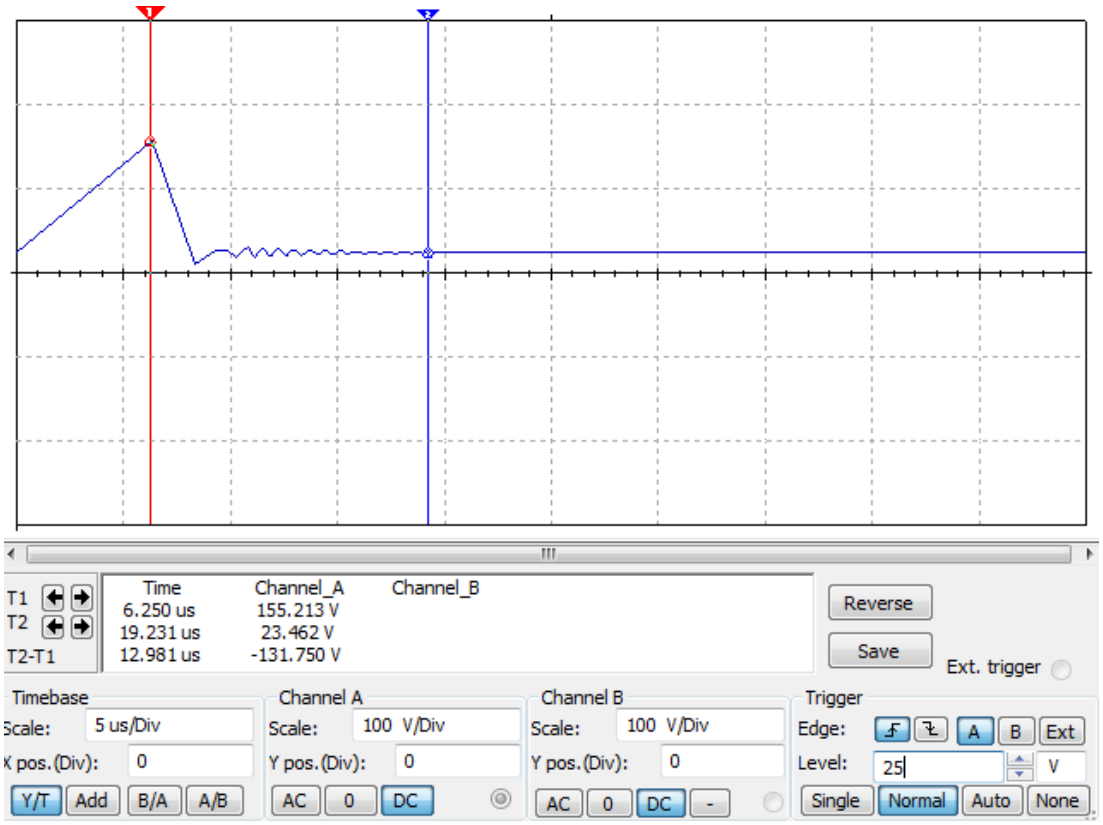
Kun transistorin kanta on vedetty käyttöjännitteeseen, myös sen antama vuotovirta häviää käytännössä olemattomiin ollen joitain pikoampeereja. Samalla myös lähdön jännite tippui nanovolttien tasolle (Kuva 27).



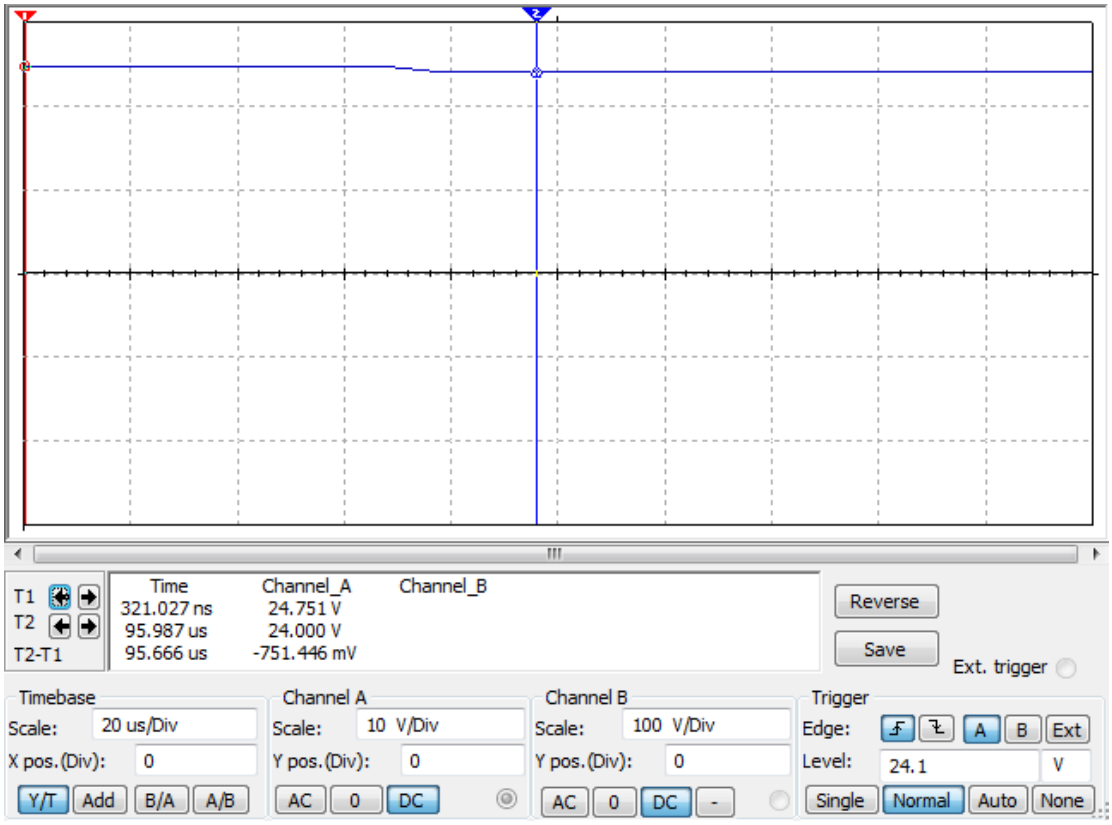
Kuva 27. Lopullinen simulointi kytkentä

Kytkentä siis toimii mainiosti kuorman ollessa resistiivinen.

Kun kuorman tyyppi muuttuu resistiivisestä vastuskuormasta induktiiviseen kelakuormaan, kohtaamme jälleen uuden ongelman. Kelan ollessa virrallinen indusoituu siihen jännite. Virran katketessa kelalta, pyrkii siihen indusoitunut jännite purkautumaan. Koska virtapiiri ei ole enää suljettu eikä näin ollen jännite pääse purkautumaan, tapahtuu kelassa nk. itseinduktio, joka lähtee kasvattamaan kelan jännitettä hallitsemattomasti (Kuva 28). Jossain vaiheessa jännite kasvaa niin suureksi, että tapahtuu läpilyönti. Läpilyönti tapahtuu transistorin Q1 puolijohdepinnoilla ja normaalisti tuhoaa transistorin. Induktiivisen kuormaan indusoitunut jännite täytyy siis hallitusti purkaa, vahingoittamatta muita komponentteja. Tällöin kytkentään lisätään ns. suojadiodi, jonka kautta jännite saadaan hallitusti purettua.

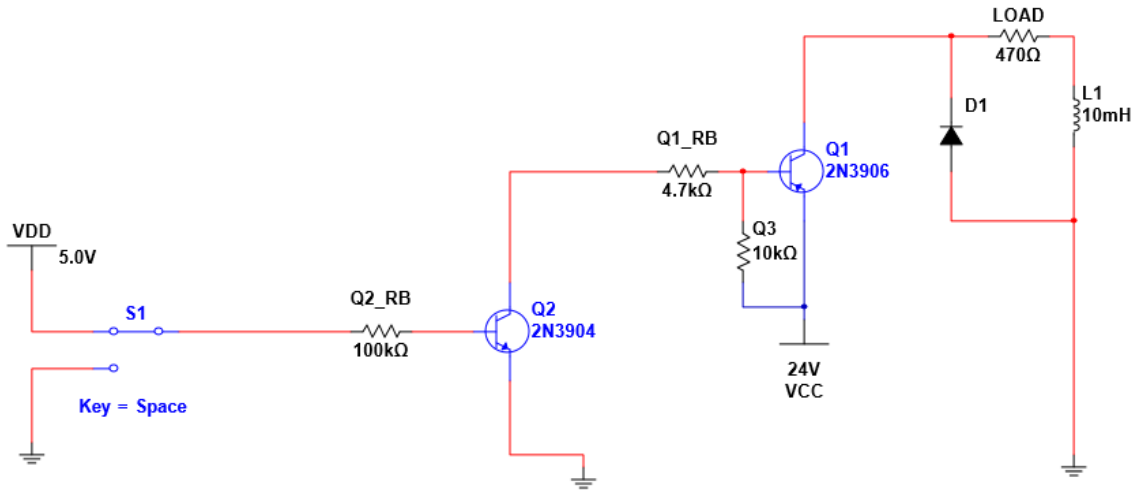


Kuva 28. Induktiivinen kuorma, ilman suojadiodia



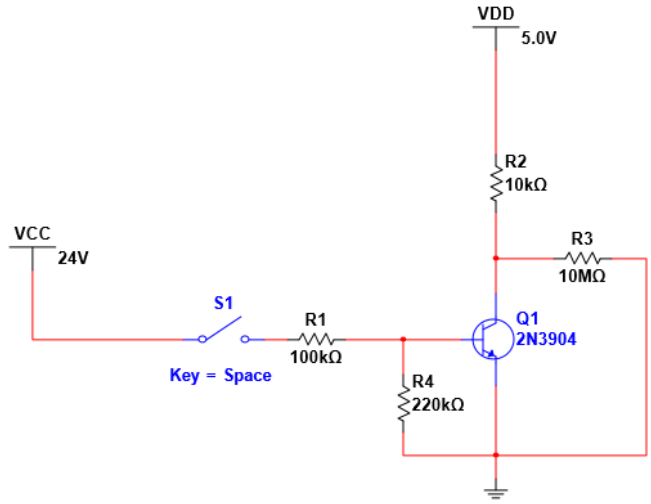
Kuva 29. Induktiivinen kuorma, suojadiodilla

Ilman suojadiodia olevassa tapauksessa voimme todeta, että virran katketessa, kela lähtee nopeasti kasvattamaan jännitettä sen kohotessa aina yli 150V. Datalehtien mukaan transistorin 2N3906 maksimi Uce jännite on 40V. Ilman suojadiodia tämä ylittyy moninkertaisesti ja suurella todennäköisyydellä aiheuttaa transistorin vaurioitumisen. Kun piiriin on kytketty suojadiodi, purkautuu kelan jännite hallitusti ja siististi, aiheuttaen vain alle yhden voltin jännitteen nousun. Suojadiodin tulee olla kytkettynä transistorin kollektorin ja maatason välille estosuuntaan (Kuva 30).



Kuva 30. Kytkentään lisätty suojadiodi D1

Tulopuolen sovittaminen 24V järjestelmään on hieman helpommin toteutettavissa, kuin lähtöpuoli. Vastaavasti käytämme hyödyksi transistoreja, kuten edellä todettiin. Toiminta on vastaavanlainen, kuin lähtöpuolella, mutta tällöin tulo ohjaa transistoria, joka puolestaan syöttää ohjausjännitteen siirtorekisterille. Tällöinkin on syytä käyttää transistorilla kantavastusta, vaikka siirtorekisterin tulon virta onkin noin 1nA luokkaa. (Kuva 31). Kytkentään on myös tällöin hyvä lisätä alas vetovastus, jolla transistori saadaan sammumaan nopeammin, eikä tällöin esiinny vuotovirtaa transistorin kollektorin ja emitterin välillä.



Kuva 31. Tulopuolen sovitus 24VDC järjestelmään

Kuvassa vastus R3 esittää siirtorekisterin sisääntuloimpedanssia, R2 toimii siirtorekisterin ylös vetovastuksena, R4 on transistorin kannan alas vetovastus ja vastus R1 transistorin kantavastus. Koska kytkimen S1 ollessa suljettuna, siirtorekisterin tuloon tuleva jännite asettuu maatasolle ja vastaavasti kytkimen ollessa auki asettuu tulo käyttöjännitteen (5V) tasolle, tulee tällöin ohjelmallisesti invertoida siirtorekisteriltä saatu tulotieto oikean toiminnan aikaan saamiseksi.

4 TOIMINTA

4.1 Järjestelmä

Järjestelmän keskusyksikkö toimii Linux käyttöjärjestelmän päällä. Käyttöjärjestelmästä on poistettu ns. kaikki turhuudet, kuten graafinen käyttöliittymä GNOME, KDE, jne., jolloin järjestelmän mahdollinen uudelleen käynnistys tapahtuu mahdollisimman nopeasti. Tällä tavoin saadaan myös hyödynnettyä prosessorin laskentateho mahdollisimman tehokkaasti. Käyttöjärjestelmään on kuitenkin asennettu joitain palveluita kuten SSH palvelu, jolloin järjestelmän etäkäyttö ja mahdolliset huoltotoimenpiteet voidaan suorittaa ilman fyysistä kosketusta keskusyksikön kanssa. Tällöin myös paikallisnäyttö ja muut HID (Human Initerface Device) laitteet ovat tarpeettomia.

Järjestelmän toiminta perustuu toimilaitteista saatuihin syötteisiin, niiden perusteella tehdyt päätelmät ja näiden hyödyntäminen toimilaitteiden lähdöissä.

4.2 Kehitysympäristö

Keskusyksikön käyttämä ohjelmisto on kirjoitettu C/C++ kielellä ja on lähes sellaisenaan yhteensopiva Windows sekä Linux ympäristössä. Ohjelmiston koekäyttö, virheenkorjaus sekä itse ohjelmointi tapahtui Windows ympäristössä, tämän tarjotessa jo ennestään tutut kehitystyökalut sekä tutun järjestelmäympäristön. Keskusyksikön ohjelmiston kehitykseen käytettiin Microsoft Visual Studio Community 2013, joka on saatavilla ilmaiseksi Microsoftin kotisivuilta. Kehitystyökalusta on saatavilla myös uudempia versioita esim. 2015, mutta yhteensopivuuden takaamiseksi tämän hetkisen kehitysvaiheen osalta päädyttiin pysyä hieman vanhemmassa ohjelmistoversiossa. Standardi kirjastojen lisäksi käytettiin Allego nimistä kirjastoa, joka tarjoaa helpon tuen mm. grafiikalle (2D), näppäimistölle, hiirelle, tiedostojärjestelmälle, asetustiedostoille, jne... Kirjasto on hyvin samankaltainen suositun SDL(Simple DirectMedia Layer) kirjaston kanssa, joka tarjoaa lähes samat ominaisuudet. Allegro sekä SDL kirjastojen hyvänä puolena voidaan pitää myös sitä, että molemmat kirjastot ovat lähes ”Platform Independent”, eli käyttöjärjestelmä ja alustavapaita. Esimerkiksi Allegro kirjasto toimii niin Windows ympäristössä, kuin myös Linux x86 sekä Linux AMR ympäristöissä. Lisäksi Allegro kirjaston etuna voidaan todeta, että kirjasto on jo entuudestaan tuttu, joten tämän osalta ei aikaa tuhraantunut uuden kirjaston opiskeluun.

Visual Studio toimii myös AVR prosessorien kehitysalustana. Lisäosana saatavilla oleva Visual Micro tuo Visual Studioon laajennuksen, joka sallii AVR prosessorien ohjelmoinnin suoraan ohjelmasta. Laajennus käyttää hyödykseen Arduino kehitysympäristössä mukana tulevia kirjastoja sekä kääntäjää AVR-gcc. Arduino ympäristössä toimiminen onnistuu myös heidän omalla kehitystyökalulla ArduinoIDE, mutta Visual Studio on tuttu jo vuosien takaa, jolloin sama kehitysympäristö keskusyksikön ja toimilaitteiden osalta on ihanneratkaisu. Lisäksi Arduinon oma kehitysympäristö huomattavan paljon riisutumpi ja ominaisuuksiltaan vähäisempi, kuin käytetty Visual Studio.

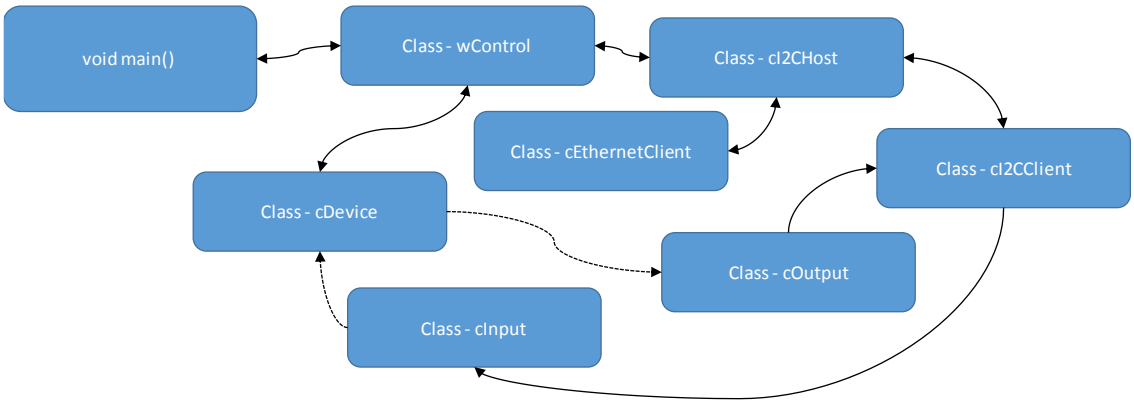
4.3 Järjestelmänohjelmisto

Ensimmäiset keskusyksikössä toimineet ohjelmistoversiot olivat lähinnä testimielessä tehtyjä ohjelmia jossa kaikki oli ns. ”hard coded”, eli kaikki muuttujat olivat ohjelmoitu ohjelmaan ja niiden muuttaminen asetustiedostojen tai muun yhteyden kautta oli mahdotonta.

Kun ohjelmistoa lopulta lähdettiin kehittämään joustavampaan suuntaan, lisättiin siihen ensimmäiseksi toimilaitteiden tietojen haku ja niiden vertailu asetustiedoston sisältämään kuvaukseen. Ennen kuin toimilaitteilta voidaan tietoa lähteä pyytämään, täytyy muodostaa yhteys I^2C Host – ohjaimeen, jonka kautta tieto toimilaitteille kulkee. Tämä prosessi olisi voitu suorittaa yksinkertaisesti skannaamalla IP avaruus lävitse ja katsoa mistä osoitteista vastaus löytyy. Host - ohjain täytyy kuitenkin jotenkin identifioida, joten päädyttiin IP osoitteen asettamiseen manuaalisesti, jolloin tiedetään, mikä laite pitäisi missäkin osoitteessa olla. Kun yhteys Host - ohjaimeen on saavutettu, lähetetään sille kysely I^2C väylässä sijaitsevista laitteista. Host – ohjain välittää kyselyn eteenpäin skannaamalla I^2C väylän koko osoiteavaruuden ja vastaanottamalla löydettyjen Client – ohjainten osoitteet ja tyypit. Nämä tallennetaan Host – ohjaimeen, joka lähettää ne takaisin keskusyksikköön, joka myös taltio saamansa tiedot. Tämän jälkeen keskusyksikkö vertailee asetustiedostossa asetettuja laitetietoja ja ilmoittaa mahdollisista ristiriidoista sekä mahdollisen uudelleen konfiguroinnin tarpeesta.

4.4 Ohjelmistorakenne

Itse ydinohjelmisto koostuu tuloista, lähdöistä ja laitteista sekä näiden välisistä vuorovaikutuksesta (Kuva 32).



Kuva 32. Ohjelmistorakenne ja yhteydet

Kuva esittää keskusyksikön ohjelmisto rakennetta yleisellä tasolla sekä luokkien välistä yhteyttä. Kuten tiedämme, jokainen ohjelma alkaa pääfunktiosta, niin kuin tässäkin tapauksessa ”void main()”. Pääfunktioon luodaan luokkarakenne wControl (Liite 1), joka pitää sisällään kaiken sen tiedon, jota järjestelmä tarvitsee toimiakseen. wControl luokka haaraantuu kahteen eri luokkatietueeseen. cI2CHost sekä cDevice.

cI2CHost luokka pitää sisällään puolestaan Host – ohjaimen alaisuudessa toimivat Client – ohjaimet sekä Host - ohjaimeen Ethernet yhteyttä ylläpitävä cEthernetClient. cI2CClient luokkaan on sisällytetty Client – ohjaimien tulot (cInput) sekä lähdöt (cOutput), jotka päivitetään luokkiin fyysisiltä laitteilta.

cDevice (Liite 2) luokka, joka toimii wControl pääluokan alaisuudessa, prosessoi tulojen perusteella lähtöjen arvot. Kuvassa 32 esitetyt yhteydet, cDevice sekä cInput (Liite 3) ja cOutput (Liite 4) luokkien välillä on esitetty katkoviivalla, johtuen yhteyden laadusta. Katkoviivalla esitetty yhteys kuvaa ainoastaan viitettä ko. elementtiin, tätä kutsutaan myös osoittimeksi. Yhtenäinen viiva sitä vastoin kuvaa fyysisesti muistipaikkaan sijoitettua tietoa tai luokkaa.
Tulot ja lähdöt ovat siis tallennettu cI2CClient luokan muistipaikkaan ja cDevice luokkaan on tallennettu viittaukset ko. tulon tai lähdön muistipaikkaan. Tällöin ei ohjelmassa tarvitse tehdä turhia kopioita muuttujista joka puolestaan säästää muistikapasiteettia. Tällä tavoin ehkäistään myös mahdollisia ohjelmavirheitä ja muistivuotoja, sillä muuttujia pitäisi kopioida luokasta toiseen, jotta molemmissa luokissa säilyisi samat kappaleet muuttujista. Eli jos cDevice asettaa lähdölle tilan 1, sama tila muuttuu välittömästi cI2CClient luokan lähtömuuttujaan. Ohjelmistorakenne myös mahdollistaa cDevice luokalle tulo- ja lähtömuuttujia useasta eri cI2CClient – luokasta, joten sitä ei ole sidottu yhteen ainoaan toimilaitteeseen tai Host – ohjaimeen.

cDevice sisältää siis toiminnallisen osan ohjelmasta ja säätelee sen tuottamaa toimintaa. Luokka on helposti rinnastettavissa perinteisen logiikkaohjelmoinnin erityyppisiin portteihin joilla saadaan annetuista tuloista tietty lähtöarvo. Tätä mallia onkin sovellettu ohjelmarakenteeseen tuomalla mukaan nk. virtuaaliset lähdöt ja tulot. Virtuaalisilla lähdöillä voidaan cDevice toimintoja ketjuttaa, kuin normaalissa logiikka ohjelmoinnissa yhdistettäisiin erilaisia loogisia operaatioita. Virtuaalisten lähtöjen lisäksi cDevice tuloihin voidaan liittää fyysisiä lähtöjä. Lisäksi virtuaalisia tuloja voidaan liittää esimerkiksi ajastimiin, kellokytkimiin tai käyttöliittymiin, jolloin järjestelmästä saadaan erittäin skaalautuva. Tällä hetkellä tuetut tulot ja lähdöt ovat kuitenkin ainoastaan tyypeiltään digitaalisia, mutta ohjelmistoon on rakennettu valmius analogisen tiedon käsittelylle, jolloin toimintojen ja toimilaitteiden kirjo kasvaa entisestään.

Tällä hetkellä tuettuja cDevice - tulopuolen loogisia operaatioita ovat:

- OR (Tai)
- AND (Ja)

cDevice – luokan tulopuoli edustaa siis perinteisiä loogisia operaatioita. Tämän hetken kehitysversiossa ei vielä ole toteutettu kaikkia loogisia operaatioita, kuten XOR, NAND, NOR, koska näille ei vielä testivaiheessa ole ollut tarvetta. Vastaavasti lähtöpuolella on omat operaationsa, jotka sitten jo hieman eroavatkin perinteisestä logiikka ajattelusta. Lähtöpuolen operaatioita ovat tällä hetkellä:

- Forced On / Off (Pakotettu tila, jolloin lähtö asetetaan joka kerta, kun luokka päivitetään)
- Latch High Edge (Tilan vaihto nousevalla reunalla)
- Latch Low Edge (Tilan vaihto laskevalla reunalla)
- Pulse High Edge (Pulssi nousevalla reunalla)
- Pulse Low Edge (Pulssi laskevalla reunalla)
- Pulse High + Low Edge (Pulssi nousevalla ja laskevalla reunalla)
- On High Edge + Off Low Edge (Päälle nousevalla reunalla ja Pois laskevalla reunalla)
- On High Edge (Päälle nousevalla reunalla)
- Off High Edge (Pois nousevalla reunalla)
- On Low Edge (Päälle laskevalla reunalla)

- Off Low Edge (Pois laskevalla reunalla)

Perinteisesti on totuttu, että logiikkaohjelmoinnissa lähtö päivitetään joka ohjelmakierron aikana (vrt. Forced On / Off). Työn mallissa on kuitenkin otettu erilainen lähtökohta lähtöjen ohjaukselle. Mallissa lähtöjä ei joka kerta aseteta tiettyyn tilaan (poikkeuksena Forced On / Off), vaan lähdön tila päivitetään muutoksen seurauksena, jolloin samaa lähtöä voidaan ohjata useassa eri cDevice – luokassa samanaikaisesti. Lisäksi mallissa on mahdollista ohjata useaa lähtöä samalla cDevice – luokalla.

Järjestelyllä on myös vaaransa mahdollisien tahattomien virheiden seurauksena tapahtuvien virheellisten ohjauksien muodossa.

Luokkaan on olemassa jo jatkokehitys-suunnitelma, jossa jokaiselle erilliselle tulolle ja lähdölle on määriteltävissä esimerkiksi seuraavia toisistaan erillään olevia toimintoja:

- Invertointi
- On-Delay
- Off-Delay

Näillä lisätoiminnoilla saadaan entisestään laajennettua toiminnollisuutta. Lisätoimintojen lisäksi on myös ajatuksen tasolla mietitty useamman cDevice luokan linkitystä uudelleen. Uuden mallin mukaan jokaiseen cDevice luokkaan voitaisiin tulomuuttujaksi asettaa toinen cDevice luokka, jolloin erillisiä välimuuttujia ei tarvittaisi.

4.5 Tiedonsiirto

4.5.1 Ethernet

Koska UDP – tekniikka ei sisällä kuittauksia lähetettyjen viestien perille menosta, on tämä ominaisuus kuitenkin välttämätön laitteiston toiminnan kannalta. Tekniikka, jolla varmistetaan sanoman perille meno, on varsin yksinkertainen. Keskusyksikkö lähettää sanoman Host – ohjaimelle, joka puolestaan välittää pyynnöt I^2C -väylää pitkin Client – ohjaimille. Tällöin keskusyksikkö lähettää koko tuntemansa I^2C – väylän ohjainten tiedot ja pyynnöt kerralla jolloin Host – ohjaimen tehtäväksi jää välittää ne eteenpäin oikeaan osoitteeseen. Host – ohjaimen saatua tieto Client – ohjaimilta, yhdistää se saadut tiedot ja lähettää ne takaisin keskusyksikölle. Em. syklillä keskusyksikkö voi varmistua, että tieto on saavuttanut Host – ohjaimen (Kuva 33).

Byte	1	2	3	4
CPU -> I2CHost	FB	0x54	0x3B	0x3E

Start Byte
Command - Type Request
Int. Mark
End Byte

Kuva 33. Host – ohjaimen I²C väylään liitettyjen laitteiden tyyppien tiedustelu.

Sanoma sisältää ensimmäiseksi yhteyden alkuvaiheessa generoidun satunnaisen tavun (”FirstByte”). Tällä pyritään estämään samassa verkossa toimivien muiden tahaton tai tahallinen toimilaitteiden ohjaaminen. Seuraavaksi on tavu, joka kertoo mitä sanoma pitää sisällään. Keskusyksikön lähettämässä sanomassa on lisäksi tavu 3, joka on eräänlainen välimerkki, joka on käytössä monessa keskusyksikön lähettämässä sanomassa päättämässä sanoman osion. Viimeisenä tavuna sanomassa on lopetustavu, jolla todennetaan vastaanottajalle viestin päättyminen. Lopetustavulla ei varsinaisesti todeta viestin päättymistä, vaan siitä lähinnä tarkistetaan viestin rakenteen oikeellisuus ja se, että viesti on saapunut kokonaisuudessaan perille. Host – ohjain vastaa vastaavasti keskusyksikölle, mutta sisällyttää nyt löytämänsä Client – toimilaitteiden määrän, osoitteen sekä tyypin sanomaan (Kuva 34).

Byte	1	2	3	4	5	6	7	x	x	x+n
I2CHost -> CPU	FB	0x54	X	X	X	X	X	...	...	0x3E

Start Byte
Command - Type Request
Number Of I2C Clients
I2C Client 1 Address
I2C Client 1 Type
I2C Client 2 Address
I2C Client 2 Type
...
...
End Byte

Kuva 34. Host – ohjaimen vastaus I²C väylään liitetyistä laitteista.

4.5.2 I2C Host

Varsinainen tiedonvaihto keskusyksikön ja Host – ohjaimen välillä aloittaa toimintansa, kun järjestelmä on todettu asetustiedoston kanssa yhteensopivaksi. Keskusyksikkö kokoaa Host – ohjaimelle lähetettävän sanomanohjaimen Client – laitteiden tyyppien mukaan (Kuva 38).

Sanoma sisältää Client – laitteiden mahdolliset ohjattavat lähdöt, mutta kuitenkin aina toimilaitteen osoitteen, sillä kaikilla toimilaitteilla ei ole ohjattavia lähtöjä tai suureita. Jos esimerkiksi järjestelmä sisältäisi yhden Host – ohjaimen ja tähän olisi liitettynä seuraavat Client – toimilaitteet: 8xDI(10), 8xDI+8xDO(50), 8xDO(89), muodostuisi lähetettävä sanoma kuvan 35 mukaiseksi.

Byte	1	2	2	3	4	5	6	7	8
CPU -> I2CHost	FB	0x44	0x03	0x0A	0x32	X	0x59	X	0x3E
	Start Byte	Command - Data exchange	Number of I2C lients	I2C Client 1 Address	I2C Client 2 Address	I2C Client 2 Outputs 1-8	I2C Client 3 Address	I2C Client 2 Outputs 1-8	End Byte

Kuva 35. Esimerkkitapauksen lähetettävä sanoma

Host – Ohjaimen saatua tieto keskusyksiköltä alkaa se purkamaan viestiä, joka sisältää siis erillisen aloitustavun, erillisen käskytavun, Client – ohjainten määrän, -osoitteen sekä mahdollisen lähtöjen tilan (em. tapauksessa 8-bittiä, tavu 5). Lopuksi viesti lopetetaan lopetustavulla. Mikäli viestissä ilmenee epämääräisyyksiä tai virheitä, Host – ohjain hylkää viestin ja jää odottamaan uutta viestiä. Mikäli viestin aloitus sekä lopetustavu ovat ohjaimen kannalta hyväksyttäviä, lähtee se purkamaan viestiä tavu kerrallaan. Samalla kun viestiä puretaan, lähetetään siitä soveltuvat osat eteenpäin osoitetiedon perusteella. Kun kaikki tieto on käsitelty, Host – ohjain lähtee tiedustelemaan Client – ohjaimilta vastaavasti tietoa lähetettäväksi takaisin keskusyksikölle. Ohjain käy läpi jokaisen pyydetyn osoitteen ja yhdistää niistä saadut viestit yhdeksi kokonaisuudeksi, vastaavasti kuin keskusyksikön lähettämässä sanomassa, ja lähettää ne taikaisin keskusyksikölle (Kuva 36).

Byte	1	2	2	3	4	5	6	7
I2CHost -> CPU	FB	0x44	0x02	0x0A	X	0x32	X	0x3E
	Start Byte	Command - Data exchange	Number of I2C lients	I2C Client 1 Address	I2C Client 1 Inputs 1-8	I2C Client 2 Address	I2C Client 2 Inputs 1-8	End Byte

Kuva 36. Esimerkki I²C Host vastaus keskusyksikölle.

Vastauksessa huomioitavaa on, että Host – ohjaimen lähettämä I2C Client määrä poikkeaa järjestelmässä olevasta toimilaitteiden määrästä. Tämä johtuu siitä, että toimilaitteella (8xDO) ei ole yhtään tulotietoa, jota keskusyksikkö voisi hyödyntää, joten sitä ei sisällytetty sanomaan. Koska kehityksen kohteena oleva järjestelmä on vielä varsin testaus vaiheessa, on sanomarakenteet pyritty yksinkertaistamaan mahdollisten virheiden etsimisen kannalta. Tällaisessa sanomarakenteessa emme siis voi tietää, onko sanoma toimilaitteelle 89 mennyt perille, sillä mitään kuittausta ei em. toimilaitteelta ole pyydetty. Olisikin siis hyvä sisällyttää ainakin osoitetieto sanomaan, jotta tiedetään toimilaitteen olemassaolo (Kuva 37).

Byte	1	2	2	3	4	5	6	7	8
I2CHost -> CPU	FB	0x44	0x03	0x0A	X	0x32	X	0x3E	0x3E
	Start Byte	Command - Data exchange	Number of I2C lients	I2C Client 1 Address	I2C Client 1 Inputs 1-8	I2C Client 2 Address	I2C Client 2 Inputs 1-8	I2C Client 3 Address	End Byte

Kuva 37. Parannettu I²C Host vastaus keskusyksikölle.

4.5.3 I2C Client

Kun Host – ohjain on vastaanottanut viestin keskusyksiköltä, lähtee se purkamaan sitä ja välittää sen eteenpäin Client – toimilaitteelle. Tällöin Ethernet rajapinta vaihtuu I²C väylään, mutta viesti muoto säilyy samankaltaisena. I²C väylässä on kuitenkin yleisesti käy-

tössä rekisterit jonne kirjoitetaan tai josta luetaan. Järjestelmään rakennettiin hieman samankaltainen rakenne. Viesti sisältää toimilaitteen rekisterin (Kuva 39), sekä varsinaisen kirjoitettavan datan ja lopetustavun. Mikäli vastaanottaja on tyypiltään sellainen, joka ei sisällä ohjattavia lähtöjä, ei data osuutta viestissä ole. Järjestelmään on tyypitetty kuvan 38 mukaiset toimilaiteet sekä niiden tunnukset.

Nimike	Tunnus	Tulot	Lähdöt	Tyyppi
DI8	0x0A	8	0	24VDC Tulot
DI16	0x0B	16	0	24VDC Tulot
DI8DO8	0x0C	8	8	24VDC Tulot / 50mA transistori lähdöt (PNP)
DO8	0x0D	0	8	50mA transistori lähdöt (PNP)
DO16	0x0E	0	16	50mA transistori lähdöt (PNP)
RO8	0x14	0	8	250VAC/16A relelähtö

Kuva 38.I2C Client digitaaliset tyypit

Jos oletamme, että vastaanottaja olisi tyypiltään DOx16, olisi kirjoitettavan datan osuus tällöin kaksi tavua. Järjestelmään on siis rakennettu kuvan 39 mukaiset rekisterit joiden avulla toimilaitteiden kanssa kommunikoidaan.

Rekisteri	Nimike	Data	Tyypit		
0x10	Asettaa lähdöt 1-8	1 bytes	DO8	DI8DO8	RO8
0x11	Asettaa lähdöt 1-16	2 bytes	DO16		
0x12	Pyytää tulot 1-8		DI8DO8	DI8	
0x13	Pyytää tulot 1-16		DI16		
0x14	Asettaa lähdöt 1-8 ja pyytää tulot 1-8	1 bytes	DI8DO8		
0x52	Reboot		Kaikki		
0x54	Type request		Kaikki		

Kuva 39. I2C Client rekisterit

Kuten kuvasta 39 voimme todeta, esimerkiksi toimilaite, jolla on kahdeksan tuloa ja kahdeksan lähtöä, voidaan tuolta toimilaitteelta pyytää ainoastaan tulotiedot, mutta ei asettaa lähtöjä. Toimilaitteiden tulojen ja lähtöjen tiedonvaihdon lisäksi, on määritelty rekisteri tyypin tiedustelulle sekä rekisteri uudelleen käynnistystä varten. Toimilaitteiden rekisteri käsittää 8-bitin rekisteriavaruuden, jolla katetaan 256 eri rekisteriä. Samaa osoiteavaruutta käytetään myös toimilaitteiden tyypeissä.

5 POHDINTA

Työn alussa oli olemassa ajatuksen tasolla suunnitelma toimivasta automaatiojärjestelmästä, mutta työn edetessä törmättiin useisiin haasteisiin, joista ei alussa ollut vielä tietoa. Järjestelmän rakenne koki muutoksen tiedonsiirtotekniikan osalta, jossa siirryttiin aliväyliin, tällöin myös järjestelmän suunnittelun kokonaisuuden hallintaan liittyvät kysymykset toivat uusia haasteita.

Järjestelmän suunnittelun kokonaisuuden hallinta yllätti sen monimuotoisuudellaan, sillä yhden henkilön projektissa dokumentoinnilla ja järjestelmällisyydellä on suuri rooli toimivan kokonaisuuden aikaan saamiseksi. Myös tiedonsiirtyminen verkkotekniikasta osittain väylätekniikkaan toi projektiin lisää haasteita, jolloin ei riittänyt vain yhden tiedonsiirtotekniikan toteuttaminen, vaan kahden erillisen tekniikan toimiminen saumattomasti yhteen.

Työn tuloksena suunniteltu järjestelmä ei ole valmis, vaan antaa hyvän lähtötilanteen todellisen suunnittelun pohjaksi. Uusia ideoita, toimintoja ja parannuksia järjestelmään on jo huomattavasti, mutta toistaiseksi resurssipula on jarruttanut järjestelmän kehitystyötä. Järjestelmästä on käynnissä joitain pieniä testisovelluksia, joiden perusteella kehitystyötä jatketaan toimivan kokonaisuuden aikaan saamiseksi.

Myös tämän hetkinen dokumentointi, niin ohjelmiston, kuin muun materiaalin osalta on ainoastaan omaan käyttöön tarkoitettua ja ei sellaisenaan sovellu laajempaan levikkiin. Dokumentoinnin yhtenäistäminen sekä ohjelmiston koodin yhtenäistäminen sekä jonkinlaisten käytäntöjen käyttöönottaminen olisi ensisijaisen tärkeätä, mikäli projektia haluttaisiin saada yleiseen levitykseen. Apua tullaan siis tarvitsemaan, mikäli järjestelmästä haluttaisiin laajemmalle levitykselle sopiva formaatti, sillä jo itse projektinhallinta on yhdelle ihmiselle vaativaa.

LÄHTEET

Anttila, A. 2000. TCP/IP-tekniikka. Juva: WSOY – Kirjapainoyksikkö.

Arduino.cc. Viitattu 11.11.2015. https://www.arduino.cc/en/Main/ArduinoBoardNano

I²C Bus. Viitattu 9.11.2015. http://www.i2c-bus.org/

I²C Bus Specification. Viitattu 9.11.2015. http://i2c.info/i2c-bus-specification

I²C Wikipedia. Viitattu 9.11.2015. https://en.wikipedia.org/wiki/I%C2%B2C

Microsoft, TCP/IP Protocol Architecture. Viitattu 9.11.2015. https://technet.microsoft.com/en-us/library/cc958821.aspx

NXP Semiconductors. 74HC165 Product Datasheet. Viitattu 10.11.2015. http://www.nxp.com/documents/data_sheet/74HC_HCT165.pdf

NXP Semiconductors. 74HC595 Product Datasheet. Viitattu 10.11.2015. http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf

NXP Semiconductors. P82B715 Product data sheet. Viitattu 9.11.2015. http://www.nxp.com/documents/data_sheet/P82B715.pdf

SparkFun. I²C Tutorial. Viitattu 11.11.2015. https://learn.sparkfun.com/tutorials/i2c

SPI - Serial Peripheral Interface. Viitattu 9.11.2015. http://www.mct.net/faq/spi.html

LIITTEET

Liite 1. Järjestelmän pääohjelman rakennetta kuvaava otsikkotiedosto wControl.h

```
#ifndef wControl_h_
#define wControl_h_


#include "main.h"
#include "cDevice.h"
#include "cI2CHost.h"

//Main class
class wControl
{
private:
    vector <cI2CHost>   nI2CHost;
    vector <cDevice>    nDevice;

    vector <cInput>     nVirtualInput;
    vector <cOutput>    nVirtualOutput;

    void    UpdateDevicePointers();

public:

    wControl();
    ~wControl();

    void        Clear();

    cDevice     *GetDevice(int Num);
    cI2CHost    *GetI2CHost(int Num);

    int         GetNumDevices();
    int         GetNumI2CHosts();

    bool        CreateDevices(int Num);
    bool        CreateI2CHost(int Num);

    bool        AddDevice();
    bool        AddI2CHost();

    void        DeleteDevice(int Num);
    void        DeleteI2CHost(int Num);

    bool        AddVirtualInput(int Type, bool Inverted);
    bool        AddVirtualInput(int Type);
    bool        AddVirtualOutput(int Type);

    cInput      *GetVirtualInput(int Num);
    cOutput     *GetVirtualOutput(int Num);

    void        Update();
};


#endif
```

Liite 2. Järjestelmän laitteiden rakennetta kuvaava otsikkotiedosto cDevice.h. 1 (2)

```
#ifndef cDevice_h_
#define cDevice_h_

#include "Defines.h"
#include "cInput.h"
#include "cOutput.h"

//Digital Output Types
#define dDeviceOutputForcedOnOff                0
#define dDeviceOutputLatchHighEdge              1
#define dDeviceOutputLatchLowEdge               2
#define dDeviceOutputPulseHighEdge              3
#define dDeviceOutputPulseLowEdge               4
#define dDeviceOutputPulseHighEdgePulseLowEdge  5
#define dDeviceOutputOnHighEdgeOffLowEdge       6
#define dDeviceOutputOnHighEdge                 7
#define dDeviceOutputOffHighEdge                8
#define dDeviceOutputOnLowEdge                  9
#define dDeviceOutputOffLowEdge                 10

//Analog Output Types
#define dDeviceOutputAnalog                     100

//Digital Input Types
#define dDeviceInputOR          0   //OR
#define dDeviceInputAND         1   //AND

//Analog Input Types
#define dDeviceInputAnalog      100

//Device Type
#define dDeviceDigitalInputDigitalOutput    1
#define dDeviceDigitalInputAnalogOutput     2
#define dDeviceAnalogInputDigitalOutput     3
#define dDeviceAnalogInputAnalogOutput      4


class cDevice
{
protected:
    int     nType;

    bool    nActive;
    int     nInputType;
    int     nOutputType;
    bool    nForcedState;

    unsigned int nInputValue;
    unsigned int nPrevInputValue;
    unsigned int nOutputValue;
    unsigned int nPrevOutputValue;
    unsigned int nForcedOutputValue;


    vector <unsigned int> nInputId;
    vector <unsigned int> nOutputToInputId;
    vector <unsigned int> nOutputId;

    vector <cInput*>    nInput;
    vector <cOutput*>   nOutputToInput;
    vector <cOutput*>   nOutput;

    char    nName[dDefaultStringLenght];

    void    UpdateDigitalInput();
    void    UpdateDigitalOutput();
    void    UpdateAnalogInput();
    void    UpdateAnalogOutput();

public:
    cDevice();
    ~cDevice();

    cDevice operator = (cDevice&);

```

```
76      void    Clear();
77
78      void    SetType(int Type);
79      int     GetType();
80
81      void    SetActive(bool Active);
82      bool    GetActive();
83
84      void    SetInputType(int Type);
85      int     GetInputType();
86
87      void    SetOutputType(int Type);
88      int     GetOutputType();
89
90      int     GetInputValue();
91      int     GetInputPrevValue();
92
93      int     GetOutputValue();
94      int     GetOutputPrevValue();
95
96      void    SetOutputValue(int Value);
97      void    SetForcedOutputValue(int Value);
98      void    SetForcedOutput(bool Forced);
99      bool    GetForcedOutput();
100     void    ChangeOutputState(); //Only used in digital output mode
101
102     bool    AddInput(cInput *Input);
103     void    DeleteInput(unsigned int Num);
104     void    SetInput(unsigned int Num, cInput *Input);
105     cInput  *GetInput(unsigned int Num);
106     int     GetNumInputs();
107
108     bool    AddOutput(cOutput *Output);
109     void    DeleteOutput(unsigned int Num);
110     void    SetOutput(unsigned int Num, cOutput *Output);
111     cOutput *GetOutput(unsigned int Num);
112     int     GetNumOutputs();
113
114     unsigned int    GetInputId(unsigned int Num);
115     unsigned int    GetOutputId(unsigned int Num);
116
117     void    SetName(char *Name);
118     char    *GetName();
119
120     void    Update();
121 };
122
123 #endif
```

Liite 3. Järjestelmän tulojen rakennetta kuvaava otsikkotiedosto cInput.h

```
#ifndef cInput_h_
#define cInput_h_

#include "Defines.h"

#define dInputTypeDigital   0
#define dInputTypeAnalog    1

#define dInputAnalogMax     1024
#define dInputAnalogMin     0

#define dInputDigitalLow    0
#define dInputDigitalHigh   1

class cInput
{
protected:
    int     nType;

    bool    nActive;

    bool    nInverted; //Used only with digital input

    unsigned int nValue;
    unsigned int nPrevValue;

    unsigned int nId;

    char nName[dDefaultStringLenght];

public:
    cInput();
    ~cInput();

    cInput operator = (cInput&);

    unsigned int GetId();

    void    Clear();

    void    SetActive(bool Active);
    bool    GetActive();

    void    SetType(int Type);
    int     GetType();

    void    SetValue(int Value);
    int     GetValue();
    int     GetPrevValue();

    void    SetInverted(bool Invert);
    bool    GetInverted();

    void    SetName(char *Name);
    char    *GetName();

    void    Update();
};

#endif
```

Liite 4. Järjestelmän lähtöjen rakennetta kuvaava otsikkotiedosto cOutput.h

```
#ifndef cOutput_h_
#define cOutput_h_

#include "Defines.h"

#define dOutputTypeDigital  0
#define dOutputTypeAnalog   1

#define dOutputAnalogMin    0
#define dOutputAnalogMax    1024

#define dOutputDigitalLow   0
#define dOutputDigitalHigh  1

int CheckOutputValueRange(int Type, int Value);

class cOutput
{
protected:
    int     nType;

    bool    nActive;

    bool    nForced;

    unsigned int nValue;
    unsigned int nPrevValue;
    unsigned int nForcedValue;

    unsigned int nId;

    char nName[dDefaultStringLenght];

public:
    cOutput();
    ~cOutput();

    cOutput operator = (cOutput&);

    unsigned int GetId();

    void    Clear();

    void    SetActive(bool Active);
    bool    GetActive();

    void    SetType(int Type);
    int     GetType();

    void    SetValue(int Value);
    int     GetValue();
    int     GetPrevValue();

    void    SetForecedValue(int Value);
    int     GetForecedValue();

    int     GetCurrentValue();

    void    SetForced(bool Forced);
    bool    GetForced();

    void    SetName(char *Name);
    char    *GetName();

    void    Update();
};

#endif
```