OULUN AMMATTIKORKEAKOULU

Ari Korvala

I/O-MODUULIN KORVAUS OHJELMOITAVALLA LOGIIKALLA

I/O-MODUULIN KORVAUS OHJELMOITAVALLA LOGIIKALLA

Ari Korvala
Opinnäytetyö
Syksy 2016
Automaatiotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Automaatiotekniikan koulutusohjelma

Tekijä(t): Ari Korvala
Opinnäytetyön nimi: I/O-moduulin korvaus ohjelmoitavalla logiikalla
Työn ohjaaja(t): Tero Hietanen
Työn valmistumislukukausi ja -vuosi: Syksy 2016 Sivumäärä: 34 + 2 liitettä

Työn aiheena oli I/O-moduulin korvaaminen ohjelmoitavalla logiikalla. Työn tavoitteena oli valita järjestelmään sopiva logiikka, ohjelmoida siihen valopalkkien kirkkaudensäätö sekä muut järjestelmän vaatimat toiminnot, suunnitella RS-485-muunnin signaalinmuunnosta varten sekä integroida laitteisto Rollmark-järjestelmään.

Logiikan valinta tehtiin siltä vaadittavan nopeuden perusteella. Päädyttiin tulokseen että Siemens S7-1200 -sarjan logiikka riittäisi tarkoitukseen. Logiikan ohjelma koostui pääohjelmasta sekä useasta toimintolohkosta. Pääohjelma ohjelmoitiin Ladder Logic -ohjelmointikielellä, kun taas toimintolohkot kirjoitettiin SCL-kielellä. Valopalkkien kirkkaudensäätö sisältyi toimintolohkoihin tehtyihin ohjelmiin. RS-485-muunnin suunniteltiin kolmessa osassa. Ensin laskettiin jännitteenjako 24 voltista 5 volttiin, jota RS-485-signaali käyttää. Toisena suunniteltiin prototyyppi, jonka avulla laitteen toimivuus testattiin käytännössä. Viimeisenä suunniteltiin lopullinen piirilevy, johon RS-485-muunnin laitetaan.

Laitteisto integroitiin Rollmark-järjestelmään sopimalla logiikan sekä Linux PC:n välisen kommunikoinnin menetelmistä. Tiedonsiirron käyttämät viestityypit ja viestien rakenteet dokumentoitiin yritykseen. I/O-moduulin korvaaminen saatiin suoritettua onnistuneesti loppuun asti. Ohjelmoitava logiikka tarjoaa joustavuutta, jota I/O-moduulilta puuttuu. Logiikka taipuu asiakkaan toiveen mukaisesti toiminnallisesti.

Asiasanat: ohjelmoitava logiikka, RS-485, SCL, tiedonsiirto

ABSTRACT

Oulu University of Applied Sciences
Automation technology

Author(s): Ari Korvala
Title of thesis: Replacing I/O-module with a programmable logic controller
Supervisor(s): Tero Hietanen
Term and year when the thesis was submitted: Fall 2016 Pages: 34 + 2 appendices

---

The subject of the work was replacing I/O-module with a programmable logic controller. The goal of the work was to choose a suitable logic controller for the system, program the light beam intensity control and rest of the systems required functions on it, plan a RS-485-converter for signal conversion and to integrate the hardware into the Rollmark-system.

The logic controller selection was made on the basis of the required hardware speed. It was decided that the Siemens S7-1200 -series would suffice for the purpose. The logic controllers program consisted of the main program and several function blocks. The main program was made with ladder logic -code, while the function blocks were written SCL-code language. The light beam intensity control was included in the function blocks. The RS-485-converter was planned in three stages. The first stage was using formulas to create a voltage divider, which drops the voltage from 24 volts to five volts, which the converter uses. The second stage was the planning of the prototype, which was later used to test the functionality of the system. Last stage was the planning of the final version of the RS-485 converter, which was going to a printed circuit board.

The hardware was integrated into the Rollmark-system. Message types and message structures, used in the communication between the logic controller and Linux computer were documented in the company. Replacing the I/O-module was successful. The programmable logic offers more flexibility, which the I/O-module doesn’t. The logic controller bends more easily according to customers wishes

---

Keywords: programmable logic controller, RS-485, SCL, communication

ALKULAUSE

Haluaisin kiittää työni ohjaajaa Tero Hietasta työn käynnistymisen auttamisesta sekä loppuun viemisestä. Haluaisin myös kiittää koko SR-Instrumentsin väkeä tuesta ja avusta työn kanssa. Erityisesti haluan kiittää Ari Poutasta logiikan ja Linux PC:n välisen kommunikoinnin kehityksen avustamisesta.

Oulussa 6.10.2016

Ari Korvala

SISÄLLYS

TIIVISTELMÄ 3
ABSTRACT 4
ALKULAUSE 5
SISÄLLYS 6
1 JOHDANTO 8
2 SR-INSTRUMENTS OY 9
2.1 Rollmark-järjestelmä 9
2.2 I/O-moduuli 11
3 OHJELMOITAVA LOGIIKKA 13
3.1 TIA-Portal 13
3.2 Ohjelmointikielet 14
3.2.1 Ladder Logic -kieli 14
3.2.2 Function Block Diagram -kieli 15
3.2.3 Structured Control Language 16
3.3 Logiikan valinta 16
3.4 Logiikan ohjelmointi 16
3.4.1 Ohjelman rakenne 17
3.4.2 TCON, TSEND ja TRCV 18
3.4.3 MC_MotionControl 19
3.4.4 Valopalkkien ohjaus 20
3.4.5 Valopalkkien ohjauksen toimintolohkot 21
4 INTEGROINTI JÄRJESTELMÄÄN 26
4.1 Kommunikaatio Linux PC:n kanssa 26
4.2 Viestityypit ja viestien rakenne 26
4.3 Liitos järjestelmään 28
4.4 RS-485-muunnin 29
4.5 Logiikan muut toiminnot järjestelmässä 30
5 YHTEENVETO 32
LÄHTEET 34

LIITTEET

Liite 1. Kytkentäkaavio RS-485-muunnin

Liite 2. Communications specification for Linux PC - Logic Controller

1 JOHDANTO

Työssä korvattiin SR-Instrumentsin valmistaman, valssaamovirheitä havaitsevan, Rollmark-järjestelmän I/O-moduuli ohjelmoitavalla logiikalla. I/O-moduulin tehtävä järjestelmässä oli ohjelmoida järjestelmän valopalkkien kirkkaudet. Työn tavoitteena oli valita sopiva logiikka järjestelmään, ohjelmoida siihen aikakaavion mukaiset sekä muut järjestelmän toiminnot, luoda tiedonsiirto logiikan ja Linux-tietokoneen välille sekä integroida logiikka lopulliseen järjestelmään.

Valopalkkien ohjaus Rollmark-järjestelmässä tapahtui RS-485-signaaleilla. Logiikan ollessa kykenemätön RS-485-tiedonsiirtoon jouduttiin suunnittelemaan piirilevy, joka muuntaa logiikalta saadun bittitiedon RS-485-signaaliksi.

Suurin ongelma työn aikana oli tiedonsiirto logiikan ja Linux tietokoneen välillä. Lähetetty ja vastaanotettu data jouduttiin muuttamaan logiikalle sopivaksi logiikan tiedonlukujärjestyksen ja datatyyppien vuoksi.

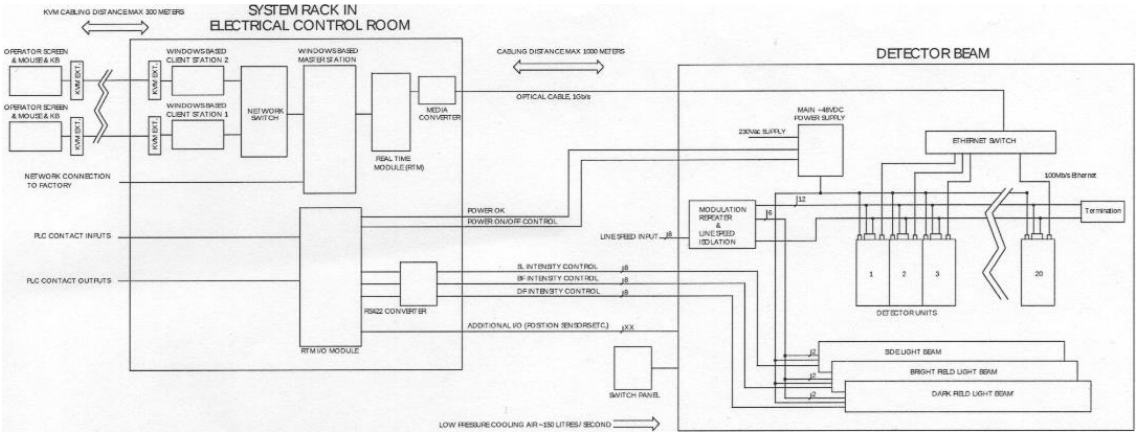
2 SR-INSTRUMENTS OY

SR-Instruments Oy on vuonna 1995 perustettu terästuotannon linjoille valmistusvirheitä havaitsevien laitteiden suunnittelu ja valmistus yritys. Muita samanlaisia yrityksiä on maailmalla, mutta yrityksen itse kehittämä teknologia tekee siitä ainutlaatuisen. Yritys on tehnyt yhteistyötä maailmanlaajuisesti eri yrityksien kanssa kehittäessään laitteita ja teknologiaa.

SR-Instruments Oy valmistaa tällä hetkellä kahta erityyppistä laitetta: MPH-sarja sekä uusi Rollmark-laitteisto. Tässä työssä korvattiin Rollmark-järjestelmästä I/O-moduuli ohjelmoitavalla logiikalla. Ohjelmoitava logiikka antaa järjestelmälle joustavuutta asiakkaiden toiveiden täyttämiseksi.

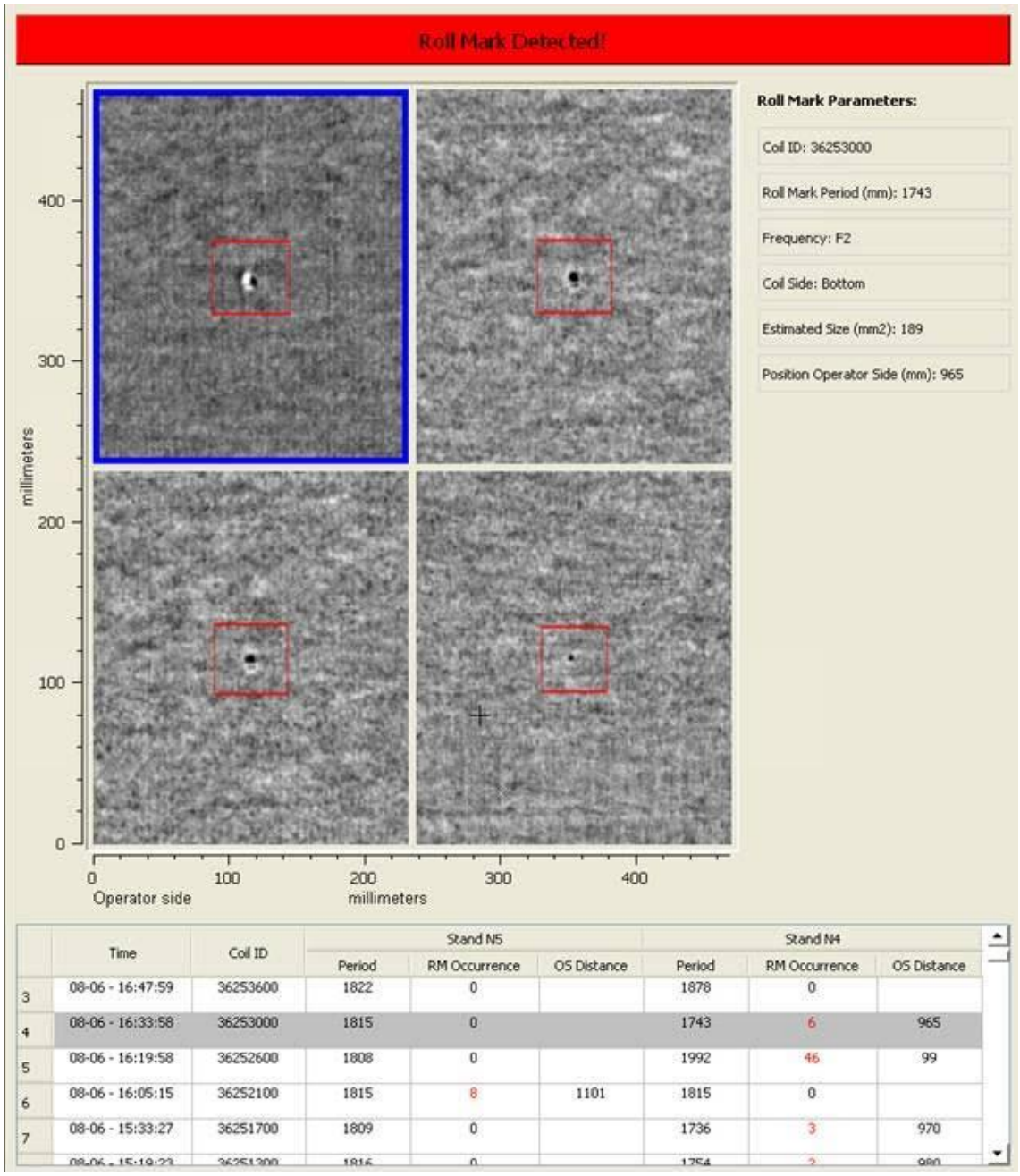
2.1 Rollmark-järjestelmä

Rollmark-järjestelmän tehtävä on havaita erilaisia vikoja kuten valssimerkkejä teräksen valmistuslinjalla. Järjestelmä koostuu detektoripalkista sekä järjestelmätelineestä. Detektoripalkissa on kolme valopalkkia, brightfield, darkfield sekä sidelight, valopalkkien LED-ohjaimet, korkeintaan 20 detektoria, toistinkortti sekä Ethernet-kytkin. Järjestelmä telineessä on isäntätietokone, Linux PC myöhemmin, jolla ohjataan järjestelmää, RTM- eli Real Time Module -moduuli, RS-485-muunnin sekä I/O-moduuli (kuva 1). Järjestelmään kytketään ainakin yksi ulkopuolinen Windows-tietokone, jolla voidaan valvoa ja kauko-ohjata järjestelmää. Laitteiston ohjelmistoa voidaan myös päivittää tätä kautta.



KUVA 1. Rollmark-järjestelmä (1.)

I/O-moduuli ohjaa LED-ohjaimia, jotka säätävät valopalkkien kirkkauden. Ohjausarvot järjestelmä saa Linux-tietokoneelta. Laite asennetaan teräksen valmistuslinjalle, jossa se tutkii teräksen pintaa valon heijastuksia käyttäen. Operaattori näkee pienimmätkin virheet tietokoneelta, joka on valvomossa. Havaitut virheet ja poikkeamat teräksen pinnassa tallentuvat listaan (kuva 2), josta niitä voidaan tarkastella. Käyttöliittymästä näkee myös teräskelan tunnuksen, millä valmistuslinjalla se tapahtui, poikkeamien tapahtumien taajuuden, poikkeamien määrän sekä etäisyyden käyttäjän puolelta katsottuna.



KUVA 2. Käyttöliittymän lista poikkeamista teräksessä

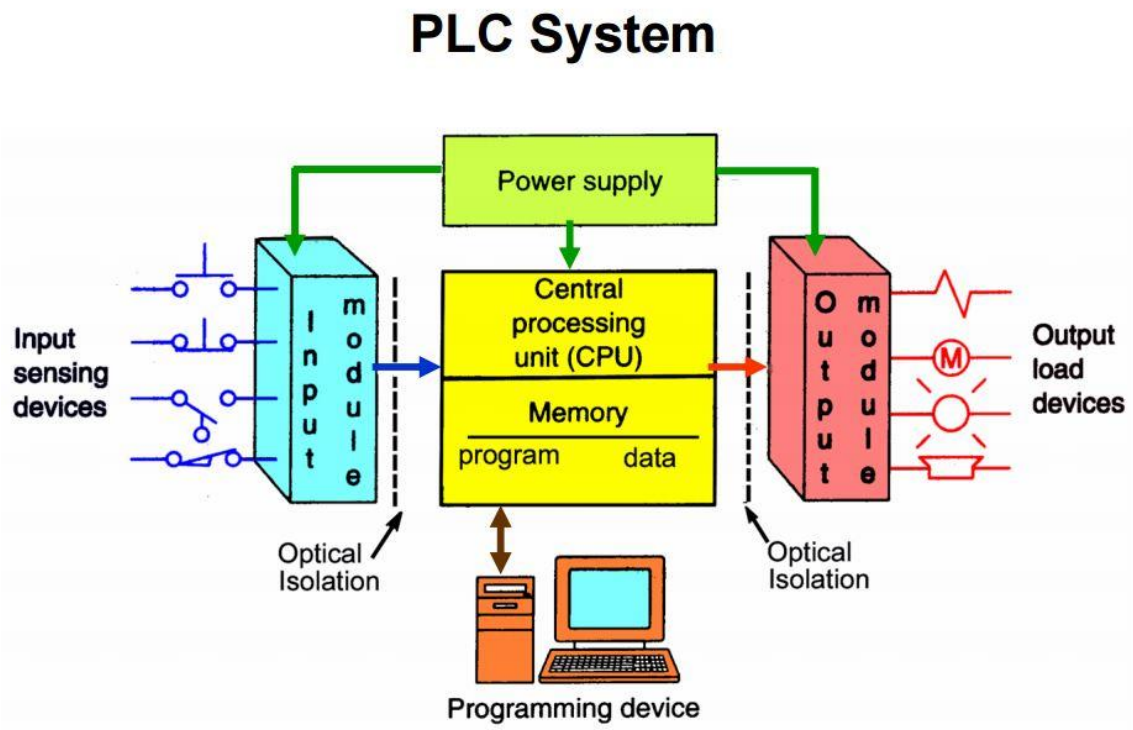
2.2 I/O-moduuli

I/O-moduuli oli alkuperäisessä järjestelmässä Linux-tietokoneessa PCI-väylässä. Tietokoneelle kirjoitetun ohjelman avulla sillä lähetettiin bittisignaaleja laitteistolle lattakaapelia pitkin. Signaali muunnetaan RS-485-muotoon RS-485-muuntimella, josta se viedään toistinkortille. Toistinkortilla signaali muunnetaan bitti-signaaliksi ja takaisin RS-485-signaaliksi. Toistinkortilta signaalit viedään jokaiseen valopalkin LED-ohjaimille, joista signaalit viedään seuraavalle LED-

ohjaimelle. Laitteistoon ei pääse käsiksi tehtaalla sen ollessa toiminnassa, joten tietokoneen I/O-moduulin sekä toistinkortin välissä olevassa RS-485-muuntimessa oli LED-valoja diagnostiikkaa varten.

3 OHJELMOITAVA LOGIIKKA

Ohjelmoitava logiikka on teollisuudessa käytetty puolijohde tietokone. Se valvoo logiikan tuloja ja lähtöjä sekä tekee loogisia päätöksiä automatisoiduille prosesseille ja laitteille. Ohjelmoitavat logiikat esitettiin ensimmäistä kertaa maailmalle 1960-luvun loppupuolella. Niiden tarkoitus oli korvata rele-logiikka-järjestelmät. Ohjelmoitavat logiikat koostuvat prosessorista, muistista sekä I/O-moduulista (kuva 3). Logiikat lukevat analogisien tai digitaalisien signaalien tiloja erilaisilta antureilta ja kytkimiltä. Lähdöillä voidaan ohjata erilaisia laitteita, kuten moottoreita tai venttileitä. (2.)



KUVA 3. Ohjelmoitava logiikka kokonaisuutena (2.)

3.1 TIA-Portal

Ohjelmoitavat logiikat vaativat niiden ohjelmoimiseen ohjelmointilaitteen. Ohjelmointilaite on yleensä tietokone, jolle on asennettu logiikan valmistajan ohjelmointityökalu, mutta joissakin tapauksissa ohjelmoitavat logiikat voidaan ohjelmoida suoraan logiikassa olevasta näytöstä. Tätä tapaa voidaan käyttää

pienempien ohjelmien tekemiseen, mutta sitä ei suositella suurempien ohjelmien luomiseen.

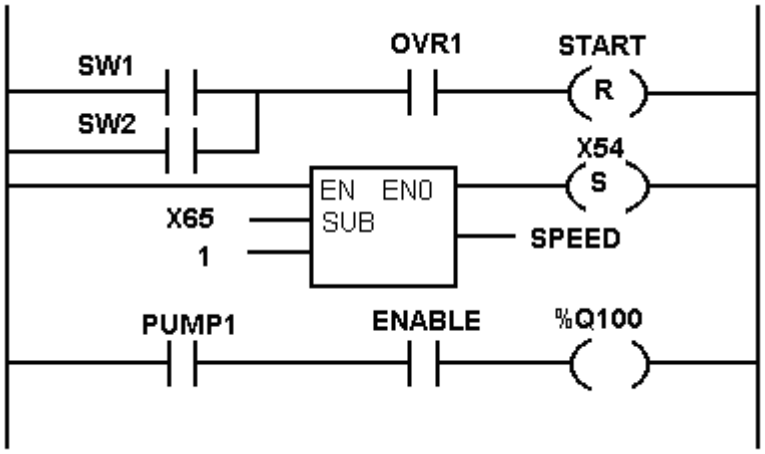
Työssä käytetty Siemensin logiikka ohjelmoitiin TIA Portal -ohjelmointityökalulla. TIA Portal on monipuolinen logiikoita varten valmistettu Siemensin oma projektinluontityökalu. Sitä päivitetään jatkuvasti asiakkaiden palautteiden ja ongelmien mukaisesti. Projektiin voidaan valita useampia logiikoita sekä lisätä moduuleja tarpeen mukaan. Ohjelma kirjoitetaan lohkoihin Ladder-, Function Block Diagram- tai Structured Control Language -ohjelmointikielellä, joista logiikka suorittaa toiminnot lohkojen numerointien tai lohkokutsujen perusteella ylhäältä alas, vasemmalta oikealle. Jokainen logiikan ohjelma vaatii OB1-lohkon. Tämä lohko on projektin pääohjelma.

3.2 Ohjelmointikielet

Ohjelmoitavia logiikoita voidaan ohjelmoida eri kielillä. Näitä kieliä ovat mm. Ladder Logic, Function Block ja Siemensin logiikoissa käytetty Structured Control Language. Logiikoita valmistetaan myös ohjelmoitaviksi C-kielillä ja jopa Linux-pohjaisilla kielillä.

3.2.1 Ladder Logic -kieli

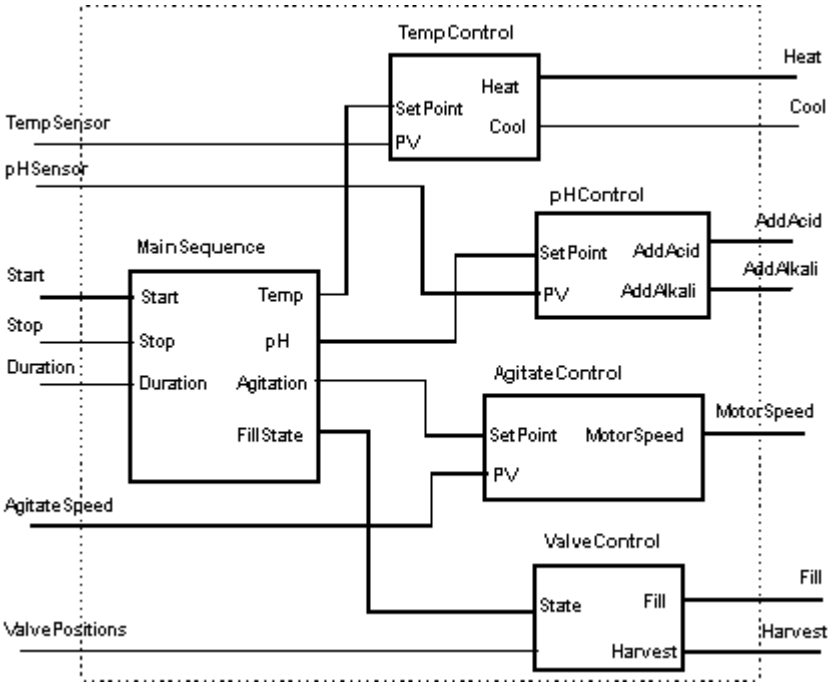
Ladder Logic -kieli oli alun perin tapa dokumentoida releтoimintojen suunnittelua ja rakennetta. Myöhemmin siitä kehitettiin standardisoitu ohjelmointikieli logiikoiden ohjelmointia varten. Kieli otettiin aluksi käyttöön sen yksinkertaisuuden takia. Sen käyttö ja opettelu oli helppoa ja teki tehtaiden henkilöstölle mahdollista luoda ohjelmia ilman ylimääräisiä koulutuksia ohjelmointikieliä varten. Kehitystyöt ja huoltotehtävät yksinkertaistuivat, koska Ladder Logic -kieli muistutti relekaavioita (kuva 4). Ladder logic -kieli on käytännöllisin yksinkertaisiin ohjausjärjestelmiin, mutta sitä voidaan myös käyttää monipuolisten ja vaativien ohjelmien luomiseen. (3.)



KUVA 4. Ladder logic -ohjelmointi (4.)

3.2.2 Function Block Diagram -kieli

Function block diagram eli FBD on toinen graafinen ohjelmointikieli logiikoille. Sillä kuvataan tulojen ja lähtöjen muuttujien toimintoja toimintalohkoja käyttäen. Toimintalohkot ovat joko valmiina ohjelmointityökaluun suunniteltu tai ne ovat ohjelman laatijan itse tekemiä. FBD muistuttaa ladder logic -kieltä, mutta se on enemmän hajautetun oloista (kuva 5), eikä muodosta ladder-kielen tunnettua tikapuukaaviota. (5.)



KUVA 5. Function block diagram (5.)

3.2.3 Structured Control Language

Structured control language, lyhyesti SCL, on korkean tason PASCAL-ohjelmointikielestä johdettu tekstipohjainen ohjelmointikieli. Korkean tason kielielementtien lisäksi SCL sisältää myös tyypilliset logiikoiden toiminnot kuten tulot, lähdöt, ajastimet, muistibitit ym. Toisin sanoen SCL täydentää ja laajentaa ohjelmointiohjelmia ja niiden ladder logic- sekä function block diagram –kieliä. (6, s. 18)

Projektiin tehty pääohjelma kirjoitettiin Ladder Logic -kielellä, koska se oli helpoin ratkaisu siihen tarkoitukseen. Pääohjelman käyttämät toimintolohkot kirjoitettiin SCL-kielellä. SCL-kielen käyttö kuului työn rakenteeseen.

3.3 Logiikan valinta

Projektin logiikka valittiin prosessin vaatimusten perusteilla. Logiikalta vaadittiin riittävä suoritusnopeus, lähtöjä riittävästi sekä mahdollisuus avoimeen TCP/IP-kommunikaatioon. Näiden vaatimusten perusteilla valittiin logiikaksi Siemens S7-1200 CPU 1214C DC/DC/DC. Lähtöjä logiikassa ei ollut tarpeeksi, joten projektiin liitettiin myös kahdeksanlähtöinen digital output –moduuli. Logiikalta olisi myös vaadittu RS-485-signaalilähtöjä, mutta tutkimusten perusteella kyseisillä signaalityypeillä varustettuja ohjelmoitavia logiikoita ei ole. Siemens valmistaa kommunikaatiomoduuleita, joissa on RS-485 -mahdollisuus, mutta moduuleita olisi projektiin tarvinut vähintään kuusi kappaletta. Tämä olisi nostanut projektin hintaa sekä logiikkakokonaisuuden kokoa. Päädyttiin tulokseen, että logiikan jälkeiseen johdotukseen suunnitellaan ja rakennetaan RS-485 -muunnin komponenteista.

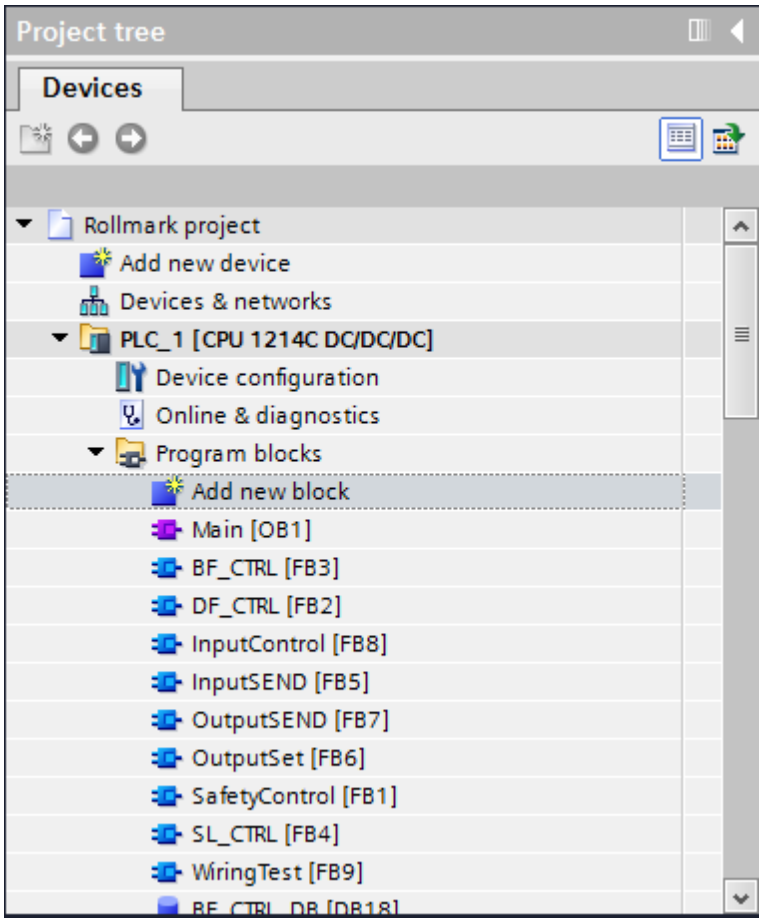
3.4 Logiikan ohjelmointi

Logiikka ohjelmoitiin projektiin vaiheittain. Pääohjelma OB1 ohjelmoitiin Ladder-kielellä. Heti alkuun luotiin ensimmäinen versio valopalkkien ohjaustoiminnoista. Ohjelman toimintoja ja toimintatapoja testatessa ohjelmaa jouduttiin muuttamaan useaan otteeseen toisenlaiseksi. Kun valopalkkien ohjaus saatiin ohjelmoitua toimivaksi, luotiin ohjelmaan muut järjestelmän toiminnot.

Kommunikaatio Linux PC:n kanssa tehtiin pääohjelmaan sen vähäisen rakenteellisen koon vuoksi.

3.4.1 Ohjelman rakenne

Ohjelma sisältää pääohjelman sekä yhdeksän toimintolohkoa (kuva 6), joita pääohjelma suorittaa käskyn saadessaan. Ohjelmaan tehty toimintolohko ”SafetyControl” ei ole käytössä toistaiseksi. Sen tarkoituksena oli saada Linux PC:ltä parametrit, joilla se valvoisi tulojen tilaa, ja tällä ohjaisi lähtöjen muutoksia tehtaan alueella. Se kuitenkin otettiin pois käytöstä sen tarpeen puutteen vuoksi. Tilalle tehtiin ”InputControl”-toimintolohko, joka valvoo kiinteitä tuloja ja ohjaa lähtöjä niiden tilojen perusteella. Toimintolohko ohjelmoidaan asiakkaan tarpeiden mukaan, mutta voidaan jättää tyhjäksi, jos se koetaan tarpeettomaksi.

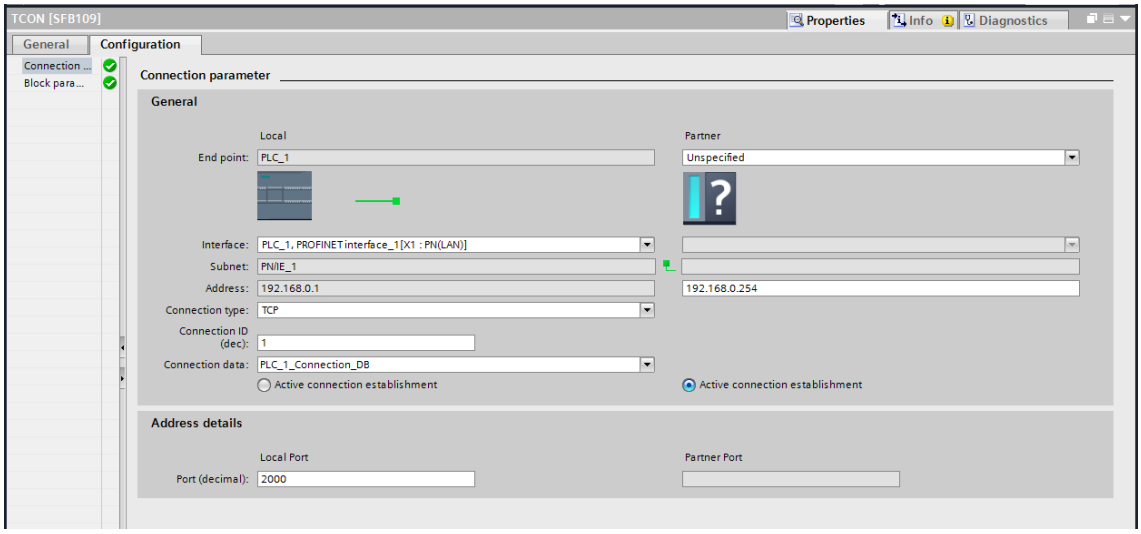


KUVA 6. TIA-Portal-hierarkia projektissa

Pääohjelmaan tehtiin kommunikaatio-ohjaus Linux PC:n kanssa, valopalkkien toimintolohkojen ohjaussekvenssi, yllä mainittu ”InputControl” ja ”SafetyControl”, ”Pulse Train Output”-ohjaukset sekä ohjelmat tiedon lähettämiselle Linux PC:lle. Toimintolohkot nimettiin ohjelmaan niiden toimintojen perusteella.

3.4.2 TCON, TSEND ja TRCV

TCON, TSEND sekä TRCV ovat Siemensin logiikoissa käytettäviä avoimen kommunikoinnin toimintalohkoja. Logiikka voi kommunikoida minkä tahansa laitteen kanssa, jossa on Ethernet-portti, näiden toimintojen avulla. TCON-toimintalohko muodostaa yhteyden kommunikointikumppanin kanssa saadessaan käskyn. Toimintalohkoon määritellään yhteyden tunnus, yhteydessä käytetty protokolla, käytetyt TCP/UDP-portit, kumppanin IP-osoite sekä yhteyden muodostava laite (kuva 7). TCON-lohko tukee TCP-, UDP- sekä TCP-to-ISO-protokollia tiedonsiirrossa. TCON-toimintalohkoilla voidaan muodostaa useampi yhteys eri laitteiden välillä. Tuolloin järjestelmä vaatii reitittimen laitteiden väliin, koska logiikassa on vain yksi Ethernet-portti. (7.)



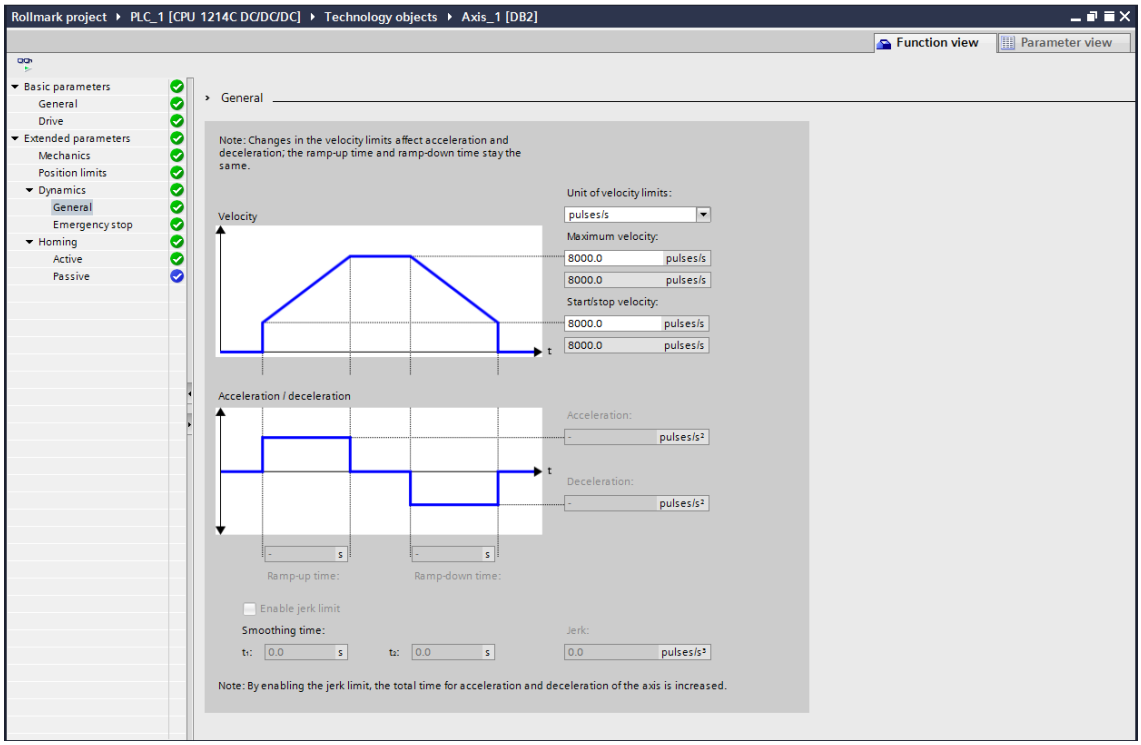
KUVA 7. TCON toimintalohkon yhteysparametrit

TSEND-toimilohkot vastaa tiedon lähettämisestä vastaanottavalle laitteelle. Lähetetty tietotyyppi voi olla mitä tahansa logiikan sisäisessä ohjelmoinnissa. TSEND-toimilohkoihin määritellään yhteystunnus sekä lähetetty tieto. Tiedon lähettämiseen yhdelle vastaanottavalle laitteelle voidaan käyttää useampaa TSEND-toimilohkoa.

TRCV-toimilohko vastaanottaa kommunikaatiokumppanin lähettämää dataa. TSEND-lohkosta poiketen, TRCV-lohkoja tarvitaan vain yksi kappale yhteystunnusta kohden, eikä periaatteessa useamman lohkon käyttäminen ole suositeltua. Useamman TRCV-lohkon käyttäminen samalle yhteydelle vie vain muistitilaa logiikasta. Vastaanotettu data tallettuu määritettyyn muotoon, josta logiikan ohjelma voi sitä lukea käskettäessä.

3.4.3 MC_MotionControl

MC_MotionControl-toimilohko on Siemensin logiikoissa käytetty servomoottoreille tarkoitettu ohjainlohko. Toiminto on monipuolinen ja tarkka tapa suorittaa servomoottoreiden ohjauksia logiikoilla. Toiminnalle voidaan säätää moottorin kiihtyvyys- ja hidastusvauhdit, aloituspiste käytännön sijainnilla, raja-arvoja, vähittäis- ja enimmäispyörimisnopeudet sekä yhden moottorinkierroksen tuottama liikkumisetäisyys (kuva 8). Servomoottoreita ohjataan muilla toimintolohkoilla. Aloituspisteelle ajoon on oma toimintolohkonsa, kuten myös muille toiminnoille.



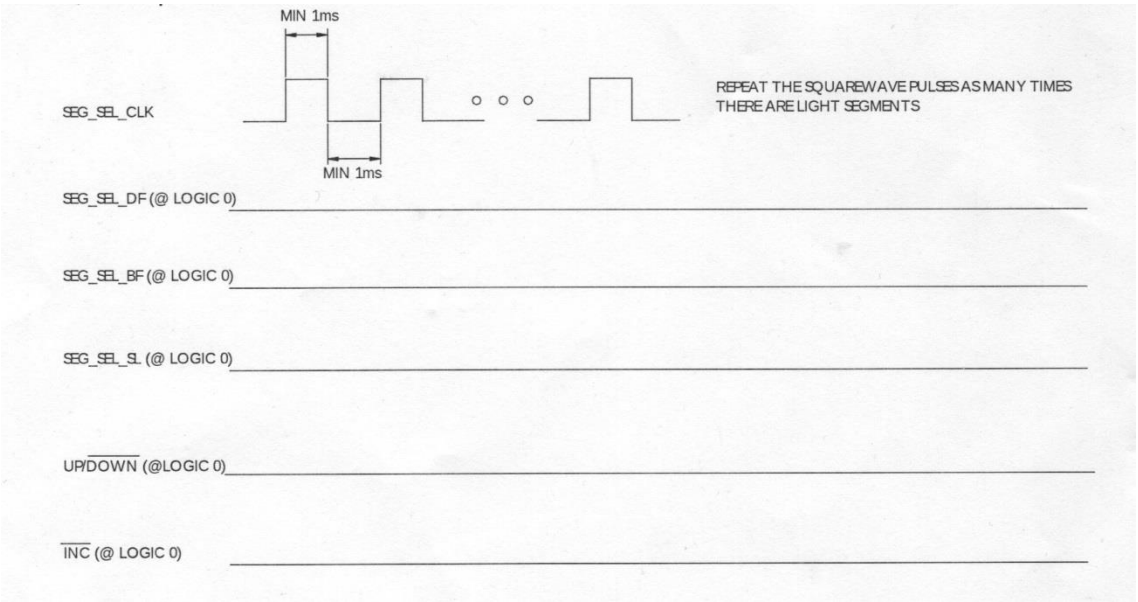
KUVA 8. MC_MotionControl asetuksia

Servomoottorit toimivat pulssisignaaleilla. Tätä ominaisuutta käytettiin työssä hyväksi. Valopalkkien ohjauspulssit sekä resetointipulssit lähetettiin servo-ohjauksina logiikalta. MC_MotionControl-toimilohkon asetukset ohjelmoitiin projektiin minimaalisesti. Ylimääräisiä toimintoja ei käytetä ja ohjauksille asetettiin kiinteä pulssitaajuus.

MC_MotionControl-toimilohkon lähtöjen tilaa ei voi muuttaa muulla menetelmällä kuin sen ohjaustoimintolohkoilla. Tämän vuoksi järjestelmään jouduttiin ottamaan kaksi lähtöä käyttöön, joilla luotiin invertoitu tila INC-signaalille (kuva 10) sekä annettiin yksittäiset pulssit SEG_SEL_CLK-signaalille. Kaksi lähtöparia liitettiin yhteen logiikan jälkeiseen RS-485-muuntimeen, jotta järjestelmä saatiin toimimaan.

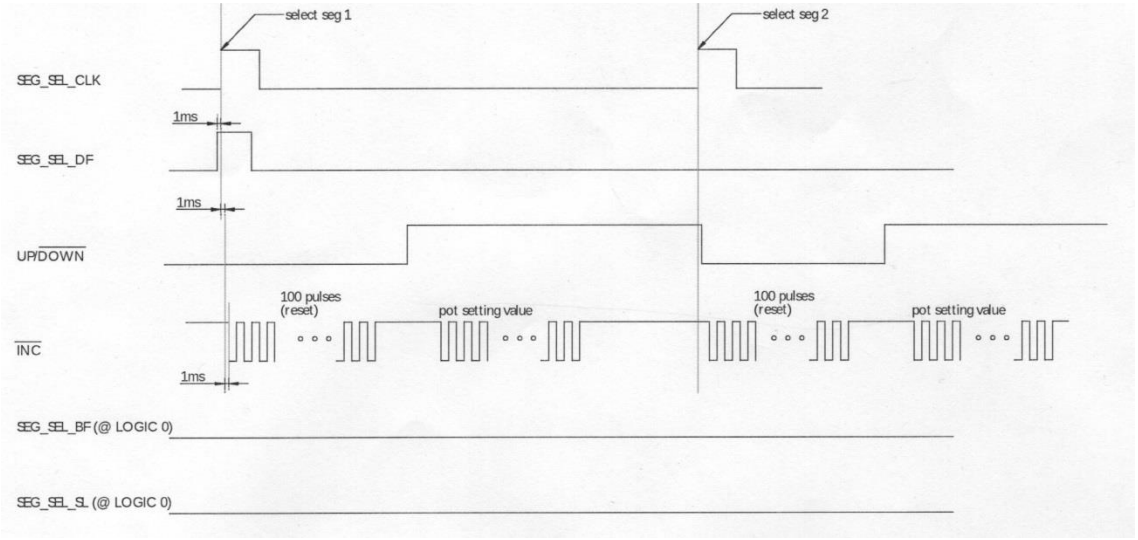
3.4.4 Valopalkkien ohjaus

Rollmark-järjestelmän valopalkit ohjataan pulssisignaaleilla. Logiikkaan luotu ohjelma lähettää kolmelle valopalkille ohjausarvonsa pulssikaavion mukaisesti. Ensin ohjattavien valopalkkien segmenttien ”sijainti” ohjattavassa järjestelmässä palautetaan oletusarvoonsa. Tämä tapahtuu lähettämällä niin monta pulssia valopalkille, kuin valosegmenttejä on järjestelmässä (kuva 9). Tämä tehdään aina ennen valopalkkien kirkkauden säätöä. (1, s. 15.)



KUVA 9. Resetointi sekvenssi (1, s. 15.)

Valopalkkien kirkkauden säätö ja resetointi tehtiin MC_MoveRelative-toimintolohkoilla. Pulssikaaviossa (kuva 9) näkyvällä INC-signaalilla ohjattiin valojen kirkkauksia. Jokaiselle segmentille annettiin aina ensin 100 pulssia UP/DOWN-signaalin ollessa nollatilassa. Tämän jälkeen Linux-tietokoneelta saadun viestin mukaisesti annettiin pulsseja UP/DOWN-signaalin ollessa ykköstilassa. Ohjattava valopalkki valitaan SEG_SEL_DF-, SEG_SEL_BF- tai SEG_SEL_SL-signaalilla. SEG_SEL_CLK-signaalilla vaihdetaan järjestelmässä ohjelmoitavan valopalkin segmenttiä. (1, s. 16.)



KUVA 10. Valopalkkien säädön pulssidiagrammi (1, s. 16.)

3.4.5 Valopalkkien ohjauksen toimintolohkot

Valopalkkien ohjauksia varten luotiin jokaiselle valopalkille oma toimintolohko. Kaikkien valopalkkien säädön tapahtuessa jokaisella ohjelmointikerralla, tehtiin toimintalohkojen suoriutumisesta toisistaan riippuvaisia. Darkfield-valopalkin ohjelmoinnin loppuminen käynnistää brightfield-valopalkin ohjauksen ja brightfield-valopalkin ohjelmoinnin loppuminen vuorostaan käynnistää sidelight-valopalkin ohjauksen. Toimintalohkot ovat melko samanlaisia toisiinsa nähden. Pieniä eroja syntyi valopalkkien ohjelmoinnin aloitukseen darkfield-toimintalohkoon sekä lopetukseen sidelight-toimintalohkoon.

Toimintalohkojen ohjelmiin tehtiin kolme osiota. Ohjelma tunnistaa osiot #CaseIndex-nimisen muuttujan avulla. Osion valinta tehtiin perustumaan Linux

PC:ltä vastaanotettuun dataan. Ohjelma lukee listan arvoa i ja suorittaa ohjelman toista tai viimeistä osiota sen perusteella. Ohjelman ensimmäinen osio suoritetaan aina ensin kutakin toimintalohkoa suoritettaessa, mutta vain kerran kunkin valopalkin ohjelmoinnissa, sillä ohjelman ensimmäisessä osiossa suoritetaan toiminnot, jotka täytyy suorittaa vain kerran valopalkkien ohjelmoinnin sekvenssin aikana.

#CaseIndex-muuttujan saadessaan arvon ’2’ ohjelma siirtyy sekvenssissä toistuvaan vaiheeseen. Ohjelman toistuva osio toistuu niin monta kertaa, kuin vastaanotetussa datassa määritellään. Logiikka lukee vastaanotettua dataa listasta, jonka arvoa se vertaa. Arvon ollessa alle 100, logiikka tietää että se on valopalkille tarkoitettu kirkkausarvo. Yli 100 olevat luvut logiikka ymmärtää ohjelman toiston lopettamismerkkinä. Tällaisen arvon saadessaan, logiikka siirtyy #CaseIndex arvon ’3’ vaiheeseen, eli viimeistelyyn. #CaseIndex valinnassa on myös mukana muuttujia, jotka toimintolohko saa sen ulkopuolelta. Näillä muuttujilla käynnistetään valopalkkien ohjelmointi (kuva 11).

```
// Selection of sequence

IF #InitDF AND NOT #Muuttuja11 AND NOT #DF_CTRL_DONE THEN // Is executed always once at start of LB setting

    #CaseIndex := INT#1; // First part of sequence

ELSIF #InitDF AND #Muuttuja11 AND #Settings[#i] < REAL#100 AND NOT #DF_CTRL_DONE THEN // If value[#i] of re-
ceived data is < 100, they are LB values

    #CaseIndex := INT#2; // Looping part of sequence

ELSIF #InitDF AND #Muuttuja11 AND #Settings[#i] > REAL#100 AND NOT #DF_CTRL_DONE THEN // If value[#i] of re-
ceived data is > 100, move to next LB (BF)

    #CaseIndex := INT#3; // Finishing part of sequence

END_IF;
```

KUVA 11. #CaseIndex-muuttujan valinta ohjelmassa

Näissä kolmessa toimintolohkossa, kuten muissakin toimintolohkoissa, ohjataan lähtöjen tiloja. Tilat ohjataan 0- tai 1-tilaan komennolla ”Lähtö1 := 1;”. Ohjelma suorittaa tilan muutoksen lähdössä kuitenkin vasta ohjelman suoritettua itsensä loppuun asti. Valopalkkien ohjaukseen tehtyjen toimintolohkojen ohjelman suoritusaika on noin kaksi sekuntia toimintolohkoa kohden riippuen valopalkkien segmenttien määristä. Tämän vuoksi ohjelmaan luotiin useampi kohta, jossa ohjelma keskeyttää toimintolohkon suorittamisen ja muuntaa ohjatut arvot haluttuihin tiloihin. Tätä toimintoa varten käytettiin ”RETURN;”-käskyä (kuva 12).

```
IF "EN_INV" = 0 AND #CaseIndex = 1 THEN
    "EN_INV" := 1;
    RETURN;
END_IF;
```

KUVA 12. Ohjelman suorittamisen keskeytys ohjelmassa

Toimintolohkoissa suoritettavassa toistuvassa sekvenssissä sekä viimeistelyosiossa käytettiin bool- eli 0 tai 1 -muuttujia paljon. Muuttujat nimettiin ohjelmaan kuvaavasti #PhaseX-nimisiksi kuvaamaan sekvenssin eri vaiheita. Muuttujilla oli myös ohjelmassa tärkeä osa ohjelman suorituksen kannalta. #PhaseX-muuttujan arvoa verrataan melkein jokaisessa sekvenssin osassa ennen sen suorittamista. Ohjelmassa sekvenssin kaikki osat varmistavat, että #PhaseX-muuttujan arvo on vastaavassa tilassa sekvenssin vaiheeseen nähden (kuva 13). Tämä myös mahdollistaa RETURN-käskyn käytön ohjelmassa logiikan lähtöjen ohjausta varten.

```
IF #CaseIndex = 2 AND #Phase1 = 0 AND #Phase2 = 0 AND #Phase3 = 0 AND #Phase4 = 0 AND #Phase5 = 0 AND #Phase6
= 0 AND #Phase7 = 0 AND #Phase8 = 0 THEN
    "UP/DOWN" := 0;
    #Phase1 := 1;
    RETURN;
END_IF;
```

KUVA 13. Toistuvan sekvenssin ensimmäinen vaihe

Toimintolohkoihin tehtiin myös sekvenssin mukainen INC-pulssien lähettäminen. Pulssit tehtiin MC_MoveRelative-toiminnoilla ja jokaiseen toimintolohkoon luotiin kaksi näitä toimintoja, toisella nollataan kirkkaus ja toisella säädetään kirkkaus haluttuun tasoon. Kirkkauden nollaus tapahtuu aina samoilla toiminnon asetuksilla. 100 pulssia lähetetään valopalkin segmentille UP/DOWN-signaalin ollessa 0-tilassa. Kirkkauden säätö tapahtuu UP/DOWN-signaalin ollessa 1-tilassa, mutta pulssien määrä vaihtelee. MC_MoveRelative-toiminnon lähettämä pulssimäärä määritellään sen ”Distance”-parametriin. Ohjelmaan toiminto määriteltiin lukemaan pulssimäärä #Settings[#i]-listasta. #Settings[] sisältää Linux PC:n lähettämän valopalkkien parametrit ja #i on muuttuja ohjelmassa, jolla määritellään mistä kohdasta listaa toiminto lukee arvonsa.

```
WHILE #Muuttuja13 = 0 AND #Settings[#i] > REAL#0 AND #CaseIndex = 2 AND #Phase7 = 1 AND #Phase2 = 0 AND
#Phase3 = 0 AND #Phase4 = 0 AND #Phase5 = 0 AND #Phase6 = 0 AND #Phase1 = 0 AND #Phase8 = 0 DO
    #MC_MoveRelative_Instance_1(Axis := "Axis_1",
                                Execute := #Muuttuja8,              // This MC_MoveRelative_Instance sends the
INC pulses to set the LB's brightness
                                Distance := (#Settings[#i] - 1),    // Amount of INC pulses sent, DO NOT CHANGE
                                Velocity := REAL#8000,              // Frequency of INC. Changing the
frequency must be done to all MC_MoveRelative_Instances AND Technology objects -> Axis_1 -> Configuration ->
General -> Maximum velocity AND Start velocity
                                Done=>#Muuttuja13);
    #Muuttuja8 := 0;
END_WHILE;
```

KUVA 14. MC_MoveRelative kirkkaudensäädössä

INC-pulssien lähettämisen kannalta MC_MoveRelative-toiminto on täydellinen, lukuunottamatta ohjelmaan luotua pakollista ominaisuutta. INC-pulssia ohjataan valopalkeille kahdella lähdöllä. Toinen pulssiohjaus on oletuksena 1-tilassa kokoajan, sillä valopalkkien ohjaukset toimivat laskevalla reunalla. Kuten kuvassa 14 näkyy, ”Distance”-parametri on määritelty (#Settings[#i] – 1). Tämä on juurikin sen vuoksi, että toisen INC-pulssin ohjauksen laskiessaan 0-tilaan, antaa se ohjausvallan MC_MoveRelative-ohjattavalle pulssilähdölle. Tämän tapahtuessa se antaa yhden laskevan reunan INC-pulssin. Tämä toimisi ilman mitään muita ongelmia, ellei valopalkkia joskus ajettaisi nolla-kirkkauteen. Tämä korjattiin lisäämällä ohjelmaan #PhaseX-vertailuun myös #Settings[#i]-vertailu. Kirkkauden nollauksen jälkeen ennen toisen INC-signaalin nollaamista, ohjelma vertailee #Settings[#i]-arvoa. Ohjelma jättää INC-signaalin koskematta, jos se on nolla.

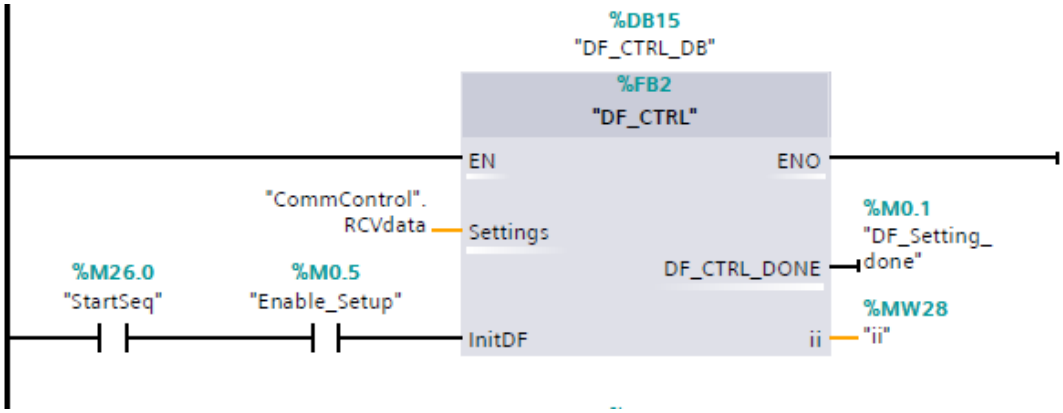
DF- sekä BF-toimintolohkojen sekvenssien viimeistelyvaiheisiin tehtiin yksi toiminto, jota ei tarvittu SL-toimintolohkon lopussa. Kuten aiemmin mainittu, ohjelma lukee valopalkkien kirkkausparametrit #Settings[]-listasta. Tämän mahdollistamiseksi täytyi tieto mistä kohtaa #Settings[]-listaa luettiin saada seuraavalle toimintolohkolle (kuva 15). Arvoon #i lisätään ohjelmassa yksi, ettei seuraava toimintolohko lue arvoa, jota on jo käytetty kertaalleen.

```
IF NOT #DF_CTRL_DONE AND #CaseIndex = 3 THEN
    #ii := #i + + INT#1;    // Sets ii output to #i + 1, BF LB setting starts reading its values at #i + 1
```

KUVA 15. #i-muuttuja lähdölle

Tieto #i-muuttujan arvosta saatiin ulos lisäämällä DF- sekä BF-toimintolohkoihin lähtö, josta ohjelma syöttää ulos numeromuuttujan. Muuttuja tallennetaan

logiikan muistiin ja seuraava toimintolohko lukee ja aloittaa listan käymisen siitä kohdasta (kuva 16).



KUVA 16. DF-valopalkin toimintolohko, kuvassa %MW28 on muistiin tallennettu muuttuja

4 INTEGROINTI JÄRJESTELMÄÄN

4.1 Kommunikaatio Linux PC:n kanssa

Logiikan ja Linux PC:n välinen kommunikaatio tapahtuu binäärimuodossa. Dataa lähetetään neljän tavun rykelmissä, jotka logiikka muuntaa reaaliarvoisiksi luvuiksi. Kommunikaatiossa käytetään TCP/IP-standardia. Linux PC:hen asennettiin ylimääräinen verkkokortti kommunikointia varten. Ylimääräinen verkkokortti ei kommunikoi muiden järjestelmässä olevien laitteiden kanssa. (8, s. 3.)

Logiikka käyttää reaaliarvoisia lukuja suurimpaan osaan ohjaustoiminnoista, koska ”Array of”-datalohkon datatyyppiä ei voi muuttaa ilman logiikan ohjelman muuttamista siihen tarkoitetulla ohjelmistolla ja logiikan vastaanottama data tallentuu kaikki yhteen ”Array of”-datalohkoon. Tämä toteutustapa tehtiin MC_MotionControl-teknologian vuoksi. Toiminto, joka ohjaa INC-pulssia, vaatii reaaliarvoiset numeroarvot sen ohjaukseen. (8, s. 3.)

Koska 1 tavu voi esittää arvoa nollasta 255:een, 32-bittinen parametri täytyy lähettää neljänä tavuna. Logiikka käyttää ”high byte”-arkkitehtuuria. Tässä tapauksessa logiikka lähettää korkeampiarvoiset kahdeksan bittiä ensin, joita seuraa alempiarvoiset kahdeksan bittiä. Linux PC edustaa ”low byte”-arkkitehtuuria, joka lähettää ensin alempiarvoiset kahdeksan bittiä, jonka jälkeen se lähettää korkeampiarvoisemmat kahdeksan bittiä. Tämän arkkitehtuurin käyttö vaatii sen, että tavun paikkoja vaihdetaan. (8, s. 3.)

Esimerkiksi: Linux PC lähettää 32-bittisen dataviestin A1A2A3A4 logiikalle. Linux PC:n täytyy ensin vaihtaa viesti soveltuvaksi ”high byte”-arkkitehtuurille, joten lähetetty viesti olisi A4A3A2A1. Linux PC:n täytyy myös vaihtaa tavujen paikkaa vastaanottaessa dataa logiikalta. (8, s. 3.)

4.2 Viestityypit ja viestien rakenne

Logiikan ja Linux PC:n välillä kulkevat viestit voidaan jakaa kuuteen kategoriaan: valopalkkien ohjausviestit, lähtöjen ohjausviestit, tulojen

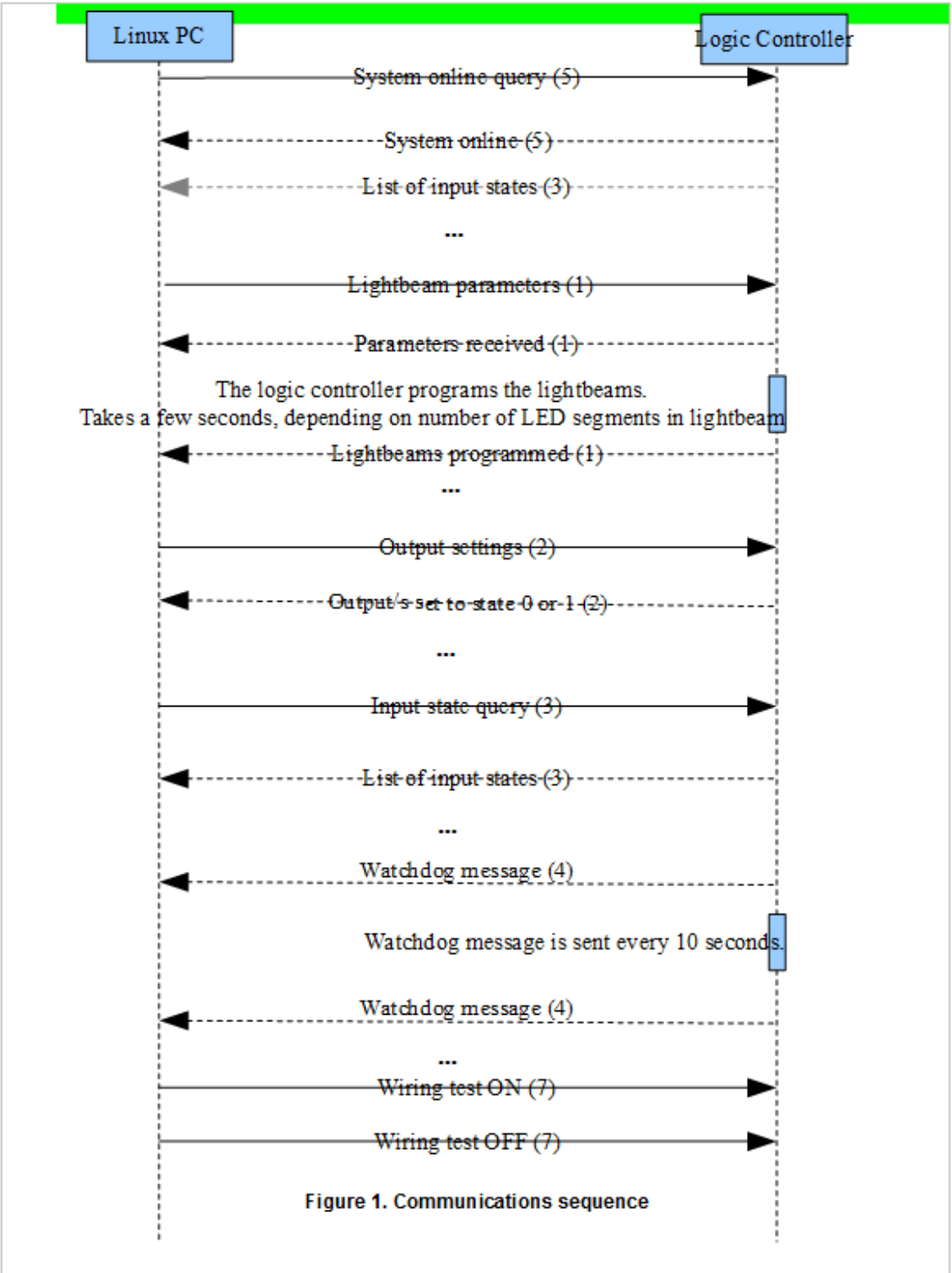
valvontaviestit, yhteyden valvontaviestit, kaapeloinnin testiviesti sekä yhteyden testausviesti. Jokaisella viestityypillä on oma tunnuksensa viestin kolmannessa tavurykelmässä (taulukko 1).

Viestityyppi	**Viestin tunnus**
Valopalkkien ohjausviestit	1
Lähtöjen ohjausviestit	2
Tulojen valvontaviestit	3
yhteyden valvontaviestit	4
Yhteyden testausviesti	5
Kaapeloinnin testiviesti	7

TAULUKKO 1. Viestityyppien tunnukset

Viestityypin tunnus on vasta kolmannessa tavurykelmässä verkon rakenteen vuoksi. Linux PC:n ohjelma lähettää viestin alussa sovellustunnuksen sekä protokolla tunnuksen. Logiikan lähetetty data ei sisällä näitä tavuja, sillä logiikka ei kommunikoi verkossa muiden laitteiden kanssa.

Kommunikointi logiikan ja Linux PC:n välillä toimii sekvenssimäisesti. Järjestelmän käynnistyksessä Linux PC muodostaa yhteyden logiikkaan yhteyden testausviestillä, johon logiikka vastaa viestin tunnuksella sekä lähettää kaikkien tulojen tilan listamuodossa. Tämän jälkeen Linux PC lähettää valopalkkien ohjausparametrit, jonka logiikka kuittaa viestityypin tunnuksella. Logiikka ohjelmoi valopalkit parametrien mukaisesti ja lähettää Linux PC:lle viestin tästä. 10 sekunnin välein yhteyden muodostamisesta logiikka lähettää ”watchdog”-viestiä eli yhteyden valvontaviestiä Linux PC:lle. Tällä varmistetaan että yhteys laitteiden välillä ei ole katkennut. Linux PC voi kysyä logiikalta tulojen tiloja sekä lähettää lähtöjen ohjauskäskyn. Näihin viesteihin logiikka kuittaa viestityypin tunnuksilla (kuva 17). Tarkemmat tiedot viestien rakenteesta ja sisällöistä liitteessä 2.

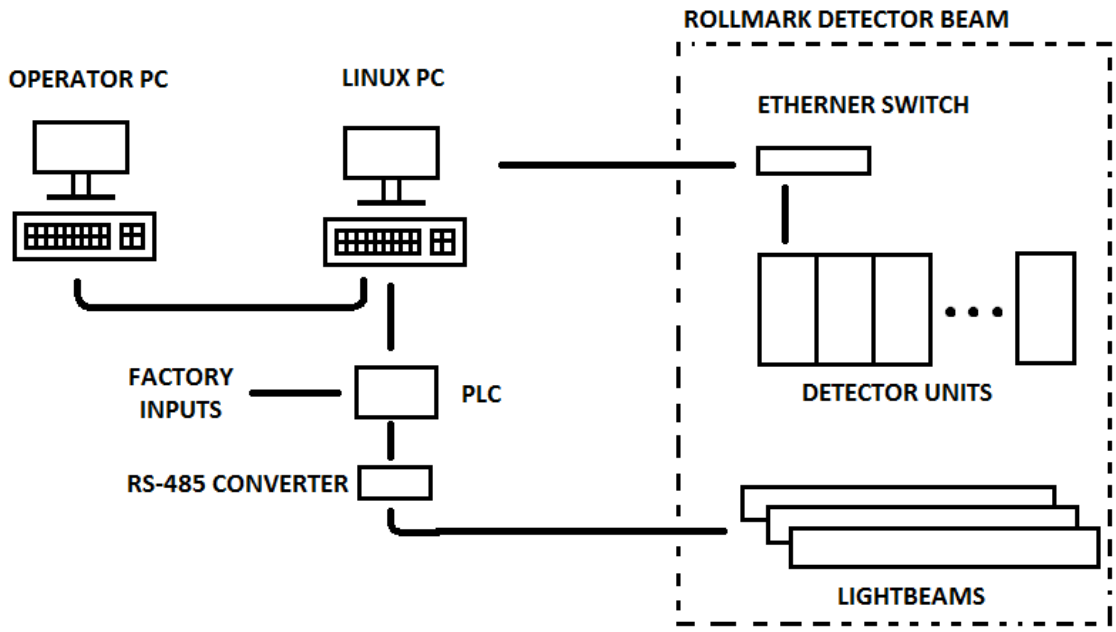


KUVA 17. Kommunikaatiosekvenssi (8, s. 8.)

4.3 Liitos järjestelmään

Logiikka liitettiin järjestelmään Linux PC:n sekä Rollmark-järjestelmän väliin (kuva 18). Linux PC:ltä menee Ethernet-kaapeli läheiseen järjestelmän

käyttämään sähkökaappiin. Logiikka liitetään tehtaaseen osana suurempaa järjestelmää. Logiikkaan tuodaan asiakkaan tarpeiden mukaisesti tehtaalta signaalitietoja, joiden perusteella ohjataan järjestelmän muita toimintoja. Logiikan lähettämät 24 voltin ohjausviestit piti muuntaa RS-485-signaaliksi, koska Rollmark-järjestelmä toimii kyseisellä signaalityypillä.



KUVA 18. Rollmark-järjestelmän tiedonsiirtoreitit

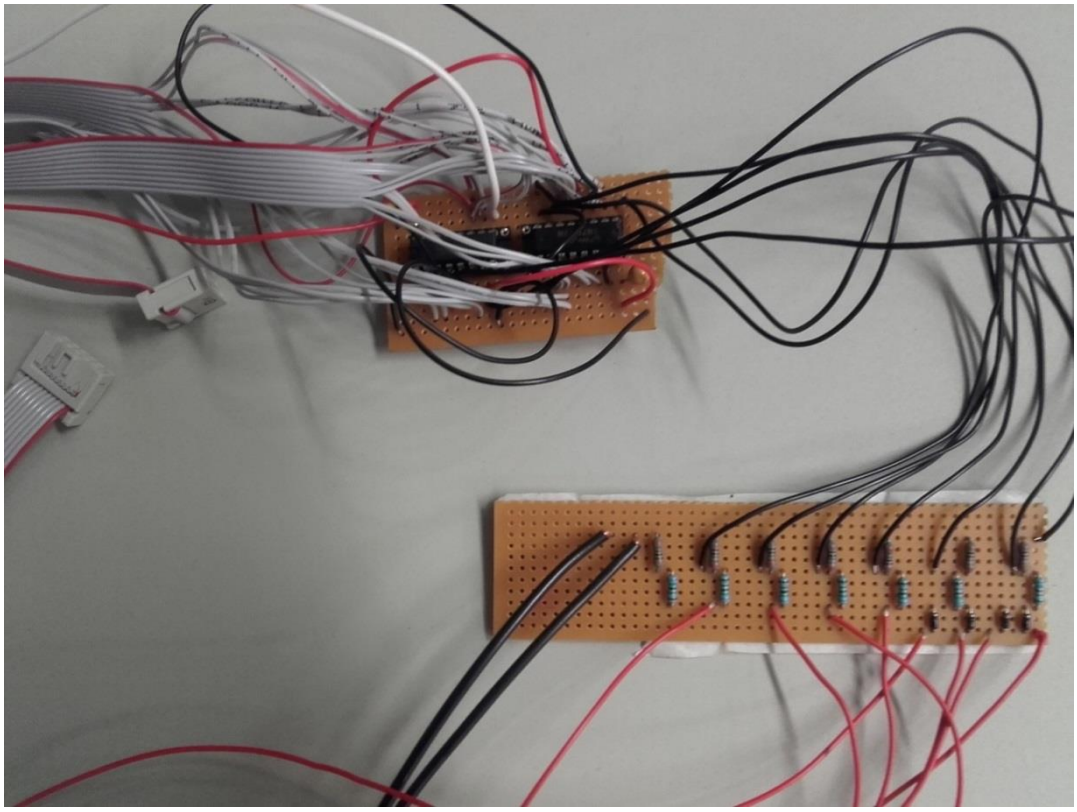
4.4 RS-485-muunnin

Rollmark-järjestelmän LED-ohjaimia ohjataan RS-485 signaaleilla. Järjestelmään suunniteltiin RS-485-muuntimen prototyyppi (kuva 19) tämän vuoksi. Muuntimen tarkoitus on muuntaa yksi signaali kahdeksi. Tulosignaalin ollessa 1-tilassa, lähtee RS-485-piiriltä toista linjaa pitkin 5 voltin signaali ja toiselta 0 voltin signaali. Tulosignaalin ollessa 0-tilassa on lähtevät signaalit päinvastaiset.

Logiikalta tuodaan lähdöiltä kaapelia pitkin signaalit muuntimille. Muuntimessa logiikan käyttämä 24 voltin jännite laskettiin 5 volttiin (kaava 1). 5 voltin jännitteet vietiin mikropiireille, jotka muunsivat jänniteviestin RS-485-signaaleiksi. Signaalit johdettiin 10-johdinta sisältävillä lattakaapeleilla valopalkkien LED-ohjaimille.

$$U1 = \frac{R1}{R1+R2} * U$$ *Kaava 1.*

Mikropiirien ollessa herkkä häiriöille, täytyi ohjausjänniteelle tehdä jännitteensuodatus. Piirien käyttöjännitteiden rinnalle asennettiin kondensaattori, joka suodattaa häiriöt jännitteestä. Johdotuskaavio ja tarkemmat tiedot käytetyistä vastuksista ja mikropiireistä liitteessä 1.



KUVA 19. RS-485 muuntimen prototyyppi

4.5 Logiikan muut toiminnot järjestelmässä

Järjestelmä valmistetaan yrityksessä asiakkaiden tilauksesta asiakkaan tarpeiden mukaisesti valmistajan parhaan kyvyn mukaisesti. Logiikkaan voidaan lisätä toimintoja, joilla voidaan valvoa tai ohjata tehtaan muita toimintoja. Logiikkaan ohjelmoitiin alustavasti vain yksi kiinteä toiminto, jolla voitaisiin testata järjestelmän johdotus. Toiminta asettaa sekvenssimäisesti signaalit 1-tilaan järjestyksessä sekunnin välein ja toistaa toimintoa, kunnes logiikka saa käskyn lopettaa. Toiminto käynnistyy logiikan saadessa Linux PC:ltä aloituskäskyn. Kyseinen toiminto lisättiin logiikkaan alkuperäisen järjestelmän

testaamisessa tulleiden ongelmien vuoksi. Toiminnon tarkoitus oli helpottaa vian paikantamista järjestelmässä.

5 YHTEENVETO

Työn tavoitteena ollut I/O-kortin korvaaminen ohjelmoitavalla logiikalla saatiin tehtyä. Ohjelmoitava logiikka valittiin järjestelmän vaatimusten mukaisesti sekä ylimääräinen lähtömoduuli nähtiin tarpeelliseksi lisättäväksi projektiin. Logiikka kommunikoi Linux PC:n kanssa halutulla tavalla sekä logiikan tärkein ominaisuus, valopalkkien kirkkauden säätö, toimi myös aikakaavioiden mukaisesti. Logiikan jälkeinen RS-485-muuntimesta tehty prototyyppi toimi testauksissa juuri kuten sen pitikin. Muuntimesta tehtiin johdotuspiirustus, josta myöhemmin tehdään tarkempi piirustus ja piirilevy.

Logiikan pääohjelma OB1 onnistui rakenteellisesti sekä toimivuuden kannalta erittäin hyvin. Suoritusjärjestys mahdollistaa ohjelman toimivuuden ja se on myöhemmin helposti muunneltavissa. Toimintolohkoja lukuunottamatta valopalkkien kirkkaudensäädöt onnistuivat myös yllättävän hyvin. Vaikka osa kirjoitetusta ohjelmasta näyttääkin sekavalta, toimivat toimintolohkot moitteettomasti testauksen aikana. Valopalkkien kirkkaudensäädöt olisi voinut ehkä tehdä yhteen toimintolohkoon, mutta työn aikataulun vuoksi näin helpommaksi tehdä kolme miltei samankaltaista toimintolohkoa yhden massiivisen toimintolohkon sijasta. Kyseiset toimintolohkot myös sisältävät vaiheita, jotka olisi voitu tehdä vähemmillä vaiheilla.

Suurimpana haasteena työssä oli ehdottomasti kommunikointi logiikan ja Linux PC:n välillä. Kokemuksen puute tästä aiheesta ei auttanut juurikaan. Ohjeita löytyi kuitenkin Siemensin internetpalstoilta, joilla pääsin alkuun. Monet omat testaukset auttoivat myös pääsemään asian ytimeen. Kommunikoinnin toimintaan saaminen vaati tiivistä yhteistyötä Poutasen Arin kanssa hänen ohjelmoidessaan Linux PC:n kommunikointimenetelmiä. Viestien rakenteiden piti olla juurikin tietynlaiset ja jokaiselle viestityypille oma viestitunnuksensa.

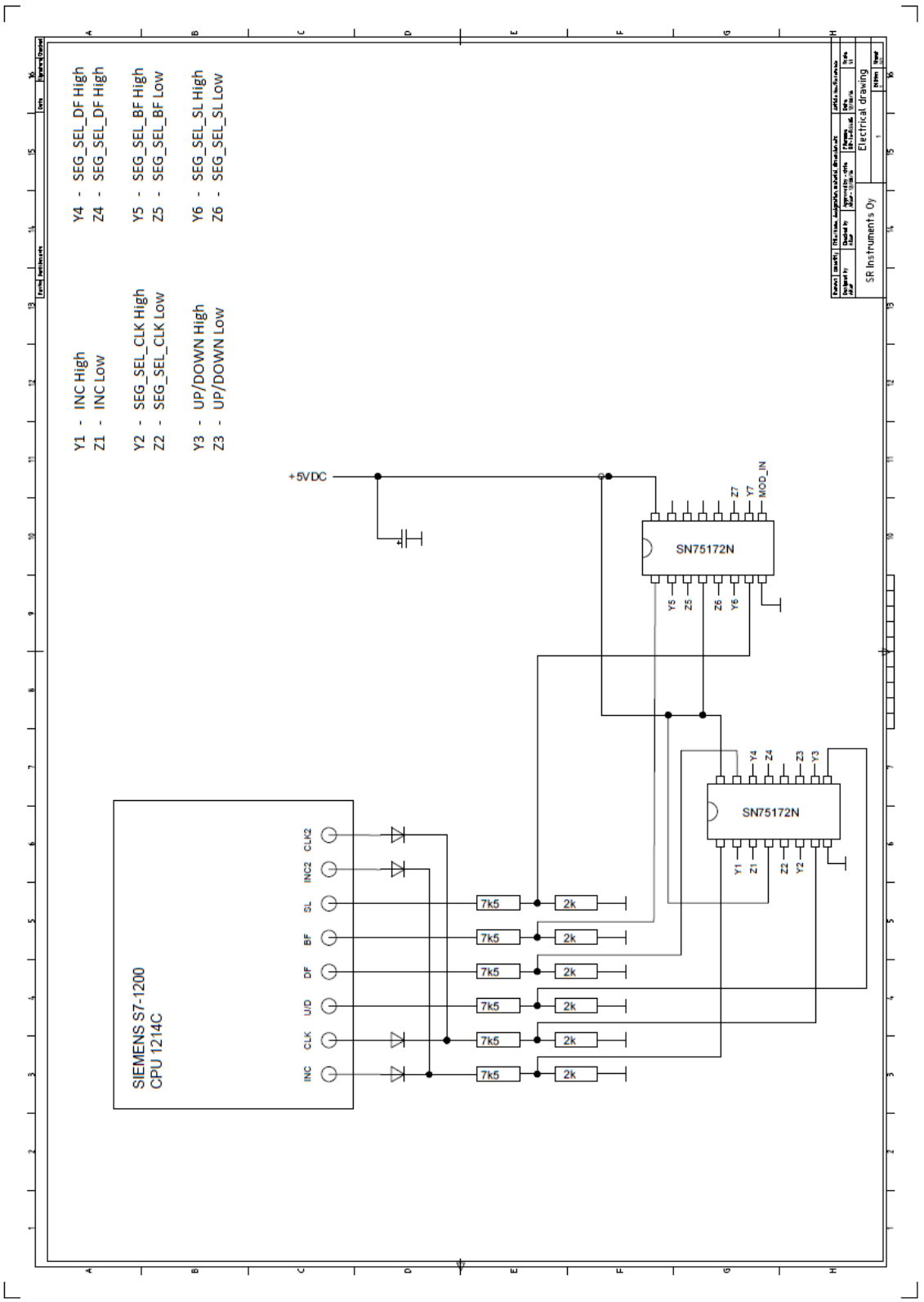
Miltei viimeisenä tehty RS-485-muunnin vaati pientä uudelleenopettelua. Elektroniikan opit olivat unohtuneet jo sen verran, että en kyennyt tätä ilman opastusta tekemään. Jännitteenjakoa sekä suodatuskondensaattoria lukuunottamatta kuitenkin suoritin tämänkin projektin osan itsenäisesti loppuun.

Logiikan valinta tehtiin siltä vaadittavian ehtojen perusteella. Jos projektiin olisi valittu Siemens S7-1500 -sarjan logiikka, olisi toimintolohkojen kirjoittaminen SCL-kielellä ollut helpompaa. S7-1500-sarjassa on yksi ominaisuus minkä olisin halunnut, mitä 1200-sarjassa ei ollut. ”Wait xx”-komennolla logiikan ohjelma odottaa määrätyn ajan, kunnes se jatkaa ohjelman suoritusta. Tässä projektissa jokainen viive tehtiin ajastimilla. Ohjelma olisi selkeytynyt sekä lyhentynyt, jos ”wait”-komento olisi ollut käytössä.

Tuotekehityksen puolesta I/O-moduulin korvaaminen ohjelmoitavalla logiikalla oli kannattavaa. Ohjelmoitava logiikka tarjoaa joustavuutta, mitä I/O-moduulin kiinteä elektroniikka ei tarjoa. Logiikka on myös paljon yksinkertaisempi ratkaisu ja sen laajentamismahdollisuudet ovat melkein rajattomat.

LÄHTEET

1. Pyörret, Seppo 2008. Rollmark specification. SR Instruments Oy.
2. Gonzales, Carlos 2015. Engineering Essentials: What Is a Programmable Logic Controller? Machine Design. Saatavissa: http://machinedesign.com/engineering-essentials/engineering-essentials-what-programmable-logic-controller. Hakupäivä 8.9.2016.
3. PLC History. Saatavissa: http://www.plcs.net/chapters/history2.htm. Hakupäivä 12.9.2016.
4. Anand, Lakshmi 2008. Mechatronics. Unit 4: Programmable Logic Controller. Ladder Diagram. Saatavissa: http://www.ustudy.in/node/2914. Hakupäivä 12.9.2016.
5. Van der Wal, Eelco. Structuring Program Development with IEC 61131-3. Saatavissa: http://www.plcopen.org/pages/benefits/program_development/. Hakupäivä 12.9.2016.
6. Structured Control Language for S7-300/S7-400 Programming. 1998. Manual. Siemens. Saatavissa: https://cache.industry.siemens.com/dl/files/188/1137188/att_27471/v1/SCLV4_e.pdf. Hakupäivä 12.9.2016.
7. Open User Communication with TCON, S7-1500 CPU. 2014. Application description 02/2014. Siemens. Saatavissa: https://cache.industry.siemens.com/dl/files/807/58875807/att_110536/v1/58875807_net_tcon_s7-1500_en.pdf. Hakupäivä 8.9.2016.
8. Korvala, Ari 2016. Communication spec for Linux PC – logic controller. SR Instruments Oy.


Y1 - INC High
Z1 - INC Low
Y2 - SEG_SEL_CLK High
Z2 - SEG_SEL_CLK Low
Y3 - UP/DOWN High
Z3 - UP/DOWN Low
Y4 - SEG_SEL_DF High
Z4 - SEG_SEL_DF High
Y5 - SEG_SEL_BF High
Z5 - SEG_SEL_BF Low
Y6 - SEG_SEL_SL High
Z6 - SEG_SEL_SL Low
+5VDC
SN75172N
SN75172N
MOD_IN
SIEMENS S7-1200
CPU 1214C
CLK2
INC2
SL
BF
DF
U/D
CLK
INC
7k5
2k
SR Instruments Oy
Electrical drawing

Communication Specification

Linux PC – Siemens s7-1200

Of the Rollmark Applications

Table of Contents

1. About this document **3**
1.1. Terminology and format of writing **3**
2. Hardware description of the communication **4**
2.1. Communication Format **4**
2.2. Communication Speed **4**
2.3. Reception And Sending of 32-bit Parameters in the Message data **4**
2.4. Communication Protocol Addresses **4**
The logic controller IP address and used port is set inside the program of the logic controller. Also, the Linux PC IP address is set into the logic controllers program. The logic controller will not communicate with any device with a different IP address. 4
3. Message Definitions Between Linux PC and logic controller **5**
3.1. Lightbeam messages **5**
3.1.1. Lightbeam parameters 5
3.1.2. Lightbeam parameters received 5
3.1.3. Lightbeams programmed 6
3.2. Output Messages **6**
3.2.1. Set output/s 6
3.2.2. Outputs set message 6
3.3. Input Message **7**
3.3.1. Input state message 7
3.3.2. Input query message 7
3.4. Watchdog Message **7**
3.5. Online diagnostic Message **8**
3.5.1. System online message 8
3.5.2. System online reply messages 8
4. Communications sequence between Linux PC and logic controller **9**
Change record **10**

1. About this document

This document specifies the TCP/IP communication between the Linux PC and the logic controller.

1.1 Terminology and format of writing

Whenever there is a description of data message, it is written in hexadecimal format '0 0' where data to be sent is 00 00 00 00 00 00 00 00. These 16 digits form an 8-byte message, which the logic controller converts to float-point numbers.

2. Hardware description of the communication

2.1 Communication Format

Communication is done in binary format. Data is sent in 4-byte clusters, which the logic controller converts into float-point numbers. The used communication standard is TCP/IP. Linux PC includes additional Ethernet card for the communication.

The logic controller uses float-point numbers for most of the control operations. This was made so because the Motion Control technology, which controls the INC pulse, requires float-point number values for its control and because the logic controls received data is all saved into one array. The data type of array cannot be changed without changing the program via programming software.

2.2 Communication Speed

The communication speed between Linux PC and logic controller is 100 Mbps.

2.3 Reception And Sending of 32-bit Parameters in the Message data

Since one byte can only represent a value from 0 to 255, a 32-bit parameter has to be sent in four bytes. The logic controller uses the “high byte” architecture. In this case, the logic controller sends higher 8 bits first with the lower 8 bits following.

The Linux PC represents the “low byte” architecture which sends the lowest 8 bits first with the higher bits following. The use of this architecture requires that the places of bytes need to be swapped.

For example: The Linux PC sends a 32-bit data message, A1A2A3A4 to the logic controller. The Linux PC must first change the message to fit the “high byte” architecture of the logic controller. So the message sent is A4A3A2A1.

The Linux PC also has to swap the bytes in received messages from the logic controller.

2.4 Communication Protocol Addresses

The logic controller IP address and used port is set inside the program of the logic controller. Also, the Linux PC IP address is set into the logic controllers program. The logic controller will not communicate with any device with a different IP address.

3. Message Definitions Between Linux PC and logic controller

This section describes all the Linux PC – logic controller messages. They use the TCP/IP protocol. Only the bytes inside the data section of the Ethernet frame are specified here, because the first bytes in the Ethernet frame are written elsewhere. The messages are divided into groups depending on their purpose and nature. The grouping is described in the following table.

Description	Message ID
Lightbeam messages	1
Output messages	2
Input message	3
Watchdog message	4
Online diagnostic messages	5
Wiring test messages	7

Table 1. TCP/IP message groups

3.1 Lightbeam messages

3.1.1 Lightbeam parameters

With this message the Linux PC sends the lightbeam parameters to the logic controller. The logic controller applies to parameters to the lightbeams accordingly. Lenght of sent message depends on the amount of segments to be programmed in Rollmark.

Direction	Message	Length bytes
PC → Logic controller	`0 5 1 <DF Values> 101 <BF Values> 101 <SL Values> 101`	Varies

Field	Description	Type
0	Application ID (0 = rollmark)	
5	Protocol ID (5 = logic controller communication)	
1	Identifier of Lightbeam message	4-byte
<DF Values>	Contains float-point numbers 0-100. Used to program the DF lightbeams intensities. Amount of these varies on the Rollmark.	4-byte
101	Logic controller uses this to move on to next lightbeams setting or lightbeam settings are done (after SL setting).	
<BF Values>	Contains float-point numbers 0-100. Used to program the BF lightbeams intensities. Amount of these varies on the Rollmark.	4-byte
<SL Values>	Contains float-point numbers 0-100. Used to program the SL lightbeams intensities. Amount of these varies on the Rollmark.	4-byte

3.1.2 Lightbeam parameters received

Logic controller replies to the Linux PC after receiving lightbeam parameters.

Direction	Message	Length bytes
PC ← Logic controller	1 1	8

Field	Description	Type
1	Identifier of lightbeam message	4-byte
1	“Parameters received” message	4-byte

3.1.3 Lightbeams programmed

Logic controller sends a message after the lightbeams have been programmed.

Direction	Message	Length bytes
PC ← Logic Controller	1 2	8

Field	Description	Type
1	Identifier of lightbeam message	4-byte
2	“Lightbeams programmed” message	4-byte

3.2 Output Messages

3.2.1 Set output/s

This message is used to control the logic controllers outputs. Multiple outputs can be set to state 0 or 1 at the same time by adding (<Outputnumber> <state 0 or 1>)'s to the message before 101. Lenght of the message also varies depending on the amount of outputs controlled by the message

Direction	Message	Length bytes
PC → Logic Controller	0 5 2 <Output number (0-9)> <state 0 or 1> 101	Varies

Field	Description	Type
0	Application ID (0 = rollmark)	4-byte
5	Protocol ID (5 = logic controller communication)	4-byte
2	Identifier of output message	4-byte
<Output number (0-9)>	Determines which output, 0-9, is controlled.	4-byte

<state 0 or 1>	Determines in which state the output is set to, 0 or 1.	4-byte
101	A breakpoint in the message, to notify the logic controller when the message ends.	4-byte

3.2.2 Outputs set message

The Logic controller replies to the Linux PC after outputs set to state 0 or 1 after receiving the orders to set output/s to state 0 or 1.

Direction	Message	Length bytes
PC ← Logic Controller	`2 <Out0> <Out1> <Out2> <Out3> <Out4> <Out5> <Out6> <Out7> <Out8> <Out9>`	44

Field	Description	Type
2	Identifier of output message	4-byte
<OutX>	State of output X, 0 or 1.	4-byte

3.3 Input Message

3.3.1 Input state message

The logic controller sends this message anytime an input changes it's state from 0 to 1 or 1 to 0 and once when communications connection is established between the logic controller and Linux PC. The message contains the state of all inputs in order 0 to 13.

Direction	Message	Length bytes
PC ← Logic Controller	`3 <In0> <In1> <In2> <In3> <In4> <In5> <In6> <In7> <In8> <In9> <In10> <In11> <In12> <In13>`	60

Field	Description	Type
3	Identifier of input message	4-byte
<InX>	State of input X, 0 or 1.	4-byte

3.3.2 Input query message

The Linux PC can ask the logic controller the state of inputs with this message. The logic controller replies with Input state message.

Direction	Message	Length bytes
PC → Logic Controller	0 5 3	12

Field	Description	Type
0	Application ID (0 = rollmark)	4-byte
5	Protocol ID (5 = logic controller communication)	4-byte
3	Identifier of input message.	4-byte

3.4 Watchdog Message

The logic controller sends a watchdog message every 10 seconds. This message is used to inform the Linux PC that the communication connection is alive.

Direction	Message	Length bytes
PC ← Logic Controller	4	4

Field	Description	Type
4	Watchdog message.	4-byte

3.5 Online diagnostic Message

3.5.1 System online message

This message is sent from the Linux PC to the logic controller to ask if the system is online.

Direction	Message	Length bytes
PC → Logic Controller	0 5 5	12

Field	Description	Type
0	Application ID (0 = rollmark)	4-byte
5	Protocol ID (5 = logic controller communication)	4-byte
5	Identifier of online diagnostic message.	4-byte

3.5.2 System online reply messages

The logic controller replies to the Linux PC if the system is online after being asked.

Direction	Message	Length bytes
PC ← Logic Controller	5	4

Field	Description	Type
5	System online message.	4-byte

3.6 Wiring test messages

3.6.1 Wiring test ON messages

This message is sent from the Linux PC to the logic controller to start the wiring test.

Direction	Message	Length bytes
PC → Logic Controller	0 5 7 1	16

Field	Description	Type
0	Application ID (0 = rollmark)	4-byte
5	Protocol ID (5 = logic controller communication)	4-byte
7	Identifier of wiring test message.	4-byte
1	Wiring test ON	4-byte

3.6.2 Wiring test OFF message

This message is sent from the Linux PC to the logic controller to stop the wiring test.

Direction	Message	Length bytes
PC → Logic Controller	0 5 7 0	16

Field	Description	Type
0	Application ID (0 = rollmark)	4-byte
5	Protocol ID (5 = logic controller communication)	4-byte

7	Identifier of wiring test message.	4-byte
0	Wiring test OFF	4-byte

4. Communications sequence between Linux PC and logic controller

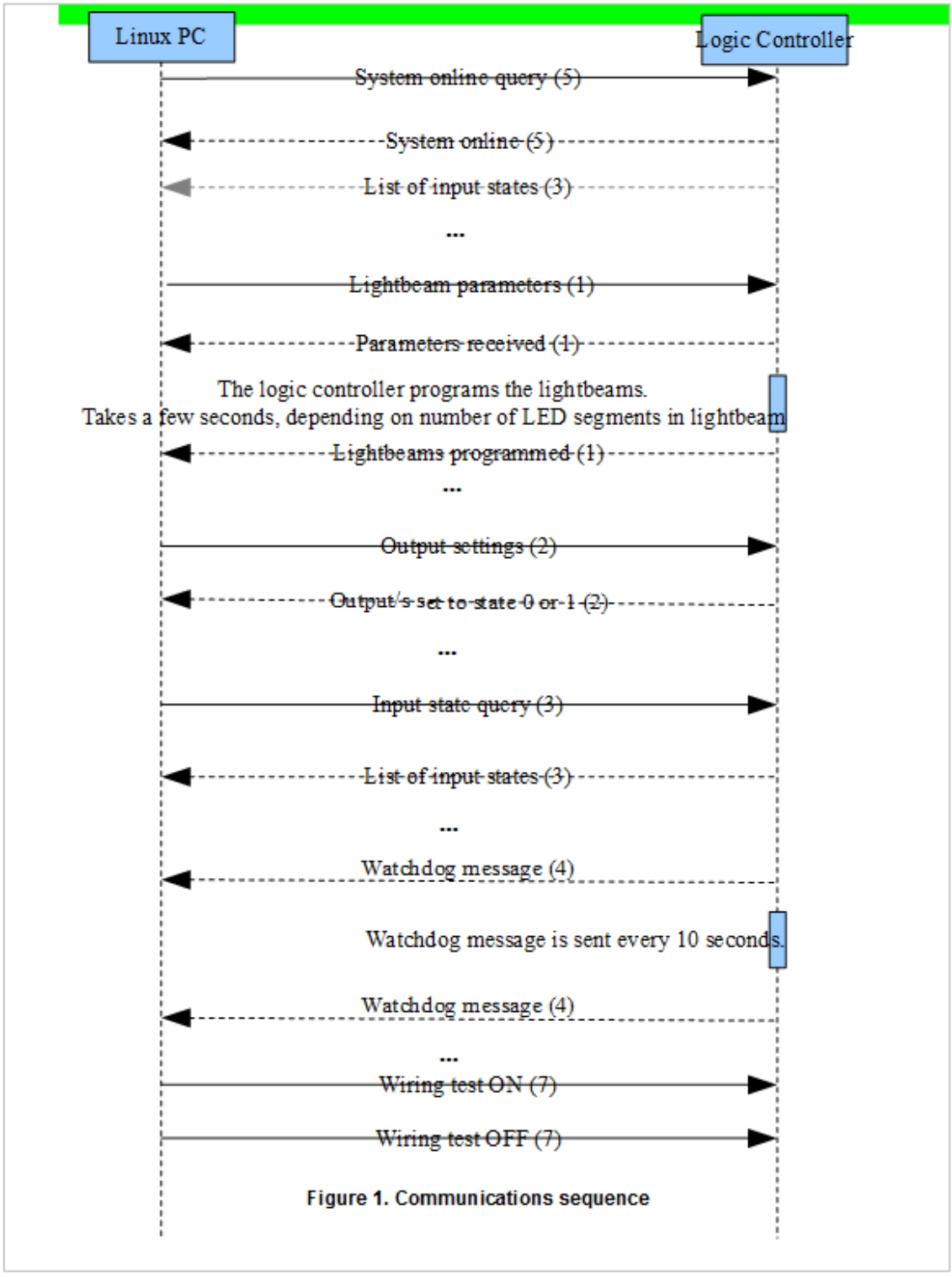


Figure 1. Communications sequence

Change record

Rev	Date issued	Person	Description of Change
A	2016-06-30	A. Korvala	First draft.
B	2016-06-30	A. Korvala	More details on Chapter 2 and recalculated message lenghts
C	2016-08-31	A. Korvala	Added wiring test messages