Pekka Linna

IEC 60870-5-101 -PROTOKOLLAN KÄYTTÖ SIEMENS S7 -POHJAISISSA KAUKOKÄYTTÖRATKAISUISSA

Opinnäytetyö
CENTRIA AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Syyskuu 2013

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö Kokkola-Pietarsaari	**Aika** Syyskuu 2013	**Tekijä/tekijät** Pekka Linna
Koulutusohjelma Tietotekniikan koulutusohjelma		
Työn nimi IEC 60870-5-101 -protokollan käyttö Siemens S7 -pohjaisissa kaukokäyttöratkaisuissa		
Työn ohjaaja Hannu Ala-Pöntiö		**Sivumäärä** 39
Työelämäohjaaja Matti Pajukangas		

Työn tarkoituksena oli luoda toimiva rajapinta Siemens S7 -pohjaiselle logiikalle, kun kommunikointiprotokollana on IEC 60870-5-101. Tavoitteena oli lisäksi saada tehtyä toimiva rajapinta myös Siemensin uusimman sukupolven ohjelmointityökalulla, jolle ei ollut vielä omia ohjelmalohkoja tehty tätä protokollaa varten.

Työssä perehdyttiin IEC 60870-5-101 -protokollan ja Siemensin ohjelmalohkojen rakenteeseen ja toimintaan. Niiden perusteella luotiin logiikkaohjelma, johon itse rajapinta määriteltiin.

Tuloksina saatiin toimiva rajapinta ja onnistunut kommunikointi logiikan ja SCADA-järjestelmän välille molemmilla kokeilluilla ohjelmointityökaluilla. Opinnäytetyön avulla Apex Automation Oy sai tietoa ja kokemusta kyseisen logiikkaperheen ja kommunikointiprotokollan yhdistämisestä ja käytöstä.

Asiasanat
kommunikointiprotokolla, ohjelmoitava logiikka, rajapinta, SCADA

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Unit	Date	Author
Kokkola-Pietarsaari	September 2013	Pekka Linna

Degree programme
Information Technology

Name of thesis
Using IEC 60870-5-101 protocol in Siemens S7 based remote access systems

Instructor	Pages
Hannu Ala-Pöntiö	39

Supervisor
Matti Pajukangas

The objective of this thesis was to create a working interface for Siemens S7 based PLC using communication protocol IEC 60870-5-101. In addition, the aim was to create a working interface using Siemens' newest generation programming tool, which did not have its own program blocks for this protocol.

While working on the thesis, the researcher familiarized himself with the structure and operation of the IEC 60870-5-101 protocol and Siemens program blocks. Based on this, the PLC program was created. The interface itself was defined to the PLC program.

A working interface and successful communication between the PLC and the SCADA system was found with both of the tested programming tools. With the help of this thesis, Apex Automation Oy received valuable knowledge and experience of combining and utilizing the aforementioned PLC family and communication protocol.

Key words
communication protocol, interface, PLC, SCADA

LYHENTEET

APCI	Application Protocol Control Information
APDU	Application Protocol Data Unit
ASDU	Application Service Data Unit
EPA	Enhanced Performance Architecture
FBD	Function Block Diagram
IEC	International Electrotechnical Commission
I/O	Input/Output
LAD	Ladder Diagram
OSI	Open Systems Interconnection
PLC	Programmable Logic Controller
RAM	Random Access Memory
RTU	Remote Terminal Unit
SCADA	Supervisory Control And Data Acquisition
STL	Statement List
TIA	Totally Integrated Automation

TAULUKOT

1 JOHDANTO

Monelle nykyaikaiselle automaatiojärjestelmälle löytyy kaksi yhteistä nimittäjää, ohjelmoitava logiikka ja käytönvalvontajärjestelmä, eli Supervisory Control And Data Acquisition (SCADA) -järjestelmä. Ohjelmoitava logiikka kerää antureiden, kytkimien ja erilaisten mittausten avulla kentältä informaatiota sekä ohjaa toimilaitteita. Käytönvalvontajärjestelmä puolestaan tuo tämän kerätyn informaation valvomoon operaattorin eli käyttäjän saataville sekä toimittaa käyttäjän antamia ohjauksia ja asetusarvoja ohjelmoitavalle logiikalle. Jotta nämä toiminnot suuntaan tai toiseen olisivat mitenkään mahdollisia, täytyy logiikka ja käytönvalvontajärjestelmä saada keskustelemaan keskenään.

Erilaisia kommunikointiprotokollia löytyy useita, ja jotkut niistä ovat nousseet vuosien saatossa jo melkoiseen hallitsija-asemaan. Yksi tällainen on esimerkiksi Modiconin kehittämä Modbus-prokolla. Myös International Electrotechnical Commissionin (IEC) kehittämät protokollat, kuten IEC 60870-5-101 ja IEC 60870-5-104 ovat ottaneet markkinoilta oman jalansijansa. IEC:n 101- ja 104-protokollat sisältävät kuitenkin yhden suuren edun Modbusiin verrattuna, sillä niissä kentältä tuleva informaatio voidaan varustaa aikaleimalla. Aikaleiman avulla saadaan helposti selville, milloin esimerkiksi jokin kriittinen kytkin on vaihtanut tilaansa tai hälytys on mennyt päälle.

Kommunikointiprotokollia on useita, mutta laaja on myös ohjelmoitavien logiikoiden kenttä. Ja kuten voi olettaa, eri protokollien sovittaminen eri valmistajien logiikoihin ei aina käy samalla tavalla. Minun tehtäväni ja myös tämän opinnäytetyön tarkoitus oli luoda toimiva rajapinta Siemens S7 -logiikkaperheen tuotteen ja käytönvalvontajärjestelmän välille, kun ne keskustelevat IEC 60870-5-101 -protokollalla. Rajapinta on se portti, jonka kautta käytönvalvontajärjestelmä pääsee lukemaan logiikan muistista tietoja, tai vastaavasti kirjoittamaan sinne. Rajapintaan piti sisältyä automaatiojärjestelmien neljä oleellisinta toimintoa, eli tilatietojen ja mittausten lukeminen sekä binaariohjausten ja parametrien kirjoittaminen. Työ rajattiin vain logiikkaohjelman tekoon, jolloin ylätasoa, eli valvomosovellusta ei siihen sisällytetty. Toki logiikkaohjelma ja rajapinta testattiin toimivan

SCADA-järjestelmän kanssa, mutta sen konfiguroimisen hoiti eri henkilö. Testauksessa käytettiin ABB:n MicroSCADAa.

Työllä pyrittiin luomaan hyvä perusta Siemens S7 -logiikan ja IEC 60870-5-101 -protokollan käytölle ja saamaan siitä Apex Automation Oy:lle tietoa ja kokemusta, jota se voi myöhemmin hyödyntää projekteissaan. Jotta haluttu rajapinta olisi mahdollista luoda, on logiikkaohjelmassa käytettävä Siemensin toimittamia ohjelmalohkoja. Nämä ohjelmalohkot ovat ainakin toistaiseksi saatavilla vain yhtiön vanhemmalle Simatic Manager Step 7 -ohjelmointityökalulle. Niinpä työn yhdeksi tavoitteeksi asetettiin ohjelmalohkojen onnistunut kääntäminen Siemensin uusimman sukupolven ohjelmointityökalulle Totally Integrated Automation (TIA) -portaalille ja rajapinnan toimiminen myös TIA:lla tehdyllä ohjelmalla.

2 SCADA-JÄRJESTELMÄT

SCADA-järjestelmä käsittää niin laitteistoin kuin myös ohjelmiston. SCADA oli alun perin tarkoitettu kommunikoimaan vain Remote Terminal Units (RTU) -asemien kanssa ja toimimaan niiden isäntänä, mutta myöhemmin SCADA:n ohjelmistot kehitettiin yhteensopiviksi myös ohjelmoitavien logiikoiden kanssa. Jotkut SCADA-järjestelmiä tekevät yritykset käyttävät sellaisia ohjelmistoja, mihin vain heillä on käyttöoikeus ja mitkä ovat erityisesti vain heidän käyttötarkoituksiin suunniteltuja. Heidän lisäkseen on olemassa myös yrityksiä, jotka luottavat itsenäisten SCADA ohjelmistokehittäjien tekemään avoimeen ohjelmistoon, mikä tukee laajasti eri kommunikointi protokollia. (Strauss 2003.)

SCADAn tehtävä on tarjota seuraavat toiminnot:

- Näyttää reaaliaikaisen datan, jonka se saa kentältä.
- Tallentaa kerättyä dataa.
- Aktivoi hälytykset.
- Näyttää tapahtumaraportit ja häiriötallennukset, kun niitä pyydetään.
- Antaa ohjauskäskyjä kenttälaitteille.
- Käyttöliittymä operaattorille. (Strauss 2013.)

SCADAn tiedonkeruu viittaa kentältä tulevan informaation vastaanottoon, sen analysointiin ja käsittelyyn. Reaaliaikainen data näytetään yleensä erilaisilla grafiikoilla käyttäjän määrittelyjen mukaisesti. Datalle voidaan antaa raja-arvoja, joiden alittaminen tai ylittäminen aiheuttaa hälytyksen. Ohjauskäskyt SCADAlta kenttälaitteille lähtevät joko käyttäjän antamina tai automaattisesti ennalta määrättyjen asetusten mukaisesti. Se, miten data tallennetaan ja ”arkistoidaan”, riippuu laitteistosta, ohjelmistosta ja käyttäjän tekemästä konfiguraatiosta. Kerättyä dataa tarvitaan erilaisten trendien muodostamiseen, vian hakuun ja raportointiin. Graafisella käyttöliittymällä käyttäjä pääsee näkemään tehdaslaitoksen tms. tilan ja ohjaamaan sitä. Hyvä käyttöliittymä sisältää eritasoisia näyttöjä, joista päänäytön pitäisi antaa kokonaiskuva laitoksesta vain tärkeimmillä tiedoilla. Alatason näyttöihin pilkotaan

laitos ja sen prosessit tms. tarkemmin ja näytetään laajempi informaatio sekä mahdolliset ohjaukset. (Strauss 2013.)

SCADAn ja ala-asemien välisiä kommunikointiprotokollia ovat esimerkiksi Modiconin kehittämät sarjaliikenteinen Modbus RTU ja Transmission Control Protocol/Internet Protocol (TCP/IP) -pohjainen Modbus TCP. Lisäksi käytettyjä protokollia ovat mm. IEC 61850, IEC 60870-5-101 ja sen TCP/IP-pohjainen versio IEC 60870-5-104 sekä Distributed Network Protocol (DNP3). (Kang & Robles 2009.)

3 IEC 60870-5-101

IEC:n Tekninen Komitea 57 kehitti IEC 60870-5-101 -sarjaliikennestandardin vuonna 1995. IEC 60870-5-101 hyödyntää useita asiakirjoja IEC 60870-5 -standardista, ja se olikin aikanaan ensimmäinen täydellisesti toimiva SCADA-protokolla IEC 60870-5:n standardeista. Se määrittelee kaikki tarvittavat sovellustason funktiot ja tieto-oliot tarjotakseen luotettavan etäkäyttösovelluksen maantieteellisesti laajasti hajautetuille prosesseille. IEC 60870-5-101 kattaa yleisen kommunikoinnin RTU-asemien kanssa, mukaan lukien tietotyypit ja palvelut, jotka ovat sähköasemien ja ala-asemien kanssa yhteensopivia. Datatyypit ovat yleisiä, joten ne sopivat myös laajemmin SCADA-sovelluksiin. (Reynders, Mackay & Wright 2005.)

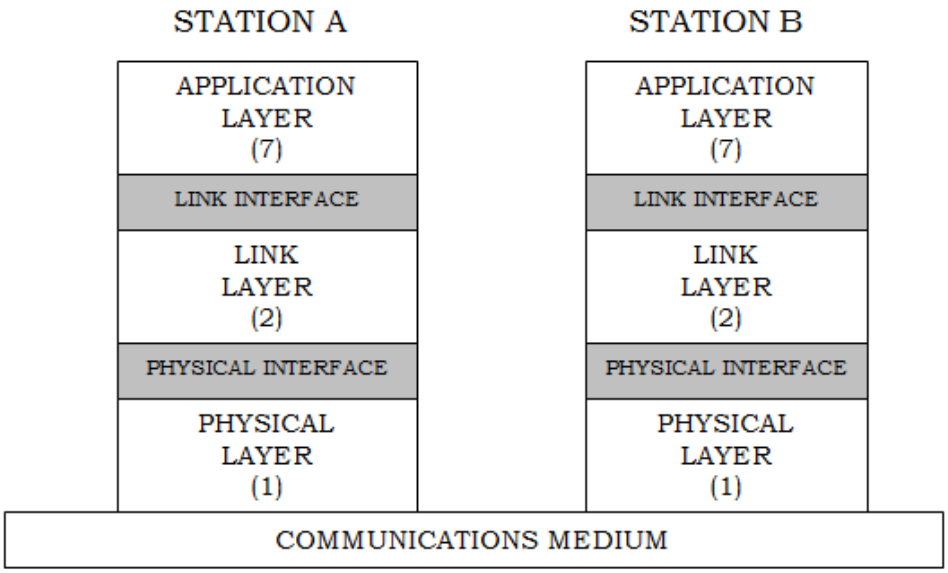
Laitteita, joiden välillä kommunikointi tapahtuu, kutsutaan yleensä asemiksi, ja jokainen asema suorittaa omia sovellusprosesseja (Application Processes). Isäntäasema (Master) voi esimerkiksi hallita tietokantaa, joka sisältää kaikki orja-asemien (Slave) keräämät tiedot ja lähettää orja-asemille käskyjä. Orja-asemien sovellusprosesseihin kuuluvat muun muassa paikallisten mittaustietojen tarkkailu, niiden lukeminen ja tallentaminen sekä paikallisten käskyjen suorittaminen. Isäntäaseman ja orja-asemien sovellusprosessit ovat yhteydessä toisiinsa määritellyn kommunikointiprotokollan mukaisesti. (Cole 2002.)

3.1 IEC 60870-5-101 -kerrokset

IEC 60870-5-101 perustuu Enhanced Performance Architecture (EPA) -malliin, ja se käsittää vain kolme Open System Interconnection (OSI) -kerrosta. Nämä kerrokset ovat

- Fyysinen kerros (Physical Layer)
- Yhteyskerros (Link Layer)
- Sovelluskerros (Application Layer) (ABB Oy 2012).

Protokollan kerrokset havainnollistetaan kuviossa 1.



KUVIO 1. Asemien välinen kommunikointi EPA-mallin mukaisesti (Cole 2002.)

Jokaisen aseman OSI-kerrokset voidaan kuvitella yhdeksi protokollanipuksi. Kun esimerkiksi kuviossa 1 näkyvä Asema A lähettää tietoja Asema B:lle, sovellusdata otetaan vastaan protokollanipun huipulla. Sieltä sovellusdata liikkuu alaspäin läpäisten jokaisen kerroksen ja keräten niiltä kaiken tarvittavan lisädatan, jonka protokolla vaatii toimiakseen. Lopulta tämä yhteen kerätty datajono saapuu sarjamuodossa protokollanipun pohjalle, ja se siirretään sieltä vastaavasti Asema B:n protokollanipun pohjalle. Asema B:ssä datanippu etenee kerros kerrokselta ylöspäin, ja jokainen kerros nappaa siitä itselleen kuuluvan datan, kunnes alkuperäinen sovellusdata saapuu protokollanipun huipulle ja toimitetaan Asema B:n sovellusprosessille. Jokaisen kerroksen data siirretään siis aina vastaavaan kerrokseen vastaavalle paikalle kohdeasemassa. (Cole 2002.)

3.1.1 Fyysinen kerros

Fyysinen kerros huolehtii bittien ja tavujen vastaanottamisesta ja lähettämisestä ulkoisten tietoliikenneyhteyksien välityksellä. Sitä ei kiinnosta bittien ja tavujen merkitys, ainoastaan niiden siirtäminen. Se tarjoaa fyysisen rajapinnan esimerkiksi kupari- tai kituliitynnälle käyttäen International Telecommunication Union (ITU-T)

-standardin V.24/V.28 (RS-232) tai X.24/X.27 (RS-485) signaaleja. (Reyenders ym. 2005.)

3.1.2 Yhteyskerros

Yhteyskerros on vastuussa siitä, että siirrettävä data kulkee yhteystunnelissa, ja takaa, että vastaanottaja saa datan täydellisenä ilman virheitä. Se käyttää asynkronista FT1.2-kehysformaattia, joka sisältää sellaista ohjausinformaatiota kuten määränpään osoitteen ja tarkistusdataa virheiden havaitsemiseen. Vaihtoehtona olisi ollut myös tehokkaampi FT2-kehysformaatti, mutta se vaatii oman harvinaisemman laitteiston, mikä nostaisi hintoja. Niinpä on hyväksytty se pieni menetys tehokkuudessa, että protokolla olisi helposti käytettävissä ja soveltuisi yleisesti teollisuuden käytössä oleviin laitteistoihin. (Cole 2002.)

Yhteyskerros voi toimia joko balansoimattomassa tai balansoidussa tilassa. Balansoimattomassa tilassa vain isäntäasema voi aloittaa tietojen siirron. Aina kun isäntäasema haluaa tietoja orja-asemilta, se luo yhteyden ja suorittaa kyselyt. Näin ei pääse syntymään sellaisia virhetilanteita, että isäntäasema ja orja-asemat yrittäisivät lähettää tietoja samaan aikaan, mikä on balansoimattoman tilan etu. Balansoidussa tilassa taas mikä tahansa asema voi toimia ensisijaisena asemana ja näin aloittaa tietojen siirron. (Reyenders ym. 2005.)

Yhteyskerros tarjoaa ”käyttäjälle” eli sovellus-kerrokselle kolme erilaista siirtotapaa, jolla siirtää halutut tiedot. Nämä tavat ovat seuraavat:

- Lähetä/Ei vastausta (Send/No Reply)
- Lähetä/Vahvista (Send/Confirm)
- Pyydä/Vastaa (Request/Respond) (Reyenders ym. 2005.)3

Lähetä/Ei vastausta -tapaa käytetään, kun lähetettävään viestiin tai käskyyn ei ole tarpeellista saada kuittausta vastaanottavalta asemalta. Lähetä/Vahvista-tapa valitaan, kun lähetettävän käskyn tai datan on mentävä luotettavasti perille. Tällöin vastaanottaja joutuu kuittaamaan lähettäjälle, että toimitus on onnistunut. Pyydä/Vastaa-tapa on samankaltainen kuin Lähetä/Vahvista-tapa, mutta siinä pelkän

kuittausviestin sijaan lähetetään pyydetty data, jos se on saatavilla. (Reyenders ym. 2005.)

3.1.3 Sovelluskerros

Sovelluskerroksella määritellään Application Service Data Unit (ASDU) -kehys. Jokaisessa ASDUssa on tietoelementti, joka sisältää varsinaisen sovellusdatan eli asemien sovellusprosessien välillä liikuteltavan datan. ASDU-kehyksen tyyppi kertoo, mitä kyseinen ASDU tekee ja minkälaiselle datalle se on määritelty. Tyypit ovat valmiiksi määriteltyjä, ja niitä on kaikkiaan 58 erilaista. (Reyenders ym. 2005.) ASDUn rakenteeseen pureudutaan tarkemmin myöhemmässä vaiheessa. Seuraavassa on muutama esimerkki erilaisista ASDU-tyypeistä.

Valvontasuunta (Monitor Direction):

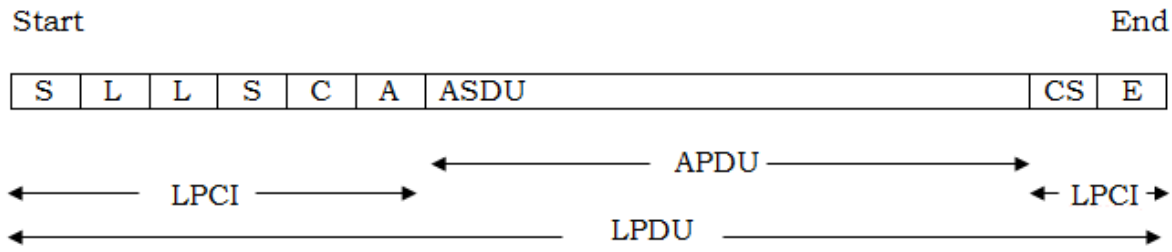
- yksittäisen bitin lukeminen
- yksittäisen bitin lukeminen aikaleimalla
- mittausarvon lukeminen, normaali arvo
- mittausarvon lukeminen, normaali arvo aikaleimalla
- mittausarvon lukeminen, skaalattu arvo
- mittausarvon lukeminen, skaalattu arvo aikaleimalla. (Reyenders ym. 2005.)

Ohjaussuunta (Control Direction):

- yksittäisen bitin ohjaus
- asetusarvo, normaali arvo
- asetusarvo, skaalattu arvo
- orja-asemien yleinen kysely
- ajan synkronointi
- testikäsky. (Reyenders ym. 2005.)

3.2 IEC 60870-5-101 -viestin rakenne

IEC 60870-5-101 -viesteillä on asemien ulkopuolella sisäkkäinen rakenne, joka johtuu protokollan kerrosmaisesta rakenteesta. Kaikki kuviossa 2 näkyvät data-alueet ovat yhden tai useamman tavun (8 bittiä) kokoisia. EPA-mallin protokollissa on ASDUihin lisätty Application Protocol Control Information (APCI) -kehys, että siitä saadaan muodostettua Application Protocol Data Unit (APDU). IEC 60870-5-101 -protokollassa APCI ei kuitenkaan ole tarpeellinen, joten siinä APDU on yhtä kuin ASDU. (Cole 2002.)



KUVIO 2. IEC 60870-5-101 -viestin rakenne (Cole 2002.)

Kun APDU saapuu yhteyskerrokseen, yhteyskerros lisää siihen protokollan vaatiman oman ohjausdatan Link Protocol Control Informationin (LPCI). Näin muodostuu lopullinen Link Protocol Data Unit (LPDU) -kehys, joka siirretään fyysisen kerroksen kautta vastaanottavaan asemaan. LPDU-kehys tarjoaa erittäin korkealuokkaisen dataeheyden (IEC eheysluokka 12), ja vastaanotetussa kehyksessä pitääkin olla neljä virheellistä bittiä, ennen kuin havaittavissa oleva kehysvirhe ilmenee. (Cole 2002.)

LPDU-kehys koostuu kahdesta osasta, otsikosta ja rungosta, ja otsikossa määritellään rungon pituus. Otsikko sisältää merkit S+L+L+S jolloin runko sisältää loput, eli C+A+ASDU+CS+E. (Näiden merkkien tarkoitus ja toiminnat on selitetty alkaen sivulta 10.) Merkkien välillä ei saa olla juuri minkäänlaista väliä, korkeintaan yhden bitin siirtoon kuluva aika. Jokaista kehystä edeltää ja myös seuraa tietty odotusaika, mikä ilmenee jatkuvana 1-tason bittivirtana. Jos minkä tahansa asynkronisen merkin aloitusbitti otetaan virheellisesti vastaan 1-tason bittinä, merkin aloitusta siirretään niin kauan, että ensimmäinen 0-tason bitti saadaan. Tämä viivästyttää

kyseisen merkin aloitusta ja samalla vääristää sitä. Koska merkkien välissä ei ole juuri tyhjää, myös seuraavat merkit viivästyvät ja vääristyvät. Jos tällainen virhe pääsee tapahtumaan vastaanotetun kehyksen rungossa, lopetusmerkkiä (merkki E) työnnetään eteenpäin eli siirretään tarpeeksi kehyksen jälkeiselle odotusajalle. Vastaanottaja huomaa tämän suurella varmuudella, koska päätösmerkin bittikuvio on erityisesti valittu eroamaan odotusajan bittivirrasta. (Cole 2002.)

Kehyksen otsikolla on kiinteä muoto ja pituus, ja se on erikseen suojattu merkkien viivästymisiltä tai data- ja pariteetti-bittivirheiltä. Tämän ansiosta vastaanottaja voi luottaa kehyksen pituusmerkkiin (merkki L), ja se tietää tarkasti, mistä kohtaa etsiä päätösmerkkiä, kun se tarkistaa kehyksen yhtenäisyyttä. (Cole 2002.)

3.2.1 LPCI-kehys

LPCI-kehys sisältää useita merkkejä, jotka esitellään seuraavaksi.

LPCI = S+L+L+S+C+A+CS+E

S = aloitusmerkki, jolla on kiinteästi määritelty bittikuvio

L = pituusmerkki, joka kertoo C+A+ASDUn pituuden tavuissa. Pituusmerkki toistetaan kahdesti, ja molempien merkkien pitää olla täsmälleen samanlaiset, että LPCI-kehys hyväksytään.

C = yhteyden ohjausmerkki, jolla on keskeinen rooli siirtomenettelyssä. Ohjausmerkki sisältää

- LPCI-kehyksen funktion eli kertoo, lähettääkö se sovellusdataa, jolle halutaan vahvistus vastaanottajalta (Send/Confirm) jne. vai onko kyseessä esimerkiksi yhteyden nollausviesti tai yhteyden tilakysely
- primaaribitin, joka kertoo, onko LPCI-kehys yhteyden avaava vai vastaako se pyydettyyn kyselyyn
- laskentabitin, joka ilmoittaa mahdollisista bittihäviöistä tai päällekkäisyyksistä, kun viestin suunta on isäntäasemasta orja-asemaan

– ylivuotobitin, jonka orja-asema asettaa päälle, jos datan lähettäminen aiheuttaa ylivuodon sen puskurissa
– datan prioriteettibitin, jolla kerrotaan, onko data luokkaa 1 vai 2. Ykkösluokan datalla on korkeampi prioriteetti, kuin kakkosluokan datalla.

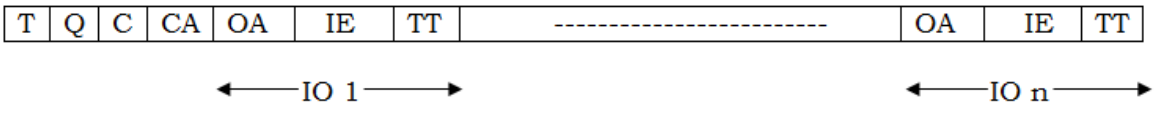
A = osoitekenttä, joka on 1–2 tavua pitkä. Kun viesti lähetään isäntäasemasta orja-asemaan, osoitekenttä sisältää sen orja-aseman osoitteen, jolle viestiä ollaan lähettämässä. Kun taas orja-asema lähettää viestiä isäntäasemalle, osoitekenttä sisältää lähettävän orja-aseman osoitteen niin, että isäntäasema voi yksilöidä, miltä asemalta kyseinen viesti on peräisin.

CS = tarkistusmerkki, joka koskee dataa vain C+A+ASDU-alueelta.

E = päätösmerkki, jolla on kiinteästi määritelty bittikuvio. (Reyenders ym. 2005.)

3.2.2 ASDU

Jokainen LPDU-kehys sisältää vain yhden ASDUn, mikä rajoittaa ASDUn koon 255 tavuun. ASDU koostuu kahdesta osasta, datayksiköiden tunnisteista ja itse datasta, joka taas koostuu yhdestä tai useammasta tietoyksiköstä. Datayksikön tunnisteet määrittelevät datan tyypin sekä datan osoitteen ja antavat lisäinformaatiota siirron syistä. Vaikka IEC 60870-5-101 tarjoaa erilaisia valmiiksi määriteltyjä ASDUja, niiden yleisrakenne on silti samanlainen. (Reyenders ym. 2005.) Kuviossa 3 esitetään ASDUn rakenne.

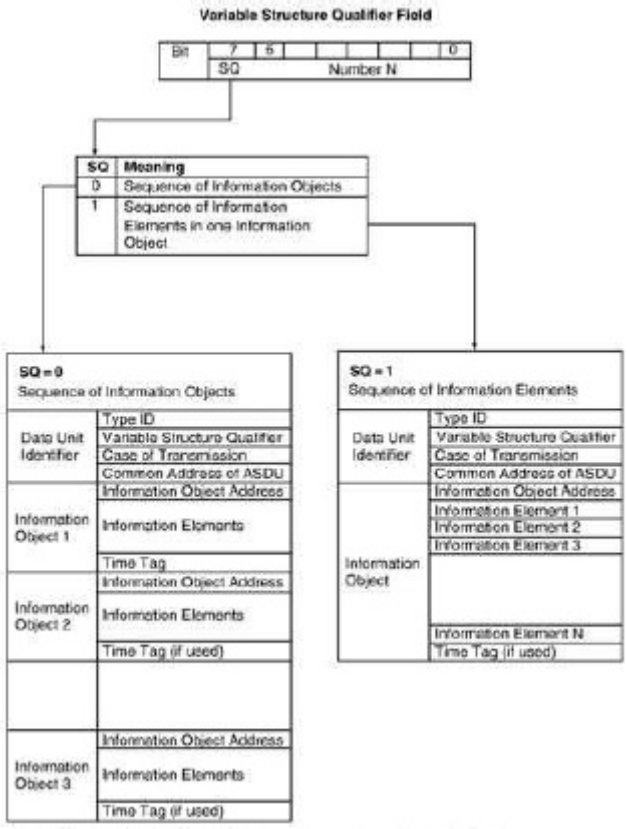


KUVIO 3. ASDU:n rakenne (Cole 2002.)

T (= Tyypin tunniste) on yhden tavun kokoinen, ja se kertoo, mitä kyseinen ASDU tekee ja minkälaiselle datalle se on määritelty. Tyyppi on etumerkitön kokonaisluku välillä 1–255, kun 0 ei ole käytössä. Valmiiksi määritellyt ASDU-tyypit löytyvät välitä 1–126, ja tyypit 128–255 on jätetty laitetoimittajien erikoiskäyttöön. On tärkeää

muistaa, että tyyppimäärittely vaikuttaa koko ASDUun. Joten jos ASDU sisältää useamman tietoyksikön, ne ovat kaikki samantyyppisiä, esimerkiksi yksittäisen bitin lukeminen. Tyyppi myös määrittelee, onko kyseisen ASDUn tietoyksiköillä aikaleimaa, koska valittavia tyyppejä ovat esimerkiksi skaalattu mittausarvo ja skaalattu mittausarvo aikaleimalla. (Reyenders ym. 2005.)

Q (= Muuttujien rakenne) -kenttä on yhden tavun kokoinen, ja se kertoo tietoyksiköiden tai tietoelementtien määrän ja sen, miten niihin osoitetaan. Kenttä sisältää seitsemänbittisen kokonaisluvun lisäksi yhden SQ (Structure Qualifier) -bitin, jolla kerrotaan, kumpi kahdesta mahdollisesta tietorakenteesta on käytössä. Jos SQ-bitti on 0, rakenne on tietoyksiköiden jono, jolloin jokaisella tietoyksiköllä on oma aikaleima ja osoite. Tietoyksiköiden määrä kerrotaan seitsemänbittisellä luvulla, joten yhdessä ASDUssa voi olla korkeintaan 128 tietoyksikköä. Mikäli SQ-bitti on 1, rakenne käsittää vain yhden tietoyksikön. Se taas voi sisältää useita tietoelementtejä, jotka ovat samaa tyyppiä, esimerkiksi skaalattu mittausarvo. Tällöin käytössä on vain yksi tietoyksikön osoite ja vain yksi aikaleima. (Reyenders ym. 2005.) Kuviossa 4 esitetään muuttujien mahdolliset rakenteet.



KUVIO 4. Muuttujien rakenne ASDUssa (Reyenders ym. 2005.)

C (= Siirron syy) -kenttää käytetään viestien reitittämiseen tietoverkoissa ja asemien sisällä ohjaamalla ASDU oikean ohjelman tai tehtävän käsittelyyn. Kenttä on 1–2 tavun kokoinen. Ensimmäinen tavu koostuu kuusibittisestä Cause Of Transmission (COT) -koodista, yhdestä Positive/Negative (PN) -bitistä ja yhdestä testibitistä. Toisen tavun tarpeellisuus riippuu käytettävästä järjestelmästä. Jos järjestelmässä on useampi ohjauskäskyjä antava asema, tavussa kerrotaan käskyn antaneen aseman osoite. COT-koodin tarkoituksena on tulkita kohdeasemalle viestin tietoja. Jokaiselle valmiiksi määritellylle ASDUlle on määritelty myös COT-koodi, ja niitä ovat esimerkiksi syklinen/spontaani siirto, yhteyden avaus, pyyntö tai pyyntöön vastaaminen, aktivointi/deaktivointi jne. PN-bitti toimii kuittausbittinä ohjauskäskyjen yhteydessä. Se näyttää valvontasuuntaan, onko annettu ohjauskäsky suoritettu vai ei. Jos PN-bitti ei ole oleellinen ohjauskäskyn yhteydessä, se laitetaan nollaksi. Testibitti on 1 vain silloin, kun ASDU on määritelty testikäyttöön. Sitä käytetään vain laitteiston ja viestinnän testaamisessa, ei mihinkään muuhun. (Reyenders ym. 2005.)

CA (= Yleinen osoite) -kenttä on 1–2 tavun kokoinen. Siitä käytetään nimitystä yleinen osoite, koska sen määrittelemä osoite on yhteinen kaikelle datalle, mitä ASDU sisältää. Ohjaussuuntaan osoite kertoo sen aseman, mihin kyseinen ASDU on menossa, joten osoitetta 0 ei käytetä. Valvontasuuntaan yleinen osoite taas kertoo datan lähettävän aseman osoitteen, niin että isäntäasema osaa yksilöidä saapuvan datan ja tallentaa sen oikeaan paikkaan järjestelmässä. Jos CA-kentässä on korkein mahdollinen lukema, eli 0xFF (255) tai 0xFFFF (65535), ASDU on globaali ja se esitetään järjestelmän jokaiselle asemalle. Niiden käyttö on kuitenkin rajoitettu vain muutaman ASDUn tasolle, ja ne ovat

- kaikkien asemien kysely -komento (ASDU-tyyppi 100)
- laskurien kysely -komento (101)
- kellon synkronointi -komento (103)
- prosessin nollaus -komento (105). (Reyenders ym. 2005.)

OA (= Tietoyksikön osoitekenttä) on ensimmäinen osa tietoyksikköä, ja sen avulla tunnistetaan data yksityiskohtaisesti määrätystä asemasta. Kenttä voi olla yhdestä kolmeen tavun kokoinen, mutta kolmen tavun osoitekenttää käytetään vain, kun

tehdään struktuurisia osoitejärjestelmiä. Järjestelmätasolla tietty data voidaan uniikisti tunnistaa yleisen osoitteen ja tietoyksikön osoitteen kombinaatiolla (CA+OA). (Reyenders ym. 2005.)

IE (= Tietoelementti) mikä on osa tietoyksikköä. Kuten jo aikaisemmin kävi ilmi, yhdessä ASDUssa voi olla useita tietoyksiköitä, jotka sisältävät tietoelementin. Vaihtoehtoisesti ASDUssa voi olla vain yksi tietoyksikkö, joka sisältää useita identtisiä tietoelementtejä. Olipa rakenne mikä tahansa, tietoelementit ovat keskeisiä komponentteja tiedonvälityksessä sisältäen ”varsinaisen” datan, ja niitä on käytetty rakennuspalikkoina, kun ASDU on määritelty protokollaan. (Reyenders ym. 2005.)

TT (= Tietoyksikön aikaleima) on käytössä, jos ASDUn tyypiksi on valittu aikaleiman sisältävä tyyppi (Reyenders ym. 2005).

4 OHJELMOITAVA LOGIIKKA

Programmable Logic Controller (PLC) eli ohjelmoitava logiikka, on pieni mikroprosessorilla varustettu tietokone, jota arkikielessä kutsutaan monesti pelkäksi logiikaksi. Logiikoita käytetään automaatioprosesseissa, kuten esimerkiksi koneiden tai erilaisten tuotantolinjojen ohjauksessa. Ennen logiikoiden kehittämistä vastaavat ohjaukset jouduttiin toteuttamaan sadoilla tai tuhansilla releillä ja ajastimilla. Nykyään tuo määrä pystytään korvaamaan yhdellä ainoalla ohjelmoitavalla logiikalla. (Keinänen, Kärkkäinen, Lähetkangas & Sumujärvi 2007, 212.)

Kaikki prosessin tai järjestelmän tilaa havainnoivat anturit ja kytkimet kytketään ohjelmoitavan logiikan tuloihin. Ne voivat olla joko binäärisiä (päällä/pois) tuloja tai analogisia tuloja (esim. mittauksia). Logiikan lähtöihin kytketään releiden ja kontaktorien avulla kaikki toimilaitteet, esimerkiksi venttiilit, moottorit ja merkkilamput, joita logiikka ohjaa siihen ladatun ohjelmakoodin perusteella. Logiikka suorittaa ohjelmaa ns. kiertävästi. Aluksi se lukee kaikkien tulojen tilat ja kirjoittaa ne muistiin. Tämän jälkeen se etenee ohjelman läpi rivi kerrallaan ja käsittelee ja toteuttaa tulokset siinä järjestyksessä, kun ohjelmaa luetaan. Kierron lopuksi logiikka kirjoittaa lähtöjen tilat ja palaa taas alkuun. Ohjelmointikieliä ovat Statement List (STL) eli käskylista, Ladder Diagram (LAD) eli ns. tikapuukaavio ja Function Block Diagram (FBD) eli toimintalohko-ohjelmointi. Ennen kuin ohjelma ladataan logiikkaan, sille suoritetaan compile-toiminto eli se käännetään konekielelle, niin että logiikka ymmärtää sen. (Keinänen ym. 2007, 223–225.)

Monessa automaatiototeutuksessa toimilaitteet sijaitsevat kaukana itse prosessiasemasta. Silloin on järkevin viedä tulo- ja lähtöpiirit lähemmäksi toimilaitteiden luo ja säästää kaapelointikustannuksissa. Tällöin puhutaan hajautetusta I/O:sta. Se sisältää itse hajautusaseman, johon lisätään käyttötarpeiden mukaisesti lisämoduuleita. Ne ovat yleensä tehonsyöttökortteja, digitaalisia tai analogisia I/O-kortteja, erilaisia sarjaliikennekortteja jne. Hajautusasema voi sisältää myös itse prosessiaseman, eli logiikkaohjaimen. Sellainen on esimerkiksi Siemensin ET200S-sarjan logiikka IM151-8 PN/DP CPU. Logiikan tyyppinimessä IM tulee

sanoista Interface Module, ja PN/DP tarkoittaa, että logiikka tukee liityntöjä Profinet ja Profibus DP kenttäväylillä. (Siemens AG 2012a.)

5 YRITYKSEN ESITTELY

Kokkolalainen Apex Automation Oy on vuonna 1993 perustettu insinööritoimisto. Se on erikoistunut automaatio- ja sähkösuunnitteluun ja on kasvanut merkittäväksi kokonaistoimituksiin kykeneväksi toimijaksi alallaan. Nykyään Apex Automation Oy:llä on noin 50 työntekijää. Vaikka Apex Automation Oy:n päätoimipaikka sijaitsee Kokkolassa, sen toiminta-alueeseen kuuluu koko Suomen lisäksi myös ulkomaat. (Apex Automation Oy 2011.)

Apex Automation Oy tarjoaa palveluitaan laajasti energian tuotanto- ja jakeluyhtiöille ja alan järjestelmätoimittajille, teollisuuden laite- ja järjestelmätoimittajille, prosessiteollisuudelle, sähköasennusliikkeille sekä liike- ja julkiskiinteistöjen rakennuttajille. Palvelut ovat saatavilla joko ”Avaimet käteen” -periaatteella tai pienempinä osatoimituksina. Yrityksen keskeisimpiä palveluita ovat muun muassa ohjelmoitavien logiikoiden, prosessiautomaatiojärjestelmien ja valvomoiden sovellusohjelmointi, teollisuus- ja rakennussähköistysten suunnittelu ja asennusvalvonta, instrumentointisuunnittelu sekä keskusvalmistus. Myös kone- ja sähköturvallisuuspalvelut ovat Apex Automation Oy:n vahvoja alueilta, ja lisäksi yritys tarjoaa myös alan koulutus- ja konsultointipalveluita. (Apex Automation Oy 2011.)

6 KÄYTETTY LAITTEISTO

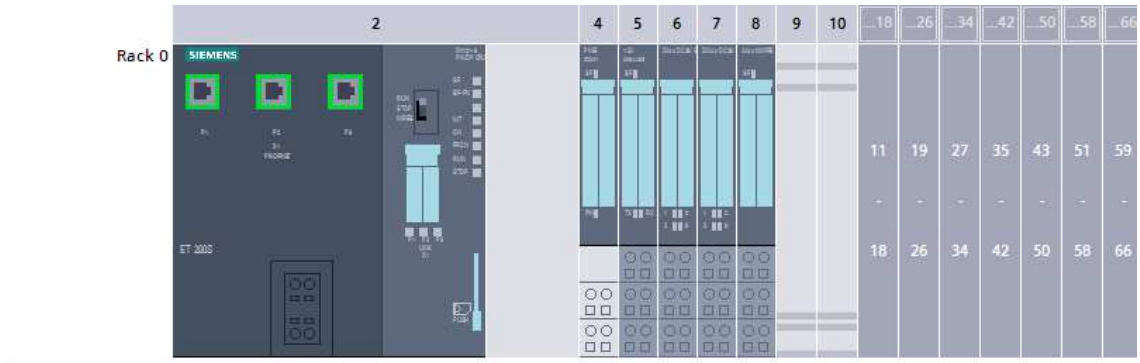
Kun tässä luvussa puhutaan logiikasta, tarkoitetaan nimenomaan työssä käytettyä logiikkaa (IM151-8 PN/DP CPU) eikä logiikoita yleisesti. Käytetty logiikka on siis Siemensin S7-perheestä ET200S-sarjalainen (tuotenumero 6ES7 151-8AB01-0AB0) ja perustuu ”isoveljeensä” S7-perheen 300-sarjan logiikkaan. Sen firmware-versio on tällä hetkellä uusin eli V3.2. Myöhemmin firmware on mahdollista päivittää uudempaan siihen tarkoitetulla muistikortilla. Logiikan muistialue rakentuu kolmesta eri muistista, ja näitä ovat latausmuisti (load memory), järjestelmämuisti (system memory) ja työmuisti (work memory).

Latausmuisti on yhtä kuin muistikortti, ja logiikka ei toimi ilman, että siihen on asennettu muistikortti. Tällöin latausmuistin koko on myös suoraan verrannollinen muistikortin kokoon, ja Siemensin suositus tälle logiikalle on 128KB:n kortti tai vastaavasti 512KB:n kortti, kun logiikkaan tehdään web-serveri. Tarvittaessa logiikassa voidaan käyttää myös kapasiteetiltaan suurempia muistikortteja, kuten 2 MB, 4 MB tai 8 MB. Latausmuistiin tallennetaan kaikki koodilohkot, datablokit ja järjestelmän data, joka sisältää laitteistomäärittelyn, liitynnät ja lisämoduulien parametrit. (Siemens AG 2010; Siemens AG 2013.)

Järjestelmämuisti on integroitu logiikkaan, eikä sitä voi laajentaa. Se sisältää muistibittien, ajastimien ja laskureiden osoitealueet, ja sitä käytetään myös väliaikaisten temp-muuttujien tallennuspaikkana. Lisäksi järjestelmämuistissa on I/O-alueen prosessikuva, joka on oletusasetuksena 128 tavua molemmille, tuloille ja lähdöille. Myös työmuisti on integroitu logiikkaan, eikä se näin ollen ole laajennettavissa. Työmuistia käytetään logiikkakoodin ajamiseen sekä käsittelemään ohjelman dataa. Ohjelma pyörii ainoastaan työ- ja järjestelmämuistissa. Logiikan työmuistin koko on 192 KB, mistä 64 KB on retentiivista muistia. Jos logiikassa tapahtuu sähkökatkos tai sille tehdään uudelleen käynnistys, retentiiviseen muistiin tallennetut tiedot säilyvät. Ohjelmassa käytetyt datablokit voidaan määrittää joko retentiivisiksi tai ei-retentiivisiksi. Sähkökatkon jälkeen logiikka lataa retentiivisten datablokkien viimeisiksi jääneet arvot retentiiviseltä muistialueelta. Jos taas datablokkia ei ole

määritelty retentiiviseksi, käynnistyksen yhteydessä logiikka alustaa datablokit ja asettaa muuttujat määriteltyihin alkuarvoihin. (Siemens AG 2010.)

Logiikka on laajennettavissa kaikkiaan 64 moduulin kokoiseksi paketiksi. Tässä työssä minulla oli logiikkaan liitettynä kaikkiaan viisi eri korttia: sarjaliikennekortti, 4-kanavainen digitaalinen tulokortti, 4-kanavainen digitaalinen lähtökortti ja 4-kanavainen analoginen tulokortti. Tämä kokoonpano pitää määrittää myös itse logiikkaan. Tällöin ohjelmointityökalulla tehdään laitteistomäärittely eli Hardware (HW) -konfiguraatio, joka ladataan logiikkaan. HW-konfiguraatiossa kokoonpano tehdään samalla periaatteella kuin fyysisestikin, eli valitaan listasta oikea logiikka ja oikeat kortit. Korttien järjestyksen pitää olla täsmälleen sama kuin fyysisessä kokoonpanossa, tai muuten logiikka menee virhetilaan. HW-konfiguraatiossa tehdään kaikki logiikan asetukset ja siinä määritetään myös I/O- ja muistialueet halutuiksi. Kuviossa 5 näkyy testiohjelman HW-konfiguraatio.



Device overview

Module	Rack	Slot	I address	Q address	Type	Order no.	Firmware
IM151-8 PN/DP CPU	0	2			IM 151-8 PN/DP CPU	6ES7 151-8AB01-0AB0	V3.2
PROFINET interface_1	0	2 X1	2047*		PROFINET interface		
	0	2 X2					
PM-E 24VDC_1	0	4	2043*		PM-E 24VDC	6ES7 138-4CA01-0AA0	
1 SI_1	0	5	272...275	272...275	1 SI	6ES7 138-4DF01-0AB0	V1.3
4DI x 24VDC ST_1	0	6	0.0...0.3		4DI x 24VDC ST	6ES7 131-4BD01-0AA0	
4DO x 24VDC / 0.5A ST_1	0	7		0.0...0.3	4DO x 24VDC / 0.5...	6ES7 132-4BD02-0AA0	
4AI x I 2WIRE ST_1	0	8	100...107		4AI x I 2WIRE ST	6ES7 134-4GD00-0AB0	

KUVIO 5. Testiohjelman HW-konfiguraatio

7 RAJAPINNAN LUONTI

Tässä luvussa esiteltävä rajapinnan luominen tarkoittaa sitä ohjelmointirajapintaa, minkä avulla logiikka ja SCADA keskusteleva keskenään. Tämän rajapinnan kautta liikkuu kaikki järjestelmien välinen data. Periaatteena on, että SCADA-ohjelman tekijän ei tarvitse tietää rajapinnan alapuolisesta logiikkaohjelmasta ja sen toiminnasta. Hänelle riittää vain tieto määritellyistä prosessipisteistä, joista SCADA lukee ja joihin se kirjoittaa, ja siitä minkä tyyppisiä nämä pisteet ovat. Tämän työn rajapintaan on määritelty yleisimmin käytettävät toiminnot, kuten binaaritietojen (tilatietojen) ja mittausarvojen lukeminen sekä binaariohjausten ja parametrien kirjoittaminen. Lisäksi IEC-protokolliin oleellisesti liittyvät toiminnot, esimerkiksi ajan synkronointi ja ala-asemien yleinen kysely, ovat tuettuina. Niinpä tässä rajapinnan luomisessa keskitytään vain niihin ohjelmalohkoihin, joita edellä mainitut toiminnot toimiakseen vaativat. On myös syytä mainita heti aluksi, että tässä työssä logiikka laitettiin orja-asemaksi.

Työssä käytetty Siemensin Siplus RIC IEC -paketti sisältää itse ohjelmoitavan logiikan (151-8 PN/DP CPU), 1 Serial Interface (1SI) -sarjaliikennekortin ja 128 KB:n IEC 60870-5-101 Master/Slave -muistikortin, joka toimii samalla lisenssinä kyseiselle paketille. Lisäksi mukana tulee CD-levy, joka sisältää Siemensin ohjelmointikirjastot IEC-60870-5-101:lle, -103:lle ja -104:lle. Näistä kirjastoista löytyvät tarvittavat ohjelmalohkot rajapinnan luomiseen, haluttiinpa logiikka asettaa sitten isäntäasemaksi tai orja-asemaksi. Kirjastot sisältävät niin datablokkeja kuin toimilohkojakin.

Toimilohkoja ovat Function Blockit (FB:t) ja Functionsit (FC:t). FB:t sisältävät oman muistin, eli niihin liitetään aina oma instanssi datablokki, johon toimilohko tallentaa tietoja. FC:llä ei ole omaa muistia, ja kun logiikkaohjelma suorittaa niiden koodia, se tallentaa tiedot väliaikaisiin muistipaikkoihin. Kun FC on suoritettu, nämä väliaikaisiin muistipaikkoihin tallennetut tiedot menetetään. Datablokit (Data Block, DB) puolestaan voivat olla joko globaaleja tai instansseja. Globaali datablokki toimii yleisenä tietovarastona, josta mikä tahansa toimilohko voi lukea tietoja tai vastaavasti kirjoittaa sinne. Instanssi datablokki on puolestaan FB:n ”henkilökohtainen”

tallennuspaikka, ja aina kun FB:tä kutsutaan ohjelmakierrossa, sille pitää määrittää oma instanssi datablokki.

Kaikki Siemensin kirjaston ohjelmalohkot ovat suojattuja (Know-How-Protected), eli niiden lähdekoodia ei pääse tarkastelemaan. Jos käyttäjä avaa lohkon, se saa vain ilmoituksen lohkon suojauksesta. Tällä estetään se, ettei käyttäjä pääse tekemään vahinkoa menemällä muuttelemaan toimivia lohkoja. Lisäksi suojauksella pidetään omat ohjelmointiniksit ja -keksinnöt visusti omassa tiedossa. Ohjelmalohkot on suunniteltu Siemensin Simatic Manager Step 7 -ohjelmointityökalulle, eikä niitä ainakaan vielä ole saatavilla suoraan yhtiön uuden sukupolven ohjelmointityökalulle TIA-portaalille.

Ohjelmalohkot on kuitenkin mahdollista kääntää TIA:lle ja saada ne toimimaan. Yksi keino on aloittaa projekti Simatic Managerilla, johon kopioi ohjelmalohkot kirjastosta. Kun lohkot on kopioitu, suoritetaan compile-toiminto eli käännetään ohjelma konekielelle. Tallennetaan projekti ja avataan se seuraavaksi TIA-portaalilla. TIA:ssa suoritetaan migrate-toiminto, joka kääntää ohjelman TIA:lle sopivaksi. Tämä opinnäytetyö tehtiin aluksi Simatic Managerilla, ja testausten jälkeen käänsin ohjelmalohkot myös TIA:lle. Lohkot toimivat moitteettomasti TIA:lla, ja myös sillä sai luotua toimivan rajapinnan.

7.1 IEC-yhteyslohko, S7_IEC_Config

Ohjelmointikirjastoista löytyvä S7_IEC_Config-lohko on koko toiminnan tärkein palanen. Se luo ns. yhteystunnelin ja pitää huolta yhteydestä, jonka avulla sovelluslohkot voidaan yhdistää prosessin valvontaan ja ohjaukseen. Siemensin kirjastossa S7_IEC_Config-lohkon FB-numeroksi on annettu FB100 ja tästä eteenpäin siitä käytetään lyhyempää nimitystä FB100-lohko. Siemensin suositus on, että lohkokirjastossa olevien lohkojen numeroita ei mentäisi muuttamaan, joten pidin kaikki lohkot vakionumeroissaan. FB100-lohkossa määritellään IEC 60870-5-101 -protokollan fyysisen kerroksen ja yhteyskerroksen parametreja. Kuviossa 6 esitetään FB100-lohko ja kaikki sen muuttujat.

```
CALL  "S7_IEC_Config" , "S7_IEC"            FB100 / DB100
 Registration_Code       :=DW#16#[redacted]
 Line_ID                 :=1
 L1_Laddr_HW             :=272
 L1_Data_Wait_Time       :=T#10MS
 L1_Time_RTS_OFF         :=T#10MS
 L2_Length_Link_Address  :=1
 L2_Link_Address         :=L#10
 L2_TimeOut_SendConfirm  :=T#2S
 L2_TimeOut_RcvInfo      :=T#30S
 L2_Repeats_on_Timeout   :=2
 L2_Balanced_mode        :=FALSE
 L2_Dir_Bit              :=FALSE
 L2_E5_as_ACK_NACK       :=TRUE
 L2_UnbalSlave_CL2_NACK  :=FALSE
 L7_Length_ASDU_Address  :=1
 L7_Length_Info_Addresses:=2
 L7_With_Originator      :=FALSE
 L7_Block_Len            :=120
 Buffer_Handling         :=B#16#0
 First_internal_DB_No    :=50
 Do_Restart              :=FALSE
 No_of_Send_Buffers      :=2
 Send_Buffer_Dim         :=L#4096
 Cascade_P_Application   :=
 Cascade_Mode            :=B#16#0
 P_Application           :="P_App"           MD10
 L2_Error_Link           :=
 C_isActive              :=
 FB_RetVal               :="FB100_RetVal"    MW14
 Buffer_Info_lost        :=
 Time_DS                 :=
 Time_IV                 :=
 Time_SY                 :=
 Time_Diff               :=
NOP   0
```

KUVIO 6. FB100-lohko STL-ohjelmointikielellä

Ohjelmointikirjastossa tulee mukana DB100, jota voi käyttää FB100:n instanssidatablokkina. DB100:ssa on annettu kaikille muuttujille oletusarvot, ja näistä vielä suurin osa soveltuu sellaisenaan suoraan käyttöön. Koska useampaa muuttujaa voidaan käyttää alkuperäisillä määrityksillä, keskitytään näissä toimilohkojen esittelyissä vain lohkojen tärkeimpiin muuttujiin, jotka ovat toiminnan ja käyttäjän kannalta välttämättömiä.

Kaupallisista syistä FB100-lohkolle täytyy antaa kahdeksanmerkkinen rekisteröintikoodi. Tämä koodi toimitetaan muistikortin mukana, ja se löytyy muistikortin pakettiin liimatusta tarrasta. Ehkä hieman hämäävästi tarrassa lukee myös ”Serial number(s)” ja perässä kahdeksanmerkkinen sarjanumero. Tämä ei kuitenkaan ole haluttu rekisteröintikoodi, vaan samasta tarrasta löytyy kaksi muutakin merkkisarjaa, joista toinen on orja-asemalle ja toinen isäntäasemalle. Näistä valitaan sen

mukaisesti, kumpi kyseisestä asemasta halutaan tehdä. Mikäli rekisteröintikoodi ei täsmää mukana toimitettuun koodiin, on käytössä vain 15 minuutin testaustila. Kun aika on täyttynyt, kommunikointi voi toimia enää vain oikealla rekisteröintikoodilla.

Laddr_Hw-muuttujaan laitetaan sarjaliikennekortin input-osoite. Tämän löytää HW-konfiguraatiosta, mistä sen pystyy myös itse määrittelemään. Tässä työssä käytettiin alkuperäistä osoitetta 272, jonka HW-konfiguraatio kortille antoi. Link_Address- ja Lenght_Link_Address-muuttujilla määritellään logiikan eli aseman osoite. Tästä osoitteesta käytetään monesti nimitystä asemanumero. Lenght_Link_Address-muuttuja kertoo tavuina, millä välillä tuo aseman osoite voi olla. Jos muuttujan arvo on 1, kuten kuviossa 6, aseman osoitteen voi antaa väliltä 1–254. Mikäli muuttuja arvo taas olisi 2, aseman osoitealue olisi 1–65534. Balanced_Mode-muuttujalla määritetään yhteyskerroksen toimintatila eli valitaan, onko se balansoimaton, jolloin muuttujalle annetaan arvo 0. Jos tila on balansoitu, muuttuja saa arvon 1. Balansoimattomassa tilassa vain isäntäasema voi avata yhteyden, mutta balansoidussa sen voi tehdä niin isäntä- kuin orja-asemakin. Oletusasetuksena yhteyskerros asetetaan balansoimattomaan tilaan ja se myös pidettiin.

Lenght_Asdu_Address-muuttujalla annetaan ASDUn yleisen osoitekentän koko, joka on 1–2 tavua. Lenght_Info_Addresses-muuttujalla puolestaan määritellään IEC:n tietoyksiköiden osoitealue. Muuttujan arvo annetaan tavuina, ja se voi olla 1–3 tavua. Työssä pidin kyseisen arvon vakiona (2), jolloin tietoyksiköiden osoitealueeksi tulee 1–65535. Eri tietotyyppien osoitteet jaoin taulukon 1 mukaisesti.

TAULUKKO 1. IEC:n tietoyksiköiden osoitealueet testiohjelmassa

DATA TYPE	IEC INFORMATION OBJECT ADDRESS
Tilatiedot (Single Point Information)	1–999
Mittaukset (Measured Values)	1000–1999
Binaari ohjaukset (Single Commands)	2000–2999
Parametrit (Setpoints)	3000->

Kun FB100-lohko ja sen instanssi datablokki on ladattu logiikkaan ja logiikka asetetaan RUN-tilaan, IEC-applikaatio luo logiikan Random Access Memory (RAM)

-muistiin datablokkeja, joiden määrä riippuu FB100:n asetuksista. Oletusasetuksilla näitä datablokkeja luodaan neljä kappaletta, ja ne ovat

- Diagnostic-DB
- Quit Buffer -DB
- Send Buffer prior 0 -DB
- Send Buffer prior 1 -DB.

FB100:n muuttujalla First_Internal_Db_No määritetään ensimmäisen RAM-muistiin luotavan datablokin numero. Oletusarvona se on 50, ja tämän muuttujan kanssa pitää olla tarkkana, ettei tule päällekkäisyyksiä logiikkaohjelmassa käytettävien datablokkien ja IEC:n logiikkaan luomien datablokkien kanssa. Nämä IEC:n luomat datablokit näkyvät vain, kun ohjelmointityökalulla mennään online-tilaan eli avataan yhteys logiikkaan, jolloin nähdään logiikan muistin sisältämät lohkot ja datablokit. Diagnostic- ja Quit Buffer -datablokit luodaan aina ja oletusasetuksilla Send Buffer -datablokkeja luodaan kahden korkeimman prioriteetin viesteille. No_of_Send_Buffers-muuttujalla voidaan määrittää, mille viesti prioriteeteille datablokit luodaan alueen ollessa 1–16. Näiden datablokkien koko puolestaan määritellään Send_Buffers_Dim-muuttujalla oletusarvon ollessa 4096KB.

On huomattava, että IEC:n logiikkaan luomat datablokit ovat retentiivisiä, eli ne varaavat kokonsa verran retentiivistä muistia. Retentiivinen muistialue ei monessa logiikassa kuitenkaan ole järin suuri, joten sen saa helposti täytettyä, jos Send Buffer datablokkeja määrittää monelle viestiprioriteetille. Lisäksi kaikki IEC-lohkojen instanssidatablokit luodaan oletuksena retentiivisiksi ja yleensä myös varsinaiseen logiikkaohjelmaan tarvitaan retentiivistä muistia. Niinpä logiikka täytyy mitoittaa sen mukaisesti, että muistialueet eivät täyty, vaikka kyseessä olisi suurempikin projekti.

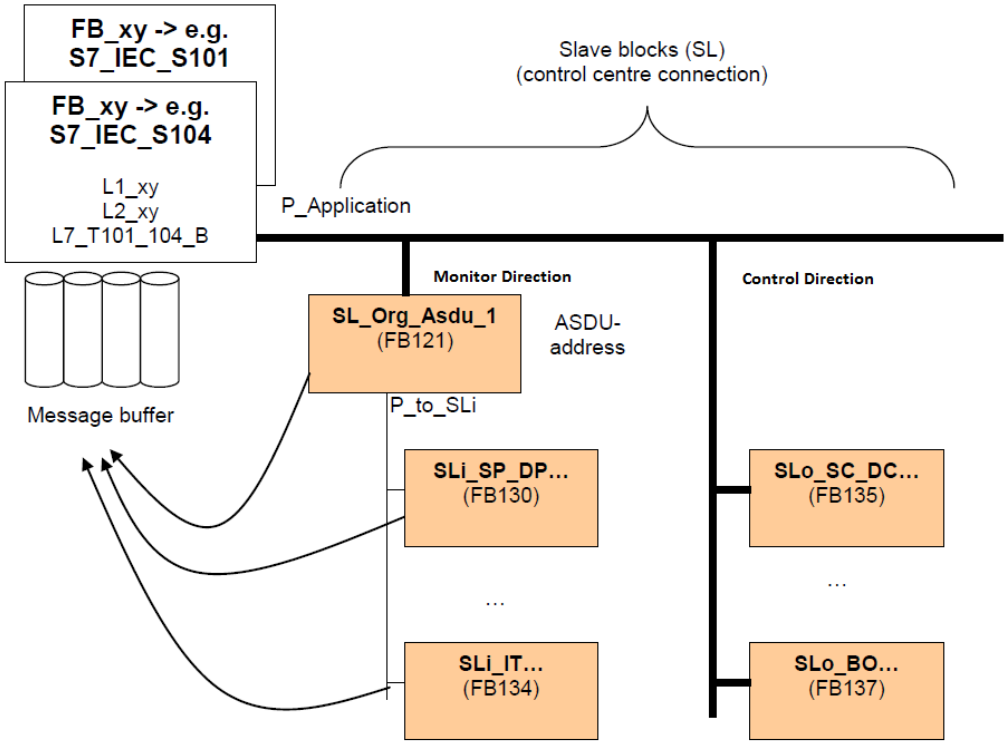
Toinen asia, johon on tärkeää kiinnittää huomiota, on FB100-lohkon instanssidatablokin lataaminen logiikkaan. Kuten edellä kerrottiin, IEC-applikaatio luo logiikkaan ensimmäisen käynnistyksen yhteydessä sisäisiä datablokkeja. Kun nämä datablokit on kerran luotu ja ladataan FB100:n instanssidatablokki uudelleen logiikkaan, IEC luo sisäiset datablokit uudestaan vanhojen lisäksi. Testien perusteel-

la sillä ei ole vaikutusta, vaikka valitsee FB100:n latausvaiheessa ”korvaa olemassa oleva datablokki”. Tämä oli suurin yksittäinen ongelma, joka aiheutti hankaluuksia, ennen kuin se selvisi. Ohjelmaa tehdessäni ja sitä testaillessani latasin tottuneesti aina koko ohjelman logiikkaan ja korvasin vanhan. Ihmetystä aiheutti, kun kohta oli logiikan retentiivinen muisti täynnä ja FB100-lohko antoi virheviestiä ”ongelma datablokkien luomisessa”. Eli sitä joutuu tarkkailemaan, ettei näitä sisäisiä datablokkeja luo turhaan useampaan kertaan ja täytä näin logiikan muistia.

P_Application-muuttujaan liitetään pointteri, millä FB100-lohko ja sovelluslohkot yhdistetään. Pointteri on tyypiltään tuplasana (DWORD) ja sen voi luoda esimerkiksi logiikan M-alueelle tai väliaikaisiin muuttujiin (temp-muuttujat). Itse loin kyseisen muuttujan M-alueelle osoitteeseen MD10, joten se varaa muistialueesta tavut 10–13, eli seuraava vapaa bitti on M14.0. Annoin pointterille nimeksi ”P_App”, että saa heti jonkinlaisen kuvan siitä, mikä se on. FB_RetVal-muuttuja kertoo lohkon tilasta ja mikäli sen arvo on nolla, asiat ovat hyvin ja lohko toimii ilman virheitä. Jos käytössä on muistikortin ja rekisteröintikoodin välisestä ristiriidasta aiheutuva testaustila, jäljellä ovela aika juoksee muuttujassa. Mikäli muuttujan arvo on W#16#8101 ja siitä eteenpäin, lohko on virhetilassa.

7.2 Sovelluslohkot

Kaikki sovelluslohkot kytketään IEC:n yhteyslohkoon (FB100-lohko) P_Application-pointterin kautta. Ero tulee siinä, että ohjauslohkot (SLo-alkuiset) kytketään suoraan, mutta tiedonkeruulohkot (SLi-alkuiset) kytketään SL_Org_Asdu_1-lohkon kautta. Yhteyslohkon pointteri tuodaan SL_Org_Asdu_1-lohkolle, josta puolestaan kytketään oma P_SLi-pointteri tiedonkeruulohkoille. Eli ASDUn osoitteen ohjaus valvontasuuntaan tapahtuu tätä kautta. Tiedonkeruulohkot kirjoittavat datansa yhteyslohkon luomiin viestipuskureihin, joissa data jaetaan oikeaan puskuriin sen prioriteetin perusteella. IEC:n lohkorakenne ja lohkojen yhdistäminen ilmenee kuviosta 7.



KUVIO 7. IEC:n lohkorakenne ja lohkojen yhdistäminen (mukaillen SIEMENS AG 2012b.)

7.2.1 Sovelluslohkojen symboliset nimet

Sovelluslohkojen symbolisista nimistä voi suoraan lukea lohkojen tärkeimmät tiedot, kuten niiden toiminnan, datatyypin, rakenteen ja koon. Tässä luvussa avataan tätä symbolista nimeämistä ja käytetään esimerkkinä tilatietojen siirtoon tarkoitettua lohkoa SLi_SP_DP_s128.

SLi_SP_DP_s128 SLi tulee sanoista Slave blocks input, eli toisin sanoen se tarkoittaa orja-aseman tiedonkeruulohkoa. Ohjauslohkot merkitään lyhenteellä SLo (Slave blocks output).

SLi_**SP**_**DP**_s128 SP on lyhennetty sanoista Single Point Information ja DP sanoista Douple Point Information. Nämä merkinnät kertovat, mitä datatyyppejä sovelluslohko tukee.

SLi_SP_DP_**s**128 s tarkoittaa ”peräkkäistä käsittelyä”, jolloin jokaiselle tietoyksikölle annetaan oma osoite (information object address) ja

osoitteiden jako aloitetaan määritellystä kohdasta (start address). Jos aloitusosoitteeksi on valittu esimerkiksi 100, se on ensimmäisen tietoyksikön osoite, 101 toisen, 102 kolmannen jne.

SLi_SP_DP_s**128** Luku 128 kertoo käsiteltävien tietoyksiköiden maksimimäärän. Mikäli maksimimäärä ei ole riittävästi, voidaan lohkoa kutsua ohjelmassa useamman kerran, jolloin sille luodaan aina uusi instanssidatablokki. Näin esimerkiksi kahdella lohkokutsulla päästäisiin 256:n tietoyksikköön. Ohjauslohkoissa maksimimäärää ei kerrota luvulla, vaan sen tilalla on pelkkä x. Ohjauslohko nimittäin itsessään ei rajoita tietoyksiköiden määrää, joten teoreettisesti se pystyisi käsittelemään niitä miten paljon tahansa. Käytännössä tietokohteiden määrän rajoittaa vapaana olevien osoitteiden määrä ja datablokin maksimikoko (logiikkakohtainen). (Siemens AG 2012b.)

7.2.2 ASDU-ohjauslohko, SL_Org_Asdu_1

SL_Org_Asdu_1-lohkon FB-numero kirjastossa on FB121, ja tästä eteenpäin siitä käytetään lyhyempää nimitystä FB121-lohko. ASDUt kulkevat tiedonkeruulohkoille FB121-lohkon kautta, jonka lisäksi lohkon tehtäviä ovat muun muassa rajapinnan muodostaminen tiedonkeruulohkoille, käynnistysviestin lähettäminen, yleisen asemakyselyn käsittely, testiviesteihin vastaaminen ja ajan synkronointi. Kuviossa 8 esitetään FB121-lohko ja kaikki sen muuttujan.

```
A     "Aina_1"                    M0.1
=     L      0.0
BLD   103
A     "Aina_0"                    M0.0
=     L      0.1
BLD   103
CALL  "SL_Org_Asdu_1" , DB121     FB121
 Comp_ID             :=B#16#1
 P_Application       :="P_App"     MD10
 ASDU_Adr            :=L#10
 Accept_ClockSync    :=L0.0
 Sim_GI              :=L0.1
 Set_Time_Correction:=T#0MS
 SF_Originator       :=B#16#0
 P_SLi               :="P_SLi"     MD16
 Reset               :=
 TimeSync            :=
 GI_Runs             :=
NOP   0
```

KUVIO 8. FB121-lohko STL-ohjelmointikielellä

Jokaiselle sovelluslohkolle pitää antaa yksilöllinen tunniste Comp_ID-muuttujaan. Tämä tunniste on diagnostiikka-datablokkia varten, jonka IEC-applikaatio luo logiikkaan. P_Application-muuttujaan tuodaan pointteri yhteyslohkolta (FB100-lohko) ja ASDU_Adr-muuttujaan annetaan ASDUn osoite, jonka lohko käsittelee. Jos, ja tässä tapauksessa kun, ajan synkronointi halutaan mahdolliseksi, pitää Accept_ClockSync-muuttuja asettaa 1-tilaan. Kun tämä on sallittu, isäntäasemalta eli tässä tapauksessa valvomokoneelta voidaan logiikan kellonaika asettaa samaksi kuin valvomokoneen aika. P_SLi-muuttujaan liitetään pointteri, jolla lohko yhdistetään tiedonkeruulohkoihin. Pointteri on kooltaan tuplasana (DWORD), ja sen voi luoda joko logiikan M-alueelle tai väliaikaisiin muuttujiin (temp-muuttujat). Loin pointterin M-alueelle osoitteeseen MD14, ja koska kyseessä on tuplasana, seuraava vapaa bitti on osoitteessa M18.0. Nimeksi pointterille annoin ”P_SLi”, että siitä näkee heti, mikä on sen tehtävä.

7.2.3 Tilatiedot, SLi_SP_DP_s128

SLi_SP_DP_s128-lohkon FB-numeroksi lohkokirjastossa on määritelty FB130 ja tästä eteenpäin siitä käytetään lyhyempää nimitystä FB130-lohko. FB130-lohko on tarkoitettu tilatietojen lukemiseen, ja sillä voi lukea tyypiltään yksi- tai kaksibittisiä

tietoyksiköitä. Tilatiedoilla saadaan tarpeellista informaatiota prosessin tilasta, esimerkiksi hälytykset, anturitiedot jne. Yhdellä FB130-lohkon instanssidatablokilla päästään maksimissaan 128:n tietoyksikköön. Määrää voi kasvattaa kutsumalla lohkoa ohjelmassa useamman kerran ja luomalla aina uuden instanssidatablokin. Kuviossa 9 esitetään FB130-lohko ja kaikki sen muuttujat.

```
A     "Aina_1"                                          M0.1
=     L      0.0
BLD   103
CALL  "SLi_SP_DP_s128" , DB130                         FB130
 Comp_ID                 :=B#16#2
 P_SLi                   :="P_SLi"                     MD16
 First_Source_Pos        :="Tilatiedot".LI100_HI_haly  DB1.DBX0.0
 First_IEC_Info_Adr      :=L#1
 Src_Struct_Type         :=B#16#0
 Val_Type                :=B#16#1
 No_of_Infos             :=6
 Tx_Prio                 :=1
 Time_3_7                :=L0.0
 Time_Stamp_spo          :=TRUE
 Time_Stamp_cyc          :=FALSE
 Time_Stamp_req          :=FALSE
 Set_NT                  :=FALSE
 Inro_QOI                :=B#16#14
 Send_Cyclic_Interval_sec:=0
 Phase_Offset_Cyc_Interv :=0
 IEC_InfoAdr_FeedBack    :=L#0
 FB_RetVal               :=
NOP   0
```

KUVIO 9. FB130-lohko STL-ohjelmointikielellä

Comp_ID-muuttujaan määritetään yksilöllinen tunniste ja P_SLi-muuttujaan tuodaan pointteri FB121-lohkolta. First_Source_Pos-muuttujalla lohkolle kerrotaan, mistä löytyy ensimmäinen kohde, jota luetaan. Loin tilatiedoille oman datablokin, jonne määrittelin muutamia tilatietoja. Koska halusin, että IEC lukee tilatiedot heti ensimmäisestä lähtien, First_Source_Pos-muuttujaan piti antaa tilatieto-DB:n ensimmäisen bitin osoite eli DB1.DBX0.0. Ohjelmointitapojen ja -tottumusten mukaan tilatiedot voidaan ohjelmoida myös logiikan M-alueelle ja lukea sieltä. Tällöin muuttujalle annetaan ensimmäisen halutun tilatiedon osoite, esimerkiksi M100.0, ja lohko ymmärtää sen. Lohkolla voidaan lukea myös suoraan logiikan tuloja, jolloin muuttujaan kirjoitetaan esimerkiksi I0.0.

First_IEC_Info_Adr-muuttujalla lohkolle kerrotaan, mistä kohtaa tietoyksiköiden osoitealuetta tilatiedot alkavat. Määrittelin tilatiedot heti osoitealueen alkuun (kts. s. 23), joten muuttujalle kirjoitetaan arvo 1. No_of_Infos-muuttujaan kirjoitetaan, montako tilatietoa (tietoyksikköä) lohkon pitää lukea. Kun lohkolle on kerrottu ensimmäisen tietoyksikön osoite ja tietoyksiköiden määrä, se laskee itse jokaisen tietoyksikön osoitteen. Taulukosta 2 käy ilmi IEC-osoitteiden määrittely esimerkin avulla.

TAULUKKO 2. Esimerkki IEC-osoitteiden määrittelystä

TILATIETO	OSOITE LOGIIKASSA	IEC-OSOITE
Tilatieto_1	DB1.DBX0.0	1
Tilatieto_2	DB1.DBX0.1	2
Tilatieto_3	DB1.DBX0.2	3
Tilatieto_4	DB1.DBX0.3	4
Tilatieto_5	DB1.DBX0.4	5
Tilatieto_6	DB1.DBX0.5	6

Scr_Struct_Type-muuttujalla määritetään tietoyksiköiden rakenne. 0 tarkoittaa yksinapaista bittikenttää ja 1 kaksinapaista bittikenttää. Val_Type-muuttujaan määritellään tietoyksikön datatyyppi. Mikäli muuttujalle kirjoitetaan 1, kyseessä on tyypiltään yksibittinen (Single Point Information) tietoyksikkö. Jos muuttujalle taas kirjoitetaan 3, tietoyksikön tyyppi on kaksibittinen (Double Point Information).

Time_3_7-muuttujalla määritetään, onko käytössä lyhyt vai pitkä aikaleima. Testissä annoin kyseiselle muuttujalle arvon 1, eli logiikassa muodostetaan pitkä (7 tavua) aikaleima. Pitkä aikaleima tarkoittaa sitä, että koko aikaleima tulee logiikasta ja valvomo ei lisää siihen mitään. Kolmella eri Time_Stamp-muuttujalla valitaan, milloin aikaleimaa käytetään. Vaihtoehdot ovat spontaani siirto (Time_Stamp_spo), syklinen siirto (Time_Stamp_cyc) ja asemakyselyjen siirrot (Time_Stamp_req). Itse määrittelin aikaleiman käyttöön vain spontaaneissa siirroissa, eli kun esimerkiksi hälytysbitti menee päälle, siitä jää myös aikaleima valvomolle.

7.2.4 Mittaukset, SLi_Me_ABC_s32

SLi_Me_ABC_s32-lohkon FB-numero on lohkokirjastossa määritelty FB133:ksi ja tästä eteenpäin lohkoa kutsutaan lyhyemmin FB133-lohkoksi. Yhdellä FB133-lohkon instanssidatablokilla voidaan lukea maksimissaan 32 mittausarvoa. Kutsumalla lohkoa ohjelmassa uudellaan ja luomalla sille lisää instanssidatablokkeja, luettavien mittausarvojen määrää voidaan kasvattaa. Lohko tukee kolmea eri mittausarvon tyyppiä ja näitä ovat: normaaliarvo, skaalattuarvo ja liukuluku. Kuviossa 10 esitetään FB133-lohko ja kaikki sen muuttujat.

```
A     "Aina_1"                                  M0.1
=     L      0.0
BLD   103
CALL  "SLi_ME_ABC_s32" , DB133                  FB133
 Comp_ID                 :=B#16#3
 P_SLi                   :="P_SLi"              MD16
 First_Source_Pos        :="Mittaukset".LI100   DB2.DBD0
 First_IEC_Info_Adr      :=L#1000
 Src_Struct_Type         :=B#16#2
 Val_Type                :=B#16#D
 No_of_Infos             :=4
 Tx_Prio                 :=1
 Time_3_7                :=L0.0
 Time_Stamp_spo          :=FALSE
 Time_Stamp_cyc          :=FALSE
 Time_Stamp_req          :=FALSE
 Set_NT                  :=FALSE
 Reset_Threshold         :=FALSE
 Use_Initiation_Method   :=FALSE
 Inro_QOI                :=B#16#14
 Send_Cyclic_Interval_sec:=0
 Phase_Offset_Cyc_Interv :=0
 No_Cyclic_Infos         :=0
 Threshold_Value         :=1.000000e-003
 Threshold_Sensivity     :=5
 FB_RetVal               :=
NOP   0
```

KUVIO 10. FB133-lohko STL-ohjelmointikielellä

Lohkon tunnisteeksi (Comp_ID) laitoin 3, koska periaatteenani oli määritellä kaikkien lohkojen tunnisteet juoksevasti 1:stä eteenpäin. P_SLi-muuttujaa tuodaan pointteri FB121-lohkolta. Kaikille mittauksille tein oman ”Mittaukset”-datablokin, jonka muuttujiin siirsin mittausarvot. First_Source_Pos-muuttujaan tuodaan ensimmäisen luettavaksi halutun mittauksen osoite eli tässä tapauksessa DB2.DBD0. Tuosta osoitteesta voi havaita, että muuttujan koko on tuplasana, eli se on tyypiltään Real. First_Source_Pos-muuttujaan voisi kirjoittaa myös jonkin M-

alueen osoitteen, esimerkiksi MD100, jolloin lohko lähtee lukemaan siitä eteenpäin.

First_IEC_Info_Adr-muuttujaan kerrotaan, mistä kohtaa tietoyksiköiden osoitealuetta mittaukset alkavat. Koska itse jaoin osoitealueen siten, että mittaukset alkavat osoitteesta 1000, kirjoitetaan se muuttujalle. Tällöin Mittaukset-DB:n ensimmäinen mittaus saa IEC-osoitteen 1000 ja sitä seuraavien osoite kasvaa aina yhdellä. Koska olen määritellyt No_of_Infos-muuttujaan, että lohkon pitää lukea neljä mittausta (tietoyksikköä), lohko laskee silloin viimeisen mittauksen IEC-osoitteeksi 1003.

Src_Struct_Type-muuttujalla määritellään tietoyksiköiden rakenne. Jos muuttujalle antaa arvon 0, rakenne on kahden tavun kokoinen mittausarvo (Integer). Mikäli muuttujalle antaa arvon 2, on rakenne neljän tavun kokoinen mittausarvo (Short Real). Val_Type-muuttujalla määritellään sitten itse tietoyksikön tyyppi, ja vaihtoehtoja on kolme erilaista. Muuttujan arvolla 9 tietoyksikön tyypiksi valitaan mittauksen normaaliarvo eli ns. raaka-arvo. Muuttujan arvolla 11 tietoyksikön tyyppi on skaalattu arvo ja muuttujan arvolla 13 tyyppinä on liukuluku. Testiohjelmassa asetin tietoyksiköiden rakenteeksi neljän tavun kokoisen mittausarvon ja tyypiksi liukuluvun, jolloin SCADA pystyy lukemaan desimaalilukuja. Time_3_7-muuttujalla valitaan, onko käytössä pitkä vai lyhyt aikaleima. Minä laitoin arvoksi ykkösen, jolloin logiikassa muodostetaan pitkä aikaleima. Kuitenkin jätin aikaleiman pois kaikista siirtotavoista, ja tämän voi valita eri Time_Stamp-muuttujilla.

7.2.5 Binaariohjaukset, SLo_SC_DC_RC_sx

SLo_SC_DC_RC_sx-lohkon FB-numero on lohkokirjastossa FB135, ja tästä eteenpäin siitä käytetään lyhyempää nimitystä FB135-lohko. Monessa sovelluksessa valvomosta pitää pystyä ohjaamaan toimilaitteita. Sitä varten on käytössä binaariohjaukset, joilla voidaan esimerkiksi käynnistää ja pysäyttää moottoreita, ohjata venttiileitä, suorittaa häiriökuittauksia jne. Lohko ottaa vastaan käskyjä isäntäasemalta ja kirjoittaa ne määrättyyn osoitteeseen logiikassa. Ohjausten maksimimäärää ei ole ennalta rajoitettu, vaan kuten aikaisemmin jo mainittiin, sen rajoit-

taa vapaana olevien osoitteiden määrä ja datablokin maksimikoko, joka riippuu logiikasta. Kuviossa 11 esitetään FB135-lohko ja kaikki sen muuttujat.

```
CALL  "SLo_SC_DC_RC_sx" , DB135                          FB135
 Comp_ID               :=B#16#4
 P_Application         :="P_App"                         MD10
 ASDU_Adr              :=L#10
 First_InfoAdr         :=L#2000
 First_Destination_Pos:="Binaari_ohjaukset".P1_Start     DB3.DBX0.0
 No_of_Infos           :=4
 Dst_Struct_Type       :=B#16#0
 Send_Termination      :=TRUE
 Lock                  :=FALSE
 Break                 :=FALSE
 Time_Q0               :=T#1S
 Time_Q1               :=T#1S
 Time_Q2               :=T#10S
 Cmd_Buffer_Dim        :=0
 CMD_RUN               :=
 CMD_Buf_DB            :=
NOP   0
```

KUVIO 11. FB135-lohko STL-ohjelmointikielellä

Koska kyseessä on ohjauslohko, sille tuodaan pointteri suoraan IEC:n yhteyslohkolta FB100. ASDU_Adr-muuttujaan tuodaan sen ASDUn osoite, joka on odotettavissa tälle lohkolle, eli tässä tapauksessa 10. First_InfoAdr-muuttujaan määritellään ensimmäisen tietokohteen osoite, ja se on tässä 2000. Loin ohjelmaan binaariohjauksille oman datablokin ja sinne halutut ohjaukset. Tämän datablokin ensimmäisen bitin osoite (DB3.DBX0.0) viedään First_Destination_Pos-muuttujaan, jolloin lohko tietää, mistä kirjoittaminen aloitetaan. Ohjausten IEC-osoitteet lasketaan samalla periaatteella kuin tiedonkeruulohkoissakin. Eli jos isäntäasema haluaa kirjoittaa binaariohjausten datablokista esimerkiksi neljännen ohjauksen (DB3.DBX0.3) päälle, se kirjoittaa (arvon 1) asemanumero 10:n osoitteeseen 2003. First_Destination_Pos-muuttujan ei tarvitse välttämättä olla datablokissa, vaan ohjaukset voi kirjoittaa myös M-alueelle. Lohko voi kirjoittaa myös suoraan lähtöjä, jolloin muuttujalle viedään esim. Q0.0

No_of_Infos-muuttujalla lohkolle kerrotaan, kuinka monta tietoyksikköä sille kuuluu ensimmäisestä osoitteesta alkaen. Dst_Struct_Type-muuttuja määrittää lohkolle kirjoitettavien tietoyksiköiden rakenteen. Muuttujan arvolla 0 rakenteeksi valitaan

yksibittinen ohjaus (Single Point Information), ja arvolla 1 rakenne on kaksibittinen (Douple Point Information).

7.2.6 Parametrit, SLo_SE_ABC_sx

SLo_SE_ABC_sx-lohkon FB-numero lohkokirjastossa on FB136, ja tästä eteenpäin siitä käytetään lyhyempää nimitystä FB136-lohko. Parametrien kirjoittaminen valvomosta logiikalle on oleellinen osa automaatiosovelluksia, ja niitä voivat olla esimerkiksi hälytysrajojen asettaminen mittauksiin sekä säätimen parametrien määrittäminen. FB136-lohko ottaa parametreja vastaan normaaliarvoina, skaalattuina arvoina tai liukulukuina. Se kirjoittaa parametrit määrättyyn kohteeseen joko integer-tyyppisenä tai lyhyinä reaalilukuina. Kuviossa 12 on esitetty FB136-lohko ja kaikki sen muuttujat.

```
CALL  "SLo_SE_ABC_sx" , DB136                                   FB136
 Comp_ID               :=B#16#5
 P_Application         :="P_App"                                MD10
 ASDU_Adr              :=L#10
 First_InfoAdr         :=L#3000
 First_Destination_Pos:="Parametrit_valvomosta".Mittaus_HI     DB4.DBD0
 No_of_Infos           :=4
 Dst_Struct_Type       :=B#16#2
 Send_Termination      :=TRUE
 Lock                  :=FALSE

NOP   0
```

KUVIO 12. FB136-lohko STL-ohjelmointikielellä

Comp_ID-muuttujaan annetaan yksilöllinen tunniste ja P_Application-muuttujaan tuodaan pointteri yhteyslohkolta (FB100-lohko). ASDUn osoite kirjoitetaan ASDU_Ard-muuttujaan ja First_InfoAdr-muuttujalla lohkolle kerrotaan, mistä kohtaa tietoyksiköiden osoitealuetta löytyy ensimmäinen parametri. Minun osoitejaossani parametrit lähtevät 3000:sta, ja periaatteessa niitä voisi jatkua aina osoitealueen loppuun asti. Tässä testiohjelmassa en ole kuitenkaan luonut kuin neljä parametria, ja näiden määrä kerrotaan lohkolle No_of_Infos-muuttujalla.

Parametreille tein oman datablokin, jonka nimesin ”Parametrit_valvomosta”. Tähän datablokkiin loin tarpeelliset muuttujat, joiden tyypiksi laitoin Real eli reaaliluku. Tämän datablokin ensimmäisen muuttujan osoite (DB4.DBD0) kirjoitetaan lohkon First_Destination_Pos-muuttujalle. Kirjoitettavien parametrien rakenne määritellään lohkolle Dst_Struct_Type-muuttujalla. Jos kyseessä on kahden tavun kokoinen integer-arvo, muuttujalle kirjoitetaan 0. Vastaavasti neljän tavun kokoisella lyhyellä reaaliluvulla (Short Real) muuttujalle kirjoitetaan arvoksi 2. Koska loin datablokkiin muuttujat, joiden datatyyppi on reaaliluku, valitsin Dst_Struct_Typelle luonnollisesti arvon 2. Tällöin valvomosta voidaan antaa logiikalle parametreja, joiden arvo on esim. 37.2.

8 JOHTOPÄÄTÖKSET JA POHDINTA

Lähtökohdat opinnäytetyön tekoon olivat siinä, ettei minulla ollut käytännössä minkäänlaista tietoa tai kokemusta IEC 60870-5-101 -protokollasta, saati rajapinnan luomisesta sille. Käytetty logiikka oli sentään ehtinyt tulla tutuksi aikaisemman projektin kautta. Myös ohjelmointityökaluissa Simatic Managerin käyttö vaati hieman totuttelua ja asioiden palauttamista mieleen, koska edellisistä käyttökerroista oli jo ehtinyt kulua aikaa. TIA-portaali sen sijaan oli tullut tutuksi varsinkin viimeisen vuoden aikana, joten sen suhteen ei tarvinnut suurempia ihmetellä.

Koska käytetty protokolla oli minulle uusi, täytyi tietenkin hankkia tietoa ja perehtyä siihen. Tiedon hakemisessa tuli hyvin ilmi se, että nykypäivän ja myös tämän alan valtakieli on englanti. Ainakaan minä en löytänyt mistään suomenkielistä materiaalia käytettyyn protokollaan liittyen, mutta se ei ollut itse ongelma. Hankalalta ja välillä aivan toivottomalta tuntui kääntää käytettyjä termejä tms. suomeksi, koska opinnäytetyön kirjoitusohjeissa sitä kuitenkin edellytettiin. Joskus sai aivan tosissaan miettiä, mikä jokin termi on suomen kielellä, kun työelämässäkin on päivittäin tottunut käyttämään niiden englanninkielisiä nimityksiä. Näiden ”pakollisten” käännösten ja aiheen teknispainotteisen sanaston vuoksi en ole täysin tyytyväinen tekstiin, koska vaarana oli, että siitä tulee turhan kankeaa luettavaa.

Kun aloittelin opinnäytetyötä, kokeneemmilta työkavereilta tuli vinkkiä, että käytetyn logiikan ja protokollan kombinaatio ei Suomessa ole välttämättä se kaikkein yleisin. En tätä aivan heti uskonut, koska käytetty protokolla on kuitenkin jo vuosia vanha. Kuitenkin siinä vaiheessa, kun Siemensin Suomen tukipalvelusta vastattiin, että heillä on ko. tuotteesta kovin vähän kokemuksia, oli pakko uskoa. Siinä sai sitten olla yhteyksissä Saksaan, kun eräässä vaiheessa tuli tarvetta tuelle. Työn tarkoitus oli siis luoda toimiva rajapinta Siemensin S7-logiikkaan, kun kommunikointiprotokollana logiikan ja valvomon välillä on IEC 60870-5-101. Kaiken perehtymisen, opiskelun, konfiguroimisen, logiikkaohjelmoinnin ja pohtimisen jälkeen lopputuloksena oli rajapinta, jonka kautta data kulki hienosti ja halutusti molempiin suuntiin. Työn aluksi päätettiin, että rajapintaan luodaan toiminnot tilatietojen ja mittausten lukemiselle sekä binaariohjausten ja parametrien kirjoittamiselle. Nämä

toimivatkin hienosti heti kolmen ensiksi mainitun kohdalla, mutta parametrien kirjoittamisessa näytti jotain epäonnistuvan. Jos kirjoitin valvomokoneelta parametrin arvoksi 87.6, se ilmestyi logiikan muuttujaan arvona 88. Tai vastaavasti esimerkiksi 87.4 näkyi logiikan muuttujassa arvona 87, eli vaikutti siltä, että jossain tapahtui pyöristämistä. Loppujen lopuksi tälle löytyi yksinkertainen selitys, sillä testauksessa käytetyn SCADAn konfiguroinnissa oli kirjoitettavien parametrien tyyppi väärin. Tyyppinä oli normaaliarvo, kun olin logiikkakoodissa määritellyt sen liukuluvuksi. Kun SCADAsta vaihdettiin tyyppi oikeaksi, lopulta parametritkin menivät halutulla tavalla perille.

Suurin ongelma, joka työn aikana tuli vastaan, oli logiikan retentiivisen muistin täyttyminen IEC-applikaation luodessa sisäisiä datablokkeja ja tämän mainitsinkin itse tekstissä. Ennen kuin käsitin, mistä oli kyse, oli ongelmassa pohtimista. Kun menin logiikan online-tilaan ja luotuja datablokkeja oli jo 15 kappaletta, muisti täynnä ja IEC:n yhteyslohko näytti virhettä ”ongelma datablokkien luomisen kanssa”, en voinut kuin ihmetellä, kuinka paljon niitä oikein pitää luoda. Toki heti pystyi päättelemään, että jokin ohjelmassani tms. pahasti epäonnistuu, koska käytettyä tuotetta myydään, eli sen muistin määrän on riitettävä yhteyden muodostamiseen. Jostain tarvitsin kuitenkin vahvistuksen, paljonko niitä sisäisiä datablokkeja oikeasti pitäisi muistiin ilmestyä. Sen tiesin, että Send Buffer -datablokkeja pitäisi tulla vain kaksi kappaletta, koska olin sen yhteyslohkoon näin määritellyt. Siemensin IEC-manuaali ei kuitenkaan tarkentanut, mitä muita datablokkeja muistiin luodaan näiden Send Buffer -datablokkien lisäksi. Tähän tarvitsin tietoa ja vahvistusta Siemensiltä ja jouduin näin olemaan yhteydessä Saksan osastoon.

Mielenkiintoisena välihuomiona voidaan kertoa sekin, että Siemensin henkilö Saksasta halusi nähdä valmiin ohjelmani, ja hänen mielestään se oli juuri niin kuin pitääkin. Häneltä sain vahvistuksen luotavien datablokkien määrästä, mutta oli edelleen mysteeri kaikille, miksi minulla oli logiikan muistiin tullut niin iso kasa datablokkeja. Ei auttanut kuin lähteä puhtaalta pöydältä ja samalla tyhjentää myös logiikan muisti. Ohjelmaa uusiessani tein sen kerralla valmiiksi, latasin logiikkaan, ja yhteys toimi heti moitteettomasti. Testejä suorittaessani huomasin jossain vaiheessa, että logiikan muistiin oli taas ilmestynyt tuplaten sisäisiä datablokkeja alkuperäiseen neljään verrattuna. Tässä vaiheessa piti alkaa tosissaan miettiä, mitä

olen tehnyt. Yleensä logiikkaohjelmaa testatessa siihen pitää aina tehdä muutoksia, jolloin se ladataan uudelleen logiikkaan. Lisäksi olin käyttänyt välillä sähköt pois logiikasta, joten näiden kautta pääsin jäljille. Latasin logiikkaan aina yhden IEC-lohkon instanssidatablokin kerrallaan, käytin logiikasta sähköt pois ja katsoin seuraukset. Syyllinen löytyi tietenkin yhteyslohkon (FB100-lohko) datablokista, jonka uudelleen lataamisen ja logiikan käynnistämisen jälkeen IEC loi sisäiset datablokit aina uudelleen. En osaa sanoa, onko tämä vika vai ominaisuus, mutta tällaisen huomion tein.

Ohjelmalohkojen kääntäminen TIA-portaalille ja sillä toimivan rajapinnan luominen onnistuivat myös, mikä oli yksi työn tavoitteista. Sen toteutuminen oli siksikin tärkeää, koska TIA-portaalin käyttö kasvaa jatkuvasti ja sillä tehdään jo varsin paljon projekteja Apex Automation Oy:lla. Niinpä ainakaan IEC 60870-5-101 -protokolla ei automaattisesti sulje pois TIA:n käyttöä projekteissa. Ohjelmalohkojen kääntämistäkään ei tarvitse joka kerta tehdä, sillä tästä työstä jää yritykselle hyvä ohjelmapohja, jossa lohkot ovat jo valmiiksi. IEC 60870-5-104 -protokollan ja Siemensin S7-logiikan käyttämisestä yrityksellä oli jo kokemusta, ja nyt siihen voidaan lisätä IEC 60870-5-101. Tätä opinnäytetyötä – niin kirjaa kuin itse logiikkaohjelmiakin – voidaan soveltaa ja käyttää hyödyksi yrityksen tulevissa projekteissa.

LÄHTEET

ABB Oy. 2012. MicroSCADA Pro SYS 600 9.3 IEC 60870-5-101 Master Protocol. PDF-dokumentti. Luettu 5.8.2013.

Apex Automation Oy. 2011. Www-dokumentti. Saatavissa: http://www.apexautomation.fi/fi. Luettu 2.7.2013.

Cole, B. 2002. A General introduction to the IEC 60870-5-101 standars fo a communication that supports basic telecontrol tasks. Www-dokumentti. Saatavissa: http://www.trianglemicroworks.com/iec60870-5/downloaddocs.htm. Luettu 2.8.2013.

Kang, D. & Robles, R. 2009. Compartmentalization of Protocols in SCADA Communication. PDF-dokumentti. Saatavissa: http://www.sersc.org/journals/IJAST/vol8/4.pdf. Luettu: 23.8.2013.

Keinänen, T., Kärkkäinen, P., Lähetkangas, M. & Sumujärvi, M. 2007. Automaatiojärjestelmien logiikat ja ohjaustekniikat. Helsinki: WSOY.

Reynders, D., Mackay, S. & Wright, E. 2005. Industrial Data Communications: Elsevier. Sähköinen kirja. Saatavissa: http://www.scribd.com/doc/53430270/87/The-IEC-60870-5-standard. Luettu 17.8.2013.

Siemens AG. 2010. IM 151-8 PN/DP CPU Interface Operating Instructions. PDF-dokumentti. Saatavissa: http://cache.automation.siemens.com/dnl/Tc/Tc1MjI3NQAA_47409312_HB/et200s_im151_8_pn_dp_cpu_operating_instructions_en-US_en-US.pdf. Luettu 20.8.2013.

Siemens AG. 2012a. Simatic ET 200 For distributed automation solutions. PDF-dokumentti. Saatavissa: http://www.automation.siemens.com/salesmaterial-as/brochure/en/brochure_simatic-et200_en.pdf. Luettu 19.8.2013.

Siemens AG. 2012b. Siplus RIC IEC on S7. PDF-dokumentti. Luettu 15.8.2013.

Siemens AG. 2013. IM151-8 PN/DP CPU Interface Module for ET200S Released for Delivery. WWW-dokumentti. Saatavissa: http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&objId=30851575&nodeid0=26997359&load=content&lang=en&siteid=cseus&aktprim=0&objaction=csview&extranet=standard&viewreg=WW. Luettu 20.8.2013.

Strauss, C. 2003. Electrical Network and Communication Systems: Elsevier. Sähköinen kirja. Saatavissa: http://www.scribd.com/doc/8545096/Practical-Electrical-Network-Automation-and-Communication-Systems. Luettu 5.8.2013.